

PROBLEME DE INFORMATICĂ

date la olimpiade în

2019 2018 2017 2016 2015
2014 2013 2012 2011 2010
2009 2008 2007 2006 2005
2004 2003 2002 2001 2000

la clasa a 10-a

... draft (ciornă) ...

Dr. Adrian Răbâea

2 aprilie 2020

Prefață

Stilul acestor cărți este ... ca și cum aş vorbi cu nepoții mei (și chiar cu mine însuși!) ... încercând împreună să găsim o rezolvare cât mai bună pentru o problemă dată la olimpiadă.

Ideea, de a scrie aceste culegeri de probleme date la olimpiadele de informatică, a apărut acum câțiva ani când am întrebat un student (care nu reușea să rezolve niște probleme foarte simple): "Ce te faci dacă un *elev*, care știe că ești *student* și că studiezi și informatică, te roagă, din când în când, să-l ajuți să rezolve câte o problemă de informatică dată la gimnaziu la olimpiadă, sau pur și simplu ca temă de casă, și tu, aproape de fiecare dată, nu îl poți ajuta? Eu cred că nu este prea bine și poate că ... *te faci* ... de râs!" Pe *vremea mea* (!), când eram elev de gimnaziu, un *student* era ca un fel de ... *zeu!* Cu trecerea anilor am înțeles că nu este chiar aşa! și încă ceva: nu am reușit să înțeleg de ce, atunci când cineva nu poate să rezolve corect o problemă de informatică de clasa a 6-a, dată la olimpiada de informatică sau ca temă de casă, folosește această *scuză*: "eu nu am făcut informatică în liceu!" și acest *cineva* este "*zeul*" sau "*zeița*" despre care vorbeam!.

Îmi doresc să prezint elemente simple și desene bune care să ajute la descoperirea unei rezolvări optime sau care să obțină cât mai multe puncte la olimpiadă.

Sunt convins că este util să studiem cu atenție cât mai multe probleme *rezolvate!* Din această cauză cred că sunt utile și primele versiuni în care sunt prezentate chiar și numai enunțurile și indicațiile "oficiale" de rezolvare. Acestea se găsesc în multe locuri; aici încerc să le pun pe "*toate la un loc*"!

Limbajul de programare se alege în funcție de problema pe care o avem de rezolvat. Cu niște ani în urmă alegerea era mai simplă: dacă era o problemă de calcul se alegea Fortran iar dacă era o problemă de prelucrarea masivă a datelor atunci se alegea Cobol. Acum alegerea este ceva mai dificilă! :-) Vezi, de exemplu, IOI2019¹ și IOI2015².

Cred că, de cele mai multe ori, este foarte greu să gândim "simplu" și să nu "ne complicăm" atunci când cautăm o rezolvare pentru o problemă dată la olimpiadă. Acesta este motivul pentru care vom analiza cu foarte mare atenție atât exemplele date în enunțurile problemelor cât și "restrictiile" care apar acolo (ele sigur "ascund" ceva interesant din punct de vedere al algoritmului de rezolvare!).

Am început câteva cărți (pentru clasele de liceu) cu mai mulți ani în urmă, pentru perioada 2000-2007 ([28] - [32]), cu anii în ordine crescătoare!). A urmat o pauză de câțiva ani (destul de mulți!). Am observat că acele cursuri s-au "împrăștiat" un pic "pe net" ([44] - [49])! Încerc acum să ajung acolo unde am rămas ... plecând mereu din prezent ... până când nu va mai fi posibil ... să că, de această dată, anii sunt în ordine ... descreșcătoare! :-)

Acum voi continua ... cu "*Enunțuri*", "*Indicații de rezolvare*" și "*Coduri sursă*" ale problemelor date la olimpiadele județene și naționale.

Pentru liceu perioada acoperită este de "azi" (până când va exista acest "azi" pentru mine!) până în anul 2000 (aveam deja perioada 2000-2007!).

Pentru gimnaziu perioada acoperită este de "azi" până în anul 2010 (nu am prea mult timp disponibil și, oricum, calculatoarele folosite la olimpiade înapoi de 2010 erau ceva mai 'slabe' și ... restricțiile de memorie, din enunțurile problemelor, par 'ciudate' acum!).

Mai departe, va urma "*Rezolvări detaliante*" ... pentru unele probleme numai (tot din cauza lipsei timpului necesar pentru toate!). Prioritate vor avea problemele de gimnaziu (nu pentru că sunt mai 'ușoare' ci pentru că ... elevii de liceu se descurcă și singuri!). Totuși, vor fi prezentate și "*Rezolvări detaliante*" ale problemelor de liceu (pe care le-am considerat în mod subiectiv!) interesante și utile.

Vom încerca și câteva probleme de ... IOI ... dar asta este o treabă ... nu prea ușoară! Cred totuși că este mai bine să prezint numai enunțuri ale problemelor date la IOI în ultimii câțiva ani!

¹ IOI2019 a permis utilizarea limbajelor de programare C++ și Java (<https://ioi2019.az/en-content-14.html#CompetitionEquipment>)

² IOI2015 a permis utilizarea limbajelor de programare C++, Java, Pascal, Python și Rubi (<http://ioi2015.kz/content/view/1/265>)

(asta aşa, ca să vedem cum sunt problemele la acest nivel!). Cei care ajung acolo sau vor să ajungă acolo (la IOI) nu au nevoie de ajutorul meu! Se descurcă singuri! La *"Indicații de rezolvare"* voi prezenta numai ... numele algoritmilor clasici folosiți în rezolvare.

"ALGORITMI utili la olimpiadele de informatică", separat pentru gimnaziu și liceu, sper să fie de folos!

Și un ultim gând: ce am strâns și am scris în aceste cărți se adresează celor interesati de aceste teme! Nu cârcotașilor! Sunt evidente *sursele* "de pe net" (și *locurile* în care au fost folosite). Nu sunt necesare precizări suplimentare!

Adrese utile:

<https://stats.ioinformatics.org/halloffame/>
<https://stats.ioinformatics.org/tasks/>
<http://stats.ioinformatics.org/results/ROU>

<http://www.cplusplus.com/>
<http://www.infolcup.com/>
<https://www.infoarena.ro/>
<https://www.infogim.ro/>
<http://www.olimpiada.info/>
<https://www.pbinfo.ro/>
<http://www.usaco.org/>

<http://algopedia.ro/>
<http://campion.edu.ro/>
<https://codeforces.com/>
<https://cpbook.net/>
<https://csacademy.com/>
<https://gazeta.info.ro/revigoram-ginfo/>
<https://oj.uz/problems/source/22>
<https://profs.info.uaic.ro/~infogim/2019/index.html>
<http://varena.ro/>
<https://wandbox.org/>

https://en.cppreference.com/w/cpp/language/operator_alternative

Despre ...

Universitatea Tehnică din Cluj-Napoca, Centrul Universitar Nord din Baia-Mare,
Facultatea de Științe, Departamentul de Matematică-Informatică (2018-2009)

Universitatea "Ovidius" din Constanța
Facultatea de Matematică și Informatică, Departamentul de Informatică (2009-1992)

Centrul de Informatică și organizare CINOR, București, (1992-1979)

Facultatea de matematică și informatică, București (1979-1974)

Despre ...

<http://adrianrabaea.scienceontheweb.net/>
<https://www.scribd.com/user/243528817/Adrian-Rabaea>
https://scholar.google.com/citations?user=-sse_1wAAAAJ&hl=en
<https://www.scopus.com/authid/detail.uri?origin=resultslist&authorId=56122389200&zone=>
<http://www.facebook.com/adrian.rabaea>

Cuprins

Lista figurilor	xvii
Lista tabelelor	xix
Lista programelor	xx
I OJI - Olimpiada județeană de informatică	1
1 OJI 2019	2
1.1 pif	2
1.1.1 Indicații de rezolvare	3
1.1.2 *Rezolvare detaliată	5
1.1.3 Cod sursă	5
1.2 traseu	15
1.2.1 Indicații de rezolvare	15
1.2.2 *Rezolvare detaliată	17
1.2.3 Cod sursă	17
1.3 yinyang	22
1.3.1 Indicații de rezolvare	23
1.3.2 *Rezolvare detaliată	23
1.3.3 Cod sursă	23
2 OJI 2018	28
2.1 castel	28
2.1.1 Indicații de rezolvare	30
2.1.2 *Rezolvare detaliată	30
2.1.3 Cod sursă	30
2.2 eq4	32
2.2.1 Indicații de rezolvare	33
2.2.2 *Rezolvare detaliată	34
2.2.3 Cod sursă	34
2.3 turnuri	35
2.3.1 Indicații de rezolvare	37
2.3.2 *Rezolvare detaliată	38
2.3.3 Cod sursă	38
3 OJI 2017	40
3.1 caps	40
3.1.1 Indicații de rezolvare	41
3.1.2 *Rezolvare detaliată	42
3.1.3 Cod sursă	42
3.2 rover	50
3.2.1 Indicații de rezolvare	51
3.2.2 *Rezolvare detaliată	51
3.2.3 Cod sursă	51
3.3 sir	58
3.3.1 Indicații de rezolvare	59
3.3.2 *Rezolvare detaliată	59

3.3.3	Cod sursă	59
4	OJI 2016	65
4.1	interesant	65
4.1.1	Indicații de rezolvare	66
4.1.2	*Rezolvare detaliată	66
4.1.3	Cod sursă	66
4.2	miting	77
4.2.1	Indicații de rezolvare	78
4.2.2	*Rezolvare detaliată	80
4.2.3	Cod sursă	80
5	OJI 2015	93
5.1	charlie	93
5.1.1	Indicații de rezolvare	94
5.1.2	*Rezolvare detaliată	95
5.1.3	Cod sursă	95
5.2	panda	105
5.2.1	Indicații de rezolvare	107
5.2.2	*Rezolvare detaliată	107
5.2.3	Cod sursă	107
6	OJI 2014	121
6.1	ferma	121
6.1.1	Indicații de rezolvare	122
6.1.2	*Rezolvare detaliată	122
6.1.3	Cod sursă	123
6.2	triunghi	132
6.2.1	Indicații de rezolvare	133
6.2.2	*Rezolvare detaliată	133
6.2.3	Cod sursă	133
7	OJI 2013	139
7.1	calcule	139
7.1.1	Indicații de rezolvare	140
7.1.2	*Rezolvare detaliată	140
7.1.3	Cod sursă	140
7.2	zona	142
7.2.1	Indicații de rezolvare	144
7.2.2	*Rezolvare detaliată	145
7.2.3	Cod sursă	145
8	OJI 2012	150
8.1	compresie	150
8.1.1	Indicații de rezolvare	151
8.1.2	*Rezolvare detaliată	151
8.1.3	Cod sursă	152
8.2	culori	156
8.2.1	Indicații de rezolvare	157
8.2.2	*Rezolvare detaliată	158
8.2.3	Cod sursă	158
9	OJI 2011	166
9.1	ai	166
9.1.1	Indicații de rezolvare	168
9.1.2	*Rezolvare detaliată	168
9.1.3	Cod sursă	168
9.2	expresie	177
9.2.1	Indicații de rezolvare	178
9.2.2	*Rezolvare detaliată	180
9.2.3	Cod sursă	180

10 OJI 2010	187
10.1 Expoziție	187
10.1.1 Indicații de rezolvare	188
10.1.2 *Rezolvare detaliată	188
10.1.3 Cod sursă	188
10.2 Text	190
10.2.1 Indicații de rezolvare	191
10.2.2 *Rezolvare detaliată	192
10.2.3 Cod sursă	192
11 OJI 2009	194
11.1 Insule	194
11.1.1 Indicații de rezolvare	195
11.1.2 *Rezolvare detaliată	195
11.1.3 Cod sursă	195
11.2 Rețetă	198
11.2.1 Indicații de rezolvare	199
11.2.2 *Rezolvare detaliată	199
11.2.3 Cod sursă	199
12 OJI 2008	203
12.1 Colaj	203
12.1.1 Indicații de rezolvare	204
12.1.2 *Rezolvare detaliată	206
12.1.3 Cod sursă	206
12.2 Piața	210
12.2.1 Indicații de rezolvare	211
12.2.2 *Rezolvare detaliată	212
12.2.3 Cod sursă	212
13 OJI 2007	215
13.1 Alee	215
13.1.1 Indicații de rezolvare	216
13.1.2 Rezolvare detaliată	216
13.1.3 Cod sursă	218
13.2 Dir	220
13.2.1 Indicații de rezolvare	221
13.2.2 Rezolvare detaliată	221
13.2.3 Cod sursă	225
14 OJI 2006	229
14.1 Ecuații	229
14.1.1 Indicații de rezolvare	230
14.1.2 Rezolvare detaliată	230
14.1.3 Cod sursă	231
14.2 Sudest	233
14.2.1 Indicații de rezolvare	234
14.2.2 Rezolvare detaliată	234
14.2.3 Cod sursă	237
15 OJI 2005	239
15.1 Lăcusta	239
15.1.1 Indicații de rezolvare	239
15.1.2 Rezolvare detaliată	240
15.1.3 Cod sursă	245
15.2 Scara	246
15.2.1 Indicații de rezolvare	247
15.2.2 Rezolvare detaliată	247
15.2.3 Cod sursă	249

16 OJI 2004	251
16.1 Perle	251
16.1.1 Indicații de rezolvare	252
16.1.2 Rezolvare detaliată	252
16.1.3 Cod sursă	254
16.2 Romeo și Julieta	256
16.2.1 Indicații de rezolvare	257
16.2.2 Rezolvare detaliată	257
16.2.3 Cod sursă	259
17 OJI 2003	261
17.1 Spirala	261
17.1.1 Indicații de rezolvare	262
17.1.2 Rezolvare detaliată	262
17.1.3 Cod sursă	268
17.2 Taxe	270
17.2.1 Indicații de rezolvare	270
17.2.2 Rezolvare detaliată	270
17.2.3 Cod sursă	271
18 OJI 2002	274
18.1 Cod strămos	274
18.1.1 *Indicații de rezolvare	274
18.1.2 Rezolvare detaliată	274
18.1.3 *Cod sursă	277
18.2 Triangulații	277
18.2.1 *Indicații de rezolvare	278
18.2.2 Rezolvare detaliată	278
18.2.3 *Cod sursă	279
II ONI - Olimpiada națională de informatică	280
19 ONI 2019	281
19.1 artifact	281
19.1.1 Indicații de rezolvare	282
19.1.2 *Rezolvare detaliată	282
19.1.3 Cod sursă	282
19.2 Scara	288
19.2.1 Indicații de rezolvare	289
19.2.2 *Rezolvare detaliată	289
19.2.3 Cod sursă	289
19.3 walle	291
19.3.1 Indicații de rezolvare	292
19.3.2 *Rezolvare detaliată	293
19.3.3 Cod sursă	293
19.4 nozero	311
19.4.1 Indicații de rezolvare	312
19.4.2 *Rezolvare detaliată	313
19.4.3 Cod sursă	313
19.5 pericol	316
19.5.1 Indicații de rezolvare	317
19.5.2 *Rezolvare detaliată	318
19.5.3 Cod sursă	318
19.6 vip	324
19.6.1 Indicații de rezolvare	325
19.6.2 *Rezolvare detaliată	326
19.6.3 Cod sursă	326

20 ONI 2018	332
20.1 anagrame	332
20.1.1 Indicații de rezolvare	333
20.1.2 *Rezolvare detaliată	334
20.1.3 Cod sursă	334
20.2 multimi	349
20.2.1 Indicații de rezolvare	350
20.2.2 *Rezolvare detaliată	351
20.2.3 Cod sursă	351
20.3 siruri	364
20.3.1 Indicații de rezolvare	365
20.3.2 *Rezolvare detaliată	366
20.3.3 Cod sursă	366
20.4 agora	380
20.4.1 Indicații de rezolvare	381
20.4.2 *Rezolvare detaliată	381
20.4.3 Cod sursă	382
20.5 grup	383
20.5.1 Indicații de rezolvare	384
20.5.2 *Rezolvare detaliată	385
20.5.3 Cod sursă	385
20.6 proiectoare	387
20.6.1 Indicații de rezolvare	388
20.6.2 *Rezolvare detaliată	388
20.6.3 Cod sursă	388
21 ONI 2017	392
21.1 multisum	392
21.1.1 Indicații de rezolvare	393
21.1.2 *Rezolvare detaliată	394
21.1.3 Cod sursă	394
21.2 puzzle	400
21.2.1 Indicații de rezolvare	401
21.2.2 *Rezolvare detaliată	403
21.2.3 Cod sursă	403
21.3 taietura	405
21.3.1 Indicații de rezolvare	406
21.3.2 *Rezolvare detaliată	406
21.3.3 Cod sursă	406
21.4 100m	409
21.4.1 Indicații de rezolvare	410
21.4.2 *Rezolvare detaliată	411
21.4.3 Cod sursă	411
21.5 camp	414
21.5.1 Indicații de rezolvare	415
21.5.2 *Rezolvare detaliată	415
21.5.3 Cod sursă	415
21.6 identice	421
21.6.1 Indicații de rezolvare	422
21.6.2 *Rezolvare detaliată	422
21.6.3 Cod sursă	422
22 ONI 2016	431
22.1 calc	431
22.1.1 Indicații de rezolvare	432
22.1.2 *Rezolvare detaliată	432
22.1.3 Cod sursă	432
22.2 elmer	437
22.2.1 Indicații de rezolvare	438
22.2.2 *Rezolvare detaliată	439
22.2.3 Cod sursă	439

22.3	tort	447
22.3.1	Indicații de rezolvare	448
22.3.2	*Rezolvare detaliată	448
22.3.3	Cod sursă	449
22.4	intrus	452
22.4.1	Indicații de rezolvare	453
22.4.2	*Rezolvare detaliată	454
22.4.3	Cod sursă	454
22.5	movedel	460
22.5.1	Indicații de rezolvare	461
22.5.2	*Rezolvare detaliată	461
22.5.3	Cod sursă	461
22.6	undo	463
22.6.1	Indicații de rezolvare	464
22.6.2	*Rezolvare detaliată	464
22.6.3	Cod sursă	465
23	ONI 2015	469
23.1	cabana	469
23.1.1	Indicații de rezolvare	470
23.1.2	*Rezolvare detaliată	471
23.1.3	Cod sursă	471
23.2	fence	480
23.2.1	Indicații de rezolvare	481
23.2.2	*Rezolvare detaliată	482
23.2.3	Cod sursă	483
23.3	nmult	492
23.3.1	Indicații de rezolvare	493
23.3.2	*Rezolvare detaliată	494
23.3.3	Cod sursă	494
23.4	procente	501
23.4.1	Indicații de rezolvare	502
23.4.2	*Rezolvare detaliată	503
23.4.3	Cod sursă	503
23.5	robotics	508
23.5.1	Indicații de rezolvare	510
23.5.2	*Rezolvare detaliată	511
23.5.3	Cod sursă	511
23.6	sablon	518
23.6.1	Indicații de rezolvare	519
23.6.2	*Rezolvare detaliată	520
23.6.3	Cod sursă	520
24	ONI 2014	527
24.1	joc	527
24.1.1	Indicații de rezolvare	529
24.1.2	*Rezolvare detaliată	529
24.1.3	Cod sursă	529
24.2	spion	534
24.2.1	Indicații de rezolvare	535
24.2.2	*Rezolvare detaliată	536
24.2.3	Cod sursă	536
24.3	zimeria	537
24.3.1	Indicații de rezolvare	538
24.3.2	*Rezolvare detaliată	538
24.3.3	Cod sursă	538
24.4	puteri	539
24.4.1	Indicații de rezolvare	540
24.4.2	*Rezolvare detaliată	541
24.4.3	Cod sursă	541
24.5	rascoala	542

24.5.1	Indicații de rezolvare	544
24.5.2	*Rezolvare detaliată	544
24.5.3	Cod sursă	544
24.6	stiva	547
24.6.1	Indicații de rezolvare	549
24.6.2	*Rezolvare detaliată	549
24.6.3	Cod sursă	549
25	ONI 2013	554
25.1	cumpanit	554
25.1.1	Indicații de rezolvare	554
25.1.2	*Rezolvare detaliată	555
25.1.3	Cod sursă	555
25.2	romb	561
25.2.1	Indicații de rezolvare	562
25.2.2	*Rezolvare detaliată	563
25.2.3	Cod sursă	563
25.3	showroom	565
25.3.1	Indicații de rezolvare	567
25.3.2	*Rezolvare detaliată	568
25.3.3	Cod sursă	568
25.4	flori	571
25.4.1	Indicații de rezolvare	572
25.4.2	*Rezolvare detaliată	572
25.4.3	Cod sursă	572
25.5	taxa	575
25.5.1	Indicații de rezolvare	576
25.5.2	*Rezolvare detaliată	576
25.5.3	Cod sursă	576
25.6	zone	586
25.6.1	Indicații de rezolvare	587
25.6.2	*Rezolvare detaliată	588
25.6.3	Cod sursă	588
26	ONI 2012	597
26.1	cutie	597
26.1.1	Indicații de rezolvare	598
26.1.2	*Rezolvare detaliată	598
26.1.3	Cod sursă	598
26.2	gheizere	599
26.2.1	Indicații de rezolvare	601
26.2.2	*Rezolvare detaliată	601
26.2.3	Cod sursă	601
26.3	plus	605
26.3.1	Indicații de rezolvare	606
26.3.2	*Rezolvare detaliată	607
26.3.3	Cod sursă	607
26.4	amedie	611
26.4.1	Indicații de rezolvare	612
26.4.2	*Rezolvare detaliată	612
26.4.3	Cod sursă	612
26.5	drept	614
26.5.1	Indicații de rezolvare	615
26.5.2	*Rezolvare detaliată	615
26.5.3	Cod sursă	615
26.6	poly	618
26.6.1	Indicații de rezolvare	619
26.6.2	*Rezolvare detaliată	619
26.6.3	Cod sursă	619

27 ONI 2011	623
27.1 mxl	623
27.1.1 Indicații de rezolvare	624
27.1.2 *Rezolvare detaliată	624
27.1.3 Cod sursă	624
27.2 segmente	628
27.2.1 Indicații de rezolvare	629
27.2.2 *Rezolvare detaliată	630
27.2.3 Cod sursă	630
27.3 tsunami	633
27.3.1 Indicații de rezolvare	634
27.3.2 *Rezolvare detaliată	634
27.3.3 Cod sursă	634
27.4 acces	637
27.4.1 Indicații de rezolvare	638
27.4.2 *Rezolvare detaliată	639
27.4.3 Cod sursă	639
27.5 expresie	643
27.5.1 Indicații de rezolvare	644
27.5.2 *Rezolvare detaliată	644
27.5.3 Cod sursă	644
27.6 telecab	650
27.6.1 Indicații de rezolvare	652
27.6.2 *Rezolvare detaliată	653
27.6.3 Cod sursă	654
28 ONI 2010	658
28.1 dreptunghiuri	658
28.1.1 Indicații de rezolvare	659
28.1.2 *Rezolvare detaliată	659
28.1.3 Cod sursă	659
28.2 gaz	664
28.2.1 Indicații de rezolvare	665
28.2.2 Rezolvare detaliată	666
28.2.3 Cod sursă	666
28.3 xp	670
28.3.1 Indicații de rezolvare	671
28.3.2 *Rezolvare detaliată	672
28.3.3 Cod sursă	672
28.4 mesaje	679
28.4.1 Indicații de rezolvare	680
28.4.2 *Rezolvare detaliată	681
28.4.3 Cod sursă	681
28.5 petrecere	690
28.5.1 Indicații de rezolvare	691
28.5.2 *Rezolvare detaliată	691
28.5.3 Cod sursă	692
28.6 triunghi	695
28.6.1 Indicații de rezolvare	697
28.6.2 *Rezolvare detaliată	697
28.6.3 Cod sursă	697
29 ONI 2009	702
29.1 magic	702
29.1.1 Indicații de rezolvare	703
29.1.2 *Rezolvare detaliată	704
29.1.3 Cod sursă	704
29.2 reactii	707
29.2.1 Indicații de rezolvare	708
29.2.2 *Rezolvare detaliată	709
29.2.3 Cod sursă	709

29.3	text	711
29.3.1	Indicații de rezolvare	712
29.3.2	*Rezolvare detaliată	713
29.3.3	Cod sursă	713
29.4	bile	719
29.4.1	Indicații de rezolvare	720
29.4.2	*Rezolvare detaliată	720
29.4.3	Cod sursă	720
29.5	check-in	722
29.5.1	Indicații de rezolvare	723
29.5.2	*Rezolvare detaliată	723
29.5.3	Cod sursă	723
29.6	volei	724
29.6.1	Indicații de rezolvare	725
29.6.2	*Rezolvare detaliată	726
29.6.3	Cod sursă	726
30	ONI 2008	729
30.1	banda	729
30.1.1	Indicații de rezolvare	730
30.1.2	*Rezolvare detaliată	731
30.1.3	Cod sursă	731
30.2	pavare	732
30.2.1	Indicații de rezolvare	733
30.2.2	*Rezolvare detaliată	734
30.2.3	Cod sursă	734
30.3	poarta	737
30.3.1	Indicații de rezolvare	738
30.3.2	*Rezolvare detaliată	738
30.3.3	Cod sursă	739
30.4	aranjare	740
30.4.1	Indicații de rezolvare	741
30.4.2	*Rezolvare detaliată	741
30.4.3	Cod sursă	741
30.5	bile	742
30.5.1	Indicații de rezolvare	743
30.5.2	*Rezolvare detaliată	745
30.5.3	Cod sursă	745
30.6	subgeom	747
30.6.1	Indicații de rezolvare	748
30.6.2	*Rezolvare detaliată	749
30.6.3	Cod sursă	749
31	ONI 2007	751
31.1	Apel	751
31.1.1	Indicații de rezolvare	752
31.1.2	*Rezolvare detaliată	752
31.1.3	*Cod sursă	752
31.2	Castel	753
31.2.1	Indicații de rezolvare	754
31.2.2	Rezolvare detaliată	754
31.2.3	Cod sursă	765
31.3	Excursie	766
31.3.1	Indicații de rezolvare	768
31.3.2	Rezolvare detaliată	768
31.3.3	Cod sursă	770
31.4	Matrice	772
31.4.1	Indicații de rezolvare	773
31.4.2	Rezolvare detaliată	773
31.4.3	Cod sursă	775
31.5	Rânduri	776

31.5.1 Indicații de rezolvare	777
31.5.2 Rezolvare detaliată	778
31.5.3 Cod sursă	779
31.6 Zidar	780
31.6.1 Indicații de rezolvare	782
31.6.2 *Rezolvare detaliată	782
31.6.3 Cod sursă	782
32 ONI 2006	784
32.1 Bombo	784
32.1.1 Indicații de rezolvare	785
32.1.2 Rezolvare detaliată	785
32.1.3 Cod sursă	787
32.2 Cub	790
32.2.1 Indicații de rezolvare	792
32.2.2 Rezolvare detaliată	793
32.2.3 Cod sursă	795
32.3 Logic	797
32.3.1 Indicații de rezolvare	798
32.3.2 *Rezolvare detaliată	799
32.3.3 Cod sursă	799
32.4 Medie	801
32.4.1 Indicații de rezolvare	802
32.4.2 Rezolvare detaliată	802
32.4.3 Cod sursă	804
32.5 Prieteni	805
32.5.1 Indicații de rezolvare	806
32.5.2 Rezolvare detaliată	807
32.5.3 Cod sursă	808
32.6 SG1	810
32.6.1 Indicații de rezolvare	811
32.6.2 Rezolvare detaliată	811
32.6.3 Cod sursă	812
33 ONI 2005	815
33.1 Antena	815
33.1.1 Indicații de rezolvare	816
33.1.2 Rezolvare detaliată	817
33.1.3 *Cod sursă	819
33.2 Avere	819
33.2.1 Indicații de rezolvare	820
33.2.2 Rezolvare detaliată	821
33.2.3 *Cod sursă	822
33.3 Păianjen	822
33.3.1 Indicații de rezolvare	823
33.3.2 Rezolvare detaliată	823
33.3.3 *Cod sursă	832
33.4 Joc	832
33.4.1 Indicații de rezolvare	834
33.4.2 Rezolvare detaliată	834
33.4.3 *Cod sursă	847
33.5 Suma	847
33.5.1 Indicații de rezolvare	848
33.5.2 Rezolvare detaliată	850
33.5.3 *Cod sursă	855
33.6 Vizibil	855
33.6.1 Indicații de rezolvare	855
33.6.2 Rezolvare detaliată	856
33.6.3 *Cod sursă	857

34 ONI 2004	858
34.1 Găina	858
34.1.1 Indicații de rezolvare	859
34.1.2 Rezolvare detaliată	860
34.1.3 Cod sursă	860
34.2 Rez	862
34.2.1 Indicații de rezolvare	863
34.2.2 Rezolvare detaliată	863
34.2.3 Cod sursă	864
34.3 Sortări	865
34.3.1 Indicații de rezolvare	866
34.3.2 Rezolvare detaliată	866
34.3.3 Cod sursă	867
34.4 Cuvinte	867
34.4.1 Indicații de rezolvare	868
34.4.2 Rezolvare detaliată	868
34.4.3 Cod sursă	869
34.5 Puncte	870
34.5.1 Indicații de rezolvare	871
34.5.2 Rezolvare detaliată	871
34.5.3 Cod sursă	872
34.6 Materom	874
34.6.1 Indicații de rezolvare	874
34.6.2 Rezolvare detaliată	875
34.6.3 Cod sursă	876
35 ONI 2003	879
35.1 Asediu	879
35.1.1 Indicații de rezolvare	880
35.1.2 Rezolvare detaliată	880
35.1.3 Cod sursă	882
35.2 Muzeu	884
35.2.1 Indicații de rezolvare	885
35.2.2 Rezolvare detaliată	885
35.2.3 Cod sursă	888
35.3 Munte	890
35.3.1 Indicații de rezolvare	890
35.3.2 Rezolvare detaliată	891
35.3.3 Cod sursă	896
35.4 Partiție	898
35.4.1 Indicații de rezolvare	899
35.4.2 Rezolvare detaliată	900
35.4.3 Cod sursă	903
35.5 Rubine	903
35.5.1 Indicații de rezolvare	905
35.5.2 Rezolvare detaliată	905
35.5.3 Cod sursă	908
35.6 Scufița	910
35.6.1 Indicații de rezolvare	912
35.6.2 Rezolvare detaliată	912
35.6.3 Cod sursă	913
36 ONI 2002	916
36.1 Hotel	916
36.1.1 Indicații de rezolvare	917
36.1.2 Rezolvare detaliată	918
36.1.3 Cod sursă	919
36.2 Lac	921
36.2.1 Indicații de rezolvare	922
36.2.2 Rezolvare detaliată	923
36.2.3 Cod sursă	925

36.3	Logic	927
36.3.1	Indicații de rezolvare	929
36.3.2	Rezolvare detaliată	930
36.3.3	Cod sursă	935
36.4	Foto	936
36.4.1	Indicații de rezolvare	937
36.4.2	Rezolvare detaliată	938
36.4.3	Cod sursă	942
36.5	Balanța	945
36.5.1	Indicații de rezolvare	946
36.5.2	Rezolvare detaliată	947
36.5.3	Cod sursă	948
36.6	Aliniere	949
36.6.1	Indicații de rezolvare	950
36.6.2	Rezolvare detaliată	951
36.6.3	Cod sursă	952
37	ONI 2001	955
37.1	Alpinistul	955
37.1.1	*Indicații de rezolvare	956
37.1.2	Rezolvare detaliată	956
37.1.3	*Cod sursă	961
37.2	Asfaltare	961
37.2.1	*Indicații de rezolvare	961
37.2.2	Rezolvare detaliată	961
37.2.3	*Cod sursă	962
37.3	Oracolul decide	962
37.3.1	*Indicații de rezolvare	963
37.3.2	Rezolvare detaliată	963
37.3.3	*Cod sursă	966
37.4	Alipiri	966
37.4.1	*Indicații de rezolvare	967
37.4.2	Rezolvare detaliată	967
37.4.3	*Cod sursă	971
37.5	Drum	971
37.5.1	*Indicații de rezolvare	971
37.5.2	Rezolvare detaliată	971
37.5.3	*Cod sursă	976
37.6	Pavări	976
37.6.1	*Indicații de rezolvare	976
37.6.2	Rezolvare detaliată	976
37.6.3	*Cod sursă	977
38	ONI 2000	978
38.1	Castelul	978
38.1.1	*Indicații de rezolvare	979
38.1.2	Rezolvare detaliată	979
38.1.3	Cod sursă	985
38.2	Drumuri scurte	988
38.2.1	*Indicații de rezolvare	989
38.2.2	Rezolvare detaliată	989
38.2.3	Cod sursă	991
38.3	Lac - Scafandril	992
38.3.1	Indicații de rezolvare	993
38.3.2	Rezolvare detaliată	994
38.3.3	*Cod sursă	995
38.4	Pariul broșușelor	995
38.4.1	*Indicații de rezolvare	996
38.4.2	Rezolvare detaliată	996
38.4.3	Cod sursă	998
38.5	Obiective turistice	999

38.5.1 *Indicații de rezolvare	1001
38.5.2 Rezolvare detaliată	1001
38.5.3 Cod sursă	1003
38.6 Taxi	1007
38.6.1 *Indicații de rezolvare	1008
38.6.2 Rezolvare detaliată	1008
38.6.3 Cod sursă	1009
Index	1011
Bibliografie	1014

Lista figurilor

4.1	Miting	78
4.2	Miting	78
4.3	Miting	78
4.4	mitingIR1	79
4.5	mitingIR2	79
6.1	ferma	121
6.2	fermaIR	122
6.3	triunghi	132
7.1	zona	143
8.1	compresie	150
9.1	ai	166
12.1	colaj	204
12.2	colaj	204
12.3	colaj	204
12.4	colaj	205
13.1	Dir	220
17.1	Spirala1	261
17.2	Spirala2	261
17.3	Spirala3	261
18.1	Triang	278
31.1	Sigla ONI 2007	751
31.2	Castel1	753
31.3	Excursie	767
31.4	Zid valid	781
31.5	Ziduri invalide	781
31.6	Zidar4	781
32.1	Sigla ONI 2006	784
32.2	Cub1	790
32.3	Cub2	792
33.1	sigla ONI 2005	815
33.2	Antena	816
33.3	Paianjen	822
34.1	Sigla ONI 2004	858
34.2	Gaina	859
34.3	Rez1	862
34.4	rez2	862
34.5	Poarta	871
35.1	Sigla ONI 2003	879
35.2	Asediu	880

36.1	Sigla ONI 2002	916
36.2	Logic1	928
36.3	Logic2	928
36.4	Logic3	928
36.5	Logic4	929
36.6	Foto	937
37.1	Sigla ONI 2001	955
38.1	Sigla ONI 2000	978
38.2	Drum scurt	989
38.3	Scafandru	992
38.4	Obiective	1001

Lista tabelelor

Lista programelor

1.1.1	pif_30p_1.cpp	5
1.1.2	pif_30p_2.cpp	6
1.1.3	pif_50p.cpp	6
1.1.4	pif_70p_1.cpp	7
1.1.5	pif_70p_2.cpp	8
1.1.6	pif_70p_3.cpp	10
1.1.7	pif_90p_1.cpp	11
1.1.8	pif_90p_2.cpp	12
1.1.9	pif_90p_3.cpp	13
1.2.1	traseu40.cpp	17
1.2.2	traseu90_1.cpp	18
1.2.3	traseu90_2.cpp	19
1.2.4	traseu90_3.cpp	20
1.2.5	traseu90_4.cpp	21
1.2.6	traseu90_5.cpp	21
1.3.1	yinyang45.cpp	23
1.3.2	yinyang90_1.cpp	24
1.3.3	yinyang90_2.cpp	25
2.1.1	castel.cpp	30
2.2.1	eq4.cpp	34
2.3.1	turnuri.cpp	38
3.1.1	adrian-100.cpp	42
3.1.2	manolache-100.cpp	44
3.1.3	manolache-v1_100.cpp	45
3.1.4	manolache-v2_100.cpp	46
3.1.5	MLT-100.cpp	48
3.2.1	adrian-100.cpp	51
3.2.2	GM_rover.cpp	53
3.2.3	MLT_Rover.cpp	54
3.2.4	MLT_Rover_STL.cpp	56
3.3.1	adrian-100.cpp	59
3.3.2	GM1.cpp	60
3.3.3	GM2.cpp	61
3.3.4	sir_100.cpp	62
3.3.5	sirEm.cpp	63
4.1.1	interesant_en1.cpp	66
4.1.2	interesant_en2.cpp	68
4.1.3	interesant_en3.cpp	69
4.1.4	interesant_gc1.cpp	71
4.1.5	interesant_gc2.cpp	72
4.1.6	interesant_gc3.cpp	73
4.1.7	interesant_mot.cpp	75
4.1.8	interesant_nv.cpp	75
4.2.1	miting.cpp	80
4.2.2	miting_en.cpp	82
4.2.3	miting_gc2.cpp	84
4.2.4	miting_gc3.cpp	86
4.2.5	miting_gc4.cpp	88
4.2.6	miting_gc5.cpp	90

5.1.1	charlie_adriana.cpp	96
5.1.2	charlie_eugen0.cpp	96
5.1.3	charlie_eugen1.cpp	97
5.1.4	charlie_eugen2.cpp	99
5.1.5	charlie_LilianaSchiopu.cpp	100
5.1.6	charlie_marcel.cpp	101
5.1.7	charlie_radu_v.cpp	102
5.1.8	charlie_SC.cpp	103
5.1.9	charlie_zoli.cpp	104
5.2.1	panda_adriana.cpp	107
5.2.2	panda_eugen0.cpp	109
5.2.3	panda_eugen1.cpp	110
5.2.4	panda_eugen2.cpp	112
5.2.5	panda_Liliana_Schiopu.cpp	113
5.2.6	panda_marcel.cpp	114
5.2.7	panda_radu.cpp	116
5.2.8	panda_zoli1.cpp	117
6.1.1	fermadaniel.cpp	123
6.1.2	fermavlad.cpp	125
6.1.3	flore_nerecursiv.cpp	126
6.1.4	flore_recursiv.cpp	129
6.2.1	triunghi_LS.cpp	133
6.2.2	triunghi_PD.cpp	135
6.2.3	zoli_triunghi.cpp	136
7.1.1	calcFUCuBS.cpp	140
7.1.2	calcFUfaraBS.cpp	141
7.1.3	calcule.cpp	141
7.1.4	vcalcule.cpp	142
7.2.1	Dzona.cpp	146
7.2.2	Gzona.cpp	147
8.1.1	compresie_cristina_sichim.cpp	152
8.1.2	compresie_eugen_nodea1.cpp	152
8.1.3	compresie_eugen_nodea2.cpp	154
8.1.4	compresie_silviu_candale.cpp	155
8.2.1	culoriEM.cpp	158
8.2.2	culori.cpp	159
8.2.3	culori1.cpp	160
8.2.4	culoriCS.cpp	160
8.2.5	culoriEN.cpp	161
8.2.6	culoriLI.cpp	162
8.2.7	culoriLS.cpp	163
8.2.8	culoriSC.cpp	164
9.1.1	ai.cpp	168
9.1.2	ai2.cpp	171
9.1.3	ai3.cpp	175
9.2.1	expresie_rec.cpp	180
9.2.2	expresie_stive1.cpp	183
9.2.3	expresie_stive2.cpp	184
10.1.1	exp_comb.cpp	188
10.1.2	exp_din.cpp	189
10.2.1	text.pas	192
11.1.1	insule94.cpp	195
11.1.2	insule.cpp	196
11.2.1	RETETA.cpp	199
11.2.2	RETETAV.cpp	201
12.1.1	COLAJ_40.cpp	206
12.1.2	Colaj_50.cpp	207
12.1.3	colaj_C.cpp	208
12.2.1	PI_1.pas	212
12.2.2	PI_2.pas	212

12.2.3	PI_3.pas	213
12.2.4	PI_OK.pas	214
13.1.1	Alee1.java	216
13.1.2	alee.c	218
13.2.1	dir1.java	221
13.2.2	dir2.java	223
13.2.3	dir_em.c	225
13.2.4	dir.cpp	226
14.1.1	ecuati1.java	230
14.1.2	ecuati1.cpp	231
14.2.1	sudest1.java	234
14.2.2	sudest2.java	236
14.2.3	sudest.cpp	237
15.1.1	lacusta1.java	240
15.1.2	lacusta2.java	241
15.1.3	lacusta3.java	242
15.1.4	lacusta4.java	243
15.1.5	lacusta5.java	244
15.1.6	lacusta.c	245
15.2.1	scara.java	247
15.2.2	scara.cpp	249
16.1.1	perle.java	252
16.1.2	perle.c	255
16.2.1	rj.java	257
16.2.2	rj.cpp	259
17.1.1	spirala1.java	262
17.1.2	spirala2.java	264
17.1.3	spirala3.java	265
17.1.4	spirala2.pas	268
17.2.1	taxe.java	270
17.2.2	taxe.pas	271
18.1.1	codstramos1.java	274
18.1.2	codstramos2.java	276
18.2.1	triangulatii.java	278
19.1.1	artifact.c	282
19.1.2	artifact0-pr.cpp	283
19.1.3	artifact1-pr.cpp	284
19.1.4	razvan-artifact2-sum.cpp	287
19.2.1	scara_MD1.cpp	289
19.2.2	scara1.cpp	290
19.2.3	scara2.cpp	290
19.3.1	adrian-100.cpp	293
19.3.2	adrian-direct.cpp	296
19.3.3	adrian-mask.cpp	298
19.3.4	adrian-one-portal.cpp	300
19.3.5	walle01.cpp	302
19.3.6	walle1.cpp	304
19.3.7	walle2.cpp	307
19.3.8	wallecos2.cpp	309
19.4.1	brute-dani.cpp	313
19.4.2	razvan-nozero.cpp	314
19.4.3	sursa-dani.cpp	315
19.5.1	adrian-100.cpp	318
19.5.2	pericol_bogdan_100.cpp	319
19.5.3	pericol_bogdan_n2log.cpp	320
19.5.4	pericol_bogdan_nlog2_better_vector.cpp	321
19.5.5	pericol_bogdan_nlog2_vector.cpp	322
19.5.6	pericol_bogdan_vmax2log.cpp	323
19.6.1	brute-dani.cpp	326
19.6.2	max-lexicog.cpp	328

19.6.3	nsgima2.cpp	329
20.1.1	adrian-100.cpp	334
20.1.2	anagrame_chiriac_back.cpp	336
20.1.3	anagrame_cos1.cpp	337
20.1.4	anagrame_cos2.cpp	338
20.1.5	anagrame2.cpp	340
20.1.6	anagrame3.cpp	342
20.1.7	anagrame4.cpp	345
20.1.8	anagrame-turturica.cpp	347
20.2.1	multimi_alin_60p.cpp	351
20.2.2	multimi-assert-100.cpp	354
20.2.3	multimi-eficientizat.cpp	356
20.2.4	multimi-nlog.cpp	358
20.2.5	multimi-turturica.cpp	360
20.2.6	priv_final.cpp	362
20.3.1	siruri_1.cpp	366
20.3.2	siruri_2.cpp	368
20.3.3	siruri_turturica.cpp	375
20.3.4	siruriscanf.cpp	378
20.4.1	agora.cpp	382
20.5.1	grup.cpp	385
20.6.1	proiectoare.cpp	388
21.1.1	multisum_100p_1.cpp	394
21.1.2	multisum_100p_2.cpp	396
21.1.3	multisum_100p_3.cpp	398
21.2.1	puzzle_100p.cpp	403
21.3.1	taietura_100p_1.cpp	407
21.3.2	taietura_100p_2.cpp	407
21.4.1	100m_40p.cpp	411
21.4.2	100m_100p_1.cpp	411
21.4.3	100m_100p_2.cpp	412
21.4.4	100m_100p_3.cpp	412
21.4.5	100m_100p_4.cpp	413
21.5.1	camp_30p.cpp	415
21.5.2	camp_45p.cpp	416
21.5.3	camp_100p_1.cpp	417
21.5.4	camp_100p_2.cpp	419
21.6.1	identice_100p_1.cpp	422
21.6.2	identice_100p_2.cpp	425
21.6.3	identice_100p_3.cpp	428
22.1.1	calc2.cpp	432
22.1.2	calc3.cpp	434
22.1.3	calc4.cpp	435
22.2.1	elmer1.cpp	439
22.2.2	elmer2.cpp	441
22.2.3	elmer3.cpp	443
22.2.4	elmer4.cpp	445
22.3.1	tort1.cpp	449
22.3.2	tort2.cpp	449
22.3.3	tort3.cpp	450
22.3.4	tort4.cpp	451
22.4.1	intrus1.cpp	454
22.4.2	intrus2.cpp	456
22.4.3	intrus3.cpp	458
22.5.1	movedel1.cpp	461
22.5.2	movedel2.cpp	462
22.5.3	movede3.cpp	463
22.6.1	undo1_1.cpp	465
22.6.2	undo1_2.cpp	465
22.6.3	undo2.cpp	466

22.6.4	undo3.cpp	467
22.6.5	undo4.cpp	468
23.1.1	1_cabana.cpp	471
23.1.2	2_cabana.cpp	474
23.1.3	3_cabana.cpp	476
23.1.4	4_cabana.cpp	478
23.1.5	5_cabana.cpp	479
23.2.1	1_fence.cpp	483
23.2.2	2_fence.cpp	484
23.2.3	3_fence.cpp	486
23.2.4	4_fence.cpp	487
23.2.5	5_fence.cpp	490
23.3.1	1_nmult.cpp	494
23.3.2	2_nmult.cpp	494
23.3.3	3_nmult.cpp	495
23.3.4	4_nmult.cpp	496
23.3.5	5_nmult.cpp	497
23.3.6	6_nmult.cpp	498
23.3.7	7_nmult.cpp	499
23.3.8	8_nmult.cpp	500
23.3.9	9_nmult.cpp	500
23.4.1	1_procente.cpp	503
23.4.2	2_procente.cpp	504
23.4.3	3_procente.cpp	505
23.4.4	4_procente.cpp	507
23.5.1	1_robots.cpp	511
23.5.2	2_robots.cpp	513
23.5.3	3_robots.cpp	516
23.6.1	1_sablon.cpp	520
23.6.2	2_sablon.cpp	522
23.6.3	3_sablon.cpp	523
23.6.4	6_sablon.cpp	524
24.1.1	joc.cpp	529
24.2.1	spion_1.cpp	536
24.3.1	zimeria.cpp	538
24.4.1	puteri.cpp	541
24.5.1	rascoala.cpp	544
24.6.1	stiva_90p.cpp	549
24.6.2	stiva_100p.cpp	552
25.1.1	AC_cumpanit.cpp	555
25.1.2	CC_cumpanit.cpp	557
25.1.3	CP1_cumpanit.cpp	559
25.2.1	romb.cpp	563
25.2.2	rombMN.cpp	564
25.3.1	showroom_powers.cpp	568
25.4.1	CC_flori.cpp	572
25.4.2	DPA1_flori.cpp	573
25.4.3	DPA2_flori.cpp	573
25.4.4	flori_MN.cpp	574
25.4.5	flori_VG.cpp	574
25.5.1	taxa_GM1.cpp	576
25.5.2	taxa_GM2.cpp	577
25.5.3	taxa_GM3.cpp	578
25.5.4	taxa_MN1.cpp	579
25.5.5	taxa_MN2.cpp	580
25.5.6	taxa_PC1.cpp	581
25.5.7	taxa_PC2.cpp	583
25.5.8	taxa_PC3.cpp	585
25.6.1	zone_AC1.cpp	588
25.6.2	zone_AC2.cpp	590

25.6.3	zone_MN.cpp	592
25.6.4	zone_PC.cpp	593
26.1.1	cutie.cpp	599
26.2.1	gheizere.cpp	601
26.2.2	gheizereM.cpp	603
26.3.1	plusCS.cpp	607
26.3.2	plusSC.cpp	608
26.4.1	amedie.cpp	612
26.5.1	drept.cpp	615
26.5.2	M_Stroe.cpp	616
26.6.1	poly_bkt.cpp	619
26.6.2	poly_n2.cpp	620
26.6.3	poly_recurciv.cpp	621
26.6.4	poly_triumghi.cpp	621
27.1.1	mxl.cpp	624
27.1.2	mxl1.cpp	626
27.2.1	segmente.cpp	630
27.2.2	segmente1.cpp	631
27.3.1	tsunami.cpp	634
27.3.2	sunami1.cpp	636
27.4.1	acces.cpp	639
27.4.2	acces1.cpp	640
27.4.3	acces2.cpp	642
27.5.1	expresie.cpp	644
27.5.2	expresie1.cpp	646
27.5.3	expresie2.cpp	648
27.6.1	telecab.cpp	654
27.6.2	telecab1.cpp	655
28.1.1	dreptunghiuri-1-100.cpp	659
28.1.2	dreptunghiuri-2-100.cpp	660
28.1.3	dreptunghiuri-3-70.cpp	662
28.2.1	gaz-1-N_SG^2.cpp	666
28.2.2	gaz-2-N_SG^2.cpp	666
28.2.3	gaz-N^2.cpp	667
28.2.4	gaz-N^3.cpp	668
28.2.5	gaz-N_SG.cpp	669
28.3.1	xp.cpp	672
28.4.1	mesaje-1.cpp	681
28.4.2	mesaje-2.cpp	683
28.4.3	mesaje-3.cpp	685
28.4.4	mesaje-4.cpp	686
28.4.5	mesaje-5.cpp	688
28.4.6	mesaje-6.cpp	689
28.5.1	petrecere-1.cpp	692
28.5.2	petrecere-2.cpp	693
28.5.3	petrecere-3.cpp	694
28.5.4	petrecere-4.cpp	694
28.6.1	triunghi-1.cpp	697
28.6.2	triunghi-2.cpp	698
28.6.3	triunghi-3.cpp	699
28.6.4	triunghi-4.cpp	700
28.6.5	triunghi-5.cpp	701
29.1.1	magic_50.cpp	704
29.1.2	magic_S.cpp	705
29.2.1	reactii_50.cpp	709
29.2.2	reactii_DI.cpp	710
29.3.1	text_bkt.cpp	713
29.3.2	text_em.cpp	714
29.3.3	text_mia.cpp	716
29.4.1	bile.cpp	720

29.5.1	checkin.cpp	723
29.6.1	volei.cpp	726
30.1.1	banda.pas	731
30.2.1	pavare.pas	734
30.3.1	poarta.pas	739
30.4.1	aranjare.cpp	741
30.5.1	.c	745
30.5.2	.cpp	746
30.6.1	subgeom.cpp	749
31.2.1	castel1.java	754
31.2.2	castel2.java	756
31.2.3	castel3.java	758
31.2.4	castel4.java	760
31.2.5	castel5.java	762
31.2.6	castel6.java	764
31.2.7	castel.cpp	765
31.3.1	excursie.java	768
31.3.2	.cpp	770
31.4.1	matrice.java	773
31.4.2	matrice.cpp	775
31.5.1	randuri.java	778
31.5.2	randuri.cpp	779
31.6.1	zidar.cpp	782
32.1.1	bombo.java	785
32.1.2	bombo.pas	787
32.1.3	bombo2.pas	789
32.2.1	cub.java	793
32.2.2	cub.cpp	795
32.3.1	logic.cpp	799
32.4.1	medie.java	802
32.4.2	medi.cpp	804
32.5.1	prieteni.java	807
32.5.2	prieteni.cpp	808
32.6.1	sg1.java	811
32.6.2	sg1.pas	812
32.6.3	sg1_1.pas	813
33.1.1	antena.java	817
33.2.1	avere1.java	821
33.3.1	paianjen1.java	824
33.3.2	paianjen2.java	825
33.3.3	paianjen3.java	827
33.3.4	paianjen4.java	830
33.4.1	joc1.java	834
33.4.2	joc2.java	836
33.4.3	joc3.java	838
33.4.4	joc4.java	840
33.4.5	joc5.java	842
33.4.6	joc6.java	845
33.5.1	suma1.java	850
33.5.2	suma2.java	851
33.5.3	suma3.java	853
33.6.1	vizibil.java	856
34.1.1	gaina.java	860
34.1.2	gaina.pas	860
34.2.1	rez.java	863
34.2.2	rez.cpp	864
34.3.1	sortari.java	866
34.3.2	sortari.pas	867
34.4.1	cuvinte.java	869
34.4.2	cuvinte.c	869

34.5.1	puncte.java	871
34.5.2	puncte.c	872
34.6.1	materom.java	875
34.6.2	materom.pas	876
35.1.1	asediu.java	880
35.1.2	asediu.pas	882
35.2.1	muzeu.java	886
35.2.2	muzeu.pas	888
35.3.1	munte.cpp	896
35.4.1	PartitieNrGenerare.java	900
35.4.2	PartitieNrImpare1.java	901
35.4.3	PartitieNrImpare2.java	901
35.4.4	partitie.pas	903
35.5.1	rubine.java	905
35.5.2	rubine.pas	908
35.6.1	scufita.java	912
35.6.2	scufita.pas	913
36.1.1	hotel.java	918
36.1.2	hotel.pas	919
36.2.1	lac.java	923
36.2.2	lac.pas	925
36.3.1	logic1.java	930
36.3.2	logic2.java	931
36.3.3	logic3.java	932
36.3.4	logic4.java	934
36.3.5	logic.pas	935
36.4.1	foto1.java	938
36.4.2	foto2.java	939
36.4.3	foto3.java	941
36.4.4	foto.pas	942
36.5.1	balanta.java	947
36.5.2	balanta.pas	948
36.6.1	aliniere.java	951
36.6.2	aliniere.pas	952
37.1.1	alpinist1.java	956
37.1.2	alpinist2.java	958
37.2.1	asfaltare.java	961
37.3.1	oracol1.java	963
37.3.2	oracol2.java	965
37.4.1	alipiri1.java	967
37.4.2	alipiri2.java	969
37.5.1	drum1.java	971
37.5.2	drum2.java	973
37.6.1	pavari.java	976
38.1.1	castel1.java	979
38.1.2	castel2.java	982
38.1.3	castel.pas	985
38.2.1	graf1.java	989
38.2.2	graf2.java	990
38.2.3	graf.pas	991
38.3.1	lac.java	994
38.4.1	frog1.java	996
38.4.2	frog2.java	997
38.4.3	frog.pas	998
38.5.1	ob.java	1001
38.5.2	ob.pas	1003
38.6.1	taxi.java	1008
38.6.2	taxi.pas	1009

Partea I

OJI - Olimpiada județeană de informatică

Capitolul 1

OJI 2019

1.1 pif

Problema 1 - pif

90 de puncte

După ce a primit de la Simonet, profesorul său de studii sociale, tema pentru proiect, Tânărului Trevor i-a venit ideea jocului "Pay it forward". Pentru cei care nu știu acest joc, el constă în ajutarea de către Trevor a oamenilor aflați la ananghie. Aceștia la rândul lor vor ajuta alți oameni și aşa mai departe.

Fiecare participant (inclusiv Trevor) trebuie să realizeze câte k fapte bune prin care să ajute oamenii. Vârstnicii și tinerii își îndeplinesc în mod diferit această sarcină. Vârstnicii au nevoie de zv zile pentru a introduce în joc o altă persoană, iar tinerii au nevoie de zt zile. Astfel dacă un vârstnic, respectiv un Tânăr, intră în joc în ziua i , el va introduce la rândul lui în joc prima persoană în ziua $i + zv$, respectiv în ziua $i + zt$ Tânărul, a doua persoană în ziua $i + 2 * zv$, respectiv în ziua $i + 2 * zt$ Tânărul și aşa mai departe. Astfel numărul de persoane care participă la joc poate fi diferit în funcție de cum sunt alese persoanele vârstnice și cele tinere. Trevor dorește ca în joc să fie realizate în total cât mai multe fapte bune, dar fiecare participant să aducă în joc maximum $(k + 1)/2$ tineri și maximum $(k + 1)/2$ vârstnici. Participanții pot aduce mai puține persoane de un anumit tip, dar nu au voie să depășească numărul de $(k + 1)/2$ persoane de același tip.

Cerințe

Care este numărul fb de fapte bune care mai sunt de realizat, după trecerea a n zile, de către persoanele intrate deja în joc, astfel încât numărul total de fapte bune așteptate (și cele realizate și cele nerealizate) să fie maxim?

Date de intrare

Fișierul de intrare **pif.in** conține pe prima linie numărul natural n , pe a doua linie numărul k și pe a treia linie numerele zv și zt separate printr-un spațiu.

Date de ieșire

În fișierul de ieșire **pif.out** se va scrie restul împărțirii lui fb , cu semnificația din enunț, la 1234567 (fb modulo 1234567).

Restricții și precizări

- $1 \leq n \leq 10^6$
- $1 \leq k, zt, zv \leq n$
- Pentru teste în valoare de 30 de puncte $fb \leq 10^6$
- Pentru teste în valoare de 30 de puncte $zv = zt = 1$
- Pentru teste în valoare de 20 de puncte $zv = zt \neq 1$
- Pentru teste în valoare de 70 de puncte $k \cdot n \leq 10^6$

Exemple

pif.in	pif.out	Explicații																																				
4 2 1 2	7	<p>$n = 4, k = 2, zv = 1, zt = 2$</p> <p>Avem 16 moduri posibile în care se pot alege persoanele vârstnice și tinere. Dintre ele doar 5 respectă condiția ca numărul vârstnicilor și al tinerilor să fie maxim 1. Dintre cele 5 doar două obțin un număr maxim de fapte bune așteptate.</p> <p>Notăm cu T pe Trevor, cu Vn persoanele vârstnice și cu Tn persoanele tinere.</p> <p>Unul dintre cele 2 cazuri cu număr maxim de fapte bune este următorul:</p> <table border="1"> <thead> <tr> <th>Ziua</th> <th>Persoane datoare să ajute</th> <th>Persoane ajutate</th> <th>Explicație</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>T</td> <td>-</td> <td>T începe jocul (intră în joc)</td> </tr> <tr> <td>1</td> <td>T</td> <td>-</td> <td>T nu ajută pe nimeni (nu au trecut 2 zile)</td> </tr> <tr> <td>2</td> <td>T</td> <td>V1</td> <td>T ajută V1</td> </tr> <tr> <td>3</td> <td>T</td> <td>-</td> <td>T nu ajută pe nimeni (nu au trecut 4 zile)</td> </tr> <tr> <td>3</td> <td>V1</td> <td>V2</td> <td>V1 ajută V2</td> </tr> <tr> <td>4</td> <td>T</td> <td>T1</td> <td>T ajută T1</td> </tr> <tr> <td>4</td> <td>V1</td> <td>T2</td> <td>V1 ajută T2</td> </tr> <tr> <td>4</td> <td>V2</td> <td>V3</td> <td>V2 ajută V3</td> </tr> </tbody> </table>	Ziua	Persoane datoare să ajute	Persoane ajutate	Explicație	0	T	-	T începe jocul (intră în joc)	1	T	-	T nu ajută pe nimeni (nu au trecut 2 zile)	2	T	V1	T ajută V1	3	T	-	T nu ajută pe nimeni (nu au trecut 4 zile)	3	V1	V2	V1 ajută V2	4	T	T1	T ajută T1	4	V1	T2	V1 ajută T2	4	V2	V3	V2 ajută V3
Ziua	Persoane datoare să ajute	Persoane ajutate	Explicație																																			
0	T	-	T începe jocul (intră în joc)																																			
1	T	-	T nu ajută pe nimeni (nu au trecut 2 zile)																																			
2	T	V1	T ajută V1																																			
3	T	-	T nu ajută pe nimeni (nu au trecut 4 zile)																																			
3	V1	V2	V1 ajută V2																																			
4	T	T1	T ajută T1																																			
4	V1	T2	V1 ajută T2																																			
4	V2	V3	V2 ajută V3																																			

Timp maxim de executare/test: **1.0** secunde

Memorie: total **64 MB** din care pentru stivă **8 MB**

Dimensiune maximă a sursei: **15 KB**

Sursa: pif.cpp, pif.c sau pif.pas va fi salvată în folderul care are drept nume ID-ul tău.

1.1.1 Indicații de rezolvare

Descrierea unor soluții posibile

Structurile de date utilizate: vectori

Gradul de dificultate: 2

Soluția 1 - 30 p (sursa: pif_30p_1.cpp)

Utilizăm o funcție recursivă ce este apelată pentru fiecare persoană ce intră în joc. Această funcție se autoapeleză de k ori, câte un apel pentru fiecare nouă persoană adusă în joc de persoana curentă și transmite prin parametri ziua în care noua persoană a intrat în joc și tipul de persoană (tânăr sau vârstnic).

Ieșirea din recursivitate se realizează atunci când ziua depășește valoarea lui n .

Pentru a obține numărul maxim de fapte bune trebuie ca în joc să intre cât mai multe persoane. Acest lucru se obține dacă se aleg întotdeauna mai întâi tipul de persoane care au cel mai mic număr de zile pentru a realiza o faptă bună. Dacă $zv < zt$ se vor alege mai întâi vârstnici. Dacă $zt < zv$ se vor alege mai întâi tineri.

Tot pentru a obține numărul maxim de fapte bune, dacă valoarea lui k este impară se vor alege $(k + 1)/2$ persoane din tipul cu număr mai mic de zile pentru a realiza o faptă, iar apoi $(k - 1)/2$ persoane din celălalt tip. Dacă valoarea lui k este pară se vor alege $k/2$ persoane din ambele tipuri.

Funcția returnează numărul de fapte bune rămasă nerealizate.

Complexitate: $O(k^{\frac{n}{z_f+z_b}})$

Soluția 2 - 70 p (sursa: pif_70p_1.cpp)

Pentru a evita apelurile redundante ale funcției utilizăm doi vectori în care memorăm valorile returnate de funcție pe măsură ce apelurile recursive se desfășoară. Un vector pentru vârstnici și unul pentru tineri. Înainte de fiecare apel verificăm dacă funcția a fost deja apelată pentru valorile respective (ziua și tipul de persoană), dacă a fost deja apelată luăm valoarea din vectori, dacă nu a fost apelată o apelăm.

Complexitate: $O(k \cdot n)$

Soluția 3 - 30 p (sursa: pif_30p_2.cpp)

Pentru cazul $zt = zv = 1$ numărul de fapte bune este același indiferent de ordinea în care ar fi alese persoanele vârstnice și tinere, iar numărul de persoane noi care intră în joc în fiecare zi este dat de un sir Fibonacci generalizat în care fiecare valoare se obține prin însumarea ultimelor k valori anterioare: $x_n = \sum_{i=n-k}^{n-1} x_i$.

Folosim un vector în care memorăm aceste valori.

Parcurgem vectorul și calculăm numărul de persoane din ziua curentă pe baza valorilor din zilele anterioare.

La sfârșit calculăm numărul de fapte bune rămase nerealizate de către cei care au intrat în joc în ultimele k zile.

Complexitate: $O(n)$

Soluția 4 - 50 p (sursa: pif_50p.cpp)

În general pentru cazul în care $zt = zv$ procedăm la fel ca în cazul anterior doar că în loc să parcurgem vectorul cu pasul 1, vom parurge vectorul cu pasul zv . Niște pentru această situație nu contează ordinea în care alegem persoanele vârstnice și pe cele tinere.

Complexitate: $O(\frac{n}{zv})$

Soluția 5 - 70 p (sursa: pif_70p_2.cpp)

Pentru situațiile în care $zt \neq zv$ procedăm asemănător, dar păstrăm numărul de tineri și vârstnici care intră în joc în fiecare zi, în vectori separați, pentru că durează un număr de zile diferit până când finalizează realizarea celor k fapte bune. Fie acestea nv pentru vârstnici și nt pentru tineri. Parcurgem vectorii în paralel și pentru fiecare poziție i , adunăm numărul de persoane de pe poziția i la numărul de persoane de pe pozițiile $i + zv, i + 2zv, \dots$ pentru vârstnici și $i + zt, i + 2zt, \dots$ pentru tineri.

La adunare trebuie ținut seama de faptul că fiecare persoană trebuie să aleagă întotdeauna mai întâi $(k+1)/2$ persoane din tipul de persoane care au cel mai mic număr de zile pentru a realiza o faptă bună și apoi $k/2$ persoane din tipul celălalt. Dacă $zv < zt$ vom aduna în vectorul nv pe pozițiile $i + zv, i + 2zv, \dots, i + (k+1)/2zv$ valorile din $nv[i]$ și pe pozițiile $i + zt, i + 2zt, \dots, i + (k+1)/2zt$ valorile din $nt[i]$. În vectorul nt vom aduna pe pozițiile $i + ((k+1)/2+1)zv, i + ((k+1)/2+2)zv, \dots, kzv$ valoarea din $nv[i]$ și pe pozițiile $i + ((k+1)/2+1)zt, i + ((k+1)/2+2)zt, \dots, kzt$ valorile din $nt[i]$.

Dacă $zt < zv$ vom proceda analog inversând vectorii nv și nt .

Toate valorile care ar trebui adunate pe poziții mai mari de cătă n în cei doi vectori reprezintă fapte bune ce mai sunt de realizat după trecerea celor n zile.

Complexitate $O(k \cdot n)$

Soluția 6 - ?? p (sursa pif_??p.cpp)

Soluția 7 - 90 p (sursa pif_90p.cpp)

La soluția anterioară, pentru a elimina redundanța adunărilor la fiecare pas i , folosim căte un vector suplimentar pentru fiecare tip de persoană (unul pentru tineri și unul pentru vârstnici), în care memorăm numărul de persoane active în fiecare zi (adică numărul de persoane care mai au de realizat fapte bune). Astfel numărul de persoane noi care intră în joc în ziua i va depinde doar de numărul de persoane active din ziua $i - zv$, respectiv $i - zt$. La fiecare pas actualizăm numărul de persoane active, ținând seama de modul cum sunt alese tipurile de persoane.

Complexitate $O(n)$

Soluția 8 - 70 p (sursa: pif_70p_3.cpp)

Să ne propunem să calculăm în fiecare zi căte persoane intră în joc.

Fie $v[i]$ și $t[i]$, numărul vârstnicilor respectiv al tinerilor care intră în joc în ziua i . Evident $v[0] = 0$ și $t[0] = 1$.

Cum putem afla, la finalul fiecărei zile, căte fapte bune mai rămân de efectuat? Pornim cu o variabilă $target$ egală cu k , fiind sarcina lui Trevor, pe care o vom adapta din aproape în aproape pentru a reflecta numărul de fapte bune ce trebuie efectuate la finalul zilei i .

Astfel, la finalul zilei i , cele $v[i] + t[i]$ persoane noi introduse trebuie scăzute din target, încărcând ele au fost introduse de căte o persoană ce acum are o sarcină cu 1 mai mică. De asemenea, cele $v[i] + t[i]$ persoane noi introduse, au la rândul lor de adus căte k persoane în joc fiecare. Reiese deci că variabila $target$ va crește cu $(v[i] + t[i]) * (k - 1)$.

Să ne concentrăm acum atenția asupra calculului valorilor $v[i]$ și $t[i]$.

Putem presupune $zt < zv$, celălalt caz fiind similar. Este destul de intuitiv faptul că dacă o persoană aduce întâi în joc tineri, căt de mult se poate, și apoi vârstnici, vor fi mai multe persoane și în consecință mai multe fapte bune. Acest lucru se întâmplă deoarece tinerii au o frecvență de lucru mai bună, $zt < zv$, și cu căt sunt introduși mai devreme în joc, la rândul lor vor introduce întâi în joc tineri și faptele bune vor apărea într-un ritm accelerat.

Astfel dacă $k = 2 * t$, conform enunțului, atunci o persoană va trebui să introducă t tineri și t vârstnici. În schimb dacă $k = 2 * t + 1$, există șansa de a adăuga mai mulți tineri, mai exact $t + 1$, rămânând de adăugat t vârstnici. În general fiecare persoană va introduce întâi kt tineri și apoi

kv vârstnici, $kt + kv = k$, kt și kv având valorile stabilite mai devreme, în funcție de paritatea lui k .

În ziua i , câte un Tânăr este introdus în joc de către un Tânăr introdus în ziua $i - zt$, $i - 2 * zt$, ..., $i - kt * zt$. De asemenea câte un Tânăr va fi introdus în joc de către un vârstnic din ziua $i - zv$, $i - 2 * zv$, ..., $i - kt * zv$.

În consecință avem:

$$\begin{aligned} t[i] &= t[i - zt] + t[i - 2 * zt] + \dots + t[i - kt * zt] + \\ &\quad + v[i - zv] + v[i - 2 * zv] + \dots + v[i - kt * zv] \end{aligned}$$

(vom considera nule valorile cu index negativ)

În cazul lui $v[i]$ raționamentul este similar, cu diferența că orice persoană ce introduce un vârstnic trebuie să fi adăugat înainte kt tineri. Deci indicii sunt translatați.

$$\begin{aligned} v[i] &= t[i - (1 + kt) * zt] + t[i - (2 + kt) * zt] + \dots + t[i - (kv + kt) * zt] + \\ &\quad + v[i - (1 + kt) * zv] + v[i - (2 + kt) * zv] + \dots + v[i - (kv + kt) * zv] \end{aligned}$$

Întrucât pentru calculul lui $v[i]$ și $t[i]$ se fac $O(k)$ pași, avem o complexitate finală $O(n * k)$. Complexitate $O(k * n)$

Soluția 9 - 90 p (sursa: pif_90p_2.cpp)

Puteam optimiza recurențele de mai sus, observând că valorile $t[]$ sunt însumate din zt în zt , iar valorile $v[]$ sunt însumate din zv în zv . Astfel dacă calculăm sume parțiale cu un pas egal cu zt , respectiv zv , putem reduce calculul lui $v[i]$ și $t[i]$ la $O(1)$.

De exemplu, dacă am calcula

$$st[j] = t[j] + t[j - zt] + t[j - 2 * zt] + \dots,$$

care de fapt este

$$st[j] = t[j] + st[j - zt],$$

putem calcula $t[i]$ din formulă, atunci când $zt < zv$, ca fiind

$$t[i] = st[i - zt] - st[i - (kt + 1) * zt]$$

Complexitate $O(n)$

1.1.2 *Rezolvare detaliată

1.1.3 Cod sursă

Listing 1.1.1: pif_30p_1.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 ifstream in("pif.in");
6 ofstream out("pif.out");
7
8 int n, k, zb, zf;
9
10 int f(int z, int sex, int za, int sa)
11 {
12     if(z<=n)
13     {
14         int s=0, zz=zf;
15         if(sex==1)
16         {
17             zz=zb;
18         }
19         int vk=k/2;
20         if(k%2==1)
21         {
22             vk++;
23         }
24         if(zf<zb)
25         {

```

```

26         for(int i=1;i<=vk;i++)
27             s=s+f(z+zz*i,0,z,sex);
28         for(int i=vk+1;i<=k;i++)
29             s=s+f(z+zz*i,1,z,sex);
30     }
31     else
32     {
33         for(int i=1;i<=vk;i++)
34             s=s+f(z+zz*i,1,z,sex);
35         for(int i=vk+1;i<=k;i++)
36             s=s+f(z+zz*i,0,z,sex);
37     }
38     return s%1234567;
39 }
40 else
41 {
42     return 1;
43 }
44 }
45
46 int main()
47 {
48     in>>n>>k>>zf>>zb;
49     out << f(0,1,0,1) << endl;
50     return 0;
51 }
```

Listing 1.1.2: pif_30p_2.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 ifstream in("pif.in");
6 ofstream out("pif.out");
7
8 long long n,k,zb,zf;
9 long long nb[1000010],s;
10
11 int main()
12 {
13     in>>n>>k>>zf>>zb;
14     nb[0]=1;
15     nb[1]=1;
16     int sk=1;
17
18     for(int i=2;i<=n;i++)
19     {
20         sk=2*sk;
21         if(i>k) sk-=nb[i-k-1];
22         if(sk<0) sk+=1234567;
23         sk%=1234567;
24         nb[i]=sk;
25     }
26
27     for(int i=0;i<k;i++)
28     {
29         s+=(k-i)*nb[n-i];
30         s%=1234567;
31     }
32     out << s << endl;
33     return 0;
34 }
```

Listing 1.1.3: pif_50p.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 ifstream in("pif.in");
6 ofstream out("pif.out");
7
8 int n,k,zb,zf;
```

```

9
10 long long npn[1000010], // numarul de persoane noi
11     npa[1000010], // numarul de persoane active
12     s[1000010]; // numarul total de persoane
13
14 int main()
15 {
16     in>>n>>k>>zf>>zb;
17     npa[0]=1; // Trevor
18     npn[0]=1;
19     s[0]=1; // Trevor
20     for(int i=zf;i<=n;i+=zf)
21     {
22         npa[i]=2*npa[i-zf];
23         npa[i]%=1234567;
24         if(i>=k*zf) npa[i]-=npn[i-k*zf];
25         if(npa[i]<0) npa[i]+=1234567;
26         npn[i]=npa[i-zf];
27         s[i]=s[i-zf]+npn[i];
28         s[i]%=1234567;
29     }
30     long long ntfbr=0;
31     for(int i=1;i<=k+1;i++)
32     {
33         if(n-(n)%zf-(k-i+1)*zf>=0)
34         {
35             ntfbr=ntfbr+(i-1)*npn[n-(n)%zf-(k-i+1)*zf];
36             ntfbr%=1234567;
37         }
38     }
39     out << ntfbr << endl;
40     return 0;
41 }
```

Listing 1.1.4: pif_70p_1.cpp

```

1 #include <fstream>
2
3 using namespace std;
4
5 ifstream in("pif.in");
6 ofstream out("pif.out");
7
8 int n,k,zb,zf,f[1000010][2],m[1000010][2];
9
10 int p(int z,int sex,int za,int sa)
11 {
12     int s=0;
13     if(z<=n)
14     {
15         int s=0,zz=zf;
16         if(sex==1)
17             zz=zb;
18
19         int vk=k/2;
20         if(k%2==1)
21             vk++;
22
23         if(zf<zb)
24         {
25             for(int i=1;i<=vk;i++)
26             {
27                 if(m[z+zz*i][0]==1)
28                     s=s+f[z+zz*i][0];
29                 else
30                 {
31                     f[z+zz*i][0]=p(z+zz*i,0,z,sex);
32                     m[z+zz*i][0]=1;
33                     s=s+f[z+zz*i][0];
34                 }
35             }
36
37             for(int i=vk+1;i<=k;i++)
38                 if(m[z+zz*i][1]==1)
39                     s=s+f[z+zz*i][1];

```

```

40             else
41             {
42                 f[z+zz*i][1]=p(z+zz*i,1,z,sex);
43                 m[z+zz*i][1]=1;
44                 s=s+f[z+zz*i][1];
45             }
46         }
47     else
48     {
49         for(int i=1;i<=vk;i++)
50         {
51             if(m[z+zz*i][1]==1)
52                 s=s+f[z+zz*i][1];
53             else
54             {
55                 f[z+zz*i][1]=p(z+zz*i,1,z,sex);
56                 m[z+zz*i][1]=1;
57                 s=s+f[z+zz*i][1];
58             }
59         }
60         for(int i=vk+1;i<=k;i++)
61             if(m[z+zz*i][0]==1)
62                 s=s+f[z+zz*i][0];
63             else
64             {
65                 f[z+zz*i][0]=p(z+zz*i,0,z,sex);
66                 m[z+zz*i][0]=1;
67                 s=s+f[z+zz*i][0];
68             }
69         }
70     }
71     return s%1234567;
72 }
73 else
74 {
75     return 1;
76 }
77 }
78 }
79
80 int main()
81 {
82     in>>n>>k>>zf>>zb;
83     out << p(0,1,0,1) << endl;
84 }
```

Listing 1.1.5: pif_70p_2.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 ifstream in("pif.in");
6 ofstream out("pif.out");
7
8 int n,k,zb,zf;
9 int nf[1000010],nb[1000010],s;
10
11 int main()
12 {
13     in>>n>>k>>zf>>zb;
14     nb[0]=1;
15     int vk=k/2;
16     if(k%2==1)
17         vk++;
18
19     if(zf<zb)
20     {
21         for(int i=0;i<=n;i++)
22         {
23             for(int j=1;j<=vk;j++)
24             {
25                 if(i+j*zf<=n)
26                 {
27                     nf[i+j*zf]+=nf[i];
28                 }
29             }
30         }
31     }
32 }
```

```

28             nf[i+j*zf]%=1234567;
29         }
30     else
31     {
32         s+=nf[i];
33         s%=1234567;
34     }
35
36     if(i+j*zb<=n)
37     {
38         nf[i+j*zb]+=nb[i];
39         nf[i+j*zb]%=1234567;
40     }
41     else
42     {
43         s+=nb[i];
44         s%=1234567;
45     }
46 }
47
48 for(int j=vk+1;j<=k;j++)
49 {
50     if(i+j*zf<=n)
51     {
52         nb[i+j*zf]+=nf[i];
53         nb[i+j*zf]%=1234567;
54     }
55     else
56     {
57         s+=nf[i];
58         s%=1234567;
59     }
60
61     if(i+j*zb<=n)
62     {
63         nb[i+j*zb]+=nb[i];
64         nb[i+j*zb]%=1234567;
65     }
66     else
67     {
68         s+=nb[i];
69         s%=1234567;
70     }
71 }
72 }
73 }
74 else
75 {
76     for(int i=0;i<=n;i++)
77     {
78         for(int j=1;j<=vk;j++)
79         {
80             if(i+j*zf<=n)
81             {
82                 nb[i+j*zf]+=nf[i];
83                 nb[i+j*zf]%=1234567;
84             }
85             else
86             {
87                 s+=nf[i];
88                 s%=1234567;
89             }
90
91             if(i+j*zb<=n)
92             {
93                 nb[i+j*zb]+=nb[i];
94                 nb[i+j*zb]%=1234567;
95             }
96             else
97             {
98                 s+=nb[i];
99                 s%=1234567;
100            }
101        }
102
103    for(int j=vk+1;j<=k;j++)

```

```

104         {
105             if(i+j*zf<=n)
106             {
107                 nf[i+j*zf]+=nf[i];
108                 nf[i+j*zf]%=1234567;
109             }
110             else
111             {
112                 s+=nf[i];
113                 s%=1234567;
114             }
115
116             if(i+j*zb<=n)
117             {
118                 nf[i+j*zb]+=nb[i];
119                 nf[i+j*zb]%=1234567;
120             }
121             else
122             {
123                 s+=nb[i];
124                 s%=1234567;
125             }
126         }
127     }
128 }
129 out << s << endl;
130 return 0;
131 }
```

Listing 1.1.6: pif_70p_3.cpp

```

1 // O(n*k)
2 #include <stdio.h>
3
4 #define MOD 1234567
5 #define MAXN 1000001
6
7 int b[MAXN], f[MAXN];
8
9 int main()
10 {
11     FILE *fin = fopen("pif.in", "r");
12     FILE *fout = fopen("pif.out", "w");
13     int n, k, zf, zb, kf, kb, of, ob;
14
15     fscanf(fin, "%d %d %d %d", &n, &k, &zf, &zb);
16     if (zb < zf)
17     {
18         kb = (k + 1)/2;
19         kf = k - kb;
20         ob = 0;
21         of = kb;
22     }
23     else
24     {
25         kf = (k + 1)/2;
26         kb = k - kf;
27         of = 0;
28         ob = kf;
29     }
30
31     int target = k;
32     b[0] = 1;
33     for (int i = 1; i <= n; i++)
34     {
35         for (int j = 1; j <= kb && (j + ob)*zb <= i; j++)
36             b[i] = (b[i] + b[i - (j + ob)*zb]) % MOD;
37         for (int j = 1; j <= kb && (j + ob)*zf <= i; j++)
38             b[i] = (b[i] + f[i - (j + ob)*zf]) % MOD;
39         for (int j = 1; j <= kf && (j + of)*zb <= i; j++)
40             f[i] = (f[i] + b[i - (j + of)*zb]) % MOD;
41         for (int j = 1; j <= kf && (j + of)*zf <= i; j++)
42             f[i] = (f[i] + f[i - (j + of)*zf]) % MOD;
43
44     target = (target + 1LL*(b[i] + f[i])*(k - 1)) % MOD;
}
```

```

45     }
46
47     fprintf(fout, "%d", target);
48     fclose(fin);
49     fclose(fout);
50
51     return 0;
52 }
```

Listing 1.1.7: pif_90p_1.cpp

```

1 #include <fstream>
2 #include <iostream>
3
4 using namespace std;
5
6 ifstream in("0-pif.in");
7 ofstream out("pif.out");
8
9 int n,k,zb,zf;
10
11 int npnf[1000010],// numarul de persoane de sex feminin noi
12     npaff[1000010],// numarul de persoane de sex feminin active pt fete
13     npabf[1000010],// numarul de persoane de sex feminin active pt baieti
14     sf[1000010];// numarul total de persoane de sex feminin
15 int npnb[1000010],// numarul de persoane de sex masculin noi
16     npabf[1000010],// numarul de persoane de sex masculin active pt fete
17     npabb[1000010],// numarul de persoane de sex masculin active pt baieti
18     sb[1000010];// numarul total de persoane de sex masculin
19
20 int main()
21 {
22     in>>n>>k>>zf>>zb;
23     if(zf<=zb)
24     {
25         npnb[1]=1;
26         npabf[1]=1; // Trevor
27         sb[0]=1; // Trevor
28         sb[1]=1; // Trevor
29     }
30     else
31     {
32         swap(zf,zb);
33         npnf[1]=1;
34         npaff[1]=1; // Trevor
35         sf[0]=1; // Trevor
36         sf[1]=1; // Trevor
37     }
38
39     int vk=k/2;
40     if(k%2==1)vk++;
41
42     for(int i=2;i<=n+1;i++)
43     {
44         if(i>zf)
45         {
46             npnf[i]=npaff[i-zf];
47             if(i>zb) npnf[i]+=npabf[i-zb];
48             npaff[i]=npaff[i-zf]+npnf[i];
49             npabf[i]=npabf[i-zf];//???
50             if(i>vk*zf)
51             {
52                 npaff[i]-=npnf[i-vk*zf];
53                 if(npaff[i]<0) npaff[i]+=1234567;
54                 npabf[i]+=npnf[i-vk*zf];
55                 if(i>k*zf)
56                 {
57                     npabf[i]-=npnf[i-k*zf];
58                     if(npabf[i]<0) npabf[i]+=1234567;
59                 }
60             }
61         }
62
63         npnf[i]%=1234567;
64         npaff[i]%=1234567;
```

```

65      npafb[i]%=1234567;
66      if(i>zf)
67      {
68          npnb[i]=npafb[i-zf];
69          if(i>zb)
70          {
71              npnb[i]+=npabb[i-zb];
72              npabb[i]=npabb[i-zb];
73              npabf[i]=npabf[i-zb];
74          }
75
76          npabf[i]+=npnb[i];
77          if(i>vk*zb)
78          {
79              npabf[i]-=npnb[i-vk*zb];
80              if(npabf[i]<0) npabf[i]+=1234567;
81              npabb[i]+=npnb[i-vk*zb];
82              if(i>k*zb)
83              {
84                  npabb[i]-=npnb[i-k*zb];
85                  if(npabb[i]<0) npabb[i]+=1234567;
86              }
87          }
88      }
89
90      npnb[i]%=1234567;
91      npabf[i]%=1234567;
92      npabb[i]%=1234567;
93      sb[i]=sb[i-1]+npnb[i];
94      sb[i]%=1234567;
95      sf[i]=sf[i-1]+npnf[i];
96      sf[i]%=1234567;
97  }
98
99  long long ntfbr=0;
100 for(int i=1;i<=k;i++)
101 {
102     if(n+1-(i-1)*zf>=1)
103     {
104         if(n+1-i*zf-1>=0)
105         {
106             long long dif=(sf[n+1-(i-1)*zf]-sf[n+1-i*zf]);
107             if(dif<0) dif+=1234567;
108             ntfbr=ntfbr+(k-i+1)*dif;
109         }
110     else
111         ntfbr=ntfbr+(k-i+1)*sf[n+1-(i-1)*zf];
112
113     ntfbr%=1234567;
114 }
115
116     if(n+1-(i-1)*zb>=1)
117     {
118         if(n+1-i*zb-1>=0)
119         {
120             long long dif=(sb[n+1-(i-1)*zb]-sb[n+1-i*zb]);
121             if(dif<0) dif+=1234567;
122             ntfbr=ntfbr+(k-i+1)*dif;
123         }
124     else
125         ntfbr=ntfbr+(k-i+1)*sb[n+1-(i-1)*zb];
126
127     ntfbr%=1234567;
128 }
129 }
130
131     out << ntfbr << endl;
132     cout << ntfbr << endl;
133
134     return 0;
135 }
```

Listing 1.1.8: pif_90p_2.cpp

```

2 #include <stdio.h>
3 #include <iostream>
4
5 using namespace std;
6
7 #define MOD 1234567
8 #define MAXN 1000001
9
10 int b[MAXN], f[MAXN];
11
12 int query(int *x, int a, int b, int r)
13 {
14     if (b < 0) return 0;
15     if (a < r) return x[b];
16     return (x[b] - x[a - r] + MOD) % MOD;
17 }
18
19 int main()
20 {
21     FILE *fin = fopen("0-pif.in", "r");
22     FILE *fout = fopen("pif.out", "w");
23     int n, k, zf, zb, kf, kb, of, ob;
24
25     fscanf(fin, "%d %d %d %d", &n, &k, &zf, &zb);
26     if (zb < zf)
27     {
28         kb = (k + 1)/2;
29         kf = k - kb;
30         ob = 0;
31         of = kb;
32     }
33     else
34     {
35         kf = (k + 1)/2;
36         kb = k - kf;
37         of = 0;
38         ob = kf;
39     }
40
41     int target = k; b[0] = 1;
42     for (int i = 1; i <= n; i++)
43     {
44         int nb = (query(b, i - (kb + ob)*zb, i - (1 + ob)*zb, zb)
45                  + query(f, i - (kb + ob)*zf, i - (1 + ob)*zf, zf)) % MOD;
46         int nf = (query(b, i - (kf + of)*zb, i - (1 + of)*zb, zb)
47                  + query(f, i - (kf + of)*zf, i - (1 + of)*zf, zf)) % MOD;
48
49         b[i] = (nb + ((i >= zb) ? b[i - zb] : 0)) % MOD;
50         f[i] = (nf + ((i >= zf) ? f[i - zf] : 0)) % MOD;
51
52         target = (target + 1LL*(nb + nf)*(k - 1)) % MOD;
53     }
54
55     fprintf(fout, "%d", target);
56     fclose(fin);
57     fclose(fout);
58
59     cout<<target;
60
61     return 0;
62 }
```

Listing 1.1.9: pif_90p_3.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <algorithm>
5
6 using namespace std;
7
8 static const int kModulo = 1234567;
9
10 int main()
11 {
```

```

12     ifstream cin("0-pif.in");
13 //ofstream cout("pif.out");
14 ofstream fout("pif.out");
15
16     int N, K, days[2]; cin >> N >> K >> days[0] >> days[1];
17
18     int first_half = (K + 1) / 2;
19
20     vector<int> brought_by[2][2], delta[2][2];
21
22     for (int i = 0; i < 2; ++i)
23         for (int j = 0; j < 2; ++j)
24         {
25             brought_by[i][j] = delta[i][j] = vector<int>(N + 1, 0);
26         }
27
28     if (days[0] > days[1])
29     {
30         swap(days[0], days[1]);
31         brought_by[0][0][0] = 1;
32         delta[0][0][days[0]] = -1;
33     }
34     else
35     {
36         brought_by[1][1][0] = 1;
37         delta[1][1][days[1]] = -1;
38     }
39
40     int total = 0;
41     for (int i = 0; i <= N; ++i)
42     {
43         for (int brought = 0; brought < 2; ++brought)
44             for (int by = 0; by < 2; ++by)
45             {
46                 int &current = brought_by[brought][by][i];
47 // same as days[by] ago and taking delta into account
48                 if (i >= days[by])
49                     current =
50                         (current +
51                             brought_by[brought][by][i - days[by]]) % kModulo;
52                 current = (current + delta[brought][by][i]) % kModulo;
53                 total = (total + current) % kModulo;
54             }
55
56         for (int brought = 0; brought < 2; ++brought)
57         {
58             int current = brought_by[brought][0][i]+brought_by[brought][1][i];
59 // so these will generate 0's for (k + 1) / 2 days and then
60 // 1's for the next
61 // first one is after days[brought] days
62             int start = i + days[brought];
63             int middle = start + days[brought] * first_half;
64             int end = start + days[brought] * K;
65             if (start > N)
66                 continue;
67             delta[0][brought][start] =
68                 (delta[0][brought][start] + current) % kModulo;
69             if (middle > N)
70                 continue;
71             delta[0][brought][middle] =
72                 (delta[0][brought][middle] + kModulo - current) % kModulo;
73             delta[1][brought][middle] =
74                 (delta[1][brought][middle] + current) % kModulo;
75             if (end > N)
76                 continue;
77             delta[1][brought][end] =
78                 (delta[1][brought][end] + kModulo - current) % kModulo;
79         }
80     }
81
82     fout << (1LL * total * (K - 1) + 1) % kModulo << "\n";
83
84     cout << (1LL * total * (K - 1) + 1) % kModulo << "\n";
85
86 }
```

1.2 traseu

Problema 2 - traseu

90 de puncte

O suprafață de teren de formă dreptunghiulară este divizată în N fașii orizontale și M fașii verticale, de lățimi egale. Se formează astfel $N \times M$ zone de formă pătrată, cu latura egală cu o unitate. Astfel, suprafața este reprezentată sub forma unui tablou bidimensional cu N linii și M coloane, în care pentru fiecare zonă este memorat un număr ce reprezintă altitudinea zonei respective. Interesant este că în tablou apar toate valorile $1, 2, \dots, N \cdot M$. Suprafața este destinată turismului. Deoarece spre laturile de **Est** și **Sud** ale suprafeței există peisaje de o frumusețe uimitoare, se dorește găsirea unor trasee turistice în care deplasarea să se realizeze cu pași de lungime unitară mergând doar spre **Est** și spre **Sud**. O comisie, care trebuie să rezolve această problemă, a stabilit că un traseu este atractiv dacă și numai dacă ultima poziție a traseului are altitudinea mai mare decât prima poziție a traseului. Un traseu poate începe, respectiv se poate încheia, în oricare dintre zonele terenului, cu respectarea condițiilor anterioare.

Cerințe

Se cere să se determine numărul maxim Z de zone pe care le poate avea un traseu atractiv.

Date de intrare

În fișierul de intrare **traseu.in** se află scrise pe prima linie numerele N și M , cu semnificația din enunț. Pe fiecare dintre următoarele N linii se află scrise câte M numere naturale, reprezentând, elementele tabloului bidimensional precizat în enunț. Numerele aflate pe aceeași linie a fișierului sunt separate prin câte un spațiu.

Date de ieșire

În fișierul de ieșire **traseu.out** se va scrie numărul Z , cu semnificația din enunț. Dacă nu există niciun traseu atractiv, atunci se va scrie 0.

Restricții și precizări

- $1 \leq N \leq 500$
- $1 \leq M \leq 500$
- Pentru teste în valoare de 40 de puncte, $N \leq 50$ și $M \leq 50$

Exemple

traseu.in	traseu.out	Explicații
3 4 12 11 10 6 7 5 4 3 9 2 8 1	4	Traseele atractive de lungime 2 sunt: 7-9, 4-8, 2-8 Traseele atractive de lungime 3 sunt: 5-2-8, 5-4-8 Traseele atractive de lungime 4 (maximă) sunt: 7-5-4-8, 7-5-2-8, 7-9-2-8.
3 3 5 8 7 9 6 4 3 1 2	3	Traseele atractive de lungime 2 sunt: 5-8, 5-9, 1-2 Traseele atractive de lungime 3 (maximă) sunt: 5-9-6,5-8-6,5-8-7

Timp maxim de executare/test: **1.0** secunde

Memorie: total **64 MB** din care pentru stivă **8 MB**

Dimensiune maximă a sursei: **15 KB**

Sursa: traseu.cpp, traseu.c sau traseu.pas va fi salvată în folderul care are drept nume ID-ul tău.

1.2.1 Indicații de rezolvare

Descriere a unei/unor soluții posibile

Gradul de dificultate: 2

Structuri de date utilizate: matrice, vectori, alocare dinamică, liste dublu înălțuite

Soluția 1 - 40 puncte – $O(N * M * N * M)$ (sursa: traseu40.cpp)

Pentru fiecare poziție (x_1, y_1) din matrice se parcurg pozițiile (x_2, y_2) aflate spre **Sud** și spre **Est**, $x_1 \leq x_2 \leq N$ și $y_1 \leq y_2 \leq M$ și se verifică dacă $a[x_1][y_1] < a[x_2][y_2]$.

Soluția 2 - 90 puncte – $O(N * M * N)$ (sursa: traseu90_1.cpp)

Faptul că avem numere distincte și că apar toate numerele de la 1 la $N * M$, simplifică oarecum problema.

Putem parcurge valorile în ordine crescătoare și pentru o anumită valoare v , vom încerca să găsim un traseu optim ce începe la poziția valorii v . Fie această poziție (p, q) . Evident, finalul unui astfel de traseu se va afla în submatricea $A[p..N][q..M]$.

Dacă eliminăm din matrice valorile tratate anterior, cu excepția valorii v , toate valorile rămase vor fi mai mari, și deci vor satisface monotonia de altitudine din enunț. Avem cu o constrângere mai puțin.

Astfel, vom modela liniile matricei cu *liste înlănțuite*, preferabil *dublu înlănțuite* și *circulare*, pentru a avea acces ușor la ultimul element al liniei. Avem deci un vector de liste $\text{row}[1..N]$.

Fiecare nod modelează o celulă din matrice și va conține (r, c) , linia și coloana celulei respective.

Pentru a putea șterge un nod, cel ce corespunde valorii v , este util să știm pentru fiecare valoare $1..N * M$ ce nod îi corespunde. Acest lucru se poate face cu ușurință în etapa de citire a datelor din fișier, atunci când se construiesc liste.

Pozиїile ce candidează pentru finalul traseului ce începe la poziția (p, q) , se pot afla pe liniile $p, p + 1, \dots, N$. Întrucât vrem ca traseele să fie cât mai lungi, pentru fiecare dintre aceste linii va fi suficient să analizăm doar ultimul element al listei, i.e. cel mai apropiat de capătul dreapta al matricei. Mai rămâne de verificat dacă avem un drum valid, adică coloana nodului candidat este $\geq q$.

Reiese ușor că avem un algoritm $O(N * M * N)$. O apariție destul de neașteptată a *listelor înlănțuite*!

Soluția 3 - 90 puncte – $O(N * M * \min(M, N))$ (sursa: traseu90_2.cpp)

Să ne concentrăm atenția asupra formei 1D a problemei, i.e. atunci când avem o singură linie.

Astfel, presupunem că avem un vector $A[1..n]$ cu elemente distincte și dorim să identificăm două pozиїi $i < j$ cu $A[i] < A[j]$, care maximizează $j - i$.

O primă observație e că dacă avem $A[p] < A[q]$ cu $p < q$, o soluție (q, r) , $q < r$ nu ar fi optimă întrucât (p, r) este o soluție mai bună, $r - p > r - q$. Putem concluziona că lista candidaților, ca puncte de start, este *descrescătoare* (de la stânga la dreapta).

În consecință, dacă poziția p este un candidat bun de start, atunci $A[p]$ va trebui să fie mai mic decât toate elementele din stânga lui.

O observație similară în ceea ce privește punctele de oprire arată că dacă $A[p] < A[q]$ cu $p < q$, atunci o soluție (r, p) cu $r < p$ ar fi suboptimă întrucât (r, q) este superioară. Din nou avem de-a face cu o listă descrescătoare de candidați. Un candidat p ca punct de oprire este bun, dacă $A[p]$ este mai mare decât toate valorile din dreapta lui.

Bazându-ne pe aceste observații putem defini:

$$\begin{aligned} \text{left}[i] &= \min A[k] | 1 \leq k \leq i \\ \text{right}[i] &= \max A[k] | i \leq k \leq n \end{aligned}$$

Calculul acestor vectori se face pur și simplu prin determinarea minimului respectiv a maximului prin două parcurgeri, una de la stânga la dreapta și una în sens invers.

Acum putem folosi "two pointers technique". Pornim cu $i = j = 1$, și pentru un i fixat, atât timp cât $\text{left}[i] < \text{right}[j]$ putem incrementa j , extinzând astfel soluția curentă. Când vom trece la următorul candidat $i' > i$ cu $\text{left}[i'] < \text{left}[i]$, din cauza monotoniei se vede că j nu va mai trebui resetat, putând continua din poziția lui curentă. Avem astfel o soluție $O(N)$.

Să încercăm mai departe să adaptăm soluția 1D la varianta bidimensională.

O abordare foarte utilă în multe situații, este fixarea a două linii u și v , uv , și căutarea soluțiilor ce au punctul de *start* pe linia u și punctul de *stop* pe linia v .

Odată cu fixarea celor două linii, am stabilit numărul de celule vizitate prin pași *Nord* –> *Sud*, rămânând de rezolvat problema pe direcția *Vest* –> *Est*.

Similar cu soluția anterioară, putem calcula $\text{left}[]$ în linia u , respectiv $\text{right}[]$ în linia v .

Implementarea deține similar, în plus pentru o soluție validă trebuie să fie satisfăcută relația $i \leq j$.

Complexitatea finală devine $O(M * N * \min(M, N))$, dacă transpunem matricea *a.i.* numărul liniilor să fie mai mic decât numărul coloanelor, în cazul în care acest lucru nu este deja respectat. De observat că prin transpunere, liniile devin coloane și reciproc, și în consecință vom schimba direcțiile de mers, Sud devenind Est și invers, însă lungimea traseului optim nu se schimbă.

Soluția 4 - 90 puncte – $O(N * M * (M + N))$ (sursa: traseu90_3.cpp)

Similar cu soluția precedentă, avem că un traseu ce începe la poziția (p, q) nu poate fi optim, dacă există (p', q') , $p' \leq p$, $q' \leq q$ și $A[p'][q'] < A[p][q]$. În acest caz traseul ar putea începe la poziția (p', q') și ar fi mai lung. Deci, o poziție (p, q) reprezintă un candidat bun ca punct de start, dacă $A[p][q]$ este mai mic decât toate elementele din subtabloul $A[1..p][1..q]$, bineîntăles exceptând $A[p][q]$.

Conform observației de mai sus, are sens să definim:

$$\min[i][j] = \min\{A[p][q] | 1 \leq p \leq i \& 1 \leq q \leq j\}$$

Se observă că minimul nu are cum să crească dacă ne deplasăm de la stânga la dreapta sau de sus în jos. Deci, liniile și coloanele matricei $\min[]$ sunt descrescătoare. Avem practic un "tablou Young" descrescător, i.e. $\min[i-1][j] \geq \min[i][j]$, $\min[i][j-1] \geq \min[i][j]$.

Tablourile Young au numeroase proprietăți interesante, una dintre ele fiind faptul că operația de căutare a unei valori se poate efectua în timp liniar, $O(M + N)$. Vom adapta ideea din spatele acestui algoritm mai jos.

În cazul problemei noastre, pentru fiecare poziție (i, j) vom încerca să identificăm un traseu optim ce se termină în această poziție. Punctul de start va fi poziționat evident în submatricea $A[1..i][1..j]$.

Distingem următoarele cazuri:

(a) $\min[1][j] < A[i][j]$ – Putem renunța să căutăm candidați în coloana j , întrucât $A[1][j]$ este cel mai depărtat posibil și verifică proprietatea din enunț.

(b) $\min[1][j] > A[i][j]$ – Putem renunța să căutăm candidați în linia 1, întrucât din cauza monotoniei lui $\min[]$, valorile din stânga lui $\min[1][j]$ sunt mai mari.

(c) cazul de egalitate poate fi evitat, întrucât valorile sunt distințe și $A[i][j]$ nu poate fi egal cu alte elemente.

Deci putem elibera fie o linie, fie o coloană, reducând zona de căutare la una mai mică. Procesul se repetă în această manieră până când zona de căutare devine vidă. Mai concret, se pornește cu colțul dreapta-sus, $(p, q) = (1, j)$, și în urma comparației dintre $\min[p][q]$ și $A[i][j]$, fie se incrementează p , ori se decrementează q . Se vor face maxim $i + j$ pași, ceea ce conduce la o soluție finală $O(M * N * (M + N))$.

Solutia 5 - 90 puncte – $O(N * M * N)$ (sursa: traseu90_4.cpp)

Vom procesa elementele în ordine crescătoare, și dacă la un moment dat avem o valoarea v , aflată la coordonatele (x, y) vrem să aflăm cel mai lung traseu care se termină la această poziție.

Asta înseamnă că trebuie ca traseul să înceapă la una din valorile mai mici, valori care au fost deja procesate. Iar dintre acestea se dorește cea la distanță maximă de (x, y) mai sus și mai la stânga.

Putem ține astfel pentru fiecare linie cea mai din stânga poziție procesată: $best[i] = y_{min}$ astfel încât (i, y) a fost deja procesat

Când suntem la valoarea v , de la poziția (x, y) putem verifica toate liniile de la 1 la x , și importantă este doar cea mai din stânga poziție procesată.

Deci pentru linia i , ne uitam la $best[i]$

(a) Dacă $best[i] \leq y$, atunci traseul care începe la $(i, best[i])$ și se termină la (x, y) e un potențial traseu maxim. Dacă este mai lung decât cel mai lung traseu găsit până la momentul actual actualizăm răspunsul.

(b) Dacă $best[i] > y$, sau $best[i]$ încă nu a fost calculat nu facem nimic.

Deoarece pentru fiecare valoare s-au procesat maxim N liniii complexitatea finală este $O(N * M * N)$.

1.2.2 *Rezolvare detaliată**1.2.3 Cod sursă**

Listing 1.2.1: traseu40.cpp

```

1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
```

```

4
5  using namespace std;
6
7  int main()
8  {
9      ifstream cin("traseu.in");
10     ofstream cout("traseu.out");
11
12     int N, M; cin >> N >> M;
13
14     vector< vector<int> > V(N, vector<int>(M));
15     for (int i = 0; i < N; ++i)
16         for (int j = 0; j < M; ++j)
17             cin >> V[i][j];
18
19     int answer = 0;
20     for (int i = 0; i < N; ++i)
21         for (int j = 0; j < M; ++j)
22             for (int k = i; k < N; ++k)
23                 for (int l = j; l < M; ++l)
24                     if (V[i][j] < V[k][l])
25                         answer = max(answer, k - i + l - j);
26
27     if (answer == 0)
28         answer = -1;
29
30     cout << answer + 1 << "\n";
31 }

```

Listing 1.2.2: traseu90_1.cpp

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct NODE
5 {
6     int r, c, v;
7     struct NODE *prev;
8     struct NODE *next;
9 };
10
11 void ddelete(struct NODE *n)
12 {
13     n->prev->next = n->next;
14     n->next->prev = n->prev;
15     free(n);
16 }
17
18 struct NODE* insert(struct NODE *prev, int r, int c, int v)
19 {
20     struct NODE *next = prev->next;
21     struct NODE *n = (struct NODE *)malloc(sizeof(struct NODE));
22     n->r = r; n->c = c; n->v = v;
23     n->next = next;
24     n->prev = prev;
25     next->prev = n;
26     prev->next = n;
27     return n;
28 }
29
30 int main()
31 {
32     FILE *f = fopen("traseu.in", "r");
33     FILE *g = fopen("traseu.out", "w");
34
35     struct NODE row[500];
36     struct NODE *NODE[250000];
37
38     int m, n, v;
39
40     fscanf(f, "%d %d", &m, &n);
41     for (int i = 0; i < m; i++)
42     {
43         row[i].next = row[i].prev = &row[i];
44         for (int j = 0; j < n; j++)

```

```

45         {
46             fscanf(f, "%d", &v); v--;
47             NODE[v] = insert(row[i].prev, i, j, v);
48         }
49     }
50
51     int ans = 0;
52     for (int v = 0; v < m*n; v++)
53     {
54         struct NODE *x = NODE[v];
55         for (int i = x->r; i < m; i++)
56         {
57             struct NODE *y = row[i].prev;
58             int d = i - x->r + y->c - x->c;
59             if (y->next != y && x->c <= y->c && d > ans) ans = d;
60         }
61         ddelete(x);
62     }
63
64     if(ans == 0)fprintf(g, "0");
65     else fprintf(g, "%d", ans + 1);
66
67     fclose(f);
68     fclose(g);
69
70     return 0;
71 }
```

Listing 1.2.3: traseu90_2.cpp

```

1 #include <stdio.h>
2
3 int a[500][500], l[500][500], r[500][500];
4
5 int main()
6 {
7     FILE *f = fopen("traseu.in", "r");
8     FILE *g = fopen("traseu.out", "w");
9     int m, n, x, ok;
10
11    fscanf(f, "%d %d", &m, &n);
12    for (int i = 0; i < m; i++)
13    {
14        for (int j = 0; j < n; j++)
15        {
16            fscanf(f, "%d", &x);
17            if (m < n) a[i][j] = x; else a[j][i] = x;
18        }
19    }
20
21    if (n <= m) m ^= n, n ^= m, m ^= n;
22
23    ok=1;
24    for(int i=0;i<m;i++)
25    {
26        for(int j=0;j<n;j++)
27        {
28            if((j+1<n && a[i][j]<a[i][j+1]) ||
29                (i+1<m && a[i][j]<a[i+1][j]))
30            {
31                ok=0;
32                i=m;
33                j=n;
34            }
35        }
36    }
37
38    if(ok==1)
39    {
40        fprintf(g,"0");
41        fclose(g);
42        return 0;
43    }
44
45    for (int i = 0; i < m; i++)
```

```

46         for (int j = 0; j < n; j++)
47             l[i][j] = (j == 0 || a[i][j] < l[i][j-1]) ? a[i][j] : l[i][j-1];
48
49     for (int i = 0; i < m; i++)
50         for (int j = n-1; j >= 0; j--)
51             r[i][j] = (j == n-1 || a[i][j] > r[i][j+1]) ? a[i][j] : r[i][j+1];
52
53     int ans = 0;
54     for (int i = 0; i < m; i++)
55     {
56         for (int j = i; j < m; j++)
57         {
58             for (int p = 0, q = 0; p < n; p++)
59             {
60                 while (q+1 < n && l[i][p] <= r[j][q+1])
61                     q++;
62                 if (p <= q && l[i][p] <= r[j][q] && ans < j-i+q-p)
63                     ans = j-i+q-p;
64             }
65         }
66     }
67
68     fprintf(g, "%d", ans + 1);
69
70     fclose(f);
71     fclose(g);
72
73     return 0;
74 }
```

Listing 1.2.4: traseu90_3.cpp

```

1 // O(m*n*(m + n))
2 #include <stdio.h>
3
4 int a[500][500];
5
6 int main()
7 {
8     FILE *f = fopen("traseu.in", "r");
9     FILE *g = fopen("traseu.out", "w");
10
11    int m, n, x, ans = 0;
12
13    fscanf(f, "%d %d", &m, &n);
14    for (int i = 0; i < m; i++)
15    {
16        for (int j = 0; j < n; j++)
17        {
18            fscanf(f, "%d", &x);
19            a[i][j] = x;
20            if (i && a[i-1][j] < a[i][j]) a[i][j] = a[i-1][j];
21            if (j && a[i][j-1] < a[i][j]) a[i][j] = a[i][j-1];
22            int p = 0, q = j;
23            while (p <= i && q >= 0)
24            {
25                if (a[p][q] <= x)
26                {
27                    if (i-p+j-q > ans)
28                        ans = i-p+j-q;
29                    q--;
30                }
31                else
32                    p++;
33            }
34        }
35    }
36
37    fprintf(g, "%d", ans ? (ans + 1) : 0);
38
39    fclose(f);
40    fclose(g);
41
42    return 0;
43 }
```

Listing 1.2.5: traseu90_4.cpp

```

1 #include <fstream>
2 #include <iostream>
3 #include <vector>
4 #include <algorithm>
5 #include <cassert>
6
7 using namespace std;
8
9 const static int kMaxN = 1000;
10
11 int main()
12 {
13     ifstream cin("traseu.in");
14     ofstream cout("traseu.out");
15
16     int N, M; assert(cin >> N >> M);
17     assert(1 <= N && N <= kMaxN);
18     assert(1 <= M && M <= kMaxN);
19
20     vector< pair<int, int> > pos(N * M, make_pair(-1, -1));
21
22     for (int i = 0; i < N; ++i)
23         for (int j = 0; j < M; ++j)
24         {
25             int x; assert(cin >> x);
26             assert(1 <= x && x <= N * M);
27             --x;
28             assert(pos[x] == make_pair(-1, -1));
29             pos[x] = make_pair(i, j);
30         }
31
32     vector<int> best(N, M);
33     int answer = -1;
34     for (int i = 0; i < N * M; ++i)
35     {
36         int x, y; tie(x, y) = pos[i];
37         for (int j = 0; j <= x; ++j)
38             if (best[j] <= y)
39                 answer = max(answer, x - j + y - best[j]);
40
41         best[x] = min(best[x], y);
42     }
43
44     cout << answer + 1 << "\n";
45 }
```

Listing 1.2.6: traseu90_5.cpp

```

1 //O((n*m*n)
2 //parcurea crescatoare a valorilor,
3 //eliminarea elementelor vizitate
4 //si compararea doar cu ultimul element din linie
5 #include <fstream>
6
7 using namespace std;
8
9 ifstream fin("traseu.in");
10 ofstream fout("traseu.out");
11
12 struct pozitie
13 {
14     short l,c;
15 };
16
17 int main()
18 {
19     int n,m;
20     fin>>n>>m;
21     pozitie f[n*m+5];
22     short pleft[n+5][m+5],pright[n+5][m+5];
23     int a[n+5][m+5],d,dm,i,j,i1,i2,j1,j2;
24
25     for(i=1;i<=n;i++)
```

```

26     {
27         pright[i][0]=1;
28         for(j=1; j<=m; j++)
29         {
30             fin>>a[i][j];
31             pleft[i][j]=j-1;
32             pright[i][j]=j+1;
33             f[a[i][j]]={i,j};
34         }
35         pleft[i][m+1]=m;
36     }
37     dm=0;
38     for(i=1; i<=n*m; i++)
39     {
40         i1=f[i].l; j1=f[i].c;
41         pleft[i1][pright[i1][j1]]=pleft[i1][j1];
42         pright[i1][pleft[i1][j1]]=pright[i1][j1];
43         for(i2=i1; i2<=n; i2++)
44         {
45             j2=pleft[i2][m+1];
46             if(j1<=j2){
47                 dm=max(dm, i2-i1+j2-j1);
48             }
49         }
50     }
51
52     if(dm==0)
53         fout<<0;
54     else
55         fout<<dm+1;
56
57     fout.close();
58     fin.close();
59     return 0;
60 }
```

1.3 yinyang

Problema 3 - yinyang

90 de puncte

Se dă o matrice A cu N linii și M coloane, cu valori cuprinse între 1 și $N \cdot M$ inclusiv, nu neapărat distințe. O operație constă în selectarea a două linii sau două coloane consecutive și interschimbarea acestora (swap). O matrice **yin-yang** este o matrice în care $A[i][j] \geq A[i][j-1]$, pentru orice pereche (i, j) cu $1 \leq i \leq N$ și $2 \leq j \leq M$ și $A[i][j] \geq A[i-1][j]$, pentru orice pereche (i, j) cu $2 \leq i \leq N$ și $1 \leq j \leq M$.

Cerințe

Să se determine numărul minim de operații necesare pentru a transforma matricea dată într-o matrice **yin-yang**.

Date de intrare

În fișierul de intrare **yinyang.in** se află scrise pe prima linie numerele naturale N și M , cu semnificația din enunț. Pe fiecare dintre următoarele N linii se află câte M numere naturale, reprezentând elementele matricei date A . Numerele aflate pe aceeași linie a fișierului sunt separate prin câte un spațiu.

Date de ieșire

În fișierul **yinyang.out** se va scrie numărul minim de operații cerut sau -1 dacă nu există soluție.

Restricții și precizări

- $1 \leq N, M \leq 100$
- Pentru teste în valoare de 9 puncte: $1 \leq N, M \leq 5$
- Pentru alte teste în valoare de 18 puncte: $N = 1$
- Pentru alte teste în valoare de 36 de puncte elementele din matrice sunt DISTINCTE

Exemplu

yinyang.in	yinyang.out	Explicații								
2 3 1 2 4 3 5 6	0	Matricea dată este matrice yin-yang								
2 3 6 6 5 4 6 2	3	Operațiile pot fi următoarele: swap(linia 1 , linia 2), swap(coloana 2, coloana 3), swap(coloana 1, coloana 2). Matricea dată va ajunge la final în forma yin-yang: <table style="margin-left: auto; margin-right: auto;"> <tr> <td style="border: 1px solid black; padding: 5px; text-align: center;">6 6 5</td> <td style="border: 1px solid black; padding: 5px; text-align: center;">4 6 2</td> <td style="border: 1px solid black; padding: 5px; text-align: center;">4 2 6</td> <td style="border: 1px solid black; padding: 5px; text-align: center;">2 4 6</td> </tr> <tr> <td style="border: 1px solid black; padding: 5px; text-align: center;">4 6 2</td> <td style="border: 1px solid black; padding: 5px; text-align: center;">6 6 5</td> <td style="border: 1px solid black; padding: 5px; text-align: center;">6 5 6</td> <td style="border: 1px solid black; padding: 5px; text-align: center;">5 6 6</td> </tr> </table>	6 6 5	4 6 2	4 2 6	2 4 6	4 6 2	6 6 5	6 5 6	5 6 6
6 6 5	4 6 2	4 2 6	2 4 6							
4 6 2	6 6 5	6 5 6	5 6 6							

Timp maxim de executare/test: 1.0 secunde

Memorie: total 64 MB din care pentru stivă 8 MB

Dimensiune maximă a sursei: 15 KB

Sursa: yinyang.cpp, yinyang.c sau yinyang.pas va fi salvată în folderul care are drept nume ID-ul tău.

1.3.1 Indicații de rezolvare

Descriere a unei/unor soluții posibile

Structurile de date utilizate: matrici

Gradul de dificultate: 2

Soluție - 90 puncte - Complexitate $O(N^3)$ (sursa: yinyang90.cpp)

O primă observație este că dacă avem 2 elemente X și Y aflate pe aceeași linie sau aceeași coloană, indiferent ce operații aplicăm asupra matricei, cele 2 elemente vor rămâne pe aceeași linie sau coloană.

Astfel, putem deduce că trebuie să existe o *relație de ordine* între liniile și coloanele matricei. Mai exact, pentru oricare două linii $L1$ și $L2$ din matrice, avem $A[L1][x] \leq A[L2][x]$ pentru orice coloană x de la 1 la M , sau avem $A[L2][x] \leq A[L1][x]$ pentru orice coloană x de la 1 la M .

Indiferent de operațiile pe coloane pe care le aplicăm, relațiile între elementele de pe aceeași coloană din cele două linii nu se vor schimba. În concluzie, putem spune că linia cu elementele mai mici este "mai mică" și aceasta trebuie să apară înaintea liniei "mai mari" în matrice.

Presupunem fără a restrânge generalitatea problemei că linia $L1$ este "mai mică" decât linia $L2$. Dacă $L1$ apare după linia $L2$, atunci spunem că aceste linii formează o inversiune, inversiune care trebuie să fie rezolvată la un moment dat printr-o operație de interschimbare între linii. Analog pentru coloane.

În concluzie, rămâne de aflat care este numărul de inversiuni atât pentru linii, cât și pentru coloane în matrice. Din moment ce restricțiile sunt foarte mici, putem fixa pe rând fiecare pereche de linii $L1$ și $L2$ (cu $L1 < L2$). Dacă linia $L1$ este "mai mare" decât linia $L2$, liniile prezintă o inversiune. Dacă $L1$ este linia "mai mică", nu este nevoie de nici o operație.

Dacă există două linii $L1$ și $L2$ care nu se află în relație de ordine (nu putem spune că $L1$ este "mai mică" sau "mai mare" decât $L2$), atunci nu avem soluție. Formal, nu avem soluție dacă și numai dacă există două linii $L1$ și $L2$ diferite și două coloane diferite X și Y pentru care avem $A[L1][X] < A[L2][X]$ și $A[L1][Y] > A[L2][Y]$.

Din moment ce fixăm fiecare pereche de linii/coloane, iar verificarea condiției de existență este $O(N)$, complexitatea finală va fi $O(N^3)$.

1.3.2 *Rezolvare detaliată

1.3.3 Cod sursă

Listing 1.3.1: yinyang45.cpp

¹ `#include<stdio.h>`

```

2
3 #define NMAX 105
4
5 int n, m, answer;
6 int matrix[NMAX][NMAX];
7
8 int main ()
9 {
10    freopen("yinyang.in", "r", stdin);
11    freopen("yinyang.out", "w", stdout);
12
13    scanf("%d%d", &n, &m);
14
15    for(int i = 1; i <= n; i++)
16        for(int j = 1; j <= m; j++)
17            scanf("%d", &matrix[i][j]);
18
19    for(int i = 1; i <= m; i++)
20        for(int j = i + 1; j <= m; j++)
21            if(matrix[1][i] > matrix[1][j])
22                answer++;
23
24    for(int i = 1; i <= n; i++)
25        for(int j = i + 1; j <= n; j++)
26            if(matrix[i][1] > matrix[j][1])
27                answer++;
28
29    printf("%d\n", answer);
30
31    return 0;
32 }
```

Listing 1.3.2: yinyang90_1.cpp

```

1 #include<stdio.h>
2 #include<algorithm>
3 #include<cassert>
4
5 using namespace std;
6
7 #define NMAX 104
8
9 int n, m, matrix[NMAX][NMAX], newMatrix[NMAX][NMAX];
10 int lineIndex[NMAX], columIndex[NMAX];
11
12 void readInput()
13 {
14     scanf("%d%d", &n, &m);
15     assert(1 <= n && n <= 100 && 1 <= m && m <= 100);
16     for(int i = 1; i <= n; i++)
17         for(int j = 1; j <= m; j++)
18         {
19             scanf("%d", &matrix[i][j]);
20             assert(1 <= matrix[i][j] && matrix[i][j] <= n * m);
21         }
22 }
23
24 int cmpLine(const int& a, const int& b)
25 {
26     for(int i = 1; i <= m; i++)
27         if(matrix[a][i] != matrix[b][i])
28             return matrix[a][i] < matrix[b][i];
29     return a < b;
30 }
31
32 int cmpColum(const int& a, const int& b)
33 {
34     for(int i = 1; i <= n; i++)
35         if(matrix[i][a] != matrix[i][b])
36             return matrix[i][a] < matrix[i][b];
37     return a < b;
38 }
39
40 void sortMatrix()
41 {
```

```

42     for(int i = 1; i <= n; i++)
43         lineIndex[i] = i;
44
45     sort(lineIndex + 1, lineIndex + n + 1, cmpLine);
46
47     for(int i = 1; i <= m; i++)
48         columIndex[i] = i;
49
50     sort(columIndex + 1, columIndex + m + 1, cmpColum);
51
52     for(int i = 1; i <= n; i++)
53         for(int j = 1; j <= m; j++)
54             newMatrix[i][j] = matrix[lineIndex[i]][columIndex[j]];
55 }
56
57 bool checkSolution()
58 {
59     for(int i = 1; i <= n; i++)
60     {
61         for(int j = 1; j <= m; j++)
62         {
63             if(newMatrix[i][j] < newMatrix[i - 1][j] ||
64                newMatrix[i][j] < newMatrix[i][j - 1])
65                 return false;
66         }
67     }
68     return true;
69 }
70
71 void writeOutput()
72 {
73     if(!checkSolution())
74     {
75         printf("-1\n");
76         return ;
77     }
78     int answer = 0;
79
80     for(int i = 2; i <= n; i++)
81         for(int j = 1; j < i; j++)
82             answer += (lineIndex[i] < lineIndex[j]);
83
84     for(int i = 2; i <= m; i++)
85         for(int j = 1; j < i; j++)
86             answer += (columIndex[i] < columIndex[j]);
87
88     printf("%d\n", answer);
89 }
90
91 int main ()
92 {
93     freopen("yinyang.in", "r", stdin);
94     freopen("yinyang.out", "w", stdout);
95
96     readInput();
97     sortMatrix();
98     writeOutput();
99
100    return 0;
101 }
```

Listing 1.3.3: yinyang90_2.cpp

```

1 //yin-yang
2 ///O(n*m*(n+m))
3 #include<stdio.h>
4
5 int n,m,a[1002][1002],i,j,k,x,nr,c1,c2;
6
7 int main()
8 {
9     FILE *fin,*fout;
10    fin= fopen("yinyang.in","rt");
11    fout=fopen("yinyang.out","wt");
12
```

```

13     fscanf(fin, "%d %d\n", &n, &m);
14     for(i=1; i<=n; i++)
15     {
16         for(j=1; j<=m; j++)
17         {
18             fscanf(fin, "%d ", &x);
19             a[i][j]=x;
20         }
21     }
22
23     nr=0;
24     for(i=1; i<=n-1; i++)
25     {
26         for(j=i+1; j<=n; j++)
27         {
28             int r=0,c1=0,c2=0;
29             for(k=1; k<=m; k++)
30             {
31                 if(a[i][k]<a[j][k])
32                 {
33                     if(r==0) r=-1;
34                     c1++;
35                 }
36                 if(a[i][k]>a[j][k])
37                 {
38                     if(r==0) r+=1;
39                     c2++;
40                 }
41             }
42             if(c1 && c2)
43             {
44                 fprintf(fout, "-1");
45                 fclose(fout);
46                 fclose(fin);
47                 return 0;
48             }
49         }
50         if(r>0) nr++;
51     }
52 }
53
54     for(i=1; i<=m-1; i++)
55     {
56         for(j=i+1; j<=m; j++)
57         {
58             int r=0,c1=0,c2=0;
59             for(k=1; k<=n; k++)
60             {
61                 if(a[k][i]<a[k][j])
62                 {
63                     if(r==0) r=-1;
64                     c1++;
65                 }
66
67                 if(a[k][i]>a[k][j])
68                 {
69                     if(r==0) r+=1;
70                     c2++;
71                 }
72             }
73         }
74         if(c1 && c2)
75         {
76             fprintf(fout, "-1");
77             fclose(fout);
78             fclose(fin);
79             return 0;
80         }
81     }
82     if(r>0) nr++;
83 }
84
85     fprintf(fout, "%d",nr);
86     fclose(fout); fclose(fin);

```

```
89     return 0;
90 }
```

Capitolul 2

OJI 2018

2.1 castel

Problema 1 - castel

100 de puncte

Arheologii au descoperit pe un platou muntos greu accesibil ruinele unui castel medieval, pe care l-au fotografiat din elicopter, obținând harta digitizată a acestuia. Harta este memorată sub forma unui tablou bidimensional H , compus din $N \times N$ pătrate cu latura egală cu unitatea, având ca elemente numere naturale între 0 și 15, care codifică forma pereților fiecarui pătrat unitar. Dacă scriem numărul natural $H[i][j]$ în baza 2, folosind exact 4 cifre binare, fiecare bit dă informații despre unul dintre pereții posibil de construit pe fiecare latură a păratului unitar din poziția (i, j) , astfel:

- dacă bitul de pe poziția 0 are valoarea 1, atunci există perete pe latura vestică (latura din stânga);
- dacă bitul de pe poziția 1 are valoarea 1, atunci există perete pe latura sudică (latura de jos);
- dacă bitul de pe poziția 2 are valoarea 1, atunci există perete pe laturaestică (latura din dreapta);
- dacă bitul de pe poziția 3 are valoarea 1, atunci există perete pe latura nordică (latura de sus);
- un bit de valoare 0 indică lipsa peretelui corespunzător acestuia;

Pentru un număr scris în baza 2, numerotarea cifrelor începe cu poziția 0, de la dreapta la stânga.

Castelul este interesant deoarece, pentru realizarea unei mai bune apărări, camerele ce-l compun sunt construite fie independent, fie una în interiorul alteia. Orice camera este construită la o distanță de cel puțin o unitate față de zidul ce împrejmuiște castelul sau față de pereții altor camere.

Folosind harta, arheologii doresc să afle informații privind numărul camerelor și camera de arie maximă. Prin arie a unei camere se înțelege numărul păratelor unitate cuprinse în interiorul pereților acesteia, fără a socoti ariile camerelor construite în interiorul ei.

Cerințe

Cunoscând codificarea hărții castelului, să se determine:

1. numărul total al camerelor din castel
2. aria maximă a unei camere
3. coordonatele colțurilor din stânga-sus, respectiv dreapta-jos a camerei cu aria maximă. Dacă există mai multe camere având aceeași arie maximă, atunci se vor afișa coordonatele camerei având colțul din stânga-sus ($lin1, col1$) cu $lin1$ minimă, iar la linii egale pe aceea cu $col1$ minimă.

Date de intrare

Datele de intrare se citesc din fișierul **castel.in**, care are următoarea structură:

- Pe prima linie se află numărul natural C , care poate fi egal cu 1, 2 sau 3, în funcție de cerința ce trebuie rezolvată;
- Pe linia următoare se află numărul natural N , reprezentând dimensiunea hărții;
- Pe următoarele N linii se găsesc câte N numere naturale din intervalul $[0, 15]$, separate prin căte un spațiu, reprezentând harta castelului.

Date de ieșire

Datele de ieșire se vor scrie în fișierul **castel.out**, astfel:

- Dacă $C = 1$, pe prima linie se va scrie numărul total al camerelor din castel;
- Dacă $C = 2$, pe prima linie se va scrie aria maximă a unei camere din castel;
- Dacă $C = 3$, pe prima linie se vor scrie 4 numere naturale $lin1\ col1\ lin2\ col2$, separate prin câte un spațiu, reprezentând coordonatele colțurilor din stânga-sus, respectiv dreapta-jos ale camerei de arie maximă.

Restricții și precizări

- $2 < n \leq 100$
- Se garantează că în castel există cel puțin o cameră;
- Testele care au $C = 1$ totalizează 20 de puncte;
- Testele care au $C = 2$ totalizează 50 de puncte;
- Testele care au $C = 3$ totalizează 20 de puncte;
- Se acordă 10 puncte din oficiu.

Exemple

castel.in	castel.out	Explicații
<pre> 1 9 0 2 0 0 0 0 0 0 0 4 1 5 1 0 0 2 2 0 0 0 1 0 2 0 4 1 1 1 4 1 0 4 9 1 2 1 2 1 0 1 0 2 0 4 3 6 5 9 8 1 0 1 2 1 0 1 0 8 4 1 4 1 5 5 1 4 1 3 1 4 3 2 1 0 6 1 4 7 1 0 8 8 8 8 0 0 8 0 0 0 0 0 0 0 0 </pre>	6	<p>În figură este reprezentată harta castelului</p> <p>codificat în fișierul de intrare. Acesta conține 6 camere.</p>
<pre> 2 9 0 2 0 0 0 0 0 0 0 4 1 5 1 0 0 2 2 0 0 0 1 0 2 0 4 1 1 1 4 1 0 4 9 1 2 1 2 1 0 1 0 2 0 4 3 6 5 9 8 1 0 1 2 1 0 1 0 8 4 1 4 1 5 5 1 4 1 3 1 4 3 2 1 0 6 1 4 7 1 0 8 8 8 8 0 0 8 0 0 0 0 0 0 0 0 </pre>	11	Aria maximă a unei camere este 11.
<pre> 3 9 0 2 0 0 0 0 0 0 0 4 1 5 1 0 0 2 2 0 0 0 1 0 2 0 4 1 1 1 4 1 0 4 9 1 2 1 2 1 0 1 0 2 0 4 3 6 5 9 8 1 0 1 2 1 0 1 0 8 4 1 4 1 5 5 1 4 1 3 1 4 3 2 1 0 6 1 4 7 1 0 8 8 8 8 0 0 8 0 0 0 0 0 0 0 0 </pre>	5 5 7 8	Camera cu aria maximă are coordonatele (5,5) - (7,8)

Timp maxim de executare/test: **0.2** secunde

Memorie: total **64 MB** din care pentru stivă **32 MB**

Dimensiune maximă a sursei: **30 KB**

2.1.1 Indicații de rezolvare

prof. Alin Burța - Colegiul Național B.P. Hasdeu Buzău

Cerința a) se rezolvă ușor, identificând colțul din stânga-sus al fiecărei camere, după ce observăm că elementul corespunzător acestuia poate avea una dintre valorile 9, 11, 13 sau 15. Memorăm într-un tablou coordonatele astfel determinate, în vederea rezolvării punctelor următoare.

Pentru rezolvarea cerințelor b) și c), luăm pe rând fiecare cameră și îi determinăm aria folosind un *algoritm de umplere (fill)*. Pornim din elementul aflat în colțul stânga-sus al camerei curente (este cu siguranță o poziție accesibilă) și, utilizând o *coadă* sau un algoritm recursiv, parcurgem toate părțile unitare accesibile, contorizându-le. Actualizăm permanent aria maximă și camera cu această arie.

Având în vedere că fiecare pătrat unitate este vizitat o singură dată, complexitatea algoritmului este $O(n^2)$.

2.1.2 *Rezolvare detaliată

2.1.3 Cod sursă

Listing 2.1.1: castel.cpp

```

1 #include <iostream>
2 #include <fstream>
3
4 #define Nmax 102
5 #define Cmax Nmax*Nmax
6 #define InFile "castel.in"
7 #define OutFile "castel.out"
8
9 using namespace std;
10
11 short A[Nmax][Nmax], Viz[Nmax][Nmax];
12 short N;
13 short C;
14
15 short ValBit(short No, short Pos)
16 {
17     return ((1<<Pos) & No)>>Pos;
18 }
19
20 void Fill(short lin, short col, short &Aria, short &elin, short &ecol)
21 {
22     short i, j;
23     short Clin[Cmax], Ccol[Cmax];
24     short CrrLin, CrrCol, Prim, Ultim;
25
26     Aria = 0;
27     elin = lin; ecol = col;
28     Prim = 1; Ultim = 1; Clin[Ultim] = lin; Ccol[Ultim] = col;
29     while(Prim <= Ultim)
30     {
31         CrrLin = Clin[Prim]; CrrCol = Ccol[Prim++];
32         Viz[CrrLin][CrrCol] = 1;
33         Aria++;
34
35         if(elin<=CrrLin && ecol<=CrrCol)
36             elin = CrrLin, ecol = CrrCol;
37
38         if(!ValBit(A[CrrLin][CrrCol], 3) && !Viz[CrrLin-1][CrrCol]) //Nord
39         {
40             Ultim++;
41             Clin[Ultim] = CrrLin-1;
42             Ccol[Ultim] = CrrCol;
43             Viz[CrrLin-1][CrrCol] = 1;

```

```

44         }
45
46         if(!ValBit(A[CrrLin][CrrCol], 2) && !Viz[CrrLin][CrrCol+1]) //Est
47     {
48             Ultim++;
49             Clin[Ultim] = CrrLin;
50             Ccol[Ultim] = CrrCol+1;
51             Viz[CrrLin][CrrCol+1] = 1;
52         }
53
54         if(!ValBit(A[CrrLin][CrrCol], 1) && !Viz[CrrLin+1][CrrCol]) //Sud
55     {
56             Ultim++;
57             Clin[Ultim] = CrrLin+1;
58             Ccol[Ultim] = CrrCol;
59             Viz[CrrLin+1][CrrCol] = 1;
60         }
61
62         if(!ValBit(A[CrrLin][CrrCol], 0) && !Viz[CrrLin][CrrCol-1]) //Vest
63     {
64             Ultim++;
65             Clin[Ultim] = CrrLin;
66             Ccol[Ultim] = CrrCol-1;
67             Viz[CrrLin][CrrCol-1] = 1;
68         }
69     }
70 }
71
72 int main()
73 {
74     int i, j, k;
75     short Slin[Cmax], Scol[Cmax], Svf = 0;
76     short slin, scol, elin, ecol, Aria;
77     short maxil, maxic, maxfl, maxfc, AriaMax=0, NrCamere;
78     ifstream Fin(InFile);
79
80     //citire
81     Fin >> C >> N;
82     for(i=1;i<=N;i++)
83         for(j=1;j<=N;j++)
84     {
85         Fin >> A[i][j];
86         if(A[i][j] == 9 || A[i][j] == 11 ||
87             A[i][j] == 15 || A[i][j] == 13)
88         {
89             Svf++; Slin[Svf] = i; Scol[Svf] = j;
90         }
91     }
92     Fin.close();
93
94     //initializare
95     for(i=0;i<=N+1;i++)
96     {
97         Viz[0][i] = Viz[N+1][i] = 15;
98         Viz[i][0] = Viz[i][N+1] = 15;
99     }
100
101    for(i=1;i<=N;i++)
102        for(j=1;j<=N;j++)
103            Viz[i][j] = 0;
104
105    //rezolvare
106    NrCamere = Svf;
107    while(Svf)
108    {
109        slin = Slin[Svf]; scol = Scol[Svf];
110        Fill(slin, scol, Aria, elin, ecol);
111        Svf--;
112        if(Aria>=AriaMax)
113            maxil = slin, maxic = scol,
114            maxfl = elin, maxfc = ecol,
115            AriaMax = Aria;
116    }
117
118    ofstream Fou(OutFile);
119    if ( C == 1) Fou<<NrCamere<<'\n';

```

```

120     if ( C == 2 ) Fou<<AriaMax<<'\n';
121     if ( C == 3 ) Fou<<maxil<< ' <<maxic<< ' <<maxfl<< ' <<maxfc<<'\n';
122     Fou.close();
123     return 0;
124 }
```

2.2 eq4

Problema 2 - eq4

100 de puncte

Se dă o expresie matematică în care pot să apară literele x, y, z, t , cifre și semnele + sau -.

Cifrele alăturate formează numere. Literele reprezintă variabile. O variabilă poate fi precedată de un număr. Între variabilă și numărul care o precede nu există alte caractere. Un grup format dintr-o literă și, eventual, un număr care o precede formează un monom. Un monom nu conține mai multe litere. Numărul care apare într-un monom se numește coeficient.

Expresia poate să conțină și numere care nu sunt urmate de o variabilă. Aceste numere se numesc termeni liberi.

Expresia este deci alcătuită din monoame și termeni liberi. Fiecare monom și fiecare termen liber este precedat de unul dintre semnele + sau -.

Exemple:

Expresii corecte	Expresii incorecte
$-x + 100$	$x + 100$ (x nu este precedat de + sau -)
$+3x + 2y - 3z + 7x - 15 - 3 + 8z - 7y$	$+x + y - 3zt$ ($3zt$ nu este monom, deoarece conține două litere)
$+10x-7y+3x-7+5z-8t-z-x-y+3$	$-x + y - 34*t + 5z - 5u$ (în expresie apar caractere nepermise, în acest caz spații, litera u și semnul *)

Valoarea matematică a unei expresii este valoarea care se obține dacă înlocuim literele care apar în expresie cu valori numerice și efectuăm calculele. Valoarea unui monom se obține înmulțind coeficientul monomului cu valoarea pe care o are variabila care apare în respectivul monom. De exemplu, valoarea expresiei $+3x^4$ pentru $x = 2$ este 6.

Cerințe

Fiind dată o expresie corectă, să se determine:

1. valoarea matematică a expresiei dacă x, y, z și t au valoarea 1.
2. numărul de cvasitete distincte (x, y, z, t) , de valori întregi care aparțin unui interval dat $[a, b]$, pentru care expresia matematică corespunzătoare expresiei date este egală cu o valoare dată E . Două cvasitete sunt distincte dacă există cel puțin o poziție pentru care valorile corespunzătoare sunt diferite.

Date de intrare

Datele de intrare se citesc din fișierul **eq4.in**, care are următoarea structură:

- pe prima linie se află numărul natural C , care poate fi egal cu 1 sau 2, în funcție de cerința ce trebuie rezolvată;
- pe a doua linie se află expresia dată;
- pe a treia linie se află valorile a și b E , separate prin câte un spațiu.

Date de ieșire

Datele de ieșire se vor scrie în fișierul **eq4.out** astfel:

- Dacă $C = 1$, pe prima linie se va scrie răspunsul la cerința 1;
- Dacă $C = 2$, pe prima linie se va scrie răspunsul la cerința 2.

Restricții și precizări

- coeficienții sunt numere naturale, având cel mult 4 cifre
- $2 \leq$ lungimea expresiei ≤ 100000
- $-500 \leq a \leq b \leq 500$
- $-10^{15} \leq E \leq 10^{15}$
- Testele care au $C = 1$ totalizează 20 de puncte;

- Testele care au $C = 2$ totalizează 70 de puncte;
- în cel puțin 30% dintre teste, în expresia dată apar cel mult trei dintre literele x, y, z sau t .
- Se acordă 10 puncte din oficiu.

Exemple

eq4.in	eq4.out	Explicații
1 +10x-7y+3x-7+5z-8t-z-x-y+3 -1 1 0	-4	Se rezolvă cerința 1: Valoarea expresiei este: $10-7+3-7+5-8-1-1-1+3 = -4$
1 -x+1 -1 1 0	0	Se rezolvă cerința 1: Valoarea expresiei este $-1+1 = 0$
2 +10x-7y+3x-7+5z-8t-z-x-y+3 -1 1 0	8	Se rezolvă cerința 2: Sunt 8 cvarțete: $(-1,-1,0,-1), (0,-1,-1,0), (0,-1,1,1), (0,0,-1,-1), (0,0,1,0), (0,1,1,-1), (1,0,0,1), (1,1,0,0)$ pentru care expresia este egală cu 0.
2 -x+1+0z -1 1 0	27	Se rezolvă cerința 2: Sunt 27 cvarțete: $(1,-1,-1,-1), (1,-1,-1,0), (1,-1,-1,1), (1,-1,0,-1), (1,-1,0,0), (1,-1,0,1)$ etc pentru care expresia este egală cu 0.

Timp maxim de executare/test: **1.5** secunde

Memorie: total **64 MB** din care pentru stivă **32 MB**

Dimensiune maximă a sursei: **30 KB**

2.2.1 Indicații de rezolvare

prof. Stelian Ciurea - Universitatea "Lucian Blaga", Sibiu

Pentru început parsăm expresia dată pentru a calcula coeficienții necunoscuteelor și termenul liber, care apar în *forma canonică* a expresiei (forma care se obține după reducerea termenilor asemenea, de exemplu, din expresia $+5x+4x-x$ rezultă expresia canonică $8x$).

Cerința 1:

Afișăm suma dintre cei patru coeficienți și termenul liber.

Cerința 2:

Fie forma canonică

$$c_1x + c_2y + c_3z + c_4t + t_1$$

pe care am obținut-o prin parsare.

Avem de calculat numărul de soluții întregi ale ecuației

$$c_1x + c_2y + c_3z + c_4t + t_1 = E$$

sau

$$c_3z + c_4t + t_1 = E - c_1x - c_2y$$

Generăm (de exemplu prin două instrucțiuni for imbricate) toate perchile de valori (x, y) cu x și y aparținând intervalului $[a, b]$ și calculăm și reținem într-un tablou unidimensional sau un vector valorile expresiei $E - c_1x - c_2y$. Să notăm acest vector cu V . Apoi sortăm crescător vectorul V .

Apoi avem două posibilități:

a) Procedăm similar pentru a calcula toate valorile expresiei

$$c_3z + c_4t + t_1$$

pentru z și t aparținând intervalului $[a, b]$.

Pentru fiecare valoare a acestei expresii determinăm prin căutare binară numărul de apariții în V ; suma acestor numere reprezintă rezultatul problemei.

b) Determinăm toate valorile expresiei

$$c_3z + c_4t + t_1$$

pentru z și t aparținând intervalului $[a, b]$ și le reținem într-un tablou sau un vector, fie el W .

Sortăm și W , apoi printr-un algoritm similar cu cel de interclasare determinăm numărul de valori egale din cei doi vectori, având în vedere că dacă o valoare oarecare apare de $n1$ ori în V și de $n2$ ori în W , la rezultat vom aduna valoarea $n1 * n2$.

În ambele cazuri, complexitatea algoritmului este $(b - a)^2 * \log(b - a)$

Problema este inspirată de problema eq5 a lui Mihai Pătrașcu propusă de acesta la ONI2002.

2.2.2 *Rezolvare detaliată

2.2.3 Cod sursă

Listing 2.2.1: eq4.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <algorithm>
4 #include <string>
5
6 using namespace std;
7
8 int valori[1100000];
9 string expr;
10 int cx,cy,cz,ct,tl,a,b,E;
11 int nrcrt,semn,ceri;
12
13 ifstream f("eq4.in");
14 ofstream g("eq4.out");
15
16 long long rezultat;
17
18 int main()
19 {
20     cout << "Hello world!" << endl;
21     f >> celi;
22     f >> expr;
23     expr+='+';
24     f >> a >> b >> E;
25     semn = +1;
26     for (int i=0;i<expr.size();i++)
27     {
28         if (expr[i]=='+')
29         {
30             tl = tl + semn*nrcrt;
31             nrcrt = 0;
32             semn = +1;
33         }
34
35         if (expr[i]=='-')
36         {
37             tl = tl + semn*nrcrt;
38             nrcrt = 0;
39             semn = -1;
40         }
41
42         if (expr[i]>='0' && expr[i]<='9')
43             nrcrt = 10*nrcrt+expr[i]-'0';
44
45         if (expr[i]=='x')
46         {
47             if (expr[i-1]=='+')
48                 nrcrt=1;
49             if (expr[i-1]=='-')
50                 nrcrt=1;
51
52             cx = cx + semn*nrcrt;
53             nrcrt = 0;
54         }
55
56         if (expr[i]=='y')
57         {
58             if (expr[i-1]=='+')
59                 nrcrt=1;
60             if (expr[i-1]=='-')
61                 nrcrt=1;
62
63             cy = cy + semn*nrcrt;
64             nrcrt = 0;
65         }

```

```

66         if (expr[i]=='z')
67     {
68         if (expr[i-1]=='+')
69             nrcrt=1;
70         if (expr[i-1]=='-')
71             nrcrt=1;
72         cz = cz + semn*nrcrt;
73         nrcrt = 0;
74     }
75
76     if (expr[i]=='t')
77     {
78         if (expr[i-1]=='+')
79             nrcrt=1;
80         if (expr[i-1]=='-')
81             nrcrt=1;
82         ct = ct + semn*nrcrt;
83         nrcrt = 0;
84     }
85 }
86
87 cout << cx << ' ' << cy << ' ' << cz << ' ' << ct << ' ' << tl << endl;
88 if (ceri==1)
89 {
90     cout << cx+cy+cz+ct+tl<<endl;
91     g << cx+cy+cz+ct+tl<<endl;
92     return 0;
93 }
94
95
96 int poz=0;
97 for (int x=a;x<=b;x++)
98     for (int y=a;y<=b;y++)
99     {
100         int val = E - cx*x -cy*y;
101         valori[poz]=val;
102         // cout << x << ' ' << y << ' ' << val << endl;
103         poz++;
104     }
105
106 cout << endl;
107
108 sort(valori,valori+poz);
109 //for (int i=0;i<poz;i++) cout << valori[i]<< ' ';
110 //cout << endl;
111 for (int z=a;z<=b;z++)
112     for (int t=a;t<=b;t++)
113     {
114         int val = cz*z + ct*t + tl;
115         if (val<valori[0] || val>valori[poz-1])
116             continue;
117         int possup = upper_bound(valori,valori+poz,val)-valori;
118         int pozinf = lower_bound(valori,valori+poz,val)-valori;
119         int nrapt = possup - pozinf;
120         rezultat += nrapt;
121         // cout << z<< ' '<< t<< ' '<< val << ' '<< nrapt << ' '<< rezultat << endl;
122     }
123
124 cout << rezultat << endl;
125 g << rezultat << endl;
126
127 return 0;
128 }

```

2.3 turnuri

Problema 3 - turnuri

100 de puncte

Cel mai nou proiect imobiliar din capitală este compus din N blocuri-turn, construite unul lângă altul, de-a lungul unui bulevard central și numerotate de la 1 la N . Pentru fiecare turn se cunoaște numărul etajelor din care este compus acesta și se mai știe că nu există două turnuri cu același număr de etaje. Ultimele norme urbanistice definesc coeficientul de frumusețe al turnului cu numărul T , ca fiind numărul turnurilor din secvența de turnuri care începe cu turnul S , se

termină cu turnul D și are următoarele proprietăți:

- $1 \leq S \leq T \leq D \leq N$
- numărul etajelor fiecărui turn din secvență, cu excepția turnului T , este mai mic decât numărul de etaje ale turnului T ;
 - Dacă $S \neq 1$ atunci turnul $S - 1$ este cel mai apropiat turn din stânga turnului T , care are un număr de etaje strict mai mare decât turnul T ;
 - Dacă $D \neq N$ atunci turnul $D + 1$ este cel mai apropiat turn din dreapta turnului T , care are un număr de etaje strict mai mare decât turnul T ;

Coefficientul de frumusețe al întregului ansamblu de turnuri este suma coeficienților de frumusețe avuți de turnurile componente.

Dezvoltatorul proiectului dorește să renunțe la unul dintre turnuri și să construiască în locul acestuia un restaurant subteran, acesta considerându-se un turn cu zero etaje. Dezvoltatorul dorește să calculeze coeficientul de frumusețe al ansamblului de turnuri, pentru fiecare posibilă amplasare a restaurantului.

Cerințe

Cunoscând numărul N de turnuri și numărul etajelor fiecărui, determinați coeficientul de frumusețe al ansamblului de turnuri pentru toate cele N posibilități de amplasare ale restaurantului, pe pozițiile 1, 2, ..., N .

Date de intrare

Datele de intrare se citesc din fișierul **turnuri.in**, care are următoarea structură:

- pe prima linie se află numărul natural N , reprezentând numărul de turnuri;
- pe a doua linie se află N valori naturale nenule, separate prin câte un spațiu, reprezentând numărul etajelor turnurilor;

Date de ieșire

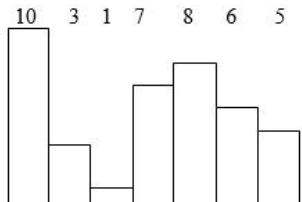
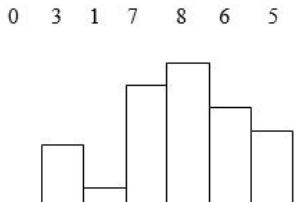
Datele de ieșire se vor scrie în fișierul **turnuri.out**, pe linii separate, astfel: pe linia i ($1 \leq i \leq N$) se găsește un număr natural reprezentând coeficientul de frumusețe al ansamblului dacă restaurantul s-ar construi în locul turnului i .

Restricții și precizări

- $1 \leq N \leq 100000$
- Numărul de etaje ale unui turn este un număr natural între 1 și 1000000000
- Pentru teste în valoare de 30 de puncte, avem $N \leq 100$
- Pentru teste în valoare de încă 30 de puncte, avem $N \leq 2000$
- Se acordă 10 puncte din oficiu.

Exemple

turnuri.in	turnuri.out	Explicații
-------------------	--------------------	-------------------

7 10 3 1 7 8 6 5 22 22 22 21 22 22	<p>Figura 1 este reprezentarea grafică a fișierului de intrare.</p> <p>Dacă restaurantul se construiește în locul turnului 1 (vezi figura 2), avem următorii coeficienți de frumusețe:</p> <ul style="list-style-type: none"> Restaurantul are coeficientul 1 (el însuși) Turnul 2 are coeficientul 3 (secvența compusă din turnurile 1,2 și 3) Turnul 3 are coeficientul 1 (el însuși) Turnul 4 are coeficientul 4 (secvența compusă din turnurile 1,2, 3 și 4) Turnul 5 are coeficientul 7 (secvența compusă din toate turnurile) Turnul 6 are coeficientul 2 (secvența compusă din turnurile 6 și 7) Turnul 7 are coeficientul 1 (el însuși)
	<div style="border: 1px solid black; padding: 10px; text-align: center;"> <p>Coefficientul de frumusețe al ansamblului este:</p> $1 + 3 + 1 + 4 + 7 + 2 + 1 = 19$  <p>Figura 1</p>  <p>Figura 2</p> </div>

Timp maxim de executare/test: 0.5 secunde

Memorie: total 2MB din care pentru stivă 2MB

Dimensiune maximă a sursei: 15KB

2.3.1 Indicații de rezolvare

Adrian BUDĂU, student la Universitatea Bucuresti

Pentru 30 de puncte soluția este foarte simplă. Pentru fiecare turn, se setează înălțimea acestuia la 0, iar apoi se calculează coeficientul de frumusețe al fiecărui turn căutând S și D corespunzător, iterativ (folosind while).

Acest algoritm are complexitate $O(N^3)$.

Pentru 60 de puncte, una din metode este îmbunătățirea algoritmului care calculează S și D corespunzător fiecărui turn, unde se va construi restaurantul.

Aceasta se poate face folosind o stivă în care memorăm indicii turnurilor care încă nu și-au găsit D -ul corespunzător, până la pasul curent.

Este evident că turnurile acestea sunt păstrate în ordine descrescătoare după numărul de etaje:

- Fie două turnuri i, j cu $i < j$. Dacă numărul de etaje ale lui j este mai mare ca cel al lui i , D -ul corespunzător pentru i este j .

Atunci când suntem la turnul i , scoatem din stivă toate turnurile cu mai puține etaje decât turnul i (deoarece D -ul lor corespunzător va fi $i - 1$) și introducem în stivă valoarea i . În acest moment se poate afla și S -ul corespunzător pentru turnul i (este $k + 1$ unde k este turnul din stivă aflat sub i).

Întrucât un element poate fi introdus cel mult o dată și sters din stivă cel mult o dată atunci complexitatea calculării lui S și D este $O(N)$ și, deoarece trebuie fixat locul unde se construiește restaurantul, complexitatea finală devine $O(N^2)$.

O altă metodă de a obține 60 de puncte, este de a calcula frumusețea originală (ca la soluția de 30 de puncte) și apoi, pentru fiecare turn, să se calculeze cu cât s-ar schimba frumusețea dacă acesta ar deveni restaurant.

Dacă restaurantul se va construi în locul turnului i , atunci frumusețea totală va scădea cu frumusețea lui $i - 1$ (deoarece frumusețea restaurantului este 1), dar pentru fiecare turn X , care

îl avea pe i ca $D + 1$, frumusețea va crește. Asemănător și pentru fiecare turn Y care îl avea pe i ca $S - 1$.

Pentru un astfel de turn X dacă notam cu $S2$ turnul cel mai apropiat din stânga astfel încât există cel mult un turn mai înalt de la $S2$ la X , și cu $D2$ turnul cel mai apropiat din dreapta astfel încât există cel mult un turn mai înalt de la X la $D2$.

Dacă în locul lui $D + 1$ corespunzător lui X s-ar construi un restaurant, atunci frumusețea ar crește cu $D2 - D$.

Pentru a afla pe $D2$ și $S2$, se poate proceda ca la soluția de 30 de puncte. Se caută iterativ S și D pentru o poziție X , iar apoi continuând de la aceste poziții se caută tot iterativ $S2$ și $D2$. Complexitatea acestei soluții este tot $O(N^2)$ deși în practică se comportă mai bine decât cea anterioară.

Ea va obține tot 60 de puncte.

Pentru soluția de 90 de puncte se pot folosi ambele soluții de 60 de puncte. Se poate folosi tehnică cu stive de la prima soluție, pentru a afla pe S , D , $S2$ și $D2$, iar apoi cu formulele de la a doua soluție să se determine răspunsul.

S și D se obțin în mod direct, la fel ca la soluția de 60 de puncte, însă $S2$ și $D2$ sunt mai speciale. Pentru ele se poate ține o a doua stivă, în care se mențin elementele din sir pentru care s-a calculat D , dar nu s-a calculat $D2$.

Atunci când se scot elementele din prima stivă, se pun în a doua stivă. Trebuie însă avut grija pentru că elementele fiind scoase din vârful primei stivei, se obțin în ordine crescătoare a numărului de etaje, iar în a doua stivă ar trebui introduse în ordine descrescătoare. Trebuie să se eliminate toate, și apoi introduse toate în a doua stivă în ordinea corectă (nu este necesară o sortare, doar să fie parcurse în ordinea potrivită).

2.3.2 *Rezolvare detaliată

2.3.3 Cod sursă

Listing 2.3.1: turnuri.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <algorithm>
5 #include <stack>
6 #include <cassert>
7
8 using namespace std;
9
10 bool is_unique(vector<int> V)
11 {
12     sort(V.begin(), V.end());
13     for (int i = 1; i < int(V.size()); ++i)
14         if (V[i] == V[i - 1])
15             return false;
16     return true;
17 }
18
19 int main()
20 {
21     ifstream cin("turnuri.in");
22     ofstream cout("turnuri.out");
23
24     int N; assert(cin >> N);
25     assert(1 <= N && N <= 100 * 1000);
26     vector<int> V(N);
27
28     for (int i = 0; i < N; ++i)
29     {
30         assert(cin >> V[i]);
31         assert(1 <= V[i] && V[i] <= 1000 * 1000 * 1000);
32     }

```

```

33     assert(is_unique(V));
34
35     vector<int> left1(N, -1), left2(N, -1), right1(N, N), right2(N, N);
36
37     vector<int> stack1;
38     vector<int> stack2;
39
40     for (int i = 0; i < N; ++i)
41     {
42         int from_erase = stack1.size();
43         while (from_erase > 0 && V[stack1[from_erase - 1]] < V[i])
44             --from_erase;
45
46         // We have to erase from stack1 positions from_erase, from_erase+1, ...
47         // first lets set right2
48
49         while (!stack2.empty() && V[stack2.back()] < V[i])
50         {
51             right2[stack2.back()] = i;
52             stack2.pop_back();
53         }
54
55         // and left2, this could be one of two things
56         // either its the top element of stack2
57         //   - a position P that doesnt have right2 set
58         // or stack1[from_erase - 2]
59         //   - since its also bigger
60         // take whichever is closest
61         if (!stack2.empty())
62             left2[i] = stack2.back();
63         }
64
65         if (from_erase > 1 && stack1[from_erase - 2] > left2[i])
66         {
67             left2[i] = stack1[from_erase - 2];
68         }
69
70         // now we can add the elements
71         // from stack1 positions from_erase, from_erase + 1, ...
72         // because they are waiting for their right2 to be set
73         // all elements in stack2 are bigger than V[i] so
74         // bigger than these elements
75         // its still a nondecreasing sequence
76         for (int j = from_erase; j < int(stack1.size()); ++j)
77         {
78             right1[stack1[j]] = i;
79             stack2.push_back(stack1[j]);
80         }
81
82         stack1.resize(from_erase);
83         if (!stack1.empty())
84             left1[i] = stack1.back();
85
86         stack1.push_back(i);
87     }
88
89     int64_t total = 0;
90     for (int i = 0; i < N; ++i)
91         total += right1[i] - left1[i] - 1;
92
93     vector<int64_t> answer(N, total);
94
95     for (int i = 0; i < N; ++i)
96     {
97         answer[i] -= right1[i] - left1[i] - 2;
98
99         if (right1[i] != N)
100             answer[right1[i]] += right2[i] - right1[i];
101         if (left1[i] != -1)
102             answer[left1[i]] += left1[i] - left2[i];
103     }
104
105     for (int i = 0; i < N; ++i)
106         cout << answer[i] << "\n";
107
108 }
```

Capitolul 3

OJI 2017

3.1 caps

Problema 1 - caps

100 de puncte

Miruna a descoperit un nou joc. Ea dispune de litere mari și mici ale alfabetului englez și construiește succesiv siruri de litere din ce în ce mai lungi. Ea definește operația CAPS a unei litere, ca fiind transformarea literei respective din literă mare în literă mică sau invers, din litera mică în literă mare.

Pentru fiecare sir S , Miruna asociază un nou sir S_C , numit sir CAPS, care se obține aplicând operația CAPS asupra tuturor literelor din sirul S . Miruna a inventat o altă operație pentru un sir de litere S , numită NEXT, prin care obține un nou sir S_N care are structura SS_cS_cS (este format în ordine de la stânga la dreapta din literele lui S , apoi de două ori succesiv literele sirului S_C , iar apoi urmează din nou literele sirului S). De exemplu, sirului $S = \text{"Ham"}$ îi corespunde sirul CAPS $S_C = \text{"hAM"}$ și dacă se aplică și operația NEXT asupra sirului S , obține sirul $S_N = \text{"HamhAMhAMHam"}$.

Initial, Miruna construiește un sir S de K litere. Apoi, ea construiește un nou sir obținut prin aplicarea operației NEXT asupra sirului S . Miruna dorește să obțină succesiv siruri de litere din ce în ce mai lungi aplicând operația NEXT asupra sirului construit în etapa precedentă.

Astfel, pentru $K = 3$ și $S = \text{"Ham"}$, Miruna va construi sirurile

"HamhAMhAMHam",

"HamhAMhAMHamhAMHamHamhAMhAMHamHamhAMhAMHamhAMhAMHam"

și aşa mai departe. Miruna continuă procedeul de construire până când obține un sir final suficient de lung.

Cerinte

Miruna vă roagă să răspundeti la Q întrebări de tipul:

"Dacă se dă un număr natural N , ce literă este în sirul final pe poziția N și de câte ori a apărut această literă în sirul final, de la începutul sirului final până la poziția N inclusiv?".

Date de intrare

Pe prima linie a fișierului **caps.in** se află două numere naturale separate prin spațiu reprezentând valorile K (lungimea sirului inițial) și Q (numărul de interogări). Pe linia următoare se află sirul inițial S de lungime K . Pe următoarele Q linii se va afla câte un număr N , reprezentând cerința unei întrebări.

Date de ieșire

În fișierul de ieșire **caps.out**, se vor afla Q linii, iar pe fiecare linie câte două valori separate cu un spațiu reprezentând răspunsul la o întrebare (litera de pe poziția N în sirul final și numărul său de apariții până la poziția N inclusiv).

Restricții și precizări

- $1 < K \leq 100000$
 - $0 < Q \leq 50000$
 - $0 < N \leq 10^{18}$

• Pentru fiecare test se acordă 40% din punctaj dacă toate literele interogărilor din test sunt corecte și 60% din punctaj dacă toate numerele de apariții ale literelor, până la pozițiile N din interogările testului, sunt corecte.

- Pentru teste în valoare de 15 puncte: $K \leq 250$, $Q \leq 1000$, $N \leq 3000$.
- Pentru alte teste în valoare de 20 de puncte: $N \leq 100000$.
- Pentru alte teste în valoare de 20 de puncte: $K \leq 3000$, $Q \leq 1000$.
- Miruna vă garantează că a construit un sir final de lungime mai mare decât N .
- Prima poziție în sir este considerată poziția 1.

Exemple

caps.in	caps.out	Explicații
3 1 Ham 5	A 1	Pe poziția 5 se va afla litera A, numărul de apariții al ei de la poziția 1 la poziția 5 este 1.

Timp maxim de executare/test: **1.0** secunde

Memorie: total **64 MB** din care pentru stivă **32 MB**

Dimensiune maximă a sursei: **10 KB**

3.1.1 Indicații de rezolvare

prof. Gheorghe Manolache, Colegiul Național de Informatică, Piatra Neamț

Problema construiește un sir după regula precizată.

Se observă că se poate construi sirul cu o operație mai simplă și anume prin adăugarea la sfârșitul sirului precedent a sirului obținut cu operația CAPS. Se poate observa că litera de pe poziția N este dată de paritatea numărului de biți din reprezentarea binara a lui N , dacă vom considera prima poziție zero.

Pentru a calcula numărul de apariții al acestei cifre se observă că pe fiecare secvență de lungime dublă a sirului inițial aven mereu sirul inițial și sirul CAPS. Se precalculează numărul de apariții al fiecarei litere în sirul inițial și în sirul CAPS.

Pentru fiecare interogare, numărul de apariții al literei de pe poziția N se obține calculând câte grupe duble (formate sir și sir CAPS avem până la poziția N). Dacă mai rămâne un rest atunci se determină din ce sir face parte acesta și până la ce poziție se caută litera cu ajutorul valorilor precalculate.

Se pot obține punctaje diferite după complexitatea implementării (descriere Adrian Budău).

1) Pentru 15 puncte (+10 din oficiu), $Q \leq 1.000$ și, $N \leq 3.000$

Se poate pentru fiecare query să se construiască efectiv sirul de caractere, și să afișeze caracterul de pe poziția N și apoi să se numere în $O(N)$ de cate ori apare acel caracter până la poziția N .

2) Pentru cele încă 20 de puncte ($N \leq 100.000$).

Se poate face inițial de la începutul sirului de caractere până la primele 100.000 de caractere și apoi folosind sume parțiale pentru fiecare caracter se poate afla în $O(1)$ pentru fiecare query al cătelea este. Complexitatea este $O(1)$ pe query și $O(N * SIGMA)$ unde $SIGMA$ este mărimea alfabetului (în problema aceasta $52 = 26 * 2$).

3) Pentru celelalte 20 de puncte ($K3000$ și $Q1000$).

Se poate afla cel mai mic număr de forma $X = K * 4^P$ astfel încât $N \leq X$. De aici se poate afla cu o soluție recursiva litera corespunzătoare poziției N .

La fiecare pas considerăm cele 4 intervale $[1, K * 4^P - 1]$, $[K * 4^P - 1 + 1, 2K * 4^P - 1]$, $[2K * 4^P - 1, 3K * 4^P - 1]$ și $[3K * 4^P - 1, K * 4^{P+1}]$. N se poate încadra numai în unul din aceste 4 intervale, și dacă se încadrează în primul sau ultimul putem continua recursiv cu N relativ la acel interval, iar dacă se încadrează în al doilea sau al treilea se tine minte că trebuie inversat răspunsul final (din litera mica în litera mare și viceversa) și se continua recursiv cu N relativ la acel interval.

De exemplu daca $N = 39$ și $K = 7$.

Găsim $X = 7 * 4^2$. Obținem cele 4 intervale $[1, 28]$ $[29, 56]$ $[57, 84]$ $[85, 112]$. 39 se încadrează în $[29, 56]$, se ține minte ca trebuie inversat răspunsul și se continua recursiv cu $N = 39 - 29 + 1 = 11$ și $INVERSIUNI = 1$

Găsim $X = 7 * 4$. Obținem cele 4 intervale $[1, 7]$ $[8, 14]$ $[15, 21]$ $[22, 28]$. 11 se încadrează în $[8, 14]$, se tine minte că trebuie inversat răspunsul și se continua recursiv cu $N = 11 - 8 + 1 = 4$ și $INVERSIUNI = 2$

$X = 4$, deci vrem al 4-lea caracter inversat de 2 ori, deci răspunsul ar fi $S[4]$ unde S este sirul de litere inițial.

Dacă se și precalculează de câte ori apare fiecare caracter în fiecare string de lungime $K * 4^P$ până când $K * 4^P$ depășește 10^8 se poate de fiecare dată când se coboară în recursivitate să se adune la răspuns de câte ori a apărut intervalele de dinainte (în exemplu daca $N = 57$, atunci ar trebui să se adauge la răspuns de cate ori apare caracterul găsit în sirul de mărime $7 * 4$, adică intervalul $[1, 28]$ și de cate ori apare inversul lui într-un sir de mărime $7 * 4$, adică intervalul $[29, 56]$).

La final când $N \leq K$ se poate face în $O(K)$, obținând astfel complexitate $O(K + \log^2 N)$ per query.

4) Combinând ideile de la 2 și 3 putem să reduce complexitatea direct la $O(\log^2 N)$ per query cu $O(K * \sigma)$ precalculare.

Pentru a reduce de la $O(\log^2 N)$ la $O(\log N)$ putem să nu îl căutăm la fiecare pas în recursie pe $X = K * 4^P$ cu NX . După ce l-am găsit pe cel pentru N din fișierul de intrare se verifică mai întâi $X/4$, apoi $X/4^2$ și aşa mai departe.

O altă soluție, mai ușoara de codat se bazează pe următoarea observație:

Dacă transformăm N în 2 valori: PART și INDEX unde PART reprezintă numărul concatenării sirului inițial (cu sau fără inversiune din literă mică în literă mare și viceversa) și INDEX poziția în această concatenare atunci răspunsul este $S[INDEX]$ sau $\text{invers}(S[\text{index}])$ în funcție de PART. Mai exact dacă numărul de biți de 1 din descompunerea lui PART este par atunci răspunsul este $S[INDEX]$ altfel este $\text{invers}(S[\text{index}])$.

Iar pentru a numără de câte ori apare litera răspuns în toate concatenările 1, 2, ..., PART - 1 se poate face următoarea observație:

- Dacă PART - 1 e par atunci oricare 2 concatenări de pe pozițiile $2k + 1$ și $2k + 2$ sunt diferite. Asta înseamnă că jumate din concatenări sunt normale, și jumate sunt inversele lor.

Deci la răspuns s-ar adăuga

$$\begin{aligned} & (\text{PART} - 1) / 2 * \text{APARITII[răspuns]} + \\ & + (\text{PART} - 1)/2 * \text{APARITII[invers(răspuns)]} + \\ & + \text{de câte ori apare în ultima parte.} \end{aligned}$$

- Dacă PART - 1 e impar, atunci PART - 2 e par și se aplică același raționament și se mai adaugă ce e în PART - 1, verificând dacă e invers sau nu (folosind trucul de mai sus, care numără numărul de biți de 1).

Soluția obținuta este deci $O(K * SIGMA)$ precalculare și $O(\log N)$ pe query. Se poate obține $O(1)$ pe query folosind fie funcții non standard (`__builtin_parity`) sau o dinamică pe biți, dar aşa ceva nu era necesar pentru 100 de puncte.

Iar o altă metodă de a calcula de câte ori apare un caracter până la o poziție în sirul original (pe lângă cea cu sumele parțiale pentru fiecare literă) este să se țină pentru fiecare literă pozițiile pe care se află acea literă și să se folosească căutare binară să se afle câte poziții sunt mai mici ca una anume.

Astfel se obține $O(K)$ precalculare și $O(\log N + \log K)$ pe query, dar în practică se comportă la fel ca soluția precedentă.

3.1.2 *Rezolvare detaliată

3.1.3 Cod sursă

Listing 3.1.1: adrian-100.cpp

```

1 #include <iostream>
2 #include <fstream>
```

```

3 #include <vector>
4 #include <algorithm>
5 #include <cassert>
6
7 using namespace std;
8
9 int bits(int64_t N)
10 {
11     int have = 0;
12     while (N > 0)
13     {
14         ++have;
15         N -= (N & -N);
16     }
17     return have;
18 }
19
20 int map(char c)
21 {
22     if (c >= 'A' && c <= 'Z')
23         return c - 'A';
24     return c - 'a' + 26;
25 }
26
27 char rev(char c) {
28     return c ^ 'A' ^ 'a';
29 }
30
31 int main ()
32 {
33     ifstream cin("caps.in");
34     ofstream cout("caps.out");
35
36     int K, Q; assert(cin >> K >> Q);
37     assert(1 <= K && K <= 100 * 1000);
38     assert(1 <= Q && Q <= 50 * 1000);
39
40     string S; cin >> S;
41     assert(int(S.size()) == K);
42     for (auto &c : S)
43         assert((c >= 'a' && c <= 'z') || (c >= 'A' || c <= 'Z'));
44
45     int total_count[52];
46     fill(total_count, total_count + 52, 0);
47     for (auto &c : S)
48         total_count[map(c)]++;
49
50     vector<int> positions[52];
51     for (int i = 0; i < int(S.size()); ++i)
52         positions[map(S[i])].push_back(i);
53
54     for (int i = 0; i < Q; ++i)
55     {
56         int64_t N;
57         assert(cin >> N);
58         assert(1 <= N && N <= 1000LL * 1000 * 1000 * 1000 * 1000 * 1000);
59         --N;
60
61         int64_t part = N / K;
62         int index = N % K;
63         char answer;
64         if (bits(part) % 2)
65             answer = rev(S[index]);
66         else
67             answer = S[index];
68
69         int64_t many = 0;
70         int64_t normal = part / 2, reversed = part / 2;
71         if (part % 2)
72         {
73             if (bits(part - 1) % 2)
74                 ++reversed;
75             else
76                 ++normal;
77         }
78

```

```

79     many += normal * total_count[map(answer)];
80     many += reversed * total_count[map(rev(answer))];
81     char search_for = S[index];
82
83     many += upper_bound(positions[map(search_for)].begin(),
84                           positions[map(search_for)].end(), index) -
85             positions[map(search_for)].begin();
86     cout << answer << " " << many << "\n";
87 }
88 }
```

Listing 3.1.2: manolache-100.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 ifstream in("caps.in");
6 ofstream out("caps.out");
7
8 char s1[100010],s2[100010];
9 unsigned int f1a[27][100010],f1A[27][100010],f2a[27][100010],f2A[27][100010];
10 long long l1,q,n,cat,r,dif,w;
11
12 int main()
13 {
14     in>>l1>>q;in.get();
15     //in.get(s1,101);
16     in>>s1;
17     for(int i=0;i<l1;++i)
18     {
19         if(s1[i]>='a')
20         {
21             s2[i]=s1[i]-32;
22             f1a[s1[i]-'a'][i]=1;
23             f2A[s2[i]-'A'][i]=1;
24         }
25         else
26         {
27             s2[i]=s1[i]+32;
28             f1A[s1[i]-'A'][i]=1;
29             f2a[s2[i]-'a'][i]=1;
30         }
31     }
32     for(int i=0;i<26;++i)
33     {
34         for(int j=1;j<l1;++j)
35         {
36             f1a[i][j]+=f1a[i][j-1];
37             f1A[i][j]+=f1A[i][j-1];
38             f2a[i][j]+=f2a[i][j-1];
39             f2A[i][j]+=f2A[i][j-1];
40         }
41     }
42     long long d=211*l1;
43     char c;
44     for(;q;--q)
45     {
46         in>>n;
47         cat=n/d;//nr bucati duble complete
48         r=n%d;//rest de cautat
49         long long nr=n/l1;//nr bucati mici
50         if(n%l1) nr++; //nr de bucati simple;
51         w=__builtin_parityll(nr-1ll); //numerotare de la 0
52         if(!w)//incepe cu s1
53         {
54             if(r==0ll) //se termina la sf s1
55             {
56                 c=s1[l1-1];
57                 dif=0ll;
58             }
59             else
60             {
61                 if(r<=l1) //se termina pe bucata s1
62                 {
63                     c=s1[r-1];
64                     if(c>='a')
65                         dif=f1a[c-'a'][r-1];
66                     else
67                         dif=f2A[c-'A'][r-1];
68                 }
69             }
70         }
71     }
72 }
```

```

63                     dif=f1A[c-'A'][r-1];
64     }
65     else // de termina pe s1 dar nu e luata si precedenta,
66         // care e evident s2
67     {
68         c=s1[r-11-1];
69         if(c>='a')
70             dif=f2a[c-'a'][11-1]+f1a[c-'a'][r-11-1];
71         else
72             dif=f2A[c-'A'][11-1]+f1A[c-'A'][r-11-1];
73     }
74     else //incepe cu s2
75     if(r==011) //se termina cu s2
76     {
77         c=s2[11-1];
78         dif=011;
79     }
80     else
81         if(r<=11) //se termina pe bucată s2
82     {
83         c=s2[r-1];
84         if(c>='a')
85             dif=f2a[c-'a'][r-1];
86         else
87             dif=f2A[c-'A'][r-1];
88     }
89     else //se termina pe s2 dar e luata si precedenta care e s1
90     {
91         c=s2[r-11-1];
92         if(c>='a')
93             dif=f1a[c-'a'][11-1]+f2a[c-'a'][r-11-1];
94         else
95             dif=f1A[c-'A'][11-1]+f2A[c-'A'][r-11-1];
96     }
97
98     unsigned long long sol=011;
99     if(c>='a')
100         sol=dif+111*cat*(f1a[c-'a'][11-1]+f2a[c-'a'][11-1]);
101     else
102         sol=dif+111*cat*(f1A[c-'A'][11-1]+f2A[c-'A'][11-1]);
103     out<<c<<' '<<sol<<'\n';
104 }
105 out.close();
106 return 0;
107 }
```

Listing 3.1.3: manolache-v1_100.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 //ifstream in("caps.in");
6 ofstream out("caps.out");
7 char s1[100010],s2[100010],cc;
8 int f1a[27][100010],f1A[27][100010],f2a[27][100010],f2A[27][100010];
9 long long l1,q,n,cat,r,dif,w;
10
11 int main()
12 {
13     freopen("caps.in","r",stdin);
14     scanf("%lld %lld",&l1,&q);
15     //cout<<l1<<' '<<q;
16     //in>>l1>>q;in.get();
17     //in.get(s1,101);
18     scanf("%c%s",&cc,s1);
19     //cout<<s1<<' ';
20     //in>>s1;
21     for(int i=0;i<l1;++i)
22         if(s1[i]>='a')
23             {s2[i]=s1[i]-32;f1a[s1[i]-'a'][i]=1;f2A[s2[i]-'A'][i]=1;}
24         else
25             {s2[i]=s1[i]+32;f1A[s1[i]-'A'][i]=1;f2a[s2[i]-'a'][i]=1;}
26     for(int i=0;i<26;++i)
27         for(int j=1;j<l1;++j)
```

```

28     {
29         f1a[i][j] += f1a[i][j-1];
30         f1A[i][j] += f1A[i][j-1];
31         f2a[i][j] += f2a[i][j-1];
32         f2A[i][j] += f2A[i][j-1];
33     }
34     long long d=211*l1;
35     char c;
36     for(;q;--q)
37     {
38         //in>>n;
39         scanf("%lld",&n);
40         cat=n/d;//nr bucati duble complete
41         r=n%d;//rest de cautat
42         long long nr=n/l1;//nr bucati mici
43         if(n%l1) nr++;
44         w=__builtin_parityll(nr-1ll); //numerotare de la 0
45         if(!w)//incepe cu s1
46             if(r==0ll) //se termina la sf s1
47                 {c=s1[l1-1];
48                  dif=0ll;
49                }
50             else
51                 if(r<=l1) //se termina pe bucata s1
52                 {
53                     c=s1[r-1];
54                     if(c>='a')
55                         dif=f1a[c-'a'][r-1];
56                     else
57                         dif=f1A[c-'A'][r-1];
58                 }
59             else // de termina pe s1 dar nu e luata si precedenta,
60             // care e evident s2
61             {
62                 c=s1[r-l1-1];
63                 if(c>='a')
64                     dif=f2a[c-'a'][l1-1]+f1a[c-'a'][r-l1-1];
65                 else
66                     dif=f2A[c-'A'][l1-1]+f1A[c-'A'][r-l1-1];
67             }
68         else //incepe cu s2
69             if(r==0ll) //se termina cu s2
70             {
71                 c=s2[l1-1];
72                 dif=0ll;
73             }
74             else
75                 if(r<=l1) //se termina pe bucata s2
76                 {
77                     c=s2[r-1];
78                     if(c>='a')
79                         dif=f2a[c-'a'][r-1];
80                     else
81                         dif=f2A[c-'A'][r-1];
82                 }
83             else //se termina pe s2 dar e luata si precedenta care e s1
84             {
85                 c=s2[r-l1-1];
86                 if(c>='a')
87                     dif=f1a[c-'a'][l1-1]+f2a[c-'a'][r-l1-1];
88                 else
89                     dif=f1A[c-'A'][l1-1]+f2A[c-'A'][r-l1-1];
90             }
91
92         unsigned long long sol=0ll;
93         if(c>='a')
94             sol=dif+1ll*cat*(f1a[c-'a'][l1-1]+f2a[c-'a'][l1-1]);
95         else
96             sol=dif+1ll*cat*(f1A[c-'A'][l1-1]+f2A[c-'A'][l1-1]);
97         out<<c<<' '<<sol<<'\n';
98     }
99     out.close();
100    return 0;
101 }
```

Listing 3.1.4: manolache-v2_100.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 //ifstream in("caps.in");
6 ofstream out("caps.out");
7 char s1[100010],s2[100010],cc;
8 int f1a[27][100010],f1A[27][100010],f2a[27][100010],f2A[27][100010];
9 long long l1,q,n,cat,r,dif,w;
10
11 int pnrbitti(long long w)
12 {
13     int t=0;
14     while(w)
15     {
16         t+=w%211;
17         w/=211;
18     }
19     return t%2;
20 }
21
22 int main()
23 {
24     freopen("caps.in","r",stdin);
25     scanf("%lld %lld",&l1,&q);
26     scanf("%c%s",&cc,s1);
27     for(int i=0;i<l1;++i)
28         if(s1[i]>='a')
29             {s2[i]=s1[i]-32;f1a[s1[i]-'a'][i]=1;f2A[s2[i]-'A'][i]=1;}
30         else
31             {s2[i]=s1[i]+32;f1A[s1[i]-'A'][i]=1;f2a[s2[i]-'a'][i]=1;}
32     for(int i=0;i<26;++i)
33         for(int j=1;j<l1;++j)
34         {
35             f1a[i][j]+=f1a[i][j-1];
36             f1A[i][j]+=f1A[i][j-1];
37             f2a[i][j]+=f2a[i][j-1];
38             f2A[i][j]+=f2A[i][j-1];
39         }
40     long long d=211*l1;
41     char c;
42     for(;q-->0)
43     {
44         //in>>n;
45         scanf("%lld",&n);
46         cat=n/d;//nr bucati duble complete
47         r=n%d;//rest de cautat
48         long long nr=n/l1;//nr bucati mici
49         if(n%l1) nr++;//nr de bucati simple;
50         w=pnrbitti(nr-l1); //numerotare de la 0
51         if(!w)//incepe cu s1
52             if(r==0) //se termina la sf s1
53                 {c=s1[l1-1];
54                  dif=0;
55                 }
56             else
57                 if(r<=l1) //se termina pe bucată s1
58                 {
59                     c=s1[r-1];
60                     if(c>='a')
61                         dif=f1a[c-'a'][r-1];
62                     else
63                         dif=f1A[c-'A'][r-1];
64                 }
65             else // de termina pe s1 dar nu e luata si precedenta,
66                 // care e evident s2
67             {
68                 c=s1[r-l1-1];
69                 if(c>='a')
70                     dif=f2a[c-'a'][l1-1]+f1a[c-'a'][r-l1-1];
71                 else
72                     dif=f2A[c-'A'][l1-1]+f1A[c-'A'][r-l1-1];
73             }
74         else //incepe cu s2
75             if(r==0) //se termina cu s2

```

```

76         {
77             c=s2[l1-1];
78             dif=0l1;
79         }
80     else
81         if(r<=l1) //se termina pe bucată s2
82         {
83             c=s2[r-1];
84             if(c>='a')
85                 dif=f2a[c-'a'][r-1];
86             else
87                 dif=f2A[c-'A'][r-1];
88         }
89     else // se termina pe s2 dar e luata si precedenta
90         // care e s1
91     {
92         c=s2[r-l1-1];
93         if(c>='a')
94             dif=f1a[c-'a'][l1-1]+f2a[c-'a'][r-l1-1];
95         else
96             dif=f1A[c-'A'][l1-1]+f2A[c-'A'][r-l1-1];
97     }
98
99     unsigned long long sol=0ll;
100    if(c>='a')
101        sol=dif+1ll*cat*(f1a[c-'a'][l1-1]+f2a[c-'a'][l1-1]);
102    else
103        sol=dif+1ll*cat*(f1A[c-'A'][l1-1]+f2A[c-'A'][l1-1]);
104    out<<c<<' '<<sol<<'\n';
105 }
106
107 out.close();
108 return 0;
109 }
```

Listing 3.1.5: MLT-100.cpp

```

1 // prof. Mircea Lupșe-Turpan - Liceul Teoretic Grigore Moisil Timisoara
2 #include <iostream>
3 using namespace std;
4
5 ifstream fin("caps.in");
6 ofstream fout("caps.out");
7
8 const int KMax = 100005;
9 char X[KMax], Y[KMax];
10 int Frecv1[60][KMax], Frecv2[60][KMax];
11 int K, Q;
12 long long N, Nr1, Nr2, Sol;
13
14 void Read()
15 {
16     fin >> K >> Q;
17
18     for(int i = 1; i <= K; ++i)
19         fin >> X[i];
20
21     for(int i = 1; i <= K; ++i)
22         if(X[i] >= 'a' && X[i] <= 'z')
23             Y[i] = X[i] - 32;
24         else
25             Y[i] = X[i] + 32;
26
27     for(int i = 1; i <= K; ++i)
28     {
29         for(int j = 0; j < 60; ++j)
30         {
31             Frecv1[j][i] = Frecv1[j][i-1];
32             Frecv2[j][i] = Frecv2[j][i-1];
33         }
34         Frecv1[X[i] - 'A'][i]++;
35         Frecv2[Y[i] - 'A'][i]++;
36     }
37 }
```

```

39  char DEI(long long N, long long Nr,int Switch)
40  {
41      if(N <= K)
42      {
43          if(Switch == 1)
44          {
45              Sol += Frecv1[X[N] - 'A'][N];
46              return X[N];
47          }
48          else
49          {
50              Sol += Frecv2[Y[N] - 'A'][N];
51              return Y[N];
52          }
53      }
54
55      if(N <= 2 * K)
56      {
57          if(Switch == 1)
58              Nr1++;
59          else
60              Nr2++;
61
62          return DEI(N - K,Nr/4,1-Switch);
63      }
64
65      if(N <= 3 * K)
66      {
67          Nr1++; Nr2++;
68          return DEI(N - 2*K,Nr/4,1-Switch);
69      }
70
71      if(N <= 4 * K)
72      {
73          if(Switch == 1)
74          {
75              Nr1++;
76              Nr2+=2;
77          }
78          else
79          {
80              Nr2++;
81              Nr1+=2;
82          }
83
84          return DEI(N - 3*K,Nr/4,Switch);
85      }
86
87      if(N > Nr * 3/4)
88      {
89          Nr1 += (Nr * 3/8) / K;
90          Nr2 += (Nr * 3/8) / K;
91          return DEI(N - Nr * 3/4,Nr/4,Switch);
92      }
93
94      if(N > Nr * 2/4)
95      {
96          Nr1 += (Nr * 2/8) / K;
97          Nr2 += (Nr * 2/8) / K;
98          return DEI(N - Nr * 2/4,Nr/4,1 - Switch);
99      }
100
101     if(N > Nr * 1/4)
102     {
103         Nr1 += (Nr * 1/8) / K;
104         Nr2 += (Nr * 1/8) / K;
105         return DEI(N - Nr * 1/4,Nr/4,1 - Switch);
106     }
107
108     return DEI(N,Nr/4,Switch);
109 }
110
111 void SolveandPrint()
112 {
113     while(Q--)
114     {

```

```

115     fin >> N;
116     long long Nr = K;
117     while(Nr < N)
118         Nr *=4;
119     Sol = 0; Nr1 = Nr2 = 0;
120     char Letter = DEI(N,Nr,1);
121     Sol += Nr1 * Frecv1[Letter - 'A'][K];
122     Sol += Nr2 * Frecv2[Letter - 'A'][K];
123     fout << Letter << " " << Sol << "\n";
124 }
125 }
126
127 int main()
128 {
129     Read();
130     SolveandPrint();
131     return 0;
132 }
```

3.2 rover

Problema 2 - rover

100 de puncte

NASA plănuiește o nouă misiune Rover pe Marte în anul 2020. Principalul obiectiv al acestei misiuni este de a determina, cu ajutorul unui nou Rover, dacă a existat în trecut viață pe Marte. Până când va fi lansată misiunea, Roverul este supus la tot felul de teste în laboratoarele NASA.

Într-unul din teste, Roverul trebuie să parcurgă o suprafață de forma unui caroaj cu N linii și N coloane. Acesta pornește din zona de coordonate $(1, 1)$ și trebuie să ajungă în zona de coordonate (N, N) , la fiecare pas putându-se deplasa din zona în care se află într-o din zonele învecinate la nord, sud, est sau vest.

Pentru fiecare zonă de coordonate (i, j) se cunoaște $A_{i,j}$, stabilitatea terenului din acea zonă.

Știind că Roverul are o greutate G , o zonă cu stabilitatea terenului cel puțin egală cu G se consideră o zonă sigură pentru deplasarea Roverului, iar o zonă cu stabilitatea terenului mai mică decât G se consideră o zonă periculoasă pentru Rover.

Cerințe

- Determinați numărul minim posibil de zone periculoase pe care le traversează Roverul pentru a ajunge din zona $(1, 1)$ în zona (N, N) .

- Determinați greutatea maximă pe care o poate avea un Rover care să ajungă din zona $(1, 1)$ în zona (N, N) , fără a traversa nicio zonă periculoasă pentru el.

Date de intrare

Pe prima linie a fișierului de intrare **rover.in** se găsește numărul natural V a cărui valoare poate fi doar 1 sau 2. Dacă V este 1, pe a doua linie a fișierului de intrare se găsesc două numere naturale N și G cu semnificația din enunț, iar dacă V este 2, pe a doua linie a fișierului de intrare se află doar numărul N . Pe următoarele N linii se află câte N numere $A_{i,j}$, reprezentând stabilitatea terenului din zona (i, j) .

Date de ieșire

Fișierul de ieșire este **rover.out**.

Dacă valoarea lui V este 1, atunci fișierul de ieșire va conține pe prima linie un număr natural reprezentând numărul minim de zone periculoase pe care trebuie să le traverseze Roverul de greutate G .

Dacă valoarea lui V este 2, atunci fișierul de ieșire va conține pe prima linie un număr natural reprezentând greutatea maximă a unui Rover care poate ajunge din zona $(1, 1)$ în zona (N, N) fără a traversa zone periculoase pentru el.

Restricții și precizări

- Pentru 50% din teste $V = 1$, pentru 50 % din teste $V = 2$.
- $1 \leq N \leq 500$
- $1 \leq G \leq 5000$
- $1 \leq A_{i,j} \leq 10000$
- Zonele de coordonate $(1, 1)$ și (N, N) nu sunt zone periculoase pentru Rover.
- Roverul nu va trece de mai multe ori prin aceeași zonă.

Exemple

rover.in	rover.out	Explicații
1 5 5 5 1 3 4 7 5 2 1 8 5 2 9 5 3 3 4 1 1 1 9 5 1 6 1 8	3	Numărul minim de zone periculoase traversate de la poziția (1,1) până la poziția (5,5) este 3. Un traseu posibil este: 5 1 3 4 7 5 2 1 8 5 2 9 5 3 3 4 1 1 1 9 5 1 6 1 8
2 5 5 1 3 4 7 5 2 1 8 5 2 9 5 3 3 4 1 1 1 9 5 1 6 1 8	2	Greutatea maximă a unui Rover care poate ajunge din zona (1,1) în zona (5,5) fără a trece prin zone periculoase pentru el este 2. Un traseu posibil este: 5 1 3 4 7 5 2 1 8 5 2 9 5 3 3 4 1 1 1 9 5 1 6 1 8

Timp maxim de executare/test: **1.5** secunde

Memorie: total **64 MB** din care pentru stivă **32 MB**

Dimensiune maximă a sursei: **10 KB**

3.2.1 Indicații de rezolvare

Prof. Mircea Lupășe-Turpan, Liceul Teoretic "Grigore Moisil" Timișoara

Cerința 1

Algoritm Lee care folosește un *deque* în loc de o *coadă*.

Se observă că este util să parcurgem prima dată zonele sigure, prin urmare dacă urmează să ne expandăm într-o zonă sigură vom adăuga această zonă în fața cozii, iar dacă urmează să ne expandăm într-o zonă periculoasă vom adăuga această zonă în spatele cozii.

Cerința 2

Căutarea binară a soluției + *Algoritmul Lee*

Pentru o anumită greutate fixată se încearcă găsirea unui drum de la poziția (1,1) la poziția (N,N) trecând doar prin zone de duritate mai mare sau egală cu greutatea fixată inițial.

Dacă există un traseu care să respecte condițiile de mai sus, atunci se reține această greutate ca fiind o soluție posibilă și se încearcă cu o valoare mai mică pentru greutate.

Dacă nu există un traseu care să respecte condițiile de mai sus, atunci se încearcă cu o valoare mai mare pentru greutate.

3.2.2 *Rezolvare detaliată

3.2.3 Cod sursă

Listing 3.2.1: adrian-100.cpp

```

1 // #include <iostream>
2 // #include <fstream>
3 // #include <vector>
4 // #include <algorithm>
5 // #include <queue>
6 // #include <cassert>
7
8 #include<bits/stdc++.h>
9
10 using namespace std;
11
12 static const int dx[4] = {-1, 0, 1, 0};

```

```

13 static const int dy[4] = {0, -1, 0, 1};
14
15 int minimum(const vector< vector<int> > &A, int G)
16 {
17     int N = A.size();
18     vector< vector<int> > dist(N, vector<int>(N, numeric_limits<int>::max()/2));
19
20     queue< pair<int, int> > current, next;
21     current.emplace(0, 0);
22     dist[0][0] = int(A[0][0] < G);
23
24     int current_cost = dist[0][0];
25     while (current.size() || next.size())
26     {
27         if (current.empty())
28         {
29             current.swap(next);
30             ++current_cost;
31             continue;
32         }
33
34         int x, y; tie(x, y) = current.front();
35         current.pop();
36         if (dist[x][y] != current_cost)
37             continue;
38
39         for (int k = 0; k < 4; ++k)
40         {
41             int newx = x + dx[k];
42             int newy = y + dy[k];
43             if (newx >= 0 && newy >= 0 && newx < N && newy < N)
44                 if (dist[newx][newy] > dist[x][y] + int(A[newx][newy] < G))
45                 {
46                     dist[newx][newy] = dist[x][y] + int(A[newx][newy] < G);
47                     if (A[newx][newy] < G)
48                         next.emplace(newx, newy);
49                     else
50                         current.emplace(newx, newy);
51                 }
52         }
53     }
54     return dist[N - 1][N - 1];
55 }
56
57 int main()
58 {
59     ifstream cin("rover.in");
60     ofstream cout("rover.out");
61
62     int V; assert(cin >> V);
63     assert(1 <= V && V <= 2);
64     int N, G; assert(cin >> N);
65     assert(1 <= N && N <= 1000);
66     if (V == 1)
67     {
68         assert(cin >> G);
69         assert(1 <= G && G <= 5000);
70     }
71
72     vector< vector<int> > A(N, vector<int>(N));
73     for (int i = 0; i < N; ++i)
74         for (int j = 0; j < N; ++j)
75         {
76             assert(cin >> A[i][j]);
77             assert(0 <= A[i][j] && A[i][j] <= 10000);
78         }
79
80     if (V == 1)
81     {
82         assert(A[0][0] >= G && A[N - 1][N - 1] >= G);
83         cout << minimum(A, G) << "\n";
84         return 0;
85     }
86
87     int answer;
88     int max_value = 0;

```

```

89     for (auto &line : A)
90         for (auto &v : line)
91             max_value = max(max_value, v);
92     int step;
93     for (step = 1; step < max_value; step <= 1);
94     for (answer = 0; step; step >= 1)
95         if (minimum(A, answer + step) == 0)
96             answer += step;
97     cout << answer << "\n";
98 }

```

Listing 3.2.2: GM_rover.cpp

```

1 #include<bits/stdc++.h>
2
3 using namespace std;
4
5 ifstream f("rover.in");
6 ofstream g("rover.out");
7
8 int a[1001][1001],b[1001][1001],inf=2000000001,t,n,G;
9 int dx[] = {-1,0,1,0}, dy[] = {0,1,0,-1};
10
11 struct nod
12 {
13     int p1,p2;
14 };
15
16 nod aux;
17 deque <nod>Q;
18
19 void p1()
20 {
21     memset(b,-1,sizeof(b));
22     aux.p1=aux.p2=1;
23     Q.push_back(aux);
24     b[1][1]=0;
25     while(!Q.empty())
26     {
27         int X=Q.front().p1, Y=Q.front().p2;
28         Q.pop_front();
29         for(int k=0;k<4;k++)
30         {
31             int x1=X+dx[k],y1=Y+dy[k];
32             if((x1>= 1) && (x1<=n) && (y1>=1) && (y1<= n) && b[x1][y1]==-1)
33             {
34                 if(a[x1][y1]<G)
35                 {
36                     b[x1][y1]=b[X][Y] + 1;
37                     aux.p1=x1,aux.p2=y1;
38                     Q.push_back(aux);
39                 }
40                 else
41                 {
42                     b[x1][y1]=b[X][Y];
43                     aux.p1=x1,aux.p2=y1;
44                     Q.push_front(aux);
45                 }
46             }
47         }
48     }
49     g<<b[n][n]<<"\n";
50 }
51
52 bool Lee(int w)
53 {
54     memset(b,-1,sizeof(b));
55     aux.p1=aux.p2=1;
56     Q.push_back(aux);
57     b[1][1]=0;
58     while(!Q.empty())
59     {
60         int X = Q.front().p1, Y=Q.front().p2;
61         Q.pop_front();
62         for(int k=0;k<4;k++)

```

```

63         {
64             int x1=X+dx[k], y1=Y+dy[k];
65             if((x1>=1) && (x1<=n) && (y1>= 1) && (y1<= n) && b[x1][y1]== -1)
66             {
67                 if(a[x1][y1]>= w)
68                 {
69                     b[x1][y1]=b[X][Y]+1;
70                     aux.p1=x1,aux.p2=y1;
71                     Q.push_back(aux);
72                 }
73             }
74         }
75     }
76     return (b[n][n] != -1);
77 }
78
79 void p2x()
80 {
81     int sol;
82     int mx=0;
83     for (auto &line : a)
84         for (auto &v : line)
85             mx=max(mx, v);
86     int step;
87     for (step=1; step < mx; step<<=1);
88     for (sol =0; step; step >>= 1)
89         if (Lee(sol+step))
90             sol+=step;
91     g<<sol<< "\n";
92 }
93
94 int main()
95 {
96     f>>t>>n; if(t==1) f>>G;
97     for(int i=1;i<=n; ++i)
98         for(int j=1; j<=n; ++j)
99             f>>a[i][j];
100    if(t==1)
101        p1();
102    else
103        p2x();
104    g.close();
105    return 0;
106 }
```

Listing 3.2.3: MLT_Rover.cpp

```

1 // prof. Mircea Lupse-Turpan - Liceul Grigore Moisil Timisoara - 100 puncte
2 #include <fstream>
3 #include <cstring>
4
5 using namespace std;
6
7 ifstream fin("rover.in");
8 ofstream fout("rover.out");
9
10 struct Cell
11 {
12     int x, y;
13     Cell * next;
14 };
15
16 const int NMax = 505;
17 const int oo = 10000;
18
19 int A[NMax][NMax], DP[NMax][NMax];
20 int V, N, G;
21 int dx[] = {-1,0,1,0}, dy[] = {0,1,0,-1};
22
23 Cell * First, * Last;
24
25 void Add_Front(int V1, int V2)
26 {
27     Cell * p = new Cell;
28     p -> x = V1;
```

```

29     p -> y = V2;
30     p -> next = First;
31     if(!First)
32         Last = p;
33     First = p;
34 }
35
36 void Add_Back(int V1, int V2)
37 {
38     Cell * p = new Cell;
39     p -> x = V1;
40     p -> y = V2;
41     p -> next = NULL;
42     if(!First)
43         First = p;
44     else
45         Last -> next = p;
46     Last = p;
47 }
48
49 void Delete_Front()
50 {
51     Cell * p;
52     p = First;
53     First = First -> next;
54     delete p;
55     if(!First) Last = NULL;
56 }
57
58 bool isEmpty()
59 {
60     return (First == NULL);
61 }
62
63 void Read()
64 {
65     fin >> V >> N;
66     if(V == 1) fin >> G;
67     for(int i = 1; i <= N; ++i)
68         for(int j = 1; j <= N; ++j)
69             fin >> A[i][j];
70 }
71
72 inline bool Inside(int x, int y)
73 {
74     return ( (x >= 1) && (x <= N) && (y >= 1) && (y <= N) );
75 }
76
77 void Solve1()
78 {
79     memset(DP,-1,sizeof(DP));
80     Add_Back(1,1);
81     DP[1][1] = 0;
82
83     while(!isEmpty())
84     {
85         int X = First -> x, Y = First -> y;
86         Delete_Front();
87         for(int k = 0; k < 4; k++)
88         {
89             int NewX = X + dx[k], NewY = Y + dy[k];
90             if(Inside(NewX,NewY) && DP[NewX][NewY] == -1)
91             {
92                 if(A[NewX][NewY] < G)
93                 {
94                     DP[NewX][NewY] = DP[X][Y] + 1;
95                     Add_Back(NewX,NewY);
96                 }
97                 else
98                 {
99                     DP[NewX][NewY] = DP[X][Y];
100                    Add_Front(NewX,NewY);
101                }
102            }
103        }
104    }

```

```

105     fout << DP[N][N] << "\n";
106 }
107
108 bool Lee(int Value)
109 {
110     memset(DP,-1,sizeof(DP));
111     Add_Back(1,1);
112     DP[1][1] = 0;
113     while(!isEmpty())
114     {
115         int X = First -> x, Y = First -> y;
116         Delete_Front();
117         for(int k = 0; k < 4; k++)
118         {
119             int NewX = X + dx[k], NewY = Y + dy[k];
120             if(Inside(NewX,NewY) && DP[NewX][NewY] == -1)
121             {
122                 if(A[NewX][NewY] >= Value)
123                 {
124                     DP[NewX][NewY] = DP[X][Y] + 1;
125                     Add_Back(NewX,NewY);
126                 }
127             }
128         }
129     }
130     return (DP[N][N] != -1);
131 }
132
133 void Solve2()
134 {
135     int Left = 1, Right = oo, Sol = -1;
136
137     while(Left <= Right)
138     {
139         int Mid = (Left + Right) / 2;
140         if(Lee(Mid))
141         {
142             Sol = Mid;
143             Left = Mid + 1;
144         }
145         else
146             Right = Mid - 1;
147     }
148     fout << Sol << "\n";
149 }
150
151 int main()
152 {
153     Read();
154     if(V == 1)
155         Solve1();
156     else
157         Solve2();
158     return 0;
159 }
```

Listing 3.2.4: MLT_Rover_STL.cpp

```

1 // prof. Mircea Lupse-Turpan - Liceul Grigore Moisil Timisoara - 100 puncte
2 #include <iostream>
3 #include <deque>
4 #include <cstring>
5
6 using namespace std;
7
8 ifstream fin("rover.in");
9 ofstream fout("rover.out");
10
11 const int NMax = 505;
12 const int oo = 10000;
13
14 int A[NMax][NMax], DP[NMax][NMax];
15 int V, N, G;
16 int dx[] = {-1, 0, 1, 0}, dy[] = {0, 1, 0, -1};
17
```

```

18  deque<pair<int,int>> Q;
19
20 void Read()
21 {
22     fin >> V >> N;
23     if(V == 1) fin >> G;
24     for(int i = 1; i <= N; ++i)
25         for(int j = 1; j <= N; ++j)
26             fin >> A[i][j];
27 }
28
29 inline bool Inside(int x, int y)
30 {
31     return ( (x >= 1) && (x <= N) && (y >= 1) && (y <= N) );
32 }
33
34 void Solve1()
35 {
36     memset(DP,-1,sizeof(DP));
37     Q.push_back(make_pair(1,1));
38     DP[1][1] = 0;
39
40     while(!Q.empty())
41     {
42         int X = Q.front().first, Y = Q.front().second;
43         Q.pop_front();
44         for(int k = 0; k < 4; k++)
45         {
46             int NewX = X + dx[k], NewY = Y + dy[k];
47             if(Inside(NewX,NewY) && DP[NewX][NewY] == -1)
48             {
49                 if(A[NewX][NewY] < G)
50                 {
51                     DP[NewX][NewY] = DP[X][Y] + 1;
52                     Q.push_back(make_pair(NewX,NewY));
53                 }
54                 else
55                 {
56                     DP[NewX][NewY] = DP[X][Y];
57                     Q.push_front(make_pair(NewX,NewY));
58                 }
59             }
60         }
61     }
62     fout << DP[N][N] << "\n";
63 }
64
65 bool Lee(int Value)
66 {
67     memset(DP,-1,sizeof(DP));
68     Q.push_back(make_pair(1,1));
69     DP[1][1] = 0;
70     while(!Q.empty())
71     {
72         int X = Q.front().first, Y = Q.front().second;
73         Q.pop_front();
74         for(int k = 0; k < 4; k++)
75         {
76             int NewX = X + dx[k], NewY = Y + dy[k];
77             if(Inside(NewX,NewY) && DP[NewX][NewY] == -1)
78             {
79                 if(A[NewX][NewY] >= Value)
80                 {
81                     DP[NewX][NewY] = DP[X][Y] + 1;
82                     Q.push_back(make_pair(NewX,NewY));
83                 }
84             }
85         }
86     }
87     return (DP[N][N] != -1);
88 }
89
90 void Solve2()
91 {
92     int Left = 1, Right = oo, Sol = -1;
93 }
```

```

94     while(Left <= Right)
95     {
96         int Mid = (Left + Right) / 2;
97         if(Lee(Mid))
98         {
99             Sol = Mid;
100            Left = Mid + 1;
101        }
102        else
103            Right = Mid - 1;
104    }
105    fout << Sol << "\n";
106 }
107
108 int main()
109 {
110     Read();
111     if(V == 1)
112         Solve1();
113     else
114         Solve2();
115     return 0;
116 }
```

3.3 sir

Problema 3 - sir

100 de puncte

Corneluș a învățat să numere. El pornește întotdeauna de la 1, numără din 1 în 1, nu greșește niciodată numărul următor, însă ezită uneori și atunci spune numărul curent de mai multe ori. Sora lui, Corina, îl urmărește și face tot felul de calcule asupra modurilor în care numără fratele ei. Astfel, ea urmărește până la cât numără (U), câte numere spune în total (N) și, pentru a aprecia cât de ezitant este, numărul maxim de repetări (R) ale unei valori. De exemplu, el poate număra până la 8 astfel: 1 2 3 3 4 5 6 7 7 7 8 8. În acest caz, numără până la 8 ($U = 8$), spune 13 numere ($N = 13$) și ezită cel mai mult la 7, spunându-l de 4 ori ($R = 4$).

Cerințe

- 1) Cunoscând numărul total de numere N și ultimul număr spus U , trebuie să calculați câte siruri diferite au exact N numere și se termină cu numărul U .
- 2) Cunoscând numărul total de numere N și numărul maxim de repetări R ale unei valori, trebuie să calculați câte siruri diferite au exact N numere și fiecare valoare se repetă de cel mult R ori.

Deoarece numărul de siruri poate fi foarte mare, calculați restul împărțirii acestui număr la 20173333.

Date de intrare

Din fișierul **sir.in** se citesc trei numere naturale, P , N și X , scrise în această ordine, cu câte un spațiu între ele. P poate avea una dintre valorile 1 sau 2, iar N este numărul de numere din sir. Când P are valoarea 1, numărul X reprezintă ultimul număr spus (U), iar când P are valoarea 2, X reprezintă numărul maxim de repetări ale unei valori (R).

Date de ieșire

În fișierul **sir.out** se scrie o singură valoare, astfel:

- dacă P a avut valoarea 1, valoarea reprezintă numărul de siruri distincte care au exact N numere și se termină cu numărul X ;
- dacă P a avut valoare 2, valoarea reprezintă numărul de siruri distincte care au exact N numere și fiecare număr se repetă de cel mult X ori.

În ambele cazuri, deoarece numărul rezultat poate fi foarte mare, se va scrie restul împărțirii acestui număr la 20173333.

Restricții și precizări

- $1 \leq N \leq 100000$
- $X \leq N$
- teste cu $P = 1$ vor totaliza 50% din punctaj, restul de 50% din punctaj fiind pentru $P = 2$;
- pentru teste cumulând 50 de puncte valoarea lui N nu depășește 1000;
- Ultima valoare spusă poate să apară de mai multe ori.

Exemple

sir.in	sir.out	Explicații
1 5 3	6	Se rezolvă cerința 1. Pentru $N=5$, $X=3$, sunt 6 siruri care au exact N numere și se termină cu 3: 1 1 1 2 3, 1 1 2 2 3, 1 1 2 3 3, 1 2 2 2 3, 1 2 2 3 3 și 1 2 3 3 3.
2 5 2	8	Se rezolvă cerința 2. Pentru $N=5$, $X=2$, sunt 8 siruri care au exact N numere și fiecare număr se repetă de cel mult 2 ori 1 1 2 2 3, 1 1 2 3 3, 1 1 2 3 4, 1 2 2 3 3, 1 2 2 3 4, 1 2 3 3 4, 1 2 3 4 4, 1 2 3 4 5.
2 10 3	274	Se rezolvă cerința 2. Pentru $N=10$, $X=3$, sunt 274 de siruri care au exact 10 numere și fiecare număr se repetă de cel mult 3 ori.

Timp maxim de executare/test: **0.2** secunde

Memorie: total **16 MB**

Dimensiune maximă a sursei: **5 KB**

3.3.1 Indicații de rezolvare

prof. Rodica Pîntea - Colegiul Național "Grigore Moisil" București

Pentru valorile citite se calculează (modulo 20173333)

Cerință 1: Combinări($N-1, X-1$);

Se poate obține rezultatul prin calcul direct efectuând simplificări pe parcurs sau calculând recurența cu memorizare $C(a,b)=C(a-1,b)+C(a-1,b-1)$. Se poate utiliza și invers modular.

Cerință 2:

Soluția $O(N * X)$: pentru fiecare L de la 1 la N , se calculează optim recurențele:

$$NSOL(u, 1) = \sum_{i=1}^X NSOL(u - 1, i)$$

$$NSOL(u, ap) = NSOL(u, ap - 1) \text{ pentru } 1 < ap \leq X,$$

unde am notat cu $NSOL(L, u, ap)$ numărul sirurilor care au L componente, ultima cifră u și aceasta apare de ap ori.

Soluția $O(N)$: pentru fiecare L de la 1 la $N+1$, se calculează optim recurența:

$NSOL(L) = \sum_{L=1}^X NSOL(L - 1)$ unde am notat cu $NSOL(L)$ numărul sirurilor care au L componente și ultima valoare se repetă o singură dată.

Soluția este dată de $NSOL(N + 1)$.

3.3.2 *Rezolvare detaliată

3.3.3 Cod sursă

Listing 3.3.1: adrian-100.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <cassert>
5
6 using namespace std;
7
8 static const int kModulo = 20173333;
9
10 int main()
11 {

```

```

12     ifstream cin("sir.in");
13     ofstream cout("sir.out");
14
15     int T, N, X; assert(cin >> T >> N >> X);
16     assert(1 <= T && T <= 2);
17     assert(1 <= N && N <= 100 * 1000);
18     assert(1 <= X && X <= N);
19     if (T == 1)
20     {
21         vector<int> prime(N + 1, 0);
22         for (int i = 2; i <= N; ++i)
23             if (prime[i] == 0)
24                 for (int j = i; j <= N; j += i)
25                     prime[j] = i;
26
27         vector<int> exponent(N + 1, 0);
28         for (int i = N - X + 1; i <= N - 1; ++i)
29             for (int j = i; j > 1; j /= prime[j])
30                 ++exponent[prime[j]];
31         for (int i = 1; i < X; ++i)
32             for (int j = i; j > 1; j /= prime[j])
33                 --exponent[prime[j]];
34
35         int answer = 1;
36         for (int i = 1; i <= N; ++i)
37             for (int j = 0; j < exponent[i]; ++j)
38                 answer = (1LL * answer * i) % kModulo;
39         cout << answer << "\n";
40         return 0;
41     }
42
43     vector<int> fibK(N + 2, 0);
44     fibK[1] = 1;
45     fibK[2] = 1;
46     for (int i = 3; i <= N + 1; ++i)
47     {
48         fibK[i] = (2 * fibK[i - 1] - fibK[max(i - X - 1, 0)]) % kModulo;
49         if (fibK[i] < 0)
50             fibK[i] += kModulo;
51     }
52     cout << fibK[N + 1] << "\n";
53 }
```

Listing 3.3.2: GM1.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 ifstream f("sir.in");
6 ofstream g("sir.out");
7
8 int P,N,X;
9 const int MOD = 20173333;
10
11 inline int InversModular(int x)
12 {
13     int put=MOD-2,rez=1;
14     while(put)
15     {
16         if(put%2)
17         {
18             rez=(1LL*rez*x)%MOD;
19         }
20         x=(1LL*x*x)%MOD;
21         put/=2;
22     }
23     return rez;
24 }
25
26 //calc comb(n,k)=n! * (inv k!) * inv (n-k) !
27 inline int fact(int p)
28 {
29     int sol=1;
30     for(int i=2;i<=p;++i)
```

```

31     sol=(111*sol*i)%MOD;
32     return sol;
33 }
34
35 int comb(int n, int k)
36 {
37     int w=(111*fact(n)*InversModular(fact(k)))%MOD;
38     w=(111*w*InversModular(fact(n-k)))%MOD;
39     return w;
40 }
41
42 int p1()
43 {
44     return comb(N-1,X-1);
45 }
46
47 int p2()
48 {
49     vector <int> a;
50     for(int i=0;i<=N;++i) a.push_back(0);
51     int S=1;
52     a[0]=1;
53     for(int i=1; i<=N; ++i)
54     {
55         a[i]=S;
56         if(i>=X)
57             S=((S+a[i]-a[i-X]+MOD))%MOD;
58         else
59             S=(S+a[i])%MOD;
60     }
61     return a[N];
62 }
63
64 int main()
65 {
66     f>>P>>N>>X;
67     if(P==1)
68         g<<p1()<<"\n";
69     else
70         g<<p2()<<"\n";
71     return 0;
72 }
```

Listing 3.3.3: GM2.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 ifstream f("sir.in");
6 ofstream g("sir.out");
7
8 int P,N,X;
9 const int MOD = 20173333;
10 int a[50005];
11
12 void desc(int x)
13 {   int e=0,d=2;
14     while(d*d<=x)
15     {   e=0;
16         while(x%d==0) {x/=d;e++;}
17         a[d]+=e;
18         d++;
19     }
20     a[x]++;
21 }
22
23 void desc1(int x)
24 {   int e=0,d=2;
25     while(d*d<=x)
26     {   e=0;
27         while(x%d==0) {x/=d;e++;}
28         a[d]-=e;
29         d++;
30     }
}
```

```

31     a[x]--;
32 }
33
34 int comb1(int n, int m)
35 {
36     int i;
37     for(i=n;i>=n-m+1;i--)
38         desc(i);
39     for(i=2;i<=m;i++)
40         desc1(i);
41     int s=1;
42     for(i=2;i<=50002;i++)
43         for(int j=1;j<=a[i];j++) s=(1ll*s*i)%MOD;
44     return s;
45 }
46
47 int p1()
48 {
49     return comb1(N-1,X-1);
50 }
51
52 int p2()
53 {
54     vector <int> a;
55     for(int i=0;i<=N;++i) a.push_back(0);
56     int S=1;
57     a[0]=1;
58     for(int i=1; i<=N; ++i)
59     {
60         a[i]=S;
61         if(i>=X)
62             S=((S+a[i]-a[i-X]+MOD))%MOD;
63         else
64             S=(S+a[i])%MOD;
65     }
66     return a[N];
67 }
68
69 int main()
70 {
71     f>>p>>N>>X;
72     if(P==1)
73         g<<p1() << "\n";
74     else
75         g<<p2() << "\n";
76     return 0;
77 }
```

Listing 3.3.4: sir_100.cpp

```

1 #include <fstream>
2
3 #define MOD 20173333
4
5 using namespace std;
6
7 int n, p,x,v[100001];
8
9 ifstream fin("sir.in");
10 ofstream fout("sir.out");
11
12 long long comb(int n, int u)
13 {
14     n--;u--;
15     int i,j;
16     if(u>n/2) u=n-u;
17     v[0]=1;
18     for(i=1;i<=n; i++)
19     {
20         for(j=i/2;j>=0;j--)
21             v[j]=(v[j]+v[j-1])%MOD;
22         v[i/2+1]=v[i/2];
23     }
24     return v[u];
25 }
```

```

26
27 long long rec(int n, int r)
28 {
29     int i;
30     v[1]=1;v[2]=1;
31     for(i=2;i<=n;i++)
32     {
33         v[i+1]=2*v[i]%MOD;
34         if(i>r)
35         {
36             if(v[i+1]>=v[i-r])
37                 v[i+1]-=v[i-r];
38             else
39                 v[i+1]=v[i+1]+MOD-v[i-r];
40         }
41     }
42     return v[n+1];
43 }
44
45 int main()
46 {
47     fin>>p>>n>>x;
48     if(p==1)
49         fout<<comb(n,x)<<endl;
50     else
51         fout<<rec(n,x)<<endl;
52     return 0;
53 }
```

Listing 3.3.5: sirEm.cpp

```

1 // sir-Em O(n)
2 #include <fstream>
3
4 using namespace std;
5
6 #define M 20173333
7
8 int a[131072];
9
10 int inv(long long x)
11 {long long p=1,n=M-2;
12     while (n>0)
13     {    if (n%2==1)
14         {    p=(p*x)%M; n--; }
15     else
16         { x=(x*x)%M; n/=2; }
17     }
18     return p;
19 }
20
21 int main()
22 {
23     ifstream fi("sir.in"); ofstream fo("sir.out");
24     int n,u,i,p;
25     long long r,t;
26     fi>>p;
27     fi>>n>>u;
28     if(p==1)
29     { /*
30         a[0]=1; // triunghiul lui Pascal
31         for(i=1;i<n;i++)
32         { a[i]=1;
33             for(j=i-1;j>=0;j--)
34                 a[j]=(a[j]+a[j-1])%M;
35             }
36             fo<<a[u-1]<<"\n";
37         */
38         for(r=1,i=2;i<n;i++)
39             r=(r*i)%M;
40         for(t=1,i=2;i<u;i++)
41             t=(t*i)%M;
42         r=(r*inv(t))%M;
43         for(t=1,i=2;i<=n-u;i++)
44             t=(t*i)%M;
```

```
45         r=(r*inv(t))%M;
46         fo<<r<<"\n";
47     }
48     else
49     {   a[1]=1;
50     r=(n==u);
51     for(i=2;i<=n;i++)
52     {   if(i>u) t=a[i-1]+M-a[i-u-1];
53     else t=a[i-1];
54     a[i]=(a[i-1]+t)%M;
55     if(i>=n-u+1) r=(r+t)%M;
56     }
57     fo<<r<<"\n";
58   }
59   return 0;
60 }
```

Capitolul 4

OJI 2016

4.1 interesant

Problema 1 - interesant

100 de puncte

Se consideră o mulțime S care conține N siruri de caractere formate din litere mici ale alfabetului englezesc.

Un sir de caractere se numește **interesant** în raport cu celelalte siruri ale mulțimii, dacă nu există un alt sir în mulțime care să-l conțină ca subșir.

De exemplu, dacă mulțimea S conține sirurile **abc**, **bde** și **abcdef**, atunci singurul sir interesant este **abcdef** deoarece **abc** și **bde** nu îl conțin ca subșir. Mai mult, **abc** și **bde** sunt subșiruri în **abcdef**, deci nu sunt interesante.

Cerințe

Fiind dată o mulțime S formată din N siruri de caractere se cere:

1. Să se determine cel mai lung sir. Dacă sunt mai multe siruri având aceeași lungime maximă, se cere cel mai mic din punct de vedere lexicografic.
2. Să se determine toate sirurile interesante din mulțimea S .

Date de intrare

Fișierul de intrare **interesant.in** conține pe prima linie două numere naturale p și N , despărțite prin spațiu. Pentru toate testele de intrare, numărul p poate avea doar valoarea 1 sau valoarea 2. Pe următoarele N linii, se găsesc sirurile de caractere, câte unul pe linie.

Date de ieșire

Dacă valoarea lui p este 1, se va rezolva numai cerința 1.

În acest caz, în fișierul de ieșire **interesant.out** se va scrie cel mai lung sir dintre cele citite. Dacă există mai multe siruri de aceeași lungime, se va scrie cel mai mic din punct de vedere lexicografic.

Dacă valoarea lui p este 2, se va rezolva numai cerința 2.

În acest caz, fișierul de ieșire **interesant.out** va conține pe prima linie o valoare K ce reprezintă numărul de siruri **interesante**, iar pe următoarele K linii, sirurile interesante în ordinea în care apar în fișierul de intrare.

Restricții și precizări

- $2 \leq N \leq 200$
- Lungimea unui sir va fi cuprinsă între 1 și 5000
- Un subșir al sirului de caractere $C_0C_1C_2\dots C_k$ se definește ca fiind o succesiune de caractere $C_{i_1}C_{i_2}C_{i_3}\dots C_{i_k}$, unde $0 \leq i_1 < i_2 < i_3 < \dots < i_k \leq k$.
 - Fișierul de intrare NU conține siruri identice.
 - Pentru rezolvarea corectă a primei cerințe se acordă 20 de puncte, iar pentru cerința a două se acordă 80 de puncte.

Exemplu

interesant.in	interesant.out	Explicații
1 5 abcacaaz ad abcacaad acd zyt	abcacaad	p=1 Fișierul de intrare conține 5 siruri. abcacaad este sirul de lungime maximă. sirul abcacaaz are aceeași lungime, dar este mai mare din punct de vedere lexicografic. Atenție! Pentru acest test se rezolvă doar cerința 1.
2 5 abcacaad ad zayyt acd zyt	2 abcacaad zayyt	p=2 ad, acd sunt subșiruri al lui abcacaad, iar zyt este subșir al lui zayyt Atenție! Pentru acest test se rezolvă doar cerința 2.

Timp maxim de executare/test: **1.5** secunde

Memorie: total **8 MB**

Dimensiune maximă a sursei: **10 KB**

4.1.1 Indicații de rezolvare

Nicu Vlad-Laurențiu, Liceul Teoretic "M. Kogălniceanu", Vaslui

Soluția propusă analizează sirurile pe parcursul citirii din fișier.

Cerința 1 - 20 puncte

Rezolvarea este clasică: determinăm lungimea maximă a unui sir, iar pentru lungimi egale se alege sirul cel mai mic lexicografic

Cerința 2 - 80 puncte

Rezolvarea cerinței presupune:

- a) verificarea unui sir dacă este subșir al altui sir
- b) utilizarea unei stive care reține sirurile "distincte"

În funcție de tipul de verificare ales: căutare secvențială (subșir, caracter), căutare asemănătoare interclasării (parcurgerea paralelă a sirurilor), căutare binară, frecvențe de apariții, precum și a modului de implementare utilizat se obțin punctaje parțiale diferențiate.

4.1.2 *Rezolvare detaliată

4.1.3 Cod sursă

Listing 4.1.1: interesant_en1.cpp

```

1 /*
2     prof. Eugen Nodea
3     Colegiul Național "Tudor Vladimirescu", Tg-Jiu
4 */
5 #include <fstream>
6 #include <cstring>
7
8 using namespace std;
9
10 ifstream f("interesant.in");
11 ofstream g("interesant.out");
12
13 const int Dim = 5005;
14 typedef char CUV[Dim];
15
16 char s[Dim], sol[Dim];
17 int N, Max, nr, Ns;
18 char cuv[201][Dim];
19 bool Erase[201];
20 int Len[201];

```

```

21
22 int main()
23 {
24     int l, i, j, ls, lc, p, L;
25     char *p;
26     bool ok;
27
28     f >> P >> Ns; f.get();
29     if ( P == 1 )
30     {
31         f >> s;
32         while( !f.eof() )
33         {
34             l = strlen(s);
35             if (l > Max)
36             {
37                 strcpy(sol, s);
38                 Max = l;
39             }
40             else
41                 if (l == Max && strcmp(sol, s) > 0 )
42                     strcpy(sol, s);
43             f >> s;
44         }
45         g << sol << "\n";
46     }
47     else
48     {
49         f >> cuv[N++];
50         Len[0] = strlen(cuv[0]);
51         f >> s;
52         while( --Ns )
53         {
54             ls = strlen(s);
55             for(i=0; i < N; ++i)
56             {
57                 if ( !Erase[i] )
58                 {
59                     lc = Len[i];
60                     if (lc >= ls)
61                     {
62                         p = strchr(cuv[i], s[0]);
63                         ok = (p != 0);
64                         for(j=1; j<ls && ok; ++j)
65                         {
66                             L = lc - (p - cuv[i]);
67                             p = strchr(p + 1, s[j]);
68                             ok = (p != 0 && L >= ls - j);
69                         }
70                         if (ok) break;
71                     }
72                 else
73                 {
74                     p = strchr(s ,cuv[i][0]);
75                     ok = (p != 0);
76                     for(j=1; j<lc && ok; ++j)
77                     {
78                         L = ls - (p - s);
79                         p = strchr(p + 1, cuv[i][j]);
80                         ok = (p != 0 && L >= lc - j);
81                     }
82                     if (ok)
83                     {
84                         Erase[i] = 1;
85                         ok = 0;
86                     }
87                 }
88             }
89         }
90         if (!ok)
91         {
92             Len[N] = ls;
93             strcpy(cuv[N++], s);
94         }
95         f >> s;

```

```

97         }
98
99         nr = N;
100        for(i=0; i < N; ++i)
101            if (Erase[i]) --nr;
102            g << nr << "\n";
103            for(i=0; i < N; ++i)
104                if (!Erase[i])
105                    g << cuv[i] << "\n";
106    }
107
108    return 0;
109 }
```

Listing 4.1.2: interesant_en2.cpp

```

1  /*
2   prof. Eugen Nodea
3   Colegiul National "Tudor Vladimirescu", Tg-Jiu
4 */
5 # include <iostream>
6 # include <cstring>
7
8 using namespace std;
9
10 ifstream f("interesant.in");
11 ofstream g("interesant.out");
12
13 const int Dim = 5003;
14 typedef char CUV[Dim];
15
16 char s[Dim], sol[Dim];
17 int N, Max, nr, Ns;
18 char cuv[201][Dim], *p;
19 bool Erase[201];
20 int Len[201];
21
22 int main()
23 {
24     int l, i, j, ls, lc, P, L;
25     bool ok;
26
27     f >> P >> Ns;
28     f.get();
29     if (P == 1)
30     {
31         f >> s;
32         while( !f.eof() )
33         {
34             l = strlen(s);
35             if (l > Max)
36             {
37                 memcpy(sol, s, l);
38                 Max = l;
39             }
40             else
41                 if (l == Max && memcmp(sol, s, l) > 0 )
42                     memcpy(sol, s, l);
43             f >> s;
44         }
45         g << sol << "\n";
46     }
47     else
48     {
49         f >> cuv[N++];
50         Len[0] = strlen(cuv[0]);
51         f >> s;
52         while( --Ns ){
53             ls = strlen(s);
54             for(i=0; i < N; ++i)
55             {
56                 if ( !Erase[i] )
57                 {
58                     lc = Len[i];
59                     if (lc >= ls)
```

```

60
61             {
62                 p = (char*) memchr(cuv[i], s[0], lc);
63                 ok = (p != 0);
64                 j = 1;
65                 while(j<ls && ok)
66                 {
67                     L = lc - (p - cuv[i]);
68                     p = (char*) memchr(p + 1, s[j], L);
69                     ok = (p != 0 && L >= ls - j);
70                     ++j;
71                 }
72                 if (ok) break;
73             }
74             else
75             {
76                 p = (char*) memchr(s ,cuv[i][0], ls);
77                 ok = (p != 0);
78                 j = 1;
79                 while(j<lc && ok)
80                 {
81                     L = ls - (p - s);
82                     p = (char*) memchr(p + 1, cuv[i][j], L);
83                     ok = (p != 0 && L >= lc - j);
84                     ++j;
85                 }
86                 if (ok)
87                 {
88                     Erase[i] = 1;
89                     ok = 0;
90                 }
91             }
92         }
93         if (!ok)
94         {
95             Len[N] = ls;
96             memcpy(cuv[N++], s, ls);
97         }
98         f >> s;
99     }
100
101    nr = N;
102    for(i=0; i < N; ++i)
103        if (Erase[i]) --nr;
104
105    g << nr << "\n";
106    for(i=0; i < N; ++i)
107        if (! Erase[i])
108            g << cuv[i] << "\n";
109    }
110    return 0;
111 }
```

Listing 4.1.3: interesant_en3.cpp

```

1 /*
2      prof. Eugen Nodea
3      Colegiul National "Tudor Vladimirescu", Tg-Jiu
4 */
5 # include <iostream>
6 # include <vector>
7 # include <string>
8
9 using namespace std;
10
11 ifstream f("interesant.in");
12 ofstream g("interesant.out");
13
14 string s, sol;
15 int N, p, Max, nr;
16 vector <string> cuv;
17 bool Erase[201];
18 int L[201];
19
20 int main()
```

```

21  {
22      int l, i, j, k, ls, lc, M;
23      bool ok;
24
25      f >> p >> N; f.get();
26      if ( p == 1 ){
27          for(i=1; i<=N; ++i)
28          {
29              f >> s;
30              l = s.length();
31              if (l > Max)
32              {
33                  sol = s;
34                  Max = l;
35              }
36              else
37                  if (l == Max && sol > s)
38                      sol = s;
39          }
40          g << sol << "\n";
41      }
42      else
43      {
44          f >> s;
45          cuv.push_back(s);
46          L[0] = cuv[0].length();
47          M = N;
48          while( --M )
49          {
50              f >> s;
51              ls = s.length();
52              for(i=0; i<cuv.size(); ++i)
53              {
54                  if ( !Erase[i] )
55                  {
56                      lc = L[i];
57                      if (lc >= ls)
58                      {
59                          k = cuv[i].find_first_of(s[0]);
60                          ok = (k >= 0);
61                          j = 1;
62                          while ( j<ls && ok )
63                          {
64                              k = cuv[i].find_first_of(s[j], k+1);
65                              ok = (k >= 0 && lc - k >= ls - j);
66                              ++j;
67                          }
68                          if (ok)
69                          {
70                              ++nr;
71                              break;
72                          }
73                      }
74                  else
75                  {
76                      k = s.find_first_of(cuv[i][0]);
77                      ok = (k >= 0);
78                      j = 1;
79                      while ( j<lc && ok )
80                      {
81                          k = s.find_first_of(cuv[i][j], k+1);
82                          ok = (k >= 0 && ls - k >= lc - j);
83                          ++j;
84                      }
85                      if (ok)
86                      {
87                          Erase[i] = 1;
88                          ++nr;
89                          ok = 0;
90                      }
91                  }
92              }
93          }
94          if (!ok)
95          {
96              cuv.push_back(s);

```

```

97             i = cuv.size() - 1;
98             L[i] = ls;
99         }
100     }
101     g << N - nr << "\n";
102     for(i=0; i<cuv.size(); ++i)
103         if (!Erase[i])
104             g << cuv[i] << "\n";
105     }
106     return 0;
107 }
```

Listing 4.1.4: interesant_gc1.cpp

```

1 // prof. Constantin Galatan
2
3 #include <iostream>
4 #include <cstring>
5 #include <iostream>
6 #include <algorithm>
7
8 using namespace std;
9
10 ifstream fin("interesant.in");
11 ofstream fout("interesant.out");
12
13 inline bool Substr(int& n, int& m, char* a, char* b);
14
15 int N, p, n, Lmax, Ns, len[201];
16 char s[5003], smax[5003], S[201][5003];
17 bool isSubs[201];
18
19 int main()
20 {
21     fin >> p >> Ns;
22     fin.get();
23
24     if (p == 1)
25     {
26         while (fin.getline(s, 10003))
27         {
28             n = strlen(s);
29             if (n > Lmax || (n == Lmax && strcmp(s, smax) < 0))
30             {
31                 Lmax = n;
32                 strcpy(smax, s);
33             }
34         }
35         fout << smax << '\n';
36         fout.close();
37         exit(0);
38     }
39     bool ok;
40     int k = 0;
41     while (fin.getline(S[N], 10003)) // s - indicele sirului curent
42     {
43         ok = true;
44         len[N] = strlen(S[N]);
45         for (int j = 0; j < N; ++j)
46         {
47             if (isSubs[j]) continue;
48             if (len[N] <= len[j])
49                 if (Substr(len[N], len[j], S[N], S[j]))// S[i] inclus in S[j]
50                 {
51                     ok = false;
52                     break;
53                 }
54             if (Substr(len[j], len[N], S[j], S[N]))
55             {
56                 isSubs[j] = true;
57                 k++;
58             }
59         }
60         if (ok) N++;
61     }
}
```

```

62     fout << N - k << '\n';
63     for (int i = 0; i < N; ++i)
64         if ( !isSubs[i] )
65             fout << S[i] << '\n';
66
67     fin.close();
68     fout.close();
69
70     return 0;
71 }
72
73
74 inline bool Substr(int& n, int& m, char* a, char* b) // caut a in b
75 {
76     char *p = strchr(b, a[0]);
77     if ( !p ) return false;
78     for (int i = 1; i < n; ++i)
79     {
80         p = strchr(p + 1, a[i]);
81         if ( !p ) return false;
82         if ( m - (p - b) < n - i )
83             return false;
84     }
85     return true;
86 }
```

Listing 4.1.5: interesant_gc2.cpp

```

1 // prof. Constantin Galatan
2 #include <iostream>
3 #include <cstring>
4 #include <iomanip>
5 #include <algorithm>
6
7 using namespace std;
8
9 ifstream fin("interesant.in");
10 ofstream fout("interesant.out");
11
12 inline bool Substr(string& a, string& b);
13
14 int n, k, len[201];
15 int N, p, Lmax, Ns;
16 string s, smax, S[201];
17 bool isSubs[201];
18
19 int main()
20 {
21     fin >> p >> Ns;
22     fin.get();
23
24     if ( p == 1 )
25     {
26         while ( getline(fin, s) )
27         {
28             n = s.size();
29             if ( n > Lmax || (n == Lmax && s < smax) )
30             {
31                 Lmax = n;
32                 smax = s;
33             }
34         }
35         fout << smax << '\n';
36         fout.close();
37         exit(0);
38     }
39     bool ok;
40     while (getline(fin, S[N])) // s - indicele sirului curent
41     {
42         ok = true;
43         for (int j = 0; j < N; ++j)
44         {
45             if ( isSubs[j] ) continue;
46             if ( S[N].size() <= S[j].size() )
47                 if ( Substr(S[N], S[j]) ) // S[i] inclus in S[j]
```

```

48             {
49                 ok = false;
50                 break;
51             }
52             if ( Substr(S[j], S[N]) )
53             {
54                 k++;
55                 isSubs[j] = true;
56             }
57         }
58         if ( ok ) N++;
59     }
60
61     fout << N - k << '\n';
62     for (int i = 0; i < N; ++i)
63     {
64         if ( !isSubs[i] )
65             fout << S[i] << '\n';
66     }
67     fin.close();
68     fout.close();
69
70     return 0;
71 }
72 inline bool Substr(string& a, string& b) // caut a in b
73 {
74     int p = -1;
75     int n = a.size(), m = b.size();
76     for (int i = 0; i < n; ++i)
77     {
78         p = b.find(a[i], p + 1);
79         if ( p == string::npos )
80             return false;
81         if ( n - i > m - p )
82             return false;
83     }
84     return true;
85 }
```

Listing 4.1.6: interesant_gc3.cpp

```

1 // prof. Constantin Galatan
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #include <vector>
6
7 using namespace std;
8
9 ifstream fin("interesant.in");
10 ofstream fout("interesant.out");
11
12 inline bool Substr(int& n, int& m, char* a, char* b);
13 inline int Bs(const vector<int>& v, int pz);
14
15 int N, p, Lmax, Ns, n, k;
16 char s[5003], smax[5003], S[201][5003];
17 int len[201];
18 bool isSubs[201];
19 vector<int> P[200]['z' - '0' + 1];
20
21 int main()
22 {
23     fin >> p >> Ns;
24
25     fin.get();
26
27     if ( p == 1 )
28     {
29         while ( fin.getline(s, 10003) )
30         {
31             n = strlen(s);
32             if ( n > Lmax || (n == Lmax && strcmp(s, smax) < 0) )
33             {
34                 Lmax = n;
```

```

35             strcpy(smax, s);
36         }
37     }
38     fout << smax << '\n';
39     fout.close();
40     exit(0);
41 }
42 bool ok;
43
44
45 while (fin.getline(S[N], 5003))
46 {
47     ok = true;
48     len[N] = strlen(S[N]);
49     for (int i = 'a'; i <= 'z'; ++i)
50     {
51         P[N][i - 'a'].clear();
52         P[N][i - 'a'].shrink_to_fit();
53     }
54     for (char* c = S[N]; *c; ++c)
55         P[N][*c - 'a'].push_back(c - S[N]);
56
57     for (int j = 0; j < N; ++j)
58     {
59         if (isSubs[j]) continue;
60
61         if (len[N] <= len[j])
62         {
63             if (Substr(N, j, S[N], S[j])) // S[N] subsir in S[j]
64             {
65                 ok = false;
66                 break;
67             }
68         }
69         else
70         {
71             if (Substr(j, N, S[j], S[N]))
72             {
73                 isSubs[j] = true;
74                 k++;
75             }
76         }
77     }
78     if (ok) N++;
79 }
80
81 fout << N - k << '\n';
82 for (int i = 0; i < N; ++i)
83     if (!isSubs[i])
84         fout << S[i] << '\n';
85
86 fin.close();
87 fout.close();
88
89 return 0;
90 }
91
92 inline int Bs(const vector<int>& v, int val)
93 {
94     int st = 0, dr = v.size() - 1, m, next_pos = -1;
95
96     while (st <= dr)
97     {
98         m = (st + dr) / 2;
99         if (v[m] > val)
100         {
101             dr = m - 1;
102             next_pos = v[m];
103         }
104         else
105             st = m + 1;
106     }
107     return next_pos;
108 }
109
110 inline bool Substr(int& i, int& j, char* a, char* b) // caut a in b

```

```

111 {
112     int n = len[i], m = len[j];
113     int pz(-1);
114     for (int i = 0; i < n; ++i)
115     {
116         pz = Bs(P[j][a[i] - 'a'], pz); // caut in P[litera] prima valoare
117                                     // mai mare decat pz
118         if (pz == -1) // n-am gasit
119             return false;
120         if (m - pz < n - i)
121             return false;
122     }
123
124     return true;
125 }
```

Listing 4.1.7: interesant_mot.cpp

```

1 //prof. Nistor Mot  O(n*n*L)
2 #include <iostream>
3
4 using namespace std;
5
6 string s[201];
7 int distinct[201];
8
9 int subsir(const string x, const string y)
10 {
11     int i,j, lx=x.length(), ly=y.length();
12     if(lx<=ly) return 0;
13     for(i=0, j=0; i<lx && j<ly; i++)
14         if(x[i]==y[j]) j++;
15     return j==ly;
16 }
17
18 int main()
19 { ifstream fi("interesant.in"); ofstream fo("interesant.out");
20     int n,p,i,j,ns=0,mx=0,ix=0,ls;
21     fi>>p>>n;
22     for(i=1;i<=n;i++)
23         fi>>s[i];
24     if(p==1)
25     { for(i=1;i<=n;i++)
26         { ls=(int)s[i].length();
27             if(ls>mx || ls==mx && s[i]<s[ix] )
28                 { mx=ls; ix=i; } }
29         fo<<s[ix];
30     }
31     else
32     { for(i=1;i<=n;i++)
33         for(j=1;j<=n;j++)
34             if(j!=i && subsir(s[j],s[i]))
35                 { ns++; distinct[i]=1; break; }
36     fo<<n-ns<<"\n";
37     for(i=1;i<=n;i++)
38         if(!distinct[i])
39             fo<<s[i]<<"\n"; }
40     return 0;
}
```

Listing 4.1.8: interesant_nv.cpp

```

1 // prof. Nicu Vlad
2 #include <algorithm>
3 #include <iostream>
4 #include <cstring>
5
6 #define N 205
7 #define M 10010
8
9 using namespace std;
10
11 ifstream cin("interesant.in");
12 ofstream cout("interesant.out");
13
```

```

14 char A[N][M], AX[M],AY[M];
15 bool viz[N];
16
17 bool verif(char A[], char B[])
18 {
19     int x=strlen(A);
20     int y=strlen(B);
21     int i=0, j=0;
22     bool ok=false;
23     while(i<x&&j<y)
24     {
25         if(A[i]!=B[j]);
26         else j++;
27         i++;
28         if(i==x && j<y) return false;
29     }
30     if(j==y) return true;
31     return false;
32 }
33
34 int main()
35 {
36     int cer=0,n;
37     cin>>cer>>n;
38     if(cer==1)
39     {
40         int i=0,maxi=0;
41         for(i=0; i<n; i++)
42         {
43             cin>>AX;
44             int x=strlen(AX);
45             if(x>maxi)
46             {
47                 maxi=x;
48                 strcpy(AY,AX);
49             }
50             else
51                 if(x==maxi)
52                     if(strcmp(AY,AX)>0)
53                         strcpy(AY,AX);
54         }
55         cout<<AY;
56     }
57     else
58     {
59         for(int i=0; i<n; i++)
60         {
61             cin>>A[i];
62             int l=strlen(A[i]);
63             for(int j=0; j<i; j++)
64             {
65                 if(!viz[j])
66                 {
67                     int k=strlen(A[j]);
68                     if(l>=k)
69                         if(verif(A[i],A[j])) viz[j]=1;
70                     else
71                         if(verif(A[j], A[i])) viz[i]=1;
72                 }
73             }
74         }
75         int nr=n;
76         for(int j=0; j<n; j++)
77             if(viz[j]) nr--;
78         cout<<nr<<"\n";
79         for(int j=0; j<n; j++)
80             if(!viz[j]) cout<<A[j]<<"\n";
81     }
82     return 0;
83 }

```

4.2 miting

Problema 2 - miting

100 de puncte

În *Orașul Liniștit* un număr de k tineri prieteni doresc să participe la un miting de protest. Deoarece cartierul în care locuiesc aceştia este mare, ei se vor deplasa spre punctul de întâlnire cu maşinile personale. Fiecare Tânăr va aduce cu el o pancartă, pe care a desenat o singură literă din mulțimea $A \dots Z$. Nu există două pancarte cu litere identice. Cele k litere formează un cuvânt, să-l notăm *cuv*, cunoscut.

Cartierul în care locuiesc tinerii poate fi codificat printr-o matrice cu $n \times m$ zone pătratice, dintre care unele sunt interzise. Se știe că o mașină consumă o unitate de combustibil la trecerea dintr-o zonă în zona vecină și nu consumă combustibil dacă staționează. Două zone sunt vecine dacă au în comun o latură. Pentru a face economie de combustibil, tinerii decid că dacă două mașini se întâlnesc într-o zonă și toate literele aflate în cele două mașini reprezintă o secvență din cuvântul *cuv*, atunci ei vor continua drumul cu o singură mașină, luând desigur toate pancartele cu ei. În caz contrar, mașinile își continuă drumul separat.

De exemplu, dacă cuvântul *cuv* este "JOS", atunci mașina care transportă litera *J* poate prelua Tânărul care aduce pancarta cu litera *O*, sau invers: mașina având litera *O* poate prelua Tânărul care aduce litera *J*. Apoi se poate continua drumul spre mașina care transportă litera *S*. În altă variantă se pot reuni mai întâi literele *S* și *O* într-o singură mașină, dacă mașinile care le transportau se întâlnesc în aceeași zonă. Totuși, între mașina care transportă doar litera *J* și cea care transportă doar litera *S* nu se poate realiza un transfer, adică o reunire a literelor.

Cerințe

Cunoscând dimensiunile cartierului n și m , cuvântul *cuv*, configurația cartierului și pozițiile inițiale ale tinerilor, se cere:

1. Aria minimă a unei submatrice a matricei care codifică cartierul, în care se situează toate pozițiile inițiale ale tinerilor.
2. Numărul minim de unități de combustibil consumați de către toate mașinile, știind că în final toți tinerii se vor reuni într-o singură mașină.

Date de intrare

Fișierul de intrare **miting.in** conține:

Pe prima linie, un număr natural p , care poate avea doar valoarea 1 sau 2.
Pe a doua linie două numere naturale n și m , separate printr-un spațiu.
Pe a treia linie, cuvântul *cuv*.

Pe următoarele n linii, câte m caractere pe linie reprezentând zonele cartierului. O zonă este interzisă dacă îi corespunde caracterul $\#$, este liberă dacă îi corespunde caracterul $_$ (underline) și este punctul de plecare al unei mașini dacă îi corespunde una dintre literele cuvântului *cuv*.

Date de ieșire

Dacă valoarea lui p este 1, se va rezolva numai cerința 1.

În acest caz, în fișierul de ieșire **miting.out** se va scrie un singur număr natural A , reprezentând aria minimă a unei submatrice a matricei care codifică cartierul, în care se situează toate pozițiile inițiale ale tinerilor.

Dacă valoarea lui p este 2, se va rezolva numai cerința 2.

În acest caz, în fișierul de ieșire **miting.out** se va scrie un singur număr natural C , reprezentând numărul minim de unități de combustibil consumate de către toate mașinile până la reunirea tinerilor, deci și a literelor, într-o singură mașină. În cazul în care nu există soluție, adică nu toți tinerii se pot reuni într-o singură mașină, se va scrie -1 .

Restricții și precizări

- $2 \leq n, m \leq 60$
- $2 \leq k \leq 10$
- Fie z numărul zonelor interzise. Atunci $0 \leq z \leq (n * m)/3$
- În fiecare unitate de timp, o mașină poate să rămână pe loc în aşteptarea alteia sau poate să treacă într-o zonă vecină, indiferent dacă zona respectivă este sau nu ocupată de o altă mașină.
 - Lungimea laturii unei zone se consideră egală cu 1.
 - Pentru rezolvarea corectă a primei cerințe se acordă 20 de puncte, iar pentru cerința a două se acordă 80 de puncte.
 - Pentru 30% dintre teste cerinței 2 se garantează $k \leq 3$.

Exemple

miting.in	miting.out	Explicație
1 4 5 JOS #_O_# #_S _#J_# __#_	9	Submatricea de arie minimă care include toate literele are colțul stânga sus la linia 1 și coloana 3 și colțul dreapta jos la linia 3 și coloana 5. Aria este egală cu numărul de zone acoperite: $3 \times 3 = 9$. Atenție! Pentru acest test se rezolvă doar cerința 1.

Figura 4.1: Miting

miting.in	miting.out	Explicație
2 5 7 BUN #_#_# N#_# #_B U_#_# #_#_#	6	O variantă de consum minim este: U se deplasează cu două poziții la dreapta. Apoi B se deplasează cu două poziții la stânga. U se deplasează din nou cu o singură poziție în sus. În final, N coboară o poziție. Remarcați că B s-a reunit cu U, apoi BU cu N. Atenție! Pentru acest test se rezolvă doar cerința 2.

Figura 4.2: Miting

miting.in	miting.out	Explicație
2 6 7 ROST O#_#_# #_# #_R # #_S_# #_T_#	9	O variantă de consum minim este: O se deplasează cu o poziție în jos, apoi cu două poziții spre dreapta, coboară o poziție și în final se deplasează o poziție spre dreapta, unde se reuneste cu R. Apoi S se deplasează cu o poziție la stânga. T urcă o poziție și se reuneste cu S. În final, mașina în care se găsesc S și T urcă două poziții și se întâlnesc cu mașina în care se găsesc R și O. În această zonă, la linia 3 și coloana 4, toate literele se reunesc într-o singură mașină. Atenție! Pentru acest test se rezolvă doar cerința 2.

Figura 4.3: Miting

Timp maxim de executare/test: 1.0 secunde

Memorie: total 16 MB

Dimensiune maximă a sursei: 10 KB

4.2.1 Indicații de rezolvare

prof. Constantin Gălățan - C. N. "Liviu Rebreanu" Bistrița

Cerința 1. (20 puncte)

Se mențin patru variabile: $imin, jmin, imax, jmax$ reprezentând coordonatele colțului stânga sus, respectiv colțul dreapta jos a dreptunghiului care include toate literele cuvântului cuv. Aceste variabile se actualizează odată cu citirea matricei de caractere. Aria maximă va fi:

$$A = (imax - imin + 1) * (jmax - jmin + 1)$$

Cerință 2. (80 de puncte)

Problema impune restricția ca două litere aflate în mașini diferite sau două secvențe de litere aflate în două mașini diferite să se poată reuni numai dacă toate literele aflate în cele două mașini se pot la rândul lor rearanja ca o secvență a cuvântului.

Numerotăm zonele accesibile ale matricei astfel încât numerele de ordine $1 \dots nc$ să corespundă zonelor corespunzătoare pozițiilor inițiale ale literelor cuvântului, respectând ordinea din cuvânt. Pentru restul zonelor accesibile ordinea de numerotare nu este importantă.

Să presupunem că dorim să reunim în aceeași mașină, literele $cuv[i \dots j]$, unde i și j sunt poziții în cuvânt. Costul reunirii depinde și de zona x în care dorim să le aducem. Definim $c[i][j][x]$ consumul minim pentru a unifica $cuv[i \dots j]$ în zona cu numărul de ordine x . Fie nc numărul de caractere ale cuvântului. Secvența $cuv[i \dots j]$ obținută într-o mașină în zona x provine prin reunirea literelor provenind din două mașini: prima având literele $cuv[i \dots k]$, iar a doua literele $cuv[k+1 \dots j]$. Consumul minim de combustibil $c[i][j][x]$ necesar pentru ca două mașini să ajungă în mod independent în zona x (Fig 1) este:

$$c[i][j][x] = \min(c[i][k][x], c[k+1][j][x]), 1 \leq i < nc, i < k < j, nc < x \leq n * n$$

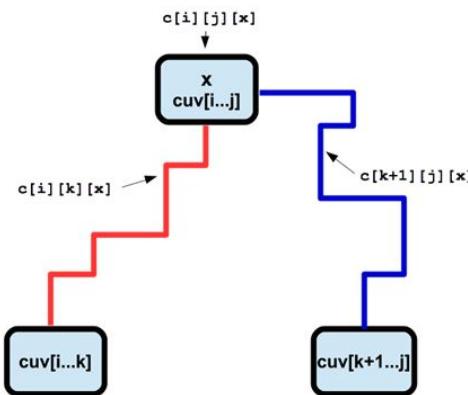


Fig. 1
Figura 4.4: mitingIR1

însă drumul minim al celor două mașini până în zona x poate avea o porțiune comună începând cu zona y . În acest caz literele se reunifică în zona y . Una dintre cele două mașini se oprește în y , iar consumul transportului până în x scade corespunzător consumului unei mașini pe drumul comun (Fig 2).

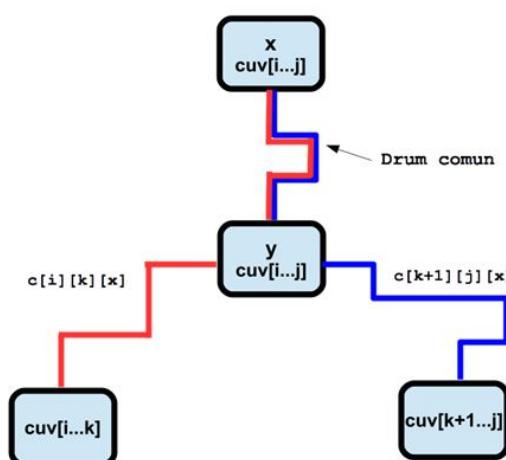


Fig. 2
Figura 4.5: mitingIR2

Valoarea $c[i][j][x]$ se poate ajusta (eventual scade) rulând un *algoritm de cost minim (Lee)*. Dacă y e zonă vecină cu x , atunci:

$$c[i][j][x] = \min(c[i][j][x], c[i][j][y] + 1)$$

În rezumat: mai întâi se determină consumurile minime pentru transportul fiecarei litere a cuvântului în fiecare celulă x a matricei cu un *algoritm de tip Lee*. Pe baza acestor rezultate, se determină consumurile minime pentru a reuni câte două litere consecutive ale cuvântului în fiecare celulă x a matricei. Urmează determinarea consumurilor minime pentru reunirea a trei litere consecutive folosind valori calculate pentru secvențe mai scurte, și aşa mai departe. În final, se determină consumurile minime pentru a reuni toate literele cuvântului în fiecare celulă x . Răspunsul este minimul acestor ultime valori.

Generarea tuturor secvențelor de lungimi 1, 2, ..., nc necesită $nc * (nc + 1)/2$ operații. Pentru fiecare asemenea operație, se aplică un *algoritm de tip Lee*, de complexitate $O(n^3)$. Complexitatea finală este $O(nc^2 * n^3)$.

4.2.2 *Rezolvare detaliată

4.2.3 Cod sursă

Listing 4.2.1: miting.cpp

```

1  /*  prof. Constantin Galatan
2   Solutie O(nc^2 * n^3)
3  */
4
5 #include <fstream>
6 #include <iostream>
7 #include <algorithm>
8 #include <vector>
9 #include <queue>
10
11 using namespace std;
12
13 ifstream fin("miting.in");
14 ofstream fout("miting.out");
15
16 const int MaxN = 61, MaxC = 10, MaxV = 3601, Inf = 1 << 13,
17     di[] = { -1, 0, 1, 0}, dj[] = { 0, 1, 0, -1};
18 using Pair = pair<short, short>;
19
20 short c[MaxC][MaxC][MaxV];
21 int M, N, n, nc, cel[MaxN][MaxN]; // cel[i][j] = numarul de ordine
22 // coresponzator celulei (i, j)
23 string cuv;
24 deque<Pair> D;
25 queue<short> Q;
26 char H[MaxN][MaxN], ch;
27
28 inline void Lee(short i, short j);
29 bool Ok(short i, short j);
30
31 int main()
32 {
33     short imin = MaxN, jmin = MaxN, imax = -1, jmax = -1, p;
34
35     fin >> p >> N >> M >> cuv;
36     fin.get();
37     for (short i = 0; i < N; ++i)
38     {
39         fin.getline(H[i], 101);
40         for (short j = 0; j < M; ++j)
41         {
42             ch = H[i][j];
43             if (isupper(ch) )
44             {
45                 D.push_front({i, j});
46                 imin = min(imin, i); jmin = min(jmin, j);
47                 imax = max(imax, i); jmax = max(jmax, j);
48             }
49             else

```

```

50             if ( ch != '#' )
51                 D.push_back({i, j});
52         }
53     }
54     fin.close();
55
56     if ( p == 1 )
57     {
58         fout << (imax - imin + 1) * (jmax - jmin + 1) << '\n';
59         fout.close();
60         exit(0);
61     }
62
63     nc = cuv.size();
64     vector<int> pos('z' + 1); // pozitiile caracterelor in cuvant
65     for ( int i = 0; i < nc; ++i )
66         pos[cuv[i]] = i;
67
68     // Dupa sortare primele nc celule vor contine literele in ordinea din cuvant
69     sort(D.begin(), D.begin() + nc, [&pos](const Pair& p1, const Pair& p2)
70         { return pos[H[p1.first][p1.second]] < pos[H[p2.first][p2.second]]; })
71     );
72
73     n = D.size();
74     for ( int k = 0, i, j; k < n; ++k )
75     {
76         i = D[k].first, j = D[k].second;
77         cel[i][j] = k;
78     }
79
80     for ( int i = 0; i < nc; ++i )
81     {
82         for ( int x = 0; x < n; ++x )
83             for ( int j = 0; j < nc; ++j )
84                 c[i][j][x] = Inf;
85
86         c[i][i][i] = 0;
87         Q.push(i);
88         Lee(i, i);
89     }
90
91     for ( int l = 2; l <= nc; ++l )
92         for ( int i = 0; i < nc - l + 1; ++i )
93         {
94             int j = i + l - 1, cmin;
95             for ( int x = 0; x < n; ++x )
96             {
97                 cmin = c[i][j][x];
98                 for ( int k = i; k < j; ++k )
99                     cmin = min(cmin, c[i][k][x] + c[k + 1][j][x]);
100
101                 if ( cmin < c[i][j][x] )
102                     c[i][j][x] = cmin,
103                     Q.push(x);
104             }
105             if ( l < nc ) Lee(i, j);
106         }
107
108     short res = Inf;
109     for ( short x = 0; x < n; ++x )
110         res = min(res, c[0][nc - 1][x]);
111
112     if ( res != Inf )
113         fout << res << '\n';
114     else
115         fout << -1 << '\n';
116
117     fout.close();
118 }
119
120 inline void Lee(short i, short j)
121 {
122     short I, J, iv, jv, x, y;
123     while ( !Q.empty() )
124     {
125         x = Q.front(); Q.pop();

```

```

126         I = D[x].first; J = D[x].second;
127         for (int dir = 0; dir < 4; ++dir)
128         {
129             iv = I + di[dir];
130             jv = J + dj[dir];
131             y = cel[iv][jv];
132             if (Ok(iv, jv) && c[i][j][y] > c[i][j][x] + 1)
133             {
134                 c[i][j][y] = c[i][j][x] + 1;
135                 Q.push(y);
136             }
137         }
138     }
139 }
140 }
141
142 inline bool Ok(short i, short j)
143 {
144     if (i < 0 || j < 0 || i >= N || j >= M) return false;
145     if (H[i][j] == '#') return false;
146     return true;
147 }
```

Listing 4.2.2: miting_en.cpp

```

1 // prof. Eugen Nodea
2 # include <fstream>
3 # include <iostream>
4 # include <cstring>
5
6 # define INF 9999
7 # define NR 105
8 # define N 11
9 # define mp make_pair
10 # define f first
11 # define s second
12
13 using namespace std;
14
15 ifstream f("miting.in");
16 ofstream g("miting.out");
17
18 short dx[] = {0, 1, 0, -1};
19 short dy[] = {1, 0, -1, 0};
20
21 int p, n, m, l, CiQ, CsQ, K;
22 short Costmin[N][N][NR*NR], nr[NR][NR], poz[NR], L[NR*NR], C[NR*NR],
23 X[NR], Y[NR], Min, test[N][N];
24 char a[NR][NR], cuv[N];
25 pair <short, short> q[NR*NR*NR];
26
27 void bordare ()
28 {
29     for (int i=0; i<=m+1; ++i)
30         a[0][i] = a[n+1][i] = '#';
31     for (int i=0; i<=n+1; ++i)
32         a[i][0] = a[i][m+1] = '#';
33 }
34
35 void LEE (short i, short j)
36 {
37     short k, lv, cv, l, c, urm, act;
38
39     while (CiQ <= CsQ)
40     {
41         l = q[CiQ].f; c = q[CiQ].s; ++CiQ;
42         act = nr[l][c];
43         for (k=0; k<4; ++k)
44         {
45             lv = l + dx[k]; cv = c + dy[k];
46             urm = nr[lv][cv];
47             if (urm != 0 &&
48                 Costmin[i][j][urm] > Costmin[i][j][act] + 1)
49             {
50                 Costmin[i][j][urm] = Costmin[i][j][act] + 1;
```

```

51             q[++CsQ] = mp(lv, cv);
52         }
53     }
54 }
55 }
56
57 int main ()
58 {
59     short i, j, k, lmin, cmin, lmax = 0, cmax = 0, I, J;
60
61     f >> p >> n >> m; f.get();
62     bordare ();
63     f.getline(cuv+1, N);
64     l = strlen(cuv+1);
65     for (i=1; i<=l; ++i) //pozitia unei litere in cuvant
66         poz[cuv[i]] = i;
67     for (i=1; i<=n; ++i) //citesc matricea
68         f >> (a[i]+1);
69
70     lmin = INF;
71     cmin = INF;
72     for (i=1; i<=n; ++i)
73     {
74         for (j=1; j<=m; ++j)
75         {
76             if (a[i][j] != '#')
77                 { //loc pe unde se poate merge
78                     ++K;
79                     nr[i][j] = K;
80                     L[K] = i; C[K] = j;
81                 }
82             if (a[i][j] != '#' && a[i][j] != '_')
83             { // e litera
84                 lmin = min(lmin, i);
85                 cmin = min(cmin, j);
86
87                 lmax = max(lmax, i);
88                 cmax = max(cmax, j);
89
90                 X[poz[a[i][j]]] = i; //pozitia literelor
91                 Y[poz[a[i][j]]] = j;
92             }
93         }
94     }
95     if (p == 1)
96     {
97         g << (lmax-lmin+1) * (cmax-cmin+1) << "\n";
98         return 0;
99     }
100    for (i=1; i<=l; ++i)
101    {
102        for (j=1; j<=K; ++j)
103            Costmin[i][j] = INF; // Costmin[i][j][K] - costul minim
104                                         // pentru a ajunge cu literele dintr-o i-j
105                                         // pe o K-a pozitie libera din matrice
106
107        Costmin[i][nr[X[i]][Y[i]]] = 0;
108
109        CiQ = 1; CsQ = 1;
110        q[ciQ] = mp(X[i], Y[i]);
111        LEE (i, i);
112    }
113
114    for (k=2; k<=l; ++k) //lungimea echipei'
115        for (i=1; i<=l-k+1; ++i)
116        { //pozitia de start a echipei'
117            j = i+k-1;
118            CiQ = 1; CsQ = 0;
119            test[i][j] = INF;
120
121            for (I=1; I<=K; ++I)
122            { //pozitia de pe harta care e libera
123                Costmin[i][j][I] = INF;
124                Min = Costmin[i][j][I];
125
126                for (J=i; J<j; ++J)

```

```

127             Min = min(Min, (short)(Costmin[i][J][I] +
128                                         Costmin[J+1][j][I]));
129
130             Costmin[i][j][I] = Min;
131             q[++CsQ] = mp(L[I], C[I]);
132         }
133
134         LEE (i, j);
135
136         for (I=1; I<=K; ++I)
137             test[i][j] = min(test[i][j], Costmin[i][j][I]);
138     }
139     Min = INF;
140     for (i=1; i<=K; ++i)
141         Min = min(Min, Costmin[1][1][i]);
142
143     if (Min == INF) g << "-1\n";
144     else g << Min << "\n";
145
146     return 0;
147 }
```

Listing 4.2.3: miting_gc2.cpp

```

1 /* prof. Constantin Galatan
2 */
3 #include <iostream>
4 #include <iomanip>
5 #include <algorithm>
6 #include <vector>
7 #include <queue>
8 #include <iomanip>
9
10 using namespace std;
11
12 ifstream fin("miting.in");
13 ofstream fout("miting.out");
14
15 const int MaxN = 61, MaxC = 11, MaxV = 3601, Inf = (1 << 13),
16     di[] = { -1, 0, 1, 0}, dj[] = { 0, 1, 0, -1};
17
18 using Matr = short[MaxC][MaxC][MaxV];
19 using PII = pair<short, short>;
20
21 Matr c;
22 int N, M, n, nc, z[MaxN][MaxN]; // Nr[i][j] = numarul de ordine
23 // corespunzator celulei(i, j)
24 vector<string> H;
25 string cuv;
26 deque<PII> D;
27
28 bool Ok(int i, int j);
29
30 int main()
31 {
32     int imin = MaxN, jmin = MaxN, imax = -1, jmax = -1, P;
33
34     fin >> P >> N >> M >> cuv;
35     H = vector<string>(N, string(M, ' '));
36     for (int i = 0; i < N; ++i)
37         for (int j = 0; j < M; ++j)
38         {
39             char c; fin >> c;
40             if ( isupper(c) )
41             {
42                 imin = min(imin, i); jmin = min(jmin, j);
43                 imax = max(imax, i); jmax = max(jmax, j);
44                 D.push_front({i, j});
45             }
46             else
47                 if ( c != '#' )
48                     D.push_back({i, j});
49             H[i][j] = c;
50         }
51
52     fin.close();
```

```

53     if ( P == 1 )
54     {
55         fout << (imax - imin + 1) * (jmax - jmin + 1) << '\n';
56         fout.close();
57         exit(0);
58     }
59     nc = cuv.size();
60
61
62     vector<int> pos('Z' + 1); // pozitiile caracterelor in cuvant
63     for ( int i = 0; i < nc; ++i )
64         pos[cuv[i]] = i;
65
66     // Dupa sortare primele nc celule vor contine literele i ordinea din cuvant
67     sort(D.begin(), D.begin() + nc,
68           [&pos](const PII& p1, const PII& p2)
69           { return pos[H[p1.first][p1.second]] < pos[H[p2.first][p2.second]]; })
70     );
71
72     n = D.size();
73
74     for ( int k = 0; k < n; ++k )
75     {
76         int i = D[k].first, j = D[k].second;
77         z[i][j] = k;
78     }
79
80     for ( int i = 0; i < nc; ++i )
81     {
82         for ( int u = 0; u < n; ++u )
83             for ( int j = 0; j < nc; ++j )
84                 c[i][j][u] = Inf;
85
86         c[i][i] = 0;
87     }
88
89     for ( int l = 1; l <= nc; ++l )
90         for ( int i = 0; i < nc - l + 1; ++i )
91         {
92             short j = i + l - 1, old;
93             queue<int> Q;
94             for ( int x = 0; x < n; ++x )
95             {
96                 if ( l >= 2 )
97                 {
98                     old = c[i][j][x];
99                     for ( int k = i; k < j; ++k )
100                         c[i][j][x] = min(c[i][j][x],
101                                           short(c[i][k][x] + c[k + 1][j][x]));
102                 }
103
104                 if ( c[i][j][x] < old || c[i][j][x] == 0 )
105                     Q.push(x);
106             }
107             if ( l == nc ) break;
108             int I, J, iv, jv, x, y;
109             while ( !Q.empty() )
110             {
111                 x = Q.front(); Q.pop();
112                 I = D[x].first;
113                 J = D[x].second;
114
115                 for ( int d = 0; d < 4; ++d )
116                 {
117                     iv = I + di[d];
118                     jv = J + dj[d];
119                     y = z[iv][jv];
120                     if ( Ok(iv, jv) && c[i][j][y] > c[i][j][x] + 1 )
121                     {
122                         c[i][j][y] = c[i][j][x] + 1;
123                         Q.push(y);
124                     }
125                 }
126             }
127         }
128

```

```

129     short res = Inf;
130     for (short x = 0; x < n; ++x)
131         res = min(res, c[0][nc - 1][x]);
132
133     if ( res != Inf )
134         fout << res << '\n';
135     else
136         fout << -1 << '\n';
137
138     fout.close();
139 }
140
141 inline bool Ok(int i, int j)
142 {
143     if ( i < 0 || j < 0 || i >= N || j >= M ) return false;
144     if ( H[i][j] == '#' ) return false;
145     return true;
146 }
```

Listing 4.2.4: miting_gc3.cpp

```

1 // prof.. Constantin Galatan
2 #include <iostream>
3 #include <iomanip>
4 #include <algorithm>
5 #include <vector>
6 #include <queue>
7
8 using namespace std;
9
10 ifstream fin("miting.in");
11 ofstream fout("miting.out");
12
13 const int MaxN = 61, MaxC = 10, MaxV = 3601, Inf = 1 << 13,
14     di[] = { -1, 0, 1, 0}, dj[] = { 0, 1, 0, -1};
15 using Pair = pair<short, short>;
16
17 short c[MaxC][MaxC][MaxV];
18 int M, N, n, nc, cel[MaxN][MaxN]; // cel[i][j] = numarul de ordine
19 // coresponzator celulei (i, j)
20 string cuv;
21 deque<Pair> D;
22 int Q[30*MaxV], L, R = -1;
23 char H[MaxN][MaxN], ch;
24 inline void Lee(short i, short j);
25 bool Ok(short i, short j);
26
27 int main()
28 {
29     short imin = MaxN, jmin = MaxN, imax = -1, jmax = -1, p;
30
31     fin >> p >> N >> M >> cuv;
32     fin.get();
33     for (short i = 0; i < N; ++i)
34     {
35         fin.getline(H[i], 101);
36         for (short j = 0; j < M; ++j)
37         {
38             ch = H[i][j];
39             if ( isupper(ch) )
40             {
41                 D.push_front({i, j});
42                 imin = min(imin, i); jmin = min(jmin, j);
43                 imax = max(imax, i); jmax = max(jmax, j);
44             }
45             else
46                 if ( ch != '#' )
47                     D.push_back({i, j});
48         }
49     }
50     fin.close();
51
52     if ( p == 1 )
53     {
54         fout << (imax - imin + 1) * (jmax - jmin + 1) << '\n';
55     }
56 }
```

```

55         fout.close();
56         exit(0);
57     }
58
59     nc = cuv.size();
60     vector<int> pos('Z' + 1); // pozitiile caracterelor in cuvant
61     for ( int i = 0; i < nc; ++i)
62         pos[cuv[i]] = i;
63
64     // Dupa sortare primele nc celule vor contine literele in ordinea din cuvant
65     sort(D.begin(), D.begin() + nc, [&pos](const Pair& p1, const Pair& p2)
66     { return pos[H[p1.first][p1.second]] < pos[H[p2.first][p2.second]]; })
67     );
68
69     n = D.size();
70     for (int k = 0, i, j; k < n; ++k)
71     {
72         i = D[k].first, j = D[k].second;
73         cel[i][j] = k;
74     }
75
76     for (int i = 0; i < nc; ++i)
77     {
78         for (int x = 0; x < n; ++x)
79             for (int j = 0; j < nc; ++j)
80                 c[i][j][x] = Inf;
81
82         c[i][i][i] = 0;
83         Q[++R] = i;
84         Lee(i, i);
85     }
86
87     for ( int l = 2; l <= nc; ++l)
88         for (int i = 0; i < nc - l + 1; ++i)
89         {
90             int j = i + l - 1, cmin;
91             for (int x = 0; x < n; ++x)
92             {
93                 cmin = c[i][j][x];
94                 for (int k = i; k < j; ++k)
95                     cmin = min(cmin, c[i][k][x] + c[k + 1][j][x]);
96
97                 if ( cmin < c[i][j][x] )
98                 {
99                     c[i][j][x] = cmin;
100                    Q[++R] = x;
101                }
102            }
103            if ( l < nc ) Lee(i, j);
104        }
105
106     short res = Inf;
107     for (short x = 0; x < n; ++x)
108         res = min(res, c[0][nc - 1][x]);
109
110     fout << res << '\n';
111
112     fout.close();
113 }
114
115 inline void Lee(short i, short j)
116 {
117     short I, J, iv, jv, x, y;
118     while ( L <= R )
119     {
120         x = Q[L++];
121         I = D[x].first; J = D[x].second;
122
123         for (int dir = 0; dir < 4; ++dir )
124         {
125             iv = I + di[dir];
126             jv = J + dj[dir];
127             y = cel[iv][jv];
128             if ( Ok(iv, jv) && c[i][j][y] > c[i][j][x] + 1)
129             {
130                 c[i][j][y] = c[i][j][x] + 1;

```

```

131             Q[++R] = y;
132         }
133     }
134 }
135 L = 0, R = -1;
136 }
137
138 inline bool Ok(short i, short j)
139 {
140     if (i < 0 || j < 0 || i >= N || j >= M) return false;
141     if (H[i][j] == '#') return false;
142     return true;
143 }
```

Listing 4.2.5: miting_gc4.cpp

```

1 /* prof. Constantin Galatan
2 */
3 #include <iostream>
4 #include <iomanip>
5 #include <algorithm>
6 #include <vector>
7 #include <queue>
8
9 using namespace std;
10
11 ifstream fin("miting.in");
12 ofstream fout("miting.out");
13
14 const short MaxN = 61, MaxC = 10, MaxV = 3601, Inf = 1 << 13,
15     di[] = { -1, 0, 1, 0}, dj[] = { 0, 1, 0, -1};
16 using Pair = pair<short, short>;
17
18 short c[MaxC][MaxC][MaxV];
19 int M, N, n, nc, cel[MaxN][MaxN]; // cel[i][j] = numarul de ordine
20 // coresponzator celulei (i, j)
21 string cuv;
22 deque<Pair> D;
23
24 char H[MaxN][MaxN], ch;
25
26 struct State
27 {
28     short x, cost;
29     bool operator > (const State& st) const
30     {
31         return cost > st.cost;
32     }
33 };
34
35 priority_queue<State, vector<State>, greater<State> > Q;
36
37 inline void Lee(short i, short j);
38 bool Ok(short i, short j);
39
40 int main()
41 {
42     short imin = MaxN, jmin = MaxN, imax = -1, jmax = -1, p;
43
44     fin >> p >> N >> M >> cuv;
45     fin.get();
46     for (short i = 0; i < N; ++i)
47     {
48         fin.getline(H[i], 101);
49         for (short j = 0; j < M; ++j)
50         {
51             ch = H[i][j];
52             if (isupper(ch) )
53             {
54                 D.push_front({i, j});
55                 imin = min(imin, i); jmin = min(jmin, j);
56                 imax = max(imax, i); jmax = max(jmax, j);
57             }
58             else
59                 if (ch != '#')
```

```

60             D.push_back({i, j});
61     }
62 }
fin.close();
64
65 if ( p == 1 )
66 {
67     fout << (imax - imin + 1) * (jmax - jmin + 1) << '\n';
68     fout.close();
69     exit(0);
70 }
71
72 nc = cuv.size();
73 vector<int> pos('Z' + 1); // pozitiile caracterelor in cuvant
74 for ( int i = 0; i < nc; ++i)
75     pos[cuv[i]] = i;
76
77 // Dupa sortare primele nc celule vor contine literele i ordinea din cuvant
78 sort(D.begin(), D.begin() + nc, [&pos](const Pair& p1, const Pair& p2)
79     { return pos[H[p1.first][p1.second]] < pos[H[p2.first][p2.second]]; }
80 );
81
82 n = D.size();
83 for (int k = 0, i, j; k < n; ++k)
84 {
85     i = D[k].first, j = D[k].second;
86     cel[i][j] = k;
87 }
88
89 for (short i = 0; i < nc; ++i)
90 {
91     for (short x = 0; x < n; ++x)
92         for (short j = 0; j < nc; ++j)
93             c[i][j][x] = Inf;
94
95     c[i][i][i] = 0;
96     Q.push({i, 0});
97     Lee(i, i);
98 }
99
100 for (short l = 2; l <= nc; ++l)
101     for (short i = 0; i < nc - l + 1; ++i)
102     {
103         short j = i + l - 1, cmin;
104         for (short x = 0; x < n; ++x)
105         {
106             cmin = c[i][j][x];
107             for (short k = i; k < j; ++k)
108                 cmin = min(cmin, (short)(c[i][k][x] + c[k + 1][j][x]));
109
110             if ( cmin < c[i][j][x] )
111                 c[i][j][x] = cmin,
112                 Q.push({x, cmin});
113         }
114         if ( l < nc ) Lee(i, j);
115     }
116
117 short res = Inf;
118 for (short x = 0; x < n; ++x)
119     res = min(res, c[0][nc - 1][x]);
120
121 fout << res << '\n';
122
123 fout.close();
124 }
125
126 inline void Lee(short i, short j)
127 {
128     short I, J, iv, jv, x, y, cost;
129     while ( !Q.empty() )
130     {
131         x = Q.top().x; cost = Q.top().cost; Q.pop();
132         if ( c[i][j][x] < cost )
133             continue;
134         I = D[x].first; J = D[x].second;
135

```

```

136         for (int dir = 0; dir < 4; ++dir)
137     {
138         iv = I + di[dir];
139         jv = J + dj[dir];
140         y = cel[iv][jv];
141         if (Ok(iv, jv) && c[i][j][y] > c[i][j][x] + 1)
142         {
143             c[i][j][y] = c[i][j][x] + 1;
144             Q.push({y, c[i][j][y]});
145         }
146     }
147 }
148 }
149
150 inline bool Ok(short i, short j)
151 {
152     if (i < 0 || j < 0 || i >= N || j >= M) return false;
153     if (H[i][j] == '#') return false;
154     return true;
155 }
```

Listing 4.2.6: miting_gc5.cpp

```

1 /* prof. Constantin Galatan
2 */
3 #include <iostream>
4 #include <algorithm>
5 #include <vector>
6 #include <queue>
7
8 using namespace std;
9
10 ifstream fin("miting.in");
11 ofstream fout("miting.out");
12
13 const short MaxN = 61, MaxC = 10, MaxV = 3601, Inf = 1 << 13,
14     di[] = { -1, 0, 1, 0}, dj[] = { 0, 1, 0, -1};
15 using Pair = pair<short, short>;
16
17 short c[MaxC][MaxC][MaxV];
18 int M, N, n, nc, cel[MaxN][MaxN]; // cel[i][j] = numarul de ordine
19 // coresponzator celulei (i, j)
20 string cuv;
21 deque<Pair> D;
22
23 char H[MaxN][MaxN], ch;
24
25 struct State
26 {
27     short i, j, x;
28     bool operator > (const State& st) const
29     {
30         return c[i][j][x] > c[st.i][st.j][st.x];
31     }
32 };
33
34 priority_queue<State, vector<State>, greater<State> > Q;
35
36 inline void Lee(short i, short j);
37 bool Ok(short i, short j);
38
39 int main()
40 {
41     short imin = MaxN, jmin = MaxN, imax = -1, jmax = -1, p;
42
43     fin >> p >> N >> M >> cuv;
44     fin.get();
45     for (short i = 0; i < N; ++i)
46     {
47
48         fin.getline(H[i], 101);
49         for (short j = 0; j < M; ++j)
50         {
51             ch = H[i][j];
52             if (isupper(ch) )
```

```

53         {
54             D.push_front({i, j});
55             imin = min(imin, i); jmin = min(jmin, j);
56             imax = max(imax, i); jmax = max(jmax, j);
57         }
58     else
59         if ( ch != '#')
60             D.push_back({i, j});
61     }
62 }
63 fin.close();
64
65 if ( p == 1 )
66 {
67     fout << (imax - imin + 1) * (jmax - jmin + 1) << '\n';
68     fout.close();
69     exit(0);
70 }
71
72 nc = cuv.size();
73 vector<int> pos('Z' + 1); // pozitiile caracterelor in cuvant
74 for ( int i = 0; i < nc; ++i)
75     pos[cuv[i]] = i;
76
77 // Dupa sortare primele nc celule vor contine literele in ordinea din cuvant
78 sort(D.begin(), D.begin() + nc, [&pos](const Pair& p1, const Pair& p2)
79     { return pos[H[p1.first][p1.second]] < pos[H[p2.first][p2.second]]; })
80 );
81
82 n = D.size();
83 for (int k = 0, i, j; k < n; ++k)
84 {
85     i = D[k].first, j = D[k].second;
86     cel[i][j] = k;
87 }
88
89 for (short i = 0; i < nc; ++i)
90 {
91     for (short x = 0; x < n; ++x)
92         for (short j = 0; j < nc; ++j)
93             c[i][j][x] = Inf;
94
95     c[i][i] = 0;
96     Q.push({i, i, i});
97     Lee(i, i);
98 }
99
100 for (short l = 2; l <= nc; ++l)
101     for (short i = 0; i < nc - l + 1; ++i)
102     {
103         short j = i + l - 1, cmin;
104         for (short x = 0; x < n; ++x)
105         {
106             cmin = c[i][j][x];
107             for (short k = i; k < j; ++k)
108                 cmin = min(cmin, (short)(c[i][k][x] + c[k + 1][j][x]));
109
110             if ( cmin < c[i][j][x] )
111                 c[i][j][x] = cmin,
112                 Q.push({i, j, x});
113         }
114         Lee(i, j);
115     }
116
117 short res = Inf;
118 for (short x = 0; x < n; ++x)
119     res = min(res, c[0][nc - 1][x]);
120
121 fout << res << '\n';
122
123 fout.close();
124 }
125
126 inline void Lee(short i, short j)
127 {
128     short I, J, iv, jv, x, y;

```

```
129     while ( !Q.empty() )  
130     {  
131         x = Q.top().x; Q.pop();  
132         I = D[x].first; J = D[x].second;  
133  
134         for (int dir = 0; dir < 4; ++dir)  
135         {  
136             iv = I + di[dir];  
137             jv = J + dj[dir];  
138             y = cel[iv][jv];  
139             if ( Ok(iv, jv) && c[i][j][y] > c[i][j][x] + 1)  
140             {  
141                 c[i][j][y] = c[i][j][x] + 1;  
142                 Q.push({i, j, y});  
143             }  
144         }  
145     }  
146 }  
147  
148 inline bool Ok(short i, short j)  
149 {  
150     if ( i < 0 || j < 0 || i >= N || j >= M ) return false;  
151     if ( H[i][j] == '#' ) return false;  
152     return true;  
153 }
```

Capitolul 5

OJI 2015

5.1 charlie

Problema 1 - charlie

100 de puncte

Charlie a decis să se joace cu literele dintr-un sir de caractere, sir ce contine doar literele mici ale alfabetului englez 'a'...'z'. Jocul constă în a elimina litere din sir după următoarea regulă: fie L_1, L_2, L_3 trei litere aflate pe poziții consecutive în sir, atunci litera L_2 poate fi eliminată dacă și numai dacă este strict mai mică lexicografic decât literele L_1 și L_3 .

Pentru a face jocul mai interesant, Charlie atașează eliminării literei L_2 un cost egal cu valoarea maximă dintre $\bar{o}(L_1)$ și $\bar{o}(L_3)$, unde prin \bar{o} (litera) înțelegem numărul de ordine al literei respective în alfabet ($\bar{o}('a') = 1, \bar{o}('b') = 2, \dots, \bar{o}('z') = 26$). Charlie aplică în mod repetat procedeul de eliminare și calculează suma costurilor eliminărilor efectuate.

Cerințe

Fieind dat un sir de caractere să se determine:

- Lungimea maximă a unei secvențe de litere alternante, adică o secvență pentru care literele aflate pe poziții consecutive sunt de forma: $L_i > L_{i+1} < L_{i+2} > L_{i+3} < L_{i+4} > \dots < L_j$.
- Suma maximă pe care o poate obține Charlie aplicând în mod repetat procedeul de eliminare a literelor, precum și sirul obținut în final.

Date de intrare

Fișierul de intrare **charlie.in** conține pe prima linie un număr natural p . Pentru toate testele de intrare, numărul p poate avea doar valoarea 1 sau valoarea 2. Pe următoarea linie se află un sir de caractere.

Date de ieșire

Dacă valoarea lui p este 1, se va rezolva numai punctul a) din cerință.

În acest caz, în fișierul de ieșire **charlie.out** se va scrie un singur număr natural L ce reprezintă lungimea maximă a unei secvențe de litere alternante.

Dacă valoarea lui p este 2, se va rezolva numai punctul b) din cerință.

În acest caz, fișierul de ieșire **charlie.out** va conține două linii. Pe prima linie se va afla sirul rezultat în urma eliminărilor repetitive de litere respectând regula enunțată, iar pe cea de-a doua linie suma maximă obținută.

Restricții și precizări

- $3 \leq$ numărul de litere ale sirului inițial ≤ 100000
- Pentru rezolvarea corectă a primei cerințe se acordă 25 de puncte, iar pentru cerința a două se acordă 75 de puncte.
 - Pentru 30% dintre teste numărul de litere ale sirului ≤ 1000

Exemple

charlie.in	charlie.out	Explicații
1 cadgfacbda	5	<p>p = 1</p> <p>Secvențele alternante corect formate sunt: <i>cad</i>, <i>facbd</i>. Lungimea maximă este 5</p> <p>Atenție! Pentru acest test se rezolvă doar cerința a).</p>
2 cbcabadbac	ccdc 21	<p>p = 2</p> <p>Șirul inițial: <i>cbcabadbac</i></p> <p>Eliminăm din secvența <i>bad</i> litera <i>a</i> și adăugăm la suma valoarea 4</p> <p>Șirul rezultat în urma eliminării este: <i>cbcabdbac</i></p> <p>Eliminăm din secvența <i>bac</i> litera <i>a</i> și adăugăm la suma valoarea 3</p> <p>Șirul rezultat în urma eliminării este: <i>cbcabdbc</i></p> <p>Eliminăm din secvența <i>dbc</i> litera <i>b</i> și adăugăm la suma valoarea 4</p> <p>Șirul rezultat în urma eliminării este: <i>cbcabdc</i></p> <p>Eliminăm din secvența <i>cab</i> litera <i>a</i> și adăugăm la suma valoarea 3</p> <p>Șirul rezultat în urma eliminării este: <i>cbcbdc</i></p> <p>Eliminăm din secvența <i>cbd</i> litera <i>b</i> și adăugăm la suma valoarea 4</p> <p>Șirul rezultat în urma eliminării este: <i>cbcdc</i></p> <p>Eliminăm din secvența <i>cbc</i> litera <i>b</i> și adăugăm la suma valoarea 3</p> <p>Șirul rezultat în urma eliminării este: <i>ccdc</i></p> <p>Nu mai sunt posibile eliminări. Suma maximă obținută este 21.</p> <p>Atenție! Pentru acest test se rezolvă doar cerința b).</p>

Timp maxim de executare/test: **0.5** secunde

Memorie: total **4 MB** din care pentru stivă **2 MB**

Dimensiune maximă a sursei: **5 KB**

5.1.1 Indicații de rezolvare

prof. Eugen Nodea, Colegiul Național "Tudor Vladimirescu", Târgu Jiu

Punctul a) 25 puncte

- printr-o parcurgere liniară a șirului se determină lungimea maximă a unei secvențe alternante de litere

Punctul b) 75 puncte

Este o problemă de *Greedy + stivă* (st)

Rezolvarea se bazează pe afirmația că ordinea în care se efectuează operațiile de eliminare a literelor nu influențează rezultatul final (vezi demonstrația).

Fie șirul de litere $s = (s[i])_{i=0,1,\dots,n-1}$, unde $n = \text{strlen}(s)$

```

push (st, s[0])
push (st, s[1])
pentru i = 2,n-1 executa
    cat_timp (s[i] > st[vf] && st[vf] < st[vf-1]) executa
        pop (st) // eliminam din stiva st[vf]
        sf_cat_timp
        push (st,s[i])
sf_pentru

```

Complexitatea : $O(n)$

charlie - Rezolvare teoretică (prof. Stelian Ciurea)

Rezolvarea problemei se bazează pe afirmația că ordinea în care se efectuează operațiile de eliminare a unui minim local nu influențează rezultatul final. Aceasta deoarece în momentul în care un element a_i care este minim local urmează să fie eliminat, vecinii lui din pozițiile a_{i-1} și a_{i+1} sunt unic determinați, indiferent ce transformări ale șirului au avut loc mai înainte.

Dar înainte de a demonstra această afirmație, este bine să observăm următoarele:

1) dacă în șir există *paliere* de două sau mai multe elemente (unde *paliere* = subșir de elemente egale aflate în poziții consecutive), elementele acestor paliere nu vor putea fi eliminate. Justificare afirmației este imediată: fie $a_i = a_{i+1}$; niciunul dintre aceste elemente nu vor putea deveni la un

moment dat *minim local*. Să presupunem prin reducere la absurd că a_i ar putea deveni minim local.

Aceasta presupune că la un moment anterior a_{i+1} a fost eliminat și în locul lui a venit un element mai mare. Ceea ce este fals deoarece a_{i+1} nu putea fi eliminat atâtă timp cât în poziția din *stânga* lui se află a_i ! Raționamentul este similar pentru a_{i-1} .

2) dacă avem două elemente oarecare a_i și a_j și $i < j$ (adică a_i se află în *stânga* lui a_j) și prin aplicarea unui număr oarecare de eliminări de minime locale din sir, a_i și a_j au rămas în sir, atunci în continuare $i < j$ (adică a_i se va afla tot în *stânga* lui a_j) chiar dacă numărul de elemente dintre pozițiile i și j s-a micșorat, eventual s-a redus la 0. Această afirmație este evidentă prin modul în care se efectuează operația de eliminare: la efectuarea unei eliminări, un element nu poate avansa spre *stânga* mai mult de o poziție. Pentru ca a_j să treacă înaintea lui a_i , ar fi necesar ca la o singură eliminare a_j să avanseze cu două poziții mai mult decât a_i ceea ce este imposibil.

Atenție! Repetăm că această afirmație se referă la situația în care a_i și a_j au rămas în sir!

Să trecem acum la justificarea afirmației de la începutul demonstrației și anume că în momentul în care un element a_i care este minim local și urmează să fie eliminat, vecinii lui din pozițiile a_{i-1} și a_{i+1} sunt unic determinați, indiferent ce transformări ale sirului au avut loc mai înainte. Evident, toată discuția are rost dacă a_i va putea fi eliminat. Dacă unul dintre vecinii lui este egal cu el, conform afirmației 1, a_i nu va putea fi eliminat și deci analiza nu mai are rost!

Se disting următoarele situații:

- a_i este *minim local*, atunci nici a_{i-1} și nici a_{i+1} nu vor putea fi eliminate înaintea lui a_i deoarece atâtă timp cât în sir există a_i , niciunul dintre cele două nu poate fi minim local!

- a_i nu este minim local; atunci cel puțin unul dintre vecinii lui este strict mai mic decât el. Să presupunem că elementul aflat în dreapta lui a_i este mai mic decât a_i : $a_i > a_{i+1}$. Atunci pentru ca a_i să poată fi eliminat este obligatoriu ca în pozițiile $i+2, i+3, \dots, L$ (L fiind ultima poziție din sir) să existe cel puțin un element strict mai mare decât a_i . Dacă toate elementele din dreapta lui a_i ar fi mai mici sau egale cu a_i , este evident că el nu va putea deveni minim local!

Să presupunem că sunt mai multe elemente mai mari decât a_i . Fie acestea în pozițiile p_1, p_2, \dots, p_k , cu $p_1 < p_2 < \dots < p_k$. Deci elementul din p_1 este cel mai din *stânga* dintre acestea și deci între pozițiile $i+1$ și $p_1 - 1$ nu există niciun element mai mare decât a_i . În aceste condiții, prin aplicarea *eliminărilor de minime locale*, singurul element mai mare decât a_i care poate să apară în poziția $i+1$ este cel aflat inițial în p_1 (adică primul element mai mare decât a_i aflat în dreapta lui a_i).

Să presupunem prin reducere la absurd că ar putea să apară un element mai mare decât a_i aflat inițial într-o poziție din dreapta lui p_1 . Conform afirmației 2, aceasta nu poate să se întâmple dacă elementul din p_1 a rămas în sir. Deci singura posibilitate ar fi ca elementul din p_1 să fie eliminat. Dar aceasta presupune că el devine minim local la un moment dat (în condițiile în care a_i este încă în sir!), ceea ce este imposibil deoarece conform ipotezei făcute, între pozițiile $i+1$ și $p_1 - 1$ nu există niciun element mai mare decât a_i , iar $a_i < a_{p_1}$!

Cazul $a_i > a_{i-1}$ se tratează similar. Cu aceasta considerăm afirmația demonstrată.

Din această demonstrație rezultă că dacă un element oarecare a_i va fi eliminat și inițial el nu este minim local, atunci în momentul în care devine minim local el va fi încadrat de cele mai apropiate elemente strict mai mari decât el aflate în *stânga* și respectiv în *dreapta* lui.

În concluzie, rezultatul aplicării eliminărilor din sir este unic indiferent de ordinea în care acestea se efectuează, deci orice program care efectuează corect toate eliminările posibile va conduce la rezultatul corect.

Singura problemă este complexitatea algoritmului.

5.1.2 *Rezolvare detaliată

5.1.3 Cod sursă

Listing 5.1.1: charlie_adriana.cpp

```

1  /*
2   Sursa 100 p
3   prof. Adriana Simulescu
4   Liceul Teoretic GRIGORE MOISIL Timisoara
5  */
6 #include<iostream>
7 #include<iostream>
8 #include<string.h>
9
10 using namespace std;
11
12 ifstream in("charlie.in");
13 ofstream out("charlie.out");
14
15 int n,p,suma;
16 char s[100001],s1[1000001];
17
18 int main()
19 { int i,L=0,l,j=0;
20   in>>p; in.get();
21   in>>s;
22
23   if(p==1)
24   {
25     i=1;
26     while (s[i-1]<=s[i]&&s[i]) i++;
27     l=2;
28     n=strlen(s);
29     for(++i;i<n;i++)
30       if((s[i]-s[i-1])*(s[i-1]-s[i-2])<0)
31         {l++; if(s[i]>s[i-1]) if(l>L) L=l;}
32     else { while (s[i-1]<=s[i]&&s[i]) i++;
33           l=2;
34         }
35
36     out<<L<<endl;
37   }
38   else{
39     i=2;
40     s1[0]=s[0];
41     s1[1]=s[1]; j=1;
42     while(s[i])
43     {
44       while(j>=1&&s1[j]<s1[j-1]&&s1[j]<s[i])
45       {
46         suma+=(max(s1[j-1],s[i])-‘a’+1);
47         j--;
48       }
49       j++;
50       s1[j]=s[i];
51       // s1[j+1]=0;
52       // out<<s1<<endl;
53       i++;
54     }
55     s1[j+1]=0;
56     out<<s1<<endl<<suma<<endl;
57   }
58 }
```

Listing 5.1.2: charlie_eugen0.cpp

```

1 /*
2   Sursa oficiala 100p - stiva
3   Complexitate: O (n)
4   prof. Eugen Nodea
5   Colegiul National "Tudor Vladimirescu", Tg-Jiu
6 */
7 # include <iostream>
8 # include <cstring>
9
10 using namespace std;
11
12 # define max(a, b) ((a) < (b) ? (b) : (a))
```

```

13 # define Lmax 100003
14
15 ifstream f("charlie.in");
16 ofstream g("charlie.out");
17
18 char s[Lmax], st[Lmax];
19 int p, L, vf, k, i, Max, j;
20 long long S;
21
22 void afis()
23 {
24     for(int i=1; i <=vf; ++i)
25         g << st[i];
26     g<< "\n";
27 }
28
29 int main()
30 {
31     f >> p; f.get();
32     f.getline(s, 100001);
33     L = strlen(s);
34
35     if (p == 1) //a
36     {
37         k = i = 0;
38         while ( i < L )
39         {
40             j = i;
41             while ( s[j] > s[j+1] && s[j+1] < s[j+2] && j + 2 < L)
42                 j += 2;
43             if (j - i >= 2)
44             {
45                 if ( j - i + 1 > Max ) Max = j - i + 1;
46                 i = j;
47             }
48             ++i;
49         }
50         g << Max << "\n";
51     }
52     else //b
53     {
54         st[1] = s[0]; st[2] = s[1];
55         i = vf = 2;
56         while ( i < L )
57         {
58             while (s[i] > st[vf] && st[vf] < st[vf-1] && vf > 1)
59             {
60                 S += max(s[i] - 'a' + 1, st[vf-1] - 'a' + 1);
61                 --vf;
62             }
63             st[++vf] = s[i];
64             ++i;
65         }
66         afis();
67         g << S << "\n";
68     }
69     return 0;
70 }

```

Listing 5.1.3: charlie_eugen1.cpp

```

1 /*
2     Sursa 100p - implementare stack-STL
3     prof. Eugen Nodea
4     Colegiul National "Tudor Vladimirescu", Tg-Jiu
5 */
6 # include <fstream>
7 # include <cstring>
8 # include <stack>
9
10 using namespace std;
11
12 # define max(a, b) ((a) < (b) ? (b) : (a))
13 # define Lmax 100001
14

```

```

15  ifstream f("charlie.in");
16  ofstream g("charlie.out");
17
18  char s[Lmax], vf_a, vf, sol[Lmax];
19  int p, L, semn, k, i, Max, j;
20  stack <char> st;
21  long long S;
22
23  void afis()
24  {
25      int i = st.size();
26      sol[i] = '\0';
27      while (!st.empty())
28      {
29          sol[--i] = st.top();
30          st.pop();
31      }
32      g << sol << "\n";
33  }
34
35  int main()
36  {
37      f >> p; f.get();
38      f.getline(s, 100001);
39      L = strlen(s);
40
41      if (p == 1) //a
42      {
43          k = i = 0;
44          while (i < L)
45          {
46              j = i;
47              while (s[j] > s[j+1] && s[j+1] < s[j+2] && j + 2 < L)
48                  j += 2;
49              if (j - i >= 2)
50              {
51                  if (j - i + 1 > Max) Max = j - i + 1;
52                  i = j;
53              }
54              ++i;
55          }
56          g << Max << "\n";
57      }
58      else //b
59      {
60          vf_a = s[0]; vf = s[1];
61          st.push(s[0]); st.push(s[1]);
62          i = 2;
63          while (i < L)
64          {
65              while (s[i] > vf && vf < vf_a && st.size() > 1)
66              {
67                  S += max(s[i] - 'a' + 1, vf_a - 'a' + 1);
68                  if (st.size() <= 2)
69                  {
70                      st.pop();
71                      vf = st.top();
72                  }
73                  else
74                  {
75                      vf = vf_a;
76                      st.pop(); // sterg vf
77                      st.pop(); // sterg vf_anterior
78                      vf_a = st.top(); // actualize vf_a
79                      st.push(vf); // adaug vf
80                  }
81              }
82              vf_a = vf; vf = s[i];
83              st.push(s[i]);
84              ++i;
85          }
86          afis();
87          g << S << "\n";
88      }
89      return 0;
90  }

```

Listing 5.1.4: charlie_eugen2.cpp

```

1  /*
2   * Sursa 43p
3   * Complexitate: O(n*n) amortizat
4   * prof. Eugen Nodea
5   * Colegiul National "Tudor Vladimirescu", Tg-Jiu
6  */
7 # include <iostream>
8 # include <cstring>
9 # include <cassert>
10
11 using namespace std;
12
13 # define max(a, b) ((a) < (b) ? (b) : (a))
14
15 ifstream f("charlie.in");
16 ofstream g("charlie.out");
17
18 char s[1000100];
19 int p, L, ok, k, i, Max, j, mm;
20 long long S;
21
22 int main()
23 {
24     f >> p;
25     f.get();
26     f.getline(s, 1000100);
27     L = strlen(s);
28     if (p == 1) //a)
29     {
30         k = i = 0;
31         while ( i < L )
32         {
33             j = i;
34             while ( s[j] > s[j+1] && s[j+1] < s[j+2] && j + 2 < L)
35                 j += 2;
36             if (j - i >= 2)
37             {
38                 if ( j - i + 1 > Max ) Max = j - i + 1;
39                 i = j;
40             }
41             ++i;
42         }
43         g << Max << "\n";
44     }
45     else //b)
46     {
47         S = 0;
48         do
49         {
50             ok = 0;
51             //caut cel mai bun candidat
52             Max = 0; j = 0;
53             for(i=1; i<L-1; ++i)
54                 if (s[i-1] > s[i] && s[i] < s[i+1])
55                 {
56                     mm = max(s[i-1], s[i+1]);
57                     if (mm >= Max)
58                     {
59                         Max = mm;
60                         j = i;
61                     }
62                 }
63             if (Max > 0)
64             {
65                 strcpy(s+j, s+j+1);
66                 L--;
67                 ok = 1;
68                 S += (Max - 'a' + 1);
69             }
70         }while (ok);
71         g << s << "\n";
72         g << S << "\n";
73     }
74     return 0;

```

75 }

Listing 5.1.5: charlie_LilianaSchiopu.cpp

```

1 //prof. Liliana Schiopu - CNFB-Craiova
2 //100p
3 //complexitate O(n)
4 #include <cstdio>
5
6 using namespace std;
7
8 FILE *f=fopen("charlie.in","r");
9 FILE *g=fopen("charlie.out","w");
10
11 char a[100003],st[100003];
12 int i,j,n,p,c,lc,lmax,ok;
13
14 int Max(char a,char b,char c)
15 {
16     if(a>=b&&a>=c)
17         return int(a)-96;
18     else
19         if(b>=a&&b>=c)
20             return int(b)-96;
21         else
22             if(c>=b&&c>=a)
23                 return int(c)-96;
24 }
25
26 int main()
27 {
28     fscanf(f,"%d",&p);
29     if(p==1)
30     {
31         while((c = fgetc(f)) != EOF)
32         {
33             if (c <= 'z' &&c>='a') {
34                 n++;
35                 a[n]=c;
36             }
37         }
38         lmax=0;
39         for(i=1;i<=n;i++)
40         {
41             lc=0;
42             ok=1;
43             j=i+1;
44             while(j<n&&ok)
45                 if(a[j-1]>a[j]&&a[j]<a[j+1])
46                     {lc+=2;j+=2;}
47                 else {ok=0;}
48             }
49             if(lc>0) {lc++;i=j-1;}
50             if(lc>lmax)
51                 lmax=lc;
52         }
53         fprintf(g,"%d",lmax);
54     }
55     else
56     if(p==2)
57     {
58         lmax=0;
59         while((c = fgetc(f)) != EOF)
60         {
61             if (c <= 'z' &&c>='a') {
62                 n++;
63                 st[n]=c;
64                 do{ ok=0;
65                     if(n>2&&st[n-1]<st[n-2]&&st[n-1]<st[n])
66                     {
67                         ok=1;
68                         lmax+=Max(st[n-2],st[n-1],st[n]);
69                         st[n-1]=st[n];
70                     }
71                 }while(ok==1);
72             }
73         }
74     }
75 }

```

```

72         }
73     }
74     for(i=1;i<=n;i++)
75         fprintf(g,"%c",st[i]);
76     fprintf(g,"\n%d",lmax);
77 }
78 return 0;
79 }
```

Listing 5.1.6: charlie_marcel.cpp

```

1  /*
2   * Sursa 100p !!! (dar care sparge memoria)
3   * prof. Marcel Dragan, Sibiu
4 */
5 #include <iostream>
6 #include <cstring>
7 #include <stack>
8
9 using namespace std;
10
11 ifstream in("charlie.in");
12 ofstream out("charlie.out");
13
14 char s[100001];
15 int p;
16
17 void afisare(stack <char> st)
18 {
19     if(!st.empty())
20     {
21         char c=st.top();
22         st.pop();
23         afisare(st);
24         out << c;
25     }
26 }
27
28 int main()
29 {
30     in>>p>>s;
31     int n=strlen(s);
32     if(p==1)
33     {
34         int ctMax=0;
35         for(int i=0;i<n;i++)
36         {
37             // cauta dif<0
38             int dif1=s[i+1]-s[i];
39             while(dif1>=0 && i+2<n)
40             {
41                 i++;
42                 dif1=s[i+1]-s[i];
43             }
44             // numara dif<0 dif>0
45             int dif2=s[i+2]-s[i+1];
46             int ct=0;
47             while(dif1<0 && dif2>0 && i+2+ct< n)
48             {
49                 ct=ct+2;
50                 dif1=s[i+1+ct]-s[i+ct];
51                 dif2=s[i+2+ct]-s[i+1+ct];
52             }
53             // i=i+ct;
54             // verif max
55             if(ctMax<ct)
56                 ctMax=ct;
57         }
58         out<<ctMax+1<<endl;
59     }
60     else
61     {
62         stack <char> st;
63         st.push(s[0]);
64         char top=s[1];
```

```

65         int val=0;
66         for(int i=2;i<n;i++)
67         {
68             while(st.size()>0 && s[i] > top && top < st.top())
69             {
70                 val=val+max(s[i]-'a'+1,st.top()-'a'+1);
71                 top=st.top();
72                 st.pop();
73             }
74             st.push(top);
75             top=s[i];
76         }
77         afisare(st);
78         out << top<<endl;
79         out << val<<endl;
80     }
81     return 0;
82 }
```

Listing 5.1.7: charlie_radu_v.cpp

```

1  /*
2   * Sursa 38p - idee asemanatoare bouble-sort
3   * prof. Radu Visinescu, Ploiesti
4  */
5 #include <iostream>
6 #include <fstream>
7 #include <cstring>
8
9 using namespace std;
10
11 ifstream fin("charlie.in");
12 ofstream fout("charlie.out");
13
14 long long v,p,i,m=0;
15 long long p2,p3,u,save_p,save_u,d,dmax,ok;
16
17 char st[1000000];
18 char t[1000000];
19 char s[1000000];
20
21 int main()
22 {
23     fin>>p;fin>>s;
24     if (p==2){
25         do{ok=0;
26             i=0;      while(i<=strlen(s)-3)
27             if ((s[i]>s[i+1])&&(s[i+1]<s[i+2]))
28             {
29                 m=(int)(s[i])-(int)('a')+1;
30                 if (m<(int)(s[i+2])-(int)('a')+1)
31                     m=(int)(s[i+2])-(int)('a')+1;
32                 strcpy(t,s+i+2);
33                 strcpy(s+i+1,t);
34                 v += m; ok=1;
35             }
36             else i++;
37         }while(ok);
38         fout<<s<<'\'n';
39         fout<<v<<'\'n';
40     }
41     else {dmax=1;save_p=0;save_u=0;
42         p2=0;u=0;
43         d=1;
44         i=0;
45         while (i<=strlen(s)-3)
46         if ((s[i]>s[i+1]) && (s[i+1]<s[i+2]))
47         {
48             u=i+2;i=i+2;
49             d=u-p2+1;
50             if (dmax<d) {save_p=p2;save_u=u;dmax=save_u-save_p+1; }
51         }
52         else { i=i+2;p2=i;u=i; }
53         p2=1;u=1;
54         d=1;
```

```

55     i=1;
56     while (i<=strlen(s)-3)
57     {
58         if ((s[i]>s[i+1]) && (s[i+1]<s[i+2]))
59         {
60             u=i+2;i=i+2;
61             d=u-p2+1;
62             if (dmax<d) {save_p=p2;save_u=u;dmax=save_u-save_p+1; }
63         }
64         else { i=i+2;p2=i;u=i; }
65         fout<<save_u-save_p+1<<'\\n';
66     }
67 fin.close();
68 fout.close();
69 return 0;
}

```

Listing 5.1.8: charlie_SC.cpp

```

1  /*
2   * Sursa 100p
3   * prof. Stelian Ciurea, Sibiu
4  */
5 #include <iostream>
6 #include <fstream>
7 #include <vector>
8
9 using namespace std;
10
11 vector <int> a;
12
13 ifstream f("charlie.in");
14 ofstream g("charlie.out");
15
16 int tip,maxim=0;
17
18 int main()
19 {
20     //cout << "Hello world!" << endl;
21     f >> tip;
22     char buf;
23     while (f>>buf)
24     {
25         //cout << buf;
26         a.push_back(buf - 'a'+1);
27         maxim = max(maxim,buf - 'a'+1);
28     }
29     if (tip == 1)
30     {
31         int n = a.size();
32         a.push_back(a[n-1]);
33         a.push_back(a[n-1]);
34         a.push_back(a[n-1]);
35         a.push_back(a[n-1]);
36         int lmaxim = 0, lcrt=0;
37         for (int i=1;i<n;i+=2)
38         {
39             if (a[i]<a[i-1]&&a[i]<a[i+1])
40             {
41                 if (lcrt==0)
42                     lcrt=3;
43                 else
44                     lcrt +=2;
45                 //cout << i << ' ' << lcrt << " ";
46             }
47             else
48                 lcrt=0;
49             lmaxim = max(lmaxim,lcrt);
50         }
51         for (int i=2;i<n;i+=2)
52         {
53             if (a[i]<a[i-1]&&a[i]<a[i+1])
54             {
55                 if (lcrt==0)
56                     lcrt=3;
57                 else

```

```

58             lcrt +=2;
59             //cout << i << ' ' << lcrt << "    ";
60         }
61     else
62         lcrt=0;
63     lmaxim = max(lmaxim,lcrt);
64 }
65 cout << lmaxim << endl;
66 g << lmaxim << endl;
67 }
68 if (tip == 2)
69 {
70     int cost=0,i,j;
71     for (j=1;j<maxim;j++)
72     {
73         //cout << j << endl;
74         vector <int> b;
75         b.push_back(a[0]);
76         for (i=1;i<a.size()-1;i++)
77             if (a[i]==j&& a[i]<min(a[i-1],a[i+1]))
78                 cost +=max(a[i-1],a[i+1]);
79             else
80                 b.push_back(a[i]);
81         b.push_back(a[a.size()-1]);
82         a = b;
83     }
84     cout << cost << endl;
85     for (i=0;i<a.size();i++)
86         g << char('a'-1+a[i]);
87     g << endl << cost << endl;
88 }
89 return 0;
90 }
```

Listing 5.1.9: charlie_zoli.cpp

```

1 /*
2     Sursa 48p - O(n*n) amortizat
3     prof. Zoltan Szabo
4     isj. Mures
5 */
6 #include <fstream>
7 #include <cstring>
8
9 using namespace std;
10
11 ifstream fin("charlie.in");
12 ofstream fout("charlie.out");
13
14 int main()
15 {
16     int n,p,i,max,l,cresc;
17     char cuv[100001];
18     fin>>p;
19     max=1;
20     fin>>cuv;
21     n=strlen(cuv);
22     if (p==1)
23     {
24         cresc=0;
25         l=1;
26         for(i=1;i<n;++i)
27         {
28             if ((cuv[i-1]>cuv[i] and !cresc) or (cuv[i-1]<cuv[i] and cresc))
29             {
30                 l++;           // creste lungimea
31                 cresc=1-cresc; // schimbam conditia pentru relatia
32             }
33             else
34             {
35                 if (l%2==0) l--;
36                 if (l>max) max=l;
37                 if (cuv[i-1]>cuv[i])
38                 {
39                     l=2;
```

```

40             cresc=1;
41         }
42     else
43     {
44         l=1;
45         cresc=0;
46     }
47 }
48 if (l%2==0) l--;
49 if (l>max) max=l;
50
51     fout<<max<<"\n";
52 }
53 else
54 {
55     int cod,s=0;
56     do
57     {
58         cod=0;
59         for (i=1;i<strlen(cuv)-1;)
60         {
61             if (cuv[i]<cuv[i-1] and cuv[i]<cuv[i+1])
62             {
63                 if(cuv[i-1]>cuv[i+1])
64                     s=s+cuv[i-1]-'a'+1;
65                 else
66                     s=s+cuv[i+1]-'a'+1;
67                 strcpy(cuv+i,cuv+i+1);
68                 cod=1;
69             }
70             else
71                 ++i;
72         }
73     }
74     while (cod);
75     fout<<cuv<<"\n";
76     fout<<s<<"\n";
77 }
78 }
79 fout.close();
80 fin.close();
81 return 0;
82 }
```

5.2 panda

Problema 2 - panda

100 de puncte

O rezervație de urși panda, privită de sus, are formă dreptunghiulară și este compusă din n rânduri identice, iar pe fiecare rând sunt m țarcuri identice cu baza pătrată. Țarcurile sunt îngrădite și sunt prevăzute cu uși către toate cele 4 țarcuri vecine. Ușile sunt prevăzute cu câte un cod de acces, ca atare acestea se închid și se deschid automat. Prin acest sistem, unele țarcuri sunt accesibile ursuleților, iar altele le sunt interzise acestora. În T țarcuri se găsește mâncare pentru ursuleți.

Ursuleții din rezervație poartă câte un microcip care le deschide automat ușile țarcurilor unde pot intra și închide automat ușile țarcurilor interzise. Un țarc este **accesibil** ursulețului dacă ultimele S cifre ale reprezentărilor binare ale codului țarcului și ale codului k de pe microcip sunt complementare. (Exemplu: pentru $S = 8$, 11101011 și 00010100 sunt complementare).

Într-un țarc este un ursulet căruia i s-a făcut foame. Ursulețul se deplasează doar paralel cu laturile dreptunghiului. Trecerea dintr-un țarc în altul vecin cu el se face într-o secundă.

Cerințe

Cunoscând n și m dimensiunile rezervației, codurile de acces de la fiecare dintre cele $n * m$ țarcuri, coordonatele celor T țarcuri cu mâncare, coordonatele țarcului L și C unde se află inițial ursulețul, codul k al microcipului său și numărul S , determinați:

a) Numărul X de țarcuri care îndeplinesc proprietatea că ultimele S cifre din reprezentarea binară a codului lor sunt complementare cu ultimele S cifre din reprezentarea binară a codului k purtat de ursuleț, cu excepția țarcului în care se află acesta inițial.

b) Numărul minim de secunde S_{min} în care poate ajunge la un țarc cu mâncare precum și numărul de țarcuri cu mâncare nt la care poate ajunge în acest timp minim.

Date de intrare

Fișierul de intrare **panda.in** conține:

- pe prima linie un număr natural p . Pentru toate testele de intrare, numărul p poate avea doar valoarea 1 sau valoarea 2;
- pe a doua linie trei numere naturale n, m și T separate prin câte un spațiu, cu semnificațiile din enunț;
- pe linia a treia patru numere naturale nenule L, C, k și S , separate prin câte un spațiu, cu semnificațiile din enunț;
- pe următoarele T linii câte două numere naturale reprezentând coordonatele țarcurilor cu mâncare;
- pe următoarele n linii câte m numere naturale, separate prin câte un spațiu, reprezentând codurile de acces la ușile din cele $n * m$ țarcuri ale rezervației.

Date de ieșire

Dacă valoarea lui p este 1, se va rezolva numai punctul a) din cerință.

În acest caz, în fișierul de ieșire **panda.out** se va scrie un singur număr natural X , reprezentând numărul total de țarcuri pe care le poate accesa ursulețul, cu excepția țarcului în care se află acesta inițial.

Dacă valoarea lui p este 2, se va rezolva numai punctul b) din cerință.

În acest caz, fișierul de ieșire **panda.out** va conține numerele naturale naturale S_{min} și nt , în această ordine, separate printr-un spațiu.

Restricții și precizări

- $2 \leq n, m \leq 500$
- $1 \leq S \leq 8$
- $1 \leq T < n * m$
- $0 \leq k$, valorile codurilor ≤ 9999
- Pentru toate testele problemei există soluție, adică ursulețul poate ajunge la cel puțin unul dintre țarcurile cu mâncare.
 - Mâncarea se poate găsi și în zone inaccesibile.
 - Pentru rezolvarea corectă a primei cerințe se acordă 20 de puncte, iar pentru cerința a două se acordă 80 de puncte.
 - Pentru 24% dintre teste, se garantează $m \leq 50$ și $n \leq 50$.
 - Pentru 20% dintre teste, se garantează $S = 1$.
 - Pentru determinarea corectă a numărului S_{min} se acordă 75% din punctajul testului, iar pentru determinarea corectă a numărului nt se acordă 25% din punctajul testului.

Exemple

panda.in	panda.out	Explicații
<pre> 1 5 6 4 3 5 1 1 1 2 5 1 2 1 4 3 15 1278 3 1278 1278 1 16 17 18 19 254 20 21 25 26 254 254 254 27 28 29 3 2 254 2 254 4 254 254 254 </pre>	19	<p>$k = 1$ și deoarece $s = 1$ trebuie ca doar ultima cifră binară a lui k să fie diferită de ultima cifră binară a codului din țarc.</p> <p>Atenție! Pentru acest test se rezolvă doar cerința a).</p>

<pre> 2 5 6 4 3 5 1 1 1 2 5 1 2 1 4 3 15 1278 3 1278 1278 1 16 17 18 19 254 20 21 25 26 254 254 254 27 28 29 3 2 254 2 254 4 254 254 254 </pre>	<p>6 1</p>	<p>Dacă notăm cu 1 țarcurile accesibile și cu 0 cele inaccesibile, obținem următoarea matrice:</p> <table border="0" style="margin-left: auto; margin-right: auto;"> <tr><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr> </table> <p>Ursulețul se află în țarcul de coordonate (3, 5) și poate ajunge la un singur țarc cu mâncare, după 6 secunde. Acest țarc este cel de la coordonatele (5, 1); drumul parcurs este: $(3, 5) \rightarrow (4, 5) \rightarrow (5, 5) \rightarrow (5, 4) \rightarrow (5, 3) \rightarrow (5, 2) \rightarrow (5, 1)$</p> <p>Atenție! Pentru acest test se rezolvă doar cerința b).</p>	0	1	0	1	1	0	1	0	1	0	1	1	0	0	1	1	1	1	0	1	0	0	1	1	1	1	1	1	1	1
0	1	0	1	1	0																											
1	0	1	0	1	1																											
0	0	1	1	1	1																											
0	1	0	0	1	1																											
1	1	1	1	1	1																											

Timp maxim de executare/test: **0.5** secunde

Memorie: total **8 MB** din care pentru stivă **2 MB**

Dimensiune maximă a sursei: **10 KB**

5.2.1 Indicații de rezolvare

Simulescu Adriana, Liceul Teoretic GRIGORE MOISIL Timișoara

Se construiește o matrice b cu elemente $b[i][j] = -1$ pentru țarcurile inaccesibile și $b[i][j] = 0$ pentru țarcurile accesibile. Pentru a determina țarcurile accesibile, se verifică condiția:

$$(a_{i,j} \mod 2^s) \text{ xor } (k \mod 2^s) = 2^s - 1$$

Pentru cerința a), se numără elementele egale cu 0 din matrice.

Pentru cerința b), se rețin coordonatele țarcurilor cu mâncare într-un vector. Se parcurge matricea b de la coordonatele ursulețului cu un *algoritm de tip LEE*, reținând în elementul $b[i][j]$ numărul minim de secunde necesar pentru a ajunge la țarcul de coordonate (i, j) .

Se determină distanța minimă la țarcurile accesibile precum și numărul de țarcuri aflat la această distanță minimă.

5.2.2 *Rezolvare detaliată

5.2.3 Cod sursă

Listing 5.2.1: panda_adriana.cpp

```

1 /*
2  Sursa 100 p - Lee + biti
3  Complexitate: O(n*m)
4  Adriana Simulescu
5  Liceul Teoretic GRIGORE MOISIL Timisoara
6 */
7 #include<fstream>
8 #include<iostream>
9
10 #define Nmax 500
11
12 using namespace std;
13
14 ifstream in("panda.in");
15 ofstream out("panda.out");
16
17 int a[Nmax+2][Nmax+2], b[Nmax+2][Nmax+2];
18 int n, m, kod, t, l, c, r, d_min=320000, imin, p, s;
19
20 struct tarc{unsigned l, c;};
21
22 tarc coada[Nmax*Nmax], papa[Nmax*Nmax];
23

```

```

24 void citire()
25 {int i,j,x,y;
26 in>>p;
27 in>>n>>m>>t>>l>>c>>kod>>s;
28 for(i=1;i<=t;i++)
29     (in>>papa[i].l>>papa[i].c);
30 for(i=1;i<=n;i++)
31     for(j=1;j<=m;j++)
32         in>>a[i][j];
33 in.close();
34 }
35
36 void lee(int l,int c)
37 {
38 int st=1,dr=1,ll,cc,pas;
39 ll=l;
40 cc=c;
41 b[ll][cc]=10;
42 coada[dr].l=l;
43 coada[dr].c=c;
44 while(st<=dr)
45 {
46 ll=coada[st].l;
47 cc=coada[st].c;
48 pas=b[ll][cc];
49 if(ll-1>0&&(b[ll-1][cc]==0||b[ll-1][cc]>pas+1))
50     {b[ll-1][cc]=pas+1;
51      dr++;
52      coada[dr].l=ll-1;
53      coada[dr].c=cc;
54     }
55
56 if(ll+1<=n&&(b[ll+1][cc]==0||b[ll+1][cc]>pas+1))
57     {b[ll+1][cc]=pas+1;
58      dr++;
59      coada[dr].l=ll+1;
60      coada[dr].c=cc;
61     }
62
63 if(cc-1>0&&(b[ll][cc-1]==0||b[ll][cc-1]>pas+1))
64     {b[ll][cc-1]=pas+1;
65      dr++;
66      coada[dr].l=ll;
67      coada[dr].c=cc-1;
68     }
69
70 if(cc+1<=m&&(b[ll][cc+1]==0||b[ll][cc+1]>pas+1))
71     {b[ll][cc+1]=pas+1;
72      dr++;
73      coada[dr].l=ll;
74      coada[dr].c=cc+1;
75     }
76
77 st++;
78 }
79
80 }
81
82
83
84 void construire()
85 {int i,j,putere2=1;
86 for(i=1;i<=s;i++)
87     putere2*=2;
88 for(i=1;i<=n;i++)
89     for(j=1;j<=m;j++)
90         {if(((a[i][j]%putere2)^(kod%putere2))&(putere2-1))==putere2-1)
91          b[i][j]=0;
92          else b[i][j]=-1;
93         }
94     }
95
96 int main()
97 {
98 int i,j,nt=0,nrt=0;
99 citire();

```

```

100 construire();
101
102 if(p==1)
103 {   for(i=1;i<=n;i++)
104     for(j=1;j<=m;j++)
105       if(b[i][j]==0)
106         nrt++;
107   cout<<b[1][c];
108   out<<nrt-1<<' \n';
109 }
110 else
111 {
112   b[1][c]=10;
113   lee(l,c);
114   for (i=1;i<=t;i++)
115     if(b[papa[i].l][papa[i].c]>1&&b[papa[i].l][papa[i].c]<d_min)
116       {d_min=b[papa[i].l][papa[i].c];
117      }
118   for(i=1;i<=t;i++)
119     if(b[papa[i].l][papa[i].c]==d_min)
120       {nt++;
121      }
122   out<<d_min-10<<' ' <<nt<<' \n';
123 }
124 out.close();
125 }
```

Listing 5.2.2: panda_eugen0.cpp

```

1 /*
2   Sursa 100 p Lee+bit
3   prof. Eugen Nodea, Tg-Jiu
4 */
5 # include <cstdio>
6 # include <queue>
7
8 # define inf 1000000
9
10 using namespace std;
11
12 const int dx[] = {0, 1, 0, -1};
13 const int dy[] = {1, 0, -1, 0};
14
15 int p, n, m, t, k, l, c, i, j, x, y, s, Min = inf, nr = -1;
16 bool key[501][501], food[501][501];
17 int T[501][501];
18 short p2[]={1, 2, 4, 8, 16, 32, 64, 128, 256, 1024};
19
20 struct cel
21 {
22   short l, c;
23 };
24
25 queue < cel > q;
26
27 void lee()
28 {
29   cel x, y;
30   int i;
31   x.l = l; x.c = c;
32   q.push(x); T[l][c] = 0;
33   while (!q.empty())
34   {
35     x = q.front(); q.pop();
36     for(i=0; i<4; ++i)
37     {
38       y.l = x.l + dx[i];
39       y.c = x.c + dy[i];
40       if (key[y.l][y.c] && T[y.l][y.c] > T[x.l][x.c] + 1)
41       {
42         q.push(y);
43         T[y.l][y.c] = T[x.l][x.c] + 1;
44         if (food[y.l][y.c])
45         {
46           if (Min > T[y.l][y.c]) Min = T[y.l][y.c];
47         }
48       }
49     }
50   }
51 }
```

```

47         }
48     }
49   }
50 }
51 }
52 }
53 int main()
54 {
55     freopen("panda.in", "r", stdin);
56     freopen("panda.out", "w", stdout);
57
58     scanf("%d%d%d%d%d%d", &p, &n, &m, &t, &l, &c, &k, &s);
59
60     k = k % p2[s];
61
62     for(i=1; i<=t; ++i)
63     {
64         scanf("%hd %hd", &x, &y);
65         food[x][y] = 1;
66     }
67
68     for(i=1; i<=n; ++i)
69     {
70         for(j=1; j<=m; ++j)
71         {
72             T[i][j] = inf;
73
74             scanf("%d", &x);
75
76             x = x % p2[s];
77             key[i][j] = ((x ^ k) == (p2[s] - 1));
78             nr += key[i][j];
79         }
80     }
81
82     lee();
83
84     if (p == 1)
85     {
86         printf("%d\n", nr);
87     }
88     else
89     {
90         printf("%d ", Min);
91         for(i=1, nr = 0; i<=n; ++i)
92             for(j=1; j<=m; ++j)
93                 if (food[i][j] && T[i][j] == Min) ++nr;
94         printf("%d\n", nr);
95     }
96     return 0;
97 }
```

Listing 5.2.3: panda_eugen1.cpp

```

1  /*
2   * Sursa 100 p Lee+bit
3   * prof. Eugen Nodea, Tg-Jiu
4  */
5 # include <iostream>
6 # include <queue>
7
8 # define inf 1000000
9
10 using namespace std;
11
12 ifstream f("panda.in");
13 ofstream g("panda.out");
14
15 const int dx[] = {0, 1, 0, -1};
16 const int dy[] = {1, 0, -1, 0};
17
18 int p, n, m, t, k, l, c, i, j, x, y, s, Min = inf, nr = -1;
19 bool key[501][501], food[501][501];
20 int T[501][501];
21 short p2[]={1, 2, 4, 8, 16, 32, 64, 128, 256, 1024};
22
23 struct cel
```

```

24  {
25      short l, c;
26  };
27
28 queue < cel > q;
29
30 void lee()
31 {
32     cel x, y;
33     int i;
34     x.l = l; x.c = c;
35     q.push(x); T[l][c] = 0;
36     while (!q.empty())
37     {
38         x = q.front(); q.pop();
39         for(i=0; i<4; ++i)
40         {
41             y.l = x.l + dx[i];
42             y.c = x.c + dy[i];
43             if (key[y.l][y.c] && T[y.l][y.c] > T[x.l][x.c] + 1)
44             {
45                 q.push(y);
46                 T[y.l][y.c] = T[x.l][x.c] + 1;
47                 if (food[y.l][y.c])
48                 {
49                     if (Min > T[y.l][y.c]) Min = T[y.l][y.c];
50                 }
51             }
52         }
53     }
54 }
55
56 int main()
57 {
58     f >> p;
59     f >> n >> m >> t;
60     f >> l >> c >> k >> s;
61
62     k = k % p2[s];
63
64     for(i=1; i<=t; ++i)
65     {
66         f >> x >> y;
67         food[x][y] = 1;
68     }
69
70     for(i=1; i<=n; ++i)
71     {
72         for(j=1; j<=m; ++j)
73         {
74             T[i][j] = inf;
75             f >> x;
76             x = x % p2[s];
77             key[i][j] = ((x ^ k) == (p2[s] - 1));
78         }
79     }
80     lee();
81
82     if (p == 1)
83     {
84         g << nr << "\n";
85     }
86     else
87     {
88         if (inf == Min)
89         {
90             g << "Nu am solutie";
91             return 0;
92         }
93         g << Min << " ";
94         for(i=1, nr = 0; i<=n; ++i)
95         {
96             for(j=1; j<=m; ++j)
97                 if (food[i][j] && T[i][j] == Min) ++nr;
98             g << nr << "\n";
99         }
100    return 0;

```

100 }

Listing 5.2.4: panda_eugen2.cpp

```

1 /*
2     Sursa 100 p Lee + verif. compl.
3     prof. Eugen Nodea, Tg-Jiu
4 */
5 # include <fstream>
6 # include <queue>
7
8 # define inf 1000000
9
10 using namespace std;
11
12 ifstream f("panda.in");
13 ofstream g("panda.out");
14
15 const int dx[] = {0, 1, 0, -1};
16 const int dy[] = {1, 0, -1, 0};
17
18 int p, n, m, t, k, l, c, i, j, x, y, s, Min = inf, nr = -1;
19 bool key[501][501], food[501][501];
20 int T[501][501];
21
22 struct cel
23 {
24     short l, c;
25 };
26
27 queue < cel > q;
28
29 void lee()
30 {
31     cel x, y;
32     int i;
33     x.l = l;
34     x.c = c;
35     q.push(x);
36     T[l][c] = 0;
37     while (!q.empty())
38     {
39         x = q.front();
40         q.pop();
41         for(i=0; i<4; ++i)
42         {
43             y.l = x.l + dx[i];
44             y.c = x.c + dy[i];
45             if (key[y.l][y.c] && T[y.l][y.c] > T[x.l][x.c] + 1)
46             {
47                 q.push(y);
48                 T[y.l][y.c] = T[x.l][x.c] + 1;
49                 if (food[y.l][y.c])
50                     if (Min > T[y.l][y.c])
51                         Min = T[y.l][y.c];
52             }
53         }
54     }
55 }
56
57 bool verif(int x, int y, int s)
58 {
59     for (int i=1; i<=s; ++i)
60     {
61         if (x % 2 == y % 2) return 0;
62         x /= 2; y /= 2;
63     }
64     return 1;
65 }
66
67 int main()
68 {
69     f >> p;
70     f >> n >> m >> t;
71     f >> l >> c >> k >> s;

```

```

72
73     for(i=1; i<=t; ++i)
74     {
75         f >> x >> y;
76         food[x][y] = 1;
77     }
78
79     for(i=1; i<=n; ++i)
80     {
81         for(j=1; j<=m; ++j)
82         {
83             T[i][j] = inf;
84             f >> x;
85             key[i][j] = verif(k, x, s);
86             nr += key[i][j];
87         }
88     }
89
90     if (p == 1)
91         g << nr << "\n";
92     else
93     {
94         if (inf == Min)
95         {
96             g << "Nu am solutie!";
97             return 0;
98         }
99         g << Min << " ";
100        for(i=1, nr = 0; i<=n; ++i)
101            for(j=1; j<=m; ++j)
102                if (food[i][j] && T[i][j] == Min) ++nr;
103        g << nr << "\n";
104    }
105    return 0;
106}

```

Listing 5.2.5: panda_Liliana_Schiopu.cpp

```

1 // prof. Liliana Schiopu - C.N.F.B., Craiova
2 // complexitate O(nxm)
3 // algoritmul Lee
4 #include <stdio.h>
5
6 using namespace std;
7
8 FILE *f=fopen("panda.in","r");
9 FILE *g=fopen("panda.out","w");
10
11 int n,m,t,i,j,p,a[510][510],b[510][510],x,y,viz[510][510];
12 int l,c,k,s,nr,coada[3][250001],ic,sc,timp[250001];
13 int i1[5]={-1,0,1,0},j1[5]={0,1,0,-1};
14 int smin=300000,nt;
15
16 int acc(int x1,int x2)
17 {
18     int i=s;
19     while(i>0)
20     {
21         if((x1%2)^(x2%2)==1)
22         {
23             x1/=2;
24             x2/=2;
25             i--;
26         }
27         else return 0;
28     }
29     return 1;
30 }
31
32 int main()
33 {
34     fscanf(f,"%d",&p);
35     fscanf(f,"%d%d%d",&n,&m,&t);
36     fscanf(f,"%d%d%d%d",&l,&c,&k,&s);
37

```

```

38     for(i=1;i<=t;i++)
39     {
40         fscanf(f,"%d%d",&x,&y);
41         b[x][y]=1;
42     }
43
44     for(i=1;i<=n;i++)
45     {
46         for(j=1;j<=m;j++)
47             fscanf(f,"%d",&a[i][j]);
48
49     if(p==1)
50     {
51         for(i=1;i<=n;i++)
52             for(j=1;j<=m;j++)
53                 if(i!=l||j!=c)
54                     if(acc(a[i][j],k))
55                         nr++;
56         fprintf(g,"%d",nr);
57     }
58     else
59     if(p==2)
60     {
61         ic=1;sc=1;
62         coada[1][ic]=l;
63         coada[2][ic]=c;
64         viz[coada[1][ic]][coada[2][ic]]=1;
65         timp[ic]=0;
66         while(ic<=sc)
67         {
68             for(i=0;i<=3;i++)
69                 if(acc(a[coada[1][ic]+i][coada[2][ic]+j],k)&&
70                     a[coada[1][ic]+i][coada[2][ic]+j]!=0)
71                     if(viz[coada[1][ic]+i][coada[2][ic]+j]==0)
72                     {
73                         sc++;
74                         coada[1][sc]=coada[1][ic]+i;
75                         coada[2][sc]=coada[2][ic]+j;
76                         viz[coada[1][sc]][coada[2][sc]]=1;
77                         timp[sc]=timp[ic]+1;
78                         if(b[coada[1][sc]][coada[2][sc]]==1)
79                         {
80                             if(smin>timp[sc])
81                             {
82                                 smin=timp[sc];
83                                 nt=1;
84                             }
85                             else
86                             {
87                                 if(smin==timp[sc])
88                                 {
89                                     nt++;
90                                 }
91                             }
92                         }
93                         fprintf(g,"%d %d",smin,nt);
94                     }
95         fclose(f);
96         fclose(g);
97         return 0;
98     }

```

Listing 5.2.6: panda_marcel.cpp

```

1  /*
2   * Sursa 100 p
3   * prof. Marcel Dragan, Sibiu
4   */
5 #include <iostream>
6 #include <queue>
7 #include <cstring>
8
9 using namespace std;
10
11 ifstream in("panda.in");

```

```

12  ofstream out("panda.out");
13
14  int p,n,m,t,l,c,s,h[501][501],lee[501][501],x;
15
16 void citire()
17 {
18     in>>p>>n>>m>>t;
19     in>>l>>c>>k>>s;
20     for(int i=1;i<=n;i++)
21     {
22         for(int j=1;j<=m;j++)
23         {
24             h[i][j]=0;
25         }
26     }
27     int xt,yt;
28     for(int i=1;i<=t;i++)
29     {
30         in>>xt>>yt;
31         h[xt][yt]=2000000;
32     }
33     int cod=1;
34     for(int i=1;i<=s;i++)
35         cod=cod*2;
36     for(int i=1;i<=n;i++)
37     {
38         for(int j=1;j<=m;j++)
39         {
40             in>>lee[i][j];
41             if(lee[i][j]%cod+k%cod==cod-1)
42                 lee[i][j]=1000000;
43             else
44                 lee[i][j]=-1;
45         }
46         int stop;
47         if(i==24)
48             stop=1;
49     }
50 }
51
52 void afis(int matr[501][501])
53 {
54     int i,j;
55     for(i=1;i<=n;i++)
56     {
57         for(j=1;j<=m;j++)
58         {
59             if(matr[i][j]<100)
60                 out<<matr[i][j]<<' \t'<<' \t';
61             else
62                 out<<matr[i][j]<<' \t';
63         }
64         out<<' \n';
65     }
66     out<<' \n';
67 }
68
69 int parurgereLee()
70 {
71     queue <int> lin,col;
72
73     lee[1][c]=1;
74     lin.push(1);
75     col.push(c);
76     int lu,cu,minT=1000000;
77
78     while(!lin.empty())
79     {
80         int lc=lin.front();
81         int cc=col.front();
82         lin.pop();
83         col.pop();
84         for(int i=-1;i<=1;i++)
85             for(int j=-1;j<=1;j++)
86             {
87                 lu=lc+i;

```

```

88         cu=cc+j;
89         if(i*i!=j*j && 1<=lu && lu<=n && 1<=cu && cu<=m)
90         {
91             int pas=1;
92             if(lee[lu][cu]!=-1 && lee[lc][cc]+1<lee[lu][cu])
93             {
94                 if(lee[lu][cu]>n*m)
95                     x++;
96                 lin.push(lu);
97                 col.push(cu);
98                 lee[lu][cu]=lee[lc][cc]+pas;
99                 if(minT>lee[lu][cu] && h[lu][cu]==2000000)
100                {
101                    minT=lee[lu][cu];
102                }
103            }
104        }
105    }
106 //    afis(lee);
107 }
108 //    afis(lee);
109 //    afis(h);
110 return minT;
111 }
112
113 void afis(int minT)
114 {
115     int gasit=0,total=0,total1=0,total2=0;
116     for(int i=1;i<=n;i++)
117     {
118         for(int j=1;j<=m;j++)
119         {
120             if(lee[i][j]==minT && h[i][j]==2000000)
121                 gasit++;
122             if(1<lee[i][j] && lee[i][j]<1000000)
123                 total++;
124             if(lee[i][j]==1000000)
125                 total1++;
126             if(lee[i][j]==-1)
127                 total2++;
128         }
129     }
130     if(p==1)
131 //        out << x << '\n';
132     out << n*m-total2-1 << '\n';
133     else
134         out << minT-1 << ' ' << gasit << '\n';
135 }
136
137 int main()
138 {
139     citire();
140     int minT=parurgereLee();
141     afis(minT);
142     return 0;
143 }
```

Listing 5.2.7: panda_radu.cpp

```

1 /*
2  * Sursa 100p
3  * prof. Radu Visinescu, Ploiesti
4 */
5 #include <iostream>
6 #include <fstream>
7
8 using namespace std;
9
10 ifstream fin("panda.in");
11 ofstream fout("panda.out");
12
13 int n,m,t,a[501][501],lungime[501][501],tarc[501][501];
14 int l,c,k,s;
15 int p;
```

```

17 int ok(int n,int k,int s)
18 {int vn[10],kn[10],p,o,modulo,i;
19     modulo=1;
20     for(i=1;i<=s;i++) modulo=modulo*2;
21     n=n%modulo;k=k%modulo;
22     p=0;
23     while (p<s) {p++;vn[p]=n%2;n=n/2;}
24     p=0;
25     while (p<s) {p++;kn[p]=k%2;k=k/2;}
26     o=1;
27     for(p=1;p<=s;p++)
28         if (!((vn[p]||kn[p])&&((vn[p]&&kn[p])==0)))
29             o=0;
30     return o;
31 }
32
33 void lee()
34 { int q[2][250000],prim,ultim,i,j;
35 for(i=1;i<=n;i++)
36     for(j=1;j<=m;j++)
37         lungime[i][j]=-1;
38 prim=ultim=0;
39 ultim++;q[0][ultim]=1;q[1][ultim]=c;lungime[1][c]=0;
40 while((prim<ultim))
41 {prim++;i=q[0][prim];j=q[1][prim];
42     if ((i>0)&&ok(a[i-1][j],k,s))
43         {if (lungime[i-1][j]==-1){lungime[i-1][j]=lungime[i][j]+1;
44             ultim++;q[0][ultim]=i-1;q[1][ultim]=j;}}
45     if ((i<n)&&ok(a[i+1][j],k,s))
46         {if (lungime[i+1][j]==-1){lungime[i+1][j]=lungime[i][j]+1;
47             ultim++;q[0][ultim]=i+1;q[1][ultim]=j;}}
48     if ((j>0)&&ok(a[i][j-1],k,s))
49         {if (lungime[i][j-1]==-1){lungime[i][j-1]=lungime[i][j]+1;
50             ultim++;q[0][ultim]=i;q[1][ultim]=j-1;}}
51     if ((j<m)&&ok(a[i][j+1],k,s))
52         {if (lungime[i][j+1]==-1){lungime[i][j+1]=lungime[i][j]+1;
53             ultim++;q[0][ultim]=i;q[1][ultim]=j+1;}}
54 }
55 }
56
57 int main()
58 {long long i,j,x,y,mi,nr;
59 fin>>p;
60 fin>>n>>m>>t;
61 fin>>l>>c>>k>>s;;
62 for (i=1;i<=t;i++)
63     {fin>>x>>y;tarc[x][y]=1;}
64 for (i=1;i<=n;i++)
65     for(j=1;j<=m;j++)
66         fin>>a[i][j];
67 lee();
68 mi=1000000;
69     for(i=1;i<=n;i++)
70         for(j=1;j<=m;j++)
71             if(tarc[i][j]==1 && mi>lungime[i][j] &&
72                 lungime[i][j]!=-1)mi=lungime[i][j];
73 if(p==1){nr=0;
74     for(i=1;i<=n;i++)
75         for(j=1;j<=m;j++)
76             if((i!=l)|| (j!=c)) if (ok(a[i][j],k,s)) nr++;
77             fout<<nr<<'\\n';
78 else {nr=0;
79     for(i=1;i<=n;i++)
80         for(j=1;j<=m;j++)
81             if(tarc[i][j]==1 && mi==lungime[i][j])nr++;
82             fout<<mi<<" "<<nr<<'\\n';
83 fin.close();
84 fout.close();
85     return 0;
86 }

```

Listing 5.2.8: panda_zoli1.cpp

```

1 /*
2  Sursa 100p

```

```

3     prof. Zoltan Szabo
4     isj Mures
5 */
6 #include <fstream>
7
8 using namespace std;
9
10 ifstream fin("panda.in");
11 ofstream fout("panda.out");
12
13 int b[501][501],c[501][501],coada[3][250000];
14
15 int main()
16 {
17     int timp,nr,prim,ultim,ultiml,i,j,l,c,k,x,y,p,n,m,t,a[501][501],gata,
18         acces,lin,col,cheie,timpmin,s,ok;
19     fin>>p;
20     fin>>n>>m>>t>>l>>c>>k>>s;
21
22     for(i=1;i<=t;++i)
23     {
24         fin>>x>>y;
25         b[x][y]=1;      //coordonatele hranei
26     }
27     cheie=1;
28     for (i=1;i<=s;++i)
29         cheie=cheie*2; // cheia pentru ultimele cifre binare
30     acces=0;
31     for(i=1;i<=n;++i)
32         for (j=1;j<=m;++j)
33         {
34             fin>>x;
35             a[i][j]=(((x%cheie) xor (k%cheie)) == cheie-1); // conditia ca
36                                         // tarcul sa se deschida
37             acces+=a[i][j]; // adunam casutele cu valori accesibile
38         }
39     acces-=a[1][c];    // in cazul in care ursuletul este pe o coordonata
40                         // accesibila, scadem, ca nu trebuie numarat
41     if (p==1)      // daca e cerinta p==1 atunci tiparim si ne oprim
42     {
43         fout<<acces<<"\n";
44         fout.close();
45         fin.close();
46         return 0;
47     }
48
49 // cazul p=2
50 coada[0][0]=1;    // coada[0] reprezinta linia
51 coada[1][0]=c;    // coada[1] reprezinta coloana
52 coada[2][0]=0;    // timpul memorat in coada, pornim de la timpul 0
53 prim=0;ultim=0;   // secenta din coada care se va prelucra cu Lee,
54                         // generand o alta secenta
55 gata=0;          // dcand se va ajunge la hrana, "gata" va fi 1
56 timpmin=300000;
57 ok=0;
58 b[1][c]=2;
59 nr=0;
60 while (!gata)
61 {
62     ultiml=ultim;
63     for(i=prim;i<=ultiml;++i)
64     {
65         lin=coada[0][i];
66         col=coada[1][i];
67         timp=coada[2][i];
68         if (col>1 and a[lin][col-1] and
69             b[lin][col-1]!=2) // tarcul nevizitat la stanga
70         {
71             coada[0][++ultim]=lin; // introducem coordonata tarcului
72                                         // in coada
73             coada[1][ultim]=col-1;
74             coada[2][ultim]=timp+1; // daca valoarea timpului
75                                         // este >0 atunci este fara salt
76
77             if (b[lin][col-1]==1)    // daca s-a ajuns la hrana
78             {

```

```

79         gata=1;                                // atunci ne pregatim de finish
80         nr++;
81         b[lin][col-1]=2;                      // numaram inca o solutie
82                                         // cu acelasi timp minim
83                                         // marcam tarcul ca si vizitat,
84                                         // sa nu numaram inca o data
85     }
86     else
87         b[lin][col-1]=2;                      // marcam tarcul ca si loc vizitat,
88                                         // sa nu trecem inca o data
89 }
90 if (col<m and a[lin][col+1] and
91     b[lin][col+1]!=2) // tarc nevizitat la dreapta
92 {
93     coada[0][++ultim]=lin;                  // introducem coordonata tarcului
94                                         // in coada
95     coada[1][ultim]=col+1;                  // daca valoarea timpului este >0
96     coada[2][ultim]=timp+1;                  // atunci este fara salt
97
98     if (b[lin][col+1]==1)                   // daca s-a ajuns la hrana
99     {
100        gata=1;                            // atunci ne pregatim de finish
101        nr++;
102        b[lin][col+1]=2;                  // numaram inca o solutie cu
103                                         // acelasi timp minim
104                                         // marcam tarcul ca si vizitat,
105                                         // sa nu numaram inca o data
106    }
107    else
108        b[lin][col+1]=2;                  // marcam tarcul ca si loc vizitat,
109                                         // sa nu trecem inca o data
110 }
111 if (lin>1 and a[lin-1][col] and
112     b[lin-1][col]!=2) // tarcul de sus nevizitat
113 {
114     coada[0][++ultim]=lin-1;              // introducem coordonata tarcului
115                                         // in coada
116     coada[1][ultim]=col;                  // daca valoarea timpului este >0
117     coada[2][ultim]=timp+1;                  // atunci este fara salt
118
119     if (b[lin-1][col]==1)                   // daca s-a ajuns la hrana
120     {
121        gata=1;                            // atunci ne pregatim de finish
122        nr++;
123        b[lin-1][col]=2;                  // numaram inca o solutie cu acelasi
124                                         // timp minim
125        b[lin-1][col]=2;                  // marcam tarcul ca si vizitat,
126                                         // sa nu numaram inca o data
127    }
128    else
129        b[lin-1][col]=2;                  // marcam tarcul ca si loc vizitat,
130                                         // sa nu trecem inca o data
131 }
132 if (lin<n and a[lin+1][col] and
133     b[lin+1][col]!=2) // tarcul de jos nevizitat
134 {
135     coada[0][++ultim]=lin+1;              // introducem coordonata tarcului
136                                         // in coada
137     coada[1][ultim]=col;                  // daca valoarea timpului este >0
138     coada[2][ultim]=timp+1;                  // atunci este fara salt
139
140     if (b[lin+1][col]==1)                   // daca s-a ajuns la hrana
141     {
142        gata=1;                            // atunci ne pregatim de finish
143        nr++;
144        b[lin+1][col]=2;                  // numaram inca o solutie cu
145                                         // acelasi timp minim
146        b[lin+1][col]=2;                  // marcam tarcul ca si vizitat,
147                                         // sa nu numaram inca o data
148    }
149    else
150        b[lin+1][col]=2;                  // marcam tarcul ca si loc vizitat,
151                                         // sa nu trecem inca o data
152
153 }
154 }
```

```
155         prim=ultim1+1;
156     }
157
158     fout<<timp+1<<" " <<nr<<"\n";
159     fout.close();
160     return 0;
161 }
```

Capitolul 6

OJI 2014

6.1 ferma

Problema 1 - ferma

Un fermier deține o fermă de formă dreptunghiulară cu lungimea m metri și lățimea n metri. Respectând principiul rotației culturilor, fermierul și-a realizat un plan pentru semănarea culturilor în noul an. Astfel, el a desenat un dreptunghi pe care l-a împărțit în $m * n$ celule, fiecare corespunzând unui metru pătrat, și a colorat în culori diferite zonele care corespund unor culturi diferite. O cultură poate fi semănată pe mai multe parcele. Două celule care au o latură comună aparțin aceleiași parcele dacă au aceeași culoare (sunt însămânțate cu aceeași cultură). Fermierul are posibilitatea să irige o sigură parcelă și dorește să aleagă parcela cu cea mai mare suprafață. Nefind mulțumit de suprafața rezultată, s-a întrebat dacă ar putea schimba cultura de pe o singură celulă, astfel încât să obțină o parcelă de suprafață mai mare.

Cerințe

Dându-se dimensiunile fermei și pentru fiecare celulă culoarea corespunzătoare culturii semănate, determinați:

Varianta 1: Suprafața maximă a unei parcele în planul inițial.

Varianta 2: Numărul liniei, respectiv al coloanei celulei pe care va semăna o altă cultură și culoarea corespunzătoare noii culturi în vederea obținerii celei mai mari parcele posibile.

Date de intrare

Fișierul de intrare **ferma.in** va conține:

- pe prima linie un număr natural v ($1 \leq v \leq 2$) indicând varianta cerinței de rezolvare;
- pe a doua linie două numere naturale m și n separate printr-un spațiu, cu semnificația din enunț;
- pe fiecare dintre următoarele m linii se găsesc câte n caractere (litere mici), reprezentând codurile culturilor ce vor fi semănate pe cele n celule corespunzătoare fiecărei linii.

Date de ieșire

Fișierul de ieșire **ferma.out** va conține:

Varianta 1 - pentru $v = 1$:

- pe prima linie numărul natural s , reprezentând suprafața maximă a unei parcele.

Varianta 2 - pentru $v = 2$:

- pe prima linie două numere naturale separate printr-un spațiu, reprezentând numărul liniei, respectiv al coloanei celulei pe care va semăna o altă cultură, în vederea obținerii unei parcele cu suprafață maximă;
- pe a doua linie un caracter reprezentând codul culorii corespunzătoare noii culturi din celula determinată.

Restricții și precizări

100 de puncte
Imagine 1

r	m	m	g	g	g	a	a
m	v	v	g	g	g	a	a
m	v	v	g	v	v	v	v
v	v	v	r	v	v	v	v
v	v	r	r	r	g	g	a
v	v	r	r	r	g	g	g
a	a	a	a	a	a	a	g

Figura 6.1: ferma

- $2 \leq m \leq 400$
- $2 \leq n \leq 400$
- Numărul de culturi distincte este cel puțin 2 și cel mult 26.
- 30% din teste vor avea pe prima linie valoarea 1, iar restul de 70% din teste vor avea pe prima linie valoarea 2.
- Pentru varianta 2 se punctează orice soluție care conduce la obținerea unei parcele cu suprafață maximă. Nu se acordă punctaje parțiale.

Exemplu

ferma.in	ferma.out	Explicații
1 7 8 rmmggaa mvvggaa mvvgvvv vvvrvvvv vvrrrgga vvrrrggg aaaaaaag	11	<p>Datele corespund imaginilor de mai sus. Numerotarea parcelelor din imaginea 2 este utilizată pentru a simplifica explicațiile de mai jos și nu influențează datele problemei și nici algoritmul de rezolvare.</p> <p>În varianta 1 se determină și se afișează suprafață maximă a unei parcele, care este egală cu 11 și corespunde parcelei 6, de culoare verde (codificată cu litera v în imaginea 1 și în fișierul de intrare).</p>
2 7 8 rmmggaa mvvggaa mvvgvvv vvvrvvvv vvrrrgga vvrrrggg aaaaaaag	3 4 v	<p>Pentru varianta 2:</p> <p>Schimbând în verde (v) culoarea celulei de pe linia 3 și coloana 4, se obține o parcelă cu suprafață $11 + 8 + 1 = 20$ (se unesc parcelele cu numărul 6 respectiv 8).</p> <p>O altă soluție corectă este:</p> <p>4 4 v</p>

Timp maxim de executare/test: **0.2** secunde

Memorie: total **32 MB** din care pentru stivă **10 MB**

Dimensiune maximă a sursei: **10 KB**

6.1.1 Indicații de rezolvare

Prof. Florentina Ungureanu - Colegiul Național de Informatică Piatra-Neamț

Varianta 1:

Utilizând un *algoritm de umplere*, se determină și se reține într-un vector suprafața fiecărei parcele, în ordinea determinării lor, aflându-se dimensiunea maximă a unei parcele.

Varianta 2:

Se continuă operațiile de la prima variantă cu căutarea unei celule care, în urma schimbării culorii, conduce la unificarea a două parcele și obținerea uneia de dimensiune maximă. Dacă nu există o astfel de celulă, se caută o celulă vecină cu parcela de dimensiune maximă determinată la prima cerință.

	1	2	3	4	5	6	7	8
1	1	2	2	3	3	3	4	4
2	5	6	6	3	3	3	4	4
3	5	6	6	3	8	8	8	8
4	6	6	6	7	8	8	8	8
5	6	6	7	7	7	9	9	11
6	6	6	7	7	7	9	9	9
7	10	10	10	10	10	10	10	9

Figura 6.2: fermaIR

6.1.2 *Rezolvare detaliată

6.1.3 Cod sursă

Listing 6.1.1: fermadaniel.cpp

```

1 // Popa Daniel - Colegiul National "Aurel Vlaicu" Orastie
2 #include <iostream>
3 #include <cstdio>
4 #include <cstring>
5
6 using namespace std;
7
8 const int lmax=402,cmax=402, //dimensiunile maxime ale terenului
9         po[4][2]={{-1,0},{0,-1},{0,1},{1,0}};
10 char h[lmax][cmax],u[lmax*cmax]; //cum sunt culturile
11 int a[lmax][cmax], //a[i][j]=x, unde x e numarul parcelei
12     s[lmax*cmax][2],vs=0, //stiva
13     t[lmax*cmax], // t[i]=x, i=nr parcelei, t[i]=aria parcelei i
14     v,m,n,nr=0;
15
16 void citire_init()
17 {
18     FILE *fin=fopen("ferma.in","r");
19     int i;
20     fscanf(fin,"%d\n%d %d\n",&v,&m,&n );
21     for(i=1;i<=m;i++)
22         fscanf(fin,"%s\n",h[i]+1);
23     fclose(fin);
24     //bordez marginea cu -1
25     for(i=0;i<=n+1;i++)a[0][i]=a[m+1][i]=-1;
26     for(i=1;i<=m;i++)a[i][0]=a[i][n+1]=-1;
27 }
28
29 void afis()
30 {
31     int i;
32     for(i=1;i<=m;i++)cout<<h[i]+1<<endl;
33 }
34
35 void afis2()
36 {
37     int i,j;
38     for(i=1;i<=m;i++)
39     {
40         for(j=1;j<=n;j++) {cout.width(3);cout<<a[i][j];}
41         cout<<endl;
42     }
43 }
44
45 //pun coord in stiva, cresc aria suprafetei nr, si marchez ca am fost pe acolo
46 void push(int l, int c,int nr)
47 {
48     vs++;s[vs][0]=l;s[vs][1]=c;t[nr]++;a[l][c]=nr;
49 }
50
51 //extrag coord din stiva
52 void pop(int &l, int &c)
53 {
54     l=s[vs][0];c=s[vs][1];vs--;
55 }
56
57 //incep de la coord li,co cu suprafata nr
58 int arie(int li,int co,int nr)
59 {
60     int l,c;
61     unsigned char cu=h[li][co];
62     vs=0;//golesc stiva si incep sa pun in ea
63     push(li,co,nr);
64     while(vs>0)//cat timp mai am in stiva
65     {
66         pop(l,c);//extrag din stiva
67         //merg in cele 4 directii
68         if((h[l+1][c]==cu)&&(a[l+1][c]==0))push(l+1,c,nr);
69         if((h[l][c+1]==cu)&&(a[l][c+1]==0))push(l,c+1,nr);
70         if((h[l-1][c]==cu)&&(a[l-1][c]==0))push(l-1,c,nr);

```

```

71         if((h[1][c-1]==cu)&&(a[1][c-1]==0))push(l,c-1,nr);
72     }
73     return t[nr];
74 }
75
76 void v1()
77 {
78     FILE *fout=fopen("ferma.out","w");
79     int i,j,ma=0,x;
80     nr=0; // sunt 0 suprafete gasite la inceput
81     for(i=1;i<=m;i++)
82         for(j=1;j<=n;j++)
83             if(a[i][j]==0)
84             {
85                 x=arie(i,j,++nr);
86                 if(x>ma)ma=x;
87             }
88     fprintf(fout,"%d\n",ma);
89     fclose(fout);
90 }
91
92 int bun(int l, int c, int &cu)
93 {
94     int i,j,p=0,ma=0,z,k;
95     char s;
96
97     for(i=0;i<4;i++)
98         {s=h[l+po[i][0]][c+po[i][1]];
99         if((s>='a')&&(s<='z'))&&(s!=h[l][c]))
100            {z=a[l+po[i][0]][c+po[i][1]];
101              p=t[z];
102              for(k=0;k<4;k++)
103                  u[a[l+po[k][0]][c+po[k][1]]]=0;
104                  u[z]=1;
105                  for(j=0;j<4;j++)
106                      if((s==h[l+po[j][0]][c+po[j][1]])&&
107                          (u[a[l+po[j][0]][c+po[j][1]]]==0))
108                          {p+=t[a[l+po[j][0]][c+po[j][1]]];
109                            u[a[l+po[j][0]][c+po[j][1]]]=1;}
110                      if((p!=t[z])&&(p>ma)) {ma=p+1;cu=s;}
111                  }
112      }
113
114 void v2()
115 {
116     FILE *fout=fopen("ferma.out","w");
117     int i,j,ma=0,x,l,c,c2,ma2=0,bnr=0,ku;
118     char cu;
119     nr=0; // sunt 0 suprafete gasite la inceput
120     for(i=1;i<=m;i++)
121         for(j=1;j<=n;j++)
122             if(a[i][j]==0)
123             {x=arie(i,j,++nr);
124              if(x>ma2){ma2=x;l2=i;c2=j;}
125             }
126     // caut sa lipesc 2,3,4 suprafete a.i. ara lor sa fie mai mare decat
127     // cea mai mare suprafata de cultura
128     for(i=1;i<=m;i++)
129         for(j=1;j<=n;j++)
130         {
131             x=bun(i,j,ku);
132             if((x>ma)&&(x>ma2)){ma=x;l=i;c=j;cu=ku;}
133         }
134     if(ma==0)//nu am reusit sa unesc 2 zone cresc cea mai mare zona cu o casuta
135     {
136         cout<<"bau"<<ma2;
137         cu=h[12][c2];bnr=a[12][c2];
138         for(i=1;(i<=m)&&(ma==0);i++) // parcurg matricea
139             for(j=1;(j<=n)&&(ma==0);j++)
140                 if(h[i][j]!=cu)//daca casuta curenta e de alta culoare decat
141                     // cea ce trebuie marita
142                 if(a[i+1][j]==bnr){l=i;c=j;ma=1;}//verific daca casuta adiacenta
143                     // face parte din parcela ce trebuie marita
144                 else if(a[i][j+1]==bnr){l=i;c=j;ma=1;}
145                 else if(a[i][j-1]==bnr){l=i;c=j;ma=1;}}

```

```

146           else
147             if(a[i-1][j]==bnr){l=i;c=j;ma=1;}
148             }
149             fprintf(fout,"%d %d\n%c\n",l,c,cu);
150             fclose(fout);
151           }
152         int main()
153         {
154           citire_init();
155           if(v==1)v1();
156           else v2();
157         }
158         return 0;
159     }
```

Listing 6.1.2: fermavlad.cpp

```

1 //Prof Nicu Vlad-Laurentiu - Liceul Teoretic "Mihail Kogalniceanu" Vaslui
2
3 #include <algorithm>
4 #include <cstdio>
5
6 using namespace std;
7
8 const int N=405;
9
10 int n, m, nrzones;
11 char a[N][N],cu;
12 int
13   b[N][N], dx[]={-1, 0, 1, 0}, dy[]={0, 1, 0, -1}, d[N*N],p;
14   bool c[N*N];
15 void filll(int x, int y)
16 {
17   b[x][y]=nrzones;
18   d[nrzones]++;
19   for(int i=0;i<4;i++)
20   {
21     if
22       (!b[x+dx[i]][y+dy[i]]&&a[x][y]==a[x+dx[i]][y+dy[i]])
23         filll(x+dx[i], y+dy[i]);
24   }
25
26 int main()
27 {
28   freopen("ferma.in", "r", stdin);
29   freopen("ferma.out", "w", stdout);
30   int i, j, k, l, sol=0, s=0,v,ma=0,p=0;
31   pair<int, int> soli;
32
33   scanf("%d\n%d%d", &v, &n, &m);
34   for(i=1;i<=n;i++)
35   {
36
37     scanf("%s", a[i]+1);
38
39   }
40   for(i=1;i<=n;i++)
41   {
42     for(j=1;j<=m;j++)
43     {
44       if(!b[i][j])
45       {
46         nrzones++;
47         filll(i, j);
48         if(ma<d[nrzones]) {ma=d[nrzones];}
49       }
50     }
51   }
52   for(i=1;i<=n;i++)
53   {
54     for(j=1;j<=m;j++)
55     {
```

```

56         char aux=a[i][j];
57         for(k=0;k<4;k++)
58         {
59             a[i][j]=a[i+dx[k]][j+dy[k]];
60             for(l=0, s=0;l<4;l++)
61             {
62                 if
63                 (!c[b[i+dx[l]][j+dy[l]]]&&a[i][j]==a[i+dx[l]][j+dy[l]])
64                 {
65                     c[b[i+dx[l]][j+dy[l]]]=1;
66                     s+=d[b[i+dx[l]][j+dy[l]]];
67                 }
68                 if(!c[b[i][j]]) s++;
69                 if(s>sol)
70                 {
71                     sol=s; cu=a[i][j];
72                     soli=make_pair(i, j);
73                 }
74                 for(l=0;l<4;l++)
75                 {
76                     c[b[i+dx[l]][j+dy[l]]]=0;
77                 }
78             }
79             a[i][j]=aux;
80         }
81     }
82
83     if(v==1) printf("%d\n",ma);
84     else{printf("%d %d\n", soli.first, soli.second);
85     printf("%c\n",cu);}
86 }
```

Listing 6.1.3: flore_nerecursiv.cpp

```

1 //Florentina Ungureanu - Colegiul National de Informatica Piatra-Neamt
2 #include <fstream>
3 #include <iostream>
4 #include <string.h>
5
6 #define nmax 410
7
8 using namespace std;
9
10 char a[nmax][nmax],c, cmax, cs;
11 unsigned b[nmax][nmax],z[nmax*nmax],smax,ssmax,s;
12 unsigned short x[nmax*nmax],y[nmax*nmax];
13 unsigned m, n, np, max, imax,jmax,is,js;
14
15 ifstream f("ferma.in");
16 ofstream g("ferma.out");
17
18 void suprafata(unsigned i, unsigned j)
19 {   int p, u;
20     p=u=1;
21     x[p]=i;y[p]=j;
22     while(p<=u)
23     { i=x[p];j=y[p];
24       if (!b[i+1][j]&&a[i+1][j]==c)
25         {s++;b[i+1][j]=np;
26          x[++u]=i+1, y[u]=j;
27        }
28       if (!b[i][j+1]&&a[i][j+1]==c)
29         {s++;b[i][j+1]=np;
30          x[++u]=i, y[u]=j+1;
31        }
32       if (!b[i-1][j]&&a[i-1][j]==c)
33         {s++;b[i-1][j]=np;
34          x[++u]=i-1, y[u]=j;
35        }
36       if (!b[i][j-1]&&a[i][j-1]==c)
37         {s++;b[i][j-1]=np;
38          x[++u]=i, y[u]=j-1;
39        }
40     p++;
```



```

169                     i=m+1; j=n+1;
170                 }
171             f.close();
172             g.close();
173         return 0;
174     }

```

Listing 6.1.4: flore_recursiv.cpp

```

1 //Florentina Ungureanu - Colegiul National de Informatica Piatra-Neamt
2 #include <iostream>
3 #include <string.h>
4
5 #define nmax 410
6
7 using namespace std;
8
9 char a[nmax][nmax],c, cmax, cs;
10 unsigned b[nmax][nmax],z[nmax*nmax],smax,ssmax,s;
11 unsigned m, n, np, max, imax,jmax,is,js;
12
13 ifstream f("ferma.in");
14 ofstream g("ferma.out");
15
16 void suprafata(unsigned i, unsigned j)
17 {
18     if (!b[i+1][j]&&a[i+1][j]==c)
19         {s++;b[i+1][j]=np;
20          suprafata (i+1, j);
21      }
22     if (!b[i][j+1]&&a[i][j+1]==c)
23         {s++;b[i][j+1]=np;
24          suprafata (i, j+1);
25      }
26     if (!b[i-1][j]&&a[i-1][j]==c)
27         {s++;b[i-1][j]=np;
28          suprafata (i-1, j);
29      }
30     if (!b[i][j-1]&&a[i][j-1]==c)
31         {s++;b[i][j-1]=np;
32          suprafata (i, j-1);
33      }
34 }
35
36 int main()
37 {
38     unsigned i,j,v;
39     char cuv[405];
40     f>>v;
41     f>>m>>n;
42     f.get();
43     for(i=1;i<=m;i++)
44     {
45         f.getline(cuv,405);
46         strcpy(a[i]+1,cuv);
47     }
48     np=0;
49     for(i=1;i<=m;i++)
50         for(j=1;j<=n;j++)
51             if(!b[i][j])
52             {
53                 np++;
54                 s=1;c=a[i][j];b[i][j]=np;
55                 suprafata(i,j);
56                 if(s>smax)
57                 {
58                     smax=s;is=i;js=j;cs=c;
59                 }
60                 z[np]=s;
61             }
62             if(v==1){g<<smax<<'\n';f.close();g.close();return 0;}
63             ssmax=0;
64             for(i=1;i<=m;i++)
65                 for(j=1;j<=n;j++)
66                 {

```

```

    if (a[i][j-1]==a[i][j+1]&&b[i][j-1]!=b[i][j+1]
67        &&
68        a[i][j-1]==a[i+1][j]&&b[i][j-1]!=b[i+1][j] &&
69            b[i][j+1]!=b[i+1][j]
70            &&
71            a[i][j-1]==a[i-1][j]&&b[i][j-1]!=b[i-1][j] &&
72                b
73                [i][j+1]!=b[i-1][j]&&b[i+1][j]!=b[i-1][j])
74                {
75                    s
76                    =z[b[i][j-1]]+z[b[i][j+1]]+z[b[i+1][j]]+z[b[i-1][j]]+1;
77                    if (s>ssmax)
78                    {
79                        ssmax=s;
80                        imax=i;
81                        jmax=j;
82                        cmax=a[i][j-1];
83                    }
84                }
85                else
86                if (a[i][j-1]==a[i][j+1]&&b[i][j-1]!=b[i][j+1]
87                    &&
88                    a[i][j-1]==a[i+1][j]&&b[i][j-1]!=b[i+1][j] &&
89                        b[i][j+1]!=b[i+1][j])
90                    {
91                        s
92                        =z[b[i][j-1]]+z[b[i][j+1]]+z[b[i+1][j]]+1;
93                        if (s>ssmax)
94                        {
95                            ssmax=s;
96                            imax=i;
97                            jmax=j;
98                            cmax=a[i][j-1];
99                        }
100                   }
101               }
102           else
103           if (a[i][j-1]==a[i][j+1]&&b[i][j-1]!=b[i][j+1]
104               &&
105               a[i][j-1]==a[i-1][j]&&b[i][j-1]!=b[i-1][j] &&
106                   b[i-1][j]!=b[i][j+1])
107                   {
108                       s
109                       =z[b[i][j-1]]+z[b[i][j+1]]+z[b[i-1][j]]+1;
110                       if (s>ssmax)
111                       {
112                           ssmax=s;
113                           imax=i;
114                           jmax=j;
115                           cmax=a[i][j-1];
116                       }
117                   }
118               }
119           else
120           if (a[i][j+1]==a[i+1][j]&&b[i][j+1]!=b[i+1][j]
121               &&
122               a[i][j+1]==a[i-1][j]&&b[i][j+1]!=b[i-1][j] &&
123                   b[i+1][j]!=b[i-1][j])
124                   {
125                       s
126                       =z[b[i][j+1]]+z[b[i+1][j]]+z[b[i-1][j]]+1;
127                       if (s>ssmax)

```

```

126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
    {
        ssmax=s;
        imax=i;
        jmax=j;
        cmax=a[i][j+1];
    }
}
else
{
    if(a[i][j-1]==a[i][j+1]&&b[i][j-1]!=b[i][j+1])
    {
        s=z[b[i][j-1]]+z[b[i][j+1]]+1;
        if(s>ssmax)
        {
            ssmax=s;
            imax=i;
            jmax=j;
            cmax=a[i][j-1];
        }
    }
    if
(a[i+1][j]==a[i-1][j]&&b[i+1][j]!=b[i-1][j])
{
    s=z[b[i+1][j]]+z[b[i-1][j]]+1;
    if(s>ssmax)
    {
        ssmax=s;
        imax=i;
        jmax=j;
        cmax=a[i+1][j];
    }
}
if
(a[i][j-1]==a[i-1][j]&&b[i][j-1]!=b[i-1][j])
{
    s=z[b[i][j-1]]+z[b[i-1][j]]+1;
    if(s>ssmax)
    {
        ssmax=s;
        imax=i;
        jmax=j;
        cmax=a[i][j-1];
    }
}
if
(a[i][j-1]==a[i+1][j]&&b[i][j-1]!=b[i+1][j])
{
    s=z[b[i][j-1]]+z[b[i+1][j]]+1;
    if(s>ssmax)
    {
        ssmax=s;
        imax=i;
        jmax=j;
        cmax=a[i][j-1];
    }
}
if
(a[i][j+1]==a[i-1][j]&&b[i][j+1]!=b[i-1][j])
{
    s=z[b[i][j+1]]+z[b[i-1][j]]+1;
    if(s>ssmax)
    {
        ssmax=s;
        imax=i;
        jmax=j;
        cmax=a[i][j+1];
    }
}
if
(a[i][j+1]==a[i+1][j]&&b[i][j+1]!=b[i+1][j])
{
    s=z[b[i][j+1]]+z[b[i+1][j]]+1;
    if(s>ssmax)
    {
        ssmax=s;
        imax=i;
        jmax=j;
    }
}

```

```

196                     cmax=a[i][j+1];
197                 }
198             }
199         }
200     }
201     if(ssmax)
202     {
203         g<<imax<<' '<<jmax<<'\n';
204         g<<cmax<<'\n';
205     }
206     else
207     {
208         for(i=1;i<=m;i++)
209             for(j=1;j<=n;j++)
210                 if(a[i][j]!=cs&&
211                     (
212                     b[i][j-1]==b[is][js]||b[i][j+1]==b[is][js] ||
213                     b[i+1][j]==b[is][js]||b[i-1][j]==b[is][js]))
214                 {
215                     g<<i<<' '<<j<<'\n';
216                     g<<cs<<'\n';
217                     i=m+1;j=n+1;
218                 }
219     }
220 }
```

6.2 triunghi

Problema 2 - triunghi

100 de puncte

Gigel este un pasionat al triunghiurilor. El colectează bețișoare de diferite lungimi și le asamblează în diferite triunghiuri. Ieri, el avea 6 bețișoare de lungimi 5, 2, 7, 3, 12 și 3. Din aceste bețișoare, Gigel a construit un triunghi de laturi 3, 3 și 5, iar bețișoarele de lungimi 2, 7, 12 au rămas nefolosite pentru că aceste lungimi nu pot forma laturile unui triunghi.

Din acest motiv, Gigel s-a hotărât să facă o colecție de bețișoare, dintre care oricum ar alege 3 elemente, acestea să nu poată forma laturile unui triunghi, proprietate pe care o vom numi în continuare *proprietate anti-triunghi*. Gigel, pornind de la setul inițial de lungimi 2, 7, 12, s-a gândit la două metode de realizare a unei colecții de 5 bețișoare cu proprietatea anti-triunghi, și anume:

1. Păstrează cel mai scurt bețișor, cel de lungime 2, și creează un set nou adăugând alte bețișoare de lungime mai mare sau egală cu cel inițial. De exemplu, următoarele 5 lungimi sunt corecte: 2, 2, 12, 50, 30.

2. Păstrează toate bețișoarele, și anume 2, 7, 12, pe care le va completa cu alte bețișoare de diferite lungimi (mai scurte sau mai lungi), astfel ca proprietatea anti-triunghi să se păstreze. Următoarele 5 lungimi respectă proprietatea anti-triunghi: 2, 7, 12, 4, 1.

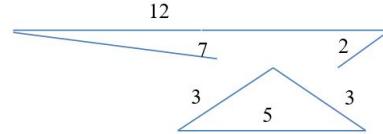


Figura 6.3: triunghi

Cerințe

Cunoscând un sir de n numere naturale nenule a_1, a_2, \dots, a_n având proprietatea anti-triunghi, și un număr k ($k > n$), se cere să construți un sir de k numere naturale având proprietatea anti-triunghi, în conformitate cu una dintre următoarele două restricții:

Varianta 1. Cel mai mic element este identic cu cel mai mic element din sirul inițial.

Varianta 2. Printre cele k elemente ale sirului construit se regăsesc toate elementele sirului inițial.

Date de intrare

Fișierul de intrare **triunghi.in** conține pe prima linie valorile numerelor v , n și k , separate prin spațiu. Linia următoare conține n numere naturale separate prin spațiu, ce formează un sir cu proprietatea anti-triunghi.

Date de ieșire

Fișierul de ieșire **triunghi.out** va conține k numere pe o singură linie.

Dacă valoarea lui v este 1, atunci fișierul va conține k numere naturale cu proprietatea anti-triunghi, separate prin spațiu, în care cel mai mic element este identic cu minimul sirului dat în fișierul de intrare.

Dacă valoarea lui v este 2, atunci fișierul va conține k numere naturale cu proprietatea anti-triunghi, separate prin spațiu, printre care se regăsesc toate elementele sirului inițial.

Restricții și precizări

- $3 \leq n < k \leq 46$;
- $1 \leq$ lungimea unui bețișor $\leq 2.000.000.000$;
- Pentru rezolvarea corectă a primei cerințe se acordă 30 de puncte, iar pentru cerința a două se acordă 70 de puncte;
 - Se garantează că întotdeauna există soluție;
 - Soluția nu este unică - se admite orice răspuns corect.

Exemple

triunghi.in	triunghi.out	Explicații
1 3 5 7 2 12	2 2 30 50 12	$v = 1$, $n = 3$, $k = 5$. În varianata 1 avem de tipărit 5 numere, valoarea minimului este 2 în ambele siruri.
2 3 5 7 2 12	1 4 12 7 2	$v = 2$, $n = 3$, $k = 5$. În varianata 2 printre elementele sirului tipărit se regăsesc toate elementele sirului inițial.

Timp maxim de executare/test: **0.1** secunde

Memorie: total **8 MB** din care pentru stivă **4 MB**

Dimensiune maximă a sursei: **5 KB**

6.2.1 Indicații de rezolvare

Autor: Szabo Zoltan - Liceul Tehnologic "Petru Maior" Reghin

Prima cerință

Se pot genera multe siruri, din care oricum am alege trei elemente, acestea să nu formeze triunghi.

De exemplu orice progresia geometrică cu rația mai mare decât 1 (1, 2, 4, 8, 16, ...).

Dintre toate sirurile cu proprietatea anti-triunghi, sirul lui Fibonacci este cel care crește cel mai încet (1, 1, 2, 3, 5, 8, 13, 21, ...)

De aceea, cel mai bun răspuns pentru prima cerință, este sirul $\min * f(i)$, cu elementele: $\min, \min, 2 * \min, 3 * \min, 5 * \min, 8 * \min, 13 * \min, 21 * \min, \dots$

A doua cerință

Primele două elemente din noul sir vor fi $b[1] = \min$ și $b[2] = \min$, dacă minimul apare de două ori în sirul a , și $b[1] = 1$ și $b[2] = 1$, dacă minimul apare o singură dată în sirul a .

Pronind de la aceste două valori initiale, un element $b[k]$ va avea valoarea

- $b[k - 1] + b[k - 2]$, dacă nu intră în conflict cu niciun element al sirului a , respectiv
- $a[p]$, dacă valoarea $b[k - 1] + b[k - 2]$ este în conflict cu elementul $a[p]$ (pentru a păstra atât elementele din a , cât și proprietatea anti-triunghi)

6.2.2 *Rezolvare detaliată

6.2.3 Cod sursă

Listing 6.2.1: triunghi_LS.cpp

¹ // Lukacs Sandor LICEUL ȘI ȘCOALA PROFESIONALĂ GHIBUĂȘI ORADEA

```

2 #include <iostream>
3 #include <fstream>
4
5 using namespace std;
6
7 ifstream f("triunghi.in");
8 ofstream g("triunghi.out");
9
10 int n,k,m,choise;//m-valoarea minima din vector
11 int nr;//numarul de bete adaugate
12 int v[50], u[50]; //v - initial, u-dupa adaugare
13
14 void citire()
15 {
16     int i;
17     f>>choise>>n>>k;
18     m=2000000000;
19     for(i=1;i<=n;i++)
20     {
21         f>>v[i];//citesc betioasere si calculez valoarea minima
22         if(v[i]<m) m=v[i];
23         u[i]=v[i];//pun toate betele in solutie pt. pct. b
24     }
25 }
26
27 void solutie1()
28 {
29     int b1,b2,b3;
30     g<<m<<" "<<m<<" ";//afisez cel mai mic bat de doua ori
31     nr=2;//am doua bete
32     b1=b2=m;
33     while(nr<k)
34     {//cat timp nu am k bete adaugate
35         b3=b2+b1;//calculez urmatoarea lungime cu care nu pot forma triunghi
36         b1=b2;b2=b3;//trec la urmatoarele 2 bete
37         g<<b2<<" ";//afisez lungimea nouui bat
38         nr++;//contabilizez batul
39     }
40     g<<"\n";
41 }
42
43 void sortare()
44 {//sortez betele initiale
45     int i,j,aux;
46     for(i=1;i<n;i++)
47         for(j=i+1;j<=n;j++)
48             if(v[i]>v[j]) {aux=v[i];v[i]=v[j];v[j]=aux;}
49 }
50
51 void solutie2()
52 {
53     //incerc sa pun bete inainte de fiecare bat i cu i<-1,
54     int i,b1,b2,b3,j; // b1-penultimul bat pus,
55                             // b2 - ultimul bat pus,
56                             // b3 - nouul bat care se adauga
57     i=1;//incepand cu primul bat din stiva initiala
58     nr=n;//am cele n bete initiale
59     b1=0;//penultimul bat pus
60     b2=1;//ultimul bat pus
61     if(b1+b2+v[i]>v[i+1])
62     {//daca adaug batul de lungime 1 pot depasi al doilea in cazul 2 2 5 7
63         b1=v[i];b2=v[i+1]; //atunci incep cu cele mai mici 2 bete
64     }
65     else
66     {//altfel adaug batul de lungime 1
67         nr = nr + 1;
68         u[nr]=1;
69     }
70
71     //incerc sa pun bete inainte de batul v[i]
72     while(nr<k && i<=n)
73     { //cat timp nu am pus k bete si mai sunt bete inainte carora pot pune
74         b3=b2+b1;//incerc sa pun b3
75         if(b3+b2<=v[i])
76             {//daca suma dintre ultimul bat adaugat si nnoul bat e ok

```

```

78         nr++; //creste numarul de bete adaugate
79         u[nr]=b3; //retin noul bat
80         b1=b2; //penultimul bat
81         b2=b3; //ultimul bat
82     }
83     else
84     { //nu mai pot pune bete inainte de v[i]
85         b1=b2; //penultimul bat adaugat
86         b2=v[i]; //ultimul bat
87         i=i+1; //trec la urmatorul bat inaintea caruia incerc sa pun
88     }
89 }
90
91 if(nr<k)
92 { //daca nu am pus toate betele inainte de betele initiale
93     //pun bete dupa ultimul bat
94     if(v[n-1]>u[nr]) b1=v[n-1]; //verific care e penultimul element
95     else b1=u[nr];
96     b2=v[n];
97     while(nr<k)
98     {
99         b3=b2+b1;
100        nr++;
101        u[nr]=b3;
102        b1=b2;
103        b2=b3;
104    }
105 }
106
107 for(i=1;i<=k;i++)
108 g<<u[i]<<" ";
109 }
110
111 int main()
112 {
113     citire();
114     if(choise==1)
115         solutie1();
116     else
117     {
118         sortare();
119         solutie2();
120     }
121     return 0;
122 }
```

Listing 6.2.2: triunghi_PD.cpp

```

1 // Popa Daniel Colegiul National Aurel Vlaicu Orastie
2 #include <iostream>
3 #include <fstream>
4 #include <algorithm>
5 #include <cstring>
6
7 using namespace std;
8
9 ifstream fin("triunghi.in");
10 ofstream fout("triunghi.out");
11
12 const long long ma=2000000001;
13 long long a[100],n,v,k,da=sizeof(a),de=sizeof(a[0]);
14
15 void v1()
16 {int i,x,mi=ma,a,b,c;
17 for(i=1;i<=n;i++)
18 {
19     fin>>x;
20     if(x<mi)mi=x;
21 }
22 a=b=mi;
23 fout<<a<<' '<<b;
24 for(i=3;i<=k;i++)
25 {
26     c=a+b;
27     fout<<' '<<c;
```

```

28         a=b;b=c;
29     }
30 }
31
32 void v2()
33 {long long i,x,y,z;
34   for(i=1;i<=n;i++) fin>>a[i];
35   sort(a+1,a+n+1);
36 //add la sf
37 while((n<k)&&(a[n]+a[n-1]<ma)) a[++n]=a[n-1]+a[n-2];
38 //add 1 la inceput, daca poate
39 if(n<k)
40 if(a[2]!=a[1]) {a[0]=1;memmove(a+1,a,da-de);n++;}
41 //incerc inca un 1
42 if(n<k)
43 if(1+a[1]<=a[2]) {a[0]=1;memmove(a+1,a,da-2*de);n++;}
44 // adau restul de numere "printre", dupa i
45 i=2;a[0]=0;
46 while(n<k)
47 {
48     x=a[i]+a[i-1];
49     if(x+a[i]<=a[i+1]) //daca pot insera un element il inserez
50     { //mut elementele la dreapta
51         memmove(a+i+1,a+i,da-(i+1)*de);
52         i++;
53         a[i]=x;
54         n++;
55     }
56     else i++; //trec la urm pozitie
57 }
58 //scrie solutie
59 for(i=1;i<=k;i++) fout<<a[i]<<' ';
60 }
61
62 int main()
63 {
64     fin>>v>>n>>k;
65     if(v==1) v1();
66     else v2();
67     fout.close();fin.close();
68     return 0;
69 }
```

Listing 6.2.3: zoli_triunghi.cpp

```

1 // Szabo Zoltan Liceul Tehnologic Petru maior Reghin
2 #include <fstream>
3
4 using namespace std;
5
6 int ok[50];
7
8 int main()
9 {
10     int pa,pb,a[50],n,k,i,j,b[50],aux,x,opt;
11
12     ifstream fin("triunghi.in");
13     ofstream fout("triunghi.out");
14
15     fin>>opt>>n>>k;
16
17     for (i=1;i<=n;i++)
18         fin>>a[i];
19
20     for (i=1;i<n;++i)
21         for (j=i+1;j<=n;++j)
22             if (a[i]>a[j])
23             {
24                 aux=a[i];
25                 a[i]=a[j];
26                 a[j]=aux;
27             }
28
29     if (opt==1)
30     {
```

```

31         b[1]=b[2]=a[1];
32         for (i=3;i<=k;i++)
33             b[i]=b[i-1]+b[i-2];
34         for (i=1;i<=k;+i)
35             fout<<b[i]<<' ';
36         fout<<"\n";
37     }
38 else
39 {
40     if (a[1]==a[2])
41     {
42         b[1]=b[2]=a[1];
43         ok[1]=ok[2]=1;           //in sirul b bifam toate elementele
44         pa=3;                  // ce apartin lui a
45     }
46 else
47 {
48     b[1]=b[2]=1;
49     if (a[1]==1)            // se bifeaza elementul 1 in cazul
50     {
51         ok[1]=1;              // in care este element din a
52         pa=2;
53     }
54     else
55         pa=1;
56 }
57
58 pb=2;
59 while (pa<n)
{
60     x=b[pb]+b[pb-1];
61
62     if (b[pb]+x<=a[pa] && x<=a[pa+1]-a[pa])
63         b[++pb]=x;
64     else
65     {
66         b[++pb]=a[pa++];
67         ok[pb]=1;
68     }
69 }
70
71 while (pa==n)
{
72     x=b[pb]+b[pb-1];
73     if(b[pb]<=a[pa]-x)
74         b[++pb]=x;
75     else
76     {
77         b[++pb]=a[pa++];
78         ok[pb]=1;
79     }
80 }
81
82
83 while (b[pb]<=2000000000-b[pb-1])
{
84     b[pb+1]=b[pb]+b[pb-1];
85     pb++;
86 }
87
88
89 for (i=1;i<=pb;i++)
90     if (ok[i])
91         fout<<b[i]<<" ";
92
93
94 x=k-n;
95 for(i=1;x && i<=pb;i++)
96     if (!ok[i])
97     {
98         fout<<b[i]<<" ";
99         --x;
100    }
101    fout<<"\n";
102 }
103
104 fin.close();
105 fout.close();
106

```

```
107     return 0;  
108 }
```

Capitolul 7

OJI 2013

7.1 calcule

Problema 1 - calcule

100 de puncte

Gigel a studiat recent şirurile cu n elemente, numere naturale. Pentru un astfel de şir S , Gigel doreşte să afle răspunsul la întrebările:

- Care este numărul minim de *subşiruri* strict crescătoare în care se poate partitiona S ?
- Care este numărul de *secvențe*, modulo 20011, cu suma elementelor divizibilă cu k care se pot obține din S ?

Cerințe

Dându-se un şir S cu n elemente numere naturale și un număr natural k se cere să se răspundă la cele două întrebări.

Date de intrare

Pe prima linie a fișierului **calcule.in** se află valorile naturale n și k separate printr-un spațiu. Pe următoarea linie se află cele n elemente ale şirului S , numere naturale separate prin câte un spațiu.

Date de ieșire

Fișierul **calcule.out** va conține două linii, pe prima linie fiind scris un număr natural reprezentând răspunsul la întrebarea a), iar pe a doua, un număr natural reprezentând răspunsul la întrebarea b).

Restricții și precizări

- $1 < n < 100000$
- S are elemente mai mici sau egale cu 20000
- $k < 50000, k < n$
- Un *subşir* al şirului S se obține selectând elemente din S în ordinea în care sunt în S , dar nu obligatoriu de pe poziții consecutive, iar o *secvență* a şirului S se obține selectând elemente în ordinea în care sunt în S , dar obligatoriu de pe poziții consecutive. Se admit și secvențe sau subşiruri cu un singur element.
 - Pentru 50% din teste $k < 10000$
 - Pentru răspuns corect la o singură cerință se acordă 50% din punctaj.
 - Mai multe subşiruri ale lui S formează o *partiție* dacă elementele reunii subşirurilor pot fi reașezate astfel încât să se obțină exact S .
 - x modulo y reprezintă restul împărțirii lui x la y .
 - În situația în care nu ati reușit să rezolvați cerința a), dar aveți un răspuns pentru b), veți scrie răspunsul pentru cerința b) pe linia 2 și nu pe prima linie!

Exemple

calcule.in	calcule.out	Explicații
10 3 5 3 8 6 9 6 2 7 9 6	4 23	<p>a) O partitie cu număr minim (4) de subșiruri crescătoare este următoarea:</p> <p>5 6 7 9 3 6 9 8 2 6</p> <p>b) Există 23 de secvențe cu suma divizibilă cu 3. Iată două dintre acestea:</p> <p>3 6 2 7</p>

Timp maxim de executare/test: **0.5** secunde

Memorie: total **64 MB** din care pentru stivă **32 MB**

Dimensiune maximă a sursei: **30 KB**

7.1.1 Indicații de rezolvare

prof. Gheorghe Manolache, Colegiul National de Informatica, Piatra-Neamț

Se observă că putem obține răspunsul la prima întrebare prin parcurgerea elementelor lui S , și crearea subșirurilor, fiecare element fiind adăugat la finalul subșirurilor deja începute, sau dacă nu e posibil, se va începe un nou subșir. Dacă se poate face completarea la mai multe siruri deja existente, se va face alipirea la subșirul care are la final o valoare cât mai mare.

Pentru eficiență, vom memora cu ajutorul unui vector sortat doar capetele acestor subșiruri, iar pentru căutare vom utiliza *căutarea binară*. Odată cu trecerea prin elementele lui S vom calcula sumele modulo k , ale elementelor până la poziția curentă, și pentru fiecare sumă vom centraliza valorile acestor resturi.

Evident, la b) răspunsul se obține folosind aceste resturi, observând că orice secvență corectă se poate face cu elementele aflate între oricare două poziții cu sumele cu același rest. Se obține astfel un algoritm de complexitate $n \log n$.

7.1.2 *Rezolvare detaliată

7.1.3 Cod sursă

Listing 7.1.1: calcFUCuBS.cpp

```

1 #include<iostream>
2
3 using namespace std;
4
5 int n,k,S[100002],s,nsolcresc,r[100002];
6 long long nrсолsk;
7
8 int bsearch(int x)
9 {
10     int i=0,j=nsolcresc,m;
11     while(i<j)
12         m=(i+j)/2, S[m]>=x? i=m+1:j=m;
13     return i;
14 }
15
16 int main()
17 {
18     int i,x,poz;
19
20     ifstream f("calcule.in");
21     ofstream g("calcule.out");
22
23     f>>n>>k;
24     for(i=1;i<=n;i++)
25     {

```

```

26         f>>x;
27         s=(s+x)%k;
28         r[s]=(r[s]+1)%20011;
29         poz=bsearch(x);
30         S[poz]=x;
31         if(poz==nsolcresc) nsolcresc++;
32     }
33
34     nrsolsk=r[0]*(r[0]+1)/2%20011;
35     for(i=1;i<k;i++)
36     nrsolsk=(nrsolsk+r[i]*(r[i]-1)/2)%20011;
37
38     g<<nsolcresc<<' \n'<<nrsolsk<<' \n';
39     f.close();
40     g.close();
41     return 0;
42 }
```

Listing 7.1.2: calcFUfaraBS.cpp

```

1 #include<fstream>
2
3 using namespace std;
4
5 int n,k,S[100002],s,nsolcresc,r[100002];
6 long long nrsolsk;
7
8 int search(int x)
9 {
10     int j=nsolcresc;
11     while(j>=0&&S[j]<x) j--;
12     return j+1;
13 }
14
15 int main()
16 {
17     int i,x,poz;
18
19     ifstream f("calcule.in");
20     ofstream g("calcule.out");
21
22     f>>n>>k;
23     for(i=1;i<=n;i++)
24     {
25         f>>x;
26         s=(s+x)%k;
27         r[s]=(r[s]+1)%20011;
28         poz=search(x);
29         S[poz]=x;
30         if(poz==nsolcresc) nsolcresc++;
31     }
32
33     nrsolsk=r[0]*(r[0]+1)/2%20011;
34     for(i=1;i<k;i++)
35     nrsolsk=(nrsolsk+r[i]*(r[i]-1)/2)%20011;
36
37     g<<nsolcresc<<' \n'<<nrsolsk<<' \n';
38     f.close();
39     g.close();
40     return 0;
41 }
```

Listing 7.1.3: calcule.cpp

```

1 #include<fstream>
2
3 using namespace std;
4
5 #define MOD 20011
6
7 int n,k,v[100010],i,sol,x,t,r[50010],s;
8 unsigned long long sol1;
9
10 ifstream f("calcule.in");
```

```

11  ofstream g("calcule.out");
12
13  int bs(int a)
14  {
15      int s=0,d,m;
16      for(d=sol;s<d;)
17      {
18          m=(s+d)/2;
19          if(v[m]>=a)
20              s=m+1;
21          else
22              d=m;
23      }
24      return s;
25  }
26
27  int main()
28  {
29      f>>n>>k;
30      for(i=1;i<=n;++i)
31      {
32          f>>x;
33          s=(s+x)%k;
34          r[s]++;
35          t=bs(x);
36          v[t]=x;
37          if(t==sol)
38              ++sol;
39      }
40
41      sol1=((r[0]%MOD)*((r[0]%MOD)+1)/2)%MOD;
42      for(i=1;i<k;++i)
43          sol1=(sol1+((r[i]%MOD)*((r[i]%MOD)-1)/2)%MOD)%MOD;
44
45      g<<sol1<<'n'<<sol1<<'n';
46      g.close();
47      return 0;
48 }
```

Listing 7.1.4: vcaculate.cpp

```

1 #include<iostream>
2
3 using namespace std;
4
5 ifstream in("calcule.in");
6 ofstream out("calcule.out");
7
8 int a[100001],r[100000];
9
10 int main()
11 { int n,i,k,x,j,s=0,sx=0;
12 r[0]=1;
13 in>>n>>k;
14 for(i=0;i<n;i++)
15 { in>>x;
16 sx=(sx+x)%k;
17 s=(s+r[sx])%20011;
18 r[sx]++;
19 for(j=x-1;a[j]==0&&j>0;j--);
20 a[j]--;a[x]++;
21 }
22 x=0;
23 for(i=1;i<20001;i++)
24     x=x+a[i];
25 out<<x<<'n'<<s;
26 return 0;
}
```

7.2 zona

Problema 2 - zona

100 de puncte

Ionuț pleacă în drumeție într-o porțiune de teren de formă pătratică cu latura de N metri. O hartă a zonei are trasat un caroaj care împarte zona în $N * N$ pătrate unitate, cu latura de 1 metru. Astfel harta zonei are aspectul unui tablou pătratic cu N linii și N coloane. Liniile și coloanele sunt numerotate de la 1 la N . Elementele tabloului bidimensional corespund pătratelor unitate. Zona poate fi parcursă străbatând oricare dintre laturile pătratelor unitate **cel mult o singură dată**.

Ionuț pleacă din punctul aflat în colțul din dreapta jos al pătratului unitate din linia X , coloana Y și se deplasează făcând un pas (parcugând o latură a unui pătrat unitate) în una din direcțiile *Nord, Est, Sud, Vest*. Pentru a reține mai ușor traseul folosește următoarea codificare pentru cele 4 direcții: 1 pentru deplasarea spre *Nord*, 2 pentru deplasarea spre *Est*, 3 pentru deplasarea spre *Sud*, respectiv 4 pentru deplasarea spre *Vest*.

Ajuns într-alt punct (colț de pătrat unitate), Ionuț continuă să se deplaceze fără a trece de mai multe ori pe aceeași latură a unui pătrat unitate.

Ionuț se oprește în momentul în care ajunge într-un punct prin care a mai trecut. Traseul străbatut între cele două treceri prin același punct delimită o zonă de teren formată din pătrate unitate.

Cerințe

Dându-se linia X și coloana Y corespunzătoare poziției de plecare a lui Ionuț, dimensiunea zonei N , lungimea traseului L și traseul determinați:

- Numărul de pași parcursi între prima și a doua trecere prin punctul de oprire.
- Numărul de pătrate unitate interioare zonei delimitată de traseul străbatut între cele două treceri prin același punct.

Date de intrare

Pe prima linie a fișierului **zona.in** se află valorile X , Y , N și L despărțite prin câte un spațiu, reprezentând coordonatele punctului de plecare, dimensiunea terenului și lungimea traseului parcurs. Pe următoarea linie se află L valori din mulțimea $\{1, 2, 3, 4\}$ despărțite prin câte un spațiu, reprezentând codificarea întregului traseu.

Date de ieșire

Fișierul **zona.out** va conține două linii, pe prima linie un număr natural reprezentând răspunsul la cerința a), iar pe linia a doua, un număr natural reprezentând răspunsul la cerința b).

Restricții și precizări

- $0 < N < 51$, $0 < X, Y < N$, $0 < L < 2501$.
- Se garantă că traseul trece de două ori prin același punct și nu parurge de două ori aceeași latură.
 - Pentru determinarea corectă a numărului de la punctul a) se acordă 20% din punctaj.
 - Pentru determinarea corectă a numărului de la punctul b) se acordă 80% din punctaj.
 - În situația în care nu ați reușit să rezolvați cerința a), dar aveți un răspuns pentru b), veți scrie răspunsul pentru cerința b) pe linia 2 și nu pe prima linie!

Exemple

zona.in	zona.out	Explicații
----------------	-----------------	-------------------

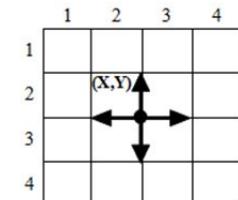


Figura 7.1: zona

2 3 7 18 2 3 3 3 4 3 4 1 1 1 1 1 2 2 2 3 3 4	16 11	După cei 18 pași de la plecare ajunge în punctul situat în colțul din dreapta jos al pătratului unitate de coordinate $(3,4)$. Ultimii 16 pași parcursi delimită 11 pătrate unitate.

Timp maxim de executare/test: **0.1** secunde

Memorie: total **64 MB** din care pentru stivă **32 MB**

Dimensiune maximă a sursei: **30 KB**

7.2.1 Indicații de rezolvare

Soluție 1

prof. Radu Vișinescu, C. N. "I.L. Caragiale", Ploiești, Prahova

Programul de rezolvare poate fi descompus în următoarele 5 etape:

PAS. I. Citirea datelor de intrare: perechea de coordonate (X, Y) , dimensiunea N , variabila lungime și elementele matricii V ce constituie codificarea drumului.

PAS. II. Parcugerea traseului până la prima poziție ce se repetă.

La fiecare arc nou parcurs se determină celulele din plan din stânga respectiv dreapta arcului. În funcție de coordonatele (X, Y) ale vârfului arcului, se folosește următoarea regulă, sintetizată în tabelul de mai jos:

Direcție	Cod direcție	Celula din stânga	Celula din dreapta
Nord	1	(X, Y)	$(X + 1, Y)$
Est	2	$(X, Y + 1)$	(X, Y)
Sud	3	$(X + 1, Y + 1)$	$(X, Y + 1)$
Vest	4	$(X + 1, Y)$	$(X + 1, Y + 1)$

Folosim o matrice A pentru memorarea în fiecare celulă a caroiajului a unei valori din mulțimea $\{0, 1, \dots, 15\}$ care, trecută în baza 2, exprimă prin cei 4 biți dacă laturile celulei fac parte din drumul definit în problemă. La început toate valorile matricii A sunt inițializate cu 0.

La fiecare pas al traseului:

A) În funcție de orientarea ultimului arc parcurs (poziția arcului față de celulă) în celulele din stânga și din dreapta arcului se adaugă valorile următoare: 1 pentru Nord, 2 pentru Est, 4 pentru Sud și 8 pentru Vest.

B) Folosim o variabilă P care în funcție de ultimele 2 orientări ale arcelor adaugă valoarea +1 pentru "cotirea" la stanga (în sens trigonometric), respectiv val -1 pentru "cotirea" la dreapta (în sens invers trigonometric). Inițial $P = 0$.

C) Tot la acest pas ținem o listă a perechilor de coordonate ale drumul în matricea L cu două linii: $L[1, J]$ pentru valoarea X , $L[2, J]$ pentru valoarea Y , unde J este o valoare între 1 și L . Inițial avem în matricea L doar coordonatele (X, Y) ale punctului de plecare. Pasul se încheie în momentul când ultima poziție ($L[1, J]$, $L[2, J]$) introdusă a fost găsită printre aceste poziții precedente.

PAS III. După terminarea pasului II cu valorile (X, Y) , în funcție de valoarea lui P determinăm o primă poziție din zona căutată, poziție de început a *algoritmului FILL*. Dacă $P > 0$ să mers trigonometric și conform tabelului de la Pas II. selectăm poziția celulei din stânga. Altfel, pentru $P < 0$ poziția celulei din dreapta (conform ordinii de mers).

PAS IV. Aplicăm următorul *algoritm de umplere*:

```
void fill(int x, int y)
{
    if (b[x][y]==0)
    {
        b[x][y]=1; NR++;
        if ((nord(baza2(a[x][y])))==0) fill(x, y+1);
        if ((est(baza2(a[x][y])))==0) fill(x+1, y);
        if ((sud(baza2(a[x][y])))==0) fill(x, y-1);
        if ((vest(baza2(a[x][y])))==0) fill(x-1, y);
    }
}
```

În care NR este o variabilă globală ce numără de câte ori se apelează funcția, iar: *baza2*, *nord*, *est*, *sud*, *vest* sunt funcții auxiliare ce extrag informațiile din celule cu privire la "pereti", informații introduse la parcurgerea traseului. Matricea B cu valori inițiale de 0 se foloseste pentru a nu trece de mai multe ori prin aceleași celule (ciclarea algoritmului **FILL**).

PAS V. Scrierea valorii de ieșire, adică a variabilei NR .

Ordinul de complexitate: Pasul de citire se efectuează în $O(N^2)$. Pasul II se repetă până când se dublează o poziție. Cum în întregul pătrat sunt cel mult N^2 perechi de coordonate și deoarece pentru fiecare nouă poziție se caută dublura în întreg vectorul ce memorează traseul deja parcurs, putem spune că acest pas este efectuat în maxim $O(N^4)$, (dacă avem $N^2 + 1$ poziții cel puțin una trebuie să se repete). Pașii III și V sunt efectuați în $O(1)$. Iar Pasul **FILL**, al IV-a, se efectuează în $O(N^2)$.

Așadar ordinul de complexitate al întregului algoritm este: $O(N^4)$.

Soluție 2

Prof. Popescu Doru Anastasiu, C. N. "Radu Greceanu", Slatina, Olt

I. Se construiește un tablou bidimesional cu elemente 0 și 1 în care 0 înseamnă punct nevizitat, iar 1 punct vizitat, odată cu citirea traseului. Tabloul este asociat colțurilor pătratelor zonei. Tot cu această ocazie se determină cerința de la punctul a) și poziția (prima) în traseu a punctului prin care se trece de două ori. Se șterg (se transformă în 0) din tablou elementele de 1 asociate primei părți din traseu, corespunzătoare punctelor până la punctul prin care se trece de două ori.

II. Se determină un punct din interiorul zonei delimitată de traseu, dacă există.

III. Folosind *algoritmul fill* se determină numărul de puncte interioare zonei (notat cu $Nr1$), apoi numărul de puncte de pe traseu ($Nr2$). $Nr2$ se poate determina direct prin numărarea punctelor eliminate.

IV. Aria (numarul de patrate din zona delimitată de traseu) va fi $Nr1 + Nr2/2 - 1$.

Soluție 3

Prof. Gheorghe Manolache, Colegiul Național de Informatică, Piatra-Neamț

Se construiește un tablou bidimensional în care se marchează traseul parcurs până la oprirea deplasării, numerotând pașii începând de la 1. Se determină ușor răspunsul la prima cerință. Se elimină marcajul la traseul care nu aparține conturului.

Pentru a afla aria zonei, vom porni din punctul de pe marginea care sigur nu este în interior (am făcut o translație a zonei) și vom aplica algoritmul *fill* marcând zona exterioară cu -1. Evident că zona rămasă va fi marcată cu *fill* cu valoarea 1. Apoi vom calcula aria zonei interioare, observând că un punct corect, are trei vecini marcați cu valori pozitive.

7.2.2 *Rezolvare detaliată

7.2.3 Cod sursă

Listing 7.2.1: Dzona.cpp

```

1 #include <fstream>
2
3 using namespace std;
4
5 ifstream fin("zona.in");
6 ofstream fout("zona.out");
7
8 int a[100][100], x,y,n,L,s,s1,Nr,cx[50000],cy[50000];
9 int dx[4]={-1, 0, 1, 0};
10 int dy[4]={ 0, 1, 0,-1};
11
12 void coordonate(int k,int x, int y, int &i, int &j){
13 if(k==1){
14 i=x-1;
15 j=y;
16 }
17 if(k==2){
18 i=x;
19 j=y+1;
20 }
21 if(k==3){
22 i=x+1;
23 j=y;
24 }
25 if(k==4){
26 i=x;
27 j=y-1;
28 }
29 }
30
31 void cit(){
32 int i,k,x1,y1,i1,j1;
33 fin>>x>>y>>n>>L;
34 x1=x;y1=y;
35 cx[0]=x;cy[0]=y;
36 a[x][y]=1;
37 for(i=1;i<=L;i++)
38 {
39 fin>>k;
40 coordonate(k,x1,y1,i1,j1);
41 cx[i]=i1;
42 cy[i]=j1;
43 if(a[i1][j1]==1) break;
44 a[i1][j1]=1;
45 x1=i1;y1=j1;
46 }
47 Nr=0;
48 for(i=0;i<L;i++)
49 if (cx[i]==i1&&cy[i]==j1)
50 break;
51 else{
52 a[cx[i]][cy[i]]=0;
53 Nr++;
54 }
55 Nr=L-Nr;
56 }
57
58 void afis(){
59 int i,j;
60 for(i=1;i<=n;i++){
61 for(j=1;j<=n;j++)
62 fout<<a[i][j]<<" ";
63 fout<<'\'n';
64 }
65 }
66
67 void fill(int x, int y){
68 int k, x1, y1;
69 s++;
70 a[x][y]=2;
71 for(k=0;k<4;k++){
72 x1=x+dx[k];
73 y1=y+dy[k];
74 if(x1>0&&y1>0&&x1<n+1&&y1<n+1&&a[x1][y1]==0)

```

```

75             fill(x1,y1);
76     }
77 }
78
79 void fill1(int x, int y){
80     int k, x1, y1;
81     s1++;
82     a[x][y]=2;
83     for(k=0;k<4;k++) {
84         x1=x+dx[k];
85         y1=y+dy[k];
86         if(x1>0&&y1>0&&x1<n+1&&y1<n+1&&a[x1][y1]==1)
87             fill1(x1,y1);
88     }
89 }
90
91 int verif(int x, int y){
92     int i1,i2,j1,j2;
93     if(a[x][y]!=0) return 0;
94     i1=x;
95     while(i1>0&&a[i1][y]==0) i1--;
96     if(i1==0) return 0;
97     i2=x;
98     while(i2<=n&&a[i2][y]==0) i2++;
99     if(i2==n+1) return 0;
100    j1=y;
101    while(j1>0&&a[x][j1]==0) j1--;
102    if(j1==0) return 0;
103    j2=y;
104    while(j2<=n&&a[x][j2]==0) j2++;
105    if(j2==n+1) return 0;
106    return 1;
107 }
108
109 int main()
110 {
111     cit();
112     //afis();
113     int i,j;
114     for(i=1;i<=n;i++)
115         for(j=1;j<=n;j++){
116             if(verif(i,j)){
117                 fill(i,j);
118                 //fout<<i<<" "<<j<<' \n';
119             }
120         fill1(cx[L],cy[L]);
121     //afis();
122     fout<<Nr<<' \n';
123     fout<<s+s1/2-1;
124     fout.close();
125     return 0;
126 }
```

Listing 7.2.2: Gzona.cpp

```

1 #include<iostream>
2
3 using namespace std;
4
5 ifstream f("zona.in");
6 ofstream g("zona.out");
7
8 int v[200][200];
9 int d[2510], n,l,x,y,sol;
10 int p,a1,a2,ax,by;
11
12 void citire()
13 {
14     f>>x>>y>>n>>l;
15     //n=100;
16     for(int i=1;i<=l;++i) f>>d[i];
17     x+=55;y+=55;
18     ax=x;by=y;
19     for(int i=0;i<=n+58;++i) v[0][i]=v[n+58][i]=v[i][0]=v[i][n+58]=-1;
20 }
```

```

21
22 void fill(int a,int b)
23 {
24     if(v[a][b]==0)
25     {
26         v[a][b]=-1;
27         fill(a+1,b);
28         fill(a,b+1);
29         fill(a-1,b);
30         fill(a,b-1);
31     }
32 }
33
34 void fill1(int a,int b)
35 {
36     if(v[a][b]==0)
37     {
38         v[a][b]=1;
39         fill1(a+1,b);
40         fill1(a,b+1);
41         fill1(a-1,b);
42         fill1(a,b-1);
43     }
44 }
45
46 void afis(int v[200][200])
47 {
48     for(int i=1;i<200;++i)
49     {
50         for(int j=1;j<200;++j)
51             if(v[i][j]>0) g<<"* "; //else g<<". ";
52             else g<<v[i][j]<<' ';
53         g<<'\'n';
54     }
55     g<<'\'n';
56 }
57
58 int merg(int x, int y)
59 {
60     sol=0;
61     while(sol<1 && v[x][y]==0)
62     {
63         sol++;
64         if(sol<=1){
65             v[x][y]=sol;
66             switch(d[sol])
67             {
68                 case 1: x--;
69                     break;
70                 case 2:
71                     y++;
72                     break;
73                 case 3:
74                     x++;
75                     break;
76                 case 4:
77                     y--;
78             }
79         }
80         //if(v[x][y]>0) break;
81     }
82     p=v[x][y];
83     if(sol>1) return 0;
84     return sol-v[x][y]+1;
85 }
86
87 void sterg(int a0, int b0)
88 {
89     sol=0;
90     while(v[a0][b0]!=p)
91     {
92         sol++;
93         if(sol<=1)
94         {
95             v[a0][b0]=0;
96             switch(d[sol])

```

```
97         {
98             case 1: a0--;
99                 break;
100            case 2:
101                b0++;
102                break;
103            case 3:
104                a0++;
105                break;
106            case 4:
107                b0--;
108        }
109    }
110}
111
112 int main(){
113     citire();
114     a1=merg(ax,by);
115     //afis(v);
116     sterg(ax,by);
117     fill(1,1);
118
119     for(int i=1;i<=n+55;++i)
120         for(int j=1;j<=n+55;++j)
121             if(v[i][j]==0) fill1(i,j);
122
123     for(int i=2;i<=n+55;++i)
124         for(int j=1;j<=n+55;++j)
125             if(v[i][j]>0 && v[i][j+1]>0 && v[i-1][j]>0 && v[i-1][j+1]>0)
126                 a2++;
127
128
129     g<<a1<<'\\n'<<a2<<'\\n';
130     g.close();
131     return 0;
132 }
```

Capitolul 8

OJI 2012

8.1 compresie

Problema 1 - compresie

100 de puncte

Se consideră un text memorat într-o matrice M , definită prin coordonatele colțului stânga sus (x_1, y_1) și coordonatele colțului dreapta jos (x_2, y_2) .

Prin aplicarea unui algoritm de compresie, matricei M i se asociază un sir de caractere, notat C_M .

Sirul de caractere C_M este construit prin aplicarea următoarelor reguli:

a) dacă matricea M are o singură linie și o singură coloană atunci C_M conține numai caracterul memorat în matrice;

b) dacă toate elementele matricei sunt identice atunci întreaga matrice M se comprimă și C_M este sirul kc , unde k reprezintă numărul de caractere din matrice, iar c caracterul memorat;

c) dacă matricea este formată din caractere diferite și are cel puțin două linii și două coloane atunci:

- matricea este împărțită în 4 submatrice A, B, C, D după cum este ilustrat în figura alăturată, unde coordonatele colțului stânga sus ale submatricei A sunt (x_1, y_1) , iar coordonatele colțului dreapta jos sunt $((x_2 + x_1)/2, (y_2 + y_1)/2)$;

- C_M este sirul $*C_A C_B C_C C_D$ unde C_A, C_B, C_C, C_D sunt sirurile de caractere obținute, în ordine, prin compresia matricelor A, B, C, D utilizând același algoritm;

d) dacă matricea este formată din caractere diferite, are o singură linie și mai multe coloane atunci C_M este sirul $*C_A C_B$ unde A, B, C_A, C_B au semnificația descrisă la punctul c);

e) dacă matricea este formată din caractere diferite, are mai multe linii și o singură coloană atunci C_M este sirul $*C_A C_C$ unde A, C, C_A, C_C au semnificația descrisă la punctul c).

	(x_1, y_1)	
A		B
C		D
		(x_2, y_2)

Figura 8.1: compresie

Cerințe

Dat fiind sirul de caractere C_M ce se obține în urma aplicării algoritmului de compresie asupra unei matrice M de dimensiune $N \times N$ să se determine:

- numărul de împărțiri care au fost necesare pentru obținerea textului compresat;
- matricea inițială din care provine textul compresat.

Date de intrare

Fisierul de intrare **compresie.in** conține pe prima linie un sir de caractere ce reprezintă textul compresat.

Date de ieșire

Fisierul de ieșire **compresie.out** conține:

- pe prima linie un număr natural ce reprezintă numărul nr de împărțiri care au fost necesare pentru obținerea textului compresat;
- pe următoarele N linii se găsesc câte N caractere, litere mici ale alfabetului englez, neseparate prin spații, ce reprezintă, în ordine, liniile matricei inițiale.

Restricții și precizări

- $2 \leq N \leq 1000$
- $0 \leq nr \leq 1000000$
- $2 \leq$ lungimea sirului compresat ≤ 1000000
- Textul memorat initial inn matricea M contine numai caractere din multimea literelor mici $\{a, \dots, z\}$.
- Pentru rezolvarea corecta a cerintei a) se acorda 20% din punctaj, iar pentru rezolvarea corecta a ambelor cerinte se acorda tot punctajul.

Exemple

compresie.in	compresie.out	Explicatii
*4b*bbab4a*abbb	3 bbbb bbab aaab aabb	Au fost efectuate 3 impartiri: 1) $M = * \begin{pmatrix} b & b \\ b & b \end{pmatrix} \begin{pmatrix} b & b \\ a & b \end{pmatrix} \begin{pmatrix} a & a \\ a & a \end{pmatrix} \begin{pmatrix} a & b \\ b & b \end{pmatrix}$ 2) $\begin{pmatrix} b & b \\ a & b \end{pmatrix} = * (b)(b)(a)(b)$ 3) $\begin{pmatrix} a & b \\ b & b \end{pmatrix} = * (a)(b)(b)(b)$
*4a*ab*aba	3 aaa aab aba	Au fost efectuate 3 impartiri: 1) $M = * \begin{pmatrix} a & a \\ a & a \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} (a \ b) (a)$ 2) $\begin{pmatrix} a \\ b \end{pmatrix} = * (a) (b)$ 3) $(a \ b) = * (a) (b)$

Timp maxim de executare/test: **0.5** secunde

Memorie: total **4 MB** din care pentru stivă **3 MB**

Dimensiune maximă a sursei: **10 KB**

8.1.1 Indicatii de rezolvare

prof. ????? - ?????

Cerinta 1

Numarul de impartirii care au fost necesare pentru obtinerea textului compresat

- nr = numarul de steluțe '*'.

Cerinta 2

Rezolvarea cerintei se face prin *divide et impera*.

Retinem matricea prin colțul stanga sus ($x1, y1$), respectiv colțul dreapta jos ($x2, y2$).

Vom parurge sirul compresat cu variabila k .

divide:

```
mx = (x2 + x1) >> 1; my = (y2 + y1) >> 1;

// reconstruiesc submatricile A,B,C,D
divide (x1, y1, mx, my); //A
divide (x1, my + 1, mx, y2); //B
divide (mx + 1, y1, x2, my); //C
divide (mx + 1, my + 1, x2, y2); //D
```

stapanește:

- Daca $x1=x2 \ \&\ y1==y2$ atunci $M[x1][x2] = sir[k]$

- Daca $sir[k] \in \{'0', ..., '9'\}$ atunci se va forma numarul x , iar submatricea definita de coordinatele coltului stanga sus ($x1, y1$) si de coordonatele coltului dreapta jos ($x2, y2$) va fi completata cu caracterul din sir asociat numarului format.

Nu este necesara etapa "combină".

8.1.2 *Rezolvare detaliată

8.1.3 Cod sursă

Listing 8.1.1: compresie_cristina_sichim.cpp

```

1  /*
2   *      Solutie prof. Cristina Sichim
3  */
4 # include <iostream>
5 # include <cstring>
6
7 using namespace std;
8
9 char s[1000001], a[1001][1001];
10 int i,n,l,j,k,Nr;
11
12 ifstream f("compresie.in");
13 ofstream g("compresie.out");
14
15 int dimensiune()
16 { n=0;
17     int i=0,nr;
18     while(i<l){ if(s[i]=='*') Nr++,i++;
19                 else if(s[i]>='a' && s[i]<='z') n++,i++;
20                 else {nr=0;
21                         while(s[i]>='0' && s[i]<='9') {nr=nr*10+(s[i]-'0');
22                                         i++;}
23                         n=n+nr;i++;}
24             }
25     }
26     i=1;
27     while(i*i<n) i++;
28     return i;
29 }
30
31 void matrice(int i1,int j1, int i2, int j2)
32 { int x,y,nr;
33     if(i1<=i2 && j1<=j2)
34     {
35         if(s[i]=='*') { i++;
36                         x=(i1+i2)/2;y=(j1+j2)/2;
37                         matrice(i1,j1,x,y);
38                         matrice(i1,y+1,x,j2);
39                         matrice(x+1,j1,i2,y);
40                         matrice(x+1,y+1,i2,j2);
41                     }
42         else if(s[i]>='a' && s[i]<='z')
43             { a[i1][j1]=s[i];i++;
44                 matrice(i1,j1+1,i1,j2); //B
45                 matrice(i1+1,j1,i2,j1); //C
46                 matrice(i1+1,j1+1,i2,j2); //D
47             }
48         else{ while(s[i]>='0' && s[i]<='9') i++;
49                 for(x=i1;x<=i2;x++)
50                     for(y=j1;y<=j2;y++) a[x][y]=s[i];
51                 i++;
52             }
53     }
54
55 int main()
56 { f>>s;
57     l=strlen(s);
58     n=dimensiune();
59     g<<Nr<<'\n';
60     i=0;
61     matrice(1,1,n,n);
62     for(i=1;i<=n;i++)
63         {for(j=1;j<=n;j++) g<<a[i][j];
64          g<<'\n';
65      }
66     f.close();g.close();
67     return 0;
68 }
```

Listing 8.1.2: compresie_eugen_nodeal.cpp

```

1  /*
2   *      Solutie prof. Eugen Nodea
3   */
4 # include <iostream>
5 # include <cmath>
6 # include <cstring>
7
8 using namespace std;
9
10 char s[1000001];
11 char M[1001][1001];
12 int L, N, nr, i, j, k;
13
14 void reconstruieste(short x1, short y1, short x2, short y2)
15 {
16     short mx, my, x, i, j;
17
18     //conditia de oprire
19     if (x1<=x2 && y1<=y2 && k<L)
20     {
21         //stapaneste
22         if (x1==x2 && y1==y2)
23             M[x1][y1] = s[k++];
24         else
25             if (s[k]>='0' && s[k] <='9')
26             {
27                 x = 0;
28                 while (s[k]>='0' && s[k]<='9')
29                 {
30                     x = x*10 + (s[k] - '0');
31                     ++k;
32                 }
33
34                 //completez submatricea
35                 for (i=x1; i<=x2; ++i)
36                     for (j=y1; j<=y2; ++j)
37                         M[i][j] = s[k];
38                     ++k;
39             }
40         else //s[k] == '*'
41         {
42             //divide
43             mx = (x2 + x1) >> 1; my = (y2 + y1) >> 1;
44             ++k;
45             // reconstruiesc submatricile A,B,C,D
46             reconstruieste(x1, y1, mx, my);           //A
47             reconstruieste(x1, my + 1, mx, y2);        //B
48             reconstruieste(mx + 1, y1, x2, my);        //C
49             reconstruieste(mx + 1, my + 1, x2, y2);    //D
50         }
51     }
52 }
53
54 int main()
55 {
56     freopen("compresie.in", "r", stdin);
57     freopen("compresie.out", "w", stdout);
58
59     fgets(s, 1000001, stdin);
60     L = strlen(s);
61
62     //punctul 1
63     k = 0; nr = 0; j = 0;
64     for (i=0; i < L; ++i)
65         if (s[i] == '*') ++nr;
66         else
67             if (s[i]>='0' && s[i]<='9')
68                 k = k*10 + (s[i] - '0');
69             else
70                 if (s[i]>='a' && s[i]<='z')
71                 {
72                     if (s[i-1]<'a' && k)
73                         j+=k, k=0;
74                     else

```

```

75           ++j;
76       }
77   N = (int) sqrt((double) j);
78   printf("%d\n", nr);
79
80 //punctul 2
81 k=0;
82 reconstruiese(1,1,N,N);
83 for (i=1; i<=N; ++i)
84 {
85     for (j=1; j<=N; ++j)
86         printf("%c",M[i][j]);
87     printf("\n");
88 }
89 return 0;
90 }
```

Listing 8.1.3: compresie_eugen_nodea2.cpp

```

1 /*
2      Solutie prof. Eugen Nodea
3      streamuri
4 */
5 # include <iostream>
6 # include <cmath>
7 # include <cstring>
8
9 using namespace std;
10
11 ifstream f("compresie.in");
12 ofstream g("compresie.out");
13
14 char s[1000001];
15 char M[1001][1001];
16 int L, N, nr, i, j, k;
17
18 void reconstruiese(short x1, short y1, short x2, short y2)
19 {
20     short mx, my, x, i, j;
21
22     //conditia de oprire
23     if (x1<=x2 && y1<=y2 && k<L)
24     {
25         //stapaneste
26         if (x1==x2 && y1==y2)
27             M[x1][y1] = s[k++];
28         else
29             if (s[k]>='0' && s[k] <='9')
30             {
31                 x = 0;
32                 while (s[k]>='0' && s[k]<='9')
33                 {
34                     x = x*10 + (s[k] - '0');
35                     ++k;
36                 }
37
38                 //completez submatricea
39                 for (i=x1; i<=x2; ++i)
40                     for (j=y1; j<=y2; ++j)
41                         M[i][j] = s[k];
42                     ++k;
43                 }
44             else //s[k] == '*'
45             {
46                 //divide
47                 mx = (x2 + x1) >> 1; my = (y2 + y1) >> 1;
48                 ++k;
49                 // reconstruiesc submatricile A,B,C,D
50                 reconstruiese(x1, y1, mx, my); //A
51                 reconstruiese(x1, my + 1, mx, y2); //B
52                 reconstruiese(mx + 1, y1, x2, my); //C
53                 reconstruiese(mx + 1, my + 1, x2, y2); //D
54             }
55     }
56 }
```

```

57
58 int main()
59 {
60     f.getline(s, 1000001);
61     f.close();
62     L = strlen(s);
63
64     //punctul 1
65     k = 0; nr = 0; j = 0;
66     for (i=0; i < L; ++i)
67         if (s[i] == '*')
68             ++nr;
69         else
70             if (s[i]>='0' && s[i]<='9')
71                 k = k*10 + (s[i] - '0');
72             else
73                 if (s[i]>='a' && s[i]<='z')
74                 {
75                     if (s[i-1]<'a' && k)
76                         j+=k, k=0;
77                     else
78                         ++j;
79                 }
80
81     N = (int) sqrt((double) j);
82     g << nr << "\n";
83
84     //punctul 3
85     k=0;
86     reconstruieste(1,1,N,N);
87     for (i=1; i<=N; ++i)
88     {
89         for (j=1; j<=N; ++j)
90             g << M[i][j];
91         g << "\n";
92     }
93     g.close();
94     return 0;
95 }
```

Listing 8.1.4: compresie_silviu_candale.cpp

```

1 /*
2      Solutie prof. Silviu Candale
3 */
4 #include <iostream>
5 #include <iostream>
6 #include <cassert>
7 #include <cstring>
8 #include <cmath>
9
10 #define NN 1000
11 //#define DEBUG
12
13 using namespace std;
14
15 ifstream fin("compresie.in");
16 ofstream fout("compresie.out");
17
18 int n;
19 char s[NN*NN+10], a[NN+2][NN+2];
20 int poz;
21 int cifra(char c)
22 {
23     return c>='0' && c<='9';
24 }
25
26 void reconstituire(int i1,int j1, int i2, int j2)
27 {
28     if(i1>i2 || j1>j2)
29         return;
30     if(s[poz] == '*')
31     {
32         int mi=(i1+i2)/2, mj=(j1+j2)/2;
33         poz++;
```

```

34     reconstituire(i1 , j1 , mi , mj);
35     reconstituire(i1 , mj+1 , mi , j2);
36     reconstituire(mi+1 , j1 , i2 , mj);
37     reconstituire(mi+1 , mj+1 , i2 , j2);
38 }
39 else
40 {
41     while( cifra(s[poz]) )
42         ++poz;
43     for(int i=i1 ; i<=i2 ; ++i)
44         for(int j=j1 ; j<=j2 ; ++j)
45             a[i][j]=s[poz];
46         ++poz;
47     }
48 }
49
50 int main()
51 {
52     fin.getline(s,NN*NN+5);
53     //
54     int tائeturi = 0,in_numar = 0,x = 0, np=0;
55     for(int i=0;s[i];++i)
56         if(s[i]=='*')
57             ++tائeturi;
58         else{
59             if(cifra(s[i]))
60                 if(in_numar)
61                     x = 10 * x + s[i]-'0';
62                 else
63                     x=s[i]-'0' , in_numar = 1;
64             else{
65                 np += x;
66                 x = 0;
67                 if(s[i]>='a' && s[i]<='z' && !cifra(s[i-1]) )
68                     np++;
69             }
70         }
71
72 #ifdef DEBUG
73     cout << np << endl;
74 #endif
75
76 n = (int)sqrt((double)np);
77 fout << tائeturi << "\n";
78 reconstituire(1,1,n,n);
79 for(int i=1 ; i<=n ; ++i)
80 {
81     for(int j=1 ; j<=n ; ++j)
82         fout << a[i][j];
83         fout << endl;
84     }
85 }

```

8.2 culori

Problema 2 - culori

100 de puncte

Pasiunea Mirunei este să coloareze. Vacanța trecută și-a petrecut-o la bunica ei la țară și pentru că se cam plătisea s-a gândit să vopsească gardul de la casa bunicii.

Gardul este compus din N scânduri dispuse una lângă alta. Miruna a găsit în garajul bunicii 5 cutii de vopsea de culori diferite: **albă, albastră, roșie, verde și galbenă**. Când a vopsit gardul, Miruna a respectat următoarele reguli:

- Dacă o scândură era vopsită cu **albă**, următoarea scândură o vopsea obligatoriu cu **albastru**
- Dacă o scândură era vopsită cu **albastru**, atunci următoarea scândură o vopsea cu **albă** sau **roșu**
- Dacă o scândură era vopsită cu **roșu**, atunci următoarea scândură o vopsea cu **albastru** sau **verde**
- Dacă o scândură era vopsită cu **verde**, atunci următoarea scândură o vopsea cu **roșu** sau **galbenă**
- Dacă o scândură era vopsită cu **galbenă**, atunci următoarea scândură o vopsea obligatoriu cu **verde**

După ce a și-a terminat treaba Miruna își admira "opera de artă" și se întreba în câte moduri diferite ar fi putut să vopsească gardul bunicii.

Cerințe

Ajutați-o pe Miruna să găsească răspunsul la întrebarea sa.

Date de intrare

Fișierul **culori.in** conține pe prima sa linie un singur număr natural N ($1 \leq N \leq 5000$).

Date de ieșire

Fișierul de ieșire **culori.out** va conține pe prima sa linie un singur număr întreg reprezentând numărul de moduri diferite în care Miruna ar fi putut să vopsească gardul bunicii.

Restricții și precizări

- $1 \leq N \leq 5000$;
- Pentru 25% dintre teste $N \leq 45$.

Exemple

culori.in	culori.out	Explicații
4	24	Gardul poate fi vopsit astfel: (alb,albastru,alb,albastru); (alb,albastru,rosu,albastru); (alb,albastru,rosu,verde); (albastru,alb,albastru,alb); (albastru,alb,albastru,rosu); (albastru,rosu,albastru,alb); (albastru,rosu,albastru,rosu); (albastru,rosu,verde,rosu); (albastru,rosu,verde,galben); (rosu,albastru,alb,albastru); (rosu,albastru,rosu,albastru); (rosu,albastru,rosu,verde); (rosu,verde,rosu,albastru); (rosu,verde,rosu,verde); (rosu,verde,galben,verde); (verde,rosu,albastru,alb); (verde,rosu,albastru,rosu); (verde,rosu,verde,rosu); (verde,rosu,verde,galben); (verde,galben,verde,rosu); (verde,galben,verde,galben); (galben,verde,rosu,albastru); (galben,verde,rosu,verde); (galben,verde,galben,verde).

Timp maxim de executare/test: **0.2** secunde

Memorie: total **1 MB** din care pentru stivă **1 MB**

Dimensiune maximă a sursei: **10 KB**

8.2.1 Indicații de rezolvare

prof. Carmen Popescu - Col. Nat. Gh. Lazăr Sibiu

Notăm

$nr[i, j] =$ numărul de variante de a vopsi primele i scânduri, dacă scândura i o vopsim cu culoarea j , $j = 1, 2, 3, 4, 5$

$S[i] =$ numărul de variante de a vopsi primele i scânduri din gard

Se observă că au loc următoarele relații:

```
S[i]=nr[i,1]+nr[i,2]+nr[i,3]+nr[i,4]+nr[i,5]
```

```
nr[i,1]=nr[i-1,2]
```

```
nr[i,2]=nr[i-1,1]+nr[i-1,3]
```

```
nr[i,3]=nr[i-1,2]+nr[i-1,4]
```

```
nr[i,4]=nr[i-1,3]+nr[i-1,5]
```

```
nr[i,5]=nr[i-1,4]
```

Cum $nr[1, j]=1$ se poate observa ușor că

$\Rightarrow nr[i, 1]=nr[i, 5]$ și

```
nr[i,2]=nr[i,4]
```

și

```
nr[i,3]=2*nr[i,2]
```

Așadar:

```

nr[i,2]=nr[i-1,1]+nr[i-1,3]=nr[i-2,2]+2*nr[i-2,2]=3*nr[i-2,2]  pt i>2
nr[i,1]=nr[i-1,2]=3*nr[i-3,2]=3*nr[i-2,1]  pt i>4
nr[i,3]=2*nr[i-1,2]=6*nr[i-3,2]=3*nr[i-2,3]  pt i>4
=> S[i]= 2*nr[i,1]+2*nr[i,2]+nr[i,3] =
      = 6*nr[i-2,1]+6*nr[i,2,2]+3*nr[i-2,3] = 3*S[i-2]

```

In concluzie obtinem:

```

S[1]=5
S[2]=8
S[3]=14
S[2k] = 3^(k-1) * S[2]  pt k>1
si S[2k+1] = 3^(k-1) *S[3]  pt k>1

```

Pentru calcularea acestor expresii se vor folosi *numere mari*.

8.2.2 *Rezolvare detaliată

8.2.3 Cod sursă

Listing 8.2.1: culoriEM.cpp

```

1 // culori O(n*L) - L = lungimea rezultatului  prof. Eugen Mot
2 #include <stdio.h>
3 #include <string.h>
4
5 #define C 1500
6
7 int w[2][C], b[2][C], r[2][C], g[2][C], y[2][C]; // de la white, blue, red etc.
8
9 void copy(int a[], int b[])
10 {
11     int i;
12     for(i=0;i<C;i++) b[i]=a[i];
13 }
14
15 void sum(int a[], int b[], int c[])
16 {
17     int i,t=0,r;
18     c[0]=(a[0]>=b[0]?a[0]:b[0]);
19     for(i=1;i<=c[0];i++)
20     {
21         r=a[i]+b[i]+t;
22         c[i]=(r>9?r-10:r);
23         t=r>9;
24     }
25     if(t)
26     {
27         c[0]++;
28         c[c[0]]=1;
29     }
30 }
31
32 int main()
33 {
34     FILE *fi,*fo;
35     int i,j,k,n;
36
37     fi=fopen("culori.in","r");
38     fscanf(fi,"%d",&n);
39     fclose(fi);
40
41     w[1][0]=b[1][0]=r[1][0]=g[1][0]=y[1][0]=1;
42     w[1][1]=b[1][1]=r[1][1]=g[1][1]=y[1][1]=1;
43
44     for(i=2;i<=n;i++)
45     {
46         j=i&1;
47         k=1-j;

```

```

48     copy(b[k],w[j]);
49     sum(w[k],r[k],b[j]);
50     sum(b[k],g[k],r[j]);
51     sum(r[k],y[k],g[j]);
52     copy(g[k],y[j]);
53 }
54
55 j=n&1;
56 k=1-j;
57 sum(w[j],b[j],w[k]);
58 sum(w[k],r[j],b[k]);
59 sum(b[k],g[j],r[k]);
60 sum(r[k],y[j],g[k]);
61 fo=fopen("culori.out", "w");
62 for(i=g[k][0];i>0;i--)
63     fprintf(fo, "%d", g[k][i]);
64 fclose(fo);
65 return 0;
66 }
```

Listing 8.2.2: culori.cpp

```

1 // prof. Carmen Popescu - Col. Nat. Gh. Lazar Sibiu
2 #include <fstream>
3
4 using namespace std;
5
6 ifstream f("culori.in");
7 ofstream g("culori.out");
8
9 int n;
10 string s;
11
12 void s_la_p(int p)
13 {
14     string::iterator it;
15     string::reverse_iterator rit;
16
17     int i,t,c;
18
19     for (i=1;i<=p;i++)
20     {
21         t=0;
22         for ( it=s.begin() ; it<s.end(); it++)
23         {
24             c=( * it)-'0';
25             c=c*3+t;
26             t=c/10;
27             c=c%10;
28             ( * it)=c+'0';
29         }
30         if (t>0)
31             s += t+'0';
32     }
33
34     for ( rit=s.rbegin(); rit<s.rend(); rit++)
35         g<<( * rit);
36
37     g<<"\n";
38 }
39
40 int main()
41 {
42     int k;
43     f>>n;
44     if (n==1)
45         g<<"5\n";
46     else
47         if (n==2)
48             g<<"8\n";
49     else
50         if (n==3)
51             g<<"14\n";
52     else
53     {
```

```

54         if (n%2==0)
55             s="8";
56         else
57             s="41";
58         k=n/2;
59         s_la_p(k-1);
60     }
61 }
```

Listing 8.2.3: culori1.cpp

```

1 // Solutie pt 25pct, fara numere mari
2
3 #include <fstream>
4 #include <cmath>
5
6 using namespace std;
7
8 ifstream f("culori.in");
9 ofstream g("culori.out");
10
11 int n;
12
13 int main()
14 {
15     int k;
16     long long v=0;
17     f>>n;
18     if (n==1)
19         g<<"5\n";
20     else
21         if (n==2)
22             g<<"8\n";
23         else
24             if (n==3)
25                 g<<"14\n";
26             else
27             {
28                 if (n%2==0)
29                     v=8;
30                 else
31                     v=14;
32
33                 k=n/2;
34                 v=v*round(pow( (double) 3,k-1));
35                 g<<v<<"\n";
36             }
37     g.close();
38 }
```

Listing 8.2.4: culoriCS.cpp

```

1 // sursa prof. Cristina Sichim
2 # include <fstream>
3
4 # define dim 2000
5
6 using namespace std;
7
8 ifstream fi("culori.in");
9 ofstream fo("culori.out");
10
11 int w[2][dim],b[2][dim],r[2][dim],g[2][dim],y[2][dim],lv,lc,i,n;
12
13 void copie(int v1[],int v2[])
14 {
15     for(int i=0;i<=v2[0];i++)
16         v1[i]=v2[i];
17
18 void suma(int rez[],int v1[], int v2[])
19 {
20     int i,t=0,u=v1[0];
21     if(v2[0]>u) u=v2[0];
22     for(i=1;i<=u;i++)
23         rez[i]=v1[i]+v2[i];
24 }
```

```

23         rez[i]=(v1[i]+v2[i]+t)%10, t=(v1[i]+v2[i]+t)/10;
24     if(t)
25         rez[u+1]=t, rez[0]=u+1;
26     else
27         rez[0]=u;
28 }
29
30 void afis(int v[])
31 {
32     for(int i=v[0];i>=1;i--)
33         fo<<v[i];
34     fo<<'\n';
35 }
36
37 int main()
38 {
39     fi>>n;
40     w[1][1]=w[1][0]=b[1][1]=b[1][0]=r[1][1]=r[1][0]
41             =g[1][1]=g[1][0]=y[1][1]=y[1][0]=1;
42     lv=1;
43     for(i=2;i<=n;i++)
44     {
45         lc=1-lv;
46         copie(w[lc],b[lv]);
47         suma(b[lc],w[lv],r[lv]); //b=w+r
48         suma(r[lc],b[lv],g[lv]); //r=b+g
49         suma(g[lc],r[lv],y[lv]); //g=r+y
50         copie(y[lc],g[lv]);
51         lv=lc;
52     }
53
54     lc=1-lv;
55     suma(w[lc],w[lv],b[lv]);
56     suma(w[lv],w[lc],g[lv]);
57     suma(w[lc],w[lv],r[lv]);
58     suma(w[lv],w[lc],y[lv]);
59
60     for(i=w[lv][0];i>=1;i--)
61         fo<<w[lv][i];
62     fo<<'\n';
63     fi.close();
64     fo.close();
65     return 0;
66 }
```

Listing 8.2.5: culoriEN.cpp

```

1 /*
2      Solutie prof. Eugen Nodea
3 */
4 # include <fstream>
5
6 using namespace std;
7
8 ifstream f ("culori.in");
9 ofstream g ("culori.out");
10
11 int n, t, k, i, j, p, y, x;
12 short a[1301];
13
14 int main()
15 {
16     f>>n;
17     if (n==1)
18     {
19         g<<5<<'\n';
20         return 0;
21     }
22
23     if (n%2)
24         p = (n-3)/2, a[1] = 4, a[2] = 1, k = 2;
25     else
26         p = (n-2)/2, a[1] = 8, k = 1;
27
28     t = 0;
```

```

29     for (i=1; i<=p; ++i)
30     {
31         for (j=1; j<=k; ++j)
32         {
33             x = a[j] * 3 + t;
34             a[j] = x%10; t=x/10;
35         }
36         while (t)
37         {
38             a[++k] = t%10;
39             t = t/10;
40         }
41     }
42
43     for (i=k; i>1; --i)
44         g<<a[i];
45     g<<a[1]<<'\\n';
46
47     return 0;
48 }
```

Listing 8.2.6: culoriLI.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 typedef struct
6 {
7     int nrc;
8     int c[2001];
9 } NUMAR;
10
11 ifstream fi("culori.in");
12 ofstream fo("culori.out");
13
14 int n,i;
15 NUMAR W1,W2,R1,R2,B1,B2,G1,G2,Y1,Y2;
16 NUMAR rez,X;
17
18 void initializare(NUMAR &X)
19 {
20     int i;
21     for (i=1;i<=2000;i++)
22         X.c[i]=0;
23     X.c[1]=1;
24     X.nrc=1;
25 }
26
27 void suma(NUMAR A, NUMAR B, NUMAR &C)
28 {
29     int s,i,t;
30     t=0;
31     initializare(C);
32     C.c[1]=0;
33     for (i=1;i<=2000;i++)
34     {
35         s=t+A.c[i]+B.c[i];
36         C.c[i]=s%10;
37         t=s/10;
38     }
39     i=2000;
40     while (C.c[i]==0)
41         i--;
42     C.nrc=i;
43 }
44
45 void scrie(NUMAR X)
46 {
47     int i;
48     //fo<<X.nrc<<"\\n";
49     for (i=X.nrc;i>=1;i--)
50         fo<<X.c[i];
51     fo<<"\\n";
52 }
```

```

53
54 int main()
55 {
56     fi>>n;
57     initializare(W1);
58     initializare(R1);
59     initializare(B1);
60     initializare(G1);
61     initializare(Y1);
62     for (i=2;i<=n;i++)
63     {
64         W2=B1;
65         suma(G1,B1,R2);
66         suma(W1,R1,B2);
67         suma(R1,Y1,G2);
68         Y2=G1;
69
70         W1=W2;
71         R1=R2;
72         B1=B2;
73         G1=G2;
74         Y1=Y2;
75     }
76     suma(W1,R1,rez);
77     X=rez;
78     suma(X,B1,rez);
79     X=rez;
80     suma(X,G1,rez);
81     X=rez;
82     suma(X,Y1,rez);
83     scrie(rez);
84     fi.close();
85     fo.close();
86     return 0;
87 }
```

Listing 8.2.7: culoriLS.cpp

```

1 // sursa Liliana Schiopu
2 #include<fstream>
3
4 using namespace std;
5
6 ifstream f("culori.in");
7 ofstream g("culori.out");
8
9 int n,i,A[10000];
10 unsigned long long nr=1;
11
12 void mul(int A[], int B)
13 {
14     int i, t = 0;
15     for (i = 1; i <= A[0] || t; i++, t /= 10)
16         A[i] = (t += A[i] * B) % 10;
17     A[0] = i - 1;
18 }
19
20 int main()
21 {
22     f>>n;
23     if(n==1) g<<5;
24     else
25         if(n==2) g<<8;
26         else
27             if(n==3) g<<14;
28             else
29                 if(n%2==0)
30                 {
31                     A[1]=8;
32                     A[0]=1;
33                     for(i=1;i<=(n-2)/2;i++)
34                         mul(A,3);
35                     for(i=A[0];i>=1;i--)
36                         g<<A[i];
37     }
```

```

38         else
39             if(n%2!=0)
40             {
41                 A[1]=14;
42                 A[0]=1;
43                 for(i=1;i<=(n-3)/2;i++)
44                     mul(A,3);
45                 for(i=A[0];i>=1;i--)
46                     g<<A[i];
47             }
48         f.close();
49         g.close();
50     return 0;
51 }
```

Listing 8.2.8: culoriSC.cpp

```

1 // Sura prof. Silviu Candale
2 #include <iostream>
3 #include <iostream>
4 #include <cassert>
5
6 #define BAZA 100000000
7 #define NRC 8
8
9 //pentru debug
10 //#define TOATE true
11 //end of debug
12
13 using namespace std;
14
15 ifstream fin ("culori.in");
16 ofstream fout("culori.out");
17
18 typedef int NrMare[5000];
19
20 int n;
21
22 NrMare x;
23
24 void Atrib(NrMare x, int a)
25 {
26     x[0] = 0;
27     while(a)
28     {
29         x[++x[0]] = a % BAZA;
30         a /= BAZA;
31     }
32 }
33
34 int NrCif(int n)
35 {
36     if( n<10 )
37         return 1;
38     int r = 0 ;
39     while( n )
40         ++r, n /= 10;
41     return r;
42 }
43
44 void Produs(NrMare x, int n)
45 {
46     int t=0, tmp;
47     for(int i=1;i<=x[0];++i)
48     {
49         tmp = x[i]*n+t;
50         x[i] = tmp % BAZA;
51         t = tmp /BAZA;
52     }
53
54     while(t)
55         x [ ++x[0] ] = t % BAZA, t /= BAZA;
56 }
57
58 void Afis(ostream &out, NrMare x)
```

```
59  {
60      out << x[ x[0] ];
61      for(int i = x[0]-1 ; i ; --i)
62      {
63          int p = BAZA;
64          while(p/10>x[i])
65              out << 0, p /= 10;
66          out << x[i] ;
67      }
68      out << endl;
69  }
70
71 int main(){
72     fin >> n;
73     assert( 0 < n && n < 5001 );
74
75     #ifdef TOATE
76     for(int m = 1 ; m<=5000 ; ++m) {
77         cout << m << endl;
78         n = m;
79     #endif
80
81     if(n==1)
82     {
83         Atrib(x,5);
84     }
85     else
86     {
87         if(n%2==1)
88             Atrib(x,14), n--;
89         else
90             Atrib(x,8);
91         n = n/2 - 1;
92         for(; n ; --n)
93             Produs(x,3);
94     }
95     Afis(fout,x);
96
97     #ifdef TOATE
98     }
99     #endif
100
101    return 0;
102 }
```

Capitolul 9

OJI 2011

9.1 ai

Problema 1 - ai

100 de puncte

Institutul Național de Robotică Avansată realizează o serie de teste ultimei generații de roboți inteligenți proiectați de specialiștii acestuia. Sistemul de testare se bazează pe o rețea de senzori formată din n segmente egale dispuse orizontal și n segmente egale dispuse vertical. Distanța între două segmente alăturate orizontale, respectiv verticale este de 1 metru. Fiecare segment orizontal este în contact cu fiecare segment vertical. Denumim *nod* un punct în care un segment orizontal și unul vertical vin în contact.

Segmentele sunt numerotate: cele orizontale de sus în jos începând de la 1 iar cele verticale de la stânga la dreapta începând de la 1.

Un nod va fi identificat prin două numere: primul reprezintă numărul segmentului orizontal iar al doilea numărul segmentului vertical care vine în contact în respectivul nod.

Într-unul dintre nodurile rețelei se află o țintă. În alte două noduri se află câte o sursă ce emite o rază laser. O astfel de sursă emite raza într-o singură direcție. Raza laser are o grosime neglijabilă. Cele două surse sunt astfel orientate încât raza emisă de fiecare "lovește" ținta. Cele două noduri în care sunt plasate sursele sunt astfel alese încât cele două raze nu se intersecțează decât în nodul unde se află ținta.

În alte două noduri ale rețelei se află câte un robot. Fiecare robot se poate deplasa dintr-un nod în cele vecine (cele aflate sus, jos, în stânga și în dreapta), dar fără să iasă din cadrul rețelei. Roboții se deplasează cu 1 m/secundă.

Se efectuează experimente în care roboții sunt programați să se deplaseze prin rețea cu scopul de a proteja țintă față de cele două raze laser. Un robot poate proteja țintă fie ocupând nodul unde se află sursa, fie ocupând un nod în care trece raza laser în drumul de la sursă către țintă (razele laser nu "occolesc" roboții). Dimensiunea roboților este atât de mică încât, în acest caz, ei protejează țintă față de raza laser doar când nodurile unde sunt sursa, țintă și robotul sunt coliniare iar robotul este între sursă și țintă. În momentul în care un robot ajunge într-un nod unde protejează țintă față de una dintre raze, el se poate opri sau poate să își continue deplasarea. Dacă își continuă deplasarea astfel încât noua poziție ocupată de acel robot și pozițiile țintei și sursei nu mai sunt coliniare, atunci acel robot nu mai protejează țintă. Din modul în care sunt alese pozițiile nodurilor pentru țintă și sursele laser rezultă că nu există nicio poziție în care un robot să protejeze simultan țintă față de ambele raze.

Fiecare robot este dotat cu o rețea neuronală și poate învăța din experimentele anterioare pe unde să se deplaseze. Pentru a mări capacitatea de adaptare a roboților, în k noduri ale rețelei sunt așezate obstacole care fac ca roboții să nu poată trece prin nodurile respective. Deoarece obstacolele folosite sunt transparente, razele laser pot trece prin acestea fără a le fi afectată intensitatea sau direcția. Două sau mai multe obstacole dispuse pe același segment, în noduri alăturate, formează un zid. Lungimea unui zid este egală cu numărul de obstacole din care este alcătuit.

Cerințe

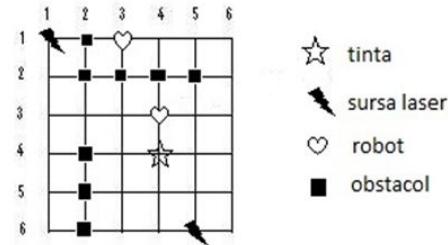


Figura 9.1: ai

- 1) Determinați lungimea maximă a unui zid.
 2) Determinați numărul minim de secunde în care cei doi roboți pot proteja ținta față de cele două raze laser.

Date de intrare

Fișierul **ai.in** conține:

- pe prima linie o valoare naturală n , reprezentând numărul segmentelor ce compun rețeaua;
- pe a doua linie cinci perechi de valori naturale separate prin câte un spațiu $T_1T_2\ S_1S_2\ S_3S_4\ R_1R_2\ R_3R_4$ cu următoarea semnificație: T_1T_2 reprezintă coordonatele nodului unde se află ținta, S_1S_2 coordonatele nodului în care este amplasată prima sursă, S_3S_4 coordonatele nodului în care este amplasată a doua sursă, R_1R_2 coordonatele poziției inițiale a primului robot, respectiv R_3R_4 coordonatele poziției inițiale a celui de-al doilea robot;
- pe a treia linie o valoare naturală k , reprezentând numărul obstacolelor din rețea;
- pe următoarele k linii se găsește câte o pereche de valori naturale separate printr-un spațiu. Fiecare prerezintă coordonatele unui nod în care este amplasat un obstacol.

Date de ieșire

Fișierul **ai.out** va conține pe prima linie un număr natural ce reprezintă răspunsul la cerința 1) iar pe a doua linie un număr natural care reprezintă răspunsul la cerința 2).

Restricții și precizări

- $n \leq 1000$
- $k \leq 150000$
- la începutul experimentului pozițiile țintei, surselor laser, roboților și obstacolelor sunt diferite.
 - roboții nu pot ocupa și nu pot trece prin nodul în care se află ținta,
 - roboții pot ocupa un nod în același timp.
 - un robot nu poate proteja ținta față de o rază decât atunci când este plasat exact într-un nod, nu și atunci când se află între două noduri.
 - un obstacol poate să aparțină în același timp atât unui zid orizontal cât și unui zid vertical.
 - dacă fișierul de ieșire conține o singură valoare, se consideră că aceasta reprezintă răspunsul la prima cerință
 - în toate testele efectuate, există cel puțin o posibilitate ca ținta să fie apărată de către una dintre raze de unul dintre roboți iar față de celalăț rază să fie apărată de celălalt robot.
 - pentru rezolvarea primei cerințe se acordă 20% din punctaj; pentru rezolvarea ambelor cerințe se acordă 100% din punctaj.

Exemple

ai.in	ai.out	Explicații
6 4 4 1 1 6 5 1 3 4 3 8 1 2 2 3 2 5 4 2 6 2 2 2 2 4 5 2	4 8	Cel mai lung zid are lungimea 4 (este cel plasat pe segmentul orizontal cu numărul 2); obstacolele de pe segmentul vertical 2 formează două ziduri cu lungimile 2 și 3. Primul robot ajunge în 4 secunde în poziția 6 5 protejând astfel ținta față de raza emisă de sursa din poziția 6 5 iar al doilea în 8 secunde în poziția 3 3 protejând astfel ținta față de raza emisă de sursa din poziția 1 1. După 8 secunde ținta e protejată față de ambele raze laser.

Timp maxim de executare/test: **1.5** secunde

Memorie: total **20 MB** din care pentru stivă **18 MB**

Dimensiune maximă a sursei: **5 KB**

9.1.1 Indicații de rezolvare

prof. Stelian Ciurea, Univ. "Lucian Blaga" Sibiu
prof. Daniela și Ovidiu Marcu, Suceava

Pentru început, se calculează și se rețin toate pozițiile în care ținta poate fi apărată față de cele două raze.

Pentru aceasta, fie se parcurge matricea prin care reprezentăm rețeaua și pentru fiecare poziție verificăm dacă este coliniară cu una dintre surse și țintă și se află între respectiva sursă și țintă, fie calculăm diferențele pe cele două coordonate între sursă și țintă - fie acestea dx și dy , calculăm $cmmdc$ -ul pentru dx și dy , iar în funcție de relația dintre coordonatele țintei și sursei avem patru cazuri;

De exemplu, dacă $x_{sursă} < x_{țintă}$ și $y_{sursă} < y_{țintă}$, atunci o poziție de coordonate x și y este coliniară cu sursa și țintă dacă $x = x_{sursă} + (dx/cmmdc)*t$, $y = y_{sursă} + (dy/cmmdc)*t$, unde $t = 0,1, \dots$ până se "ating" coordonatele țintei.

Celelalte trei cazuri se tratează similar, în loc de plus apărând după caz semnul -.

Apoi aplicăm de două ori *algoritmul Lee* cu plecarea din cele două poziții în care se află roboții.

Determinăm duratele minime în care fiecare dintre cei doi roboți ajung să apere ținta față de cele două surse.

Fie \min_{11} și \min_{12} duratele minime în care primul robot poate apăra ținta față de prima și respectiv a doua sursă.

Analog \min_{21} și \min_{22} pentru robotul 2.

Rezultatul este minimul dintre cele două combinații: robotul 1 parează sursa 1 și robotul 2 sursa 2, respectiv robotul 1 parează sursa 2 și robotul 2 sursa 1, adică rezultat = $\min(\max(\min_{11}, \min_{22}), \max(\min_{12}, \min_{21}))$ unde cu \min și \max am notat funcții care dau minimul, respectiv maximul dintre două valori.

9.1.2 *Rezolvare detaliată

9.1.3 Cod sursă

Listing 9.1.1: ai.cpp

```

1  /*
2   * Stelian Ciurea
3   * lungimea maxima in k^2
4   * Complexitate: O(n*n)
5  */
6  #include <iostream>
7  #include <fstream>
8  #include <queue>
9  #include <algorithm> // std::sort
10 #define kmax 1000000
11 #define nmax 1000
12
13 using namespace std;
14
15 struct obstacol
16 {
17     int x,y;
18 };
19
20 struct nod
21 {
22     int x,y,nrpasi;
23 };
24
25
26 obstacol obs[kmax];
27
28 int a[nmax+2][nmax+2], opt[nmax+2][nmax+2];
29 nod poz1[nmax+2], poz2[nmax+2];
30 int n, x11, x12, y11, y12, xt, yt, xr1, xr2, yr1, yr2, x, y;
31 int i,j,k,t,nrpoz1,nrpoz2,lmax,lc,kt;
32

```

```

33  ifstream fin("ai.in");
34  ofstream fout("ai.out");
35
36  int min11, min12, min21, min22;
37  int depx[4]={0,1,0,-1};
38  int depy[4]={1,0,-1,0};
39
40  int compx(obstacol a, obstacol b)
41  {
42      if (a.x > b.x)
43          return 0;
44      if (a.x == b.x && a.y > b.y)
45          return 0;
46      return 1;
47  }
48
49  int compy(obstacol a, obstacol b)
50  {
51      if (a.y > b.y)
52          return 0;
53      if (a.y == b.y && a.x > b.x)
54          return 0;
55      return 1;
56  }
57
58  void Lee(int xst, int yst)
59  {
60      queue <int> qx,qy;
61      int xu, yu, i, xc, yc;
62      qx.push(xst);
63      qy.push(yst);
64      opt[xst][yst] = 0;
65      while (!qx.empty())
66      {
67          xc = qx.front();
68          yc=qy.front();
69          qx.pop();
70          qy.pop();
71          for (i=0;i<4;i++)
72          {
73              xu = xc + depx[i];
74              yu = yc + depy[i];
75              if (a[xu][yu] == 0)
76                  if (opt[xu][yu] > opt[xc][yc] + 1)
77                  {
78                      opt[xu][yu] = opt[xc][yc] + 1;
79                      qx.push(xu);
80                      qy.push(yu);
81                  }
82          }
83      }
84  }
85
86  int ggt(int a, int b)
87  {
88      while (a!=b)
89          if (a>b) a-=b; else b-=a;
90      return a;
91  }
92
93  void calcpoz(int x11, int y11, nod poz1[], int &nrpoz)
94  {
95      int i,d,dx,dy,t=0;
96      dx = xt - x11;
97      dy = yt - y11;
98      if (dx == 0) //sunt in linie
99          if (yt < y11)
100              for (i= yt+1; i<=y11; i++)
101                  poz1[t].x = xt, poz1[t].y=i, t++;
102      else
103          for (i= y11; i<yt; i++)
104              poz1[t].x = xt, poz1[t].y=i, t++;
105      else
106          if (dy == 0) //sunt in coloana
107              if (xt < x11)
108                  for (i= xt+1; i<=x11; i++)

```

```

109             poz1[t].x = i, poz1[t].y=yt, t++;
110         else
111             for (i= x11; i<xt; i++)
112                 poz1[t].x = i, poz1[t].y=yt, t++;
113         else
114         {
115             d = ggt (abs (dx),abs (dy));
116             dx = abs (dx)/d, dy=abs (dy)/d;
117             if (x11 < xt && y11 < yt)
118                 for (t=0,x=x11,y=y11;x+dx*t<xt;t++)
119                     poz1[t].x=x+dx*t, poz1[t].y=y+dy*t;
120             if (x11 > xt && y11 > yt)
121                 for (t=0,x=x11,y=y11;x-dx*t>xt;t++)
122                     poz1[t].x=x-dx*t, poz1[t].y=y-dy*t;
123             if (x11 < xt && y11 > yt)
124                 for (t=0,x=x11,y=y11;x+dx*t<xt;t++)
125                     poz1[t].x=x+dx*t, poz1[t].y=y-dy*t;
126             if (x11 > xt && y11 < yt)
127                 for (t=0,x=x11,y=y11;x-dx*t>xt;t++)
128                     poz1[t].x=x-dx*t, poz1[t].y=y+dy*t;
129         }
130     nrpoz=t;
131 }
132
133 int main()
134 {
135     fin >> n;
136     fin >> xt >> yt >> x11 >> y11 >> x12 >> y12 >> xr1 >> yr1 >> xr2 >> yr2;
137     fin >> k;
138     for (i=1;i<=k;i++)
139     {
140         fin >> x >> y;
141         obs[i].x=x; obs[i].y=y;
142         a[x][y]=1;
143         kt++;
144     }
145 // cout << kt << endl;
146 lmax = 0;
147
148 sort (obs+1,obs+k+1,compx);
149 for (i=1;i<=k;i++)
150 {
151     //cout << obs[i].x<<" "<<obs[i].y<< " ";
152     if (obs[i].x == obs[i-1].x && obs[i].y == obs[i-1].y+1)
153         lc++;
154     else
155         lc=0;
156     if (lc > lmax)
157         lmax = lc;
158 }
159
160 sort (obs+1,obs+k+1,compy);
161 for (i=1;i<=k;i++)
162 {
163     //cout << obs[i].x<<" "<<obs[i].y<< " ";
164     if (obs[i].y == obs[i-1].y && obs[i].x == obs[i-1].x+1)
165         lc++;
166     else
167         lc=0;
168     if (lc > lmax)
169         lmax = lc;
170 }
171
172 if (lmax>=1)
173     fout << lmax+1 << '\n';
174 else
175     fout << "0\n";
176
177 calcpoz(x11,y11,poz1,nrpoz1);
178 // for (i=0;i<nrpoz1;i++)
179 //     cout << poz1[i].x<<" "<<poz1[i].y<<endl;
180 calcpoz(x12,y12,poz2,nrpoz2);
181 // for (i=0;i<nrpoz2;i++)
182 //     cout << poz2[i].x<<" "<<poz2[i].y<<endl;
183
184 for (i=0;i<=n+1;i++)

```

```

185         a[i][0] = a[0][i] = a[n+1][i] = a[i][n+1] = 1;
186         for (i=1;i<=n;i++)
187             for (j=1;j<=n;j++)
188                 opt[i][j] = n*n+1;
189
190         a[xt][yt] = 1;
191         Lee(xr1, yr1);
192         min11 = min12 = n*n + 2;
193         for (i=0;i<npoz1;i++)
194         {
195             x = poz1[i].x; y = poz1[i].y;
196             if (min11 > opt[x][y])
197                 min11 = opt[x][y];
198         }
199         for (i=0;i<npoz2;i++)
200         {
201             x = poz2[i].x; y = poz2[i].y;
202             if (min12 > opt[x][y])
203                 min12 = opt[x][y];
204         }
205
206         cout << min11 << " " << min12 << endl;
207     /*
208     for (i=1;i<=n;i++)
209     {
210         for (j=1;j<=n;j++)
211         {
212             cout.width(4);
213             cout << opt[i][j];
214         }
215         cout << endl;
216     }
217 */
218
219         Lee(xr2, yr2);
220         min21 = min22 = n*n + 2;
221         for (i=0;i<npoz1;i++)
222         {
223             x = poz1[i].x; y = poz1[i].y;
224             if (min21 > opt[x][y])
225                 min21 = opt[x][y];
226         }
227         for (i=0;i<npoz2;i++)
228         {
229             x = poz2[i].x; y = poz2[i].y;
230             if (min22 > opt[x][y])
231                 min22 = opt[x][y];
232         }
233         cout << min21 << " " << min22 << endl;
234     /*
235     for (i=1;i<=n;i++)
236     {
237         for (j=1;j<=n;j++)
238         {
239             cout.width(4);
240             cout << opt[i][j];
241         }
242         cout << endl;
243     }
244 */
245     if (max(min11,min22) < max(min12,min21))
246         fout << max(min11, min22) << endl;
247     else
248         fout << max(min12, min21) << endl;
249     fin.close();
250     fout.close();
251     return 0;
252 }
```

Listing 9.1.2: ai2.cpp

```

1  /*
2   * Stelian Ciurea
3   * lungimea maxima in k^2
4   * Complexitate: O(n*n)

```

```

5  */
6 #include <iostream>
7 #include <fstream>
8 #include <algorithm>
9 #include <queue>
10 #include <cmath>
11
12 #define kmax 1000000
13 #define nmax 1000
14
15 using namespace std;
16
17 struct obstacol
18 { int x,y;
19 };
20
21 struct nod
22 {
23     int x,y,nrpasi;
24 };
25
26 obstacol obs[kmax];
27
28 int a[nmax+2][nmax+2], opt[nmax+2][nmax+2];
29 int poz1[nmax+2], poz2[nmax+2];
30 int n, xl1, xl2, yl1, yl2, xt, yt, xr1, xr2, yr1, yr2, x, y;
31 int i,j,k,t,nrpoz1,nrpoz2,lmax,lc,kt;
32
33 ifstream fin("ai.in");
34 ofstream fout("ai.out");
35
36 int min11, min12, min21, min22;
37 int depx[4]={0,1,0,-1};
38 int depy[4]={1,0,-1,0};
39
40 int compx(obstacol a, obstacol b)
41 {
42     if (a.x > b.x)
43         return 0;
44     if (a.x == b.x && a.y > b.y)
45         return 0;
46     return 1;
47 }
48
49 int compy(obstacol a, obstacol b)
50 {
51     if (a.y > b.y)
52         return 0;
53     if (a.y == b.y && a.x > b.x)
54         return 0;
55     return 1;
56 }
57
58 void Lee(int xst, int yst)
59 {
60     queue <int> qx,qy;
61     int xu, yu, i, xc, yc, j;
62     for (i=1;i<=n;i++)
63         for (j=1;j<=n;j++)
64             opt[i][j] = n*n+1;
65     qx.push(xst);
66     qy.push(yst);
67     opt[xst][yst] = 0;
68     while (!qx.empty())
69     {
70         xc = qx.front();
71         yc=qy.front();
72         qx.pop(); qy.pop();
73         for (i=0;i<4;i++)
74         {
75             xu = xc + depx[i];
76             yu = yc + depy[i];
77             if (a[xu][yu] == 0)
78                 if (opt[xu][yu] > opt[xc][yc] + 1)
79                 {
80                     opt[xu][yu] = opt[xc][yc] + 1;

```

```

81                     qx.push(xu);
82                     qy.push(yu);
83                 }
84             }
85         }
86     }
87 }
88 int ggt(int a, int b)
89 {
90     while (a!=b)
91         if (a>b) a-=b; else b-=a;
92     return a;
93 }
94
95 void calcpoz(int x11, int y11, nod poz1[], int &nrpoz)
96 {
97     int i,d,dx,dy,t=0;
98     dx = xt - x11;
99     dy = yt - y11;
100    if (dx == 0) //sunt in linie
101        if (yt < y11)
102            for (i= yt+1; i<=y11; i++)
103                poz1[t].x = xt, poz1[t].y=i, t++;
104        else
105            for (i= y11; i<yt; i++)
106                poz1[t].x = xt, poz1[t].y=i, t++;
107    else
108        if (dy == 0) //sunt in coloana
109            if (xt < x11)
110                for (i= xt+1; i<=x11; i++)
111                    poz1[t].x = i, poz1[t].y=yt, t++;
112            else
113                for (i= x11; i<xt; i++)
114                    poz1[t].x = i, poz1[t].y=yt, t++;
115    else
116    {
117        d = ggt(abs((double)dx),abs((double)dy));
118        dx = abs((double)dx)/d, dy=abs((double)dy)/d;
119        if (x11 < xt && y11 < yt)
120            for (t=0,x=x11,y=y11;x+dx*t<xt;t++)
121                poz1[t].x=x+dx*t, poz1[t].y=y+dy*t;
122        if (x11 > xt && y11 > yt)
123            for (t=0,x=x11,y=y11;x-dx*t>xt;t++)
124                poz1[t].x=x-dx*t, poz1[t].y=y-dy*t;
125        if (x11 < xt && y11 > yt)
126            for (t=0,x=x11,y=y11;x+dx*t<xt;t++)
127                poz1[t].x=x+dx*t, poz1[t].y=y-dy*t;
128        if (x11 > xt && y11 < yt)
129            for (t=0,x=x11,y=y11;x-dx*t>xt;t++)
130                poz1[t].x=x-dx*t, poz1[t].y=y+dy*t;
131    }
132    nrpoz=t;
133 }
134
135 int main()
136 {
137     fin >> n;
138     fin >> xt >> yt >> x11 >> y11 >> x12 >> y12 >> xr1 >> yr1 >> xr2 >> yr2;
139     fin >> k;
140     for (i=1;i<=k;i++)
141     {
142         fin >> x >> y;
143         obs[i].x=x;
144         obs[i].y=y;
145         a[x][y]=1;
146         kt++;
147     }
148 //    fout << kt << endl;
149     lmax = 0;
150
151     sort(obs+1,obs+k+1,compx);
152     for (i=1;i<=k;i++)
153     {
154         //cout << obs[i].x<< " " << obs[i].y << " ";
155         if (obs[i].x == obs[i-1].x && obs[i].y == obs[i-1].y+1)
156             lc++;
}

```

```

157         else
158             lc=0;
159         if (lc > lmax)
160             lmax = lc;
161     }
162
163     sort(obs+1,obs+k+1,compy);
164     for (i=1;i<=k;i++)
165     {
166         //cout << obs[i].x<<" "<<obs[i].y<<" ";
167         if (obs[i].y == obs[i-1].y && obs[i].x == obs[i-1].x+1)
168             lc++;
169         else
170             lc=0;
171         if (lc > lmax)
172             lmax = lc;
173     }
174
175     if (lmax>=1)
176         fout << lmax+1 << '\n';
177     else
178         fout << "0\n";
179     calcpoz(x11,y11,poz1,nrpoz1);
180 //    for (i=0;i<nrpoz1;i++)
181 //        cout << poz1[i].x<<" "<<poz1[i].y<<endl;
182     calcpoz(x12,y12,poz2,nrpoz2);
183 //    for (i=0;i<nrpoz2;i++)
184 //        cout << poz2[i].x<<" "<<poz2[i].y<<endl;
185
186     for (i=0;i<=n+1;i++)
187         a[i][0] = a[0][i] = a[n+1][i] = a[i][n+1] = 1;
188
189     a[xt][yt] = 1;
190     Lee(xrl, yr1);
191     min11 = min12 = n*n + 2;
192     for (i=0;i<nrpoz1;i++)
193     {
194         x = poz1[i].x; y = poz1[i].y;
195         if (min11 > opt[x][y])
196             min11 = opt[x][y];
197     }
198     for (i=0;i<nrpoz2;i++)
199     {
200         x = poz2[i].x; y = poz2[i].y;
201         if (min12 > opt[x][y])
202             min12 = opt[x][y];
203     }
204
205     cout << min11<<" " << min12 << endl;
206     /*
207     for (i=1;i<=n;i++)
208     {
209         for (j=1;j<=n;j++)
210         {
211             cout.width(4);
212             cout << opt[i][j];
213         }
214         cout << endl;
215     }
216 */
217
218     Lee(xr2, yr2);
219     min21 = min22 = n*n + 2;
220     for (i=0;i<nrpoz1;i++)
221     {
222         x = poz1[i].x; y = poz1[i].y;
223         if (min21 > opt[x][y])
224             min21 = opt[x][y];
225     }
226     for (i=0;i<nrpoz2;i++)
227     {
228         x = poz2[i].x; y = poz2[i].y;
229         if (min22 > opt[x][y])
230             min22 = opt[x][y];
231     }
232     cout << min21<<" " << min22 << endl;

```

```

233     /*
234      for (i=1;i<=n;i++)
235      {
236          for (j=1;j<=n;j++)
237          {
238              cout.width(4);
239              cout << opt[i][j];
240          }
241          cout << endl;
242      }
243      */
244      if (max(min11,min22) < max(min12,min21))
245          fout << max(min11, min22) << endl;
246      else
247          fout << max(min12, min21) << endl;
248      fin.close();
249      fout.close();
250      return 0;
251 }
```

Listing 9.1.3: ai3.cpp

```

1  /*
2   *      Constantin Galatan
3   *      Complexitate: O(n*n)
4   */
5 #include <iostream>
6 #include <algorithm>
7 #include <queue>
8
9 using namespace std;
10
11 #define DIM 1001
12 #define pb push_back
13 #define mp make_pair
14 #define INF 0x3f3f3f3f
15
16 typedef pair<int, int> PII;
17
18 ifstream fin("ai.in");
19 ofstream fout("ai.out");
20
21 int R1[DIM][DIM];           // distantele minime ale robotului 1
22 int R2[DIM][DIM];           // distantele minime ale robotului 2
23 int t1r1 = INF, t2r1 = INF; // timpii la care robotul 1 protejeaza
24                                // tinta de cele doua surse
25 int t1r2 = INF, t2r2 = INF; // timpii la care robotul 2 protejeaza
26                                // tinta de cele doua surse
27 int n;
28 int I0, J0, is1, js1, is2, js2,    // tinta, sursele
29     ir1, jr1, ir2, jr2;           // robotii
30 const int di[] = { -1, 0, 1, 0 },   // vectori de mutare
31     dj[] = { 0, 1, 0, -1 };
32 bool obst[DIM][DIM];
33
34 queue<PII> Q;           // coada - retine nodurile in ordinea vizitarii
35
36 int robot = 1;
37 int iv, jv;
38
39 void Read();
40 int LMaxZid();
41 bool OK(int, int);
42 void Lee(int, int, int[][DIM]);
43 bool AnuleazaSursa(int, int);
44
45 int main()
46 {
47     Read();
48     fout << LMaxZid() << '\n';
49     Lee(ir1, jr1, R1);
50     robot = 2;
51     Lee(ir2, jr2, R2);
52     fout << min(max(t1r1, t2r2), max(t2r1, t1r2)) << '\n';
53     fout.close();
```

```

54     return 0;
55 }
56
57 int LMaxZid()
58 {
59     int j1, Lmax = 0, L;
60     for ( int i = 1; i <= n; ++i )
61         for ( int j = 1; j <= n; j++)
62             if ( obst[i][j] )
63             {
64                 j1 = j;
65                 while ( obst[i][j] ) j++;
66                 L = j - j1;
67                 if ( L > Lmax ) Lmax = L;
68             }
69
70     int i1;
71     for ( int j = 1; j <= n; ++j )
72         for ( int i = 1; i <= n; i++)
73             if ( obst[i][j] )
74             {
75                 i1 = i;
76                 while ( obst[i][j] ) i++;
77                 L = i - i1;
78                 if ( L > Lmax ) Lmax = L;
79             }
80     return Lmax;
81 }
82
83 void Lee(int ir, int jr, int R[][][DIM])
84 {
85     bool ok1 = false, ok2 = false;
86     R[ir][jr] = 0;
87     Q.push(mp(ir, jr));
88     int i, j;
89     while ( !Q.empty() )
90     {
91         i = Q.front().first;
92         j = Q.front().second;
93         Q.pop();
94         for ( int d = 0; d < 4; ++d )
95         {
96             iv = i + di[d];
97             jv = j + dj[d];
98             if ( OK(iv, jv) && R[i][j] + 1 < R[iv][jv])
99             {
100                 R[iv][jv] = R[i][j] + 1;
101                 Q.push(mp(iv, jv));
102
103                 // daca prima sursa e anulata
104                 if ( !ok1 && (ok1 = AnuleazaSursa(is1, js1)) )
105                 {
106                     if ( robot == 1 )
107                         t1r1 = R[iv][jv];
108                     else
109                         t1r2 = R[iv][jv];
110                 }
111
112                 // daca a doua sursa e anulata
113                 if ( !ok2 && (ok2 = AnuleazaSursa(is2, js2)) )
114                 {
115                     if ( robot == 1 )
116                         t2r1 = R[iv][jv];
117                     else
118                         t2r2 = R[iv][jv];
119                 }
120                 // daca ambele surse au fost anulate
121                 if ( ok1 && ok2 )
122                     return;
123             }
124         }
125     }
126 }
127
128 bool AnuleazaSursa(int i, int j)
129 {

```

```

130     if ( iv == i && jv == j )      return true; // pe sursa
131
132     if ( iv == i && i == IO ) // pe aceeasi orizontala sau verticala cu sursa
133     {
134         if ( j <= jv && jv < J0 ) return true; // SRT
135         if ( J0 < jv && jv<= j ) return true; // TRS
136     }
137
138     if ( jv == j && j == J0 )
139     {
140         if ( IO < iv && iv <= i ) return true; // TRS
141         if ( i <= iv && iv < IO ) return true; // SRT
142     }
143
144     if ( i > iv && iv > IO && jv > j && J0 > jv &&
145         (i - IO)*(jv - j) == (i - iv) * (J0 - j) ) // / S R T
146     return true;
147
148
149     if ( IO > iv && iv > i && jv > J0 && j > jv &&
150         (IO - i) * (jv - J0) == (IO - iv) * (j - J0) ) // / T R S
151     return true;
152
153     if ( IO > iv && iv > i && J0 > jv && jv > j &&
154         (IO - i) * (jv - j) == (J0 - j) * (iv - i) ) // \ S R T
155     return true;
156
157     if ( i > iv && iv > IO && j > jv && jv > J0 &&
158         (j - J0) * (i - iv) == (j - jv) * (i - IO) ) // \ T R S
159     return true;
160
161     return false;
162 }
163
164 bool OK(int i, int j)
165 {
166     if ( obst[i][j] ) return false;
167     if ( i == IO && j == J0 ) return false;
168     if ( i < 1 || i > n || j < 1 || j > n )
169         return false;
170     return true;
171 }
172
173 void Read()
174 {
175     fin >> n;
176     for ( int i = 1; i <= n; ++i )
177         for ( int j = 1; j <= n; ++j )
178             R1[i][j] = R2[i][j] = INF;
179
180     fin >> IO >> J0; // tinta
181     obst[IO][J0] = true; // tinta e obstacol
182     fin >> is1 >> js1 >> is2 >> js2; // sursele
183     fin >> ir1 >> jr1 >> ir2 >> jr2; // robotii
184     int x, y, K;
185     fin >> K;
186     for ( int i = 0; i < K; ++i )
187     {
188         fin >> x >> y;
189         obst[x][y] = true;
190     }
191     fin.close();
192 }

```

9.2 expresie

Problema 2 - expresie

100 de puncte

Prin convenție numim *expresie aritmetică ponderată* o expresie construită astfel:

- expresia conține numere întregi de cel mult 2 cifre despărțite prin virgulă;
- numim *k-șir* o enumerare de *k* numere despărțite prin virgulă (*k* ≥ 1);
- o expresie poate conține unul sau mai multe *k*-șiruri;
- expresia folosește paranteze rotunde și paranteze drepte.

Evaluarea expresiei se face după următoarele reguli:

- dacă expresia conține un singur k -șir atunci rezultatul expresiei este reprezentat de suma celor k numere;

Exemplu: $2, 4, 1 = 2 + 4 + 1 = 7$.

- dacă în expresie întâlnim un k -șir delimitat de paranteze rotunde rezultatul evaluării acestui k -șir va fi reprezentat de suma maximă a unui secvențe ce aparțină k -șirului, unde prin secvență se înțelege o succesiune de numere aflate pe poziții consecutive în sir;

Exemplu: $(-2, 4, -1, 3, -2, -3, 2) \Rightarrow$ secvența de sumă maximă este $4, -1, 3$ a cărui sumă este egală cu 6.

- dacă în expresie întâlnim un k -șir delimitat de paranteze pătrate, elementele k -șirului fiind numerotate $1, 2, \dots, k$, rezultatul evaluării acestui k -șir va fi reprezentat de valoarea elementului aflat pe poziția $[(k+1)/2]$ dacă sirul ar fi ordonat crescător (mediana unui sir);

Exemplu: $[-2, 9, 10, 3, 5] \Rightarrow$ sirul ordonat $[-2, 3, 5, 9, 10] \Rightarrow$ iar valoarea expresiei este egală cu 5.

- evaluarea parantezelor se face dinspre interior spre exterior.

Cerințe

Fiind dată o expresie aritmetică ponderată să se determine:

- câte numere întregi conține expresia aritmetică;
- care este valoarea expresiei aritmetice.

Date de intrare

Fișierul de intrare **expresie.in** conține pe prima linie un sir de caractere ce reprezintă o expresie aritmetică ponderată.

Date de ieșire

Fișierul de ieșire **expresie.out** va conține pe prima linie numărul de numere întregi din expresie, iar pe următoarea linie va fi scris un număr ce reprezintă valoarea expresiei aritmetice.

Restricții și precizări

- expresia se consideră corectă
- $3 \leq$ lungimea unei expresii ≤ 100000
- sirul prin care se codifică expresia poate să conțină doar următoarele caractere: cifre, paranteze rotunde și pătrate deschise și închise, caracterul virgulă, caracterul minus
 - pentru rezolvarea primei cerințe se obține 20% din valoarea fiecărui test
 - 10% dintre teste nu vor conține paranteze
 - 20% dintre teste nu vor conține paranteze imbricate

Exemple

expresie.in	expresie.out	Explicații
$2,(2,-4,1,-1,5)$	6 7	Expresia conține 6 numere întregi Valoarea expresiei este: $2+5 = 7$
$(3,-1,4),[2,3,1,8]$	7 8	$6+2$
$(2,-1,[1,2,3,4,5],-4,1)$	9 4	$(2,-1,3,-4,1) = 4$

Timp maxim de executare/test: 0.3 secunde

Memorie: total 20 MB din care pentru stivă 18 MB

Dimensiune maximă a sursei: 5 KB

9.2.1 Indicații de rezolvare

Cerință 1:

nr - numărul de numere ce se află în expresie se determină extrem de ușor acesta fiind egal cu numărul de virgule conținute de expresie +1

$$nr = nv + 1$$

Cerință 2:

Soluție 1 - folosind *stive* (Eugen Nodea)

st - stiva în care reținem *k*-sirul

r - stiva în care reținem pe ce poziție se deschide o paranteză deschisă, *kr* vârful stivei

d - stiva în care reținem pe ce poziție se deschide o paranteză dreaptă, *kd* vârful stivei

Algoritmul este următor:

- fie *k* vârful stivei *st*
- dacă întâlnim o paranteză rotundă deschisă, reținem poziția de început a unui *k*-șir delimitat de paranteze rotunde în stiva $r[+ + kr] = k + 1$
- dacă întâlnim o paranteză dreaptă deschisă, reținem poziția de început a unui *k*-șir delimitat de paranteze drepte în stiva $d[+ + kd] = k + 1$
- dacă întâlnim numerele adăugăm în stivă ($st[+ + k] = x$)
- atunci când întâlnim o paranteză rotundă închisă, vom determina secvența de sumă maximă pentru *k*-șirul de valori cuprins între indicii $r[kr], \dots, k$, și actualizăm stivele

Algoritmul de determinare a *secvenței de sumă maximă* este clasic:

- atunci când întâlnim o paranteză dreaptă închisă, vom determina mediana *k*-șirului de valori cuprins între indicii $d[kd], \dots, k$, și actualizăm stivele

x=d[kd];

y=quickselect(st,x,k,(k+x)/2);

k=x; -kd; st[k]=x;

- Pentru aflarea valorii expresiei se va face suma elementelor din stivă

quickselect() - algoritmul de determinare a *medianei unui vector* (*k*-șir) este asemănător algoritmului de sortare rapidă *quicksort*. Algoritmul alege ca "pivot" o anumită poziție, în cazul nostru poziția mediană $[(k+1)/2]$, și încercă să aducă pe aceea poziție valoarea din sirul ordonat. După execuția subprogramului elementele aflate la stânga medianei sunt mai mici, dar nu obligatoriu ordonate, iar elementele aflate în dreapta medianei sunt mai mari.

Complexitatea algoritmului de determinare a secvenței de sumă maximă este $O(kmax)$

Complexitatea algoritmului de determinare a medianei este $O(kmax)$.

Există un algoritm mai performant decât **quickselect()**, algoritmul **binmedian()**.

Referințe:

http://en.wikipedia.org/wiki/Selection_algorithm

Soluția 2 (prof. Dan Pracsiu)

Descompunem expresia în atomi, prin atomi întregi:

- număr
- paranteză rotundă deschisă
- paranteză rotundă închisă
- paranteză pătrată deschisă
- paranteză pătrată închisă

Vom utiliza o stivă de numere întregi în care se vor depune unii din atomi: numerele (care sunt cuprinse între -99 și 99), parantezele rotunde deschide (codificate prin valoarea 1000) și parantezele pătrate deschise (codificate prin valoarea 2000).

Se parcurge expresia și se completează stiva cu atomi. La identificarea unei paranteze rotunde închise, se calculează suma maximă *Smax* a secvenței de elemente care se extrag din vârful stivei până la întâlnirea valorii 1000. Această valoare *Smax* se depune în vârful stivei în locul valorii 1000. Complexitatea determinării secvenței de sumă maximă este $O(n)$, unde *n* este numărul de numere.

La identificarea în expresie a unei paranteze pătrate închise, se extrag de pe stivă toate numerele până la întâlnirea valorii 2000. Pentru aceste numere trebuie să se determine *valoarea mediană*. Pentru aceasta utilizăm o *sortare* (funcția *sort* din STL, sau *QuickSort*, sau *MergeSort*) de complexitate $O(nlogn)$. Valoarea mediană se depune apoi în vârful stivei, în locul valorii 2000.

La terminarea parcurgerii expresiei, în stivă vor rămâne doar unul sau mai multe numere cărora trebuie să li se determine suma. Această sumă reprezintă și rezultatul evaluării expresiei.

Complexitatea totală pentru această abordare este $O(n \log n)$, suficient pentru punctaj maxim.

Soluția 3 - recursivitate indirectă (Constantin Gălățan)

Expresia dată poate fi formată din una sau mai multe *subexpresii*. Acestea pot fi la rândul lor: termeni simpli (numere), sume, intervale pentru care se cere valoarea mediană, intervale pentru care se cere suma, intervale pentru care se cere subsecvența de sumă maximă sau altă subexpresie.

Se definesc funcțiile Suma(), Mediana(), Secvența().

Se parurge sirul de caractere. În funcție de valoarea primului caracter al sirului, expresia se parsează astfel:

a) Dacă primul caracter este '[' atunci expresia este un interval pentru care se va calcula valoarea mediană sau este o sumă al cărei prim termen este o mediană. În ambele cazuri se apeleză funcția Mediana() începând cu poziția următoare din sir.

Se rețin termenii medianei într-un vector pe măsură ce aceștia sunt calculați. La întâlnirea caracterului ']', funcția Mediana() returnează valoarea medianei acestui vector. Pentru calculul medianei am folosit *algoritmul nth_element()*, care determină în timp liniar poziția medianei.

b) Dacă primul caracter este '(', atunci se apeleză funcția Secvența() începând cu poziția următoare din sir.

Aceasta reține într-un vector valorile termenilor secvenței pe măsură ce aceștia sunt calculați. La întâlnirea caracterului ')' se apeleză Suma(), iar la întâlnirea unui caracter '[' se apeleză Mediana().

c) Dacă primul caracter al expresiei este '-' sau o cifră, atunci se apeleză Suma().

Această funcție apeleză la rândul său Secvența() sau Mediana() după cum caracterul curent este ')' sau ']'.

Complexitatea: $O(n)$

9.2.2 *Rezolvare detaliată

9.2.3 Cod sursă

Listing 9.2.1: expresie_rec.cpp

```

1  /*
2   *          Constantin Galatan
3   *          Recursivitate indirectă
4   *          Mediana - nth_element
5   *          Complexitate O(n)
6  */
7  #include <iostream>
8  #include <algorithm>
9  #include <vector>
10
11 using namespace std;
12
13 ifstream fin("expresie.in");
14 ofstream fout("expresie.out");
15
16 typedef vector<int> VI;
17 typedef VI::iterator IT;
18
19 string expr;
20 int n;
21 VI v, a;
22 int i, termen, semn, val, med;
23
24 int Expresie();
25 int Suma();
26 int Mediana(VI& v);
27 int Secventa(VI& v);
28 int SecvMax(IT a, IT b);
29 int Nume();
30
31 int main()
32 {
33     getline(fin, expr);
34     n = expr.length();

```

```

35
36     fout << NrNumere() << '\n';
37     fout << Expresie() << '\n';
38
39     fout.close();
40     fin.close();
41     return 0;
42 }
43
44 int NrNumere()
45 {
46     int cnt(0);
47     for ( int i = 0; i < n; ++i )
48     {
49         if ( expr[i] == '(' || expr[i] == ')' || expr[i] == ',' ||
50             expr[i] == '[' || expr[i] == ']' || expr[i] == '-' )
51             continue;
52         for (cnt++; i < n && expr[i] >='0' && expr[i] <= '9'; i++);
53     }
54     return cnt;
55 }
56
57 int Expresie()
58 {
59     int ret(0);
60     if ( expr[0] == '[' )
61     {
62         i++;
63         ret = Mediana(v);
64     }
65     else
66     if ( expr[0] == '(' )
67     {
68         i++;
69         ret = Secventa(v);
70     }
71     else
72     if ( expr[0] == '-' || (expr[0] >= '0' && expr[0] <= '9') )
73         ret = Suma();
74     if ( i < n && expr[i] == ',' )
75         ret += Suma();
76     return ret;
77 }
78
79 int Mediana(VI& v)
80 {
81     if ( i == n || expr[i] == ']' )
82     {
83         i++;
84         IT it = (v.begin() + (v.size() + 1) / 2 - 1);
85         nth_element(v.begin(), it, v.end());
86         int med = *it;
87         v.clear();
88         return med;
89     }
90
91     if ( expr[i] == '[' )
92     {
93         VI a;
94         i++;
95         med = Mediana(a);
96         v.push_back(med);
97         return Mediana(v);
98     }
99     if ( expr[i] == '(' )
100    {
101         VI a;
102         i++;
103         val = Secventa(a);
104         v.push_back(val);
105         return Mediana(v);
106     }
107     if ( expr[i] == ',' )
108     {
109         i++;
110         return Mediana(v);

```

```

111     }
112     semn = 1;
113     if ( expr[i] == '-' )
114         semn = -1, i++;
115     termen = 0;
116     for ( ;i < n && expr[i] >= '0' && expr[i] <= '9'; ++i )
117         termen = termen * 10 + expr[i] - '0';
118     v.push_back(semn*termen);
119     return Mediana(v);
120 }
121
122 int Suma()
123 {
124     if ( i == n)
125         return 0;
126     if ( expr[i] == ')' )
127         return 0;
128     if ( expr[i] == ',' )
129     {
130         i++;
131         return Suma();
132     }
133     if ( expr[i] == '(' )
134     {
135         VI a;
136         i++;
137         return Secventa(a) + Suma();
138     }
139     if ( expr[i] == '[' )
140     {
141         VI a;
142         i++;
143         return Mediana(a) + Suma();
144     }
145
146     semn = 1;
147     if ( expr[i] == '-' )
148     {
149         semn = -1;
150         i++;
151     }
152     termen = 0;
153     for ( ;expr[i] >= '0' && expr[i] <= '9'; ++i )
154         termen = termen * 10 + expr[i] - '0';
155
156     return semn * termen + Suma();
157 }
158
159
160 int SecvMax(IT a, IT b)
161 {
162     int s = *a, max = *a;
163     for ( IT it = a + 1; it != b; ++it )
164     {
165         if ( s + *it < *it )
166             s = *it;
167         else
168             s += *it;
169         if ( s > max )
170             max = s;
171     }
172     return max;
173 }
174
175 int Secventa(VI& v)
176 {
177     if ( i == n || expr[i] == ')' )
178     {
179         i++;
180         int secv = SecvMax(v.begin(), v.end());
181         v.clear();
182         return secv;
183     }
184     if ( expr[i] == ',' )
185     {
186         i++;

```

```

187         return Secventa(v);
188     }
189     if (expr[i] == '[')
190     {
191         VI a;
192         i++;
193         v.push_back(Mediana(a));
194         return Secventa(v);
195     }
196     if (expr[i] == '(')
197     {
198         VI a;
199         i++;
200         v.push_back(Secventa(a));
201         return Secventa(v);
202     }
203
204     semn = 1;
205     if (expr[i] == '-')
206         semn = -1, i++;
207     termen = 0;
208     for ( ; expr[i] >= '0' && expr[i] <= '9'; ++i)
209         termen = termen * 10 + expr[i] - '0';
210     v.push_back(semn * termen);
211
212     return Secventa(v);
213 }
```

Listing 9.2.2: expresie_stivel.cpp

```

1 /*
2  * Eugen Nodea
3  * Implementare stive
4  * Mediana - quickselect()
5  * Complexitate O(n)
6 */
7 #include <iostream>
8 #include <cstring>
9 #include <algorithm>
10
11 #define nmax 100002
12 #define swap(a,b) temp=(a); (a)=(b); (b)=temp;
13
14 using namespace std;
15
16 ifstream f("expresie.in");
17 ofstream g("expresie.out");
18
19 char ex[nmax];
20 int L,i,k,x,j,kd,kr,y,rst[nmax/2],dst[nmax/2];
21 int st[nmax],sm,Max,nr=0,S,sol;
22
23 //determinare mediana - alg. asemanator quicksort
24 int quickselect(int *a, int st, int dr, int k)
25 {
26     int i,ir,j,l,mid;
27     int p,temp;
28
29     l=st; ir=dr;
30     for(;;)
31     {
32         if (ir <= l+1)
33         {
34             if (ir == l+1 && a[ir] < a[l]) { swap(a[l],a[ir]); }
35             return a[k];
36         }
37         else {
38             mid=(l+ir) >> 1;
39             swap(a[mid],a[l+1]);
40             if (a[l] > a[ir]) { swap(a[l],a[ir]); }
41             if (a[l+1] > a[ir]) { swap(a[l+1],a[ir]); }
42             if (a[l] > a[l+1]) { swap(a[l],a[l+1]); }
43
44             i=l+1; j=ir;
45             p=a[l+1];
```

```

46         for (;;)
47         {
48             do i++; while (a[i] < p);
49             do j--; while (a[j] > p);
50             if (j < i) break;
51             swap(a[i],a[j]);
52         }
53         a[l+1]=a[j]; a[j]=p;
54         if (j >= k) ir=j-1;
55         if (j <= k) l=i;
56     }
57 }
58 }
59
60 int main()
61 {
62     f.getline(ex,nmax);
63     L=strlen(ex);
64     i=k=kd=kr=0;
65     while (i < L)
66     {
67         if (ex[i] == '(')
68         {
69             ++i; rst[++kr]=k+1;
70         }
71         if (ex[i] == '[')
72         {
73             ++i; dst[++kd]=k+1;
74         }
75         if ('0' <= ex[i] && ex[i] <= '9' || ex[i] == '-')
76         {
77             if (ex[i] == '-') sm=-1, ++i;
78             else sm=1;
79             x=0;
80             while ('0' <= ex[i] && ex[i] <= '9' && i < L)
81             {
82                 x=x*10+ex[i]-'0';
83                 ++i;
84             }
85             st[++k]=sm*x;
86             if (ex[i] == ',') ++i,++nr;
87         }
88         if (ex[i] == ')')
89         {
90             //determinam subsecventa de suma maxima
91             x=rst[kr];
92             S=st[x]; Max=S;
93             for (j=x+1; j<=k; ++j)
94             {
95                 if (S+st[j] < st[j]) S = st[j];
96                 else S += st[j];
97             }
98             k=x; --kr; st[k]=Max;
99             ++i;
100        }
101        if (ex[i] == ']')
102        {
103            //determinam mediana
104            x=dst[kd];
105            y=quicksort(st,x,k,(k+x)/2);
106            k=x; --kd; st[k]=y;
107            ++i;
108            if (ex[i] == ',') ++i,++nr;
109        }
110
111        for (i=1, sol=0; i<=k; ++i)
112        {
113            sol += st[i];
114        }
115        g<< nr+1 << '\n' << sol << '\n';
116        return 0;
117    }

```

Listing 9.2.3: expresie_stive2.cpp

```

1  /*
2   prof. Dan Pracsiu
3   Stive
4   Mediana - sortare quicksort
5   Complexitate O(n)
6 */
7 #include<iostream>
8 #include<algorithm>
9 #include <cstring>
10
11 using namespace std ;
12
13 char s[100002] ;
14 int n, k ,v;
15 int st[100001];
16
17 int SumaMax()
18 {
19     int maxim, suma;
20     maxim = st[k] ;
21     suma = st[k] > 0 ? st[k] : 0;
22     k-- ;
23     while (st[k] != 1000)
24     {
25         suma += st[k] ;
26         if (suma > maxim) maxim = suma;
27         if (suma < 0) suma = 0 ;
28         k-- ;
29     }
30     return maxim ;
31 }
32
33 int Mediana()
34 {
35     int i, nr;
36     i = k ;
37     while (st[i] != 2000) i-- ;
38     sort(st + i + 1, st + k + 1) ;
39     nr = (k + i + 1) / 2 ;
40     k = i ;
41     return st[nr] ;
42 }
43
44 int main()
45 {
46     int ii, semn, x ;
47     ifstream fin("expresie.in") ;
48     fin >> s ;
49     n = strlen(s) ;
50     fin.close() ;
51
52     k = -1;
53     for (ii=0 ; ii < n ; )
54     {
55         if (s[ii] == '(') { st[++k] = 1000; ii++; }
56         else if (s[ii] == '[') {st[++k] = 2000 ; ii++ ;}
57         else if (s[ii] == ')')
58         {
59             x = SumaMax() ;
60             st[k] = x ;
61             ii++ ;
62         }
63         else if (s[ii] == ']')
64         {
65             x = Mediana() ;
66             st[k] = x ;
67             ii++ ;
68         }
69         else if (s[ii] != ',' ) // e un numar
70         {
71             semn = 1;
72             if (s[ii] == '-')
73                 {semn = -1; ii++;}
74             x = 0 ;
75             while (ii < n && '0' <= s[ii] && s[ii] <= '9')
76             {
77                 x = x * 10 + s[ii] - '0';
78                 ii++;
79             }
80             st[k] = x ;
81             k++;
82         }
83     }
84     cout << st[0] ;
85 }
```

```
77         {
78             x = x * 10 + (s[ii] - '0') ;
79             ii++ ;
80         }
81         x = x * semn;
82         st[++k] = x ;
83     }
84     else ii++ ,v++;
85 }
86
87 int suma = 0 ;
88 for (ii=0 ; ii<=k ; ii++)
89     suma += st[ii] ;
90
91 ofstream fout("expresie.out") ;
92 fout << v+1 << "\n";
93 fout << suma << "\n";
94 fout.close() ;
95 return 0 ;
96 }
```

Capitolul 10

OJI 2010

10.1 Expoziție

Problema 1 - Expoziție

100 de puncte

Ilinca este o fetiță căreia îi place foarte mult să deseneze; ea a făcut multe desene pe care le-a numerotat de la 1 la d și apoi le-a multiplicat (toate copile poartă același număr ca și originalul după care au fost făcute). În vacanță s-a hotărât să-și deschidă propria expoziție pe gardul bunicilor care are mai multe scânduri; pe fiecare scândură ea așeză o planșă (un desen original sau o copie). Ilinca ține foarte mult la desenele ei și dorește ca fiecare desen să apară de cel puțin k ori (folosind originalul și copile acestuia).

Ilinca se întreabă în câte moduri ar putea aranja expoziția. Două moduri de aranjare sunt considerate distințe dacă diferă cel puțin prin numărul unei planșe (de exemplu: 2 1 3 3 este aceeași expoziție ca și 2 3 1 3, dar este diferită de 2 1 3 1 și de 1 3 3 1).

Cerințe

Cunoscând n numărul de scânduri din gard, d numărul desenelor originale și k numărul minim de aparții al fiecărui desen, să se determine în câte moduri poate fi aranjată expoziția, știind că Ilinca are la dispoziție oricâte copii dorește.

Date de intrare

Fișierul de intrare **expozitie.in** va conține 3 numere:

$n \ d \ k$ - numărul de scânduri, numărul desenelor originale, respectiv numărul minim de aparții

Date de ieșire

Fișierul de ieșire **expozitie.out** va conține un singur număr:

nr - numărul modurilor distințe de aranjare a expoziției

Restricții și precizări

- n, k, d sunt numere naturale
- $1 \leq n \leq 500$
- $1 \leq d \leq 500$
- $0 \leq k \leq n$

Exemple

expozitie.in	expozitie.out	Explicații
3 2 1	2	Sunt 3 scânduri, 2 desene originale și fiecare desen trebuie să apară cel puțin o dată. Există 2 moduri de aranjare. Planșele ar putea fi așezate astfel: 1 2 1 1 2 2

Timp maxim de executare/test: **0.5** secunde

Memorie: total **2 MB** din care pentru stivă **1 MB**

Dimensiune maximă a sursei: **5 KB**

10.1.1 Indicații de rezolvare

????? - ?????

Soluția 1

Deoarece fiecare desen trebuie să apară de cel puțin k ori, ne vom asigura de acest lucru afișând fiecare desen de exact k ori, au fost ocupate astfel $k * d$ scânduri, au mai rămas libere $r = n - k * d$, fiecare desen mai apare pe lângă cele k apariții de k_1, k_2, \dots, k_d ori. Dacă $r = 0$ numărul de aranjări este 1. Dacă $r < 0$ numărul de aranjări este 0. În celelalte cazuri considerăm ecuația:

$$k_1 + k_2 + k_3 + \dots + k_d = r, 0 \leq k_i \leq r \quad (1)$$

Problema se reduce la a determina numărul de soluții ale ecuației (1), acest număr este egal cu $C_{r+d-1}^r = C_{r+d-1}^{d-1}$.

Pentru demonstrație se reprezintă soluția ecuației (1) ca o secvență binară, formată din k_1 elemente 0, urmate de un element 1, urmat de k_2 elemente 0, urmate de un element 1, și.a.m.d., se adaugă în final k_d elemente 0. În secvență construită sunt r elemente 0 ($k_1 + k_2 + k_3 + \dots + k_d = r$), numărul total de elemente este $n + r - 1$. Numărul posibilităților de a aranja cele r elemente 0 este C_{r+d-1}^r .

Pentru calculul combinărilor se utilizează *triunghiul lui Pascal* care se bazează pe următoarea formulă de recurență:

$$C_m^0 = C_m^m = 1 \text{ și } C_m^n = C_{m-1}^n + C_{m-1}^{n-1}$$

Fie $m = r + d - 1$, trebuie să calculăm C_m^r .

În triunghiul lui Pascal vom utiliza *doar 2 linii*, linia curentă (*Lc*) și linia precedentă (*Lp*), deoarece rezultatul poate fi un număr foarte mare, vom implementa operațiile cu numere mari. (*exp_comb.cpp*)

Soluția 2

Problema se poate rezolva și prin *metoda programării dinamice*.

Se elimină din n cele $k * d$ scânduri pe care se afișează câte k desene de fiecare tip. Rămâne să se afișeze planșele pe $r = n - k * d$ scânduri.

Dacă $r = 0$ numărul de moduri de aranjare este 1. Dacă $r < 0$ numărul este 0. În celelalte cazuri considerăm $a[i][j] =$ numărul modurilor de aranjare dacă avem i scânduri și j desene originale; deoarece ordinea desenelor nu contează le afișăm în *odinea crescătoare* a numărului în scris pe acestea.

Este ușor de dedus că:

$a[1][j] = j$, pentru $j = 1, d$ (avem o scândură și j desene)

$a[i][1] = 1$, pentru $i = 1, n$ (avem i scânduri și 1 desen)

$a[i][j] = a[i-1][j] + a[i][j-1]$, $i = 2, n$ și $j = 2, d$ ($a[i][j-1]$ reprezintă toate posibilitățile de a afișa $j - 1$ desene pe i scânduri, trebuie adăugate toate posibilitățile de afișare care includ desenul j acest număr este $a[i-1][j]$, la cele $i - 1$ scânduri adăugăm o scândură, pe aceasta se poate afișa doar desenul j , soluțiile fără j fiind deja numărate).

Soluția se obține în $a[r][d]$.

Este suficient să memorăm *doar două coloane*, coloana curentă și coloana precedentă.

Deoarece rezultatul poate fi un *număr foarte mare*, vom implementa operațiile cu *numere mari*. (*expo_din.cpp*)

10.1.2 *Rezolvare detaliată

10.1.3 Cod sursă

Listing 10.1.1: exp_comb.cpp

```

1 #include<iostream>
2
3 #define LgNr 500
4
5 using namespace std;
6
7 typedef int BIG[LgNr];
8 int n,d,m,i,j,k,r,q;
9 BIG Lc[501],Lp[501];

```

```

10
11 void sum(BIG a, BIG b, BIG & s)
12 {
13     int i,cifra,t = 0,max;
14     if(a[0] < b[0])
15     {
16         max=b[0];
17         for(i=a[0]+1;i<=b[0];i++)
18             a[i] = 0;
19     }
20     else
21     {
22         max=a[0];
23         for(i=b[0]+1;i<=a[0];i++)
24             b[i]=0;
25     }
26
27     for(i=1;i<=max;i++)
28     {
29         cifra=a[i]+b[i]+t;
30         s[i]=cifra%10;
31         t=cifra/10;
32     }
33
34     if(t) s[i]=t; else i--;
35     s[0]=i;
36 }
37
38 int main()
39 {
40     ifstream fin("expozitie.in");
41     ofstream fout("expozitie.out");
42
43     fin>>n>>d>>k;
44     r=n-k*d;
45     if(r<0)
46         fout<<0;
47     else
48         if(r==0)
49             fout<<1;
50     else
51     {
52         m=r+d-1;
53         Lc[0][0]=Lc[0][1]=Lp[0][0]=Lp[0][1]=1;
54         for(i=1;i<=m;i++)
55         {
56             for(j=1;j<=i;j++)
57                 sum(Lp[j],Lp[j-1],Lc[j]);
58
59             for(j=0;j<=i;j++)
60                 for(q=0;q<=Lc[j][0];q++)
61                     Lp[j][q]=Lc[j][q];
62         }
63
64         for(j=Lc[r][0];j>=1;j--)
65             fout<<Lc[r][j];
66     }
67
68     fout.close();
69     return 0;
70 }
```

Listing 10.1.2: exp_din.cpp

```

1 #include<fstream>
2
3 #define LgNr 500
4
5 using namespace std;
6
7 typedef int BIG[LgNr];
8
9 int n,d,m,i,j,k,r,q,aux;
10 BIG Lc[501],Lp[501];
11
```

```

12 void sum(BIG a, BIG b, BIG & s)
13 {
14     int i,cifra,t = 0,max;
15     if(a[0] < b[0])
16     {
17         max=b[0];
18         for(i=a[0]+1;i<=b[0];i++)
19             a[i] = 0;
20     }
21     else
22     {
23         max=a[0];
24         for(i=b[0]+1;i<=a[0];i++)
25             b[i]=0;
26     }
27
28     for(i=1;i<=max;i++)
29     {
30         cifra=a[i]+b[i]+t;
31         s[i]=cifra%10;
32         t=cifra/10;
33     }
34
35     if(t)s[i]=t; else i--;
36     s[0]=i;
37 }
38
39 int main()
40 {
41     ifstream fin("expozitie.in");
42     ofstream fout("expozitie.out");
43
44     fin>>n>>d>>k;
45     r=n-k*d;
46     if(r<0)
47         fout<<0;
48     else
49         if(r==0)
50             fout<<1;
51     else
52     {
53         m=r+d-1;
54         for(i=1;i<=r;i++)
55             Lp[i][0]=Lp[i][1]=1;
56         for(i=2;i<=d;i++)
57         {
58             aux=i;
59             Lc[1][0]=0;
60             while(aux)
61             {
62                 Lc[1][0]++;
63                 Lc[1][Lc[1][0]]=aux%10;
64                 aux=aux/10;
65             }
66
67             for(j=2;j<=r;j++)
68                 sum(Lp[j],Lc[j-1],Lc[j]);
69
70             for(j=0;j<=r;j++)
71                 for(q=0;q<=Lc[j][0];q++)
72                     Lp[j][q]=Lc[j][q];
73         }
74
75         for(j=Lc[r][0];j>=1;j--)
76             fout<<Lc[r][j];
77     }
78
79     fout.close();
80     return 0;
81 }
```

10.2 Text

Problema 2 - Text

100 de puncte

Ion Petre, ca oricare adolescent, este pasionat atât de jocuri, cât și de informatică. Ultimul astfel de joc este acela de a elimina dintr-un text cuvinte astfel încât fiecare cuvânt rămas să fie urmat de un cuvânt care începe cu aceeași literă cu care se termină cuvântul precedent. Face excepție de la această regulă numai ultimul cuvânt.

Cerințe

Pentru un text dat, se cere să se afișeze numărul de cuvinte din text, apoi numărul minim de cuvinte ce pot fi eliminate astfel încât în textul rămas orice cuvânt (cu excepția ultimului) să se termine cu aceeași literă cu care începe cuvântul următor, iar în final să se afișeze cuvintele din text rămase după eliminare, fiecare cuvant fiind afișat pe câte o linie.

Date de intrare

Fișierul **text.in** conține un text scris pe mai multe linii. Pe fiecare linie se află cuvinte formate din litere mici ale alfabetului latin. Cuvintele sunt despărțite între ele prin exact câte un spațiu.

Date de ieșire

Fișierul **text.out** va conține pe primele două linii două numerele x și y , unde x va fi numărul de cuvinte din text, iar y numărul minim de cuvinte ce trebuie eliminate. Pe liniile următoare se vor afișa, în ordine, cuvintele rămase după eliminarea celor y cuvinte, câte un cuvant pe o linie.

Restricții și precizări

- Numărul de cuvinte din text este maximum 20000.
- Lungimea maximă a unui cuvânt este 20.
- Fiecare linie de text din fișierul de intrare are cel mult 200 de caractere.
- În fișier pot exista rânduri goale.
- Se acordă 10% din punctaj pentru determinarea corectă a numărului de cuvinte din text.
- Se acordă 40% din punctaj pentru rezolvarea corectă a primelor două cerințe.
- Pentru rezolvarea corectă a tuturor cerințelor se acordă tot punctajul.

Exemple

text.in	text.out	Explicații
pentru ca nu are	19	Din întregul text care este format din 19 cuvinte se elimină 13 cuvinte și se obțin, în ordine, cuvintele: ion, nu, urmareste, emisiuni, interesante, evident
timp ion spune ca nu urmaresti nici emisiuni interesante si evident nici altfel de	13 ion nu urmaresti emisiuni interesante evidenț	
emisiuni		

Timp maxim de executare/test: **0.1** secunde

Memorie: total **2 MB** din care pentru stivă **1 MB**

Dimensiune maximă a sursei: **5 KB**

10.2.1 Indicații de rezolvare

????? - ?????

Se memorează și se numără cuvintele din text.

Se calculează pentru fiecare cuvânt lungimea maximă a unui subșir de cuvinte care îndeplinește condiția din enunț.

Se determină valoarea maximă dintre lungimile calculate și se reconstituie soluția pe baza unui vector de legături de precedență.

Se poate implementa o soluție $O(n^2)$ sau $O(n * \log n)$ sau $O(n)$.

Pentru implementarea soluției liniare (optime), se rețin pentru fiecare literă, lungimea maximă a unui subșir care se termină cu litera respectivă și poziția ultimului cuvânt din acel subșir.

Solutia **text.pas** implementează algoritmul liniar.

10.2.2 *Rezolvare detaliată

10.2.3 Cod sursă

Listing 10.2.1: text.pas

```

1 const fi='text.in';fo='text.out';
2
3 var cuv:array[0..20000]of string[20];
4   l,p:array[0..20000]of integer;
5   lmax,pmax:array['a'..'z']of integer;
6   ncuv,lsol,psol:integer;
7   lit:char;
8
9 procedure citire;
10 var f:text;
11   s:string;
12   i:byte;
13 begin
14   assign(f,fi);reset(f);
15   ncuv:=0;
16   while not seeeof(f) do begin
17     readln(f,s);
18     if s=' ' then continue;
19     while s[1]=' ' do delete(s,1,1);
20     while s[length(s)]=' ' do delete(s,length(s),1);
21     i:=pos(' ',s);
22     while i><0 do begin
23       inc(ncuv);
24       cuv[ncuv]:=copy(s,1,i-1);
25       delete(s,1,i);
26       i:=pos(' ',s)
27     end;
28     inc(ncuv);cuv[ncuv]:=s
29   end;
30   close(f)
31 end;
32
33 procedure dinamica;
34 var i:integer;
35 begin
36   for i:=1 to ncuv do begin
37     lit:=cuv[i][1];
38     l[i]:=lmax[lit]+1;
39     p[i]:=pmax[lit];
40     lit:=cuv[i][length(cuv[i])];
41     if l[i]>lmax[lit] then begin
42       lmax[lit]:=l[i];
43       pmax[lit]:=i
44     end
45   end;
46   lsol:=0;
47   for lit:='a' to 'z' do
48     if lmax[lit]>lsol then begin
49       lsol:=lmax[lit];
50       psol:=pmax[lit]
51     end
52 end;
53
54 procedure scriere;
55 var f:text;
56   i:integer;
57 begin
58   assign(f,fo);rewrite(f);
59   writeln(f,ncuv,' ',ncuv-lsol);
60   i:=psol;
61   while p[i]<> 0 do begin
62     l[p[i]]:=i;
63     i:=p[i];
64   end;
65   while i<>psol do begin

```

```
66         write(f,cuv[i], ' ');
67         i:=l[i]
68     end;
69     writeln(f,cuv[psol]);
70     close(f)
71 end;
72
73 begin
74     citire;
75     dinamica;
76     scriere
77 end.
```

Capitolul 11

OJI 2009

11.1 Insule

Problema 1 - Insule

100 de puncte

Arhipelagul RGB este format din insule care aparțin țărilor R , G și B . Putem reprezenta harta arhipelagului ca o matrice cu n linii și m coloane cu elemente din mulțimea $\{0, 1, 2, 3\}$. Un element egal cu 0 reprezintă o zonă acoperită de apă; un element egal cu 1 reprezintă o zonă de pământ aparținând unei insule din țara R , iar un element egal cu 2 reprezintă o zonă de pământ aparținând unei insule din țara G , iar un element egal cu 3 reprezintă o zonă de pământ aparținând unei insule din țara B .

Se consideră că două elemente ale matricei sunt *vecine* dacă ele au aceeași valoare și fie sunt consecutive pe linie, fie sunt consecutive pe coloană. Două elemente aparțin aceleiași insule dacă ele sunt vecine sau dacă se poate ajunge de la un element la celălalt pe un drum de-a lungul căruia oricare două elemente consecutive sunt vecine.

Pentru a încuraja relațiile de colaborare dintre țările R și G , se dorește construirea unui pod care să unească o insulă aparținând țării R de o insulă aparținând țării G . Podul trebuie să respecte următoarele condiții:

- să înceapă pe o zonă cu apă consecutivă pe linie sau coloană cu o zonă aparținând țării R ;
- să se termine pe o zonă cu apă consecutivă pe linie sau coloană cu o zonă aparținând țării G ;
- să traverseze numai zone acoperite cu apă;
- oricare două elemente consecutive ale podului trebuie să fie vecine;
- lungimea podului să fie minimă (lungimea podului este egală cu numărul de elemente traversate de pod).

Cerințe

Data fiind harta arhipelagului să se determine câte insule aparțin fiecărei țări, precum și lungimea minimă a unui pod care să satisfacă condițiile din enunt.

Date de intrare

Fișierul de intrare **insule.in** conține pe prima linie numerele naturale n și m , separate prin spațiu. Pe următoarele n linii este descrisă harta arhipelagului. Pe fiecare dintre aceste n linii sunt scrise câte m valori din mulțimea $\{0, 1, 2, 3\}$; valorile nu sunt separate prin spații.

Date de ieșire

Fișierul de ieșire **insule.out** va conține o singură linie pe care vor fi scrise patru numere naturale separate prin spații $NR \ NG \ NB \ Lg$, unde NR reprezintă numărul de insule aparținând țării R , NG numărul de insule aparținând țării G , NB numărul de insule aparținând țării B , iar Lg lungimea minimă a podului.

Restricții și precizări

- $1 < n, m \leq 100$
- Se garantează că pe hartă există cel puțin un element 1, un element 2 și un element 0.
- Se acordă 40% din punctaj pentru determinarea corectă a numărului de insule din fiecare țară; se acordă punctaj integral pentru rezolvarea corectă a tuturor cerințelor.
- Începutul și sfârșitul podului pot să coincidă.
- Pentru datele de test există întotdeauna soluție.

Exemplu

insule.in	insule.out	Explicații
6 7	4 2 3 4	țara R are 4 insule, țara G are 2 insule, iar țara B are 3 insule.
1000320		Lungimea minimă a unui pod care poate fi construit este 4;
0110313		de exemplu, podul traversează celulele $(6,5)$, $(6,4)$, $(6,3)$, $(6,2)$.
3333000		
2033000		
2203011		
2000010		

Timp maxim de executare/test: 1.0 secunde

11.1.1 Indicații de rezolvare

????? - ?????

- I. Pentru de determina numărul de insule utilizăm un *algoritm de fill*.
- II. Pentru a determina lungimea minimă a podului utilizăm un *algoritm Lee* clasic. Se folosește o *coadă* în care inițial sunt plasate elemente cu valoarea 0 care au cel puțin un vecin 1.

11.1.2 *Rezolvare detaliată

11.1.3 Cod sursă

Listing 11.1.1: insule94.cpp

```

1 //Mot Nistor-Eugen - 94 puncte
2 #include <stdio.h>
3
4 #define M 151
5
6 char a[M][M],q[2][(M/4)*M];
7 int ni[4],c;
8
9 void fill(int x, int y)
10 {if(a[x][y]==c)
11   {a[x][y]=c+3;
12    fill(x-1,y); fill(x,y-1); fill(x+1,y); fill(x,y+1);}
13 }
14 int main()
15 {FILE *fi,*fo;
16  int i,j,k,m,n,t,iq,sq,sf; char ss[M];
17 fi=fopen("insule.in","rt");
18
19 fscanf(fi,"%d %d",&m,&n);
20 for(i=1;i<=m;i++) { fscanf(fi,"%s",ss);
21   for(j=1;j<=n;j++) a[i][j]=ss[j-1]-'0';
22   fclose(fi);
23
24 for(i=0;i<=m+1;i++) {a[i][0]=255; a[i][n+1]=255;}
25 for(j=0;j<=n+1;j++) {a[0][j]=255; a[m+1][j]=255;}
26
27 for(i=1;i<=m;i++)
28   for(j=1;j<=n;j++)
29     if(a[i][j]==1 || a[i][j]==2 || a[i][j]==3)
30       {c=a[i][j]; ni[c]++; fill(i,j);}
31
32 sq=0; t=0; sf=0;
33 for(i=1;i<=m;i++) for(j=1;j<=n;j++)
34   if(a[i][j]==4 &&
35     (a[i+1][j]==0 || a[i-1][j]==0 || a[i][j+1]==0 || a[i][j-1]==0))
36     {sq++; q[0][sq]=i; q[1][sq]=j;}
37
38 while (sf==0)
39   {iq=sq; t++;
40    for(k=1; k<=sq; k++)
41      
```

```

41     {i=q[0][k]; j=q[1][k];
42      if(a[i][j+1]==5 || a[i][j-1]==5 || a[i+1][j]==5 || a[i-1][j]==5)
43         {sf=2; break;}
44      if(a[i+1][j]==0) {a[i+1][j]=7; iq++; q[0][iq]=i+1; q[1][iq]=j;}
45      if(a[i-1][j]==0) {a[i-1][j]=7; iq++; q[0][iq]=i-1; q[1][iq]=j;}
46      if(a[i][j+1]==0) {a[i][j+1]=7; iq++; q[0][iq]=i; q[1][iq]=j+1;}
47      if(a[i][j-1]==0) {a[i][j-1]=7; iq++; q[0][iq]=i; q[1][iq]=j-1;}
48      if(iq==sq) {if(sf==0) {sf=1; t=0;}}
49      else {for(i=1; i<=iq-sq; i++) {q[0][i]=q[0][sq+i]; q[1][i]=q[1][sq+i];}
50          sq=iq-sq;}}
51
52 fo=fopen("insule.out","wt");
53 fprintf(fo,"%d %d %d\n",ni[1],ni[2],ni[3],t-1);
54 fclose(fo); return 1;

```

Listing 11.1.2: insule.cpp

```

1 //Emanuela Cerchez; 100 puncte
2 #include <iostream>
3 #include <string.h>
4
5 using namespace std;
6
7 #define InFile "insule.in"
8 #define OutFile "insule.out"
9 #define NMax 105
10
11 int n, m;
12 char s[NMax][NMax];
13 int dl[]={-1, 0, 1, 0};
14 int dc[]={ 0, 1, 0, -1};
15 int nr, ng, nb, lgpod;
16 struct Poz
17 {
18     int l, c;
19 } C[NMax*NMax];
20
21 void read();
22 void fill();
23 void pod();
24 void write();
25
26 int main()
27 {
28     read();
29     fill();
30     pod();
31     write();
32     return 0;
33 }
34
35 void read()
36 {
37     int i, j;
38     ifstream fin(InFile);
39     fin>>n>>m;
40     for (i=1; i<=n; i++)
41         for (j=1; j<=m; j++)
42             fin>>s[i][j];
43     fin.close();
44 }
45
46 void write()
47 {
48     ofstream fout(OutFile);
49     fout<<nr<<' '<<ng<<' '<<nb<<' '<<lgpod<<'\n';
50     fout.close();
51 }
52
53 void sterge(int i, int j, char c)
54 {
55     int k;
56     if (s[i][j]==c)
57     {
58         s[i][j]=c+3;

```

```

59         for (k=0; k<4; k++)
60             sterge(i+dl[k], j+dc[k], c);
61     }
62 }
63
64 void fill()
65 {
66     int i, j;
67     for (i=1; i<=n; i++)
68         for (j=1; j<=m; j++)
69             if (s[i][j]=='1')
70             {
71                 sterge(i, j, '1');
72                 nr++;
73             }
74             else
75                 if (s[i][j]=='2')
76                 {
77                     sterge(i, j, '2');
78                     ng++;
79                 }
80             else
81                 if (s[i][j]=='3')
82                 {
83                     sterge(i, j, '3');
84                     nb++;
85                 }
86 }
87
88 int vecin(int i, int j, char c)
89 {
90     int k;
91     for (k=0; k<4; k++)
92         if (s[i+dl[k]][j+dc[k]]==c)
93             return 1;
94     return 0;
95 }
96
97 void pod()
98 {
99     int d[NMax][NMax];
100    Poz x, y;
101
102    int i, j, k;
103    for (i=1; i<=n; i++)
104        for (j=1; j<=m; j++)
105            d[i][j]=0;
106
107    for (i=0; i<=n+1; i++)
108        d[i][0]=d[i][m+1]=NMax*NMax;
109
110    for (i=0; i<=m+1; i++)
111        d[0][i]=d[n+1][i]=NMax*NMax;
112
113    int inc=0, sf=-1;
114    //initializam coada cu celule vecine cu insule 1
115    for (i=1; i<=n; i++)
116        for (j=1; j<=m; j++)
117            if (s[i][j]=='0') //apa
118                if (vecin(i, j, '4'))
119                {
120                    C[++sf].l=i;
121                    C[sf].c=j;
122                    d[i][j]=1;
123                }
124
125    lgpod=NMax*NMax;
126
127    while (inc<=sf)
128    {
129        x=C[inc++];
130        for (k=0; k<4; k++)
131        {
132            y.l=x.l+dl[k];
133            y.c=x.c+dc[k];
134            if (d[y.l][y.c]==0 && s[y.l][y.c]=='0')

```

```

135      {
136          d[y.l][y.c]=1+d[x.l][x.c];
137          C[++sf]=y;
138          if (vecin(y.l,y.c,'5'))
139              if (d[y.l][y.c]<lgpod)
140                  lgpod=d[y.l][y.c];
141      }
142  }
143 }
144 }
```

11.2 Rețetă

Problema 2 - Rețetă

100 de puncte

Mama mea este profesoară de informatică, dar îi place foarte mult să gătească. Recent am descoperit caietul ei de rețete, care arată foarte neobișnuit. Fiecare rețetă este scrisă pe un singur rând pe care sunt precizate produsele folosite, cantitățile, precum și ordinea în care se execută operațiile. De exemplu:

(unt 50 zahar 250 ou 4) 5

ceea ce înseamnă că se amestecă 50 grame unt cu 250 grame zahăr și cu 4 ouă timp de 5 minute.

Pentru fiecare produs mama folosește întotdeauna aceeași unitate de măsură, așa că unitățile de măsură nu mai sunt precizate. Numele produsului este scris întotdeauna cu litere mici, iar produsele și cantitățile sunt separate prin spații (unul sau mai multe). Produsele care se amestecă împreună sunt încadrate între paranteze rotunde; după paranteza rotundă închisă este specificat timpul de preparare.

Evident, mama are și rețete mai complicate:

((zahar 100 ou 3)5 unt 100 nuca 200)4 (lapte 200 cacao 50 zahar 100) 3)20

Să traducem această rețetă: se amestecă 100 grame zahăr cu 3 ouă timp de cinci minute; apoi se adaugă 100 grame unt și 200 grame nucă, amestecând totul încă 4 minute. Se amestecă 200 ml lapte cu 50 grame de cacao și 100 grame zahăr timp de 3 minute, apoi se toarnă peste compozitia precedentă și se amestecă totul timp de 20 minute.

Observați că înainte sau după parantezele rotunde pot să apară sau nu spații.

Cerințe

Dată fiind o rețetă să se determine timpul total de preparare, precum și cantitățile necesare din fiecare produs.

Date de intrare

Fișierul de intrare **reteta.in** conține pe prima linie un sir de caractere care reprezintă o rețetă.

Date de ieșire

Fișierul de ieșire **reteta.out** va conține pe prima linie timpul total necesar pentru prepararea rețetei. Pe următoarele linii sunt scrise ingredientele în ordine lexicografică (ordinea din dicționar), câte un ingredient pe o linie. Pentru fiecare ingredient este specificat numele urmat de un spațiu apoi de cantitatea totală necesară.

Restricții și precizări

- $0 < \text{Lungimea unei rețete} \leq 1000$
- $1 \leq \text{Numărul de ingrediente} \leq 100$
- Numele unui ingredient este scris cu maxim 20 litere mici ale alfabetului englez.
- Timpii de preparare sunt numere naturale < 100
- Cantitățile specificate în rețete sunt numere naturale < 1000
- Pentru determinarea corectă a timpului total se acordă 30% din punctajul pe test; pentru determinarea corectă a timpului total și afișarea corectă a ingredientelor (ordonate lexicografic) se acordă integral punctajul pe test.

Exemple

reteta.in	reteta.out
((zahar 100 ou 3)5 unt 100 nuca 200)4 (lapte 200 cacao 50 zahar 100) 3)20	32 cacao 5 lapte 200 nuca 200 ou 3 unt 100 zahar 200

Timp maxim de executare/test: **1.0** secunde

11.2.1 Indicații de rezolvare

????? - ?????

Pasul I

Primul pas este să determinăm timpul total de preparare. În acest scop vom însuma numere care apar imediat după o paranteză închisă

Mai exact:

cât timp mai există o paranteză închisă:

- determin numărul următor
- îl însumez cu timpul total
- elimin din sir acest număr și paranteza) care îl precedă.

După primul pas sirul conține secvențe de tipul produs cantitate separatorii existenți fiind spații și paranteze (.

Vom defini *structura Produs* în care reținem *numele* produsului precum și *cantitatea* totală din produsul respectiv.

Vom declara un vector cu componente de tip *Produs* în care reținem produsele în ordinea alfabetică.

Pasul II

Cât timp mai există produse

- extragem un produs
- extragem numărul natural care urmează
- caut produsul respectiv în lista de produse (dacă nu găsim produsul respectiv, îl inserez în vectorul de produse pe poziția corectă, astfel încât vectorul să rămână sortat)
- însumez cantitatea de produs extrasă la cantitatea totală din produsul respectiv.

11.2.2 *Rezolvare detaliată

11.2.3 Cod sursă

Listing 11.2.1: RETETA.cpp

```

1 //prof. Emanuela Cerchez; 100 puncte
2 #include <iostream>
3 #include <string.h>
4 #include <stdlib.h>
5
6 using namespace std;
7
8 #define InFile "reteta.in"
9 #define OutFile "reteta.out"
10#define LgMax 1001
11#define NrMax 100
12#define LgP 21
13
14 int nr;           //numarul de produse
15 int n;            //lungime reteta
16 char s[LgMax];   //reteta

```

```

17 long int timp;
18
19 struct Produs
20 {
21     char nume[LgP];
22     long int cant;
23 } P[NrMax]; //produsele in ordine alfabetica
24
25 void read();
26 long int det_timp();
27 void cantitati();
28 void write();
29 int cauta(char *);
30
31 int main()
32 {
33     read();
34     timp=det_timp();
35     cantitati();
36     write();
37     return 0;
38 }
39
40 void read()
41 {
42     int i, nr, d, j;
43     ifstream fin (InFile);
44     fin.getline(s,LgMax-1);
45     n=strlen(s);
46     fin.close();
47 }
48
49 void write()
50 {
51     int i;
52     ofstream fout(OutFile);
53     fout<<timp<<'\n';
54     for (i=0; i<nr; i++)
55         fout<<P[i].nume<<' ' <<P[i].cant<<'\n';
56     fout.close();
57 }
58
59 long int det_timp()
60 // pentru a determina timpul total insumez numerele aflate
61 // dupa parantezele inchise
62 // elimin apoi aceste numere din sir
63 {
64     char *p=s, *q;
65     int a;
66     long int timp=0;
67     while (*p)
68     {
69         p=strchr(s, ')');
70         if (p)
71         {
72             q=p+1;
73
74             //sau eventualele spatii
75             while (q[0]==' ') q++;
76
77             //identific numarul
78             a=q[0]-'0';
79             if (q[1]>='0' && q[1] <='9')
80             {
81                 a=a*10+q[1]-'0';
82                 q++;
83             }
84             timp+=a;
85
86             //sterge din sir caracterele de la p+1 la q inclusiv
87             *p=NULL;
88             strcat(s,q+1);
89         }
90     }
91
92     return timp;

```

```

93 }
94
95 void cantitati()
96 {
97     char *p=s;
98     int poz;
99     p=strtok(s, " ()");
100    while (p)
101    {
102        //p indica numele unui produs
103        poz=cauta(p);
104
105        //extrag cantitatea
106        p=strtok(NULL, " ()");
107        P[poz].cant+=atoi(p);
108
109        //extrag urmatorul produs
110        p=strtok(NULL, " ()");
111    }
112 }
113
114 int cauta(char * x)
115 //cauta secvential produsul x in vectorul de produse
116 //daca produsul este gasit se returneaza pozitia sa
117 //daca produsul nu este gasit se insereaza in vectorul de produse,
118 //astfel incat vectorul de produse sa ramana sortat.
119 {
120     int i, j;
121     for (i=0; i<nr && strcmp(P[i].nume,x)<0; i++);
122     if (strcmp(x, P[i].nume)==0) return i;
123
124     //deplasez elementele de la pozitia i la nr-1 cu o pozitie la stanga
125     for (j=nr; j>i; j--) P[j]=P[j-1];
126     nr++;
127     strcpy(P[i].nume,x);
128     P[i].cant=0;
129     return i;
130 }

```

Listing 11.2.2: RETETAV.cpp

```

1 //prof. Ilie Vieru 100 puncte
2 #include<iostream>
3 #include<string.h>
4 #include<stdlib.h>
5
6 using namespace std;
7
8 ifstream f("reteta.in");
9 ofstream g("reteta.out");
10
11 struct nod
12 {
13     char nume[21];
14     long cant;
15 } v[300];
16
17 char s[10500],*p,*q,*r;
18 int su,cant,n;
19
20 int fcomp(const void **x, const void *y)
21 {
22     nod a=*((nod*)x), b=*((nod*)y);
23     return strcmp(a.nume,b.nume);
24 }
25
26 int main()
27 {
28     f.getline(s,1002);
29     strcat(s, " ");
30     while(p=strchr(s, '('))
31         if( *(p+1)=='(')
32             strcpy(p,p+1);
33         else
34             *p=' ';

```

```

35     p=s;
36     while(p=strchr(p,' '))
37     {
38         while( *(p+1)==' ')
39             strcpy(p+1,p+2);
40
41         q=strchr(p,' ');
42         r=strchr(p+1,' ');
43         if(r && r<q) q=r;
44         *q=0;
45         su+=atoi(p+1);
46         if(r==q)
47             *p=' ';
48         else
49             *p=' ';
50         strcpy(p+1,q+1);
51     }
52
53     g<<su<<' \n';
54     int i;
55     p=strtok(s+1," ");
56     while(p)
57     {
58         i=0;
59         while(i<n && strcmp(v[i].nume,p))
60             i++;
61         if(strcmp(v[i].nume,p))
62         {
63             strcpy(v[n].nume,p);
64             i=n;
65             n++;
66         }
67
68         p=strtok(NULL," ");
69         v[i].cant+=atoi(p);
70         p=strtok(NULL," ");
71     }
72
73     qsort(v,n,sizeof(nod), fcomp);
74
75     for(i=0;i<n;i++)
76         g<<v[i].nume<<" "<<v[i].cant<<' \n';
77
78     f.close();
79     g.close();
80     return 0;
81 }
```

Capitolul 12

OJI 2008

12.1 Colaj

Problema 1 - Colaj

100 de puncte

La etapa finală a *Concursului pe Echipe al Micilor Artiști*, pe primul loc s-au clasat două echipe A și B , cu același punctaj. Comisia de Evaluare, pentru a le departaja, a introdus o nouă probă de baraj care vizează atât talentul copiilor, cât și iștețimea lor.

Astfel, echipa A trebuie să realizeze un colaj alb-negru având la dispoziție o planșă dreptunghiulară de culoare albă și n dreptunghiuri de culoare neagră. Membrii acestei echipe vor trebui să lipească pe planșă toate dreptunghiurile, cu laturile paralele cu laturile planșei.

Pot exista și dreptunghiuri lipite în interiorul altui dreptunghi, sau dreptunghiuri care se intersectează, sau dreptunghiuri cu laturi pe laturile planșei, precum și suprafete din planșă neacoperite cu dreptunghiuri.

După ce așează toate dreptunghiurile, membrii echipei A trebuie să comunice echipei B numărul n de dreptunghiuri negre primite, lungimea m a laturii orizontale a planșei, lungimea p a laturii verticale a planșei, și coordonatele vârfurilor din stânga-jos și dreapta-sus ale fiecărui dreptunghi de pe planșă (coordonate referitoare la reperul cartezian xOy cu originea O în colțul din stânga-jos a planșei și cu axa de coordonate Ox , respectiv Oy , pe dreapta suport a laturii orizontale, respectiv a laturii verticale a planșei).

Pentru a câștiga concursul, echipa B trebuie să ghicească numărul zonelor continue maximale de culoare albă, conținute de colajul realizat de echipa A .

O zonă albă este considerată continuă dacă oricare ar fi două puncte P, Q din zona respectivă, se poate uni punctul P de punctul Q printr-o linie dreaptă sau frântă care să nu intersecteze interiorul nici unui dreptunghi negru. O zonă albă continuă este considerată maximală dacă nu există o altă zonă albă continuă de arie mai mare care să includă zona respectivă.

Cerințe

Scrieți un program care să citească numărul n al dreptunghiurilor negre primite de echipa A , lungimile m și p ale laturilor planșei, coordonatele vârfurilor din stânga-jos și dreapta-sus ale fiecărui dreptunghi negru primit, și care să determine numărul zonelor continue maximale de culoare albă existente în colajul realizat de echipa A .

Date de intrare

Fișierul de intrare **colaj.in** conține:

n
$m \ p$
$a_1 \ b_1 \ c_1 \ d_1$
...
$a_n \ b_n \ c_n \ d_n$

- pe prima linie, o valoare naturală n , reprezentând numărul de dreptunghiuri negre date echipei A

- pe a doua linie, 2 numere naturale, m și p , separate prin spațiu, reprezentând lungimile laturilor planșei

- următoarele n linii conțin câte patru numere naturale, separate prin câte un spațiu, care reprezintă coordonatele (a_1, b_1) și (c_1, d_1) ale vârfurilor din stânga-jos și dreapta-sus ale primului dreptunghi, ..., coordonatele (a_n, b_n) și (c_n, d_n) ale vârfurilor din stânga-jos și dreapta-sus ale celui de-al n -lea dreptunghi.

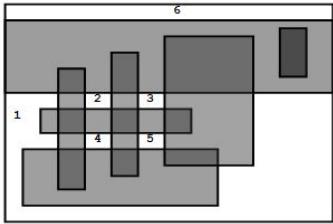
Date de ieșire

Fișierul de ieșire **colaj.out** va conține o singură linie pe care se va scrie un singur număr natural reprezentând numărul zonelor continue maximale de culoare albă, conținute de colaj.

Restricții și precizări

- $1 \leq n \leq 100$,
- $a_1 < c_1 \leq m, a_2 < c_2 \leq m, \dots, a_n < c_n \leq m$,
- $b_1 < d_1 \leq p, b_2 < d_2 \leq p, \dots, b_n < d_n \leq p$
- Toate coordonatele vârfurilor dreptunghiurilor și lungimile laturilor planșei sunt numere naturale, $0 < m, p < 8000$
 - Dacă (x, y) și (z, t) sunt coordonatele a două vârfuri din două dreptunghiuri distincte, atunci: $x \neq z$ și $y \neq t$.
 - în 40% din teste: $n < 30, m \leq 180, p \leq 180$;
 - în 40% din teste: $70 \leq n \leq 100, 180 < m < 1000, 180 < p < 1000$;
 - în 20% din teste: $50 < n < 80, 7000 < m < 8000, 7000 < p < 8000$

Exemple

colaj.in	colaj.out	Explicații
7 17 16 1 1 10 5 2 6 8 8 0 9 17 15 3 2 4 11 5 3 6 12 7 4 12 13 14 10 16 14	6	<p>$n = 7, m = 17, p = 16$. Sunt 7 dreptunghiuri negre. Colajul realizat de echipa A este cel din desenul alăturat. Se observă 6 zone continue maximale de culoare albă conținute de colaj (cele numerate în figura alăturată).</p> 

Timp maxim de executare/test: **1.0** secunde

12.1.1 Indicații de rezolvare

Se consideră următorul exemplu

prof. Carmen MINCĂ

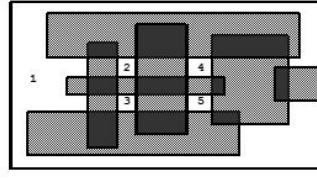
colaj.in	colaj.out	Explicație
7 25 15 1 1 16 5 14 4 21 12 4 7 15 8 19 6 25 9 2 10 23 14 5 2 6 11 8 3 12 13	5	<p>Colajul realizat de echipa A este cel din desenul alăturat. Se observă 5 suprafețe albe distincte conținute de colaj.</p> 

Figura 12.2: colaj

O soluție se poate obține pe baza metodei împărțirii în "zone elementare", care nu se intersectează cu nici un dreptunghi. Pentru exemplul dat, se obține următoarea împărțire în zone elementare:

Sunt reprezentate și vârfurile planșei: $(0; 0)$, $(25; 0)$, $(0, 15)$ și $(25; 15)$. Se observă că aceste zone elementare formează o matrice A având un număr de linii, respectiv coloane, din multimea $\{2n - 1, 2n, 2n + 1\}$.

Numărul minim de linii $2n - 1$, respectiv coloane $2n - 1$, se obține dacă există dreptunghiuri cu laturile pe marginile de jos și sus, respectiv din stânga și dreapta, ale planșei. Numărul de linii, respectiv coloane, crește cu 1 dacă niciun dreptunghi nu este situat pe marginea de jos, respectiv din stânga, a planșei.

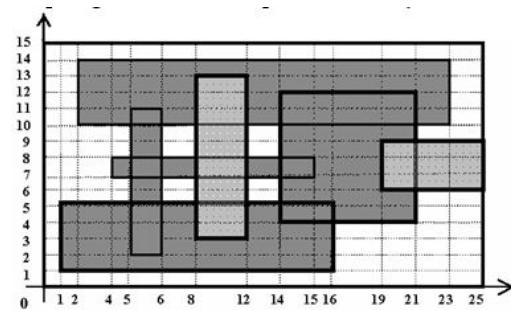


Figura 12.3: colaj

Numărul de linii, respectiv coloane, crește cu 1 dacă niciun dreptunghi nu este situat pe marginea de sus, respectiv din dreapta, a planșei.

Fie x_0 , respectiv y_0 , valoarea care se adaugă la $2n - 1$ pentru a se obține numărul de coloane, respectiv linii, ale matricei A . Pentru matricea din exercițiul 1:

Elementul $A[i][j]$ al matricei va avea valoarea 1 dacă și numai dacă există cel puțin un dreptunghi care să conțină zona elementară respectivă.

Matricea este inversată față de axa Ox .

Pentru exemplul dat, matricea A are $2n + 1$ linii și $2n$ coloane, iar conținutul ei este cel alăturat.

Pentru construcția matricei vom folosi doi vectori X și Y . Vectorul X va reține toate abscisele vârfurilor dreptunghiurilor. Vectorul Y va reține toate ordonatele vârfurilor dreptunghiurilor.

Zonele elementare din desen corespund câte unui element $A[i][j]$ din matrice și sunt dreptunghiuri care au coordonatele vârfurilor opuse: $(X[i], Y[j])$ și $(X[i+1], Y[j+1])$, $i, j = 1, 2, \dots, 2n$.

Dacă nu există dreptunghiuri incluse în interiorul unui alt dreptunghi, atunci vectorii au câte $2n$ componente, fiecare având toate valorile distincte. Altfel sunt memorate coordonatele dreptunghiurilor care sunt incluse într-un alt dreptunghi.

Se sortează crescător vectorii. Pentru exemplul dat, vectorii vor avea conținutul:

$$X=(1,2,4,5,6,8,12,14,15,16,19,21,23)$$

$$Y=(1,2,3,4,5,6,7,8,9,10,11,12,13,14)$$

Se memorează coordonatele vârfurilor dreptunghiurilor din exemplul dat într-un vector V cu n elemente de tip structură:

```
struct dr { int a,b,c,d; int pa,pb,pc, pd; };
```

unde:

1. (a,b) sunt coordonatele vârfului stânga-jos al dreptunghiului
2. (c,d) sunt coordonatele vârfului dreapta-sus al dreptunghiului
3. pa este poziția pe care apare a în vectorul sortat X
4. pb este poziția pe care apare b în vectorul sortat Y
5. pc este poziția pe care apare c în vectorul sortat X
6. pd este poziția pe care apare d în vectorul sortat Y

Se inițializează matricea A cu 0. Pentru fiecare dreptunghi, se marchează cu 1 toate zonele acoperite de acesta:

```
for(k=1;k<=n;k++)
    for(j=v[k].pa;j<v[k].pc;j++)
        for(i=v[k].pb;i<v[k].pd;i++) a[i+y0][j+x0]=1;
```

Se bordează matricea cu 1, pentru a nu ieși în exteriorul ei în timpul aplicării *algoritmului FILL*. Se caută fiecare element din matrice cu valoarea 0, se umple atât elementul cât și vecinii acestuia cu valoarea 1 și se numără zonele care au valoarea 0. Acest număr va fi numărul de suprafețe albe din colaj.

După aplicarea FILL-ului matricea A din exemplu va avea toate valorile egale cu 1.

Sursele *colajC.cpp* și *colajP.pas* constituie implementarea a acestei soluții.

O soluție care obține un punctaj parțial constă în a contrui o matrice A cu p linii și m coloane, elementele ei memorând valori 0 și 1.

Fiecăruia dreptunghi cu vârful stânga-jos, respectiv dreapta-sus, de coordonate (xa, ya) , respectiv (xb, yb) , îi corespunde în matricea A o zonă în care toate elementele au valoarea 1:

```
for(l=ya;l<yb;l++)
    for(c=xa;c<xb;c++) a[l][c]=1;
```

Pornind de la primul element din matrice cu valoarea 0, aplicând algoritmului FILL, umplim atât elementul cât și vecinii acestuia cu valoarea 1 și numărăm zonele care au valoarea 0. Acest număr va fi numărul de suprafețe albe din colaj.

Dezavantajul utilizării acestei metode constă în spațiul de memorie disponibil, insuficient pentru memorarea matricei A . Nu întotdeauna este posibilă memorarea unei matrice cu $m \times p$ ($m, p < 8000$) și cu elemente de tip *char/byte*. Prin "comprimarea" dreptunghiurilor utilizând

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	0	0	1	0	1	0	1	1	1	1	1	0	0	0
0	0	0	0	1	0	1	0	1	1	1	1	1	1	1	1
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	1	0	1	0	1	1	1	1	1	1	1	1
0	0	0	0	1	0	1	0	1	1	1	1	1	1	1	1
0	0	0	0	1	0	1	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 12.4: colaj

.

metoda împărțirii în "zone elementare", se reduce spațiul necesar memorării matricei de tip *char* la cel mult 201×201 octeti.

Un alt dezavantaj rezultă din timpul mare de executare, datorat aplicării *algoritmului recursiv FILL* pentru o matrice cu un număr mare de componente, și a dimensiunii mici a memoriei corespunzătoare *segmentului de stivă*. (vezi sursa *colaj_40.cpp* care obține 40 p).

O îmbunătățire a punctajului se poate obține atunci când valorile m și p permit declararea unei matrice A de tip *char* care să memoreze toate zonele corespunzătoare dreptunghiurilor, prin eliminarea anumitor linii și coloane. Necesitând un număr mare de operații datorate ștergerilor liniilor, respectiv coloanelor, identice din A se depășește timpul de execuție.

Se elimină anumite linii ale matricei A astfel: dacă liniile l_i, l_{i+1}, \dots, l_k din A sunt identice, reținem linia l_i , restul fiind șterse, nefiind necesare, realizându-se astfel o *comprimare* pe linii a matricei A , fiecare grupă de linii identice fiind înlocuită cu o singură linie de acest tip.

Analog, dacă coloanele c_i, c_{i+1}, \dots, c_k din A sunt identice, atunci se păstrează în A doar coloana c_i , restul se șterg nefiind necesare, realizând astfel o *comprimare* pe coloane a matricei A , pentru fiecare grupă de coloane identice păstrându-se în A o singură coloană de acest tip. (vezi sursa *colaj_50.cpp* care obține 50 p).

12.1.2 *Rezolvare detaliată

12.1.3 Cod sursă

Listing 12.1.1: COLAJ_40.cpp

```

1 #include <stdio.h>
2 #include <conio.h>
3 #include <iostream>
4
5 using namespace std;
6
7 int m,p,nl,nc,nr;
8 int n;
9 char a[230][230];
10 FILE *f,*g;
11
12 void citire() //citire date
13 {
14     f=fopen("colaj.in","r");
15     fscanf(f,"%d%d%d",&n,&m,&p);
16     nl=p-1;
17     nc=m-1;
18     int i,j;
19     for(i=0;i<=nl;i++)
20         for(j=0;j<=nc;j++)
21             a[i][j]=0;
22     int xa,ya,xb,yb,l,c;
23     for(i=1;i<=n;i++)
24     { fscanf(f,"%d%d%d%d",&xa,&ya,&xb,&yb);
25         for(l=ya;l<yb;l++)
26             for(c=xa;c<xb;c++)
27                 a[l][c]=1;
28     }
29     fclose(f);
30 }
31
32 void fill(int i,int j)
33 {
34     a[i][j]=1;
35     if((j>0)&&(a[i][j-1]==0))fill(i,j-1);
36     if((j<nc)&&(a[i][j+1]==0))fill(i,j+1);
37     if((i>0)&&(a[i-1][j]==0))fill(i-1,j);
38     if((i<nl)&&(a[i+1][j]==0))fill(i+1,j);
39
40 }
41
42 void supr()
43 { int i,j;
```

```

44     for(i=0;i<=nl;i++)
45         for(j=0;j<=nc;j++)
46             if(a[i][j]==0) {nr++;fill(i,j);}
47     }
48
49 int main()
50 { citire();
51   supr();
52   g=fopen("colaj.out","w");
53   fprintf(g,"%d",nr);
54   fclose(g);
55   return 0;
56 }
```

Listing 12.1.2: Colaj_50.cpp

```

1 #include <stdio.h>
2
3 int m,p,nl,nc,nr;
4 int n;
5 char a[240][240];
6 FILE *f,*g;
7
8 void citire() //citire date
9 {
10    f=fopen("colaj.in","r");
11    fscanf(f,"%d%d%d",&n,&m,&p);
12    nl=p-1;
13    nc=m-1;
14    int i,j;
15    for(i=0;i<=nl;i++)
16        for(j=0;j<=nc;j++)
17            a[i][j]=0;
18    int xa,ya,xb,yb,l,c;
19    for(i=1;i<=n;i++)
20        { fscanf(f,"%d%d%d%d",&xa,&ya,&xb,&yb);
21          for(l=ya;l<yb;l++)
22              for(c=xa;c<xb;c++)
23                  a[l][c]=1;
24        }
25    fclose(f);
26 }
27
28 int ver_l(int l)
29 { for(int j=0; j<=nc; j++)
30   if(a[l-1][j]!=a[l][j]) return 0;
31   return 1;
32 }
33
34 void sterg_l(int l)
35 { int i,j;
36   for(i=l;i<nl;i++)
37     for(j=0;j<=nc;j++)
38       a[i][j]=a[i+1][j];
39   nl--;
40 }
41
42 void linii()
43 {
44   int i=1;
45   while(i<=nl)
46   { if(ver_l(i)) sterg_l(i);
47     else i++; }
48 }
49
50 int ver_c(int c)
51 { for(int j=0; j<=nl; j++)
52   if(a[j][c-1]!=a[j][c]) return 0;
53   return 1;
54 }
55
56 void sterg_c(int c)
57 { int i,j;
58   for(j=c;j<nc;j++)
59     for(i=0;i<=nl;i++)
```

```

60     a[i][j]=a[i][j+1];
61     nc--;
62 }
63
64 void coloane()
65 { int j=1;
66   while(j<=nc)
67   { if(ver_c(j)) sterg_c(j);
68   else j++;}
69 }
70
71 void fill(int i,int j)
72 {
73     a[i][j]=1;
74     if((j>0)&&(a[i][j-1]==0))fill(i,j-1);
75     if((j<nc)&&(a[i][j+1]==0))fill(i,j+1);
76     if((i>0)&&(a[i-1][j]==0))fill(i-1,j);
77     if((i<nl)&&(a[i+1][j]==0))fill(i+1,j);
78 }
79
80 void supr()
81 { int i,j;
82   for(i=0;i<=nl;i++)
83     for(j=0;j<=nc;j++)
84       if(a[i][j]==0){nr++;fill(i,j);}
85 }
86
87 int main()
88 { citire();
89   linii();coloane();
90   supr();
91   g=fopen("colaj.out","w");
92   fprintf(g,"%d",nr);
93   //cout<<nr<<' ';
94   fclose(g);
95   return 0;
96 }
```

Listing 12.1.3: colaj_C.cpp

```

1 //prof. Carmen Minca
2 #include <stdio.h>
3
4 struct dr
5 {
6     int a,b,c,d;
7     int pa,pb,pc,pd;
8 } v[115];
9 int m,p,x[200],y[200];
10 int n,k,nr,nl,nc,x0=1,y0=1;
11 char a[210][210];
12 FILE *f,*g;
13
14 int ver(int i, int xa, int xb, int xc, int xd)
15 { int j;
16   for(j=1;j<=i;j++)
17     if((v[j].a<xa)&&(v[j].b<xb)&&(v[j].c>xc)&&(v[j].d>xd))return 0;
18   return 1;
19 }
20
21 void citire()
22 {                                     //citire date+ crearea lui X si Y
23   f=fopen("colaj.in","r");
24   fscanf(f,"%d%d%d",&n,&m,&p);
25   int i,xa,xb,xc,xd,r=0;
26   for(i=1;i<=n;i++)
27   { fscanf(f,"%d%d%d%d",&xa,&xb,&xc,&xd);
28     if(ver(r,xa,xb,xc,xd))
29     { r++;
30       v[r].a=xa; v[r].b=xb; v[r].c=xc; v[r].d=xd;
31       x[++k]=v[r].a; y[k]=v[r].b;
32       x[++k]=v[r].c; y[k]=v[r].d;
33       if((v[r].a==0)|| (v[r].c==0))x0--; //verific daca exista dreptunghi
34       if((v[r].b==0)|| (v[r].d==0))y0--; //cu un varf pe Ox sau Oy sau in O
35       //daca nu am doar pe Oy =>inarc in matrice de la coloana 2
36     }
37   }
38 }
```

```

36     //daca nu am doar pe Ox=>inarc in matrice de la linia 2
37     //nici pe Ox nici pe Oy=> incep cu linia 2 coloana 2
38     //am varf in O, OK
39 }
40 n=r;
41 fclose(f);
42 }
43
44 int poz(int z[],int i,int j)
45 { int m=-1, ind, s;
46     while(i<j)
47     { ind=0;
48         if(z[i]>z[j])
49             {ind = 1; s=z[i]; z[i]=z[j]; z[j]=s;}
50         if(ind)
51             {if(m===-1)i++;
52             else j--;
53             m=-m;
54         }
55         else
56             if(m===-1)j--;
57         else i++;
58     }
59     return i;
60 }
61
62 void dv(int z[], int s, int d)    //sortez prin Qsort
63 { if(d>s)
64     {int k;
65      k=poz(z,s,d);
66      dv(z,s,k-1);dv(z,k+1,d);}
67 }
68
69 int caut(int z[], int a)
70 { int i=1, j,m;           //caut fiecare coordonata varf in X sau Y
71   j=k;
72   while(i<=j)
73   {m=(i+j)/2;
74     if(z[m]==a) return m;
75     else
76       if(a<z[m]) j=m-1;
77       else i=m+1;
78   }
79   return i;
80 }
81
82 void pozitii()           //stabilesc pozitia varfurilor dreptunghiurilor
83 {int i;                   //fata de zonele elementare
84     for(i=1;i<=n;i++)
85     { v[i].pa=caut(x,v[i].a);
86       v[i].pb=caut(y,v[i].b);
87       v[i].pc=caut(x,v[i].c);
88       v[i].pd=caut(y,v[i].d);
89     }
90 }
91
92 void matrice()
93 { int i,j,k;
94   for(k=1;k<=n;k++)           //formez matricea
95   for(j=v[k].pa;j<v[k].pc;j++)
96     for(i=v[k].pb;i<v[k].pd ;i++)a[i+y0][j+x0]=1;
97 }
98
99 void bordare()
100 { int i;
101   nl=y0+2*n-1;
102   nc=x0+2*n-1;
103   if(m>x[k])nc++;
104   if(p>y[k])nl++;
105   for(i=0;i<=nl+1;i++)        //bordez matricea cu 1
106     a[i][0]=a[i][nc+1]=1;
107   for(i=1;i<=nc+1;i++)
108     a[0][i]=a[nl+1][i]=1;
109 }
110
111 void fill(int i,int j)

```

```

112 { if(a[i][j]==0)
113 { a[i][j]=1;
114     fill(i,j-1); fill(i,j+1); fill(i-1,j); fill(i+1,j);
115 }
116 }
117
118 void supr()
119 { int i,j;
120   for(i=1;i<=nl;i++)
121     for(j=1;j<=nc;j++)
122       if(!a[i][j]) {nr++;fill(i,j);}
123 }
124
125 int main()
126 { citire();
127   dv(x,1,k); dv(y,1,k); //sortez vectorii x,y cu Quicksort
128   pozitii(); //caut v[k].pa...pd
129   matrice(); //formez matricea
130   bordare();
131   supr();
132   g=fopen("colaj.out", "w");
133   fprintf(g, "%d", nr);
134   //printf("\n%d", nr);
135   fclose(g);
136   return 0;
137 }
```

12.2 Piață

Problema 2 - Piață

100 de puncte

Ionuț pleacă la sfârșit de săptămână să se relaxeze într-un parc de distracții. La intrarea în parc se află o piață mare, pavată cu plăci de marmură de aceeași dimensiune. Fiecare placă are scris pe ea un singur număr dintre $f(1)$, $f(2)$, $f(3)$, ..., $f(n)$, unde $f(k)$ este suma cifrelor lui k , pentru k din multimea $\{1, 2, \dots, n\}$. Piață are forma unui tablou bidimensional cu n linii și n coloane. Plăcile care alcătuiesc piață sunt așezate astfel:

- pe prima linie sunt plăci cu numerele $f(1)$, $f(2)$, ..., $f(n-2)$, $f(n-1)$, $f(n)$ (în această ordine de la stânga la dreapta);
- pe linia a doua sunt plăci cu numerele $f(n)$, $f(1)$, $f(2)$, $f(3)$, ..., $f(n-1)$, (în această ordine de la stânga la dreapta);
- pe linia a treia sunt plăci cu numerele $f(n-1)$, $f(n)$, $f(1)$, $f(2)$, $f(3)$, ..., $f(n-2)$ (în această ordine de la stânga la dreapta);
- ...
- pe ultima linie sunt plăci cu numerele $f(2)$, ..., $f(n-2)$, $f(n-1)$, $f(n)$, $f(1)$ (în această ordine de la stânga la dreapta).

Părinții lui Ionuț vor ca și în această zi, fiul lor să rezolve măcar o problemă cu sume. Astfel acestia îi propun lui Ionuț să determine suma numerelor aflate pe porțiunea dreptunghiulară din piață având colțurile în pozițiile în care se găsesc așezări ei. Tatăl se află pe linia iT și coloana jT (colțul stânga-sus), iar mama pe linia iM și coloana jM (colțul dreapta-jos). Porțiunea din piață pentru care se dorește suma este în formă dreptunghiulară, cu laturile paralele cu marginile pieței (vezi zona plină din exemplu). Dacă Ionuț va calcula suma cerută, atunci el va fi recompensat în parcul de distracții, de către părinții lui.

Cerințe

Determinați suma cerută de părinții lui Ionuț.

Date de intrare

Fișierul de intrare **piata.in** conține pe prima linie numărul natural n reprezentând dimensiunea pieței. Pe linia a doua se află despărțite printr-un spațiu numerele naturale iT și jT . Pe linia a treia se află despărțite printr-un spațiu numerele naturale iM și jM .

Date de ieșire

Fișierul de ieșire **piata.out**, va conține pe prima linie suma cerută.

Restricții și precizări

$$2 \leq n \leq 40000$$

$$1 \leq iT, jT, iM, jM \leq n$$

$$iT \leq iM$$

$$jT \leq jM$$

Suma cerută de părinții lui Ionuț nu depășește niciodată valoarea 2100000000.

20% din teste au $n \leq 250$

30% din teste au $250 \leq n \leq 10000$

30% din teste au $10001 \leq n \leq 28000$

20% din teste au $28001 \leq n \leq 40000$

Exemple

piata.in	piata.out	Explicații																																				
6 2 3 6 5	51	<p>Piața arată astfel:</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr> <tr><td>6</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr> <tr><td>5</td><td>6</td><td>1</td><td>2</td><td>3</td><td>4</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>1</td><td>2</td><td>3</td></tr> <tr><td>3</td><td>4</td><td>5</td><td>6</td><td>1</td><td>2</td></tr> <tr><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>1</td></tr> </table> <p>(marcată mai sus) este 51.</p> <p>Suma numerelor din porțiunea cerută</p>	1	2	3	4	5	6	6	1	2	3	4	5	5	6	1	2	3	4	4	5	6	1	2	3	3	4	5	6	1	2	2	3	4	5	6	1
1	2	3	4	5	6																																	
6	1	2	3	4	5																																	
5	6	1	2	3	4																																	
4	5	6	1	2	3																																	
3	4	5	6	1	2																																	
2	3	4	5	6	1																																	

Timp maxim de executare/test: **1.0** secunde

12.2.1 Indicații de rezolvare

prof. Doru Popescu Anastasiu

Se observă că un element de pe linia i , coloana j este egal cu:

```
su(j-i+1), daca j>=i
su(n+j-i+1), daca j<i
```

unde:

$su(k)$ este suma cifrelor lui k .

Dacă nu ne dăm seama de acest lucru va trebui să utilizăm un vector cu elementele de pe prima linie, după care folosind elementele lui putem accesa fiecare componentă din tablou.

Nu trebuie să construim tabloul pentru a calcula suma dorită.

O linie (începând cu a doua) din tabloul ce se definește în enunț se poate construi în funcție de precedență.

Pentru a calcula suma cerută, trebuie să calculăm suma de pe prima linie a subtabloului (cu colțul stânga sus (iT, jT) și colțul din dreapta jos (iM, jM)), după care suma de pe linia i ($i > iT$) din subtabloul este egală cu suma de pe linia $i - 1$ din tablou, din care scădem ultimul element al acestei linii (de pe coloana jM , pentru că nu mai face parte din linia i) și adunăm elementul de pe coloana jT , linia i (care este singur element de pe linia i ce nu se regăsește și pe linia $i - 1$ din subtabloul)

```
{suma de pe linia iT}
s:=0;
for j:=jT to jM do
  if j>=iT then s:=s+su(j-iT+1)
  else s:=s+su(n+j-iT+1);
{sumele de pe liniile iT+1, iT+2, ..., iM}
s1:=s;{suma de pe linia anterioară}
for i:=iT+1 to iM do
begin
  {elementul de pe linia i, coloana jM}
  if jM>=i-1 then e1:=su(jM-(i-1)+1) else e1:= su(n+jM-(i-1)+1);
  {elementul de pe linia i, coloana jT}
  if jT>=i then e2:=su(jT-i+1) else e2:= su(n+jT-i+1);
```

```

s:=s+s1-e1+e2;
s1:=s1-e1+e2;
end;

```

Se scrie în fișier *s*

12.2.2 *Rezolvare detaliată

12.2.3 Cod sursă

Listing 12.2.1: PI_1.pas

```

1 program dpa_piata;
2 const fi='piata.in';
3         fo='piata.out';
4 var f:text;
5     n,iT,jT,iM,jM,i,j:longint;
6     s,e1,e2,s1:longint;
7     a:array[1..170,1..170]of integer;
8
9 function fu(k:longint):longint;
10 var s:longint;
11 begin
12     s:=0;
13     while k>0 do
14         begin
15             s:=s+(k mod 10);
16             k:=k div 10;
17         end;
18     fu:=s;
19 end;
20
21 begin
22     assign(f,fi);reset(f);
23     readin(f,n);
24     readln(f,iT,jT);
25     readln(f,iM,jM);
26     close(f);
27     {linia 1}
28     for j:=1 to n do a[1,j]:=fu(j);
29     {liniile 2, 3, ..., n}
30     for i:=2 to n do
31     begin
32         for j:=2 to n do
33             a[i,j]:=a[i-1,j-1];
34         a[i,1]:=a[i-1,n];
35     end;
36     s:=0;
37     for i:=iT to iM do
38         for j:=jT to jM do
39             s:=s+a[i,j];
40     assign(f,fo);rewrite(f);
41     writeln(f,s);
42     close(f);
43 end.

```

Listing 12.2.2: PI_2.pas

```

1 program dpa_piata;
2 const fi='piata.in';
3         fo='piata.out';
4 var f:text;
5     n,iT,jT,iM,jM,i,j:longint;
6     s,e1,e2,s1:longint;
7     a:array[1..250,1..250]of byte;
8
9 function fu(k:longint):byte;
10 var s:longint;
11 begin

```

```

12   s:=0;
13   while k>0 do
14     begin
15       s:=s+(k mod 10);
16       k:=k div 10;
17     end;
18   fu:=s;
19 end;
20
21 begin
22 assign(f,fi);reset(f);
23 readln(f,n);
24 readln(f,iT,jT);
25 readln(f,iM,jM);
26 close(f);
27 {linia 1}
28 for j:=1 to n do a[1,j]:=fu(j);
29 {liniile 2, 3, ..., n}
30 for i:=2 to n do
31 begin
32   for j:=2 to n do
33     a[i,j]:=a[i-1,j-1];
34   a[i,1]:=a[i-1,n];
35 end;
36 s:=0;
37 for i:=iT to iM do
38   for j:=jT to jM do
39     s:=s+a[i,j];
40 assign(f,fo);rewrite(f);
41 writeln(f,s);
42 close(f);
43 end.

```

Listing 12.2.3: PI_3.pas

```

1 program dpa_piata;
2 const fi='piata.in';
3       fo='piata.out';
4 var f:text;
5   n,iT,jT,iM,jM,i,j:longint;
6   s:longint;
7   v,v1:array[1..28002]of byte;
8
9 function fu(k:longint):byte;
10 var s:byte;
11 begin
12   s:=0;
13   while k>0 do
14     begin
15       s:=s+(k mod 10);
16       k:=k div 10;
17     end;
18   fu:=s;
19 end;
20
21 begin
22 assign(f,fi);reset(f);
23 readln(f,n);
24 readln(f,iT,jT);
25 readln(f,iM,jM);
26 close(f);
27 {linia 1}
28 s:=0;
29 for j:=1 to n do v1[j]:=fu(j);
30 if iT=1 then
31   for j:=jT to jM do s:=s+v1[j];
32 {liniile 2, 3, ..., n}
33 for i:=2 to n do
34 begin
35   for j:=2 to n do
36     v[j]:=v1[j-1];
37   v[1]:=v1[n];
38   v1:=v;
39   if (iT<=i)and(i<=iM) then
40     for j:=jT to jM do s:=s+v1[j];

```

```

41 end;
42 assign(f,fo); rewrite(f);
43 writeln(f,s);
44 close(f);
45 end.
```

Listing 12.2.4: PI_OK.pas

```

1 program dpa_piata;
2 const fi='piata.in';
3   fo='piata.out';
4 var f:text;
5   n,iT,jT,iM,jM,i,j:longint;
6   s:longint;
7   v,v1:array[1..28002]of byte;
8
9 function fu(k:longint):byte;
10 var s:byte;
11 begin
12   s:=0;
13   while k>0 do
14     begin
15       s:=s+(k mod 10);
16       k:=k div 10;
17     end;
18   fu:=s;
19 end;
20
21 begin
22 assign(f,fi);reset(f);
23 readln(f,n);
24 readln(f,iT,jT);
25 readln(f,iM,jM);
26 close(f);
27 {linia 1}
28 s:=0;
29 for j:=1 to n do v1[j]:=fu(j);
30 if iT=1 then
31   for j:=jT to jM do s:=s+v1[j];
32 {liniile 2, 3, ..., n}
33 for i:=2 to n do
34 begin
35   for j:=2 to n do
36     v[j]:=v1[j-1];
37   v[1]:=v1[n];
38   v1:=v;
39   if (iT<=i)and(i<=iM) then
40     for j:=jT to jM do s:=s+v1[j];
41   end;
42 assign(f,fo); rewrite(f);
43 writeln(f,s);
44 close(f);
45 end.
```

Capitolul 13

OJI 2007

13.1 Alee

Parcul orașului a fost neglijat mult timp, astfel că acum toate aleile sunt distruse. Prin urmare, anul acesta Primăria și-a propus să facă reamenajări.

Parcul are forma unui pătrat cu latura de n metri și este înconjurat de un gard care are exact două porți. Proiectanții de la Primărie au realizat o hartă a parcului și au trasat pe hartă un carioaj care împarte parcul în $n \times n$ zone pătrate cu latura de 1 metru. Astfel harta parcului are aspectul unei matrice pătratice cu n linii și n coloane. Liniile și respectiv coloanele sunt numerotate de la 1 la n . Elementele matricei corespund zonelor pătrate de latură 1 metru. O astfel de zonă poate să conțină un copac sau este liberă.

Edilii orașului doresc să paveze cu un număr minim de dale pătrate cu latura de 1 metru zonele libere (fără copaci) ale parcului, astfel încât să se obțină o aleă continuă de la o poartă la alta.

Cerință

Scrieți un program care să determine numărul minim de dale necesare pentru construirea unei alei continue de la o poartă la cealaltă.

Date de intrare

Fișierul de intrare **alee.in** conține pe prima linie două valori naturale n și m separate printr-un spațiu, reprezentând dimensiunea parcului, respectiv numărul de copaci care se găsesc în parc.

Fiecare dintre următoarele m linii conține câte două numere naturale x și y separate printr-un spațiu, reprezentând pozițiile copacilor în parc (x reprezintă linia, iar y reprezintă coloana zonei în care se află copacul).

Ultima linie a fișierului conține patru numere naturale $x_1 \ y_1 \ x_2 \ y_2$, separate prin câte un spațiu, reprezentând pozițiile celor două porți (x_1, y_1 reprezintă linia și respectiv coloana zonei ce conține prima poartă, iar x_2, y_2 reprezintă linia și respectiv coloana zonei ce conține cea de a doua poartă).

Date de ieșire

Fișierul de ieșire **alee.out** va conține o singură linie pe care va fi scris un număr natural care reprezintă numărul minim de dale necesare pentru construirea aleii.

Restricții și precizări

- $1 \leq n \leq 175$
- $1 \leq m < n * n$
- Aleea este continuă dacă oricare două plăci consecutive au o latură comună.
- Aleea începe cu zona unde se găsește prima poartă și se termină cu zona unde se găsește cea de a două poartă.
 - Pozițiile porților sunt distințe și corespund unor zone libere.
 - Pentru datele de test există întotdeauna soluție.

Exemplu

alee.in	alee.out	Explicații
8 6	15	O modalitate de a construi aleea
2 7		cu număr minim de dale este:
3 3		OO-----
4 6		--OO---x-
5 4		--xO----
7 3		---OOx--
7 5		---xO---
1 1 8 8		----OO--
		--x-xOO-
		-----OO
		(cu x am marcat copaci, cu - zonele libere, iar cu O dalele aleii).

Timp maxim de execuție/test: 1 secundă

13.1.1 Indicații de rezolvare

Este o problemă "clasică" a cărei rezolvare se bazează pe algoritmul lui Lee.

Reprezentarea informațiilor:

Vom utiliza o matrice A în care inițial vom reține valoarea -2 pentru zonele libere, respectiv valoarea -1 pentru zonele în care se află un copac.

Ulterior, pe măsură ce explorăm matricea în scopul determinării unei alei de lungime minimă vom reține în matrice pentru pozițiile explorate un număr natural care reprezintă distanța minimă de la poziția primei porți la poziția respectivă (exprimată în numărul de dale necesare pentru construirea aleei).

Vom parurge matricea începând cu poziția primei porți.

Pozitiiile din matrice care au fost explorate le vom reține într-o coadă, în ordinea explorării lor.

La fiecare pas vom extrage din coadă o poziție (x, y) , vom explora toti cei 4 vecini liberi neexplorați ai poziției extrase și îi vom introduce în coadă. Când explorăm un vecin, reținem în matricea A pe poziția sa distanța minimă $(1 + A[x][y])$.

Procedeul se repetă cât timp coada nu este vidă și poziția în care se află cea de a doua poartă nu a fost explorată.

Rezultatul se va obține în matricea A , pe poziția celei de a două porți.

Allocarea memoriei pentru coadă ar putea reprezenta o problemă (indicele de linie și indicele de coloană trebuie să fie memorați pe 1 byte, deoarece coada poate avea maxim $175 * 175$ elemente).

O soluție de a "economisi" spațiul de memorie este de a utiliza coada în mod circular.

Problema poate fi abordată și recursiv, dar datorită adâncimii recursiei o astfel de abordare obține 80 de puncte.

13.1.2 Rezolvare detaliată

Listing 13.1.1: Alee1.java

```

1  %\begin{verbatim}
2  import java.io.*;
3  class alee
4  {
5      static StreamTokenizer st;
6      static PrintWriter out;
7
8      static final int DimMaxParc=177;
9      static final int DimMaxCoada=30625;
10
11     //deplasările pe linie și coloana pe direcțiile N,E,S,V
12     static final int[] dx = {0, -1, 0, 1, 0};
13     static final int[] dy = {0, 0, 1, 0, -1};
14
15     static int[] ql=new int[DimMaxCoada];
16     static int[] qc=new int[DimMaxCoada];
17
18     static int[][] A=new int[DimMaxParc][DimMaxParc];
19

```

```

20  /*
21   A[i][j]==-1 zona i,j este copac
22   A[i][j]==-2 zona i,j este libera
23   A[i][j]=d, zona i,j este libera, la distanta d de prima poarta
24 */
25
26  static int ix, iy, ox, oy;
27  static int n, m;
28  static int NrDale;
29
30  public static void main(String[] args) throws IOException
31  {
32     st=new StreamTokenizer(new BufferedReader(new FileReader("alee.in")));
33     out=new PrintWriter(new BufferedWriter(new FileWriter("alee.out")));
34
35     Citire();
36     Bordare();
37     DeterminNrDale();
38
39     out.println(NrDale);
40     out.close();
41 } // main
42
43  static void DeterminNrDale()
44  {
45     int IncC, SfC, k; //inceputul si sfarsitul cozii
46     int xl, xc, yl, yc;
47
48     //initializez coada
49     xl=ix;
50     xc=iy;
51     A[ix][iy]=1;
52
53     //pun pozitia initiala, poarta de intrare, in coada
54     IncC=SfC= 0;
55     ql[IncC]=xl;
56     qc[IncC]=xc;
57
58     //parcure parcul
59     while(IncC<=SfC && A[ox][oy]==-2)
60     {
61         //extrag un element din coada
62         xl=ql[IncC];
63         xc=qc[IncC];
64         IncC++;
65
66         //ma deplasez in cele patru directii posibile
67         for(k=1;k<=4;k++)
68         {
69             yl=xl+dx[k];
70             yc=xc+dy[k];
71
72             //y = urmatoarea pozitie in directia k
73             if(A[yl][yc]==-2) // y=pozitie libera cu distanta minima necalculata
74             {
75                 A[yl][yc]=A[xl][xc]+1;
76
77                 //inserez pozitia y in coada
78                 ++Sfc;
79                 ql[Sfc]=yl;
80                 qc[Sfc]=yc;
81                 } // if
82             } // for
83         } // while
84
85         if(A[ox][oy]==-2) System.out.println("Nu exista solutie ...!!!!");
86         else NrDale=A[ox][oy];
87     } //DeterminNrDale(...)

88
89  static void Bordare()
90  {
91     //bordez parcul cu pomi
92     int i, j;
93     for(i=1;i<=n;i++)
94     {
95         A[i][0]=-1;

```

```

96         A[i][n+1]=-1;
97         A[0][i]=-1;
98         A[n+1][i]=-1;
99     }
100    } // Bordare(...)

101   static void Citire() throws IOException
102   {
103     int k, i, j;
104
105     //citesc dimensiunile parcului si numarul de pomi
106     st.nextToken(); n=(int)st.nval;
107     st.nextToken(); m=(int)st.nval;
108
109     //marchez cu -2 pozitiile libere
110     for(i=1;i<=n;i++)
111       for(j=1;j<=n;j++) A[i][j]=-2;
112
113     //citesc coordonatele obstacolelor
114     for(k=1;k<=m;k++)
115     {
116       st.nextToken(); i=(int)st.nval;
117       st.nextToken(); j=(int)st.nval;
118       A[i][j]=-1;                      //marchez obstacolele cu -1
119     }
120
121     //citesc pozitiile portii de intrare si de iesire
122     st.nextToken(); ix=(int)st.nval;
123     st.nextToken(); iy=(int)st.nval;
124     st.nextToken(); ox=(int)st.nval;
125     st.nextToken(); oy=(int)st.nval;
126
127   } //Citire()
128 } // class
129 %\end{verbatim}

```

13.1.3 Cod sursă

Listing 13.1.2: aleee.c

```

1  /* prof. Emanuela Cerchez, Liceul de Informatica "Grigore Moisil" Iasi */
2  #include <stdio.h>
3  #define DimMaxParc 177
4  #define DimMaxCoada 30625
5
6  //deplasările pe linie și coloana pe directiile N,E,S,V}
7  const int dx[5] = {0, -1, 0, 1, 0};
8  const int dy[5] = {0, 0, 1, 0, -1};
9
10 typedef struct
11 {
12     unsigned char l, c; //pozitia in Parc
13     }Element;
14
15 typedef Element Coada[DimMaxCoada];
16
17 int A[DimMaxParc][DimMaxParc];
18 /**
19 A[i][j]==-1 daca in pozitia i,j se afla un copac
20 A[i][j]==-2 daca zona i,j este libera
21 A[i][j]=d, daca zona i,j este libera situata la distanta d de prima poarta
22 */
23
24 int ix, iy, ox, oy;
25 int n, m;
26 int NrDale;
27
28 void Citire(void)
29 {
30     int k, i, j;
31
32     FILE * Fin=fopen("aleee.in", "r", stdin);
33

```

```

34     scanf("%d%d", &n, &m); //citesc dimensiunile parcului si numarul de pomi
35     //marchez cu -2 pozitiile libere
36     for (i=1; i<=n; i++)
37         for (j=1; j<=n; j++) A[i][j] = -2;
38     for (k=1; k<=m; k++)
39     {
40         scanf("%d %d", &i, &j); //citesc coordonatele obstacolelor
41         A[i][j] = -1;           //marchez obstacolele cu -1
42     }
43     //citesc pozitiile portii de intrare si de iesire
44     scanf("%d %d %d %d", &ix, &iy, &ox, &oy);
45     fclose(Fin);
46 }
47
48 void Bordare(void)
49 {
50     //bordez parcul cu pomi
51     int i, j;
52     for (i=1; i<=n; i++)
53         {A[i][0] = -1;A[i][n+1] = -1;
54          A[0][i] = -1;A[n+1][i] = -1;}
55 }
56
57 void DeterminNrDale(void)
58 {
59     Coada C;
60     int IncC, SfC, k;      //inceputul si sfarsitul cozii
61     Element x, y;
62
63     //initializez coada
64     x.l = ix; x.c = iy; A[ix][iy] = 1;
65     //pun pozitia initiala, poarta de intrare, in coada}
66     IncC = SfC = 0; C[IncC] = x;
67     //parcure parcul
68     while (IncC <=SfC && A[ox][oy]==-2)
69     {
70         //extrag un element din coada
71         x = C[IncC++];
72         //ma deplasez in cele patru directii posibile
73         for (k=1; k<=4; k++)
74         {
75             y.l = x.l + dx[k]; y.c = x.c + dy[k];
76             //y - urmatoarea pozitie in directia k
77             if (A[y.l][y.c]==-2)
78                 //y- pozitie libera cu distanta minima necalculata
79                 {
80                     A[y.l][y.c] = A[x.l][x.c]+1;
81                     //inserez pozitia y in coada
82                     C[++SfC] = y;
83                 }
84         }
85     }
86     if (A[ox][oy]==-2) printf("Nu exista solutie\n");
87     else NrDale=A[ox][oy];
88 }
89
90 void Afisare(void)
91 {
92     //afisez solutia
93     FILE * Fout=fopen("alee.out", "w", stdout);
94     printf("%d\n", NrDale);
95     fclose(Fout);
96 }
97
98 void main(void)
99 {
100     Citire();
101     Bordare();
102     DeterminNrDale();
103     Afisare();
104 }
```

13.2 Dir

Costel trebuie să realizeze, împreună cu echipa sa, o aplicație software pentru gestiunea fișierelor de pe hard-disc, sarcina sa fiind aceea de a scrie un modul pentru determinarea căilor tuturor fișierelor de date aflate în structura arborescentă a folderelor de pe disc. Membrii echipei au stabilit o codificare proprie pentru memorarea structurii fișierelor de pe disc, utilizând un sir de caractere. Specificațiile tehnice sunt următoarele:

- folderul este un fișier de tip special, care poate conține fișiere și/sau foldere (acestea fiind considerate subfoldere ale folderului respectiv);
- numele folderelor încep cu o literă, au maxim 30 de caractere și sunt scrise cu majuscule;
- numele fișierelor de date încep cu o literă, au maxim 30 de caractere și sunt scrise cu minuscule;
- caracterele utilizate pentru numele fișierelor și folderelor sunt literele alfabetului englez și cifrele arabe;
- reprezentarea structurii fișierelor sub forma unui sir de caractere se realizează după următoarea regulă:

`NUME_FOLDER(lista_de_foldere_si_fisiere)`

unde `lista_de_foldere_si_fisiere`, posibil vidă, conține fișierele și/sau subfolderele folderului `NUME_FOLDER`, separate prin virgulă. Subfolderele se reprezintă respectând aceeași regulă.

De exemplu, structura de fișiere și foldere din figura de mai jos



Figura 13.1: Dir

se reprezintă prin sirul de caractere:

`FOLDER1 (FOLDER2 (), FOLDER3 (FOLDER4 (poveste, basm), basm))`

Cerință

Scrieți un program care, cunoscând sirul de caractere ce codifică o structură de fișiere de pe disc, determină calea pentru fiecare fișier de date din structură. Prin cale a unui fișier se înțelege o succesiune de foldere, fiecare folder fiind urmat de caracterul \ (backslash), începând de la folderul aflat pe cel mai înalt nivel al structurii (primul specificat în sirul ce codifică structura de fișiere), până la subfolderul în care se află fișierul de date respectiv și terminată cu numele fișierului. Căile determinate vor fi afișate în ordine lexicografică.

Date de intrare

Fișierul de intrare **dir.in** conține pe prima linie sirul de caractere ce codifică structura de fișiere de pe disc.

Date de ieșire

Fișierul de ieșire **dir.out** va conține pe prima linie un număr natural N reprezentând numărul de fișiere de date găsite. Pe următoarele N linii se vor scrie, în ordine lexicografică, căile ce permit identificarea fișierelor găsite, în formatul: $F1 \ F2 \ \dots \ Fn \ fisier$, câte o cale pe o linie.

Restricții și precizări

Sirul de caractere ce codifică structura de fișiere este nevid și conține maxim 1600 de caractere. Structura de foldere conține cel puțin un folder și cel puțin un fișier.

Numărul de fișiere de date este cel mult 100.

Lungimea căii unui fișier este de cel mult 255 caractere.

Sirul $x_1x_2\dots x_n$ este mai mic lexicografic decât sirul $y_1y_2\dots y_m$, dacă există k astfel încât $x_1 = y_1$, $x_2 = y_2$, ..., $x_{k-1} = y_{k-1}$ și $(x_k < y_k$ sau $k = n + 1)$.

Punctaj

Pentru determinarea corectă a numărului de fișiere de date se acordă 30% din punctaj. Dacă numărul de fișiere de date a fost determinat corect și căile sunt corect afișate în ordine lexicografică se acordă punctajul integral.

Exemplu

dir.in
FOLDER1 (FOLDER2 () , FOLDER3 (FOLDER4 (poveste, basm) , basm))

dir.out
3
FOLDER1\FOLDER3\FOLDER4\basm
FOLDER1\FOLDER3\FOLDER4\poveste
FOLDER1\FOLDER3\basm

Timp maxim de execuție/test: 1 secundă

13.2.1 Indicații de rezolvare

Pentru determinarea tuturor căilor fișierelor din structura dată vom utiliza o stivă ST, care va reține, la un moment dat succesiunea directoarelor de la folderul rădăcină și până la folderul curent (ultimul descoperit).

Algoritmul de rezolvare este următorul:

1. Inițial stiva este vidă.
2. Parcuregem șirul de intrare caracter cu caracter și identificăm fișierele și/sau folderele, astfel: O secvență de caractere reprezintă:
 - un folder dacă este delimitată de caracterele "(" și ")" sau ";" și "(" și conține doar majuscule;
 - un fișier dacă este delimitată de caracterele "(" și ";" sau ";" și ";" sau ";" și ")";

Dacă este identificat un folder, atunci numele acestuia se introduce în stivă și începem explorarea fișierelor și folderelor subordonate acestuia.

Dacă este identificat un fișier, atunci calea asociată este compusă din folderele din stivă, de la bază către vârf, separate prin '\'. Calea determinată este memorată în tabloul de șiruri de caractere Rez.

3. După epuizarea fișierelor și folderelor din directorul curent acesta este extras din stivă.

Se repetă procesul până când stiva devine vidă.

În final, sortăm lexicografic șirurile de caractere memorate în tabloul Rez.

Problema admite și o soluție recursivă.

13.2.2 Rezolvare detaliată

Listing 13.2.1: dir1.java

```

1 import java.io.*;
2 class dir1
3 {
4     static BufferedReader br;
5     static PrintWriter out;
6
7     static final int LGM=3001;
8     static final int MaxS=256;
9
10    static String[] ST=new String[MaxS];           // stiva
11    static String Sir;                           // sirul dat
12    static int VF;                            // varful stivei
13    static String[] Rez=new String[MaxS];        // tablou pentru memorarea solutiilor
14    static int NrSol;                         // numarul solutiilor
15
16    public static void main(String[] args) throws IOException
17    {
18        br=new BufferedReader(new FileReader("dir.in"));
19        out=new PrintWriter(new BufferedWriter(new FileWriter("dir.out")));
20
21        Sir=br.readLine();
22        //System.out.println(Sir);
23    }

```

```

24     rezolvare();
25     afisare();
26
27     out.close();
28 } // main
29
30 static void afisare()
31 {
32     int i,ok=1;
33     String tmp=new String();
34
35     //sortez alfabetic caile gasite
36     while(ok==1)
37     {
38         ok=0;
39         for(i=1;i<NrSol;i++)
40             if(Rez[i].compareTo(Rez[i+1])>0)
41             {
42                 tmp=Rez[i];
43                 Rez[i]=Rez[i+1];
44                 Rez[i+1]=tmp;
45                 ok=1;
46             }
47     }
48
49     out.println(NrSol);
50     for(i=1;i<=NrSol;i++) out.println(Rez[i]);
51 } // afisare(...)

52
53 static void rezolvare()
54 {
55     int i,j,N;
56     int inc=0, sf=0, term=0;
57
58     VF=0;                                // stiva = vida
59     NrSol=0;
60
61     N=Sir.length()-1;                    // pozitia ultimului caracter
62     String dirC=new String();
63
64     //determin directorul radacina si-l introduc in stiva
65     dirC="";
66     i=0;
67     while(Sir.charAt(i)!='(') {dirC+=Sir.charAt(i); i++;}
68     VF++;
69     ST[VF]=new String(dirC);
70     //afisstiva();

71
72     //cat timp stiva e nevida
73     while(VF!=0)
74     {
75         //System.out.println(i+" --> "+Sir.charAt(i));
76         term=0;
77         if(Sir.charAt(i)==')')
78         {
79             VF--;
80             i++;                                // urmatorul caracter
81             continue;
82         }
83
84         //caut folder
85         while(i<=N && Sir.charAt(i)!=',' && Sir.charAt(i)!= '(') i++;
86         if(i<N) inc=i++; else term=1;
87         while(i<=N && Sir.charAt(i)!=',' && Sir.charAt(i)!= '(' && Sir.charAt(i)!= ')') i++;
88         if(i<=N) sf=i; else term=1;
89
90         //System.out.println("inc = "+inc+" "+Sir.charAt(inc)+" sf = "+sf+" "+Sir.charAt(sf));
91         if(sf==inc+1) continue;
92
93         if(term==0)
94         {
95             dirC="";
96             j=inc+1;
97             while(j<sf) dirC+=Sir.charAt(j++);
98

```

```

99         //System.out.println("dirC = "+dirC);
100
101        if(
102            (Sir.charAt(inc) != '(' && Sir.charAt(sf)==')' || 
103            (Sir.charAt(inc) != ',' && Sir.charAt(sf)==',')) 
104            )
105        {
106            //inserez folderul in stiva
107            VF++;
108            ST[VF]=new String(dirC);
109            //System.out.println(dirC+" --> stiva");
110            //afisstiva();
111        }
112        else
113        {
114            //am gasit un nume de fisier
115            if(!dirC.equals(""))
116            {
117                //compun calea catre fisier si memorez rezultatul
118                NrSol++;
119                String tmp=new String();
120                String bkslash="\\";
121                tmp="";
122                for(j=1;j<=VF;j++) {tmp+=ST[j]; tmp+=bkslash;}
123                tmp+=dirC;
124                Rez[NrSol]=new String(tmp);
125
126                //System.out.println(dirC+" ==> SOLUTIE: "+tmp);
127            }
128        } // else
129    } //if
130    else VF--;
131 } // while
132 } // rezolvare
133
134 static void afisstiva()
135 {
136     int i;
137     for(i=VF;i>=1;i--) System.out.println(i+" : "+ST[i]);
138     System.out.println();
139 }
140 } // class

```

Versiunea 2:

Listing 13.2.2: dir2.java

```

1 import java.io.*;
2 import java.util.StringTokenizer;
3 class dir2
4 {
5     static BufferedReader br;
6     static PrintWriter out;
7
8     static final int LGM=3001;
9     static final int MaxS=256;
10
11    static String[] ST=new String[MaxS];      // stiva
12    static String Sir;                      // sirul dat
13    static int VF;                         // varful stivei
14    static String[] Rez=new String[MaxS];    // tablou pentru memorarea solutiilor
15    static int NrSol;                     // numarul solutiilor
16
17    public static void main(String[] args) throws IOException
18    {
19        br=new BufferedReader(new FileReader("dir.in"));
20        out=new PrintWriter(new BufferedWriter(new FileWriter("dir.out")));
21        Sir=br.readLine();
22        rezolvare();
23        afisare();
24        out.close();
25    } // main
26
27    static void afisare()

```

```

28  {
29      int i,ok=1;
30      String tmp=new String();
31
32      //sortez alfabetic caile gasite
33      while(ok==1)
34      {
35          ok=0;
36          for(i=1;i<NrSol;i++)
37          if(Rez[i].compareTo(Rez[i+1])>0)
38          {
39              tmp=Rez[i];
40              Rez[i]=Rez[i+1];
41              Rez[i+1]=tmp;
42              ok=1;
43          }
44      }
45
46      out.println(NrSol);
47      for(i=1;i<=NrSol;i++) out.println(Rez[i]);
48 } // afisare(...)

49
50 static void rezolvare()
51 {
52     StringTokenizer str=new StringTokenizer(Sir, "(),");
53     String token;
54
55     int i,j;
56     int inc=0, sf=0;
57
58     VF=0;                                // stiva = vida
59     NrSol=0;
60
61     //determin directorul radacina si-l introduc in stiva
62     token = str.nextToken();
63     sf=inc+token.length();
64
65     VF++;
66     ST[VF]=new String(token);
67
68     while(str.hasMoreTokens())
69     {
70         inc=sf;
71
72         if(Sir.charAt(sf)=='(')
73         {
74             VF--;
75             sf++;
76             continue;
77         }
78
79         if(Sir.charAt(inc+1)==')') { sf=inc+1; continue; }
80
81         //caut folder
82         token = str.nextToken();
83         sf=inc+token.length()+1;
84
85         if(
86             (Sir.charAt(inc)!='(' && Sir.charAt(sf)==')' ||
87             (Sir.charAt(inc)!=',' && Sir.charAt(sf)==',' )
88         )
89         {
90             //inserez folderul in stiva
91             VF++;
92             ST[VF]=new String(token);
93         }
94         else
95         {
96             //am gasit un nume de fisier
97             if(!token.equals(""))
98             {
99                 //compun calea catre fisier si memorez rezultatul
100                NrSol++;
101                String tmp=new String();
102                String bkslash="\\";
103                tmp="";

```

```

104         for(j=1; j<=VF; j++) {tmp+=ST[j]; tmp+=bkslash;}
105         tmp+=token;
106         Rez[NrSol]=new String(tmp);
107     }
108 } // else
109
110     inc=sf;
111 } // while
112 } // rezolvare
113 } // class

```

13.2.3 Cod sursă

Listing 13.2.3: dir_em.c

```

1 /*Emanuela Cerchez - Liceul de Informatica "Grigore Moisil" Iasi */
2 #include <stdio.h>
3 #include <string.h>
4
5 #define InFile "dir.in"
6 #define OutFile "dir.out"
7 #define LGS 1601
8 #define NMaxF 100
9 #define LGC 256
10
11 typedef char Nume[31];
12 char s[LGS];
13 Nume sol[200];
14 FILE *fout;
15 int poz=0, nr=0;
16 char a[NMaxF][LGC];
17 int nrsol=0;
18
19 void citire(void);
20 void structura(void);
21 void fisier(void);
22 void afisare(void);
23 void sortare(void);
24
25 int main()
26 {
27     citire();
28     structura();
29     sortare();
30     afisare();
31     return 0;
32 }
33
34 void citire()
35 {
36     FILE * fin = fopen(InFile, "r");
37     fscanf(fin, "%s", s);
38     fclose(fin);
39 }
40
41 void structura()
42 {
43     char *p;
44     if (!s[poz]) return;
45     p=strchr(s+poz, '(');
46     *p=NULL;
47     strcpy(sol[nr++], s+poz);
48     poz+=strlen(s+poz)+1;
49     if (s[poz]==')')
50     { nr--; poz++; }
51     else
52     {
53         do
54         {
55             if (s[poz]>='A' && s[poz]<='Z')
56                 structura();
57             else

```

```

58         if (s[poz]>='a' && s[poz]<='z')
59             fisier();
60         if (s[poz++]==')') break;
61     }
62     while (1);
63     nr--;
64   }
65 }
66
67 void fisier()
68 {char nf[31];
69 int i;
70 for (i=poz; s[i]>='a' && s[i]<='z' || s[i]<='9' && s[i]>='0'; i++)
71   nf[i-poz]=s[i];
72 nf[i-poz]=NULL;
73 for (i=0; i<nr; i++)
74   {strcat(a[nrsol],sol[i]);
75   strcat(a[nrsol],"\\\"");}
76 strcat(a[nrsol],nf);
77 nrsol++;
78 poz+=strlen(nf);
79 }
80
81 void afisare(void)
82 {
83 FILE * fout=fopen(OutFile, "w");
84 int i;
85 fprintf(fout, "%d\n", nrsol);
86 for (i=0; i<nrsol; i++)
87   fprintf(fout, "%s\n", a[i]);
88 fclose(fout);
89 }
90
91 void sortare(void)
92 {
93 int i, ok;
94 char aux[LGC];
95 do
96   {ok=1;
97   for (i=0; i<nrsol-1; i++)
98     if (strcmp(a[i],a[i+1])>0)
99       {strcpy(aux,a[i]);
100      strcpy(a[i],a[i+1]);
101      strcpy(a[i+1],aux);
102      ok=0;
103    }
104  }
105 while (!ok);
106 }
```

Listing 13.2.4: dir.cpp

```

1 /* Alin Burta - C.N. "B.P. Hasdeu Buzau */
2 #include <fstream>
3 #include <string.h>
4
5 using namespace std;
6
7 #define Fin "dir.in"
8 #define Fout "dir.out"
9 #define LGM 3001
10 #define MaxS 256
11
12 typedef char *TStiva[MaxS];
13 TStiva ST;
14 char *Sir; //Sirul dat
15 int VF; //varful stivei
16 char *Rez[MaxS]; //tablou pentru memorarea solutiilor
17 int NrSol; //numarul solutiilor
18
19 ofstream g(Fout);
20
21 void citire();
22 void rezolvare();
23 void afisare();
```

```

24 void init();
25 void Add(char *s);
26 char* Del();
27
28 int main()
29 {
30     init();
31     citire();
32     rezolvare();
33     afisare();
34     return 0;
35 }
36
37 void rezolvare()
38 {
39     int i,j,N;
40     int inc, sf, term;
41
42     N=strlen(Sir)-1;
43     char *dirC=new char[MaxS];
44     //determin directorul radacina si-l introduc in stiva
45     dirC[0]='\0';
46     i=0;
47     while(Sir[i]!='(') dirC[i]=Sir[i], i++;
48     dirC[i]='\0';
49     Add(dirC);
50     //cat timp stiva e nevida
51     while(VF)
52     {
53         term=0;
54         if(Sir[i]==')') {Del();i++;continue; }
55         if(Sir[i]=='(' & Sir[i]==')') VF--;
56         else
57         {
58             //caut folder
59             while(i<=N && !strchr(" ,()",Sir[i])) i++;
60             if(i<N) inc=i, i++;
61             else term=1;
62             while(i<=N && !strchr(" ,()",Sir[i])) i++;
63             if(i<=N) sf=i;
64             else term=1;
65
66             if(!term)
67             {
68                 j=inc+1; while(j<sf) dirC[j-inc-1]=Sir[j], j++; dirC[sf-inc-1]='\0';
69                 if(strchr(" ,()",Sir[inc]) && Sir[sf]==')')
70                 {
71                     //inserez folderul in stiva
72                     Add(dirC);
73                 }
74             else
75             {
76                 //am gasit un nume de fisier
77                 if(strcmp(dirC,"")!=0)
78                 {
79                     //compun calea catre fisier si memorez rezultatul
80                     NrSol++;
81                     char *tmp=new char[MaxS];
82                     char *bkslash="\\";
83                     tmp[0]='\0';
84                     for(j=1;j<=VF;j++) {strcat(tmp,ST[j]); strcat(tmp,bkslash);}
85                     strcat(tmp,dirC);
86                     Rez[NrSol]=new char[strlen(tmp)+1];
87                     strcpy(Rez[NrSol],tmp);
88                 }
89             }
90         }
91         else Del();
92     }
93 }
94
95 void afisare()
96 {
97     int i,ok=1;
98     char *tmp=new char[MaxS];

```

```

100     //sortez alfabetic caile gasite
101
102     while(ok)
103     {
104         ok=0;
105         for(i=1;i<NrSol;i++)
106             if(strcmp(Rez[i],Rez[i+1])>0)
107             {
108                 strcpy(tmp,Rez[i]);
109                 strcpy(Rez[i],Rez[i+1]);
110                 strcpy(Rez[i+1],tmp);
111                 ok=1;
112             }
113     }
114
115     g<<NrSol<<'\\n';
116     for(i=1;i<=NrSol;i++) g<<Rez[i]<<'\\n';
117     g.close();
118 }
119
120 void citire()
121 {
122     ifstream f(Fin);
123     f.get(Sir,3000,'\\n');
124     f.get();
125     f.close();
126 }
127
128 void init()
129 {
130     int i;
131     Sir=new char[LGM];
132     VF=0;
133     NrSol=0;
134 }
135
136 void Add(char *s)
137 {
138     if(VF<MaxS) VF++, ST[VF]=strdup(s);
139 }
140
141 char* Del()
142 {
143     if(VF) {VF--; return ST[VF+1];}
144     return NULL;
145 }
```

Capitolul 14

OJI 2006

14.1 Ecuății

Emanuela Cerchez

Să considerăm ecuații de gradul I, de forma:

$$\text{expresie_1} = \text{expresie_2}$$

Expresiile specificate sunt constituite dintr-o succesiune de operanzi, între care există semnul + sau semnul - (cu semnificația binecunoscută de adunare, respectiv scădere).

Fiecare operand este fie un număr natural, fie un număr natural urmat de litera x (litera x reprezentând necunoscută), fie doar litera x (cea ce este echivalent cu $1x$).

De exemplu:

$$2x - 5 + 10x + 4 = 20 - x$$

Observați că în ecuațiile noastre nu apar paranteze și necunoscuta este întotdeauna desemnată de litera mică x .

Cerință

Scriți un program care să rezolve ecuații de gradul I, în formatul specificat în enunțul problemei.

Datele de intrare

Fisierul de intrare **ecuatii.in** conține pe prima linie un număr natural n , reprezentând numărul de ecuații din fișier. Pe fiecare dintre următoarele n linii este scrisă câte o ecuație.

Datele de ieșire

În fișierul de ieșire **ecuatii.out** vor fi scrise n linii, câte una pentru fiecare ecuație din fișierul de intrare. Pe linia i va fi scrisă soluția ecuației de pe linia $i + 1$ din fișierul de intrare.

Dacă soluția ecuației este un număr real, atunci acesta se va scrie cu 4 zecimale. Răspunsul este considerat corect dacă diferența în valoare absolută dintre soluția corectă și soluția concurentului este < 0.001 .

În cazul în care ecuația admite o infinitate de soluții, se va scrie mesajul **infinit** (cu litere mici).

Dacă ecuația nu admite soluții, se va scrie mesajul **imposibil** (de asemenea cu litere mici).

Restricții și precizări

- $1 \leq n \leq 10$
- Lungimea unei ecuații nu depășește 255 caractere.
- Ecuațiile nu conțin spații.
- Numerele naturale care intervin în ecuații sunt ≤ 1000 .
- Punctajul pe un test se acordă dacă și numai dacă toate ecuațiile din testul respectiv au fost rezolvate corect.

Exemplu

ecuatii.in	ecuatii.out
3	3.2527
$2x-4+5x+300=98x$	infinit
$x+2=2+x$	imposibil
$3x+5=3x+2$	

Timp maxim de execuție/test: 1 secundă

14.1.1 Indicații de rezolvare

Soluția comisiei

Vom citi ecuația într-un sir de caractere, apoi vom împărți sirul în două subșiruri (membrul stâng, cu alte cuvinte caracterele până la semnul egal formează primul sir, și membrul drept, cu alte cuvinte caracterele de după semnul egal formează al doilea sir).

Problema este de a determina pentru fiecare dintre cele două subșiruri coeficientul lui x și termenul liber.

Să notăm:

$nr1$ termenul liber din membrul stâng

$nr2$ termenul liber din membrul drept

$nrx1$ coeficientul lui x din membrul stâng

$nrx2$ coeficientul lui x din membrul drept.

Soluția ecuației este $(nr2 - nr1)/(nrx1 - nrx2)$ dacă $nrx1 \neq nrx2$.

Dacă $nrx1 = nrx2$, atunci ecuația este imposibilă (dacă $nr1 \neq nr2$) sau nedeterminată (dacă $nr1 = nr2$).

Pentru a determina $nr1$ și $nrx1$, respectiv $nr2$ și $nrx2$, se prelucrează cele două siruri, identificând coeficienții necunoscutei, respectiv termenii liberi.

14.1.2 Rezolvare detaliată

Listing 14.1.1: ecuatii.java

```

1 import java.io.*;
2 import java.util.StringTokenizer;
3 class ecuatii
4 {
5     static BufferedReader br;
6     static StringTokenizer st;
7     static PrintWriter out;
8
9     static String s,s1,s2;
10
11    static int n;
12    static int nr, nrx;
13    static int nr1, nr2, nrx1, nrx2;
14    static double x;           // solutia
15
16    public static void main(String[] args) throws IOException
17    {
18        br=new BufferedReader(new FileReader("ecuatii.in"));
19        st=new StringTokenizer(br);
20        out=new PrintWriter(new BufferedWriter(new FileWriter("ecuatii.out")));
21
22        int i,pozitieegal;
23
24        st.nextToken(); n=(int)st.nval;
25        br.readLine();           // citit EOL ... de pe prima linie ... !!!
26
27        for (i=0; i<n; i++)
28        {
29            s=br.readLine();
30
31            pozitieegal=s.indexOf("=");
32            s1=s.substring(0,pozitieegal); // ... atentie ... !!!
33            s2=s.substring(pozitieegal+1);
34
35            rezolva(s1); nr1=nr; nrx1=nrx;

```

```

36         rezolva(s2); nr2=nr; nrx2=nrx;
37
38         if(nr1==nrx2)
39             if(nr1==nr2) out.println("infinit");
40             else        out.println("imposibil");
41         else
42         {
43             // vezi ... Class NumberFormat ... dar aici ...
44             String sol="";
45             x=(double)(nr2-nr1)/(nrx1-nrx2);
46             if(x<0) {sol+="-"; x=-x;}
47             x=x+0.00005;
48
49             int xi=(int)x;
50             sol+=xi;
51             sol+=".";
52             x=x-xi;
53
54             int yi=(int)(x*10000);
55             sol+=yi;
56
57             if(yi<9) sol+="000"; else
58                 if(yi<99) sol+="00"; else
59                     if(yi<999) sol+="0";
60             out.println(sol);
61         }
62     }
63     out.close();
64 } // main
65
66 static void rezolva(String s)
67 {
68     StringTokenizer str=new StringTokenizer(s,"+-");
69     String token;
70     int v;
71     int semn=1,lg=-1;
72
73     nr=0; nrx=0;
74     while(str.hasMoreTokens())
75     {
76         token = str.nextToken();
77         lg=lg+token.length()+1;
78         if (token.startsWith("x"))
79         {
80             nrx+=semn;
81         }
82         else
83         {
84             if(token.endsWith("x"))
85             {
86                 v=Integer.parseInt(token.replace("x", ""));
87                 nrx=nrx+semn*v;
88             }
89             else
90             {
91                 v=Integer.parseInt(token);
92                 nr=nr+semn*v;
93             }
94         }
95
96         if(lg<s.length())
97             if(s.substring(lg,lg+1).equals("+")) semn=1; else semn=-1;
98     } // while
99 } // rezolva(...)
```

14.1.3 Cod sursă

Listing 14.1.2: ecuatii.cpp

```

1 #include <stdio.h>
2 #include <math.h>
```

```

3 #include <stdlib.h>
4 #include <string.h>
5
6 #define InFile "ecuatii.in"
7 #define OutFile "ecuatii.out"
8 #define LgMax 300
9
10 char s[LgMax],s1[LgMax],s2[LgMax];
11 int n;
12 long nr1, nr2, nrx1, nrx2;
13
14 void rezolva(char *, long &, long &);
15
16 int main()
17 {
18     FILE * fin=fopen(InFile,"r");
19     FILE * fout=fopen(OutFile,"w");
20
21     int i;
22     char *p;
23
24     fscanf(fin,"%d", &n);
25     for (i=0; i<n; i++)
26     {
27         fscanf(fin,"%s", s1);
28         p=strchr(s1,'=');
29         strcpy(s2,p+1);
30         *p=NULL;
31         rezolva(s1, nr1, nrx1);
32         rezolva(s2, nr2, nrx2);
33         if (nr1==nr2)
34             if (nr1==nr2)
35                 fprintf(fout, "infinxit\n");
36             else
37                 fprintf(fout, "imposibil\n");
38         else
39             fprintf(fout, "% .4lf\n", ((double)(nr2-nr1)/(nrx1-nrx2)));
40     }
41
42     fclose(fin);
43     fclose(fout);
44     return 0;
45 }
46
47 void rezolva(char *s, long &nr, long &nr
48 {
49     char *p, ss[LgMax];
50     long v;
51     int semn=1, l;
52
53     strcpy(ss,s);
54     p=strtok(ss,"+-");
55     nr=0;
56     nr=0;
57     while (p)
58     {
59         l=strlen(p);
60         if (p[0]=='x')
61             nr+=semn;
62         else
63             if (p[l-1]=='x')
64             {
65                 p[l-1]=NULL;
66                 v=atol(p);
67                 nr=nr+semn*v;
68             }
69             else
70             {
71                 v=atol(p);
72                 nr=nr+semn*v;
73             }
74
75         if (s[p+l-ss]=='+')
76             semn=1;
77         else
78             semn=-1;

```

```

79
80     p=strtok(NULL, "+-");
81 }
82 }
```

14.2 Sudest

Alin Burtza

Fermierul Ion deține un teren de formă pătrată, împărțit în $N \times N$ pătrate de latură unitate, pe care cultivă cartofi. Pentru recoltarea cartofilor fermierul folosește un robot special proiectat în acest scop. Robotul pornește din pătratul din stânga sus, de coordonate $(1, 1)$ și trebuie să ajungă în pătratul din dreapta jos, de coordonate (N, N) .

Traseul robotului este programat prin memorarea unor comenzi pe o cartelă magnetică. Fiecare comandă specifică direcția de deplasare (sud sau est) și numărul de pătrate pe care le parcurge în direcția respectivă. Robotul strângе recolta doar din pătratele în care se oprește între două comenzi.

Din păcate, cartela pe care se află programul s-a deteriorat și unitatea de citire a robotului nu mai poate distinge direcția de deplasare, ci numai numărul de pași pe care trebuie să-i facă robotul la fiecare comandă. Fermierul Ion trebuie să introducă manual, pentru fiecare comandă, direcția de deplasare.

Cerință

Scriți un program care să determine cantitatea maximă de cartofi pe care o poate culege robotul, în ipoteza în care Ion specifică manual, pentru fiecare comandă, direcția urmată de robot. Se va determina și traseul pe care se obține recolta maximă.

Datele de intrare

Fișierul de intrare **sudest.in** are următoarea structură:

- Pe linia 1 se află numărul natural N , reprezentând dimensiunea parcelei de teren.
- Pe următoarele N linii se află câte N numere naturale, separate prin spații, reprezentând cantitatea de cartofi din fiecare pătrat unitate.
- Pe linia $N + 2$ se află un număr natural K reprezentând numărul de comenzi aflate pe cartela magnetică.
- Pe linia $N + 3$ se află K numerele naturale C_1, \dots, C_K , separate prin spații, reprezentând numărul de pași pe care trebuie să-i efectueze robotul la fiecare comandă.

Datele de ieșire

Fișierul de ieșire **sudest.out** va conține pe prima linie cantitatea maximă de cartofi recoltată de robot. Pe următoarele $K + 1$ linii vor fi scrise, în ordine, coordonatele pătratelor unitate ce constituie traseul pentru care se obține cantitate maximă de cartofi, câte un pătrat unitate pe o linie. Coordonatele scrise pe aceeași linie vor fi separate printr-un spațiu. Primul pătrat de pe traseu va avea coordonatele 11 , iar ultimul va avea coordonatele NN . Dacă sunt mai multe trasee pe care se obține o cantitate maximă de cartofi recoltată se va afișa unul dintre acestea.

Restricții și precizări

- $5 \leq N \leq 100$
- $2 \leq K \leq 2 * N - 2$
- $1 \leq C_1, \dots, C_K \leq 10$
- Cantitatea de cartofi dintr-un pătrat de teren este număr natural între 0 și 100 .
- Pentru fiecare set de date de intrare se garantează că există cel puțin un traseu.
- Se consideră că robotul strângе recolta și din pătratul de plecare $(1, 1)$ și din cel de sosire (N, N) .
- Pentru determinarea corectă a cantității maxime recoltate se acordă 50% din punctajul alocat testului respectiv; pentru cantitate maximă recoltată și traseu corect se acordă 100% .

Exemplu

sudest.in	sudest.out	Explicații
6	29	Un alt traseu posibil este:
1 2 1 0 4 1	1 1	1 1
1 3 3 5 1 1	3 1	1 3
2 2 1 2 1 10	5 1	1 5
4 5 3 9 2 6	6 1	2 5
1 1 3 2 0 1	6 5	6 5
10 2 4 6 5 10	6 6	6 6
5		dar costul său este $1 + 1 + 4 + 1 + 5 + 10 = 22$
2 2 1 4 1		

Timp maxim de execuție/test: 1 secundă

14.2.1 Indicații de rezolvare

soluția comisiei

Reprezentarea informațiilor

- N - numărul de linii
- K - numărul de comenzi
- $A[Nmax][Nmax]$; - memorează cantitatea de produs
- $C[Nmax][Nmax]$; - $C[i][j] =$ cantitatea maximă de cartofi culeasă pe un traseu ce pornește din $(1, 1)$ și se termină în (i, j) , respectând condițiile problemei
- $P[Nmax][Nmax]$; - $P[i][j] =$ pasul la care am ajuns în poziția i, j culegând o cantitate maximă de cartofi
- $Move[2 * Nmax]$; - memorează cele K comenzi

Parcurg șirul celor k mutări. La fiecare mutare marchez pozițiile în care pot ajunge la mutarea respectivă.

Mai exact, parcurg toate pozițiile în care am putut ajunge la pasul precedent (cele marcate în matricea P corespunzător cu numărul pasului precedent) și pentru fiecare poziție verific dacă la pasul curent pot să execut mutarea la sud.

În caz afirmativ, verific dacă în acest caz obțin o cantitate de cartofi mai mare decât cea obținută până la momentul curent (dacă da, rețin noua cantitate, și marchez în matricea P poziția în care am ajuns cu indicele mutării curente).

În mod similar procedez pentru o mutare spre est.

14.2.2 Rezolvare detaliată

Variantă după soluția oficială:

Listing 14.2.1: sudest1.java

```

1 import java.io.*;
2 class Sudest1
3 {
4     static StreamTokenizer st;
5     static PrintWriter out;
6     static final int Nmax=101;
7     static int N, K;
8     static int[][] A=new int[Nmax][Nmax]; // A[i][j]=cantitatea de cartofi
9     static int[][] C=new int[Nmax][Nmax]; // C[i][j]=cantitatea maxima ...
10    static int[][] P=new int[Nmax][Nmax]; // pas
11    static int[] Move=new int[2*Nmax]; // comenzi
12
13    public static void main(String[] args) throws IOException
14    {
15        st=new StreamTokenizer( new BufferedReader(new FileReader("sudest.in")));
16        out=new PrintWriter(new BufferedWriter(new FileWriter("sudest.out")));
17        read_data();

```

```

18     init();
19     solve();
20 } // main(...)

21
22 static void read_data() throws IOException
23 {
24     int i, j, cmd;
25     st.nextToken(); N=(int)st.nval;
26     for(i=1; i<=N; i++)
27         for(j=1; j<=N; j++) { st.nextToken(); A[i][j]=(int)st.nval; }
28     st.nextToken(); K=(int)st.nval;
29     for(cmd=1; cmd<=K; cmd++) { st.nextToken(); Move[cmd]=(int)st.nval; }
30 } // read_data()

31
32 static void init()
33 {
34     int i, j;
35     for(i=1; i<=N; i++) for(j=1; j<=N; j++) { C[i][j]=-1; P[i][j]=255; }
36 } // init()

37
38 static boolean posibil(int x, int y) { return 1<=x && 1<=y && x<=N && y<=N; }

39
40 static void solve()
41 {
42     int i, j, cmd;
43     P[1][1]=0;
44     C[1][1]=A[1][1];
45     for(cmd=1; cmd<=K; cmd++)
46         for(i=1; i<=N; i++)
47             for(j=1; j<=N; j++)
48                 if(P[i][j]==cmd-1)
49                 {
50                     if(posibil(i+Move[cmd], j)) // SUD
51                     if(C[i][j]+A[i+Move[cmd]][j]>C[i+Move[cmd]][j])
52                     {
53                         P[i+Move[cmd]][j]=cmd;
54                         C[i+Move[cmd]][j]=C[i][j]+A[i+Move[cmd]][j];
55                     }
56                     if(posibil(i, j+Move[cmd])) // EST
57                         if(C[i][j]+A[i][j+Move[cmd]]>C[i][j+Move[cmd]])
58                         {
59                             P[i][j+Move[cmd]]=cmd;
60                             C[i][j+Move[cmd]]=C[i][j]+A[i][j+Move[cmd]];
61                         }
62                 } // if
63                 out.println(C[N][N]);           drum(N, N, K);           out.close();
64 } // solve()

65
66 static void drum(int x, int y, int pas)
67 {
68     int i;
69     boolean gasit;
70     if(x==1 && y==1) out.println("1 1");
71     else
72     {
73         gasit=false;
74         if(posibil(x, y-Move[pas]))
75             if(C[x][y-Move[pas]]==C[x][y]-A[x][y] && P[x][y-Move[pas]]==pas-1)
76             {
77                 drum(x, y-Move[pas], pas-1);
78                 out.println(x+" "+y);
79                 gasit=true;
80             }
81         if(!gasit)
82             if(posibil(x-Move[pas], y))
83                 if(C[x-Move[pas]][y]==C[x][y]-A[x][y] && P[x-Move[pas]][y]==pas-1)
84                 {
85                     drum(x-Move[pas], y, pas-1);
86                     out.println(x+" "+y);
87                 }
88     } // else
89 } // drum(...)

90 } // class

```

Variantă folosind coadă:

Listing 14.2.2: sudest2.java

```

1 import java.io.*; // 11, ..., nn --> 1,...,n*n pozitii in matrice !
2 class Sudest2
3 {
4     static StreamTokenizer st;
5     static PrintWriter out;
6     static final int nmax=101;
7     static int n, k;
8     static int[][] a=new int[nmax][nmax]; // a[i][j]=cantitatea de cartofi
9     static int[][] c=new int[nmax][nmax]; // c[i][j]=cantitatea maxima ...
10    static int[][] p=new int[nmax][nmax]; // pozitia anterioara optima
11    static int[] move=new int[2*nmax]; // comenziile
12    static int ic,sc,scv,qmax=1000; // qmax=100 este prea mic ...!!!
13    static int[] q=new int[qmax]; // coada
14
15    public static void main(String[] args) throws IOException
16    {
17        st=new StreamTokenizer(new BufferedReader(new FileReader("sudest.in")));
18        out=new PrintWriter(new BufferedWriter(new FileWriter("sudest.out")));
19        read_data();
20        init();
21        solve();
22    } // main(...)

23
24    static void read_data() throws IOException
25    {
26        int i,j,cmd;
27        st.nextToken(); n=(int)st.nval;
28        for(i=1;i<=n;i++)
29            for(j=1;j<=n;j++) { st.nextToken(); a[i][j]=(int)st.nval; }
30        st.nextToken(); k=(int)st.nval;
31        for(cmd=1;cmd<=k;cmd++) { st.nextToken(); move[cmd]=(int)st.nval; }
32    } // read_data()

33
34    static void init()
35    {
36        int i,j;
37        for(i=1;i<=n;i++) for(j=1;j<=n;j++) {c[i][j]=-1; p[i][j]=255; }
38    } // init()

39
40    static void solve()
41    {
42        int i,j,ii,jj,cmd;
43        p[1][1]=0; c[1][1]=a[1][1];
44        ic=0; sc=1; q[ic]=1; // pozitia [1][1] --> q
45        scv=1; // ultimul din coada la distanta 1 de [1][1]
46        for(cmd=1; cmd<=k; cmd++)
47        {
48            while(ic!=scv)
49            {
50                i=(q[ic]-1)/n+1; j=(q[ic]-1)%n+1; ic=(ic+1)%qmax; // scot din coada
51                // propag catre Sud
52                ii=i+move[cmd];
53                jj=j;
54                if((ii>=1)&&(ii<=n))
55                    if(c[i][j]+a[ii][jj]>c[ii][jj])
56                    {
57                        c[ii][jj]=c[i][j]+a[ii][jj];
58                        p[ii][jj]=(i-1)*n+j;
59                        q[sc]=(ii-1)*n+jj; sc=(sc+1)%qmax; // pun in coada
60                    }
61
62                // propag catre Est
63                jj=j+move[cmd];
64                ii=i;
65                if((jj>=1)&&(jj<=n))
66                    if(c[i][j]+a[ii][jj]>c[ii][jj])
67                    {
68                        c[ii][jj]=c[i][j]+a[ii][jj];
69                        p[ii][jj]=(i-1)*n+j;
70                        q[sc]=(ii-1)*n+jj; sc=(sc+1)%qmax; // pun in coada
71                    }
72    } // while

```

```

73         scv=sc;
74     } // for
75
76     out.println(c[n][n]);      drum(n,n);      out.close();
77 } // solve()
78
79 static void drum(int i, int j)
80 {
81     if(i*j==0) return;
82     drum((p[i][j]-1)/n+1,(p[i][j]-1)%n+1);
83     out.println(i+" "+j);
84 } // drum(...)
85 } // class

```

14.2.3 Cod sursă

Listing 14.2.3: sudest.cpp

```

1 #include <fstream>
2
3 using namespace std;
4
5 #define Fin "sudest.in"
6 #define Fout "sudest.out"
7 #define Nmax 100
8
9 int N;      //numarul de linii
10 int K;       //numarul de comenzi
11 int A[Nmax][Nmax]; //memoreaza cantitatea de produs
12 int C[Nmax][Nmax]; //C[i][j]=cantitatea maxima de cartofi culeasa pe
13 // un traseu ce porneste din (1,1) si se termina
14 // in (i,j), respectand conditiile problemei
15 unsigned char P[Nmax][Nmax];
16 int Move[2*Nmax]; //memoreaza cele K comenzi
17
18 ofstream g(Fout);
19
20 void read_data()
21 {
22     int i,j;
23     ifstream f(Fin);
24     f>>N;
25     for(i=0;i<N;i++)
26         for(j=0;j<N;j++) f>>A[i][j];
27     f>>K;
28     for(i=1;i<=K;i++) f>>Move[i];
29     f.close();
30 }
31
32 void init()
33 {
34     int i,j;
35     for(i=0;i<N;i++)
36         for(j=0;j<N;j++) {C[i][j]=-1; P[i][j]=255;}
37 }
38
39 int posibil(int x, int y)
40 {
41     return 0<=x && 0<=y && x<N && y<N;
42 }
43
44
45 void drum(int x, int y, int pas)
46 {
47     int i, gasit;
48     if(x==0 && y==0) g<<1<<" "<<1<<'\\n';
49     else
50     {
51         gasit=0;
52         if (posibil(x,y-Move[pas]))
53             if (C[x][y-Move[pas]]==C[x][y]-A[x][y] && P[x][y-Move[pas]]==pas-1)
54                 {

```

```

55         drum(x,y-Move[pas],pas-1);
56         g<<x+1<<" "<<y+1<<endl;
57         gasit=1;}
58     if (!gasit)
59     if (posibil(x-Move[pas],y))
60         if (C[x-Move[pas]][y]==C[x][y]-A[x][y] && P[x-Move[pas]][y]==pas-1)
61         {
62             drum(x-Move[pas],y,pas-1);
63             g<<x+1<<" "<<y+1<<endl;}
64     }
65 }
66
67 void solve()
68 {
69     int i,j, h;
70     P[0][0]=0; C[0][0]=A[0][0];
71     for (i=1; i<=K; i++)
72         for (j=0; j<N; j++)
73             for (h=0; h<N; h++)
74                 if (P[j][h]==i-1)
75                 {
76                     if (posibil(j+Move[i],h))
77                         if (C[j][h]+A[j+Move[i]][h]>C[j+Move[i]][h])
78                             {P[j+Move[i]][h]=i;
79                             C[j+Move[i]][h]=C[j][h]+A[j+Move[i]][h];}
80                     if (posibil(j,Move[i]+h))
81                         if (C[j][h]+A[j][Move[i]+h]>C[j][Move[i]+h])
82                             {P[j][Move[i]+h]=i;
83                             C[j][Move[i]+h]=C[j][h]+A[j][Move[i]+h];}
84                 }
85     g<<C[N-1][N-1]<<"\n";
86     drum(N-1,N-1,K);
87     g.close();
88 }
89
90 int main()
91 {
92     read_data();
93     init();
94     solve();
95     return 0;
96 }
```

Capitolul 15

OJI 2005

15.1 Lăcusta

Nistor Eugen Moț

Se consideră o matrice dreptunghiulară cu m linii și n coloane, cu valori naturale. Traversăm matricea pornind de la colțul stânga-sus la colțul dreapta-jos. O traversare constă din mai multe deplasări. La fiecare deplasare se execută un salt pe orizontală și un pas pe verticală. Un salt înseamnă că putem trece de la o celulă la oricare alta aflată pe aceeași linie, iar un pas înseamnă că putem trece de la o celulă la celula aflată imediat sub ea. Excepție face ultima deplasare (cea în care ne aflăm pe ultima linie), când vom face doar un salt pentru a ajunge în colțul dreapta-jos, dar nu vom mai face și pasul corespunzător. Astfel traversarea va consta din vizitarea a $2m$ celule.

Cerință

Scrieți un program care să determine suma minimă care se poate obține pentru o astfel de traversare.

Datele de intrare

Fișierul de intrare **lacusta.in** conține pe prima linie două numere naturale separate printr-un spațiu $m \ n$, reprezentând numărul de linii și respectiv numărul de coloane ale matricei. Pe următoarele m linii este descrisă matricea, câte n numere pe fiecare linie, separate prin câte un spațiu.

Datele de ieșire

Fișierul de ieșire **lacusta.out** va conține o singură linie pe care va fi scrisă suma minimă găsită.

Restricții și precizări

- $1 \leq m, n \leq 100$
- Valorile elementelor matricei sunt numere întregi din intervalul $[1, 255]$.

Exemple

lacusta.in	lacusta.out	Explicatie
4 5	28	Drumul este:
3 4 5 7 9		$(1, 1) \rightarrow (1, 3) \rightarrow$
6 6 3 4 4		$(2, 3) \rightarrow (2, 2) \rightarrow$
6 3 3 9 6		$(3, 2) \rightarrow (3, 3) \rightarrow$
6 5 3 8 2		$(4, 3) \rightarrow (4, 5)$

Timp maxim de executare: 1 secundă/test

15.1.1 Indicații de rezolvare

Ginfo nr. 15/3 martie 2005

Pentru rezolvarea acestei probleme vom utiliza metoda *programării dinamice*.

Vom nota prin A matricea dată și vom construi o matrice B ale cărei elemente b_{ij} vor conține sumele minime necesare pentru a ajunge în celula (i, j) pornind din celula $(i - 1, j)$.

Vom completa inițial elementele de pe a doua linie a matricei B . Valoarea $b_{2,1}$ va fi ∞ deoarece în această celulă nu se poate ajunge. Valorile celorlalte elemente $b_{2,i}$ vor fi calculate pe baza formulei: $b_{2,i} = a_{1,1} + a_{1,i} + a_{2,i}$.

Pentru celelalte linii, valorile b_{ij} vor fi calculate pe baza formulei:

$$b_{i,j} = a_{i,j} + a_{i-1,j} + \min(b_{i-1,k}),$$

unde k variază între 1 și n . Evident, relația nu este valabilă pentru elementul de pe coloana k care corespunde minimului, deoarece nu se poate coborî direct, ci trebuie efectuat un salt orizontal. În această situație vom alege al doilea minim de pe linia anterioară.

În final alegem minimul valorilor de pe ultima linie a matricei B (fără a lua în considerare elementul de pe ultima coloană a acestei linii) la care adaugăm valoarea a_{mn} .

15.1.2 Rezolvare detaliată

Prima variantă:

Listing 15.1.1: lacusta1.java

```

1 import java.io.*;
2 class Lacustal
3 {
4     static final int oo=100000;
5     static int m,n;
6     static int[][] a,b;    // 0 <= i <= m-1;    0 <= j <= n-1
7
8     public static void main(String[] args) throws IOException
9     {
10        long t1,t2;
11        t1=System.currentTimeMillis();
12
13        int i,j;
14
15        StreamTokenizer st=new StreamTokenizer(
16            new BufferedReader(new FileReader("lacusta.in")));
17        PrintWriter out=new PrintWriter(
18            new BufferedWriter(new FileWriter("lacusta.out")));
19
20        st.nextToken(); m=(int)st.nval;
21        st.nextToken(); n=(int)st.nval;
22
23        a=new int[m][n];
24        b=new int[m][n];
25
26        for(i=0;i<m;i++)
27            for(j=0;j<n;j++)
28            {
29                st.nextToken(); a[i][j]=(int)st.nval;
30            }
31        for(i=0;i<m;i++) for(j=0;j<n;j++) b[i][j]=oo;
32
33        // prima linie (i=0) din b este oo
34        // a doua linie (i=1) din b
35        for(j=1;j<n;j++) b[1][j]=a[0][0]+a[0][j]+a[1][j];
36
37        // urmatoarele linii din b
38        for(i=2;i<m;i++)
39            for(j=0;j<n;j++)
40            b[i][j]=a[i][j]+a[i-1][j]+minLinia(i-1,j);
41
42        // "obligatoriu" (!) si ultima linie (i=n-1) dar ... fara coborare
43        b[m-1][n-1]=minLinia(m-1,n-1)+a[m-1][n-1];
44
45        out.println(b[m-1][n-1]);
46        out.close();
47        t2=System.currentTimeMillis();
48        System.out.println("Timp = "+(t2-t1));
49    } // main()
50
51    static int minLinia(int ii, int jj) // min pe linia=ii fara pozitia jj==co
52    {
53        int j,min=oo;
54        for(j=0;j<n;j++)
55            if(j!=jj)
56                if(b[ii][j]<min) min=b[ii][j];
57        return min;
58    }

```

```
58 } // minLinia(...)  
59 } // class
```

A doua variantă:

Listing 15.1.2: lacusta2.java

```
1 import java.io.*; // suplimentar ... si traseul !  
2 class Lacusta2  
3 {  
4     static final int oo=100000;  
5     static int m,n;  
6     static int[][] a,b; // 1 <= i <= m; 1 <= j <= n  
7  
8     public static void main(String[] args) throws IOException  
9     {  
10         long t1,t2;  
11         t1=System.currentTimeMillis();  
12  
13         int i,j,min,jmin,j0;  
14  
15         StreamTokenizer st=new StreamTokenizer(  
16             new BufferedReader(new FileReader("lacusta.in")));  
17         PrintWriter out=new PrintWriter(  
18             new BufferedWriter(new FileWriter("lacusta.out")));  
19         st.nextToken(); m=(int)st.nval;  
20         st.nextToken(); n=(int)st.nval;  
21         a=new int[m+1][n+1];  
22         b=new int[m+1][n+1];  
23  
24         for(i=1;i<=m;i++)  
25             for(j=1;j<=n;j++)  
26             {  
27                 st.nextToken(); a[i][j]=(int)st.nval;  
28             }  
29             for(i=1;i<=m;i++) for(j=1;j<=n;j++) b[i][j]=oo;  
30  
31             // prima linie (i=1) din b este oo  
32             // a doua linie (i=2) din b  
33             for(j=2;j<=n;j++) b[2][j]=a[1][1]+a[1][j]+a[2][j];  
34  
35             // urmatoarele linii din b  
36             for(i=3;i<=m;i++)  
37                 for(j=1;j<=n;j++) b[i][j]=a[i][j]+a[i-1][j]+minLinia(i-1,j);  
38  
39             // "obligatoriu" (!) si ultima linie (i=n) dar ... fara coborare  
40             b[m][n]=minLinia(m,n)+a[m][n];  
41  
42             out.println(b[m][n]);  
43             out.close();  
44  
45             jmin=-1; // initializare aiurea !  
46             j0=1; // pentru linia 2  
47             System.out.print(1+" "+1+" --> ");  
48             for(i=2;i<=m-1;i++) // liniile 2 .. m-1  
49             {  
50                 min=oo;  
51                 for(j=1;j<=n;j++)  
52                     if(j!=j0)  
53                         if(b[i][j]<min) { min=b[i][j]; jmin=j; }  
54                         System.out.print((i-1)+" "+jmin+" --> ");  
55                         System.out.print(i+" "+jmin+" --> ");  
56                         j0=jmin;  
57             }  
58  
59             j0=n;  
60             min=oo;  
61             for(j=1;j<n;j++)  
62                 if(j!=j0)  
63                     if(b[i][j]<min) { min=b[i][j]; jmin=j; }  
64                     System.out.print((i-1)+" "+jmin+" --> ");  
65                     System.out.print(i+" "+jmin+" --> ");  
66                     System.out.println(m+" "+n);  
67  
68             t2=System.currentTimeMillis();  
69             System.out.println("Timp = "+(t2-t1));
```

```

70 } // main()
71
72 static int minLinia(int ii, int jj) // min pe linia=ii fara pozitia jj==col
73 {
74     int j,min=oo;
75     for(j=1;j<=n;j++)
76         if(j!=jj)
77             if(b[ii][j]<min) min=b[ii][j];
78     return min;
79 } // minLinia(...)
80 } // class

```

Varianta 3:

Listing 15.1.3: lacusta3.java

```

1 import java.io.*; // fara matricea de costuri (economie de "spatiu")
2 class Lacusta3 // traseul este ... pentru depanare
3 {           // daca se cere ... se poate inregistra si apoi ...
4     static final int oo=100000;
5     static int m,n;
6     static int[][] a; // 1 <= i <= m;      1 <= j <= n
7
8     public static void main(String[] args) throws IOException
9     {
10        long t1,t2;
11        t1=System.currentTimeMillis();
12
13        int i,j;
14        int minc,minsol,jmin,j0;
15
16        StreamTokenizer st=new StreamTokenizer(
17            new BufferedReader(new FileReader("lacusta.in")));
18        PrintWriter out=new PrintWriter(
19            new BufferedWriter(new FileWriter("lacusta.out")));
20        st.nextToken(); m=(int)st.nval;
21        st.nextToken(); n=(int)st.nval;
22        a=new int[m+1][n+1];
23
24        for(i=1;i<=m;i++)
25            for(j=1;j<=n;j++)
26            {
27                st.nextToken(); a[i][j]=(int)st.nval;
28            }
29
30        minsol=oo;
31        System.out.print(1+" "+1+" --> ");
32
33        // a doua linie (i=2)
34        minc=oo;
35        jmin=-1;
36        for(j=2;j<=n;j++)
37            if(a[1][1]+a[1][j]+a[2][j]<minc) {minc=a[1][1]+a[1][j]+a[2][j]; jmin=j;}
38        System.out.print(1+" "+jmin+" --> ");
39        System.out.print(2+" "+jmin+" --> ");
40        minsol=minc;
41        j0=jmin;
42
43        jmin=-1; // initializare aiurea !
44        for(i=3;i<=m-1;i++)
45        {
46            minc=oo;
47            for(j=1;j<=n;j++)
48                if(j!=j0)
49                    if(a[i-1][j]+a[i][j]<minc)
50                        {minc=a[i-1][j]+a[i][j]; jmin=j;}
51            System.out.print((i-1)+" "+jmin+" --> ");
52            System.out.print(i+" "+jmin+" --> ");
53            minsol+=minc;
54            j0=jmin;
55        }
56
57        j0=n;
58        minc=oo;
59        for(j=1;j<=n;j++)
60            if(j!=j0)

```

```

61         if(a[m-1][j]+a[m][j]<minc)
62             {minc=a[m-1][j]+a[m][j]; jmin=j;}
63         System.out.print((m-1)+" "+jmin+" --> ");
64         System.out.print(m+" "+jmin+" --> ");
65         minsol+=minc+a[m][n];
66         System.out.println(m+" "+n);
67
68         out.println(minsol);
69         out.close();
70
71         t2=System.currentTimeMillis();
72         System.out.println("Timp = "+(t2-t1));
73     } // main()
74 } // class

```

Varianta 4:

Listing 15.1.4: lacusta4.java

```

1 import java.io.*; // fara matricea de costuri (economie de "spatiu")
2 class Lacusta4 // si ... fara matricea initiala (numai doua linii din ea !)
3 { // calculez pe masura ce citesc cate o linie !
4     static final int oo=100000;
5     static int m,n;
6     static int[][] a; // 1 <= i <= m;      1 <= j <= n
7
8     public static void main(String[] args) throws IOException
9     {
10         long t1,t2;
11         t1=System.currentTimeMillis();
12
13         int i,j,ii,jj;
14         int minc,minsol,jmin,j0;
15
16         StreamTokenizer st=new StreamTokenizer(
17             new BufferedReader(new FileReader("lacusta.in")));
18         PrintWriter out=new PrintWriter(
19             new BufferedWriter(new FileWriter("lacusta.out")));
20         st.nextToken(); m=(int)st.nval;
21         st.nextToken(); n=(int)st.nval;
22         a=new int[2][n+1];
23
24         for(i=1;i<=2;i++) // citesc numai primele doua linii
25             for(j=1;j<=n;j++)
26             {
27                 st.nextToken(); a[i%2][j]=(int)st.nval;
28             }
29
30         minsol=oo;
31         System.out.print(1+" "+1+" --> ");
32
33         // a doua linie (i=2)
34         minc=oo;
35         jmin=-1;
36         for(j=2;j<=n;j++)
37             if(a[1%2][1]+a[1][j]+a[2%2][j]<minc)
38                 {minc=a[1%2][1]+a[1%2][j]+a[2%2][j]; jmin=j;}
39         System.out.print(1+" "+jmin+" --> ");
40         System.out.print(2+" "+jmin+" --> ");
41         minsol=minc;
42         j0=jmin;
43
44         jmin=-1; // initializare aiurea !
45         for(i=3;i<=m-1;i++) // citesc mai departe cate o linie
46         {
47             for(j=1;j<=n;j++) { st.nextToken(); a[i%2][j]=(int)st.nval; }
48             minc=oo;
49             for(j=1;j<=n;j++)
50                 if(j!=j0)
51                     if(a[(i-1+2)%2][j]+a[i%2][j]<minc)
52                         {minc=a[(i-1+2)%2][j]+a[i%2][j]; jmin=j;}
53             System.out.print((i-1)+" "+jmin+" --> ");
54             System.out.print(i+" "+jmin+" --> ");
55             minsol+=minc;
56             j0=jmin;
57         }

```

```

58
59 //citesc linia m
60 for(j=1; j<=n; j++) { st.nextToken(); a[m%2][j]=(int)st.nval; }
61
62 j0=n;
63 minc=oo;
64 for(j=1; j<=n; j++)
65 if(j!=j0)
66 if(a[(m-1+2)%2][j]+a[m%2][j]<minc)
67 {minc=a[(i-1+2)%2][j]+a[i%2][j]; jmin=j;}
68 System.out.print((i-1)+" "+jmin+" --> ");
69 System.out.print(i+" "+jmin+" --> ");
70 minsol+=minc+a[m%2][n];
71 System.out.println(m+" "+n);
72
73 out.println(minsol);
74 out.close();
75
76 t2=System.currentTimeMillis();
77 System.out.println("Timp = "+(t2-t1));
78 } // main()
79 } // class

```

Varianta 5:

Listing 15.1.5: lacusta5.java

```

1 import java.io.*; // numai o linie din matricea initiala !
2 class Lacusta5
3 {
4     static final int oo=100000;
5     static int m,n;
6     static int[] a; // 1 <= j <= n
7
8     public static void main(String[] args) throws IOException
9     {
10         long t1,t2;
11         t1=System.currentTimeMillis();
12
13         int i,j,ii,jj,a11,aa;
14         int minc,minsol,jmin,j0;
15
16         StreamTokenizer st=new StreamTokenizer(
17             new BufferedReader(new FileReader("lacusta.in")));
18         PrintWriter out=new PrintWriter(
19             new BufferedWriter(new FileWriter("lacusta.out")));
20         st.nextToken(); m=(int)st.nval;
21         st.nextToken(); n=(int)st.nval;
22         a=new int[n+1];
23
24         // citesc numai prima linie
25         for(j=1; j<=n; j++) { st.nextToken(); a[j]=(int)st.nval; }
26         System.out.print(1+" "+1+" --> ");
27         a11=a[1];
28
29         // citesc a doua linie
30         st.nextToken(); a[1]=(int)st.nval;
31         minc=oo;
32         jmin=-1;
33         for(j=2; j<=n; j++)
34         {
35             aa=a[j];
36             st.nextToken(); a[j]=(int)st.nval;
37             if(aa+a[j]<minc) {minc=aa+a[j]; jmin=j;}
38         }
39         System.out.print(1+" "+jmin+" --> "+2+" "+jmin+" --> ");
40         minsol=a11+minc;
41         j0=jmin;
42
43         // citesc mai departe cate o linie si ...
44         for(i=3; i<=m-1; i++)
45         {
46             minc=oo;
47             jmin=-1;
48             for(j=1; j<=n; j++)
49             {

```

```

50         aa=a[j];
51         st.nextToken(); a[j]=(int)st.nval;
52         if(j!=j0)   if(aa+a[j]<minc) {minc=aa+a[j]; jmin=j;}
53     }
54     System.out.print((i-1)+" "+jmin+" --> "+i+" "+jmin+" --> ");
55     minsol+=minc;
56     j0=jmin;
57 }
58
59 // citoare linia m (primele n-1 componente)
60 minc=oo;
61 jmin=-1;
62 for(j=1; j<=n-1; j++)
63 {
64     aa=a[j];
65     st.nextToken(); a[j]=(int)st.nval;
66     if(aa+a[j]<minc) {minc=aa+a[j]; jmin=j;}
67 }
68 System.out.print((m-1)+" "+jmin+" --> "+m+" "+jmin+" --> ");
69 minsol+=minc;
70 j0=jmin;
71
72 // citesc ultimul element
73 st.nextToken(); a[n]=(int)st.nval;
74 minsol+=a[n];
75 System.out.println(m+" "+n);
76
77 out.println(minsol);
78 out.close();
79
80 t2=System.currentTimeMillis();
81 System.out.println("Timp = "+(t2-t1));
82 } // main()
83 } // class

```

15.1.3 Cod sursă

Listing 15.1.6: lacusta.c

```

1 #include <stdio.h>
2 #define M 100
3
4 int main()
5 {
6     FILE *fi,*fo;
7     unsigned int a[M][M],b[M][M],m,n,i,j,min1,min2,jmin;
8     fi=fopen("lacusta.in", "rt");
9
10    fscanf(fi,"%u %u", &m, &n);
11    for(i=0; i<m; i++)
12        for(j=0; j<n; j++)
13            fscanf(fi,"%u",&a[i][j]);
14
15    fclose(fi);
16    b[1][0]=32000;
17    for(i=1; i<n; i++)
18        b[1][i]=a[0][0]+a[0][i]+a[1][i];
19    for(i=1; i<m-1; i++)
20    {
21        if(b[i][0]<=b[i][1])
22        {
23            min1=b[i][0];
24            min2=b[i][1];
25            jmin=0;
26        }
27        else
28        {
29            min1=b[i][1];
30            min2=b[i][0];
31            jmin=1;
32        }
33        for(j=2; j<n; j++)

```

```

34         if(b[i][j]<min1)
35         {
36             min2=min1;
37             min1=b[i][j];
38             jmin=j;
39         }
40     else
41         if(b[i][j]<min2)
42             min2=b[i][j];
43     for(j=0; j<n; j++)
44         if(j!=jmin)
45             b[i+1][j]=min1+a[i][j]+a[i+1][j];
46         else
47             b[i+1][j]=a[i][j]+a[i+1][j]+min2;
48     }
49
50     min1=b[m-1][0];
51     for (j=1; j<n; j++)
52         if(b[m-1][j]<min1) min1=b[m-1][j];
53
54     fo=fopen("lacusta.out", "wt");
55     fprintf(fo,"%u\n", min1+a[m-1][n-1]);
56     fclose(fo);
57     return 0;
58 }
```

15.2 Scara

Marinel Serban, Emanuela Cerchez

Ion și-a construit o vilă pe frumosul vârf al unui munte. Acum proiectează o scară specială, pe care va urca de la șosea până la vilă. Diferența de nivel dintre șosea și vilă este H (deci aceasta trebuie să fie înălțimea totală a scării). Scara va avea N trepte, toate de aceeași lățime, dar de înălțimi distințe două câte două.

Ion a sesizat că efortul pe care îl depune pentru a urca o treaptă este egal cu înălțimea treptei. Dar dacă el urcă x trepte deodată, efortul depus este egal cu media aritmetică a înălțimilor acestor x trepte pe care le urcă deodată + un efort de valoare constantă p (necesar pentru a-și lua avânt).

Fiind un tip atletic, Ion poate urca mai multe trepte deodată, dar suma înălțimilor treptelor urcate deodată nu trebuie să depășească o valoare maximă M .

Cerință

Scrieți un program care să determine efortul minim necesar pentru a urca pe o scară construită conform restricțiilor problemei, precum și o modalitate de a construi scara care va fi urcată cu efort minim.

Datele de intrare

Fișierul de intrare *scara.in* va conține pe prima linie 4 numere naturale separate prin câte un spațiu $H \ N \ M \ p$ (cu semnificația din enunț).

Datele de ieșire

Fișierul de ieșire *scara.out* va conține

- pe prima linie va fi scris efortul minim necesar (cu 2 zecimale cu rotunjire);
- pe cea de a doua linie vor fi scrise N numere naturale nenule care reprezintă înălțimile celor N trepte ale scării (în ordinea de la șosea către vilă), separate prin câte un spațiu.

Restricții și precizări

- $0 < H \leq 75$
- $0 < N \leq 8$
- $0 < M < 14$
- $0 \leq p \leq 10$
- Pentru datele de test, problema are întodeauna soluție.
- Dacă există mai multe soluții (modalități de a construi scara astfel încât să obțineți efortul minim dorit), veți afișa prima soluție în ordine lexicografică.
- Spunem că vectorul $x = (x_1, x_2, \dots, x_k)$ precedă în ordine lexicografică vectorul $y = (y_1, y_2, \dots, y_k)$ dacă există $i \geq 1$ astfel încât $x_j = y_j$, pentru orice $j < i$ și $x_i < y_i$.

- Dacă a doua zecimală a efortului minim este 0, sau chiar ambele zecimale sunt 0 nu este necesar să le afișați. Deci în exemplu s-ar fi putut scrie efortul minim 9 sau 9.0.

Punctaj

Se acordă 40% din punctaj pentru prima cerință (efortul minim).

Dacă efortul minim este corect și se afișează și o soluție corectă (care respectă restricțiile problemei și corespunde efortului minim), dar această soluție nu este prima din punct de vedere lexicografic, se obține 80% din punctaj.

Pentru rezolvarea corectă și completă a ambelor cerințe se obține 100% din punctaj.

Exemple

scara.in	scara.out
10 4 5 2	9.00
	1 4 2 3

Timp maxim de executare: 5 secunde/test

15.2.1 Indicații de rezolvare

Ginfo nr. 15/3 martie 2005

Pentru rezolvarea acestei probleme vom utiliza *metoda backtracking*.

Pentru aceasta vom construi un vector a care va conține înălțimile scărilor, precum și un vector sol care va conține, în fiecare moment, cea mai bună soluție.

Deasemenea, vom construi un vector b care va conține efortul care trebuie depus pentru a urca scările (al i -lea element al acestui vector indică efortul necesar pentru a urca primele i scări).

Pentru a ne asigura că înălțimile sunt distințe vom păstra un vector de valori logice în care se vor marca înălțimile folosite în fiecare moment.

Deasemenea, la fiecare pas, va fi păstrată suma totală a înălțimilor scărilor construite la pașii anteriori.

În timpul algoritmului, trebuie să ne asigurăm că valorile vectorului b sunt cuprinse între 1 și M și sunt distințe. Deasemenea, suma lor trebuie să fie egală cu H .

La fiecare pas i , valoarea b_i va fi calculată pe baza formulei:

$$b_i = \min(b_{i-1} + a_i, \min(b_j + p + (a_{j+1} + a_{j+2} + \dots + a_i) / (i - j)))$$

unde j variază între 1 și $i - 2$, iar sumele de forma $a_{j+1} + a_{j+2} + \dots + a_i$ sunt luate în considerare numai dacă sunt cel mult egale cu M .

Dacă am reușit să ajungem la ultimul pas, vom verifica dacă soluția curentă este mai bună decât cea considerată anterior a fi cea mai bună. În acest caz atât vectorul sol , precum și efortul total minim, sunt actualizate.

În final, vom afișa efortul minim obținut, precum și elementele vectorului sol .

15.2.2 Rezolvare detaliată

Listing 15.2.1: scara.java

```

1 import java.io.*;
2 class Scara
3 {
4     static int h,n,m,p;
5     static int hmax,s;
6     static int[] x,xsol;
7     static double[] e;
8     static double esol;
9     static boolean[] ales;
10    static final int oo=8*13+1;
11
12    public static void main(String[] args) throws IOException
13    {
14        long t1,t2;
15        t1=System.currentTimeMillis();
16
17        StreamTokenizer st=new StreamTokenizer(
18            new BufferedReader(new FileReader("1-scara.in")));

```

```

19     PrintWriter out=new PrintWriter(
20             new BufferedWriter(new FileWriter("scara.out")));
21
22     st.nextToken(); h=(int)st.nval;
23     st.nextToken(); n=(int)st.nval;
24     st.nextToken(); m=(int)st.nval;
25     st.nextToken(); p=(int)st.nval;
26
27     x=new int[n+1];
28     e=new double[n+1];
29     xsol=new int[n+1];
30
31     hmax=h-((n-1)*n)/2;
32     if(m<hmax) hmax=m; // restrictie ptr mai multe trepte ==> si pentru una !
33
34     ales=new boolean[hmax+1];
35
36     esol=oo;
37     s=0;
38     f(1);
39
40     out.println((double)((int)(esol*100+0.5))/100); // 2 zecimale cu rotunjire!!!
41     for(int k=1;k<=n; k++) out.print(xsol[k]+" ");
42     out.println();
43     out.close();
44     t2=System.currentTimeMillis();
45     System.out.println("Timp = "+(t2-t1));
46 } // main()
47
48 static void f(int k) // plasez valori pe pozitia k
49 {
50     int i,j;
51     for(i=1;i<=hmax; i++)
52     {
53         if(ales[i]) continue;
54         if(s+i>h) break; // e deja prea mult !!!
55
56         x[k]=i;
57         ales[i]=true;
58         s=s+i;
59
60         if(k<n) f(k+1); else efort();
61
62         s=s-i;
63         ales[i]=false;
64     }
65 } // f(...)
66
67 static void efort()
68 {
69     if(s!=h) return;
70
71     int i,j,k,sij;
72     double e1,e2;
73
74     for(i=1;i<=n; i++)
75     {
76         e1=e[i-1]+x[i]; // pas=o treapta ==> efort=inaltime treapta
77         e2=oo;
78         sij=x[i]+x[i-1]; // pas=mai multe trepte ==> p+media_aritmetica
79         j=i-1;
80         while((j>=1)&&(sij<=m))
81         {
82             if(e[j-1]+p+(double)sij/(i-j+1)<e2)
83                 e2=e[j-1]+p+(double)sij/(i-j+1);
84             j--;
85             sij+=x[j];
86         }
87         if(e1<e2) e[i]=e1; else e[i]=e2;
88     } //for i
89
90     if(e[n]<esol-0.001)
91     {
92         esol=e[n];
93         for(k=1;k<=n; k++) xsol[k]=x[k];
94     }

```

```
95     } // efort()
96 } //class
```

15.2.3 Cod sursă

Listing 15.2.2: scara.cpp

```
1 #include <fstream>
2 #include <iomanip>
3
4 using namespace std;
5
6 #define InFile "scara.in"
7 #define OutFile "scara.out"
8 #define NMax 101
9 #define HMax 101
10
11 int sol[NMax], solmin[NMax], uz[HMax], htot;
12 double ef[NMax], efmin;
13 //ef[i]=efortul minim cu care urc primele i trepte din sol
14 int N, H, p, M;
15
16 void Citire();
17 void Gen(int);
18 void Afisare();
19 double efort ();
20
21 int main()
22 {
23     Citire();
24     efmin=(double)H*N+1;
25     Gen(1);
26     Afisare();
27     return 0;
28 }
29
30 void Citire()
31 {ifstream fin(InFile);
32     fin>>H>>N>>M>>p;
33     fin.close();}
34
35 void Afisare()
36 {
37     int i;
38     ofstream fout(OutFile);
39     fout<<setprecision(2)<<efmin<<endl;
40     for (i=1; i<N; i++)
41         fout<<solmin[i]<<' ';
42     fout<<solmin[N]<<endl;
43     fout.close();
44 }
45
46 double efort()
47 {
48     int k, j;
49     double x, sum;
50     for (k=1; k<=N; k++)
51         {x=sol[k]+ef[k-1]; // urc cu efort minim primele k-1 trepte
52          // si apoi urc treapta k de inaltime i
53          sum=sol[k];
54          for (j=2; k-j>=0; j++)
55              {// urc deodata j trepte k, k-1, ..., k-j+1, daca
56              // suma inaltimilor lor este <=M
57              // in sum calculez suma inaltimilor
58              sum+=sol[k-j+1];
59              if (sum>M) break;
60              if (sum/j+p+ef[k-j]<x)
61                  x=sum/j+ef[k-j]+p;
62          }
63          ef[k]=x;
64      }
65     return ef[N];
```

```
66 }
67
68 void Gen(int k)
69 {
70 int i;
71 double x;
72 if (k==N+1)
73 {
74     if (htot==H)
75         {x=efort();
76          if (x<efmin && efmin-x>0.001)
77              {efmin=x;
78               for (i=1; i<=N; i++) solmin[i]=sol[i];
79             }
80     }
81 else
82     for (i=1; i<=H && htot+i<=H && i<=M; i++)
83         if (!uz[i])
84         {
85             //care ar fi efortul minim daca as pune inaltimea treptei k=i?
86             sol[k]=i; htot+=i; uz[i]=1;
87             Gen(k+1);
88             uz[i]=0; htot-=i;
89         }
90 }
```

Capitolul 16

OJI 2004

16.1 Perle

Granița nu se trece ușor. Asta pentru că Balaurul Arhirel (mare pasionat de informatică) nu lasă pe nimeni să treacă decât după ce răspunde la niște întrebări ...

În acea țară există trei tipuri de perle normale (le vom nota cu 1, 2 și 3) și trei tipuri de perle magice (le vom nota cu A , B și C). Perlele magice sunt deosebite prin faptul că se pot transforma în alte perle (una sau mai multe, normale sau magice).

Perla magică de tipul A se poate transforma în orice perlă normală (una singură).

Perla magică de tipul B se poate transforma într-o perlă normală de tipul 2 și una magică de tipul B , sau într-o perlă normală de tipul 1, una magică de tipul A , una normală de tipul 3, una magică de tipul A și una magică de tipul C .

Perla magică de tipul C se poate transforma într-o perlă normală de tipul 2 sau într-o perlă normală de tipul 3, una magică de tipul B și una magică de tipul C sau într-o perlă normală de tipul 1, una normală de tipul 2 și una magică de tipul A .

Ca să rezumăm cele de mai sus putem scrie:

$$\begin{array}{rcl} A & \longrightarrow & 1 \quad | \quad 2 \quad | \quad 3 \\ B & \longrightarrow & 2B \quad | \quad 1A3AC \\ C & \longrightarrow & 2 \quad | \quad 3BC \quad | \quad 12A \end{array}$$

Balaurul Arhirel ne lasă la început să ne alegem o perlă magică (una singură), iar apoi folosind numai transformările de mai sus trebuie să obținem un anumit sir de perle normale. Când o perlă magică se transformă, perlele din stânga și din dreapta ei rămân la fel (și în aceeași ordine). De asemenea ordinea perlelor rezultate din transformare este chiar cea prezentată mai sus.

De exemplu, dacă balaurul ne cere să facem sirul de perle 21132123, putem alege o perlă magică de tipul B și următorul sir de transformări:

$$B \longrightarrow 2B \longrightarrow 21A3AC \longrightarrow 21A3A12A \longrightarrow 21132123.$$

Întrucât Balaurul nu are prea multă răbdare, el nu ne cere decât să spunem dacă se poate sau nu obține sirul respectiv de perle.

Cerință

Să se determine pentru fiecare sir de intrare dacă se poate obține prin transformările de mai sus sau nu (alegând orice primă perlă magică, la fiecare sir).

Datele de intrare

Fișierul de intrare **perle.in** are următoarea structură:

- pe prima linie numărul N , reprezentând numărul de siruri din fișierul de intrare
- urmează N linii; a i -a linie dintre cele N descrie sirul i , printr-o succesiune de numere naturale despărțite de către un spațiu. Primul număr reprezintă lungimea sirului L_i , iar următoarele L_i numere sunt tipurile de perle normale, în ordine, de la stânga la dreapta.

Datele de ieșire

Fișierul **perle.out** va conține N linii. Pe linia i se va scrie un singur număr 1 sau 0 (1 dacă se poate obține sirul respectiv (al i -lea) și 0 dacă nu se poate).

Restricții și precizări

- $0 < N < 11$

- $0 < L_i < 10001$, pentru oricare i

Exemplu

perle.in	perle.out
3	1
8 2 1 1 3 2 1 2 3	0
2 2 2	1
1 3	

Timp maxim de executare: 1 secundă/test

16.1.1 Indicații de rezolvare

Mihai Stroe, Ginfo nr. 14/4 aprilie 2004

Initial, problema pare dificilă, dar, după examinarea transformărilor posibile, se observă că este destul de simplă.

Fie un anumit sir de perle. Presupunem că acesta a fost obținut dintr-o perlă magică și ne oprim la prima contradicție.

Să determinăm, la început, perlă magică din care s-ar putea obține sirul.

Dacă sirul are lungimea 1, el se poate obține dintr-o perlă magică A.

Dacă sirul începe cu 12 și are trei perle, s-ar putea obține dintr-o perlă C.

Dacă sirul începe cu 2 și are cel puțin două perle, singura șansă de a-l obține este de a începe cu o perlă magica de tip B. Similar dacă începe cu 1 și are mai mult de trei perle.

Dacă sirul începe cu 3, s-ar putea obține numai dintr-o perlă de tip C.

Acum, având în vedere că am stabilit singura perlă magică din care sunt șanse să se obțină sirul, vom verifica dacă sirul poate fi într-adevăr obținut.

Pentru aceasta, putem scrie câte o funcție pentru fiecare perlă magică. O astfel de funcție se aplică pe sirul de intrare și execută operații (avansări pe sir sau apeluri ale funcțiilor pentru alte tipuri de perle) în funcție de tipul perlelor întâlnite.

Dacă la un moment dat nu există nici o regulă de continuare (de exemplu, pentru sirul 22), sirul nu poate fi obținut.

Funcția pentru A nu este necesară, tratarea perlei magice A putând fi inclusă în celelalte două funcții. Se observă că, în cadrul fiecărei funcții, acțiunile efectuate sunt alese determinist. Astfel, la fiecare pas se alege o acțiune posibilă sau se întrerupe căutarea și se semnalează faptul că sirul nu poate fi obținut.

Funcțiile se apeleză recursiv. Pentru a evita depășirea stivei, recursivitatea se poate simula iterativ.

Analiza complexității

La fiecare moment, deciziile se iau în timp constant. Ordinul de complexitate al operației de citire a datelor de intrare este $O(L)$, unde L reprezintă suma lungimilor sirurilor de perle din fișierul de intrare.

Fiecare caracter din fiecare sir este parcurs o singură dată.

Ordinul de complexitate al algoritmului este $O(L_i)$ pentru un sir de perle de lungime L_i .

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(L)$.

16.1.2 Rezolvare detaliată

Listing 16.1.1: perle.java

```

1 import java.io.*;
2 class Perle
3 {
4     static final int a=4, b=5, c=6;
5     static byte[] x,y;
6     static int n,lg,j1,j2;
7     static boolean ok;
8
9     public static void main(String[] args) throws IOException
10    {
11        long t1,t2;
12        t1=System.currentTimeMillis();
13        int k,i;
```

```

14
15     StreamTokenizer st=new StreamTokenizer(
16             new BufferedReader(new FileReader("perle.in")));
17     PrintWriter out=new PrintWriter(
18             new BufferedWriter(new FileWriter("perle.out")));
19     st.nextToken(); n=(int)st.nval;
20     for(k=1;k<=n;k++)
21     {
22         st.nextToken(); lg=(int)st.nval;
23         x=new byte[lg+5];
24         y=new byte[lg+5];
25         for(i=1;i<=lg;i++) {st.nextToken(); x[i]=(byte)st.nval;}
26
27         ok=true;
28         determinStart();
29         if(!ok) {out.println(0); continue;}
30
31         //afis();
32         j1=1;
33         j2=2;           // prima pozitie libera
34
35         while((ok)&&(j1<=lg))
36         {
37             cautNeterminal();
38             if(ok) schimbNeterminal();
39             if(j2>lg+1) ok=false;
40             //afis();
41         }
42
43         if(ok) out.println(1); else out.println(0);
44     }// for k
45
46     out.close();
47     t2=System.currentTimeMillis();
48     System.out.println("Timp = "+(t2-t1));
49 } // main()
50
51 static void schimbNeterminal()
52 {
53     if(y[j1]==a) {y[j1]=x[j1]; j1++;}
54     else
55     if(y[j1]==b)
56     {
57         if(x[j1]==1) inserez1A3AC();
58         else if(x[j1]==2) inserez2B();
59         else ok=false;
60     }
61     else // y[j1]==c
62     {
63         if(x[j1]==1) inserez12A();
64         else if(x[j1]==2) {y[j1]=2; j1++;}
65         else inserez3BC();
66     }
67 } // schimbNeterminal()
68
69 static void cautNeterminal()
70 {
71     while((j1<j2)&&(y[j1]<a)&&ok)
72     {
73         if(y[j1]!=x[j1]) ok=false;
74         j1++;
75     }
76 } // cautNeterminal()
77
78 static void inserez1A3AC()
79 {
80     int j;
81     j2=j2+4;
82     for(j=j2-1;j>=j1+4;j--) y[j]=y[j-4];
83     y[j1+0]=1;
84     y[j1+1]=a;
85     y[j1+2]=3;
86     y[j1+3]=a;
87     y[j1+4]=c;
88 } // inserez1A3AC()
89

```

```

90     static void inserez2B()
91     {
92         int j;
93         j2=j2+1;
94         for(j=j2-1;j>=j1+1;j--) y[j]=y[j-1];
95         y[j1+0]=2;
96         y[j1+1]=b;
97     } // inserez2B()
98
99     static void inserez12A()
100    {
101        int j;
102        j2=j2+2;
103        for(j=j2-1;j>=j1+2;j--) y[j]=y[j-2];
104        y[j1+0]=1;
105        y[j1+1]=2;
106        y[j1+2]=a;
107    } // inserez12A()
108
109    static void inserez3BC()
110    {
111        int j;
112        j2=j2+2;
113        for(j=j2-1;j>=j1+2;j--) y[j]=y[j-2];
114        y[j1+0]=3;
115        y[j1+1]=b;
116        y[j1+2]=c;
117    } // inserez3BC()
118
119    static void determinStart() // determin "neterminalul" de start
120    {
121        if(x[1]==1)
122        {
123            if(lg==1) y[1]=a;
124            else if((lg==3)&&(x[2]==2)) y[1]=c;
125            else if(lg>=5) y[1]=b;
126            else ok=false;
127        }
128        else
129        {
130            if(x[1]==2)
131            {
132                if(lg==1) y[1]=a;
133                else if(lg>=2) y[1]=b;
134                else ok=false;
135            }
136            else
137            {
138                if(lg==1) y[1]=a;
139                else if(lg>=3) y[1]=c;
140                else ok=false;
141            }
142        } // determinStart()
143
144    static void afis()
145    {
146        afisv(x);afisv(y);System.out.println();
147    }
148
149    static void afisv(byte[] v)
150    {
151        int i;
152        for(i=1;i<v.length;i++)
153            if(v[i]==a) System.out.print('A');
154            else if(v[i]==b) System.out.print('B');
155            else if(v[i]==c) System.out.print('C');
156            else if(v[i]==0) System.out.print(" ");
157            else System.out.print(v[i]);
158        System.out.println();
159    } // afisv...
160 } // class

```

16.1.3 Cod sursă

Listing 16.1.2: perle.c

```

1 #include <stdio.h>
2 #define NMAX 10000
3 int st[2][NMAX+10]; /* 1, 2, 3 e evident si 4, 5, 6 e A, B, C */
4
5 int main(void)
6 {
7     int t,l,v,i;
8     int niv[2];
9     int valid[2];
10
11    freopen("perle.in","r",stdin);
12    freopen("perle.out","w",stdout);
13
14    scanf("%d",&t);
15    while (t--)
16    {
17        scanf("%d",&l);
18        if (l == 1)
19        {
20            scanf("%d",&v);
21            printf("1\n");
22            continue;
23        }
24        else
25        {
26            valid[1] = valid[0] = 1;
27            niv[1] = niv[0] = 1;
28            st[0][0] = 5;
29            st[1][0] = 6;
30            while (l--)
31            {
32                scanf("%d",&v);
33                for(i = 0; i < 2; i++)/* trist dar doar 2 valori vreau */
34                {
35                    if (valid[i])
36                    {
37                        if (st[i][niv[i]-1] < 4)
38                        {
39                            valid[i] = (st[i][--niv[i]]==v);
40                            continue;
41                        }
42
43                        if (st[i][niv[i]-1] == 4)
44                        {
45                            niv[i]--;
46                            continue;
47                        }
48
49                        if (st[i][niv[i]-1]==5)
50                        {
51                            if (v == 3) valid[i] = 0;
52                            if (v == 1)
53                            {
54                                niv[i]--;
55                                st[i][niv[i]++] = 6;
56                                st[i][niv[i]++] = 4;
57                                st[i][niv[i]++] = 3;
58                                st[i][niv[i]++] = 4;
59                            }
60
61                            /* 2-ul e ok :D */
62                            continue;
63                        }
64
65                        /* e C in stiva */
66                        if (v == 2) niv[i]--;
67                        if (v == 1)
68                        {
69                            niv[i]--;
70                            st[i][niv[i]++] = 4;
71                            st[i][niv[i]++] = 2;
72                        }
73
74                        if (v == 3)

```

```

75             {
76                 niv[i]--;
77                 st[i][niv[i]++] = 6;
78                 st[i][niv[i]++] = 5;
79             }
80         }
81         if (!niv[i] && 1)
82             valid[i] = 0;
83     }
84 }
85 printf("%d\n", ((!niv[0] && valid[0]) || (!niv[1] && valid[1])));
86 }
87 }
88 return 0;
}

```

16.2 Romeo și Julieta

În ultima ecranizare a celebrei piese shakespeareiene Romeo și Julieta trăiesc într-un oraș modern, comunică prin e-mail și chiar învață să programeze. Într-o secvență tulburătoare sunt prezentate frămătările interioare ale celor doi eroi încercând fără succes să scrie un program care să determine un punct optim de întâlnire.

Ei au analizat harta orașului și au reprezentat-o sub forma unei matrice cu n linii și m coloane, în matrice fiind marcate cu spațiu zonele prin care se poate trece (străzi lipsite de pericole) și cu X zonele prin care nu se poate trece. De asemenea, în matrice au marcat cu R locul în care se află locuința lui Romeo, iar cu J locul în care se află locuința Julietai.

Ei se pot deplasa numai prin zonele care sunt marcate cu spațiu, din poziția curentă în oricare dintre cele 8 poziții învecinate (pe orizontală, verticală sau diagonale).

Cum lui Romeo nu îl place să aștepte și nici să se lase așteptat n-ar fi tocmai bine, ei au hotărât că trebuie să aleagă un punct de întâlnire în care atât Romeo, cât și Julieta să poată ajunge în același timp, plecând de acasă. Fiindcă la întâlniri amândoi vin într-un suflet, ei estimatează timpul necesar pentru a ajunge la întâlnire prin numărul de elemente din matrice care constituie drumul cel mai scurt de acasă până la punctul de întâlnire. Și cum probabil există mai multe puncte de întâlnire posibile, ei vor să îl aleagă pe cel în care timpul necesar pentru a ajunge la punctul de întâlnire este minim.

Cerință

Scrieți un program care să determine o poziție pe hartă la care Romeo și Julieta pot să ajungă în același timp. Dacă există mai multe soluții, programul trebuie să determine o soluție pentru care timpul este minim.

Datele de intrare

Fișierul de intrare **rj.in** conține:

- pe prima linie numerele naturale NM , care reprezintă numărul de linii și respectiv de coloane ale matricei, separate prin spațiu;
- pe fiecare dintre următoarele N linii se află M caractere (care pot fi doar R , J , X sau spațiu) reprezentând matricea.

Datele de ieșire

Fișierul de ieșire **rj.out** va conține o singură linie pe care sunt scrise trei numere naturale separate prin câte un spațiu $tmin\ x\ y$, având semnificația:

- $x\ y$ reprezintă punctul de întâlnire (x - numărul liniei, y - numărul coloanei);
- $tmin$ este timpul minim în care Romeo (respectiv Julieta) ajunge la punctul de întâlnire.

Restricții și precizări

- $1 < N, M < 101$
- Liniile și coloanele matricei sunt numerotate începând cu 1.
- Pentru datele de test există întotdeauna soluție.

Exemple

rj.in

```

5 8
XXR XXX
 X X X
J X X X
      XX
XXX XXXX

```

rj.out

```
4 4 4
```

Explicație:

Traseul lui Romeo poate fi: (1,3), (2,4), (3,4), (4,4). Timpul necesar lui Romeo pentru a ajunge de acasă la punctul de întâlnire este 4.

Traseul Julietei poate fi: (3,1), (4,2), (4,3), (4,5). Timpul necesar Julietei pentru a ajunge de acasă la punctul de întâlnire este deasemenea 4.

În plus 4, este punctul cel mai apropiat de ei cu această proprietate.

Timp maxim de executare: 1 secundă/test

16.2.1 Indicații de rezolvare

Mihai Stroe, Ginfo nr. 14/4 aprilie 2004

Problema se rezolvă folosind *algoritmul lui Lee*.

Se aplică acest algoritm folosind ca puncte de start poziția lui Romeo și poziția Julietei.

Vom folosi o matrice D în care vom pune valoarea 1 peste tot pe unde nu se poate trece, valoarea 2 în poziția în care se află Romeo inițial, valoarea 3 în poziția în care se află Julietă inițial și valoarea 0 în rest.

La fiecare pas k vom parcurge această matrice și în pozițiile vecine celor care au valoarea 2 vom pune valoarea 2, dacă acestea au valoarea 0 sau 3. Dacă o poziție are valoare 3, înseamnă că la un moment de timp anterior Julietă se putea afla în poziția respectivă. La același pas k vom mai parcurge matricea o dată și în pozițiile vecine celor care au valoarea 3 vom pune valoarea 3, dacă acestea au valoarea 0.

Dacă la pasul k poziția vecină uneia care are valoarea 3, are valoarea 2, atunci ne vom opri și k reprezintă momentul minim de timp după care cei doi se întâlnesc, iar poziția care are valoare 2 reprezintă locul întâlnirii.

La prima vedere s-ar părea că numărul k nu reprezintă momentul de timp minim la care cei doi se întâlnesc. Vom demonstra că algoritmul este corect prin metoda reducerii la absurd. Pentru aceasta avem în vedere că pozițiile marcate cu 2 reprezintă toate locurile în care se poate afla Romeo după cel mult k pași, iar cele marcate cu 3 reprezintă toate locurile în care se poate afla Julietă după cel mult k pași. Dacă k nu reprezintă momentul de timp minim la care cei doi se întâlnesc înseamnă că acesta a fost determinat mai devreme și algoritmul s-a oprit deja.

Analiza complexității

Ordinul de complexitate al operației de citire a datelor de intrare este $O(M \times N)$.

Ordinul de complexitate al acestui algoritm este $O(k \times M \times N)$, unde k reprezintă momentul în care cei doi se întâlnesc.

Ordinul de complexitate al operației de scriere a rezultatului este $O(1)$.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(k \times M \times N)$.

16.2.2 Rezolvare detaliată

Listing 16.2.1: rj.java

```

1 import java.io.*;
2 class RJ
3 {
4     static final int zid=10000;
5     static int m,n;
6     static int[][] xr,xj;
7     static int[] q1=new int[5000]; // coada sau coada circulara mai bine !
8     static int[] q2=new int[5000]; // coada

```

```

9
10    static int ic, sc;           // ic=inceput coada
11
12    public static void main(String[] args) throws IOException
13    {
14        long t1,t2;
15        t1=System.currentTimeMillis();
16
17        int i,j,ir=0,jr=0,ij=0,jj=0, imin, jmin, tmin;
18        String s;
19
20        BufferedReader br=new BufferedReader(new FileReader("rj.in"));
21        StreamTokenizer st=new StreamTokenizer(br);
22        PrintWriter out=new PrintWriter(
23            new BufferedWriter(new FileWriter("rj.out")));
24
25        st.nextToken(); m=(int)st.nval;
26        st.nextToken(); n=(int)st.nval;
27
28        xr=new int[m+2][n+2];          // matrice bordata cu zid !
29        xj=new int[m+2][n+2];          // matrice bordata cu zid !
30
31        br.readLine();                // citeste CRLF !!!
32        for(i=1;i<=m;i++)
33        {
34            s=br.readLine();
35            for(j=1;j<=n;j++)
36                if(s.charAt(j-1)=='X') xr[i][j]=xj[i][j]=zid;           // zid!
37                else if(s.charAt(j-1)=='R') {ir=i; jr=j; xr[i][j]=1;}
38                else if(s.charAt(j-1)=='J') {ij=i; jj=j; xj[i][j]=1;}
39        }
40
41        for(i=0;i<=m+1;i++) xr[i][0]=xr[i][n+1]=xj[i][0]=xj[i][n+1]=zid; // E si V
42        for(j=0;j<=n+1;j++) xr[0][j]=xr[m+1][j]=xj[0][j]=xj[m+1][j]=zid; // N si S
43
44        ic=sc=0;                     // coada vida;
45        qj[sc]=jr; qj[sc]=jr; sc++; // (ir, jr) --> coada
46        while(ic!=sc)
47        {
48            i=qj[ic]; j=qj[ic]; ic++; // scot din coada
49            fill(xr,i,j);
50        }
51
52        ic=sc=0;                     // coada vida;
53        qj[sc]=ij; qj[sc]=jj; sc++; // (ij, jj) --> coada
54        while(ic!=sc)
55        {
56            i=qj[ic]; j=qj[ic]; ic++; // scot din coada
57            fill(xj,i,j);
58        }
59
60        tmin=10000;
61        imin=jmin=0;
62        for(i=1;i<=m;i++)
63            for(j=1;j<=n;j++)
64                if(xr[i][j]==xj[i][j])
65                    if(xj[i][j]!=0)           // pot exista pozitii ramase izolate !
66                        if(xr[i][j]<tmin) {tmin=xr[i][j]; imin=i; jmin=j;}
67
68        out.println(tmin+" "+imin+" "+jmin);
69        out.close();
70        t2=System.currentTimeMillis();
71        System.out.println("Timp = "+(t2-t1));
72    } // main()
73
74    static void fill(int[][] x,int i, int j)
75    {
76        int t=x[i][j];             // timp
77        if(x[i-1][j]==0) {x[i-1][j]=t+1; qj[sc]=i-1; qj[sc]=j; sc++;} // N
78        if(x[i-1][j+1]==0) {x[i-1][j+1]=t+1; qj[sc]=i-1; qj[sc]=j+1; sc++;} // NE
79        if(x[i-1][j-1]==0) {x[i-1][j-1]=t+1; qj[sc]=i-1; qj[sc]=j-1; sc++;} // NV
80
81        if(x[i+1][j]==0) {x[i+1][j]=t+1; qj[sc]=i+1; qj[sc]=j; sc++;} // S
82        if(x[i+1][j+1]==0) {x[i+1][j+1]=t+1; qj[sc]=i+1; qj[sc]=j+1; sc++;} // SE
83        if(x[i+1][j-1]==0) {x[i+1][j-1]=t+1; qj[sc]=i+1; qj[sc]=j-1; sc++;} // SV
84

```

```

85     if(x[i][j+1]==0)      {x[i][j+1]=t+1;    q[i][sc]=i;    q[j][sc]=j+1; sc++;} // E
86     if(x[i][j-1]==0)      {x[i][j-1]=t+1;    q[i][sc]=i;    q[j][sc]=j-1; sc++;} // V
87 } // fil(...)
88 } // class

```

16.2.3 Cod sursă

Listing 16.2.2: rj.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 #define InFile "rj.in"
6 #define OutFile "rj.out"
7 #define NMax 102
8 #define NV 8
9
10 int n, m, xr, yr, xj, yj;
11 int dl[NV]={0, 1, 0, -1, -1, 1, -1, 1};
12 int dc[NV]={1, 0, -1, 0, -1, 1, 1,-1};
13 char l[NMax][NMax];
14 int r[NMax][NMax];
15
16 void citire(void);
17 void afisare(int [NMax][NMax]);
18 void parurge (int, int, int[NMax][NMax]);
19
20 int main()
21 {
22     int j[NMax][NMax];
23     citire();
24     parurge(xr, yr, r);
25     parurge(xj, yj, j);
26     afisare(j);
27     return 0;
28 }
29
30 void citire(void)
31 {
32     int i, k;
33     char c;
34
35     ifstream f(InFile);
36     f>>n>>m;
37
38     for (i=0; i<=n+1; i++) l[i][0]=l[i][m+1]='X';
39     for (i=0; i<=m+1; i++) l[0][i]=l[n+1][i]='X';
40     f.get(c);
41
42     for (i=1; i<=n; i++)
43     {
44         for (k=1; k<=m; k++)
45         {
46             f.get(c);
47             l[i][k]=c;
48             if (l[i][k]=='R') {xr=i; yr=k; l[i][k]=' ';}
49             if (l[i][k]=='J') {xj=i; yj=k; l[i][k]=' ';}
50         }
51         f.get(c);
52     }
53     f.close();
54 }
55
56 void parurge (int x0, int y0, int d[NMax][NMax])
57 {
58     struct Punct
59     {
60         int l, c;
61     } C[NMax*NMax], p;
62
63     int inc=0, sf=0, i, k;

```

```

64     for (i=0; i<=n+1; i++)
65         for (k=0; k<=m+1; k++)
66             d[i][k]=-1;
67
68     C[0].l=x0;
69     C[0].c=y0;
70     d[x0][y0]=1;
71     while (inc<=sf)
72     {
73         p=C[inc++];
74         for (i=0; i<NV; i++)
75             if (l[p.l+d1[i]][p.c+dc[i]]==' ' && d[p.l+d1[i]][p.c+dc[i]]==-1)
76             {
77                 d[p.l+d1[i]][p.c+dc[i]]=1+d[p.l][p.c];
78                 C[++sf].l=p.l+d1[i];
79                 C[sf].c=p.c+dc[i];
80             }
81     }
82 }
83
84 void afisare(int j[NMax][NMax])
85 {
86     ofstream f(OutFile);
87     int tmin=NMax*NMax+5, xmin=-1, ymin=-1, i, k;
88
89     for (i=1; i<=n; i++)
90         for (k=1; k<=m; k++)
91             if (r[i][k]==j[i][k])
92                 if (r[i][k]<tmin && r[i][k]!=-1)
93                 {
94                     tmin=r[i][k];
95                     xmin=i;
96                     ymin=k;
97                 }
98     f<<tmin<<' '<<xmin<<' '<<ymin<<endl;
99     f.close();
100 }
```

Capitolul 17

OJI 2003

17.1 Spirala

Se consideră un automat de criptare format dintr-un tablou cu n linii și n coloane, tablou ce conține toate numerele de la 1 la n^2 așezate "șerpuit" pe linii, de la prima la ultima linie, pe linile impare pornind de la stânga către dreapta, iar pe cele pare de la dreapta către stânga (ca în figura alăturată).

Numim "amestecare" operația de desfășurare în spirală a valorilor din tablou în ordinea indicată de săgeți și de reașezare a acestora în același tablou, "șerpuit" pe linii ca și în cazul precedent.

De exemplu, desfășurarea tabloului conduce la sirul: 1 2 3 4 5 12 13 14 15 16 9 8 7 6 11 10, iar reașezarea acestuia în tablou conduce la obținerea unui nou tablou reprezentat în cea de-a doua figură alăturată.

1	2	3	4
8	7	6	5
9	10	11	12
16	15	14	13

Figura 17.1: Spirala1

1	2	3	4
14	13	12	5
15	16	9	8
10	11	6	7

Figura 17.2: Spirala2

1	2	3	4
6	7	8	5
11	10	15	14
16	9	12	13

Figura 17.3: Spirala3

După orice operație de amestecare se poate relua procedeul, efectuând o nouă amestecare. S-a observat un fapt interesant: că după un număr de amestecări, unele valori ajung din nou în poziția inițială (pe care o aveau în tabloul de pornire). De exemplu, după două amestecări, tabloul de 4×4 conține 9 dintre elementele sale în exact aceeași poziție în care se aflau inițial (vezi elemente marcate din figură).

Cerință

Pentru n și k citite, scrieți un program care să determine numărul minim de amestecări ale unui tablou de n linii necesar pentru a ajunge la un tablou cu exact k elemente aflate din nou în poziția inițială.

Datele de intrare

Fișierul de intrare **spirala.in** conține pe prima linie cele două numere n și k despărțite printr-un spațiu.

Datele de ieșire

Fișierul de ieșire **spirala.out** conține o singură linie pe care se află numărul de amestecări determinat.

Restricții și precizări

- $3 \leq N \leq 50$
- Datele de intrare sunt alese astfel încât numărul minim de amestecări necesare să nu depășească $2 * 10^9$
- La încheierea programului nu se va solicita apăsarea unei taste

Exemple

spirala.in	spirala.out	spirala.in	spirala.out
4 9	2	6 36	330

Timp maxim de executare: 1 secundă/test

17.1.1 Indicații de rezolvare

Se calculează pentru fiecare poziție numărul de amestecări după care se repetă poziția respectivă (perioada principală).

Se calculează pentru toți divizorii d ai celui mai mic multiplu comun al numerelor calculate (ținut ca factori primi și exponenții corespunzători) numărul de poziții care se repetă după d amestecări.

Sursa comisiei generează divizorii cu bkt pe exponenții descompunerii în factori primi.

17.1.2 Rezolvare detaliată

Prima variantă:

Listing 17.1.1: spirala1.java

```

1 import java.io.*;           // teste 5,6,7,8,9,10 ==> timp mare!
2 class Spiralal
3 {
4     static int n,k;
5     static int[] pr;
6     static int[] p;
7
8     public static void main(String[] args) throws IOException
9     {
10         long t1,t2;
11         t1=System.currentTimeMillis();
12
13         int r=0,i,npf;
14         StreamTokenizer st=new StreamTokenizer(new BufferedReader(
15             new FileReader("spirala.in")));
16         PrintWriter out=new PrintWriter(new BufferedWriter(
17             new FileWriter("spirala.out")));

```

```

18     st.nextToken(); n=(int)st.nval;
19     st.nextToken(); k=(int)st.nval;
20
21     p=new int[n*n+1];
22     pr=new int[n*n+1];
23
24     constrp();
25     r=0;
26     for(i=1;i<=n*n;i++) pr[i]=i;
27
28     npf=nrPuncteFixe(pr);
29     while(npf!=k)
30     while((npf!=k)&&(r<4200)) // test 4 4200 ? 14280
31     {
32         r++;
33         pr=inmp(pr,p);
34         npf=nrPuncteFixe(pr);
35     }
36     System.out.print(r+":\t"); afisv(pr); System.out.println(" ==> "+npf);
37
38     out.println(r);
39     out.close();
40     t2=System.currentTimeMillis();
41     System.out.println(t2-t1);
42 }
43
44 static void constrp()
45 {
46     int i,j,k,v,kp=0;
47     int[][] a=new int[n+1][n+1];
48     i=1; v=0;
49     for(k=1;k<=n/2;k++)
50     {
51         for(j=1;j<=n;j++) a[i][j]=++v;
52         for(j=n;j>=1;j--) a[i+1][j]=++v;
53         i=i+2;
54     }
55     if(n%2==1) for(j=1;j<=n;j++) a[n][j]=++v;
56
57     // afism(a);
58     for(k=1;k<=n/2;k++) // contur dreptunghi k
59     {
60         i=k;
61         for(j=k;j<=n+1-k;j++) p[++kp]=a[i][j];
62         j=n+1-k;
63         for(i=k+1;i<=n-k;i++) p[++kp]=a[i][j];
64         i=n+1-k;
65         for(j=n+1-k;j>=k;j--) p[++kp]=a[i][j];
66         j=k;
67         for(i=n-k;i>=k+1;i--) p[++kp]=a[i][j];
68     }
69     if(n%2==1) p[n*n]=a[n/2+1][n/2+1]; // corectie ptr n=impar !!!
70     // afisv(p);
71 } // constrp()
72
73 static int[] inmp(int[] a,int[] b)
74 {
75     int i;
76     int[] c=new int[n*n+1];
77     for(i=1;i<=n*n;i++) c[i]=a[b[i]];
78     return c;
79 }
80
81 static int nrPuncteFixe(int[] pr)
82 {
83     int i, s=0;
84     for(i=1;i<=n*n;i++) if(pr[i]==i) s++;
85     return s;
86 }
87
88 static void afism(int[][] x)
89 {
90     int i,j;
91     for(i=1;i<=n;i++)
92     {
93         System.out.println();

```

```

94         for(j=1;j<=n;j++) System.out.print(x[i][j]+"\t");
95     }
96     System.out.println();
97 }
98
99 static void afisv(int[] x)
100 {
101     int i;
102     for(i=1;i<=n*n;i++) System.out.print(x[i]+" ");
103 }
104 } // class

```

A doua variantă:

Listing 17.1.2: spirala2.java

```

1 import java.io.*;           // teste 5,   6, ... ==> timp mare!
2 class Spirala2            // v1      33s    17s
3 {                          // v2      28s    14s ==> nu este un castig prea mare !
4     static int n,k,nn;
5     static int[] pr;
6     static int[] p;
7     static int[] pp;
8
9     public static void main(String[] args) throws IOException
10    {
11        long t1,t2;
12        t1=System.currentTimeMillis();
13
14        int r=0,i,npf;
15        StreamTokenizer st=new StreamTokenizer(new BufferedReader(
16                    new FileReader("spirala.in")));
17        PrintWriter out=new PrintWriter(new BufferedWriter(
18                    new FileWriter("spirala.out")));
19        st.nextToken();n=(int)st.nval;
20        st.nextToken();k=(int)st.nval;
21        nn=n*n-n-1;
22        p=new int[n*n+1];
23        pr=new int[n*n+1];
24        pp=new int[nn+1];
25
26        constrp();
27        r=0;
28        for(i=1;i<=nn;i++) pr[i]=i;
29        for(i=n+2;i<=n*n;i++) pp[i-n-1]=p[i]-n-1;
30
31        npf=nrPuncteFixe(pr);
32        while((npf!=k-n-1)&&(r<81840))
33        {
34            r++;
35            pr=inmp(pr,pp);
36            npf=nrPuncteFixe(pr);
37
38            if(r%1000==0) System.out.println(r+" "+npf);
39        }
40        System.out.println(r+" "+npf);
41        out.println(r);
42        out.close();
43        t2=System.currentTimeMillis();
44        System.out.println(t2-t1);
45    }
46
47     static void constrp()
48    {
49        int i,j,k,v,kp=0;
50        int[][] a=new int[n+1][n+1];
51        i=1; v=0;
52        for(k=1;k<=n/2;k++)
53        {
54            for(j=1;j<=n;j++) a[i][j]=++v;
55            for(j=n;j>=1;j--) a[i+1][j]=++v;
56            i=i+2;
57        }
58        if(n%2==1) for(j=1;j<=n;j++) a[n][j]=++v;
59
60        for(k=1;k<=n/2;k++) // contur dreptunghi k

```

```

61      {
62          i=k;
63          for(j=k; j<=n+1-k; j++) p[++kp]=a[i][j];
64          j=n+1-k;
65          for(i=k+1; i<=n-k; i++) p[++kp]=a[i][j];
66          i=n+1-k;
67          for(j=n+1-k; j>=k; j--) p[++kp]=a[i][j];
68          j=k;
69          for(i=n-k; i>=k+1; i--) p[++kp]=a[i][j];
70      }
71      if(n%2==1) p[n*n]=a[n/2+1][n/2+1]; // corectie ptr n=impar !!!
72 } // constrp()
73
74 static int[] inmp(int[] a, int[] b)
75 {
76     int i;
77     int[] c=new int[n*n+1];
78     for(i=1; i<=nn; i++) c[i]=a[b[i]];
79     return c;
80 }
81
82 static int nrPuncteFixe(int[] pr)
83 {
84     int i, s=0;
85     for(i=1; i<=nn; i++) if(pr[i]==i) s++;
86     return s;
87 }
88 } // class

```

Listing 17.1.3: spirala3.java

```

1 import java.io.*;           // OK ! cu cmmmc al ciclurilor
2 class Spirala3
3 {
4     static int n,k,nc,ncsol, nd;
5     static int[] p;
6     static int[] c;
7     static boolean[] vizitat;
8     static long[] lgc;           // lg cicluri
9     static long[] lgcsol;        // lg cicluri din sol ...
10    static long[] dcmmm=new long[15]; // divizorii lui cmmm
11
12    static int[] npf;           // nr puncte fixe
13    static long min=Long.MAX_VALUE;
14
15    public static void main(String[] args) throws IOException
16    {
17        long t1,t2;
18        t1=System.currentTimeMillis();
19
20        int i;
21
22        StreamTokenizer st=new StreamTokenizer(new BufferedReader(
23                                         new FileReader("spirala.in")));
24        PrintWriter out=new PrintWriter(new BufferedWriter(
25                                         new FileWriter("spirala.out")));
26
27        st.nextToken(); n=(int)st.nval;
28        st.nextToken(); k=(int)st.nval;
29
30        p=new int[n*n+1];
31        c=new int[n*n+1];
32        vizitat=new boolean[n*n+1];
33
34        constrp();
35        cicluri();
36
37        nc=0;
38        for(i=2; i<=n*n; i++) if(c[i]>0) nc++;
39
40        lgc=new long[nc];
41        npf=new int[nc];
42        lgcsol=new long[nc];
43
44        k=k-c[1];

```

```

45
46     nc=0;
47     for(i=2;i<=n*n;i++) if(c[i]>0) {lgc[nc]=i; npf[nc]=i*c[i]; nc++;}
48
49     for(i=0;i<(1<<nc);i++) if(suma(i)==k) verific(i);
50     out.println(min);
51     out.close();
52
53     t2=System.currentTimeMillis();
54     System.out.println("Timp="+ (t2-t1));
55 }
56
57 static boolean amGasit(int val, long[] x, int p, int u) // x=vector crescator
58 {
59     int m;
60     while(p<=u)
61     {
62         m=(p+u)/2;
63         if(x[m]==val) return true;
64         if(x[m]<val) p=m+1; else u=m-1;
65     }
66     return false;
67 }
68
69 static void verific(int i)
70 {
71     int d,j;
72     int ncsol=0;
73     long cmmm;
74     cmmm=1L;
75
76     j=0;
77     while(i!=0)
78     {
79         if(i%2!=0) // bit=1 pe pozitia j
80         {
81             cmmm=cmmmc(cmmm,lgc[j]);
82             if(cmmm>2000000000) return; // din enunt !!!
83             lgcsol[ncsol]=lgc[j];
84             ncsol++; // nr cicluri
85         }
86         j++;
87         i/=2;
88     }
89
90     // verific divizorii lui cmmm - pot aparea combinatii ...!!!
91     descFact(cmmm);
92     int nd1=1<<nd;
93     for(i=0;i<nd1;i++)
94     {
95         d=divizor(i);
96         if(!amGasit(d,lgcsol,0,ncsol-1))
97             if(amGasit(d,lgc,0,nc-1)) return;
98     }
99
100    if(cmmm<min) min=cmmm;
101 } // verific(...)

102
103 static int divizor(int i) // pentru generarea tuturor divizorilor
104 {
105     int p=1, j=0;
106     for(j=0; j<nd; j++) if((i&(1<<j))!=0) p*=dcmmm[j];
107     return p;
108 }

109
110 static int suma(int i)
111 {
112     int s=0, j=0;
113     for(j=0; j<nc; j++) if((i&(1<<j))!=0) s+=npf[j];
114     return s;
115 }

116
117 static void constrp()
118 {
119     int i,j,k,v,kp=0;
120     int[][] a=new int[n+1][n+1];

```

```

121     i=1; v=0;
122     for(k=1;k<=n/2;k++)
123     {
124         for(j=1;j<=n;j++) a[i][j]=++v;
125         for(j=n;j>=1;j--) a[i+1][j]=++v;
126         i=i+2;
127     }
128     if(n%2==1) for(j=1;j<=n;j++) a[n][j]=++v;
129
130     for(k=1;k<=n/2;k++) // contur dreptunghi k
131     {
132         i=k;
133         for(j=k;j<=n+1-k;j++) p[++kp]=a[i][j];
134         j=n+1-k;
135         for(i=k+1;i<=n-k;i++) p[++kp]=a[i][j];
136         i=n+1-k;
137         for(j=n+1-k;j>=k;j--) p[++kp]=a[i][j];
138         j=k;
139         for(i=n-k;i>=k+1;i--) p[++kp]=a[i][j];
140     }
141     if(n%2==1) p[n*n]=a[n/2+1][n/2+1]; // corectie ptr n=impar !!!
142 } // constrp()
143
144 static void cicluri()
145 {
146     int i,j,np;
147     for(i=1;i<=n*n;i++)
148     {
149         if(vizitat[i]) continue;
150         vizitat[i]=true;
151         np=1;
152         j=p[i]; vizitat[j]=true;
153         while(j!=i) {np++; j=p[j]; vizitat[j]=true;}
154         c[np]++;
155     }
156 } // cicluri()
157
158 static long cmmmc(long a, long b) // cmmmc(a,b)=(a*b)/cmmdc(a,b)=...
159 {
160     long dab,da;
161     long rez;
162     dab=cmmdc(a,b);
163     da=cmmdc(dab,a);
164     a/=da;
165     dab/=da;
166     b/=dab;
167     rez=a*b;
168     return rez;
169 } // cmmmc(...)
170
171 static long cmmdc(long a, long b)
172 {
173     long d,i,r;
174     if(a>b) {d=a; i=b;} else {d=b; i=a;}
175     while(i!=0) {r=d%i; d=i; i=r;}
176     return d;
177 } // cmmdc(...)
178
179 static void descFact(long nr)
180 {
181     nd=0; // nr divizori
182     long d=2;
183     if((nr==0) || (nr==1)) return;
184     while(nr%d==0) {nr=nr/d; dcmmm[nd]=d; nd++;}
185     d=3;
186     while((d*d<=nr) && (nr!=1)) // poate sa ramana un numar prim!
187     {
188         while(nr%d==0) {nr=nr/d; dcmmm[nd]=d; nd++;}
189         d=d+2;
190     }
191     if(nr!=1) {dcmmm[nd]=d; nd++;}
192 } // descFact(...)
193 } // class

```

17.1.3 Cod sursă

Listing 17.1.4: spirala2.pas

```

1 const max=50;
2     d:array[1..2,0..3]of shortint=((0,1,0,-1),(1,0,-1,0));
3
4 type divi=record care,cati:longint end;
5     fprim=record care,ex:word end;
6     mat=array[0..max+1,0..max+1]of word;
7     vect=array[1..max*max]of word;
8     vectm=array[1..200]of byte;
9
10 var x:vect;
11    dv:array[1..200]of divi;cd:byte;
12    fp:array[1..200]of fprim;cfp:byte;
13    sol:vectm;
14    n:byte;v,min,cati:longint;
15
16 procedure calc;
17 var a,b,c:mat;
18    v:vect;
19    i,j,k,imp,rez,sens:longint;
20 begin
21     for i:=0 to n+1 do begin a[i,0]:=0;a[i,n+1]:=0;a[0,i]:=0;a[n+1,i]:=0 end;
22     for i:=1 to n do
23       for j:=1 to n do c[i,j]:=0;
24     k:=1;
25     for i:=1 to n do
26       if i mod 2=1 then
27         for j:=1 to n do begin a[i,j]:=k; inc(k) end
28       else
29         for j:=n downto 1 do begin a[i,j]:=k;inc(k) end;
30     b:=a;imp:=1;rez:=0;sens:=0;
31     repeat
32       i:=1;j:=1;sens:=0;k:=1;
33       repeat
34         v[k]:=a[i,j];a[i,j]:=0;
35         if a[i+d[1,sens],j+d[2,sens]]=0 then sens:=(sens+1)mod 4;
36         i:=i+d[1,sens];
37         j:=j+d[2,sens];
38         inc(k);
39       until k>n*n;
40     k:=1;
41     for i:=1 to n do
42       if i mod 2=1 then
43         for j:=1 to n do begin
44           a[i,j]:=v[k];
45           if (v[k]=b[i,j])and (c[i,j]=0) then begin
46             c[i,j]:=imp;inc(rez);inc(x[imp])
47           end;
48           inc(k)
49         end
50       else
51         for j:=n downto 1 do begin
52           a[i,j]:=v[k];
53           if (v[k]=b[i,j])and (c[i,j]=0) then begin
54             c[i,j]:=imp;inc(rez);inc(x[imp])
55           end;
56           inc(k)
57         end;
58       inc(imp)
59     until rez=n*n
60   end;
61
62 procedure calc2;
63 var e:boolean;
64   i,j,k,n,ex:longint;
65 begin
66   cd:=0;
67   for i:=1 to max*max do
68     if x[i]>0 then
69       begin
70         inc(cd);

```

```

71           dv[cd].care:=i;
72           dv[cd].cati:=x[i]
73       end;
74   cfp:=0;
75   for i:=2 to cd do begin
76       k:=dv[i].care;
77       for j:=1 to cfp do begin
78           n:=fp[j].care;
79           ex:=0;
80           while k mod n=0 do begin
81               k:=k div n; inc(ex)
82           end;
83           if ex>fp[j].ex then fp[j].ex:=ex
84       end;
85       j:=2;
86       while k>1 do begin
87           if k mod j=0 then begin
88               inc(cfp);fp[cfp].care:=j;
89               fp[cfp].ex:=0;
90               while k mod j=0 do begin
91                   inc(fp[cfp].ex);
92                   k:=k div j
93               end
94           end;
95           inc(j)
96       end
97   end
98 end;
99
100 procedure calc3;
101 var i,j:byte;tot,k:longint;
102     soll:vectm;
103 begin
104     tot:=0;
105     for i:=1 to cd do begin
106         k:=dv[i].care;soll:=sol;
107         for j:=1 to cfp do
108             while (soll[j]>0) and (k mod fp[j].care=0) do
109             begin
110                 k:=k div fp[j].care;
111                 dec(soll[j])
112             end;
113             if k=1 then tot:=tot+dv[i].cati
114         end;
115         write(tot,'-',v,' ');
116         if tot=cati then
117             if v<min then min:=v
118     end;
119
120 procedure back(i:byte);
121 var j:byte;p:longint;
122 begin
123     if i>cfp then calc3
124     else begin
125         p:=1;
126         for j:=0 to fp[i].ex do
127             if v<maxlongint/p then begin
128                 sol[i]:=j;
129                 v:=v*p;
130                 back(i+1);
131                 v:=v div p;
132                 p:=p*fp[i].care;
133             end;
134             sol[i]:=0;
135         end
136     end;
137
138 begin
139     readln(n,cati);
140     calc; calc2;
141     min:=maxlongint;
142     v:=1;
143
144     back(1);
145
146     writeln;

```

```

147   if min=maxlongint then writeln(0)
148   else writeln(min);
149   readln
150 end.
```

17.2 Taxe

Într-o țară în care corupția este în floare și economia la pământ, pentru a obține toate aprobările necesare în scopul demarării unei afaceri, investitorul trebuie să treacă prin mai multe camere ale unei clădiri în care se află birouri.

Clădirea are un singur nivel în care birourile sunt lipite unele de altele formând un caroaj patrat de dimensiune $n \times n$. Pentru a facilita accesul în birouri, toate camerele vecine au uși între ele. În fiecare birou se află un funcționar care pretinde o taxă de trecere prin cameră (taxă ce poate fi, pentru unele camere, egală cu 0). Investitorul intră încrezător prin colțul din stânga-sus al clădirii (cum se vede de sus planul clădirii) și dorește să ajungă în colțul opus al clădirii, unde este ieșirea, plătind o taxă totală cât mai mică.

Cerință

Știind că el are în buzunar S euro și că fiecare funcționar îi ia taxa de cum intră în birou, se cere să se determine dacă el poate primi aprobările necesare și, în caz afirmativ, care este suma maximă de bani care îi rămâne în buzunar la ieșirea din clădire.

Datele de intrare

Fișierul de intrare **taxe.in** conține pe prima linie cele două numere S și n despărțite printr-un spațiu, iar pe următoarele n linii câte n numere separate prin spații ce reprezintă taxele cerute de funcționarii din fiecare birou.

Datele de ieșire

Fișerul de ieșire **taxe.out** conține o singură linie pe care se află numărul maxim de euro care îi rămân în buzunar sau valoarea -1 dacă investitorului nu-i ajung banii pentru a obține aprobarea.

Restricții și precizări

- $3 \leq N \leq 100$
- $1 \leq S \leq 10000$
- Valorile reprezentând taxele cerute de funcționarii din birouri sunt numere naturale, o taxă nedepășind valoarea de 200 de euro.
- La încheierea programului nu se va solicita apăsarea unei taste

Exemplu

taxe.in	taxe.out
10 3	3
1 2 5	
1 3 1	
0 8 1	

Timp maxim de executare: 1 secundă/test

17.2.1 Indicații de rezolvare

Se aplică un algoritm de tip Lee care expandează o coadă ce conține inițial doar starea $(1, 1, S)$ cu toate stările în care se poate ajunge dintr-o poziție dată.

Se adaugă stările noi sau se actualizează stările în care se poate ajunge cu mai mulți bani în buzunar.

17.2.2 Rezolvare detaliată

Listing 17.2.1: taxe.java

```

1 import java.io.*;
2 class Taxe
```

```

3  {
4      static int n,s;
5      static int[][] taxa;
6      static int[][] taxaMin;
7      static final int infinit=200*200;
8
9      public static void main(String[] args) throws IOException
10     {
11         int i,j,min;
12         boolean amOptimizare;
13         long t1,t2;
14         t1=System.currentTimeMillis();
15
16         StreamTokenizer st=new StreamTokenizer(
17                 new BufferedReader(new FileReader("taxe.in")));
18         PrintWriter out=new PrintWriter(
19                 new BufferedWriter(new FileWriter("taxe.out")));
20
21         st.nextToken(); s=(int)st.nval;
22         st.nextToken(); n=(int)st.nval;
23
24         taxa=new int[n+2][n+2];
25         taxaMin=new int[n+2][n+2];
26
27         for(i=1;i<=n;i++)
28             for(j=1;j<=n;j++) { st.nextToken(); taxa[i][j]=(int)st.nval; }
29         for(i=0;i<=n+1;i++)
30             for(j=0;j<=n+1;j++) taxaMin[i][j]=infinit;
31
32         taxaMin[1][1]=taxa[1][1];
33         amOptimizare=true;
34         while(amOptimizare)
35     {
36             amOptimizare=false;
37             for(i=1;i<=n;i++)
38                 for(j=1;j<=n;j++)
39                 {
40                     min=minTaxeVecini(i,j);
41                     if(min+taxa[i][j]<taxaMin[i][j])
42                     {
43                         taxaMin[i][j]=min+taxa[i][j];
44                         amOptimizare=true;
45                     }
46                 } //for
47             } // while
48             if(taxaMin[n][n]<=s) out.println(s-taxaMin[n][n]); else out.println(-1);
49             out.close();
50             t2=System.currentTimeMillis();
51             System.out.println("Timp="+ (t2-t1));
52     } // main()
53
54     static int minTaxeVecini(int i, int j)
55     {
56         int min1,min2;
57         min1=minim(taxaMin[i-1][j],taxaMin[i+1][j]);
58         min2=minim(taxaMin[i][j-1],taxaMin[i][j+1]);
59         return minim(min1,min2);
60     }
61
62     static int minim(int a, int b)
63     {
64         if(a<b) return a; else return b;
65     }
66 }
```

17.2.3 Cod sursă

Listing 17.2.2: taxe.pas

```

1 program taxe;
2 const nm=100;
3 type element=record
```

```

4           x,y:byte;
5           v:word;
6           end;
7
8 var co:array[1..2000] of element;
9 nco:word;
10 a:array[1..nm,1..nm] of byte;
11 sp:array[1..nm,1..nm] of word;
12 n,m:byte;
13 S:word;
14
15 procedure citire;
16 var i,j:byte;
17   f:text;
18 begin
19   assign(f,'pitule.in');reset(f);
20   readln(f,s,n,m);
21   for i:=1 to n do
22     for j:=1 to m do
23       read(f,a[i,j]);
24 close(f);
25 end;
26
27 procedure add(xi,yi:byte;vi:word);
28 var i:word;au:element;
29 begin
30   inc(nco);
31   sp[xi,yi]:=vi;
32   with co[nco] do
33     begin
34       x:=xi;y:=yi;v:=vi;
35     end;
36   i:=nco;
37   while (i>1)and(co[i-1].v<co[i].v)do
38     begin
39       au:=co[i-1];
40       co[i-1]:=co[i];
41       co[i]:=au;
42       dec(i);
43     end;
44 end;
45
46 procedure extr(var xi,yi:byte;var vi:word);
47 begin
48   with co[nco] do
49     begin
50       xi:=x;
51       yi:=y;
52       vi:=v;
53     end;
54   dec(nco);
55 end;
56
57 procedure rezolva;
58 var x,y:byte;v:word;f:boolean;
59 begin
60   fillchar(sp,sizeof(sp),255);
61   add(1,1,a[1,1]);
62
63   f:=true;
64   while (nco>0)and f do
65     begin
66       extr(x,y,v);
67       if (x=n)and(y=m) then f:=false
68       else
69         begin
70           if x<n then if sp[x+1,y]>sp[x,y]+a[x+1,y] then
71             add(x+1,y,sp[x,y]+a[x+1,y]);
72           if y<m then if sp[x,y+1]>sp[x,y]+a[x,y+1] then
73             add(x,y+1,sp[x,y]+a[x,y+1]);
74           if x>1 then if sp[x-1,y]>sp[x,y]+a[x-1,y] then
75             add(x-1,y,sp[x,y]+a[x-1,y]);
76           if y>1 then if sp[x,y-1]>sp[x,y]+a[x,y-1] then
77             add(x,y-1,sp[x,y]+a[x,y-1]);
78         end;
79   end;

```

```
80 end;
81
82 procedure scrie;
83 var f:text;
84 begin
85   assign(f,'spaga.out'); rewrite(f);
86   if sp[n,m]>S then writeln(f,-1)
87   else writeln(f,S-sp[n,m]);
88   close(f);
89 end;
90
91 Begin
92 citire;
93 rezolva;
94 scrie;
95 End.
```

Capitolul 18

OJI 2002

18.1 Cod strămos

Principala misiune a unei expediții științifice este de a studia evoluția vieții pe o planetă nou descoperită. În urma studiilor efectuate, cercetătorii au asociat fiecărui organism viu descoperit pe acea planetă un cod caracteristic.

Codul caracteristic este un număr natural de maximum 200 de cifre zecimale nenule.

De asemenea, cercetătorii au observat că pentru orice organism viu de pe planetă, codurile caracteristice ale strămoșilor săi pe scara evoluției se pot obține prin ștergerea unor cifre din codul caracteristic al organismului respectiv, iar un organism este cu atât mai evoluat cu cât codul său caracteristic are o valoare mai mare.

Cerință

Date fiind codurile caracteristice ale două organisme vii diferite, scrieți un program care să determine codul caracteristic al celui mai evoluat strămos comun al lor.

Datele de intrare

Fișierul de intrare **cod.in** conține

n - codul caracteristic al primului organism

m - codul caracteristic al celui de-al doilea organism

Datele de ieșire

Fișierul de ieșire **cod.out** conține pe prima linie:

p - codul celui mai evoluat strămos comun al lui n și m

Exemplu

cod.in	cod.out
7145	75
847835	

Timp maxim de executare: 1 secundă/test

18.1.1 *Indicații de rezolvare

18.1.2 Rezolvare detaliată

Listing 18.1.1: codstramos1.java

```
1 import java.io.*; // cate modalitati de stergere exista pentru x si pentru y
2 class CodStramos1 // astfel incat sa ramana coduri egale de lg maxima in x si y
3 { // si afisare matrice;
4     static int [][] a; // sunt afisate codurile ramase nu cele sterse
5     static String xx,yy; // asa ca apar dubluri de coduri !!!
6     static char[] x,y,z;
7     static int n,m;
8     static int nsol=0;
9     static PrintWriter out;
```

```

10
11  public static void main(String[] args) throws IOException
12  {
13      int i;
14      BufferedReader br=new BufferedReader(new FileReader("cod.in"));
15      out=new PrintWriter(new BufferedWriter(new FileWriter("cod.out")));
16      xx=br.readLine();
17      yy=br.readLine();
18      n=xx.length(); // coloane
19      m=yy.length(); // linii
20      x=new char[n+1];
21      y=new char[m+1];
22      for(i=0;i<n;i++) x[i+1]=xx.charAt(i);
23      for(i=0;i<m;i++) y[i+1]=yy.charAt(i);
24
25      matrad();
26      afism(a);
27
28      z=new char[a[m][n]+1];
29      sol(m,n,a[m][n]);
30      out.close();
31  }
32
33  static void matrad()
34  {
35      int i,j;
36      a=new int[m+1][n+1];
37
38      for(i=1;i<=m;i++)
39      for(j=1;j<=n;j++)
40          if(x[j]==y[i]) a[i][j]=1+a[i-1][j-1];
41          else a[i][j]=max(a[i-1][j],a[i][j-1]);
42  }
43
44  static void sol(int lin, int col,int k) throws IOException
45  {
46      int i,j,kk;
47      if(k==0)
48      {
49          ++nsol;
50          System.out.print(nsol+" : \t");
51          afisv(z);
52          return;
53      }
54      i=lin;
55      while((i>0)&&(a[i][col]==k))
56      {
57          j=col;
58          while((j>0)&&(a[i][j]==k))
59          {
60              while((j>0)&&
61                  (a[i][j]==k)&&
62                  (x[j]!=y[i]||(a[i-1][j-1]!=(k-1))))
63              {
64                  j--;
65              }
66
67              if((j>0)&&(a[i][j]==k)&&(a[i-1][j-1]==(k-1))&&(x[j]==y[i]))
68              {
69                  z[k]=y[i]; // sau x[j];
70                  sol(i-1,j-1,k-1);
71              }
72              j--;
73          }
74          i--;
75      } //while
76  }
77
78  static int max(int a, int b)
79  {
80      if(a>b) return a; else return b;
81  }
82
83  static void afisv(char[] v)
84  {
85      int i;

```

```

86     for(i=1;i<=v.length-1;i++)
87     {
88         System.out.print(v[i]);
89         out.print(v[i]);
90     }
91     System.out.println();
92     out.println();
93 }
94
95 static void afism(int[][]a)
96 {
97     int i,j;
98
99     System.out.print("      ");
100    for(j=0;j<n;j++) System.out.print(xx.charAt(j)+" ");
101    System.out.println("x");
102
103    System.out.print("  ");
104    for(j=0;j<=n;j++) System.out.print(a[0][j]+" ");
105    System.out.println();
106
107    for(i=1;i<=m;i++)
108    {
109        System.out.print(yy.charAt(i-1)+" ");
110
111        for(j=0;j<=n;j++) System.out.print(a[i][j]+" ");
112        System.out.println();
113    }
114    System.out.println("y\n");
115 }
116 }
```

Listing 18.1.2: codstramos2.java

```

1 import java.io.*;           // Cod maxim lexicografic
2 class CodStramos2    // Cod minim lexicografic ==> modificaare simpla!
3 {
4     static int [][] a;
5     static String xx,yy;
6     static char[] x,y,z;
7     static int n,m;
8
9     public static void main(String[] args) throws IOException
10    {
11        int i;
12        BufferedReader br=new BufferedReader(new FileReader("cod.in"));
13        xx=br.readLine();
14        yy=br.readLine();
15        n=xx.length();   // coloane
16        m=yy.length();   // linii
17        x=new char[n+1];
18        y=new char[m+1];
19        for(i=0;i<n;i++) x[i+1]=xx.charAt(n-1-i);
20        for(i=0;i<m;i++) y[i+1]=yy.charAt(m-1-i);
21        matrad();
22        z=new char[m][n+1];
23        sol(m,n,a[m][n]);
24        PrintWriter out=new PrintWriter(
25            new BufferedWriter( new FileWriter("cod.out")));
26        for(i=z.length-1;i>=1;i--) out.print(z[i]);
27        out.close();
28    }
29
30    static void matrad()
31    {
32        int i,j;
33        a=new int[m+1][n+1];
34
35        for(i=1;i<=m;i++)
36            for(j=1;j<=n;j++)
37                if(x[j]==y[i]) a[i][j]=1+a[i-1][j-1];
38                else          a[i][j]=max(a[i-1][j],a[i][j-1]);
39    }
40
41    static void sol(int lin, int col,int k) throws IOException
```

```

42  {
43      int i, j, kk;
44      //if(k==0) return;
45      i=lin;
46      while((i>0)&&(a[i][col]==k))
47      {
48          j=col;
49          while((j>0)&&(a[i][j]==k))
50          {
51              while((j>0)&&
52                  (a[i][j]==k)&&
53                  (x[j]!=y[i]||(a[i-1][j-1]!=(k-1)))))
54              {
55                  j--;
56              }
57
58              if((j>0)&&(a[i][j]==k)&&(a[i-1][j-1]==(k-1))&&(x[j]==y[i]))
59              {
60                  if(y[i]>z[k])
61                  {
62                      z[k]=y[i]; // sau x[j];
63                      for(kk=1;kk<k;kk++) z[kk]='0'-1; // curat z (cod<'0') in fata lui k
64                      if(k>1) sol(i-1,j-1,k-1);
65                  }
66                  j--;
67              }
68          } //while
69      }
70
71      static int max(int a, int b)
72      {
73          if(a>b) return a; else return b;
74      }
75  }

```

18.1.3 *Cod sursă

18.2 Triangulații

O triangulație a unui poligon convex este o mulțime formată din diagonale ale poligonului care nu se intersectează în interiorul poligonului ci numai în vârfuri și care împart toată suprafața poligonului în triunghiuri.

Fieind dat un poligon cu n vârfuri notate $1, 2, \dots, n$ să se genereze toate triangulațiile distințe ale poligonului. Două triangulații sunt distințte dacă diferă prin cel puțin o diagonală.

Datele de intrare: în fișierul text **triang.in** se află pe prima linie un singur număr natural reprezentând valoarea lui n ($n \leq 11$).

Datele de ieșire: în fișierul text **triang.out** se vor scrie:

- pe prima linie, numărul de triangulații distințe;

- pe fiecare din următoarele linii câte o triangulație descrisă prin diagonalele ce o compun. O diagonală va fi precizată prin două numere reprezentând cele două vârfuri care o definesc; cele două numere ce definesc o diagonală se despart prin cel puțin un spațiu, iar între perechile de numere ce reprezintă diagonalele dintr-o triangulație se va lăsa de asemenea minimum un spațiu.

Exemplu

triang.in	triang.out
5	5
	1 3 1 4
	2 4 2 5
	5 2 5 3
	3 5 3 1
	4 2 1 4

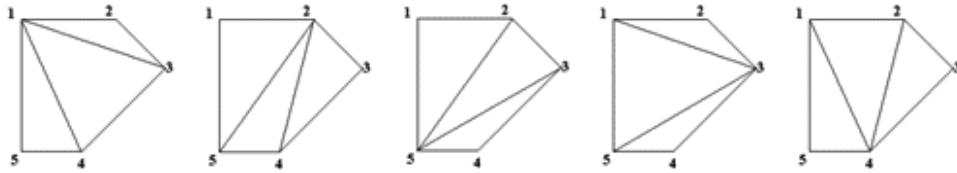


Figura 18.1: Triang

Timp maxim de executare:

7 secunde/test pe un calculator la 133 MHz.

3 secunde/test pe un calculator la peste 500 MHz.

18.2.1 *Indicații de rezolvare**18.2.2 Rezolvare detaliată**

Listing 18.2.1: triangulatii.java

```

1 import java.io.*;                                     // merge si n=12 in 3 sec
2 class Triangulatii
3 {
4     static int n;                                       // numar varfuri poligon
5     static int ndt=n-3;                                 // numar diagonale in triangulatie
6     static int nd=n*(n-3)/2;   // numarul tuturor diagonalelor
7     static int[] x;
8     static int[] v1,v2;
9     static int nsol=0;
10    static PrintWriter out;
11
12    public static void main(String[] args) throws IOException
13    {
14        long t1,t2;
15        t1=System.currentTimeMillis();
16        StreamTokenizer st=new StreamTokenizer(
17            new BufferedReader(new FileReader("triang.in")));
18        out=new PrintWriter(new BufferedWriter(new FileWriter("triang.out")));
19        st.nextToken(); n=(int)st.nval;
20        ndt=n-3;
21        nd=n*(n-3)/2;
22        x=new int[ndt+1];
23        v1=new int[nd+1];
24        v2=new int[nd+1];
25
26        if(n==3) out.println(0);
27        else
28        {
29            out.println(catalan(n-2));
30            diagonale();
31            f(1);
32        }
33        out.close();
34        t2=System.currentTimeMillis();
35        System.out.println("nsol = "+nsol+" Timp = "+(t2-t1));
36    }
37
38    static void afisd() throws IOException
39    {
40        int i;
41        ++nsol;
42        for(i=1;i<=ndt;i++) out.print(v1[x[i]]+" "+v2[x[i]]+" ");
43        out.println();
44    }
45
46    static void diagonale()
47    {
48        int i,j,k=0;
49        i=1;

```

```

50     for(j=3;j<=n-1;j++) {v1[++k]=i; v2[k]=j;}
51
52     for(i=2;i<=n-2;i++)
53       for(j=i+2;j<=n;j++) {v1[++k]=i; v2[k]=j;}
54   }
55
56   static boolean seIntersecteaza(int k, int i)
57   {
58     int j;           // i si x[j] sunt diagonale !!!
59     for(j=1;j<=k-1;j++)
60       if(((v1[x[j]]<v1[i])&&(v1[i]<v2[x[j]])&&(v2[x[j]]<v2[i])) ||
61         ((v1[i]<v1[x[j]])&&(v1[x[j]]<v2[i])&&(v2[i]<v2[x[j]])))
62         return true;
63     return false;
64   }
65
66   static void f(int k) throws IOException
67   {
68     int i;
69     for(i=x[k-1]+1; i<=nd-ndt+k; i++)
70     {
71       if(seIntersecteaza(k,i)) continue;
72       x[k]=i;
73       if(k<ndt) f(k+1); else afis();
74     }
75   }
76
77   static int catalan(int n)
78   {
79     int rez;
80     int i,j;
81     int d;
82     int[] x=new int[n+1];
83     int[] y=new int[n+1];
84
85     for(i=2;i<=n;i++) x[i]=n+i;
86     for(j=2;j<=n;j++) y[j]=j;
87
88     for(j=2;j<=n;j++)
89       for(i=2;i<=n;i++)
90       {
91         d=cmmdc(y[j],x[i]);
92         y[j]=y[j]/d;
93         x[i]=x[i]/d;
94         if(y[j]==1) break;
95       }
96     rez=1;
97     for(i=2;i<=n;i++) rez*=x[i];
98     return rez;
99   }
100
101  static int cmmdc (int a,int b)
102  {
103    int d,i,c,r;
104    if(a>b) {d=a;i=b;} else{d=b;i=a;}
105    while(i!=0) {c=d/i; r=d%i; d=i; i=r;}
106    return d;
107  }
108 } // class

```

18.2.3 *Cod sursă

Partea II

ONI - Olimpiada națională de informatică

Capitolul 19

ONI 2019

19.1 artifact

Problema 1 - artifact

100 de puncte

Arheologii au găsit un artifact care pare să conțină o ecuație care folosește simbolurile unei scrieri necunoscute încă. O serie de ipoteze au început să apară, aşa că ei își propun să rezolve ecuația în vederea descifrării simbolurilor.

Ecuția conține $N + M$ termeni, fiecare termen reprezentând un număr codificat printre-o înșiruire de simboluri, care au fost înlocuite cu literele mari ale alfabetului englez, de la A la Z .

Se presupune că suma primelor N numere trebuie să fie egală cu suma ultimelor M numere. Deasemenea, fiecare literă corespunde unei cifre de la 0 la 9, iar două litere diferite sunt asociate cu două cifre diferite.

Cerințe

Se cere să se afle câte soluții distințe admite ecuația găsită.

Date de intrare

În fișierul **artifact.in** pe prima linie se află două numere naturale nenule N și M separate printre-un spațiu. Pe a doua linie sunt scrise cele $N + M$ siruri de caractere, separate prin câte un spațiu, reprezentând termenii ecuației.

Date de ieșire

În fișierul **artifact.out** se va scrie pe prima linie numărul de soluții distințe ale ecuației date.

Restricții și precizări

- $1 \leq N, 1 \leq M, N + M \leq 5000$
- Fiecare număr este codificat prin cel mult 14 caractere
- Numerele codificate cu cel puțin două cifre nu pot avea cea mai semnificativă cifră egală cu 0
- Se garantează că ecuația are cel puțin o soluție
- Două soluții sunt distințe dacă cel puțin una din litere are valori diferite în cele două soluții

Exemplu

artifact.in	artifact.out	Explicații
3 1 A A A BA	1	$A + A + A = BA$ admite o singura soluție: $5 + 5 + 5 = 15$
2 1 THIS IS EASY	7	$THIS + IS = EASY$ admite 7 soluții: $7962 + 62 = 8024$ $5974 + 74 = 6048$ $1974 + 74 = 2048$ $2974 + 74 = 3048$ $5987 + 87 = 6074$ $1987 + 87 = 2074$ $2987 + 87 = 3074$

Timp maxim de executare/test: Windows: **2.5** secunde, Linux: **1.0** secunde

Memorie: total **128 MB**

Dimensiune maximă a sursei: **20 KB**

19.1.1 Indicații de rezolvare

Autor: prof. Pit-Rada Ionel-Vasile, Colegiul National Traian, Drobeta Turnu Severin

Solutia 1

Pentru fiecare literă care apare în sirurile date se calculează un coeficient obținut ca sumă a puterilor lui zece corespunzătoare rangurilor pozițiilor unde apare acea literă. Să presupunem că avem n litere distințe. Se obține astfel o ecuație cu cel mult zece necunoscute (literele) de forma $x_1 * c_1 + x_2 * c_2 + \dots + x_n * c_n = 0$.

Pentru rezolvarea ecuației se generează *aranjamente* de zece valori (cifrele) luate cate n și pentru fiecare aranjament se verifică în ecuație.

Solutia 2

O altă soluție, care nu va obține punctaj maxim, presupune generarea aranjamentelor de 10 luate cate n și apoi verificarea ecuației prin înlocuirea literelor cu cifre și calcularea sumelor celor N , respective M numere obținute, și apoi verificarea egalității sumelor.

19.1.2 *Rezolvare detaliată

19.1.3 Cod sursă

Listing 19.1.1: artifact.c

```

1 #include <stdio.h>
2 #include <string.h>
3
4 void swap(long long **x, long long *y)
5 {
6     long long t = *x; *x = *y; *y = t;
7 }
8
9 long long count(long long k, long long n, long long sum,
10                  long long *digits, long long *zero, long long **x)
11 {
12     if (k == n) return (sum == 0);
13     long long i, cnt = 0;
14     for (i = k; i < 10; i++)
15     {
16         swap(digits + k, digits + i);
17         if (digits[k] || zero[k])
18         {
19             cnt += count(k + 1, n, sum + digits[k]*x[k], digits, zero, x);
20         }
21         swap(digits + k, digits + i);
22     }
23     return cnt;
24 }
25
26 int main()
27 {
28     FILE    *f = fopen("artifact.in", "r");
29     FILE    *g = fopen("artifact.out", "w");
30
31     char    word[16];
32     long long pos[26], zero[10], x[10];
33     long long digits[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
34     long long n, m, i, j, ten, nLetters = 0;
35
36     for (i = 0; i < 26; i++) pos[i] = -1;
37     for (i = 0; i < 10; i++) zero[i] = +1, x[i] = 0;
38
39     fscanf(f, "%lld%lld", &n, &m);
40     for (i = 1; i <= n; i++)
41     {
42         fscanf(f, "%s", word);
43         for (j = strlen(word) - 1, ten = 1; j >= 0; j--, ten *= 10)
44         {
45             if (pos[word[j] - 'A'] == -1) pos[word[j] - 'A'] = nLetters++;

```

```

46             x[pos[word[j] - 'A']] += ten;
47         }
48     if (strlen(word) > 1) zero[pos[word[0] - 'A']] = 0;
49 }
50
51     for (i = 1; i <= m; i++)
52 {
53     fscanf(f, "%s", word);
54     for (j = strlen(word) - 1, ten = 1; j >= 0; j--, ten *= 10)
55     {
56         if (pos[word[j] - 'A'] == -1) pos[word[j] - 'A'] = nLetters++;
57         x[pos[word[j] - 'A']] -= ten;
58     }
59     if (strlen(word) > 1) zero[pos[word[0] - 'A']] = 0;
60 }
61 fprintf(g, "%lld", count(0, nLetters, 0, digits, zero, x));
62
63 fclose(f);
64 fclose(g);
65
66     return 0;
67 }
```

Listing 19.1.2: artifact0-pr.cpp

```

1 //ecuatie de forma v[1]*c[1]+v[2]*c[2]+...+v[10]*c[10]
2 //generare de aranjamente de 10 luate cate n
3 #include<iostream>
4 #include<string.h>
5
6 using namespace std;
7
8 ifstream fin ("artifact.in");
9 ofstream fout("artifact.out");
10
11 long long N,M,i,j, p[92],a[12],c[92],d[92],v[22],f[92],
12     n,k,l,x,q,nrsol,ok,ok1,s,r,y;
13 ///a[] contine codurile ASCII ale literelor implicate
14 ///p[j]=1 daca litera j este pe prima pozitie
15 ///p[j]=0 in caz contrar
16 ///n=numarul literelor implicate
17 ///v[i] este valoarea temporar asociata cu litera a[i]
18 ///c[j] este coeficientul din ecuatie asociat literei j
19 char t[12];
20 int main()
21 {
22     fin>>N>>M;
23     for(i=1;i<=N;i++)
24     {
25         fin>>t;
26         l=strlen(t);
27         if(l>1) p[t[0]]=1;
28         q=1;
29         for(j=l-1;j>=0;j--)
30         {
31             f[t[j]]++;
32             c[t[j]]+=q;
33             q*=10;
34         }
35     }
36
37     for(i=1;i<=M;i++)
38     {
39         fin>>t;
40         l=strlen(t);
41         if(l>1) p[t[0]]=1;
42         q=1;
43         for(j=l-1;j>=0;j--)
44         {
45             f[t[j]]++;
46             c[t[j]]-=q;
47             q*=10;
48         }
49     }
50 }
```

```

51     n=0;
52     for(i=65;i<=90;i++)
53         if(f[i]>0)
54             n++;a[n]=i;
55
56     /// va trebui sa asociem literelor j valorile v[j] astfel incat
57     /// suma coeficientilor c[j] asociati sa fie nula
58
59     nrsol=0;
60     ///initializare
61     for(i=1;i<=10;i++)
62         v[i]=i-1;
63     r=10;///ultima pozitie ocupata in v[]
64     do
65     {
66         ///verificam ca in v[1...n] sa nu avem 0 asociat unei litere j cu p[j]=1
67         ok=1;
68         for(i=1;i<=n;i++)
69             if(v[i]==0 && p[a[i]]==1)
70                 ok=0;break;
71
72         if(ok==1)
73         {
74             ///verificam daca avem solutie
75             s=0;
76             for(i=1;i<=n;i++)
77             {
78                 s=s+v[i]*c[a[i]];
79                 d[a[i]]=v[i];
80             }
81             if(s==0)
82                 nrsol++;
83         }
84         ///trecem la urmatoarea asociere litere-cifre
85
86         ok1=0;
87         for(i=n;i>=1;i--)
88         {
89             /// cautam cea mai buna inlocuire pentru v[i]
90             /// de observat ca in permanenta secventa v[i+1..10] este
91             /// ordonata strict crescator
92             if(i+1<=10 && v[i]<v[r])
93             {
94                 ///sigur gasim
95                 j=n+1;
96                 while(j<=r && v[i]>v[j])
97                     j++;
98                 x=v[i];
99                 v[i]=v[j];
100                v[j]=x;
101                for(k=n+1,j=i+1;k<=r;j++,k++)
102                    v[j]=v[k];
103                r=j-1;
104                ok1=1;
105                break;
106            }
107            else
108            {
109                ///nu gasim
110                r++;
111                v[r]=v[i];
112            }
113        }
114    } while(ok1==1);
115
116    fout<<nrsol;
117    fout.close();
118    fin.close();
119    return 0;
120 }

```

Listing 19.1.3: artifact1-pr.cpp

```

1 //ecuatie de forma v[1]*c[1]+v[2]*c[2]+...+v[10]*c[10]
2 //generare de aranjamente de 10 luate cate n/2 prin

```

```

3 //parcurea numerelor cu cel mult n cifre
4 ///"meet-in-the-middle", sume, sortare, reprezentare in baza 2, ...
5 #include<iostream>
6 #include<string.h>
7 #include<algorithm>
8
9 using namespace std;
10
11 ifstream fin ("artifact.in");
12 ofstream fout("artifact.out");
13
14 long long N,M,i,i1,j,p[92],f[92],a[12],c[92],d[92],cif[20],
15     j1,j2,n,k,l,x,q,nrsol,ok,ok1,s,r,y,n1,n2,p1,nc1,nc2;
16
17 struct triplet
18 {
19     long long s,v,c;
20 } c1[31002],c2[31002];
21
22 bool cmp(triplet a, triplet b)
23 {
24     if(a.s<b.s) return 1;
25     if(a.s==b.s && a.v<b.v) return 1;
26     return 0;
27 }
28
29 //a[] contine codurile ASCII ale literelor implicate
30 //p[j]=1 daca litera j este pe prima pozitie
31 //p[j]=0 in caz contrar
32 //n=numarul literelor implicate
33 //v[i] este valoarea temporar asociata cu litera a[i]
34 //c[j] este coeficientul din ecuatia asociat literei j
35
36 char t[12];
37 int ff(long long n1, long long r1, long long r2, long long &nc1, triplet c1[])
38 {
39     long long i,j,x,ok,s,y,p1;
40     p1=1;
41     nc1=0;
42     for(i=1;i<=n1;i++) p1=p1*10;
43     for(i=0;i<p1;i++)
44     {
45         for(j=0;j<=9;j++)
46             cif[j]=0;
47
48         x=i;
49         ok=1;
50         s=0;
51         y=0;
52         for(j=r1;j<=r2;j++)
53         {
54             cif[x%10]++;
55             if(cif[x%10]>1)
56             {
57                 ok=0;
58                 break;
59             }
60
61             if(x%10==0 && p[a[j]]==1)
62             {
63                 ok=0;
64                 break;
65             }
66
67             y=y+(1<<(x%10));
68             s=s+(x%10)*c[a[j]];
69             x=x/10;
70         }
71
72         if(ok==1)
73         {
74             nc1++;
75             c1[nc1].s=s;
76             c1[nc1].v=y;
77             c1[nc1].c=1;
78         }

```

```

79      }
80
81      sort(c1+1,c1+1+ncl,cmp);
82
83      j=1;
84      for(i=2;i<=ncl;i++)
85      {
86          if(c1[j].s==c1[i].s && c1[j].v==c1[i].v)
87              c1[j].c++;
88          else
89          {
90              j++;
91              c1[j]=c1[i];
92          }
93      }
94
95      nc1=j;
96      return 0;
97  }
98
99  int main()
100 {
101     fin>>N>>M;
102     for(i=1;i<=N;i++)
103     {
104         fin>>t;
105         l=strlen(t);
106         if(l>1) p[t[0]]=1;
107         q=1;
108         for(j=l-1;j>=0;j--)
109         {
110             f[t[j]]++;
111             c[t[j]]+=q;
112             q*=10;
113         }
114     }
115
116     for(i=1;i<=M;i++)
117     {
118         fin>>t;
119         l=strlen(t);
120         if(l>1) p[t[0]]=1;
121         q=1;
122         for(j=l-1;j>=0;j--)
123         {
124             f[t[j]]++;
125             c[t[j]]-=q;
126             q*=10;
127         }
128     }
129
130     n=0;
131     for(i=65;i<=90;i++)
132     {
133         if(f[i]>0)
134         {
135             n++;
136             a[n]=i;
137         }
138     }
139
140     ff(n/2,1,n/2,nc1,c1);
141     ff(n-n/2,n/2+1,n,nc2,c2);
142
143     nrsol=0;
144     for(i=1,j=nc2;i<=nc1;i++)
145     {
146         while(j>=1 && c2[j].s+c1[i].s>0) {j--;}
147         j1=j;
148         while(j1>=1 && c2[j1].s+c1[i].s==0) {j1--;}
149         i1=i;
150         while(i1<=nc1 && c1[i1].s==c1[i].s) {i1++;}
151         for(l=j1+1;l<=j;l++)
152         {
153             for(k=i;k<=i1-1;k++)
154                 if((c2[l].v & c1[k].v) == 0)

```

```

155             nrsol=nrsol+c2[1].c*c1[k].c;
156
157             i=i1-1;
158             j=j1;
159         }
160
161         fout<<nrsol<<"\n";
162         fout.close();
163         fin.close();
164         return 0;
165     }

```

Listing 19.1.4: razvan-artifact2-sum.cpp

```

1 // Problema artifact O( 11! * simboluri distințe )
2 // Razvan Turturica
3
4 #include <iostream>
5 #include <vector>
6 #include <cassert>
7 #include <algorithm>
8
9 using namespace std;
10
11 long long int coef[ 1000 ], suma,ans;
12 int fv[ 1000 ], prima[ 1000 ];
13 int cifra[ 1000 ];
14 long long int valoare[ 1000 ];
15 const long long modulo = 1000000007;
16 int stiva[ 100 ];
17 string a;
18 int n,m;
19 string sir[ 5010 ];
20
21 void bk( int nivel )
22 {
23     long long sum = 0, p;
24     if( nivel == a.size() )
25     {
26         for( int i = 1 ; i <= n + m ; i++ )
27         {
28             p = 1;
29             for( int j = sir[ i ].size() - 1 ; j >= 0 ; j-- )
30             {
31                 if( i <= n )
32                     sum += p * valoare[ sir[ i ][ j ] ];
33                 else
34                     sum -= p * valoare[ sir[ i ][ j ] ];
35                 p *= 10;
36             }
37         }
38         if( sum == 0 )
39             ans++;
40     }
41     return;
42 }
43
44 for( int i = 0 ; i <= 9 ; i++ )
45 {
46     if( fv[ i ] == 0 )
47     {
48         if( i == 0 && prima[ a[ nivel ] ] == 1 ) continue;
49         fv[ i ] = 1;
50         sum = suma;
51         valoare[ a[ nivel ] ] = i;
52         stiva[ nivel ] = i;
53         suma = ( suma + i * coef[ a[ nivel ] ] );
54         bk( nivel + 1 );
55         suma = sum;
56         fv[ i ] = 0;
57     }
58 }
59 }
60
61 int main()

```

```

62 {
63     ifstream cin("artifact.in");
64     ofstream cout("artifact.out");
65
66     cin >> n >> m;
67
68     for( int i = 1 ; i <= n + m ; i++ )
69     {
70         cin >> a;
71         if( a.size() > 1 )
72             prima[ a[ 0 ] ] = 1;
73         long long p = 1;
74         for( int j = a.size() - 1 ; j >= 0 ; j-- )
75         {
76             fv[ a[ j ] ] = 1;
77             sir[ i ] = a;
78             if( i <= n )
79             {
80                 coef[ a[ j ] ] = ( coef[ a[ j ] ] + p );
81             }
82             else
83             {
84                 coef[ a[ j ] ] = ( coef[ a[ j ] ] - p );
85             }
86             p = ( p * 10 );
87         }
88     }
89     a = "";
90     for( int i = 'A' ; i <= 'Z' ; i++ )
91         if( fv[ i ] == 1 )
92             a.push_back( i );
93
94     bk( 0 );
95     cout << ans;
96     return 0;
97 }
```

19.2 Scara

Problema 2 - Scara

100 de puncte

Avem N persoane notate cu etichetele $1, 2, \dots, N$, într-o ordine oarecare și o scară cu N trepte.

Persoanele sunt așezate în sir indian, cu față spre locul unde se află scara. Treptele scării sunt inițial neocupate.

În mod repetat persoana aflată la acel moment la capătul din dreapta al șirului se poziționează pe scară pe prima treaptă neocupată, iar fiecare dintre persoanele aflate pe treptele inferioare coboară de pe scară și se reposiționează la capătul din dreapta al șirului, începând cu cea de la prima treaptă a scării. Acțiunea se oprește atunci când sunt ocupate toate treptele pe scară.

Exemplu: Inițial la etapa 0 avem $\text{sir}=(4,2,1,3)$ și $\text{scara}=(0,0,0,0)$

Etapa 1: $(4,2,1)/(3,0,0,0)$ Etapa 2: $(4,2,3)/(0,1,0,0)$ Etapa 3: $(4,2)/3,1,0,0)$

Etapa 4: $(4,3,1)/(0,0,2,0)$ Etapa 5: $(4,3)/1,0,2,0)$ Etapa 6: $(4,1)/(0,3,2,0)$

Etapa 7: $(4)/(1,3,2,0)$ Etapa 8: $(1,3,2)/0,0,0,4)$...

Final: $()/(3,2,1,4)$

Cerințe

Se cere să se afișeze etichetele persoanelor în ordinea în care acestea sunt așezate pe scară la final.

Date de intrare

În fișierul de intrare **scara.in** se află scris pe prima linie numărul N . Pe a doua linie se află scrise N numere naturale separate prin câte un spațiu, reprezentând etichetele celor N persoane, în ordinea inițială din sir, de la stânga spre dreapta.

Date de ieșire

În fișierul de ieșire **scara.out** vor fi scrise pe prima linie și separate prin câte un spațiu cele N numere naturale reprezentând etichetele persoanelor corespunzătoare așezării pe scară.

Restricții și precizări

- $1 \leq N \leq 2000$
- Pentru 22 puncte $N \leq 15$

Exemple

scara.in	scara.out	Explicații
4 4 2 1 3	3 2 1 4	$(4,2,1,3)/(0,0,0,0)$, $(4,2,1)/(3,0,0,0)$, $(4,2,3)/(0,1,0,0)$, $(4,2)/(3,1,0,0)$, $(4,3,1)/(0,0,2,0)$, $(4,3)/(1,0,2,0)$, $(4,1)/(0,3,2,0)$, $(4)/(1,3,2,0)$, $(1,3,2)/(0,0,0,4)$, $(1,3)/(2,0,0,4)$, $(1,2)/(0,3,0,4)$, $(1)/(2,3,0,4)$, $(2,3)/(0,0,1,4)$, $(2)/(3,0,1,4)$, $(3)/(0,2,1,4)$, $()/(3,2,1,4)$
3 2 3 1	3 1 2	$(2,3,1)/(0,0,0)$, $(2,3)/(1,0,0)$, $(2,1)/(0,3,0)$, $(2)/(1,3,0)$, $(1,3)/(0,0,2)$, $(1)/(3,0,2)$, $(3)/(0,1,2)$, $()/(3,1,2)$

Timp maxim de execuțare/test: Windows: **0.5** secunde, Linux: **0.2** secunde

Memorie: total **128 MB**

Dimensiune maximă a sursei: **20 KB**

19.2.1 Indicații de rezolvare

prof. Piț-Rada Ionel-Vasile, Colegiul Național Traian, Drobeta Turnu Severin

Soluția 1 - complexitate $O(2^N)$

Se simulează activitățile de mutare a persoanelor între sir și scară, dar cu această soluție nu se va obține punctajul maxim.

Soluția 2 - complexitate $O(N^2)$

Se observă că soluția de așezare a ultimelor k persoane din sir pe primele k trepte de pe scară se poate reutiliza după așezarea persoanei $k+1$ pe treapta $k+1$, pentru a așeza cele k persoane care au fost forțate să părăsească primele k poziții de pe scară și s-au așezat în sir, prin definiția $x[j] =$ poziția din sir a persoanei care va fi așezată pe treapta j , pentru $1 \leq j \leq k$. Cu acest tip de soluție se poate obține punctaj maxim.

19.2.2 *Rezolvare detaliată

19.2.3 Cod sursă

Listing 19.2.1: scara_MD1.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 ifstream in("scara.in");
6 ofstream out("scara.out");
7
8 int n,x[10001],ff[10001],f[10001];
9
10 int main()
11 {
12     in>>n;
13
14     for(int i=1;i<=n;i++)
15         in>>x[i];
16
17     for(int i=1;i<=n;i++)
18     {
19         for(int j=1;j<i;j++)
20             ff[j]=f[j];
21
22         for(int j=1;j<i;j++)
23             f[j]=ff[ff[j]]+1;
24

```

```

25         f[i]=1;
26     }
27
28     out<<x[f[1]];
29     for(int i=2;i<=n;i++)
30         out<<' '<<x[f[i]];
31
32     out<<' \n';
33     return 0;
34 }
```

Listing 19.2.2: scara1.cpp

```

1 //O(N*N)
2 #include<iostream>
3 #include<string.h>
4
5 using namespace std;
6
7 ifstream fin ("scara.in");
8 ofstream fout("scara.out");
9
10 int N, i, k;
11
12 int main()
13 {
14     fin>>N;
15     int v[N+1], a[N+1], b[N+1];
16     for(i=1;i<=N;i++)
17     {
18         fin>>v[N+1-i];
19         b[i]=i; a[i]=i;
20     }
21     for(k=2;k<=N;k++)
22     {
23         for(int i=1;i<=k-1;i++)
24             a[i]=b[k-b[i]];
25         memcpy(b+1,a+1,k*sizeof(int));
26     }
27     for(i=1;i<=N;i++)
28         fout<<v[a[i]]<<" ";
29
30     fout<<"\n";
31     fout.close();
32     fin.close();
33     return 0;
34 }
```

Listing 19.2.3: scara2.cpp

```

1 //complexitate O(N*(2^N))
2 #include<iostream>
3
4 using namespace std;
5
6 ifstream fin ("scara.in");
7 ofstream fout("scara.out");
8
9 int N, i, a[1005], b[1005];
10
11 void sol2()
12 {
13     int i,nb,pr,x;
14     nb=0;
15     pr=N;
16     while(nb<N)
17     {
18         x=a[pr];
19         a[pr]=0;
20         pr--;
21         i=1;
22         while(b[i]!=0)
23         {
24             pr++;
```

```

25         a[pr]=b[i];
26         b[i]=0;
27         i++;
28         nb--;
29     }
30
31     b[i]=x;
32     nb++;
33 }
34 }
35
36 int main()
37 {
38     fin>>N;
39     for(i=1;i<=N;i++)
40         fin>>a[i];
41
42     sol2();
43
44     for(i=1;i<=N;i++)
45         fout<<b[i]<<" ";
46
47     fout<<"\n";
48     fout.close();
49     fin.close();
50     return 0;
51 }
```

19.3 walle

Problema 3 - walle

100 de puncte

Roboțelul WALL-E este captiv într-un labirint dreptunghiular de dimensiuni $N \times M$. Analizând harta, WALL-E constată că are de-a face cu un labirint extrem de sofisticat. El reușește să identifice următoarele tipuri de celule:

- 'W' - celula unde, la început, se află WALL-E,
- 'E' - celula 'EXIT' care poate fi accesată de WALL-E și care îl poate teleporta pe acesta instantaneu în afara labirintului, într-un loc sigur,
- '.' - celule libere, care pot fi accesate de WALL-E,
- '#' - celule de tip zid, care NU pot fi accesate de WALL-E,
- '+' - celule de tip ușă, care pot fi accesate de WALL-E, dar continuarea deplasării la o celulă vecină se poate face doar după o așteptare de exact T secunde,
- 'P' - celule de tip portal, care îl teleportează pe WALL-E instantaneu, la întâmplare, într-o dintre celelalte celule de tip portal. Dacă WALL-E accesează o celulă (x_1, y_1) de tip portal, atunci el va fi instantaneu teleportat la o altă celulă (x_2, y_2) de tip portal, iar mai departe el se va deplasa numai într-o celulă vecină cu (x_2, y_2) (nu poate sta pe loc).

WALL-E se poate deplasa într-o secundă în oricare dintre cele patru celule vecine: sus, dreapta, jos sau stânga, pe care le poate accesa. De asemenea, roboțelul NU se poate deplasa în afara labirintului decât prin intermediul celulei 'EXIT'.

Cerințe

Comportamentul haotic al portalurilor îl îngrijorează pe WALL-E, astfel că își propune să afle care este numărul minim de secunde în care, **cu certitudine**, el va putea părăsi labirintul. Dacă nu se poate determina cu certitudine acest lucru, sau dacă WALL-E nu poate părăsi labirintul, răspunsul va fi -1 .

Date de intrare

În fișierul text **walle.in** pe prima linie se află numerele N , M și T , separate prin câte un spațiu. Pe fiecare dintre următoarele N linii se află câte un sir cu câte M caractere.

Date de ieșire

În fișierul text **walle.out** se va scrie, pe prima linie, un număr natural reprezentând răspunsul găsit sau -1 .

Restricții și precizări

- $1 \leq N, M \leq 500$
- $0 \leq T \leq 1000$
- Există o singură celulă marcată cu 'W'
- Există o singură celulă marcată cu 'E'
- Numărul celulelor de tip portal este mai mare sau egal cu 2
- Se garantează că fiecare celulă de tip portal are cel puțin o celulă vecină care poate fi accesată
- Pentru teste în valoare de 19 puncte labirintul va conține exact 2 portaluri și $T = 0$.
- Pentru alte 16 puncte labirintul va conține exact 2 portaluri.
- Pentru 19 puncte $1 \leq N, M \leq 50$, labirintul va conține cel mult 6 portaluri și $T = 0$.
- Pentru alte 27 puncte $1 \leq N, M \leq 50$, labirintul va conține cel mult 6 portaluri.
- Pentru alte 10 puncte $T = 0$.

Exemple

walle.in	walle.out	Explicații
5 12 3#P..E.. .P...##....+...P.. .W....#....	5	WALL-E se va deplasa în 2 secunde la portalul de la poziția (2,3), care îl va telepoata în cel mai defavorabil caz la poziția (1, 7), de unde în 3 secunde va ajunge la EXIT. Total 5 secunde.
5 12 3P#.E.. .P...##....+...P.. .W....#....	12	WALL-E va merge către EXIT, evitând portalurile și ușa, pe drumul cel mai scurt.
1 6 3 EPPPWP	-1	Să presupunem că WALL-E se va deplasa mai întâi la portalul (1,4). Este incert ca WALL-E să poată ajunge la portalul (1,2), deoarece de la portalul(1,4), datorită comportamentului haotic al portalurilor, se poate ajunge la oricare din celelalte trei portaluri, nu doar la portalul (1,2), iar apoi teleportarea următoare îl va putea reîntoarce pe WALL-E, din același motiv, la portalul (1,4). Analog se va întâmpla dacă WALL-E se va deplasa mai întâi la portalul (1,6).

Timp maxim de executare/test: Windows: **0.4 secunde**, Linux: **0.2** secunde

Memorie: total **128 MB**

Dimensiune maximă a sursei: **20 KB**

19.3.1 Indicații de rezolvare

prof. Pit-Rada Ionel-Vasile, Colegiul Național Traian, Drobeta Turnu Severin

Soluție - complexitate $O(N * M)$

O observație importantă este că Wall-E nu poate folosi portalurile de foarte multe ori.

Dacă el intră în portalul A , este teleportat la portalul B , el nu va putea folosi un nou portal C , diferit de B , întrucât acest portal l-ar putea teleporta din nou la portalul B , situație în care s-a mai aflat anterior. În consecință el va irosi timp, strategia lui nu va fi optimă.

Deasemenea, există și posibilitatea ca Wall-E să refolosească portalul B , însă evident o singură dată, dacă folosim un rationament similar cu cel anterior.

Astfel distingem trei strategii esențiale pe care Wall-E le poate folosi:

(1) Nu va folosi portaluri.

(2) Va călători pe traseul cel mai scurt către un portal convenabil A , care îl va telepoata la un portal B , după care își va continua drumul pe traseul cel mai scurt către EXIT.

(3) Va călători pe traseul cel mai scurt către un portal convenabil A , care îl va telepoata la un portal B , apoi el va păsi în afara celulei portalului B , va reintra imediat în portal (cât se poate de repede), după care va fi teleportat la un portal C , posibil identic cu A , de unde își va

continua drumul către EXIT pe traseul cel mai scurt. De observat că în situația în care Wall-E e teleportat la portalul A , el se va afla într-o situație nouă, întrucât prima dată portalul A l-a teleportat instantaneu, neputând continua în celulele vecine.

Răspunsul final va fi minimul dintre rezultatele optime folosind fiecare dintre cele trei strategii.

Este util de calculat:

$w[i][j]$ = timpul minim în care Wall-E poate ajunge din poziția sa inițială la celula (i, j) , fără a folosi portaluri

$e[i][j]$ = timpul minim în care Wall-E poate ajunge de la celula (i, j) la EXIT, fără a folosi portaluri

Evident $e[i][j]$ se poate calcula similar cu $w[i][j]$, dacă se calculează drumul invers, de la EXIT către (i, j) .

Singura dificultate în calculul $w[.][.]$ și $e[.][.]$ o reprezintă ușile. Se pot folosi două cozi Q_1 , Q_2 , ce vor conține celulele libere (Q_1), respectiv celulele de tip ușă ce induc o întârziere de T secunde (Q_2). Folosind aceste două cozi, putem implementa o strategie similară cu *parcugerea clasică în lățime*, vizitând celulele în ordinea crescătoarea a timpilor de descoperire. Analizând doar elementele din vârful lor, putem determina care va fi celula ce va putea fi părăsită cel mai repede, astfel putând vizita vecinii neaținși încă, ce vor fi plasati corespunzator în Q_1 sau Q_2 .

Să revizităm cele trei strategii menționate anterior:

(1) Dacă celula EXIT are coordonatele (ex, ey) , $w[ex][ey]$ va furniza rezultatul optim al acestei strategii.

(2) Rezultatul este de forma $w[ax][ay] + \max\{e[bx][by] / (ax, ay) != (bx, by)\}$, unde pozițiile (ax, ay) , respectiv (bx, by) reprezintă portaluri. Pentru a evita o soluție pătratică care verifică fiecare pereche (A, B) de portaluri, se pot calcula t_1 și t_2 , cele mai mari două valori din $e[.][.]$, unde se află portaluri. Expresia de mai sus devine fie $w[ax][ay] + t_1$, fie $w[ax][ay] + t_2$, după cum $e[ax][ay]$ este sau nu egal cu t_1 .

(3) Acest caz poate fi redus la un caz similar cu cazul (2), dacă calculăm:

$e'[i][j]$ = timpul minim de la portalul situat la poziția (i, j) , cu reintrare imediată în el, urmat de o teleportare imediată la un alt portal, de unde se continuă pe traseul cel mai rapid către EXIT

Folosind din nou t_1 și t_2 se poate calcula $e'[.][.]$ similar cu (2), reducând problema la o situație care este din nou similară cu (2), unde se vor folosi $w[.][.]$ și $e'[.][.]$ în loc de $w[.][.]$ și $e[.][.]$.

19.3.2 *Rezolvare detaliată

19.3.3 Cod sursă

Listing 19.3.1: adrian-100.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <cassert>
5 #include <queue>
6 #include <algorithm>
7 #include <tuple>
8 #include <limits>
9
10 using namespace std;
11
12 static const int dx[4] = {-1, 0, 1, 0};
13 static const int dy[4] = {0, -1, 0, 1};
14 static const int kInfinite = numeric_limits<int>::max() / 2;
15
16 vector<vector<int>> get_distances(const vector<string> &board,
17                                     pair<int, int> start, int plus_time)
18 {
19     queue<pair<int, int>> normal_queue, plus_queue;
20     vector<vector<int>> distances(board.size(),
21                                         vector<int>(board[0].size(), kInfinite));
21

```

```

22
23     normal_queue.push(start);
24     distances[start.first][start.second] = 0;
25
26     while (!normal_queue.empty() || !plus_queue.empty())
27     {
28         int x, y;
29         if (plus_queue.empty())
30         {
31             tie(x, y) = normal_queue.front();
32             normal_queue.pop();
33         }
34         else
35             if (normal_queue.empty())
36             {
37                 tie(x, y) = plus_queue.front();
38                 plus_queue.pop();
39             }
40         else
41         {
42             int x1, y1, x2, y2;
43             tie(x1, y1) = normal_queue.front();
44             tie(x2, y2) = plus_queue.front();
45             if (distances[x1][y1] <= distances[x2][y2])
46             {
47                 x = x1;
48                 y = y1;
49                 normal_queue.pop();
50             }
51             else
52             {
53                 x = x2;
54                 y = y2;
55                 plus_queue.pop();
56             }
57         }
58
59         for (int i = 0; i < 4; ++i)
60         {
61             int newx = x + dx[i];
62             int newy = y + dy[i];
63             if (newx < 0 || newy < 0 || newx >= int(board.size()) ||
64                 newy >= int(board[0].size()))
65                 continue;
66             if (distances[newx][newy] == kInfinite)
67             {
68                 distances[newx][newy] = distances[x][y] + 1;
69                 if (board[newx][newy] == '+')
70                 {
71                     distances[newx][newy] += plus_time;
72                     plus_queue.emplace(newx, newy);
73                 }
74                 else
75                     if (board[newx][newy] == '.')
76                         normal_queue.emplace(newx, newy);
77                 }
78             }
79         }
80         return distances;
81     }
82
83     int main()
84     {
85         ifstream cin("walle.in");
86         ofstream cout("walle.out");
87
88         int N, M, T;
89         assert(cin >> N >> M >> T);
90         assert(1 <= N && N <= 500);
91         assert(1 <= M && M <= 500);
92         assert(0 <= T && T <= 1000);
93         vector<string> board(N);
94         for (int i = 0; i < N; ++i)
95         {
96             assert(cin >> board[i]);
97             for (auto &c : board[i])

```

```

98         assert(c == '.' || c == '+' || c == 'E' || 
99                 c == 'P' || c == 'W' || c == '#');
100    }
101
102    pair<int, int> start(-1, -1), end(-1, -1);
103    int portals = 0;
104
105    for (int i = 0; i < N; ++i)
106        for (int j = 0; j < M; ++j)
107            if (board[i][j] == 'W')
108            {
109                assert(start == make_pair(-1, -1));
110                start = make_pair(i, j);
111            }
112            else
113                if (board[i][j] == 'E')
114                {
115                    assert(end == make_pair(-1, -1));
116                    end = make_pair(i, j);
117                }
118            else
119                if (board[i][j] == 'P')
120
121                ++portals;
122                assert(start != make_pair(-1, -1));
123                assert(end != make_pair(-1, -1));
124                assert(portals > 1);
125
126                auto start_distances = get_distances(board, start, T);
127                auto end_distances = get_distances(board, end, T);
128
129                int answer = kInfinite;
130                // first lets test directly
131                answer = min(answer, start_distances[end.first][end.second]);
132
133                // then through a portal, best to know max distance
134                // from a portal to the end
135                // and second max
136                // and closest to start (we'll need that later)
137                int max_distance = 0;
138                int second_max_distance = 0;
139                int closest = kInfinite;
140                for (int i = 0; i < N; ++i)
141                    for (int j = 0; j < M; ++j)
142                        if (board[i][j] == 'P')
143                        {
144                            if (end_distances[i][j] > max_distance)
145                            {
146                                second_max_distance = max_distance;
147                                max_distance = end_distances[i][j];
148                            }
149                            else
150                                if (end_distances[i][j] > second_max_distance)
151                                {
152                                    second_max_distance = end_distances[i][j];
153                                }
154
155                            closest = min(closest, start_distances[i][j]);
156                        }
157
158                // going strictly through a portal and then to the end
159                vector<vector<int>> then_to_end(N, vector<int>(M, 0));
160                for (int i = 0; i < N; ++i)
161                    for (int j = 0; j < M; ++j)
162                        if (board[i][j] == 'P')
163                        {
164                            // let's say we hop into this portal
165                            if (end_distances[i][j] == max_distance)
166                                then_to_end[i][j] = second_max_distance;
167                            else
168                                then_to_end[i][j] = max_distance;
169                            answer = min(answer, start_distances[i][j]+then_to_end[i][j]);
170                        }
171
172                // now here's the thing, we might take 2 hops
173                // (we know 3 ore more are useless)

```

```

174     // and it's a MAYBEEEE. Why?
175     // let's say our strategy is to always take 2 hops.
176     // So going from A -> B -> Hop in B
177     //           or A -> C -> Hop in C,
178     // then we make the hop in B go to C and viceversa,
179     // and we destroyed this strategy
180     // so we take the 2nd hop only from a fixed "bad" portal
181     // let's fix that
182     // we can just assume worst case is to go through closest portal
183     // then pop out of this one, go nearby (in an empty space), go back in
184     // and then however much it takes to the end
185     for (int i = 0; i < N; ++i)
186         for (int j = 0; j < M; ++j)
187             if (board[i][j] == 'P')
188             {
189                 if (closest == kInfinite || then_to_end[i][j] == kInfinite)
190                     continue;
191                 int cost_empty = kInfinite;
192                 for (int k = 0; k < 4; ++k)
193                 {
194                     int x = i + dx[k];
195                     int y = j + dy[k];
196                     if (x < 0 || y < 0 || x >= N || y >= M)
197                         continue;
198                     if (board[x][y] == '.')
199                         cost_empty = min(cost_empty, 2);
200                     else
201                         if (board[x][y] == '+')
202                             cost_empty = min(cost_empty, 2 + T);
203                 }
204                 // well we got a shot here
205                 answer = min(answer, closest+cost_empty+then_to_end[i][j]);
206             }
207
208             if (answer == kInfinite)
209                 answer = -1;
210             cout << answer << "\n";
211         }

```

Listing 19.3.2: adrian-direct.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <cassert>
5 #include <queue>
6 #include <algorithm>
7 #include <tuple>
8 #include <limits>
9
10 using namespace std;
11
12 static const int dx[4] = {-1, 0, 1, 0};
13 static const int dy[4] = {0, -1, 0, 1};
14 static const int kInfinite = numeric_limits<int>::max() / 2;
15
16 vector< vector<int> > get_distances(const vector<string> &board,
17                                         pair<int, int> start, int plus_time)
18 {
19     queue< pair<int, int> > normal_queue, plus_queue;
20     vector< vector<int> > distances(board.size(),
21                                         vector<int>(board[0].size(), kInfinite));
22
23     normal_queue.push(start);
24     distances[start.first][start.second] = 0;
25
26     while (!normal_queue.empty() || !plus_queue.empty())
27     {
28         int x, y;
29         if (plus_queue.empty())
30         {
31             tie(x, y) = normal_queue.front();
32             normal_queue.pop();
33         }
34         else

```

```

35         if (normal_queue.empty())
36     {
37         tie(x, y) = plus_queue.front();
38         plus_queue.pop();
39     }
40     else
41     {
42         int x1, y1, x2, y2;
43         tie(x1, y1) = normal_queue.front();
44         tie(x2, y2) = plus_queue.front();
45         if (distances[x1][y1] <= distances[x2][y2])
46         {
47             x = x1;
48             y = y1;
49             normal_queue.pop();
50         }
51         else
52         {
53             x = x2;
54             y = y2;
55             plus_queue.pop();
56         }
57     }
58
59     for (int i = 0; i < 4; ++i)
60     {
61         int newx = x + dx[i];
62         int newy = y + dy[i];
63         if (newx < 0 || newy < 0 || newx >= int(board.size()) ||
64             newy >= int(board[0].size()))
65             continue;
66         if (distances[newx][newy] == kInfinite)
67         {
68             distances[newx][newy] = distances[x][y] + 1;
69             if (board[newx][newy] == '+')
70             {
71                 distances[newx][newy] += plus_time;
72                 plus_queue.emplace(newx, newy);
73             }
74             else
75                 if (board[newx][newy] == '.')
76                     normal_queue.emplace(newx, newy);
77         }
78     }
79 }
80
81     return distances;
82 }
83
84 int main()
85 {
86     ifstream cin("walle.in");
87     ofstream cout("walle.out");
88
89     int N, M, T;
90     assert(cin >> N >> M >> T);
91     assert(1 <= N && N <= 500);
92     assert(1 <= M && M <= 500);
93     assert(0 <= T && T <= 1000);
94     vector<string> board(N);
95
96     for (int i = 0; i < N; ++i)
97     {
98         assert(cin >> board[i]);
99         for (auto &c : board[i])
100             assert(c == '.' || c == '+' || c == 'E' ||
101                   c == 'P' || c == 'W' || c == '#');
102     }
103
104     pair<int, int> start(-1, -1), end(-1, -1);
105     int portals = 0;
106     for (int i = 0; i < N; ++i)
107         for (int j = 0; j < M; ++j)
108             if (board[i][j] == 'W')
109             {
110                 assert(start == make_pair(-1, -1));
111                 start = make_pair(i, j);

```

```

111         }
112     else if (board[i][j] == 'E')
113     {
114         assert(end == make_pair(-1, -1));
115         end = make_pair(i, j);
116     }
117     else
118     {
119         if (board[i][j] == 'P')
120             ++portals;
121
122         assert(start != make_pair(-1, -1));
123         assert(end != make_pair(-1, -1));
124         assert(portals > 1);
125
126         auto start_distances = get_distances(board, start, T);
127
128         int answer = kInfinite;
129         // first lets test directly
130         answer = min(answer, start_distances[end.first][end.second]);
131
132         if (answer == kInfinite)
133             answer = -1;
134         cout << answer << "\n";
135     }

```

Listing 19.3.3: adrian-mask.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <algorithm>
5 #include <vector>
6 #include <queue>
7 #include <tuple>
8 #include <limits>
9
10 using namespace std;
11
12 static const int dx[4] = {-1, 0, 1, 0};
13 static const int dy[4] = {0, -1, 0, 1};
14 static const int kInfinite = numeric_limits<int>::max() / 2;
15
16 int main()
17 {
18     ifstream cin("walle.in");
19     ofstream cout("walle.out");
20
21     int N, M, T; cin >> N >> M >> T;
22
23     vector<string> board(N);
24     for (int i = 0; i < N; ++i)
25         cin >> board[i];
26
27     pair<int, int> start, end;
28
29     vector< pair<int, int>> portals;
30     vector< vector<int> > portal_index(N, vector<int>(M, -1));
31     for (int i = 0; i < N; ++i)
32         for (int j = 0; j < M; ++j)
33             if (board[i][j] == 'W')
34                 start = make_pair(i, j);
35             else
36                 if (board[i][j] == 'E')
37                     end = make_pair(i, j);
38                 else
39                     if (board[i][j] == 'P')
40                     {
41                         portal_index[i][j] = portals.size();
42                         portals.emplace_back(i, j);
43                     }
44
45     queue< pair<int, int> > normal_queue, plus_queue, portal_queue;
46     int answer = numeric_limits<int>::max();
47     vector<int> portal_distance(portals.size(), kInfinite);
48

```

```

49     for (int mask = (1 << portals.size()) - 1; mask >= 0; --mask)
50     {
51         vector< vector<int> > distances(N, vector<int>(M, kInfinite));
52         // distances now
53         normal_queue.push(end);
54         distances[end.first][end.second] = 0;
55
56         // distances portals
57         vector< pair<int, int> > temp;
58         for (int i = 0; i < int(portals.size()); ++i)
59             if (!(1 << i) & mask)
60             {
61                 int x, y; tie(x, y) = portals[i];
62                 // can still take this portal
63                 int now = 0;
64                 for (int j = 0; j < int(portals.size()); ++j)
65                     if (i != j)
66                         now = max(now, portal_distance[j]);
67
68                 if (now == kInfinite)
69                     continue;
70                 distances[x][y] = now;
71                 temp.emplace_back(x, y);
72             }
73
74         sort(temp.begin(), temp.end(),
75              [&](pair<int, int> a, pair<int, int> b)
76             {
77                 return distances[a.first][a.second] <
78                     distances[b.first][b.second];
79             });
80
81         for (auto &p : temp)
82             portal_queue.push(p);
83
84         while (!normal_queue.empty() ||
85                !plus_queue.empty() ||
86                !portal_queue.empty())
87         {
88             int x = -1, y = -1;
89             if (!normal_queue.empty())
90                 tie(x, y) = normal_queue.front();
91
92             if (!plus_queue.empty())
93             {
94                 int x2, y2;
95                 tie(x2, y2) = plus_queue.front();
96                 if (x == -1 || distances[x][y] > distances[x2][y2])
97                 {
98                     x = x2;
99                     y = y2;
100                }
101            }
102
103            if (!portal_queue.empty())
104            {
105                int x2, y2;
106                tie(x2, y2) = portal_queue.front();
107                if (x == -1 || distances[x][y] > distances[x2][y2])
108                {
109                    x = x2;
110                    y = y2;
111                }
112            }
113
114            if (!normal_queue.empty() &&
115                make_pair(x, y) == normal_queue.front())
116                normal_queue.pop();
117            else
118                if (!plus_queue.empty() &&
119                    make_pair(x, y) == plus_queue.front())
120                    plus_queue.pop();
121                else
122                    portal_queue.pop();
123
124         for (int i = 0; i < 4; ++i)

```

```

125         {
126             int newx = x + dx[i];
127             int newy = y + dy[i];
128             if (newx < 0 || newy < 0 || newx >= N || newy >= M)
129                 continue;
130             if (distances[newx][newy] == kInfinite)
131             {
132                 distances[newx][newy] = distances[x][y] + 1;
133
134                 if (board[newx][newy] == '.')
135                     normal_queue.emplace(newx, newy);
136                 else
137                     if (board[newx][newy] == '+')
138                     {
139                         distances[newx][newy] += T;
140                         plus_queue.emplace(newx, newy);
141                     }
142             }
143             else
144                 if (board[newx][newy] == 'P')
145                 {
146                     // might be a portal from which we start
147                     int index = portal_index[newx][newy];
148                     portal_distance[index] = min(portal_distance[index],
149                                         distances[x][y] + 1);
150                 }
151             }
152         }
153     answer = min(answer, distances[start.first][start.second]);
154
155     for (int i = 0; i < int(portals.size()); ++i)
156         if ((1 << i) & mask)
157             portal_distance[i] = min(portal_distance[i],
158                                         distances[portals[i].first][portals[i].second]);
159     }
160
161     if (answer == kInfinite)
162         answer = -1;
163     cout << answer << "\n";
164 }
```

Listing 19.3.4: adrian-one-portal.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <cassert>
5 #include <queue>
6 #include <algorithm>
7 #include <tuple>
8 #include <limits>
9
10 using namespace std;
11
12 static const int dx[4] = {-1, 0, 1, 0};
13 static const int dy[4] = {0, -1, 0, 1};
14 static const int kInfinite = numeric_limits<int>::max() / 2;
15
16 vector<vector<int>> get_distances(const vector<string> &board,
17                                     pair<int, int> start, int plus_time)
18 {
19     queue<pair<int, int>> normal_queue, plus_queue;
20     vector<vector<int>> distances(board.size(),
21                                     vector<int>(board[0].size(), kInfinite));
22
23     normal_queue.push(start);
24     distances[start.first][start.second] = 0;
25
26     while (!normal_queue.empty() || !plus_queue.empty())
27     {
28         int x, y;
29         if (plus_queue.empty())
30         {
31             tie(x, y) = normal_queue.front();
32             normal_queue.pop();
```

```

33         }
34     else
35         if (normal_queue.empty())
36         {
37             tie(x, y) = plus_queue.front();
38             plus_queue.pop();
39         }
40     else
41     {
42         int x1, y1, x2, y2;
43         tie(x1, y1) = normal_queue.front();
44         tie(x2, y2) = plus_queue.front();
45         if (distances[x1][y1] <= distances[x2][y2])
46         {
47             x = x1;
48             y = y1;
49             normal_queue.pop();
50         }
51     else
52     {
53         x = x2;
54         y = y2;
55         plus_queue.pop();
56     }
57 }
58
59     for (int i = 0; i < 4; ++i)
60     {
61         int newx = x + dx[i];
62         int newy = y + dy[i];
63         if (newx < 0 || newy < 0 ||
64             newx >= int(board.size()) ||
65             newy >= int(board[0].size()))
66             continue;
67         if (distances[newx][newy] == kInfinite)
68         {
69             distances[newx][newy] = distances[x][y] + 1;
70             if (board[newx][newy] == '+')
71             {
72                 distances[newx][newy] += plus_time;
73                 plus_queue.emplace(newx, newy);
74             }
75         else
76             if (board[newx][newy] == '.')
77                 normal_queue.emplace(newx, newy);
78         }
79     }
80 }
81
82     return distances;
83 }
84
85 int main()
86 {
87     ifstream cin("walle.in");
88     ofstream cout("walle.out");
89
90     int N, M, T;
91     assert(cin >> N >> M >> T);
92     assert(1 <= N && N <= 500);
93     assert(1 <= M && M <= 500);
94     assert(0 <= T && T <= 1000);
95     vector<string> board(N);
96
97     for (int i = 0; i < N; ++i)
98     {
99         assert(cin >> board[i]);
100        for (auto &c : board[i])
101            assert(c == '.' || c == '+' || c == 'E' ||
102                  c == 'P' || c == 'W' || c == '#');
103    }
104
105    pair<int, int> start(-1, -1), end(-1, -1);
106
107    int portals = 0;
108    for (int i = 0; i < N; ++i)

```

```

109         for (int j = 0; j < M; ++j)
110             if (board[i][j] == 'W')
111             {
112                 assert(start == make_pair(-1, -1));
113                 start = make_pair(i, j);
114             }
115             else
116                 if (board[i][j] == 'E')
117                 {
118                     assert(end == make_pair(-1, -1));
119                     end = make_pair(i, j);
120                 }
121             else
122                 if (board[i][j] == 'P')
123                     ++portals;
124
125             assert(start != make_pair(-1, -1));
126             assert(end != make_pair(-1, -1));
127             assert(portals > 1);
128
129             auto start_distances = get_distances(board, start, T);
130             auto end_distances = get_distances(board, end, T);
131
132             int answer = kInfinite;
133             // first lets test directly
134             answer = min(answer, start_distances[end.first][end.second]);
135
136             // then through a portal, best to know max distance
137             // from a portal to the end
138             // and second max
139             // and closest to start (we'll need that later)
140             int max_distance = 0;
141             int second_max_distance = 0;
142             for (int i = 0; i < N; ++i)
143                 for (int j = 0; j < M; ++j)
144                     if (board[i][j] == 'P')
145                     {
146                         if (end_distances[i][j] > max_distance)
147                         {
148                             second_max_distance = max_distance;
149                             max_distance = end_distances[i][j];
150                         }
151                         else
152                             if (end_distances[i][j] > second_max_distance)
153                             {
154                                 second_max_distance = end_distances[i][j];
155                             }
156                     }
157
158             // going strictly through a portal and then to the end
159             vector<vector<int>> then_to_end(N, vector<int>(M, 0));
160             for (int i = 0; i < N; ++i)
161                 for (int j = 0; j < M; ++j)
162                     if (board[i][j] == 'P')
163                     {
164                         // let's say we hop into this portal
165                         if (end_distances[i][j] == max_distance)
166                             then_to_end[i][j] = second_max_distance;
167                         else
168                             then_to_end[i][j] = max_distance;
169
170                         answer = min(answer, start_distances[i][j] + then_to_end[i][j]);
171                     }
172
173             if (answer == kInfinite)
174                 answer = -1;
175             cout << answer << "\n";
176         }

```

Listing 19.3.5: walle01.cpp

```

1  ///O(N*M*T)
2  #include<iostream>
3  #define nmax 502
4  #define nmax2 250002

```

```

5
6  using namespace std;
7
8  ifstream fin ("walle.in");
9  ofstream fout("walle.out");
10
11 int N,M,T,le,ce,lw,cw,i,j,i1,j1,k,ul,pr,nr,r,P,y1,y,z,dmin,de1,de2,d,d1,d2;
12 int W[nmax][nmax],E[nmax][nmax],usi[nmax][nmax],infinit=1000000000;
13 int dl[4]={-1, 0,+1, 0};
14 int dc[4]={ 0,+1, 0,-1};
15 char A[nmax][nmax],ch;
16
17 struct pozitie
18 {
19     int lin,col;
20 } coada[nmax2];
21
22 void solve(int X[nmax][nmax],int lx, int cx)
23 {
24     ul=0;
25     pr=1;
26     nr=0;
27     r=nmax2-2;
28     X[lx][cx]=0;
29     ul++;
30     nr++;
31     coada[ul]={lx,cx};
32     while(nr)
33     {
34         i1=coada[pr].lin;
35         j1=coada[pr].col;
36         if(A[i1][j1]=='+' && usi[i1][j1]>0)
37         {
38             usi[i1][j1]--;
39             ul++;
40             if(ul>r)
41                 ul=1;
42             nr++;
43             coada[ul]={i1,j1};
44             pr++;
45             if(pr>r) pr=1;
46             nr--;
47             continue;
48         }
49
50         for(k=0;k<=3;k++)
51         {
52             i=i1+dl[k];
53             j=j1+dc[k];
54             if(i>=1 && i<=N && j>=1 && j<=M &&
55                 A[i][j]!='#' && X[i][j]==infinit)
56             {
57                 if(A[i1][j1]=='+')
58                     X[i][j]=1+T+X[i1][j1];
59                 else
60                     X[i][j]=1+X[i1][j1];
61
62                 if(A[i][j]!='P')
63                 {
64                     ul++;
65                     if(ul>r) ul=1;
66                     nr++;
67                     coada[ul]={i,j};
68                 }
69
70                 if(A[i][j]=='+')
71                     usi[i][j]=T;
72             }
73         }
74         pr++;
75         if(pr>r) pr=1;
76         nr--;
77     }
78 }
79 }
80

```

```

81 int main()
82 {
83     fin>>N>>M>>T;
84     for(i=1;i<=N;i++)
85     {
86         for(j=1;j<=M;j++)
87         {
88             fin>>ch;
89             A[i][j]=ch;
90             if(ch=='W') {lw=i; cw=j;}
91             if(ch=='E') {le=i; ce=j;}
92             if(ch=='P') P++;
93             W[i][j]=infinit;
94             E[i][j]=infinit;
95         }
96     }
97
98     //determinam W[i][j]=numarul minim de secunde in care se poate ajunge
99     // la [i][j] pornind de la [lw][cw], fara portaluri
100    solve(W,lw,cw);
101    solve(E,le,ce);
102    //determinam cele mai mari doua distante de la portaluri spre EXIT
103    de1=0;
104    de2=0;
105    y=0;
106    for(i=1;i<=N;i++)
107        for(j=1;j<=M;j++)
108            if(A[i][j]=='P' && E[i][j]<infinit)
109            {
110                y++;
111                if(E[i][j]>=de1)
112                {
113                    de2=de1;
114                    de1=E[i][j];
115                }
116                else
117                    if(E[i][j]>de2)
118                        de2=E[i][j];
119            }
120
121    dmin=W[le][ce];
122    // imbunatatim solutia minima folosind portaluri
123    // W trebuie sa ajunga la un portal si apoi de la orice alt portal
124    // trebuie sa se poata ajunge la E
125    for(i=1;i<=N;i++)
126        for(j=1;j<=M;j++)
127            if(A[i][j]=='P' && W[i][j]<infinit)
128            {
129                y1=y;
130                d=E[i][j];
131                d2=W[i][j];
132                if(d<infinit) y1--;
133                if(y1+1==P)
134                {
135                    if(d==de1)
136                        d1=de2;
137                    else
138                        d1=de1;
139                    if(dmin>d1+d2)
140                        dmin=d1+d2;
141                }
142            }
143
144    if(dmin==infinit)
145        dmin=-1;
146
147    fout<<dmin;
148    fout.close();
149    fin.close();
150    return 0;
151 }

```

Listing 19.3.6: walle1.cpp

```

2 #include<fstream>
3
4 #define nmax 502
5 #define nmax2 250002
6
7 using namespace std;
8
9 ifstream fin ("walle.in");
10 ofstream fout("walle.out");
11
12 int N,M,T,le,ce,lw,cw,i,j,i1,i2,j1,j2,k,ul,pr,nr,r,P,y1,y,z,
13     dmin,de1,de2,d,d1,d2,c1,dc1,dc2;
14 int W[nmax][nmax],E[nmax][nmax],usi[nmax][nmax],infinit=1000000000;
15 int dl[4]={-1, 0,+1, 0};
16 int dc[4]={ 0,+1, 0,-1};
17 char A[nmax][nmax],ch;
18
19 struct pozitie
20 {
21     int lin,col;
22 } coada[nmax2];
23
24 void solve(int X[nmax][nmax],int lx, int cx)
25 {
26     ul=0;
27     pr=1;
28     nr=0;
29     r=nmax2-2;
30     X[lx][cx]=0;
31     ul++;
32     nr++;
33     coada[ul]={lx,cx};
34
35     while(nr)
36     {
37         i1=coada[pr].lin;
38         j1=coada[pr].col;
39         if(A[i1][j1]=='+' && usi[i1][j1]>0)
40         {
41             usi[i1][j1]--;
42             ul++;
43             if(ul>r) ul=1;
44             nr++;
45             coada[ul]={i1,j1};
46             pr++;
47             if(pr>r) pr=1;
48             nr--;
49             continue;
50         }
51
52         for(k=0;k<=3;k++)
53         {
54             i=i1+dl[k];
55             j=j1+dc[k];
56             if(i>=1 && i<=N && j>=1 && j<=M &&
57                 A[i][j]!='#' && X[i][j]==infinit)
58             {
59                 if(A[i1][j1]=='+')
60                     X[i][j]=1+T+X[i1][j1];
61                 else
62                     X[i][j]=1+X[i1][j1];
63
64                 if(A[i][j]!='P')
65                 {
66                     ul++;
67                     if(ul>r) ul=1;
68                     nr++;
69                     coada[ul]={i,j};
70                 }
71
72                 if(A[i][j]=='+')
73                     usi[i][j]=T;
74             }
75         }
76     }
77     pr++;

```

```

78         if(pr>r)
79             pr=1;
80         nr--;
81     }
82 }
83
84 int main()
85 {
86     fin>>N>>M>>T;
87     for(i=1;i<=N;i++)
88     {
89         for(j=1;j<=M;j++)
90         {
91             fin>>ch;
92             A[i][j]=ch;
93             if(ch=='W') {lw=i; cw=j;}
94             if(ch=='E') {le=i; ce=j;}
95             if(ch=='P') P++;
96             W[i][j]=infinit;
97             E[i][j]=infinit;
98         }
99     }
100
101 //determinam W[i][j]=numarul minim de secunde in care se poate ajunge
102 // la [i][j] pornind de la [lw][cw], fara portaluri
103 solve(W,lw,cw);
104 solve(E,le,ce);
105
106 //determinam cele mai mari doua distante de la portaluri spre EXIT
107 de1=0;
108 de2=0;
109 y=0;
110 for(i=1;i<=N;i++)
111     for(j=1;j<=M;j++)
112         if(A[i][j]=='P' && E[i][j]<infinit)
113         {
114             y++;
115             if(E[i][j]>=de1)
116             {
117                 de2=de1;
118                 de1=E[i][j];
119             }
120             else
121                 if(E[i][j]>de2)
122                     de2=E[i][j];
123         }
124
125 dmin=W[le][ce];
126 c1=0;
127 for(k=0;k<=3;k++)
128 {
129     i2=i1+dl[k];
130     j2=j1+dc[k];
131     if(i2>=1 && i2<=N && j2>=1 && j2<=M && A[i2][j2]=='.')
132         c1++;
133 }
134
135 if(c1==0)
136     dc1=2+T;
137 else
138     dc1=2;
139
140 //imbunatatim solutia minima folosind portaluri
141 //W trebuie sa ajunga la un portal si apoi de la orice alt portal
142 //trebuie sa se poate ajunge la E
143 for(i=1;i<=N;i++)
144     for(j=1;j<=M;j++)
145         if(A[i][j]=='P' && W[i][j]<infinit)
146         {
147             y1=y;
148             d=E[i][j];
149             d2=W[i][j];
150             if(d<infinit) y1--;
151             if(y1+1==P)
152             {
153                 if(d==de1)

```

```

154             d1=de2;
155         else
156             d1=min(de1,d+dcl);
157
158         if(dmin>d1+d2)
159             dmin=d1+d2;
160     }
161 }
162
163 if(dmin==infinit)
164     dmin=-1;
165
166 fout<<dmin;
167 fout.close();
168 fin.close();
169 return 0;
170 }
```

Listing 19.3.7: walle2.cpp

```

1 //O(N*M)
2 #include<fstream>
3 #include<iostream>
4
5 #define nmax 502
6 #define nmax2 250002
7
8 using namespace std;
9
10 ifstream fin ("walle.in");
11 ofstream fout("walle.out");
12
13 int N,M,T,le,ce,lw,cw,i,j,i1,j1,i2,j2,k,ul,pr,nr,ul2,pr2,nr2,r,P,y1,y,z,
14     dmin,de1,de2,d,d1,d2,c1,dcl;
15 int W[nmax][nmax],E[nmax][nmax],usi[nmax][nmax],infinit=1000000000;
16
17 int dl[4]={-1, 0,+1, 0};
18 int dc[4]={ 0,+1, 0,-1};
19
20 char A[nmax][nmax],ch;
21
22 struct pozitie
23 {
24     int lin,col;
25 } coada[nmax2],coada2[nmax2];
26
27 void solve(int X[nmax][nmax],int lx, int cx)
28 {
29     int il,j1,k,i,j;
30     ul=0;
31     pr=1;
32     nr=0;
33     ul2=0;
34     pr2=1;
35     nr2=0;
36     r=nmax2-2;
37     X[lx][cx]=0;
38     ul++;
39     nr++;
40     coada[ul]={lx,cx};
41
42     while(nr+nr2)
43     {
44         if((nr==0) || (nr>0 && nr2>0 &&
45             X[coada[pr].lin][coada[pr].col] >
46             usi[coada2[pr2].lin][coada2[pr2].col]))
47         {
48             pr--;
49             if(pr==0) pr=r;
50             nr++;
51             coada[pr]=coada2[pr2];
52             pr2++;
53             if(pr2>r) pr2=1;
54             nr2--;
55         }
56     }
57 }
```

```

56
57     i1=coada[pr].lin;
58     j1=coada[pr].col;
59
60     for (k=0;k<=3;k++)
61     {
62         i=i1+dl[k];
63         j=j1+dc[k];
64         if(i>=1 && i<=N && j>=1 && j<=M &&
65             A[i][j]!='#' && X[i][j]==infinit)
66         {
67             if(A[i1][j1]=='+')
68                 X[i][j]=1+usi[i1][j1];
69             else
70                 X[i][j]=1+X[i1][j1];
71
72             if(A[i][j]!='P' && A[i][j]!='+')
73             {
74                 ul++;
75                 if(ul>r) ul=1;
76                 nr++;
77                 coada[ul]={i,j};
78             }
79
80             if(A[i][j]=='+')
81             {
82                 usi[i][j]=T+X[i][j];
83                 ul2++;
84                 if(ul2>r) ul2=1;
85                 nr2++;
86                 coada2[ul2]={i,j};
87             }
88         }
89     }
90
91     pr++;
92     if(pr>r) pr=1;
93     nr--;
94 }
95 }
96
97 int main()
98 {
99     fin>>N>>M>>T;
100    for(i=1;i<=N;i++)
101    {
102        for(j=1;j<=M;j++)
103        {
104            fin>>ch;
105            A[i][j]=ch;
106            if(ch=='W') {lw=i; cw=j;}
107            if(ch=='E') {le=i; ce=j;}
108            if(ch=='P') P++;
109            W[i][j]=infinit; E[i][j]=infinit;
110        }
111    }
112    /// determinam W[i][j]=numarul minim de secunde in care se poate ajunge
113    /// la [i][j] pornind de la [lw][cw], fara portaluri
114    solve(E,le,ce);
115
116    ///determinam cele mai mari doua distante de la portaluri spre EXIT
117    de1=0;
118    de2=0;
119    y=0;
120    for(i=1;i<=N;i++)
121        for(j=1;j<=M;j++)
122        {
123            if(A[i][j]=='P' && E[i][j]<infinit)
124            {
125                y++;
126                if(E[i][j]>=de1)
127                {
128                    de2=de1;
129                    de1=E[i][j];
130                    il=i;
131                    jl=j;
132            }
133        }
134    }
135 }
```

```

132         }
133     else
134     {
135         if(E[i][j]>de2) de2=E[i][j];
136     }
137 }
138
139 usi[i][j]=0;
140 }
141
142 solve(W,lw,cw);
143 dmin=W[le][ce];
144 c1=0;
145 for(k=0;k<=3;k++)
146 {
147     i2=i1+dl[k];
148     j2=j1+dc[k];
149     if(i2>=1 && i2<=N && j2>=1 && j2<=M && A[i2][j2]=='.')
150         c1++;
151 }
152
153 if(c1==0)
154     dc1=2+T;
155 else
156     dc1=2;
157
158 // imbunatatim solutia minima folosind portaluri
159 // W trebuie sa ajunga la un portal si apoi de la orice alt portal
160 // trebuie sa se poata ajunge la E
161 for(i=1;i<=N;i++)
162     for(j=1;j<=M;j++)
163         if(A[i][j]=='P' && W[i][j]<infinit)
164         {
165             y1=y;
166             d=E[i][j];
167             d2=W[i][j];
168             if(d<infinit) y1--;
169             if(y1+1==P)
170             {
171                 if(d==de1)
172                     d1=de2;
173                 else
174                     d1=min(de1,d+dc1);
175
176                 if(dmin>d1+d2)
177                     dmin=d1+d2;
178             }
179         }
180
181 if(dmin==infinit)
182     dmin=-1;
183
184 fout<<dmin;
185 fout.close();
186 fin.close();
187 return 0;
188 }

```

Listing 19.3.8: wallecos2.cpp

```

1 // O(N*M)
2 #include <stdio.h>
3 #include <string.h>
4
5 #define MAX_SIZE 502
6 #define INF      1000000000
7
8 char a[MAX_SIZE][MAX_SIZE];
9 int w[MAX_SIZE][MAX_SIZE], e[MAX_SIZE][MAX_SIZE];
10 int q[2][MAX_SIZE*MAX_SIZE];
11 int dx[] = { -1, +1, 0, 0 };
12 int dy[] = { 0, 0, -1, +1 };
13
14 void swap(int *x, int *y)
15 {

```

```

16     *x ^= *y; *y ^= *x; *x ^= *y;
17 }
18
19 void bestTimeFrom(int x, int y, char a[][MAX_SIZE], int n, int m, int t, int d[] [
    MAX_SIZE])
20 {
21     int beg[] = {0, 0};
22     int end[] = {0, 0};
23     int delay[] = {0, t};
24
25     for (int i = 0; i < n; i++)
26         for (int j = 0; j < m; j++)
27             d[i][j] = INF;
28
29     q[0][end[0]++] = x*m + y;
30     d[x][y] = 0;
31     for (int now = 0; beg[0] < end[0] || beg[1] < end[1]; now++)
32     {
33         if (beg[0] == end[0])
34         {
35             int v = q[1][beg[1]];
36             x = v / m;
37             y = v % m;
38             now = d[x][y] + delay[1];
39         }
40
41         for (int i = 0; i < 2; i++)
42             for (; beg[i] < end[i]; beg[i]++)
43             {
44                 int v = q[i][beg[i]];
45                 x = v / m;
46                 y = v % m;
47                 if (d[x][y] + delay[i] > now) break;
48                 for (int k = 0; k < 4; k++)
49                 {
50                     int r = x + dx[k], c = y + dy[k];
51                     if (d[r][c] == INF && a[r][c] != '#')
52                     {
53                         d[r][c] = now + 1;
54                         if (a[r][c] == ',') q[0][end[0]++] = r*m + c;
55                         if (a[r][c] == '+') q[1][end[1]++] = r*m + c;
56                     }
57                 }
58             }
59         }
60     }
61 }
62
63 void tryWithOnePortal(char a[][MAX_SIZE],
64                       int e[][MAX_SIZE], int n, int m, int *ans,
65                       int *m1, int *m2)
66 {
67     int x1 = -1, x2 = -1;
68     for (int i = 0; i < n; i++)
69         for (int j = 0; j < m; j++)
70             if (a[i][j] == 'P')
71             {
72                 int x3 = e[i][j];
73                 if (x3 > x2) swap(&x3, &x2);
74                 if (x2 > x1) swap(&x2, &x1);
75             }
76
77     *m1 = x1;
78     *m2 = x2;
79
80     for (int i = 0; i < n; i++)
81         for (int j = 0; j < m; j++)
82             if (a[i][j] == 'P')
83             {
84                 int d = (e[i][j] == x1) ? x2 : x1;
85                 if (w[i][j] + d < *ans)
86                     *ans = w[i][j] + d;
87             }
88 }
89
90 int reuse(int x, int y, char a[][MAX_SIZE], int t)

```

```

91  {
92      int cost = INF;
93      for (int k = 0; k < 4; k++)
94      {
95          int r = x + dx[k], c = y + dy[k];
96          if (a[r][c] == '.') return 2;
97          if (a[r][c] == '+') cost = t + 2;
98      }
99
100     return cost;
101 }
102
103 int main()
104 {
105     FILE *f = fopen("walle.in", "r");
106     FILE *g = fopen("walle.out", "w");
107
108     int n, m, t, wx, wy, ex, ey, m1, m2;
109
110     fscanf(f, "%d %d %d", &n, &m, &t);
111     memset(a[0], '#', m + 2);
112
113     for (int i = 1; i <= n; i++)
114     {
115         a[i][0] = '#';
116         fscanf(f, "%s", &a[i][1]); a[i][m + 1] = '#';
117         for (int j = 1; j <= m; j++)
118         {
119             if (a[i][j] == 'W') a[i][j] = '.', wx = i, wy = j;
120             if (a[i][j] == 'E') a[i][j] = '.', ex = i, ey = j;
121         }
122     }
123
124     memset(a[n + 1], '#', m + 2);
125     n += 2;
126     m += 2;
127
128     bestTimeFrom(wx, wy, a, n, m, t, w);
129     bestTimeFrom(ex, ey, a, n, m, t, e);
130
131     int ans = w[ex][ey];
132     tryWithOnePortal(a, w, e, n, m, &ans, &m1, &m2);
133
134     for (int i = 0; i < n; i++)
135         for (int j = 0; j < m; j++)
136             if (a[i][j] == 'P')
137             {
138                 int d = reuse(i, j, a, t) + ((e[i][j] == m1) ? m2 : m1);
139                 if (e[i][j] > d) e[i][j] = d;
140             }
141
142     tryWithOnePortal(a, w, e, n, m, &ans, &m1, &m2);
143     fprintf(g, "%d", (ans != INF) ? ans : -1);
144
145     fclose(f);
146     fclose(g);
147
148     return 0;
149 }
```

19.4 nozero

Problema 4 - nozero

Se dau N și K .

100 de puncte

Cerințe

Se cere să se determine pentru a K -a permutare în ordine lexicografică, a sirului 1, 2, 3, ..., N , câte poziții p există astfel încât nici p și nici valoarea de pe poziția p nu conțin cifra zero.

Date de intrare

Pe prima linie din fișierul de intrare **nozero.in** se află scrise numerele N și K , separate printr-un spațiu.

Date de ieșire

În fișierul de ieșire **nozero.out** se va scrie valoarea căutată.

Restricții și precizări

- $1 \leq N, K \leq 10^9$
- Pentru teste valorând 16 puncte $1 \leq K, N \leq 1000$
- Pentru alte teste valorând 33 puncte $N \leq 500000$
- Pentru alte teste valorând 14 puncte $K = 1$
- Un sir p_1, p_2, \dots, p_N este mai mic lexicografic decât un alt sir q_1, q_2, \dots, q_N , dacă există o poziție i , $1 \leq i \leq N$, astfel încât $p_i < q_i$ și $p_j = q_j$, pentru orice j , $1 \leq j < i$

Exemple

nozero.in	nozero.out	Explicații
10 2	8	<p>A doua permutare în ordine lexicografică, de lungime 10, este 1 2 3 4 5 6 7 8 10 9.</p> <p>Valoarea 9 nu conține cifra 0, dar se află pe poziția 10, care conține cifra 0.</p> <p>Valoarea 10, de la poziția 9, conține cifra 0.</p> <p>Toate celelalte 8 valori nu conțin cifra 0 și se află pe pozițiile care nu conțin cifra 0.</p>

Timp maxim de executare/test: Windows: **0.3** secunde, Linux: **0.3** secunde

Memorie: total **128 MB**

Dimensiune maximă a sursei: **20 KB**

19.4.1 Indicații de rezolvare

stud. Iordache Ioan-Bogdan, Universitatea din București

Observația principală care ne duce la rezolvarea problemei este că numărul K este mic în comparație cu numărul total de permutări de ordin N ($N!$). De aceea pentru orice $N \geq 13$ și orice $K \leq 10^9$, primele $N - 13$ poziții vor fi fixate (avem valoarea egală cu poziția) și doar ultimele cel mult 13 valori vor fi permute.

Astfel ajungem la următoarele subprobleme:

Subproblemă 1: determinam a K -a permutare în ordine lexicografică de dimensiune cel mult 13 (notăm această dimensiune cu M).

Această subproblemă se poate rezolva în mai multe moduri încrucișat lungimea permutărilor este mică. O variantă ar fi să încercăm să fixăm pe rând de la stânga la dreapta valorile. Pentru poziția i încercăm în ordine crescătoare toate valorile care nu au fost puse încă, dacă o fixăm pe aceasta știm că elementele din dreapta ei pot fi permute în $(M - i)!$ moduri. Dacă numărul de moduri în care mai putem permuta ultimele $M - i$ valori este mai mic decât K , scădem din K acest număr și încercăm o valoare mai mare pe poziția i . La final vom avea toate valorile setate, iar $K = 0$.

Subproblemă 2: determinarea numărului de valori care conțin cifra 0 și sunt mai mici sau egale decât un P dat.

Deoarece știm că primele aproximativ $N - 13$ valori (al căror număr îl vom nota cu P) sunt egale cu poziția lor din permutare, este suficient să număram câte dintre aceste valori conțin 0.

Pentru un număr de cifre c fixat avem 9^c valori de c cifre care nu conțin cifra 0. Cu această relație numărăm toate valorile care au numărul de cifre strict mai mic decât numărul de cifre ale lui P .

Mai avem de numărat valorile care au același număr de cifre cu P și nu conțin 0. Fiind mai mici decât P și având același număr de cifre, putem grupa aceste valori după lungimea celui mai lung prefix pe care îl au în comun cu P . Dacă acest prefix conține 0 nu avem ce număra, iar dacă nu, vedem câte cifre nenule mai mici strict decât cifra de pe poziția imediat următoare prefixului

din P există, iar pentru fiecare din aceste variante celelalte cifre pot fi fixate cu oricare dintre cele 9 valori nenule posibile.

Din cele două subprobleme putem determina câte poziții sunt valide (poziția nu conține cifra 0 și valoarea de la acea poziție nu conține 0).

Pentru abordări diferite ale fiecărei subprobleme se pot obține punctaje parțiale:

- 16 puncte dacă se generează iterativ primele K permutări de lungime N și se verifică fiecare poziție separat.

- 33 puncte dacă se rezolvă subproblema 1, dar apoi se parcurg toate valorile pentru a le identifica pe cele care conțin 0.

- 14 puncte pentru $K = 1$ (permutarea identică) dacă se face doar numărarea de la subproblema 2.

19.4.2 *Rezolvare detaliată

19.4.3 Cod sursă

Listing 19.4.1: brute-dani.cpp

```

1 #include<stdio.h>
2 #include<math.h>
3 #include<vector>
4 #include<map>
5
6 using namespace std;
7
8 #define NMAX 100005
9
10 map<int, bool> apparition;
11 vector<int> digits;
12
13 int n, k, answer;
14 long long fact[NMAX];
15
16 int computeFactorials(int n, int k)
17 {
18     fact[0] = 1;
19     for(int i = 1; i <= n; i++)
20     {
21         fact[i] = fact[i - 1] * i;
22         if(fact[i] > k)
23             return i - 1;
24     }
25
26     return n;
27 }
28
29
30 int noZeroVerdict(int value)
31 {
32     if(!value)
33         return 1;
34     return ((value % 10 != 0) & noZeroVerdict(value / 10));
35 }
36
37 int countFixedSolutions(int max_elem)
38 {
39     for(int i = 1; i <= max_elem; i++)
40         answer += noZeroVerdict(i);
41
42     return answer;
43 }
44
45 int nextActiveValue(int value)
46 {
47     value++;
48     while(value < n && apparition[value])
49         value++;
50 }
```

```

51     return value;
52 }
53
54 int main ()
55 {
56
57     freopen("nozero.in", "r", stdin);
58     freopen("nozero.out", "w", stdout);
59
60     scanf("%d%d", &n, &k);
61     k--;
62
63     int last = computeFactorials(n, k);
64     int answer = countFixedSolutions(n - last - 1);
65
66     for(int i = n - last; i <= n; i++)
67     {
68         int level = n - i;
69         int value = nextActiveValue(n - last - 1);
70         while(k >= fact[level])
71         {
72             k -= fact[level];
73             value = nextActiveValue(value);
74         }
75         apparition[value] = true;
76         answer += (noZeroVerdict(value) & noZeroVerdict(i));
77     }
78
79     printf("%d\n", answer);
80
81     return 0;
82 }
```

Listing 19.4.2: razvan-nozero.cpp

```

1 #include <bits/stdc++.h>
2 #include <iostream>
3
4 using namespace std;
5
6 ifstream in("nozero.in");
7 ofstream out("nozero.out");
8
9 long long n,k,ans,b,c,nr[ 100 ], fact[ 100 ], v[ 100 ], p[ 100 ], a;
10 int z;
11
12 int hasZero( int x )
13 {
14     while( x )
15     {
16         if( x % 10 == 0 )
17             return 1;
18         x /= 10;
19     }
20     return 0;
21 }
22
23 int main()
24 {
25     nr[ 0 ] = 1;
26     nr[ 1 ] = 9;
27     fact[ 0 ] = fact[ 1 ] = p[ 0 ] = 1;
28     p[ 1 ] = 2;
29     for( int i = 2 ; i <= 15 ; i++ )
30     {
31         nr[ i ] = nr[ i - 1 ] * 9;
32         fact[ i ] = fact[ i - 1 ] * i;
33         p[ i ] = i + 1;
34     }
35
36     in >> n >> k;
37     a = n;
38     n = max( 0ll , n - 15 );
39     b = n;
40 }
```

```

41     z = n;
42     string st;
43     while(z)
44     {
45         st.push_back( z % 10 + '0' );
46         z /= 10;
47     }
48     reverse(st.begin() , st.end());
49     for( int i = 1 ; i < st.size() ; i++ )
50     {
51         ans += nr[ i ];
52     }
53     int zero = 1;
54     if( st.size() == 0 )
55         zero = 0;
56     for( int i = 0 ; i < st.size() ; i++ )
57     {
58         int c = st[ i ] - '0';
59         if( c == 0 )
60         {
61             zero = 0;
62             break;
63         }
64         ans += (c - 1) * nr[ st.size() - i - 1 ];
65     }
66     ans += zero;
67     a = min( 1511 , a );
68     k--;
69     for( int i = 1 ; i <= a ; i++ )
70     {
71         c = ( k / fact[ a - i ] );
72         v[ i ] = p[ c ];
73         for( int j = c ; j <= a ; j++ )
74         {
75             p[ j ] = p[ j + 1 ];
76         }
77         k -= c * fact[ a - i ];
78         if( !hasZero( v[ i ] + n ) && !hasZero( i + n ) )
79             ans++;
80     }
81     out << ans;
82
83     return 0;
84 }
```

Listing 19.4.3: sursa-dani.cpp

```

1 #include<stdio.h>
2 #include<math.h>
3 #include<vector>
4 #include<map>
5
6 using namespace std;
7
8 #define NMAX 100005
9
10 map<int, bool> apparition;
11 vector<int> digits;
12
13 int n, k, answer;
14 long long fact[NMAX];
15
16 int computeFactorials(int n, int k)
17 {
18     fact[0] = 1;
19     for(int i = 1; i <= n; i++)
20     {
21         fact[i] = fact[i - 1] * i;
22         if(fact[i] > k)
23             return i - 1;
24     }
25     return n;
26 }
27
28 int countFixedSolutions(int max_elem)
```

```

29  {
30      int answer = 0;
31      if(!max_elem)
32          return 0;
33      while(max_elem)
34      {
35          digits.push_back(max_elem % 10);
36          max_elem /= 10;
37      }
38
39      for(int i = 1; i < (int)digits.size(); i++)
40          answer += pow(9, i);
41
42      for(int i = digits.size() - 1; i >= 0; i--)
43      {
44          if(!digits[i])
45              return answer;
46          answer += pow(9, i) * (digits[i] - 1);
47      }
48
49      return answer + 1;
50  }
51
52  int noZeroVerdict(int value)
53  {
54      if(!value)
55          return 1;
56      return ((value % 10 != 0) & noZeroVerdict(value / 10));
57  }
58
59  int nextActiveValue(int value)
60  {
61      value++;
62      while(value < n && apparition[value])
63          value++;
64
65      return value;
66  }
67
68  int main ()
69  {
70      freopen("nozero.in", "r", stdin);
71      freopen("nozero.out", "w", stdout);
72
73      scanf("%d%d", &n, &k);
74      k--;
75
76      int last = computeFactorials(n, k);
77      int answer = countFixedSolutions(n - last - 1);
78 //     printf("last = %d si answer = %d\n", last, answer);
79
80      for(int i = n - last; i <= n; i++)
81      {
82          int level = n - i;
83          int value = nextActiveValue(n - last - 1);
84          while(k >= fact[level])
85          {
86              k -= fact[level];
87              value = nextActiveValue(value);
88          }
89          apparition[value] = true;
90          answer += (noZeroVerdict(value) & noZeroVerdict(i));
91      }
92
93      printf("%d\n", answer);
94
95      return 0;
96  }

```

19.5 pericol

Problema 5 - pericol

100 de puncte

Avem o clasă cu N elevi inventivi. Pentru fiecare dintre ei se cunoaște un *coeficient de atitudine*

dine reprezentat printr-un număr natural nenul v_k . Interacțiunile din cadrul grupului de elevi al clasei produc efecte secundare importante și conducerea școlii a definit o mărime scalară numită **indicator de pericol** care măsoară influența pe care un elev o are asupra celorlalți elevi din clasă. **Indicatorul de pericol** asociat elevului k , $1 \leq k \leq N$, se obține calculând cel mai mare divizor comun $d_{k,j}$ pentru fiecare pereche (v_k, v_j) , $1 \leq j \leq N$, $j \neq k$ și apoi însumând valorile calculate.

Cerințe

Să se calculeze, pentru fiecare elev, **indicatorul de pericol** asociat lui.

Date de intrare

În fișierul text **pericol.in** pe prima linie se află numărul natural N . Pe a doua linie se află N numere naturale nenele, separate prin câte un spațiu, reprezentând *coeficienții de atitudine* ai celor N elevi.

Date de ieșire

În fișierul text **pericol.out** se vor scrie, pe prima linie, N numere naturale, separate prin câte un spațiu, al k -lea număr natural reprezentând **indicatorul de pericol** asociat celui de-al k -lea elev.

Restricții și precizări

- $1 \leq N \leq 2 \cdot 10^5$
- $1 \leq v_k \leq 10^7$, $1 \leq k \leq N$
- Pentru teste în valoare de 14 puncte $N \leq 2000$
- Pentru alte teste în valoare de 5 puncte $v_k \leq 2000$
- Pentru alte teste în valoare de 39 de puncte $v_k \leq 2 \cdot 10^6$

Exemple

pericol.in	pericol.out	Explicații
6 2 3 4 5 6 4	8 7 10 5 10 10	De exemplu indicatorul de pericol al celui de-al 5-lea elev se calculează astfel $(2,6) + (3,6) + (4,6) + (5,6) + (4,6) = 2+3+2+1+2 = 10$

Timp maxim de executare/test: Windows 15.0 secunde, Linux: 6.0 secunde

Memorie: total 512MB

Dimensiune maximă a sursei: 20KB

19.5.1 Indicații de rezolvare

stud. Budău Adrian, Universitatea din București

Soluție $N^2 \log(VMAX)$ (14p)

Calculăm pentru toate perechile posibile *cmmdc-ul* și updatăm rezultatul corespunzător pentru fiecare poziție.

Soluție $VMAX^2 \log(VMAX)$ (5p)

Menținem un *vector de frecvență* pentru valorile din sir și calculăm *cmmdc-ul* doar pentru perechi de valori distincte. Astfel pentru o anumită valoare putem calcula suma *cmmdc-urilor* cu valorile din sir iterând prin valorile distincte și înmulțind *cmmdc-ul* calculat cu frecvența valorilor respective.

Soluție $VMAX \log^2(VMAX)$ (58p) Varianta 1

Pentru fiecare pereche de tip (*valoare, divizor*) putem calcula câte valori din sir dăi *cmmdc-ul* dintre valoarea din sir și valoarea din pereche egal cu *divizor*.

Notăm cu V și d *valoarea și divizorul* ales, *raspuns[V][d]* este numărul pe care vrem să îl calculăm pentru această pereche, iar *count[d]* este numărul de numere divizibile cu d din sir.

```
raspuns[V][d]=count[d]-suma(raspuns[V][D]/D multiplu de d si divizor al lui V si D>
```

Cu aceste valori calculate este ușor să răspundem acum pentru fiecare valoare din sir. Divizorii numerelor se pot determina folosind *Ciurul lui Eratosthenes* și avem nevoie de $O(VMAX \log VMAX)$ memorie.

Soluție $VMAX \log^2(VMAX)$ (58p) Varianta 2

Având un *cmmdc* fixat pentru o valoare vrem să determinăm câte valori din sir dău acest *cmmdc* cu ea. Cazul mai ușor este când *cmmdc* = 1.

Numim *număr elementar*, un număr care conține toate numerele prime din scrierea sa la puterea 1.

Din *principiul includerii și excluderii* avem pentru o valoare fixată că numărul de valori din sir care sunt prime cu valoarea este:

```
suma( (-1)^(nr_factori_primi(v)) * count[v] / v elementar și v divizor al valorii
```

Count are semnificația de mai sus.

Pentru *cmmdc* diferit de 1 și *cmmdc* divide *valoarea*, formula devine:

```
suma( (-1)^(nr_factori_primi(v)) * count[cmmdc*v] / v elementar și v divizor al valorii /
```

Înmulțim cu *cmmdc* la final pentru a calcula suma acestor *cmmdc*-uri, nu doar numărul lor.

Putem întoarce formula de mai sus pentru un *cmmdc* fixat și un *v* fixat, ca să calculăm contribuția sa pentru suma corespunzătoare unei valori.

Pentru orice $K \geq 1$:

```
suma[K * cmmdc * V] += (-1)^(nr_factori_primi(v)) * count[cmmdc*v] * cmmdc.
```

Acum când răspundem pentru fiecare *valoare* din sir trebuie să avem grija să scădem din suma calculată pentru *valoarea* respectiva chiar *valoarea* sa (deoarece am numărat și *cmmdc*-ul cu ea însăși).

Soluție $VMAX \log(VMAX)$ (100p)

Soluția de 100p se bazează pe soluția anterioară, făcând următoarea observație:

count[p] va fi adăugat la toți multiplii de *p* cu același coeficient.

Acest coeficient este dat de următoarea formulă:

```
coef[p] = suma((-1)^(nr_factori_primi(v)) * p/v) / v divizor elementar al lui p
```

Cu acești coeficienți calculați:

```
suma[V] = suma(coef[p] * count[p] | p divizor al lui V).
```

19.5.2 *Rezolvare detaliată

19.5.3 Cod sursă

Listing 19.5.1: adrian-100.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <cassert>
5 #include <algorithm>
6
7 using namespace std;
8
9 int main()
10 {
11     ifstream cin("pericol.in");
12     ofstream cout("pericol.out");
13
14     int N; assert(cin >> N);

```

```

15     assert(2 <= N && N <= 400000);
16
17     vector<int> V(N);
18     for (int i = 0; i < N; ++i)
19     {
20         assert(cin >> V[i]);
21         assert(1 <= V[i] && V[i] <= 10000000);
22     }
23
24     int max_value = *max_element(V.begin(), V.end()) + 1;
25
26     vector<int> count(max_value, 0);
27     for (auto &x : V)
28         count[x]++;
29
30     vector<bool> elementary(max_value, true);
31     vector<int64_t> primes(max_value, 0), formula(max_value, 0),
32                           answer(max_value, 0);
33
34     primes[1] = 2;
35     // for gcd in 1..max_val
36     // for elementary in 1..max_val
37     // answer[elementary * X * gcd] +=
38     //     multiples_of[elementary * gcd] * sign[elementary] * gcd
39     // translated:
40     // formula[elementary * gcd] += sign[elementary] * gcd
41     // answer[X * V] += formula[V] * multiples_of[V]
42     for (int i = 1; i < max_value; ++i)
43     {
44         if (primes[i] == 0)
45         {
46             for (int j = i; j < max_value; j += i)
47                 primes[j]++;
48             if (1LL * i * i < max_value)
49             {
50                 for (int j = i * i; j < max_value; j += i * i)
51                     elementary[j] = false;
52             }
53         }
54
55         if (elementary[i])
56         {
57             int sign = 1;
58             if (primes[i] % 2)
59                 sign = -1;
60             for (int j = i, ratio = 1; j < max_value; j += i, ++ratio)
61                 formula[j] += sign * ratio;
62         }
63
64         int frequency = 0;
65         for (int j = i; j < max_value; j += i)
66             frequency += count[j];
67
68         for (int j = i; j < max_value; j += i)
69             answer[j] += formula[i] * int64_t(frequency);
70     }
71
72     for (auto &x : V)
73         cout << answer[x] - x << " ";
74
75     cout << "\n";
76 }
```

Listing 19.5.2: pericol_bogdan_100.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 int main()
6 {
7     ifstream cin("pericol.in");
8     ofstream cout("pericol.out");
9
10    int n;
```

```

11     cin >> n;
12     vector<int> v(n);
13     for (auto& it : v)
14         cin >> it;
15
16     int vmax = *max_element(v.begin(), v.end()) + 5;
17
18     vector<int> freqv(vmax);
19     for (auto&& it : v)
20         freqv[it]++;
21
22     int elementarCount = 0;
23     vector<int> elementars(vmax);
24     vector<bool> cannotElementar(vmax);
25     vector<int> countMult(vmax);
26     vector<int> isPrime(vmax, true); isPrime[1] = false;
27     vector<int> sign(vmax, 1);
28     vector<int> maxMult(vmax);
29
30     for (int i = 1; i < vmax; ++i)
31     {
32         if (!cannotElementar[i])
33             elementars[elementarCount++] = i;
34         if (isPrime[i])
35         {
36             if (1LL * i * i < vmax)
37             {
38                 for (int j = i * i; j < vmax; j += i*i)
39                     cannotElementar[j] = true;
40             }
41         }
42
43         for (int j = i; j < vmax; j += i)
44         {
45             countMult[i] += freqv[j];
46             if (freqv[j])
47                 maxMult[i] = max(maxMult[i], j);
48             if (isPrime[i])
49             {
50                 sign[j] *= -1;
51                 if (j != i)
52                     isPrime[j] = false;
53             }
54         }
55     }
56
57     vector<long long> coef(vmax);
58     for (int i = 0; i < elementarCount; ++i)
59     {
60         int elem = elementars[i];
61         for (int p = elem; p < vmax; p += elem)
62             coef[p] += 1LL * sign[elem] * (p / elem);
63     }
64
65     vector<long long> raspuns(vmax);
66
67     for (int p = 1; p < vmax; ++p)
68     {
69         for (int i = p; i < vmax; i += p)
70         {
71             raspuns[i] += coef[p] * countMult[p];
72         }
73     }
74
75     for (int i = 0; i < n; ++i)
76         cout << raspuns[v[i]] - v[i] << ' ';
77     cout << '\n';
78
79     return 0;
80 }
```

Listing 19.5.3: pericol_bogdan_n2log.cpp

```

1 #include <bits/stdc++.h>
2 using namespace std;
```

```

3
4 const int DIM = 200005;
5 const int VMAX = 5000005;
6
7 int v[DIM];
8
9 inline int gcd(int x, int y)
10 {
11     while (y)
12     {
13         int r = x % y;
14         x = y;
15         y = r;
16     }
17     return x;
18 }
19
20 int main()
21 {
22     ifstream cin("pericol.in");
23     ofstream cout("pericol.out");
24
25     int n; cin >> n;
26     for (int i = 0; i < n; ++i)
27         cin >> v[i];
28
29     for (int i = 0; i < n; ++i)
30     {
31         long long res = 0;
32         for (int j = 0; j < n; ++j)
33         {
34             if (i == j)
35                 continue;
36             res += gcd(v[i], v[j]);
37         }
38         cout << res << ' ';
39     }
40     cout << '\n';
41
42     return 0;
43 }
```

Listing 19.5.4: pericol_bogdan_nlog2_better_vector.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 int main()
6 {
7     ifstream cin("pericol.in");
8     ofstream cout("pericol.out");
9
10    int n; cin >> n;
11    vector<int> v(n);
12    for (auto& it : v)
13        cin >> it;
14
15    int vmax = *max_element(v.begin(), v.end()) + 5;
16    vector<int> freqv(vmax);
17    for (auto&& it : v)
18        freqv[it]++;
19
20    int elementarCount = 0;
21    vector<int> elementars(vmax);
22    vector<bool> cannotElementar(vmax);
23    vector<int> countMult(vmax);
24    vector<int> isPrime(vmax, true); isPrime[1] = false;
25    vector<int> sign(vmax, 1);
26    vector<int> maxMult(vmax);
27
28    for (int i = 1; i < vmax; ++i)
29    {
30        if (!cannotElementar[i])
31            elementars[elementarCount++] = i;
```

```

32         if (isPrime[i])
33     {
34         if (1LL * i * i < vmax)
35         {
36             for (int j = i * i; j < vmax; j += i*i)
37                 cannotElementar[j] = true;
38         }
39     }
40
41     for (int j = i; j < vmax; j += i)
42     {
43         countMult[i] += freqv[j];
44         if (freqv[j])
45             maxMult[i] = max(maxMult[i], j);
46         if (isPrime[i])
47         {
48             sign[j] *= -1;
49             if (j != i)
50                 isPrime[j] = false;
51         }
52     }
53 }
54
55 vector<long long> coef(vmax);
56 for (int i = 0; i < elementarCount; ++i)
57 {
58     int elem = elementars[i];
59     for (int p = elem; p < vmax; p += elem)
60         coef[p] += 1LL * sign[elem] * (p / elem);
61 }
62
63 vector<long long> raspuns(vmax);
64 for (int p = 1; p < vmax; ++p)
65 {
66     for (int i = p; i < vmax; i += p)
67         raspuns[i] += coef[p] * countMult[p];
68 }
69
70 for (int i = 0; i < n; ++i)
71     cout << raspuns[v[i]] - v[i] << ' ';
72 cout << '\n';
73
74 return 0;
75 }
```

Listing 19.5.5: pericol bogdan nlog2 vector.cpp

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main()
5 {
6     ifstream cin("pericol.in");
7     ofstream cout("pericol.out");
8
9     int n; cin >> n;
10
11    vector<int> v(n);
12    for (int i = 0; i < n; ++i)
13        cin >> v[i];
14
15    vector<int> freqv( *max_element(v.begin(), v.end()) + 5 );
16    for (auto& it : v)
17        freqv[it]++;
18
19    int vmax = (int)freqv.size();
20    vector<int> countDiv(vmax);
21    for (int i = 1; i < vmax; ++i)
22        for (int j = i; j < vmax; j += i)
23            countDiv[j]++;
24
25    vector<vector<int>> divs(vmax);
26    for (int i = 1; i < vmax; ++i)
27    {
28        divs[i].resize(countDiv[i]);
29    }
30
```

```

29         countDiv[i] = 0;
30     }
31
32     for (int i = 1; i < vmax; ++i)
33         for (int j = i; j < vmax; j += i)
34             divs[j][countDiv[j]] = i;
35
36     vector<int> count(vmax);
37     for (int i = 1; i < vmax; ++i)
38         for (int j = i; j < vmax; j += i)
39             count[i] += freqv[j];
40
41     vector<int> raspuns(vmax);
42     vector<long long> ansForVal(vmax);
43     for (int i = 1; i < vmax; ++i)
44     {
45         if (!freqv[i])
46             continue;
47         for (int j = 0; j < countDiv[i]; ++j)
48             raspuns[divs[i][j]] = count[divs[i][j]];
49         for (int divBigIndex=countDiv[i]-1; divBigIndex >= 0; divBigIndex--)
50         {
51             int divBig = divs[i][divBigIndex];
52             for (int divSmallIndex = countDiv[divBig] - 2;
53                  divSmallIndex >= 0; divSmallIndex--)
54             {
55                 int divSmall = divs[divBig][divSmallIndex];
56                 raspuns[divSmall] -= raspuns[divBig];
57             }
58         }
59         for (int j = 0; j < countDiv[i]; ++j)
60             ansForVal[i] += 1LL * raspuns[divs[i][j]] * divs[i][j];
61     }
62
63     for (int i = 0; i < n; ++i)
64         cout << ansForVal[v[i]] - v[i] << ' ';
65     cout << '\n';
66
67     return 0;
68 }
```

Listing 19.5.6: pericol_bogdan_vmax2log.cpp

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 inline int gcd(int x, int y)
5 {
6     while (y)
7     {
8         int r = x % y;
9         x = y;
10        y = r;
11    }
12    return x;
13 }
14
15 int main()
16 {
17     ifstream cin("pericol.in");
18     ofstream cout("pericol.out");
19
20     int n; cin >> n;
21     vector<int> v(n);
22     for (auto& it : v)
23         cin >> it;
24
25     int vmax = *max_element(v.begin(), v.end()) + 1;
26     vector<int> freqv(vmax);
27     for (int i = 0; i < n; ++i)
28         freqv[v[i]]++;
29
30     vector<long long> raspuns(vmax);
31     for (int i = 1; i < vmax; ++i)
32     {
```

```

33         if (!frecv[i])
34             continue;
35         for (int j = 1; j < vmax; ++j)
36         {
37             if (!frecv[j])
38                 continue;
39             int g = gcd(i, j);
40             raspuns[i] += 1LL * g * frecv[j];
41         }
42     }
43
44     for (int i = 0; i < n; ++i)
45         cout << raspuns[v[i]] - v[i] << ' ';
46
47     cout << '\n';
48
49     return 0;
50 }
```

19.6 vip

Problema 6 - vip

100 de puncte

Două personaje ale căror nume se vor da în datele de intrare (momentan îi numim Bossanip și Dicsi) își petrec noptile prin discotecă.

Toată lumea știe că Bossanip este membru V.I.P în toate discotecile din lume și Dicsi profită mereu de celebritatea prietenului său. Ajuns pe meleaguri străine, Dicsi s-a confruntat cu o problemă foarte mare. Cum intră la V.I.P când este pe cont propriu? Astfel, Dicsi s-a apucat de infracțiuni precum furtul de identitate. Dicsi dorește să permute literele din numele lui (să găsească o anagramă a propriului nume) astfel încât noul nume să difere prin exact K poziții de numele lui Bossanip. Mai mult, dorește ca această anagramă să fie minimă lexicografic. Dacă reușește, este posibil să se dea drept Bossanip și să intre și el ca membru V.I.P.

Cerințe

Date de intrare

În fișierul text **vip.in** pe prima linie se află numărul natural T . Pe următoarele $3 \cdot T$ linii sunt descrise T seturi de date de intrare, fiecare set ocupă câte 3 linii astfel:

- pe prima linie a unui set se află scrise două numere naturale N (lungimea numelor reale ale lui Bossanip și Dicsi) și K ;

- pe a doua linie a unui set este scris numele lui Bossanip dat printr-un sir de caractere $s1$;
- pe a treia linie a unui set este scris numele lui Dicsi dat printr-un sir de caractere $s2$.

Din fericire pentru Dicsi, cele două personaje au nume de aceeași lungime.

Date de ieșire

În fișierul text **vip.out** se vor scrie, pe fiecare din cele T linii câte un sir de caractere, pe a j -a linie este scrisă anagrama corespunzătoare testului j (noul nume al lui Dicsi) sau -1 dacă nu există o astfel de anagramă.

Restricții și precizări

- $1 \leq N, K \leq 10^5$
- Suma valorilor lui N din cadrul seturilor de test $\leq 10^6$
- Toate literele sunt litere mici ale alfabetului englez
- Dacă nu există soluție pentru un test, atunci se va afișa valoarea -1
- Un sir p_1, p_2, \dots, p_N este mai mic lexicografic decât un alt sir q_1, q_2, \dots, q_N , dacă există o poziție i , $1 \leq i \leq N$, astfel încât $p_i < q_i$ și $p_j = q_j$, pentru orice j , $1 \leq j < i$
- Pentru 25% din punctaj se poate afișa orice soluție corectă care nu este neapărat minimă lexicografic

Exemple

vip.in	vip.out	Explicații
2	caaliisv	În primul set cea mai mică anagramă a sirului vasilica , din punct de vedere lexicografic, care diferă de sirul corleone pe exact 6 poziții, este caaliisv .
8 6 corleone vasilica	-1	
5 2 marko ghita		În al doilea set nici una din anagramele sirului ghita nu poate să difere pe exact două poziții de sirul marko .

Timp maxim de executare/test: Windows: **2.0** secunde, Linux: **0.4** secunde

Memorie: total **128 MB**

Dimensiune maximă a sursei: **20 KB**

19.6.1 Indicații de rezolvare

Student Posdarascu Eugenie Daniel âAŞ Universitatea din Bucuresti

Solutie $O(N * \sigma)$

Pasul 1:

În primul rând o să încercăm o abordare prin care determinăm orice soluție corectă, nu neapărat cea minimă lexicografică. Mai mult, considerăm cazul particular $N = K$ (toate pozițiile să difere).

Dacă există un caracter X a cărei sumă a frecvențelor din primul, respectiv al doilea sir, depășește N , este evident din *principiul cutiei* că nu avem soluție.

Formal, notând $f1[x] =$ frecvența caracterului x în primul sir și $f2[x] =$ frecvența caracterului x în al doilea sir, dacă există x pentru care $f1[x] + f2[x] > N$ atunci nu avem soluție.

Dacă nu există niciun caracter x pentru care se respectă proprietatea de mai sus, atunci putem demonstra că există întotdeauna soluție. O posibilă abordare pentru reconstituirea soluției ar fi în felul următor:

1. Selectăm cel mai frecvent element din sirul 1, notăm cu X .
2. Selectăm cel mai frecvent element din sirul 2, notăm cu Y .
3. Dacă $X \neq Y$, le putem împerechea. Altfel, împerechem X cu Z , al doilea cel mai frecvent element din sirul 2.

Prin urmare, putem afirma că invariantul pentru acest caz particular este $f1[x] + f2[x] \leq N$, pentru orice X de la 'a' la 'z'.

Pasul 2:

Acum vom încerca să rezolvăm problema pentru orice K din intervalul $[0, N]$ (încă nu dorim sirul minim lexicografic).

Să presupunem că din cele N poziții disponibile, având $K =$ numărul de poziții prin care diferă sirurile, notăm $P = N - K =$ numărul de poziții în care sirurile sunt similare.

Vom încerca să fixăm toate cele P poziții în care caracterele sirurilor coincid pentru a reduce problema la varianta de la **Pasul 1**. Astfel, o operație prin care selectăm un caracter X atât din sirul 1, cât și din sirul 2, pentru a le împerechea, va scădea valoarea sumei $f1[X] + f2[X]$ cu două unități.

Din moment ce vrem să reducem problema pe cazul particular de la **Pasul 1** în care lungimea să aibă dimensiunea K , invariantul devine $f1[X] + f2[X] \leq K$. Prin urmare, toate caracterele pentru care nu se respectă această proprietate trebuie împerecheate pentru a le reduce suma frecvențelor.

În concluzie, o abordare pentru această subproblemă este în felul următor:

1. Selectăm caracterul X cu $f1[X] + f2[X]$ maxim.
2. Împerechem un caracter X din sirul 1 cu un caracter X din sirul 2.
3. Repetăm procedeul de P ori, dacă se poate.
4. Dacă am putut actualiza toate cele P împerecheri, iar la final $f1[X] + f2[X] \leq K$ pentru orice X , avem soluție și rezolvăm în continuare aplicând procedeul de la **Pasul 1**.

Astfel, invariantul pentru această subproblemă devine:

1. Pentru fiecare caracter X unde $f1[X] + f2[X] > K$, calculăm numărul de operații necesar pentru ca această sumă să scadă sub K . Mai exact, $(f1[X] + f2[X] - k + 1)/2$. Evident, această valoare trebuie să fie mai mică decât $\min(f1[X], f2[X])$ (numărul maxim de împerecheri pe care le putem face cu caracterul X).

$$(f1[X] + f2[X] - k + 1) / 2 \leq \min(f1[X], f2[X])$$

2. Numărul total de operații pentru toate caracterele îl notăm cu $total_op$. Acest număr nu trebuie să depășească P (deoarece avem voie să aplicăm maxim P împerecheri).

$$total_op = \text{Suma}((f1[X] + f2[X] - k + 1) / 2) \leq P$$

3. Primele două restricții ne garantează că nu avem nevoie de mai mult de P împerecheri. Cu toate acestea, trebuie să aplicăm fix P împerecheri, deci numărul maxim de împerecheri pe care le putem face trebuie să fie cel puțin P .

$$P \leq \text{Suma}(\min(f1[X], f2[X]))$$

Pasul 3:

Vom încerca să determinăm soluția minimă lexicografic. Selectăm fiecare poziție de la stânga la dreapta și încercăm pe rând să fixăm fiecare caracter valid. Dacă în urma unei astfel de fixări, invariantul de la **Pasul 2** se păstrează, avem soluție. Altfel, încercăm alt caracter. Menținerea invariantului se poate realiza atât în $O(1)$, cât și în $O(\sigma)$.

Dacă invariantul de la **Pasul 2** nu se păstrează de la bun început, răspunsul este -1 .

Complexitatea finală optimă: $O(N * \sigma)$. O implementare atentă în $O(N * \sigma^2)$ obține de asemenea 100 de puncte.

19.6.2 *Rezolvare detaliată

19.6.3 Cod sursă

Listing 19.6.1: brute-dani.cpp

```

1 #include<stdio.h>
2 #include<algorithm>
3 using namespace std;
4
5 #define NMAX 100005
6 #define sigma 26
7
8 char s1[NMAX], s2[NMAX];
9 int sir1[NMAX], sir2[NMAX], t, n, k;
10 int freq1[NMAX], freq2[NMAX], answer[NMAX];
11
12 void readInput()
13 {
14     scanf("%d %d\n", &n, &k);
15     scanf("%s", s1 + 1);
16     scanf("%s", s2 + 1);
17     for(int i = 1; i <= n; i++)
18     {
19         sir1[i] = s1[i] - 'a' + 1;
20         sir2[i] = s2[i] - 'a' + 1;
21         freq1[sir1[i]]++;
22         freq2[sir2[i]]++;
23     }
24 }
25
26 bool checkInvariant(int equals, int different)
27 {
28     if(equals < 0 || different < 0)
29         return false;
30     int extra = 0;
31     for(int i = 1; i <= sigma; i++)
32     {
33         int steps = 0;

```

```

34         while(freq1[i] + freq2[i] > different)
35     {
36         steps++;
37         freq1[i]--;
38         freq2[i]--;
39         equals--;
40     }
41
42     extra += min(freq1[i], freq2[i]);
43
44     if(freq1[i] < 0 || freq2[i] < 0)
45         equals = -1;
46
47     freq1[i] += steps;
48     freq2[i] += steps;
49 }
50 return equals >= 0 && equals <= extra;
51 }
52
53 bool solveProblem()
54 {
55     int equals = n - k;
56     int different = k;
57     if(!checkInvariant>equals, different))
58         return false;
59
60     for(int i = 1; i <= n; i++)
61     {
62         freq1[sir1[i]]--;
63         for(int j = 1; j <= sigma; j++)
64         {
65             if(!freq2[j])
66                 continue;
67
68             freq2[j]--;
69
70             if(sir1[i] == j)
71                 equals--;
72             else
73                 different--;
74
75             if(checkInvariant>equals, different))
76             {
77                 answer[i] = j;
78                 break;
79             }
80
81             if(sir1[i] == j)
82                 equals++;
83             else
84                 different++;
85
86             freq2[j]++;
87         }
88     }
89     return true;
90 }
91
92 void writeOutput(bool verdict)
93 {
94     if(!verdict)
95     {
96         printf("-1\n");
97         return ;
98     }
99
100    for(int i = 1; i <= n; i++)
101        printf("%c", 'a' + answer[i] - 1);
102
103    printf("\n");
104 }
105
106 void clearAll()
107 {
108     for(int i = 1; i <= sigma; i++)
109         freq1[i] = freq2[i] = 0;

```

```

110 }
111
112 int main ()
113 {
114     freopen("vip.in", "r", stdin);
115     freopen("vip.out", "w", stdout);
116
117     scanf("%d", &t);
118     for(int i = 1; i <= t; i++)
119     {
120         readInput();
121         bool ok = solveProblem();
122         writeOutput(ok);
123         clearAll();
124     }
125
126     return 0;
127 }
```

Listing 19.6.2: max-lexicog.cpp

```

1 #include<stdio.h>
2 #include<algorithm>
3
4 using namespace std;
5
6 #define NMAX 100005
7 #define sigma 26
8
9 char s1[NMAX], s2[NMAX];
10 int sir1[NMAX], sir2[NMAX], t, n, k;
11 int freq1[NMAX], freq2[NMAX], answer[NMAX];
12
13 void readInput()
14 {
15     scanf("%d %d\n", &n, &k);
16     scanf("%s", s1 + 1);
17     scanf("%s", s2 + 1);
18     for(int i = 1; i <= n; i++)
19     {
20         sir1[i] = s1[i] - 'a' + 1;
21         sir2[i] = s2[i] - 'a' + 1;
22         freq1[sir1[i]]++;
23         freq2[sir2[i]]++;
24     }
25 }
26
27 bool checkInvariant(int equals, int different)
28 {
29     if(equals < 0 || different < 0)
30         return false;
31
32     int need_extra_op = 0, max_extra = 0;
33
34     for(int i = 1; i <= sigma; i++)
35     {
36         int extra_car = freq1[i] + freq2[i] - different;
37         int extra_op = (extra_car + 1) / 2;
38         if(extra_op > min(freq1[i], freq2[i]))
39             return false;
40         max_extra += min(freq1[i], freq2[i]);
41         if(extra_op > 0)
42             need_extra_op += extra_op;
43     }
44
45     return need_extra_op <= equals && equals <= max_extra;
46 }
47
48 bool solveProblem()
49 {
50     int equals = n - k;
51     int different = k;
52     if(!checkInvariant(equals, different))
53         return false;
54 }
```

```

55     for(int i = 1; i <= n; i++)
56     {
57         freq1[sir1[i]]--;
58         for(int j = sigma; j >= 1; j--)
59         {
60             if(!freq2[j])
61                 continue;
62             freq2[j]--;
63             if(sir1[i] == j)
64                 equals--;
65             else
66                 different--;
67
68             if(checkInvariant>equals, different))
69             {
70                 answer[i] = j;
71                 break;
72             }
73
74             if(sir1[i] == j)
75                 equals++;
76             else
77                 different++;
78
79             freq2[j]++;
80         }
81     }
82     return true;
83 }
84
85 void writeOutput(bool verdict)
86 {
87     if(!verdict)
88     {
89         printf("-1\n");
90         return ;
91     }
92
93     for(int i = 1; i <= n; i++)
94         printf("%c", 'a' + answer[i] - 1);
95
96     printf("\n");
97 }
98
99 void clearAll()
100 {
101     for(int i = 1; i <= sigma; i++)
102         freq1[i] = freq2[i] = 0;
103 }
104
105 int main ()
106 {
107     freopen("vip.in","r",stdin);
108     freopen("vip.out","w",stdout);
109
110     scanf("%d", &t);
111     for(int i = 1; i <= t; i++)
112     {
113         readInput();
114         bool ok = solveProblem();
115         writeOutput(ok);
116         clearAll();
117     }
118
119     return 0;
120 }
```

Listing 19.6.3: nsgima2.cpp

```

1 #include<stdio.h>
2 #include<algorithm>
3 using namespace std;
4
5 #define NMAX 100005
6 #define sigma 26
```

```

7
8 char s1[NMAX], s2[NMAX];
9 int sir1[NMAX], sir2[NMAX], t, n, k;
10 int freq1[NMAX], freq2[NMAX], answer[NMAX];
11
12 void readInput()
13 {
14     scanf("%d %d\n", &n, &k);
15     scanf("%s", s1 + 1);
16     scanf("%s", s2 + 1);
17     for(int i = 1; i <= n; i++)
18     {
19         sir1[i] = s1[i] - 'a' + 1;
20         sir2[i] = s2[i] - 'a' + 1;
21         freq1[sir1[i]]++;
22         freq2[sir2[i]]++;
23     }
24 }
25
26 bool checkInvariant(int equals, int different)
27 {
28     if(equals < 0 || different < 0)
29         return false;
30     int need_extra_op = 0, max_extra = 0;
31     for(int i = 1; i <= sigma; i++)
32     {
33         int extra_car = freq1[i] + freq2[i] - different;
34         int extra_op = (extra_car + 1) / 2;
35         if(extra_op > min(freq1[i], freq2[i]))
36             return false;
37         max_extra += min(freq1[i], freq2[i]);
38         if(extra_op > 0)
39             need_extra_op += extra_op;
40     }
41     return need_extra_op <= equals && equals <= max_extra;
42 }
43
44 bool solveProblem()
45 {
46     int equals = n - k;
47     int different = k;
48     if(!checkInvariant(equals, different))
49         return false;
50
51     for(int i = 1; i <= n; i++)
52     {
53         freq1[sir1[i]]--;
54         for(int j = 1; j <= sigma; j++)
55         {
56             if(!freq2[j])
57                 continue;
58             freq2[j]--;
59             if(sir1[i] == j)
60                 equals--;
61             else
62                 different--;
63
64             if(checkInvariant(equals, different))
65             {
66                 answer[i] = j;
67                 break;
68             }
69
70             if(sir1[i] == j)
71                 equals++;
72             else
73                 different++;
74
75             freq2[j]++;
76         }
77     }
78     return true;
79 }
80
81 void writeOutput(bool verdict)
82 {

```

```
83     if (!verdict)
84     {
85         printf("-1\n");
86         return ;
87     }
88
89     for(int i = 1; i <= n; i++)
90         printf("%c", 'a' + answer[i] - 1);
91
92     printf("\n");
93 }
94
95 void clearAll()
96 {
97     for(int i = 1; i <= sigma; i++)
98         freq1[i] = freq2[i] = 0;
99 }
100
101 int main ()
102 {
103     freopen("vip.in", "r", stdin);
104     freopen("vip.out", "w", stdout);
105
106     scanf("%d", &t);
107     for(int i = 1; i <= t; i++)
108     {
109         readInput();
110         bool ok = solveProblem();
111         writeOutput(ok);
112         clearAll();
113     }
114
115     return 0;
116 }
```

Capitolul 20

ONI 2018

20.1 anagrame

Problema 1 - anagrame

100 de puncte

Se dă două siruri S_1 și S_2 formate doar cu litere mici.

Numim subșir de lungime K al unui sir a un sir $a' = a_{i_1}, a_{i_2}, \dots, a_{i_K}$, astfel încât să avem: $i_1 < i_2 < \dots < i_K$.

Cerințe

Să se determine lungimea maximă a unui subșir din S_1 , format prin concatenarea unor anagrame ale sirului S_2 .

Dintre toate subșirurile cu lungime maximă se va determina cel care este cel mai mic lexicografic. Un sir de lungime na se consideră mai mic lexicografic decât un sir de lungime nb dacă există un indice i , astfel încât $a_1 = b_1, a_2 = b_2, \dots, a_{i-1} = b_{i-1}$ și $a_i < b_i$.

Un sir a este anagrama unui sir b dacă sortându-le crescător pe fiecare se obțin două siruri identice.

Date de intrare

În fișierul **anagrame.in**, pe prima linie se află un număr natural P . Pe linia a doua se află sirul S_1 , iar pe a treia linie se află sirul S_2 .

Date de ieșire

În fișierul **anagrame.out**, dacă $P = 1$, atunci pe prima linie se va scrie un număr natural reprezentând lungimea maximă a unui sir cu proprietatea cerută, iar dacă $P = 2$, atunci pe prima linie se va scrie subșirul de lungime maximă cu proprietatea cerută și minim lexicografic.

Restricții și precizări

- $1 \leq \text{Lungime}(S_2) \leq \text{Lungime}(S_1) \leq 100\ 000$
- Se garantează că cel puțin o anagramă a lui S_2 apare în S_1

Exemple:

anagrame.in	anagrame.out	Explicații
1 abbaaabababbaabaabba aba	15	Deoarece apare de 11 ori, S_2 poate să apară de cel mult 5 ori. Se observă subșirul format cu litere îngroșate și subliniate abbaaabababbaabaabba deci abaaabababaabaa este un subșir de lungime maximă, egală cu 15, cu proprietatea cerută.
2 abbaaabababbaabaabba aba	abaaabaabaabaab	Se observă că subșirul verifică proprietatea cerută

Timp maxim de executare/test: **0.3** secunde

Memorie: total **64 MB** din care pentru stivă **32 MB**

Dimensiune maximă a sursei: **20 KB**

20.1.1 Indicații de rezolvare

prof. Ionel-Vasile Pit-Rada, Colegiul National "Traian", Drobeta Turnu Severin

Solutie 1

student Răzvan Turturică , Universitatea Tehnică Cluj-Napoca

Se poate calcula de la dreapta la stânga sirului s utilizând un *vector de frecvență poz*[k] cu semnificația că de la poziția $poz[k]$ putem construi exact k anagrame în dreapta.

Pentru a determina aparițiile a căror concatenare este minimă lexicografic vom avea 2 indici cu următoarele semnificații: ST - prima poziție de pe care construim următoarea anagramă și DR - ultima poziție pe care o putem accesa astfel încât să putem avea un sir de lungime maximă.

Se observă că la fiecare pas dr este $poz[i]$, $i \rightarrow k, 1$. Se vor folosi 2 vectori de frecvență auxiliari $st[i]$ - câte caractere i mai trebuie adăugate în *stivă* și $dr[i]$ câte caractere i mai avem rămase pe pozițiile din dreapta până la DR .

Dacă adăugăm în *stivă* caracterul x , atunci $st[x]$ se scade cu 1; dacă scoatem elementul x din *stivă* atunci $st[x]$ crește cu 1.

Parcugând caracterele de la ST la DR procedăm astfel:

- 1) dacă *stiva* este goală, adăugăm elementul curent în *stivă*
- 2) cât timp elementul din *vârful stivei* este mai mare decât elementul curent, iar eliminarea acestuia asigură că $st[x] \leq dr[x]$, acesta se poate elimina
- 3) se adaugă elementul curent în *stivă*

Pentru fiecare apariție de anagramă, conținutul stivei este exact cea mai mică apariție lexicografică.

Se parcurge din nou ST DR pentru a se determina poziția minimă DR' până la care se găsește subșirul din *stivă*.

$DR' + 1$ va deveni următorul indice ST pentru următoarea apariție a anagramei.

Se concatenează toate stivele.

Pentru cerința 1 se afisează lungimea sirului, iar pentru 2 se afișează sirul.

Solutia 2 - prof. Ionel-Vasile Pit-Rada, Colegiul National "Traian", Drobeta Turnu Severin

Fie $n = lungime(a)$ și $m = lungime(b)$

Partea I

Determinăm $z[1]$ astfel încât $a[z[1]...n - 1]$ este cel mai scurt *sufix* din secvența $a[0...n - 1]$ care conține o anagramă a lui $b[]$. Apoi determinăm $z[2]$ astfel încât $a[z[2]...z[1] - 1]$ este cel mai scurt sufîx din secvența $a[0...z[1] - 1]$ care conține o anagramă a lui $b[]$.

Repetăm algoritmul până nu mai putem găsi anagrame.

Să presupunem că numărul de anagrame pe care le putem concatena este w .

Aveam deci determinante *pozițiile critice* $z[w] < z[w-1] < \dots < z[1]$.

Vom "oglindii" sirul $z[]$ astfel încât să avem $z[1] < z[2] < \dots < z[w]$.

Partea a II-a

Prima anagramă trebuie selectată din secvența $a[0...z[1]]$.

Anagrama trebuie să fie minimă lexicografic și să ocupe prefix de lungime minimă.

Fie $y[1]$ capătul drept al prefixului ocupat de anagrama optimă. Deci anagrama optimă poate fi selectată din secvența $a[0...y[1]]$.

Pentru a doua anagramă vom proceda analog selectând-o dintr-un prefix de lungime minimă din secvența $a[y[1] + 1...z[2]]$. La fel ca în cazul anterior determinăm $y[2]$ minim astfel ca anagrama optimă să fie selectată din $a[y[1] + 1...y[2]]$.

A treia anagramă se determină în secvența $a[y[2] + 1...z[3]]$.

Se continuă analog.

La pasul k dorim să determinăm optim a k -a anagramă din secvența $a[p...q]$.

Toate caracterele anagramei vor fi adăugate sirului soluție $c[]$. La final se va returna poziția capătului drept al prefixului din care a fost aleasă anagrama optimă.

Construim anagrama optimă în mai multe etape:

Etapa 1:

Parcurgem secvența $a[p...q]$, de la stânga spre dreapta până găsim o poziție i astfel încât dacă nu selectăm un caracter ch egal cu $a[i]$ din secvența $a[p...i]$ atunci ratăm construcția unei anagrame.

Caracterul ch nu trebuie selectat neapărat de la pozitia i , el poate fi selectat de oriunde apare în secvența $a[p...i]$. Astfel avem libertatea ca înainte de a selecta ch să selectam orice alt caracter strict mai mic decât ch , de care este nevoie pentru construcția anagramei minime.

Vom construi o parte a anagramei, formată din caractere mai mici decât ch și poziționate la stânga poziției de unde vom selecta ch .

Să presupunem că literele care sunt mai mici decât ch , poziționate înainte de ch și de care mai este nevoie sunt $c_1 < c_2 < \dots < c_j$.

Pentru a obține minimul lexicografic caracterele selectate mai mici decât ch vor trebui să formeze un sir crescător (crescător deoarece, în caz contrar mai bine renunțăm la caracterul care strică ordinea și obținem un sir lexicographic mai mic; putem renunța deoarece caracterul nu este critic, doar $a[i]$ este critic) și procedăm astfel încât:

- c_1 să apară de număr maxim de ori în $[p...i - 1]$; fie p_1 ultima poziție unde apare c_1
- c_2 să apară de număr maxim de ori în secvența $[p_1 + 1...i - 1]$; fie p_2 ultima apariție a lui c_2
- c_3 să apară de număr maxim de ori în secvența $[p_2 + 1...i - 1]$; fie p_3 ultima apariție a lui c_3

Se continuă analog până la c_j ; fie p_j ultima apariție selectată a lui c_j ; apoi se selectează primul character ch egal cu $a[i]$ din $[p_j + 1...i]$; fie p_{ch} poziția selectată.

La etapa 2 se va proceda analog dar relativ la secvența $[p_{ch} + 1...q]$.

Se continuă până la construcția anagramei.

La fiecare etapă vom avea câte un character ch "critic" (care trebuie neapărat selectat). În anagramă optimă, în stânga oricărui caracter "critic" vom avea sau subșir vid, sau subșir crescător format cu caractere mai mici decât caracterul "critic" respectiv, limitat la stânga de caracterul "critic" precedent.

20.1.2 *Rezolvare detaliată

20.1.3 Cod sursă

Listing 20.1.1: adrian-100.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <algorithm>
5 #include <cassert>
6 #include <stack>
7
8 using namespace std;
9
10 int to_index(char c)
11 {
12     if (c >= 'a' && c <= 'z')
13         return c - 'a';
14     if (c >= 'A' && c <= 'Z')
15         return c - 'A' + 26;
16     assert(false);
17 }
18
19 int main()
20 {
21     ifstream cin("anagramme.in");
22     ofstream cout("anagramme.out");
23
24     int type; assert(cin >> type);
25     string S, P; assert(cin >> S >> P);
26
27     assert(1 <= S.size() && S.size() <= 1000 * 1000);
28     assert(1 <= P.size() && P.size() <= S.size());
29
30     for (auto &c : S)
31         assert(('a' <= c && c <= 'z') || ('A' <= c && c <= 'Z'));

```

```

32     for (auto &c : P)
33         assert((('a' <= c && c <= 'z') || ('A' <= c && c <= 'Z')));
34
35     vector<int> need(52, 0);
36     for (auto &c : P)
37         ++need[to_index(c)];
38
39     int total = P.size();
40
41     vector<int> where;
42     for (int i = S.size() - 1; i >= 0; --i)
43     {
44         if (need[to_index(S[i])] > 0)
45         {
46             --need[to_index(S[i])];
47             if (--total == 0)
48             {
49                 where.push_back(i);
50                 total = P.size();
51                 for (auto &c : P)
52                     ++need[to_index(c)];
53             }
54         }
55     }
56
57     if (type == 1)
58     {
59         cout << where.size() << "\n";
60         return 0;
61     }
62
63     reverse(where.begin(), where.end());
64     where.push_back(S.size());
65     where.erase(where.begin());
66
67     int last = 0;
68     for (auto &c : P)
69         need[to_index(c)] = 0;
70
71     vector<char> st;
72     vector<int> have(52, 0);
73     for (int i = 0; i < int(where.size()); ++i)
74     {
75         for (auto &c : P)
76             have[to_index(c)] = need[to_index(c)] = 0;
77         for (int j = last; j < where[i]; ++j)
78             ++have[to_index(S[j])];
79         for (auto &c : P)
80             ++need[to_index(c)];
81
82         st.clear();
83         for (int j = last; j < where[i]; ++j)
84         {
85             if (!need[to_index(S[j])])
86             {
87                 --have[to_index(S[j])];
88                 continue;
89             }
90
91             // i need it, and its better
92             while (!st.empty() && st.back() > S[j] &&
93                     need[to_index(st.back())] < have[to_index(st.back())])
94             {
95                 need[to_index(st.back())]++;
96                 st.pop_back();
97             }
98             st.push_back(S[j]);
99
100            need[to_index(st.back())]--;
101            have[to_index(st.back())]--;
102        }
103
104        for (auto &c : st)
105        {
106            while (S[last] != c)
107            {

```

```

108         ++last;
109     }
110     ++last;
111     cout << c;
112 }
113 cout << "\n";
115 }
```

Listing 20.1.2: anagrame_chiriac_back.cpp

```

1 #include <iostream>
2 #include <cstring>
3 #include <algorithm>
4
5 using namespace std;
6
7 const int Nmax = 1000005;
8
9 char a[Nmax], b[Nmax];
10 int n, m, maxnr[Nmax], need[26], have[26], nr_chars, have_chars, ans;
11
12 int getPos(int pos, char b[])
13 {
14     int cur = 1;
15     while(pos <= n && cur <= m)
16     {
17         if(a[pos] == b[cur])
18             ++cur;
19         if(cur == m + 1)
20             return pos + 1;
21         ++pos;
22     }
23     return -1;
24 }
25
26 int main()
27 {
28     ifstream cin("anagrame.in");
29     ofstream cout("anagrame.out");
30
31     cin >> a + 1 >> b + 1;
32     n = strlen(a + 1);
33     m = strlen(b + 1);
34
35     for(int i = 1; i <= m; i++)
36         if(++need[b[i] - 'a'] == 1)
37             ++nr_chars;
38
39     for(int i = n; i; i--)
40     {
41         if(++have[a[i] - 'a'] == need[a[i] - 'a'])
42             ++have_chars;
43         if(have_chars == nr_chars)
44         {
45             ++ans;
46             for(int i = 0; i < 26; i++)
47                 have[i] = 0;
48             have_chars = 0;
49         }
50         maxnr[i] = ans;
51     }
52
53     cout << ans << "\n";
54
55     int pos = 1, nr_left = ans;
56     while(pos <= n)
57     {
58         sort(b + 1, b + m + 1);
59         do
60         {
61             int x = getPos(pos, b);
62             if(x == -1 || maxnr[x] != nr_left - 1)
63                 continue;
64             for(int i = 1; i <= m; i++)
```

```

65         cout << b[i];
66         pos = x;
67         --nr_left;
68         break;
69     }
70     while(next_permutation(b + 1, b + m + 1));
71 }
72 cout << "\n";
73 return 0;
74 }
```

Listing 20.1.3: anagrame_cos1.cpp

```

1 #include <vector>
2 #include <string>
3 #include <fstream>
4
5 using namespace std;
6
7 struct cos1
8 {
9
10 bool haveAnagram(vector<int> &fStart, vector<int> &fEnd, vector<int> &f)
11 {
12     for (int i = 0; i < f.size(); i++)
13     {
14         if (fStart[i] - fEnd[i] < f[i]) return false;
15     }
16     return true;
17 }
18
19 int solve()
20 {
21     ifstream      f("anagrame.in");
22     ofstream      g("anagrame.out");
23
24     int           P;
25     string        A, B;
26     f >> P >> A >> B;
27
28     vector<vector<int>> pos(26);
29     vector<int>          freqB(26);
30     vector<vector<int>> freq(A.size() + 1, vector<int>(26));
31     vector<int>          len(A.size() + 1);
32
33     string          ans = "";
34
35     for (char c : B) freqB[c - 'a']++;
36
37     int last = (int)A.size();
38     len[A.size()] = 0;
39     for (int i = (int)A.size() - 1; i >= 0; i--)
40     {
41         pos[A[i] - 'a'].push_back(i);
42         freq[i] = freq[i + 1]; freq[i][A[i] - 'a']++;
43         if (haveAnagram(freq[i], freq[last], freqB))
44         {
45             len[i] = 1 + len[i + 1];
46             last = i;
47         }
48     else
49         len[i] = len[last];
50     }
51
52     for (int cnt = len[0], i = 0, j = 0; cnt; cnt--)
53     {
54         vector<int> f = freqB;
55
56         for (; j < A.size() && len[j] >= cnt - 1; j++);
57         if (j != A.size()) j--;
58
59         for (int steps = 0; steps < B.size(); steps++)
60         {
61             for (int k = 0; k < freqB.size(); k++)
62             {
```

```

63         if (!f[k]) continue;
64         int p = *pos[k].rbegin();
65         if (p >= j) continue;
66
67         if (haveAnagram(freq[p], freq[j], f))
68         {
69             i = p; break;
70         }
71     }
72
73     ans.push_back(A[i]);
74     f[A[i] - 'a']--;
75     i++;
76
77     for (int k = 0; k < freqB.size(); k++)
78     {
79         while (!pos[k].empty() && *pos[k].rbegin() < i)
80         {
81             pos[k].pop_back();
82         }
83     }
84 }
85
86 if (P==2)
87     g << ans << "\n";
88 else
89     g << (int)ans.size() / (int)B.size() << "\n";
90
91 return 0;
92 }
93
94 } a1;
95
96 int main()
97 {
98     a1.solve();
99     return 0;
100 }
```

Listing 20.1.4: anagrame_cos2.cpp

```

1 #include <vector>
2 #include <string>
3 #include <fstream>
4
5 using namespace std;
6
7 struct cosmin2
8 {
9
10     int solve()
11     {
12         ifstream      f("anagrame.in");
13         ofstream      g("anagrame.out");
14         int          P;
15         string       A, B; f >> P >> A >> B;
16         vector<int>  len(A.size() + 1);
17         vector<int>  freq(26);
18         string       ans = "";
19
20         // compute the frequencies for the second string
21
22         for (char c : B) freq[c - 'a']++;
23
24         // compute the maximum number of anagrams for any suffix
25         // of the first string
26         // we are interested only in the right most anagrams
27
28         auto crtFreq    = freq;
29         auto crtLetters = B.size();
30         len[A.size()]   = 0;
31
32         for (int i = (int)A.size() - 1; i >= 0; i--)
33         {
```

```

34         int k = A[i] - 'a';
35         len[i] = len[i + 1];
36         if (crtFreq[k])
37         {
38             crtFreq[k]--;
39             if (crtLetters == 0)
40             {
41                 len[i]++;
42                 crtFreq = freq;
43                 crtLetters = B.size();
44             }
45         }
46         // find the min lexicographically solution
47
48     for (int cnt = len[0], start = 0, end; cnt; cnt--)
49     {
50         // find the range for the current anagram -- [p, q]
51
52         for (end = start; end < A.size() && len[end] >= cnt - 1; end++);
53         if (end < A.size()) end--;
54
55         // compute the frequencies for the [p, q) range
56
57         vector<int> rangeFreq(26);
58         for (int i = start; i < end; i++) rangeFreq[A[i] - 'a']++;
59
60         // start constructing the anagram
61         // we are interested in finding the min anagram,
62         // and from these the left most one
63
64         auto crtFreq = freq;
65         auto crtLetters = B.size();
66         string anagram = "";
67         int lastUsed = -1;
68
69         for (int i = start; i < end; i++)
70         {
71             // check if we can make use of this letter
72             if (!crtFreq[A[i] - 'a']) {
73                 rangeFreq[A[i] - 'a']--;
74                 continue;
75             }
76
77             // as long as we can improve our anagram and
78             // we still have enough
79             // letters to get the full anagram in the end,
80             // keep improve it
81
82             while (anagram.size() && A[i] < *anagram.rbegin() &&
83                   crtFreq[*anagram.rbegin() - 'a'] + 1 <=
84                   rangeFreq[*anagram.rbegin() - 'a'])
85             {
86                 // throw away the letter and mark it as desired
87                 // for the future
88
89                 crtFreq[*anagram.rbegin() - 'a']++;
90                 anagram.pop_back();
91             }
92
93             // add the current letter and remove one occurrence from
94             // the frequency tables
95
96             anagram.push_back(A[i]);
97             crtFreq[A[i] - 'a']--;
98             crtLetters--;
99             lastUsed = i;
100            rangeFreq[A[i] - 'a']--;
101        }
102
103        // append the anagram to the solution
104
105        ans.append(anagram);
106
107        start = lastUsed + 1;
108    }
109    if (P==2)

```

```

110         g << ans << "\n";
111     else
112         g << (int)ans.size() / (int)B.size() << "\n";
113
114     return 0;
115 }
116
117 } cos2;
118
119 int main()
120 {
121     cos2.solve();
122     return 0;
123 }
```

Listing 20.1.5: anagrame2.cpp

```

1 //cel mai mic subsir multiplu anagramat
2 #include <iostream>
3 #include <cstring>
4
5 using namespace std;
6
7 struct vasi2
8 {
9
10 char S1[100005], S2[100005], S3[100005];
11 int P, L1, L2, L3, pm[100005], np, ok, qm[100005], nq, k;
12
13 void nrmax(char a[], char b[], int pm[], int &np, int &ok)
14 {
15     int w=0, x, fb[30]={0}, i;
16
17     for(i=0;b[i];i++)
18     {
19         x=b[i]-'a';
20         if(fb[x]==1)w++;
21     }
22
23     for(i=pm[np]-1;i>=0;i--)
24     {
25         x=a[i]-'a';
26         fb[x]--;
27         if(fb[x]==0)
28         {
29             w--;
30             if(w==0)
31             {
32                 np++; pm[np]=i;
33                 return;
34             }
35         }
36     }
37
38     ok=0;
39 }
40
41 void best(char a[], char b[], int qm[], int &nq, char c[], int &k)
42 {
43     int fa[30]={0}, fb[30]={0}, fc[30]={0}, i, r, j, l, x, y, p, q;
44
45     q=qm[nq];
46     p=pm[nq-1];
47
48     for(i=q;i<p;i++)
49     {
50         x=a[i]-'a';
51         fa[x]++;
52     }
53     for(i=0;b[i];i++)
54     {
55         x=b[i]-'a';
56         fb[x]++;
57     }
58
59     r=q;
60     for(i=q;i<p;i++)
61     {
62         x=c[i]-'a';
63         fc[x]++;
64     }
65
66     for(i=q;i<p;i++)
67     {
68         x=a[i]-'a';
69         fa[x]--;
70     }
71
72     for(i=0;b[i];i++)
73     {
74         x=b[i]-'a';
75         fb[x]--;
76     }
77
78     for(i=q;i<p;i++)
79     {
80         x=c[i]-'a';
81         fc[x]--;
82     }
83
84     for(i=q;i<p;i++)
85     {
86         x=a[i]-'a';
87         fa[x]++;
88     }
89
90     for(i=0;b[i];i++)
91     {
92         x=b[i]-'a';
93         fb[x]++;
94     }
95
96     for(i=q;i<p;i++)
97     {
98         x=c[i]-'a';
99         fc[x]++;
100    }
101 }
```

```

59         x=a[i]-'a';
60         fa[x]--;
61
62         if(fa[x]<fb[x] && fb[x]>0)
63         {
64             ///a[i] este critic
65
66             for(j=r; j<i; j++)
67             {
68                 ///frecventez toate caracterele din [r;i-1] care sunt
69                 ///mai mici decat a[i]
70                 ///si de care mai este nevoie
71
72                 y=a[j]-'a';
73                 if(y<x && fb[y]>0) {
74                     fc[y]++;
75                 }
76             }
77
78             ///selectez caracterul minim dintre cele <a[i] de pe toate
79             ///pozitiile din [r;i-1]
80             ///si actualizez r
81             ///apoi selectez caracterul minim dintre cele <a[i] de pe toate
82             ///pozitiile din [r;i-1]
83             ///actualizez r, etc
84
85             for(j=0; j<x; j++)
86             {
87                 while(fc[j]>0 && fb[j]>0 && r<i)
88                 {
89                     y=a[r]-'a';
90                     if(y==j) {
91                         c[k]=j+'a'; k++; fb[j]--;
92                     }
93                     if(y<x) fc[y]--;
94                     r++;
95                 }
96             }
97
98             ///selectez prima aparitie j a valorii a[j]==a[i] din [r;i]
99
100            for(j=r; j<=i; j++)
101            {
102                if(a[j]==='a'+x)
103                {
104                    break;
105                }
106            }
107
108            c[k]=a[j]; k++; fb[x]--;
109            r=j+1; ///actualizez r
110
111            ///refac fa[] pe intervalul [j+1;i]
112
113            for(l=j+1; l<=i; l++)
114            {
115                fa[a[l]-'a']++;
116            }
117            i=j;
118        }
119    }
120
121    c[k]=0;
122    nq--; qm[nq]=r;
123 }
124
125 int solve()
126 {
127     ifstream fin("anagram.in");
128     ofstream fout("anagram.out");
129
130     fin>>P>>S1>>S2;
131     L1=strlen(S1);
132     L2=strlen(S2);
133
134     ///pastrez in pm[] pozitiile critice

```

```

135
136     np=1;
137     pm[1]=L1;
138     ok=1;
139     nrmax(S1,S2,pm,np,ok);
140     while(ok)
141     {
142         nrmax(S1,S2,pm,np,ok);
143     }
144
145 //ultima anagrama posibila incepe la pozitia pm[2]
146 //penultima anagrama incepe la pm[3], etc
147
148     qm[np]=0;
149     nq=np;
150     k=0;
151     while(nq>1)
152     {
153         //caut cea mai mica anagrama si care se termina cel mai la stanga
154         //in secventa [ qm[nq] ; pm[nq-1]-1 ]
155         //daca se termina la pozitia i, atunci vom avea qm[nq-1]=i+1;
156
157         best(S1,S2,qm,nq,S3,k);
158     }
159
160     L3=strlen(S3);
161
162     if(P==1)
163         fout<<L3/L2<<"\n";
164     else
165         fout<<S3<<"\n";
166
167     fout.close();
168     fin.close();
169     return 0;
170 }
171
172 } va2;
173
174 int main()
175 {
176     va2.solve();
177     return 0;
178 }
```

Listing 20.1.6: anagrame3.cpp

```

1 //cel mai mic subsir multiplu anagramat
2 #include <fstream>
3 #include <cstring>
4
5 using namespace std;
6
7 struct priv2
8 {
9     char S1[100005], S2[100005], S3[100005];
10    int P, L1, L2, L3, pm[100005], np, ok, qm[100005], nq, k, i, fb[26];
11
12    void nrmax(char a[], char b[], int pm[], int &np, int &ok)
13    {
14        int w=0, x, fb[30]={0}, i;
15        for(i=0;b[i];i++)
16        {
17            x=b[i]-'a'; fb[x]++;
18            if(fb[x]==1)w++;
19        }
20
21        for(i=pm[np]-1;i>=0;i--)
22        {
23            x=a[i]-'a';
24            fb[x]--;
25            if(fb[x]==0)
26            {
27                w--;
28                if(w==0)
```

```

29             {
30                 np++;
31                 pm[np]=i;
32                 return;
33             }
34         }
35     }
36
37     ok=0;
38 }
39
40 int best(char a[], char b[], int nb, char c[], int &nc, int p, int q)
41 {
42     int i,nr=0,x[126],xb[126],xa[126],r;
43     char j;
44
45     for(i=0;i<=125;i++) {xb[i]=0; xa[i]=0; x[i]=0;}
46     for(i=p;i<=q;i++) {xa[a[i]]++;}
47     for(i=0;i<=nb-1;i++) {xb[b[i]]++;}
48
49     nr=nb;
50     r=p;
51     i=p-1;///folosim doi pointeri r si i, r<=i
52
53     ///x[] numarul de caractere din secventa [r...i]
54     ///xa[] numarul de caractere din secventa [r...q]
55     ///xb[] numarul de caractere de care mai este nevoie
56     ///pentru completarea unei anagrame
57
58     while(nr>0)
59     {
60         ///deplasam pointerul i
61
62         do
63         {
64             i++;
65             x[a[i]]++;
66         }
67         while(i<=q && xb[a[i]]+x[a[i]]<=xa[a[i]]);
68
69         ///un caracter egal cu a[i] trebuie neaparat preluat, din a[p...i]
70         ///in caz contrar, in secventa a[i+1...q] nu mai raman suficiente
71         ///caractere egale cu a[i]
72
73         ///preluam un subsir crescator, maximal ca lungime, format din
74         ///caracteri mai mici decat a[i]
75         ///si necesare anagramei
76
77         for(j='a';j<a[i];j++)
78         {
79             while(x[j]>0 && xb[j]>0)
80             {
81                 if(a[r]==j)
82                 {
83                     xb[j]--;nr--;
84                     c[nc++]=a[r];
85                 }
86
87                 x[a[r]]--;
88                 xa[a[r]]--;
89                 r++;
90             }
91         }
92
93         ///caut primul caracter din r...i egal cu a[i]
94
95         while(a[r]!=a[i])
96         {
97             x[a[r]]--;
98             xa[a[r]]--;
99             r++;
100        }
101
102        x[a[r]]--;
103        xa[a[r]]--;
104        xb[a[r]]--;

```

```

105         nr--;
106         c[nc++]=a[r];
107         r++;
108     }
109
110     return r;
111 }
112
113 int solve()
114 {
115     ifstream fin("anagrame.in");
116     ofstream fout("anagrame.out");
117
118     fin>>P>>S1>>S2;
119     L1=strlen(S1);
120     L2=strlen(S2);
121
122     //calculez frecventa caracterelor din S2[]
123
124     for(i=0;i<L2;i++)
125     {
126         fb[S2[i]-'a']++;
127     }
128
129     //pastreaz din S1 doar caracterele care apar in S2
130
131     k=0;
132     for(i=0;i<L1;i++)
133     {
134         if(fb[S1[i]-'a']>0)
135         {
136             S1[k++]=S1[i];
137         }
138     }
139
140     S1[k]=0;
141     L1=k;
142
143     //pastrez in pm[] pozitiile critice
144
145     np=0;
146     pm[0]=L1;
147     ok=1;
148     nrmax(S1,S2,pm,np,ok);
149     while(ok)
150     {
151         nrmax(S1,S2,pm,np,ok);
152     }
153
154     //oglindesc sirul pm[]
155
156     for(i=0;i<=np/2;i++)
157     {
158         swap(pm[i],pm[np-i]);
159     }
160
161     qm[1]=0;
162     nq=1;
163     L3=0;
164     while(nq<=np)
165     {
166         ///caut cea mai mica anagrama, care se termina cel mai la stanga
167         ///in secventa [ qm[nq] ; pm[nq]-1 ]
168         ///daca se termina la pozitia i, atunci vom avea qm[nq+1]=i+1;
169
170         qm[nq+1]=best(S1,S2,L2,S3,L3,qm[nq],pm[nq]-1);
171         nq++;
172     }
173
174     S3[L3]=0;
175     //fout<<L3/L2<<"\n";
176     if(P==2)
177         fout<<S3<<"\n";
178     else
179         fout<<np<<"\n";
180

```

```

181         fout.close();
182         fin.close();
183         return 0;
184     }
185 }
186 } va2;
187
188 int main()
189 {
190     va2.solve();
191     return 0;
192 }
```

Listing 20.1.7: anagrame4.cpp

```

1 //cel mai mic subsir multiplu anagramat
2 #include <fstream>
3 #include <cstring>
4
5 using namespace std;
6
7 struct priv2
8 {
9
10    char S1[100005], S2[100005], S3[100005];
11    int P, L1, L2, L3, pm[100005], np, ok, qm[100005], nq, k, i,
12        fb[126], xa[126], x[126];
13
14    void nrmax(char a[], char b[], int pm[], int &np, int &ok)
15    {
16        int w=0, x, fb[30]={0}, i;
17        for(i=0;b[i];i++)
18        {
19            x=b[i]-'a';
20            fb[x]++;
21            if(fb[x]==1) w++;
22        }
23        for(i=pm[np]-1;i>=0;i--)
24        {
25            x=a[i]-'a';
26            fb[x]--;
27            if(fb[x]==0)
28            {
29                w--;
30                if(w==0)
31                {
32                    np++;
33                    pm[np]=i;
34                    return;
35                }
36            }
37        }
38        ok=0;
39    }
40
41    int best(char a[], char b[], int nb, char c[], int &nc, int p,int q,int s)
42    {
43        int i,nr=0,xb[126],r,k;
44        char j;
45
46        for(i=s;i<=q;i++) {xa[a[i]]++;}
47        for(i='a';i<='z';i++) {xb[i]=fb[i];}
48        nr=nb;
49        r=p;
50        i=p-1;///folosim doi pointeri r si i, r<=i
51
52        ///x[] numarul de caractere din secheta [r...i]
53        ///xa[] numarul de caractere din secheta [r...q]
54        ///xb[] numarul de caractere de care mai este nevoie pentru
55        ///completarea unei anagrame
56
57        while(nr>0)
58        {
59            ///deplasam pointerul i
```

```

61         do
62     {
63         i++;
64         x[a[i]]++;
65     }
66     while(i<=q && xb[a[i]]+x[a[i]]<=xa[a[i]]);
67
68     //un caracter egal cu a[i] trebuie neaparat preluat, din a[p...i]
69     //in caz contrar, in secheta a[i+1...q] nu mai raman suficiente
70     //caracteri egale cu a[i]
71
72     //preluam un subsir crescator, maximal ca lungime, format din
73     //caracteri mai mici decat a[i]
74     //si necesare anagramei
75
76     for(j='a'; j<a[i]; j++)
77     {
78         while(x[j]>0 && xb[j]>0)
79         {
80             if(a[r]==j)
81             {
82                 xb[j]--;
83                 c[nc++]=a[r];
84             }
85
86             x[a[r]]--;
87             xa[a[r]]--;
88             r++;
89         }
90     }
91
92     //caut primul caracter din r...i egal cu a[i]
93     while(a[r]!=a[i])
94     {
95         x[a[r]]--;
96         xa[a[r]]--;
97         r++;
98     }
99
100    x[a[r]]--;
101    xa[a[r]]--;
102    xb[a[r]]--;
103    nr--;
104    c[nc++]=a[r];
105    r++;
106}
107
108 for(k=r; k<=i; k++)
109 {
110     x[a[k]]--;
111 }
112
113 return r;
114}
115
116 int solve()
117{
118     ifstream fin("anagrame.in");
119     ofstream fout("anagrame.out");
120
121     fin>>P>>S1>>S2;
122     L1=strlen(S1);
123     L2=strlen(S2);
124
125     //calculez frecventa caracterelor din S2[]
126
127     for(i=0; i<L2; i++)
128     {
129         fb[S2[i]]++;
130     }
131
132     //pastreaz din S1 doar caracterele care apar in S2
133
134     k=0;
135     for(i=0; i<L1; i++)
136     {

```

```

137         if(fb[S1[i]]>0)
138     {
139         S1[k++]=S1[i];
140     }
141 }
142
143 S1[k]=0;
144 L1=k;
145
146 //pastrez in pm[] pozitiile critice
147
148 np=0;
149 pm[0]=L1;
150 ok=1;
151 nrmax(S1,S2,pm,np,ok);
152 while(ok)
153 {
154     nrmax(S1,S2,pm,np,ok);
155 }
156
157 //oglindesc sirul pm[]
158
159 for(i=0;i<=np/2;i++)
160 {
161     swap(pm[i],pm[np-i]);
162 }
163
164 pm[0]=0;
165 qm[1]=0;
166 nq=1;
167 L3=0;
168 while(nq<=np)
169 {
170     //caut cea mai mica anagrama, care se termina cel mai la stanga
171     //in secheta [ qm[nq] ; pm[nq]-1 ]
172     //daca se termina la pozitia i, atunci vom avea qm[nq+1]=i+1;
173
174     qm[nq+1]=best(S1,S2,L2,S3,L3,qm[nq],pm[nq]-1,pm[nq-1]);
175     nq++;
176 }
177
178 S3[L3]=0;
179 if(P==1)
180     fout<<np<<"\n";
181 else
182     fout<<S3<<"\n";
183
184 fout.close();
185 fin.close();
186 return 0;
187 }
188
189 } va2;
190
191 int main()
192 {
193     va2.solve();
194     return 0;
195 }
```

Listing 20.1.8: anagrame-turturica.cpp

```

1 /**
2 * Turturica Razvan
3 * anagrame
4 * 100p
5 */
6 #include <fstream>
7 #include <cstring>
8
9 using namespace std;
10
11 ifstream fin( "anagrame.in" );
12 ofstream fout( "anagrame.out" );
13
```

```

14  char s1[100010],s2[100010],ans[100010];
15  int a,b,c,i,j,n,m,ansSize,original[26],v[26],cerinta,stopPoints[100010],
16      k,ns,lastPoz,fv[26],st[100010];
17
18 int main()
19 {
20     fin>>cerinta;
21     fin.get();
22     fin.get( s1 + 1 , 100010 );
23     fin.get();
24     fin.get( s2 + 1 , 100010 );
25
26     n = strlen( s1 + 1 );
27     m = strlen( s2 + 1 );
28
29     for( i = 1 ; i <= m ; i++ )
30     {
31         original[ s2[ i ] - 'a' ]++;
32         v[ s2[ i ] - 'a' ]++;
33     }
34
35     k = m;
36
37     for( i = n ; i >= 1 ; i-- )
38     {
39         if( v[ s1[ i ] - 'a' ] )
40         {
41             k--;
42             v[ s1[ i ] - 'a' ]--;
43         }
44         if( k == 0 )
45         {
46             stopPoints[ ++ns ] = i;
47             k = m;
48             for( j = 0 ; j < 26 ; j++ )
49                 v[ j ] = original[ j ];
50         }
51     }
52
53     stopPoints[ 0 ] = n + 1;
54     ns--;
55     lastPoz = 1;
56
57     for( i = ns ; i >= 0 ; i-- )
58     {
59         st[ 0 ] = 0;
60         for( j = lastPoz ; j < stopPoints[ i ] ; j++ )
61         {
62             fv[ s1[ j ] - 'a' ]++;
63         }
64
65         for( j = 0 ; j < 26 ; j++ )
66             v[ j ] = original[ j ];
67
68         for( j = lastPoz ; j < stopPoints[ i ] ; j++ )
69         {
70             if( v[ s1[ j ] - 'a' ] )
71             {
72                 while( st[ 0 ] && s1[ j ] < st[ st[ 0 ] ] &&
73                         v[ st[ st[ 0 ] ] - 'a' ] < fv[ st[ st[ 0 ] ] - 'a' ] )
74                 {
75                     v[ st[ st[ 0 ] ] - 'a' ]++;
76                     st[ 0 ]--;
77                 }
78
79                 st[ ++st[ 0 ] ] = s1[ j ];
80                 fv[ st[ st[ 0 ] ] - 'a' ]--;
81                 v[ st[ st[ 0 ] ] - 'a' ]--;
82             }
83             else
84             {
85                 fv[ s1[ j ] - 'a' ]--;
86             }
87         }
88
89         for( j = 1 ; j <= st[ 0 ] ; j++ )

```

```

90         {
91             ans[ ++ansSize ] = st[ j ];
92         }
93
94         for( j = 1 ; j <= st[ 0 ] ; j++ )
95         {
96             while( s1[ lastPoz ] != st[ j ] )
97                 lastPoz++;
98             lastPoz++;
99         }
100    }
101
102    if( cerinta == 1 )
103        fout<<ansSize / m;
104    else
105        fout<<( ans + 1 );
106
107    return 0;
108 }
```

20.2 multimi

Problema 2 - multimi

100 de puncte

O mulțime cu elemente numere naturale poate fi scrisă într-o *formă redusă* dacă, ordonând crescător elementele ei, diferența dintre oricare două valori alăturate este aceeași. De exemplu, mulțimea $D = \{11, 14, 17, 20, 23\}$ poate fi scrisă sub forma $D = 11 - 23/3$, precizând elementul minim, elementul maxim și diferența dintre elemente.

Date fiind N mulțimi scrise sub forma redusă, fiecare fiind notată cu o literă mare a alfabetului englez, se cere să se calculeze o expresie care poate conține:

- operația de reuniune, notată cu $+$;
- operația de intersecție, notată cu $*$;
- literele asociate mulțimilor;
- paranteze rotunde.

Considerăm că *valoarea expresiei* este mulțimea obținută după efectuarea operațiilor specifice mulțimilor considerând că operațiile de intersecție au prioritate mai mare decât cele de reuniune.

Cerințe

Cunoscând forma redusă a celor N mulțimi și o expresie, să se calculeze valoarea expresiei date.

Date de intrare

Datele de intrare se citesc din fișierul **multimi.in**, care are următoarea structură:

- Pe prima linie se află numărul natural N , reprezentând numărul mulțimilor;
- Pe următoarele N linii se află formele reduse ale celor N mulțimi, câte o mulțime pe fiecare linie;
- Pe linia $N + 2$ se află expresia ce trebuie calculată.

Date de ieșire

Datele de ieșire se vor scrie în fișierul **multimi.out**, astfel:

- Pe prima linie se va scrie numărul elementelor mulțimii obținute în urma evalării expresiei date;
- Pe linia a doua se vor scrie, în ordine crescătoare, elementele mulțimii respective, separate prin câte un spațiu.

Restricții și precizări

- $1 < N \leq 16$
- Elementele mulțimilor sunt numere naturale cuprinse între 0 și 1 000 000 000;
- Numărul elementelor unei mulțimi este maximum 10 000;
- Numărul caracterelor expresiei este cuprins între 3 și 1 000;
- Forma redusă a unei mulțimi și expresia dată nu conțin spații;

- Se garantează că, pentru toate datele de test, valoarea expresiei nu poate fi mulțimea vidă;
- Se garantează că, în teste care totalizează 30 de puncte, expresia nu conține paranteze;
- Se garantează că, în teste care totalizează 60 de puncte, cardinalul fiecărei mulțimi date la intrare nu depășește valoarea 1 000;

Exemple:

multimi.in	multimi.out	Explicații
3 A=2-8/2 C=11-23/3 B=4-16/4 A*(B+C)	2 4 8	Mulțimile au următoarele elemente: A={2, 4, 6, 8} B={4, 8, 12, 16} C={11, 14, 17, 20, 23} Efecțuând operațiile obținem: B+C = {4, 8, 11, 12, 14, 16, 17, 20, 23} și A*(B+C) = {4, 8}
3 A=2-7/1 B=1-5/1 C=3-9/3 B*A+A*C	5 2 3 4 5 6	Mulțimile au următoarele elemente: A={2, 3, 4, 5, 6, 7} B={1, 2, 3, 4, 5} C={3, 6, 9} Efecțuând operațiile obținem: B*A = {2, 3, 4, 5}, A*C = {3, 6} B*A+A*C = {2, 3, 4, 5, 6}

Timp maxim de executare/test: **3.0** secunde sub Windows, **1.0** secunde sub Linux

Memorie: total **128 MB** din care pentru stivă **32 MB**

Dimensiune maximă a sursei: **20 KB**

20.2.1 Indicații de rezolvare

prof. Alin Burța - C.N. "B.P. Hasdeu" Buzău

Soluția 1 (60 puncte)

Fiecare mulțime specificată prin forma redusă va fi "expandată" prin memorarea tuturor elementelor sale, în ordine crescătoare. Parcurgem expresia dată caracter cu caracter și o evaluăm cu ajutorul unei stive, astfel:

Cazul 1 Caracterul curent este litera corespunzătoare unei mulțimi.

- Dacă operația anterioară este o intersecție, efectuăm operația și înlocuim mulțimea din vârful stivei cu mulțimea rezultat (ne asigurăm că efectuăm cu prioritate operațiile de intersecție).

- Dacă operația anterioară este o reuniune, vom efectua operația doar dacă următoarea operație este tot o reuniune, altfel introducem mulțimea curentă în stivă, pentru evaluare ulterioară, memorând și operația rămasă neefectuată.

Cazul 2 Caracterul curent este o paranteză deschisă.

Marcăm apariția acestei paranteze în stivă, deoarece operațiile pe care le conține vor fi evaluate ulterior.

Cazul 3 Caracterul curent este o paranteză închisă.

Este momentul să realizăm toate operațiile rămase neefectuate (de regulă, doar reuniuni), până la prima paranteză deschisă marcată anterior în stivă.

Cazul 4 Parcurgerea expresiei s-a încheiat, dar stiva mai conține operații neefectuate.

Efectuăm, pe rând, operațiile rămase până la obținerea unei singure mulțimi, care va constitui rezultatul evaluării.

Operațiile de reuniune și intersecție se pot efectua în diverse moduri, precum utilizarea algoritmului de interclasare (elementele mulțimilor obținute pe parcurs sunt ordonate crescător).

Dezavantajul acestei metode este cantitatea mare de memorie necesară.

Soluția 2 (100 puncte)

Fie expresia $A^*(B+C)$ și x un element oarecare al uneia dintre mulțimile A , B , C .

Având în vedere că x poate apartine sau nu mulțimilor A , B , C , determinarea apartenenței lui x la mulțimea rezultat se reduce la a verifica dacă expresia logică a SI (b SAU c) are valoarea 1 (adevărat). Astfel, putem analiza expresia logică asociată obținută din expresia inițială prin

înlocuirea mulțimilor cu propoziții logice, a intersecției cu operația SI LOGIC, respectiv a reuniunii cu operația SAU LOGIC. Vom genera și memora toate posibilitățile de a înlocui propozițiile logice a, b, c cu valori din mulțimea {0, 1} și vom calcula valoarea expresiei logice pentru fiecare caz (sunt 2^n cazuri).

Calculăm reuniunea tuturor mulțimilor date, păstrând elementele acesteia în ordine crescătoare. Pentru fiecare element x al reuniunii, verificăm dacă x aparține mulțimii rezultat, utilizând valorile precalculate ale expresiei logice.

20.2.2 *Rezolvare detaliată

20.2.3 Cod sursă

Listing 20.2.1: multimi_alin_60p.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <cstring>
4
5 #define INFILE "multimi.in"
6 #define OUFILe "multimi.out"
7
8 #define NMAX 1001
9 #define EMAX 16017
10
11 using namespace std;
12
13 typedef struct Multime
14 {
15     int A[EMAX];
16     int card;
17 };
18
19 int N;                                //numarul multimilor
20 Multime M[28];                         //memoreaza multimele date
21 int card[28];                          //cardinalul fiecarei multimi
22 char *oper = new char[1001];           //memoreaza sirul de operatii
23 Multime st[505];
24 bool pr[505]={false};
25 char op[505]={' '};
26
27 ofstream out(OUFILE);
28
29 int strToInt(char *s)                  //transforma un sir intr-un numar intreg
30 {
31     int rez = 0, i;
32     for(i=0; i<strlen(s); ++i)
33         rez = rez * 10 + s[i] - 48;
34     return rez;
35 }
36
37 //transforma multimea din forma restransa (sir) in tablou de intregi
38 void strToSet(char *s, int v[], int &nv)
39 {
40     char *p1, *p2, *p3, aux[100];
41     int val1, val2, pas;
42
43     p1 = strchr(s,'=');
44     p2 = strchr(s,'-');
45     p3 = strchr(s,'/');
46
47     strncpy (aux, p1+1, p2-p1-1); aux[p2-p1-1] = '\0';
48     val1 = strToInt(aux);
49
50     strncpy (aux, p2+1, p3-p2-1); aux[p3-p2-1] = '\0';
51     val2 = strToInt(aux);
52     pas = strToInt(p3+1);
53
54     for(nv=0 ; val1<=val2; val1 += pas)
55         v[++nv] = val1;

```

```

56 }
57
58 //determina indicii elementelor din v intre care se afla x
59 //daca st == dr atunci x se afla in tabloul v
60
61 void BS(int v[], int nv, int x, int &st, int &dr)
62 {
63     int mij;
64     st = 1; dr = nv;
65     while( st <= dr)
66     {
67         mij = (st + dr)/2;
68         if(v[mij] == x)
69         {
70             st = dr = mij;
71             break;
72         }
73         else
74             if(x < v[mij])
75                 dr = mij - 1;
76             else
77                 st = mij + 1;
78     }
79     mij = st, st = dr, dr = mij;
80 }
81
82 //interclaseaza elementele din v1 si ve si pune rezultatul in v1
83
84 void MergeSets(int v1[], int &nv1, int v2[], int nv2 )
85 {
86     int i, j, k, v3[EMAX], nv3;
87     nv3 = 0;
88     i = j = 1;
89     while( i <= nv1 && j <= nv2 )
90         if(v1[i] < v2[j])
91             v3[++nv3] = v1[i++];
92         else
93             if(v1[i] > v2[j])
94                 v3[++nv3] = v2[j++];
95             else
96                 v3[++nv3] = v2[j], i++, j++;
97     while( i <= nv1 ) v3[++nv3] = v1[i++];
98     while( j <= nv2 ) v3[++nv3] = v2[j++];
99     for( i=1; i<=nv3; ++i) v1[i] = v3[i];
100    nv1 = nv3;
101 }
102
103 //intersecteaza doua multimi si pune rezultatul in prima
104
105 void InterSets(int v1[], int &nv1, int v2[], int nv2)
106 {
107     int i, j, k, st, dr;
108     int v3[EMAX], nv3;
109     nv3 = 0;
110     if(nv1 < nv2)
111         for(i = 1; i<=nv1; i++)
112         {
113             BS(v2, nv2, v1[i], st, dr);
114             if(st == dr)
115                 v3[++nv3] = v1[i];
116         }
117     else
118         for(i = 1; i<=nv2; i++)
119         {
120             BS(v1, nv1, v2[i], st, dr);
121             if(st == dr)
122                 v3[++nv3] = v2[i];
123         }
124     for(i = 1; i<=nv3; i++)
125         v1[i] = v3[i];
126     nv1 = nv3;
127 }
128
129 void print(int v[], int nv)      //tipareste un tablou de intregi
130 {

```

```

132     int i;
133
134     out<<nv<<'\\n';
135     for(i=1; i<=nv; i++)
136         out<<v[i]<<' ';
137     out<<'\\n';
138 }
139
140 int main()
141 {
142     cout<<"start"<<endl;
143     int i, lg, vf;
144     char *sir = new char[100];
145     ifstream IN(INFILE);
146     IN>>N;
147     IN.get();
148     for(i=1; i<=N; i++)
149     {
150         IN.get(sir, 100);
151         IN.get();
152         strTOset(sir, M[sir[0]-65].A, M[sir[0]-65].card);
153     }
154     IN.get(oper, 1001);
155     IN.get();
156     IN.close();
157
158 //rezolvare
159 vf = 0;
160 lg = strlen(oper);
161 for(i=0; i<lg; i++)
162 {
163     if('A' <= oper[i] && oper[i] <= 'Z')
164     {
165         st[++vf] = M[oper[i]-65];
166         if(oper[i+1] == '+' || oper[i+1] == '*')
167             op[vf] = oper[i+1];
168     }
169     if(oper[i] == '+' || oper[i] == '*')
170         op[vf] = oper[i];
171     if(oper[i] == '(')
172         pr[vf+1] = true;
173     if(oper[i] == ')')      // efectuez toate operatiile pana la
174                           // paranteza deschisa
175     {
176         while(vf>1 && ! pr[vf])
177             if(op[vf-1] == '*')
178             {
179                 InterSets(st[vf-1].A,st[vf-1].card, st[vf].A,st[vf].card);
180                 vf--;
181                 op[vf]=' ';
182             }
183             else
184                 if(op[vf-1] == '+')
185                 {
186                     MergeSets(st[vf-1].A,st[vf-1].card,st[vf].A,st[vf].card);
187                     vf--;
188                     op[vf]=' ';
189                 }
190
191             if( pr[vf] )
192             {
193                 pr[vf] = false;
194                 op[vf] =' ';
195             }
196     }
197
198 //fac operatiile din varful stivei, daca se poate
199
200     if(vf>1 && ! pr[vf] )    //nu am paranteaza inaintea multimi
201                           //din varful stivei
202     {
203         if(op[vf-1] == '*')
204         {
205             InterSets(st[vf-1].A, st[vf-1].card, st[vf].A, st[vf].card);
206             vf--;
207             op[vf]=' ';

```

```

208         }
209     else
210         if(op[vf-1] == '+' && oper[i+1] == '+')
211         {
212             MergeSets(st[vf-1].A, st[vf-1].card,st[vf].A,st[vf].card);
213             vf--;
214             op[vf]=' ';
215         }
216     }
217 }
218 }
219
220 while(vf>1&& !pr[vf]) //nu am paranteaza inaintea multimii din varful stivei
221 {
222     if(op[vf-1] == '*')
223     {
224         InterSets(st[vf-1].A, st[vf-1].card, st[vf].A, st[vf].card);
225         vf--;
226         op[vf]=' ';
227     }
228     else
229         if(op[vf-1] == '+')
230         {
231             MergeSets(st[vf-1].A, st[vf-1].card, st[vf].A, st[vf].card);
232             vf--;
233             op[vf]=' ';
234         }
235     }
236
237 print(st[1].A, st[1].card);
238 out.close();
239 return 0;
240 }
```

Listing 20.2.2: multimi-assert-100.cpp

```

1 #include <iostream>
2 #include <cstring>
3 #include <cassert>
4
5 using namespace std;
6
7 ifstream fin( "multimi.in" );
8 ofstream fout( "multimi.out" );
9
10 int i,j,n,m,k,a,bits[ (1<<16) ],s[20],f[20],r[20],ev[300],st[10010],
11      val[300],norm[300],ans[16 * 10010],ok,mini,bmsk;
12 char tr,multimea,exp[1010],evexp[1019];
13 int valori[ 20 ];
14
15 int INDEX = 0;
16
17 char top()
18 {
19     return evexp[ INDEX ];
20 }
21
22 char pop()
23 {
24     return evexp[ INDEX ++ ];
25 }
26
27 int solveR();
28
29 int solveP()
30 {
31     if( top() != '(' )
32         return pop()-'0';
33     else if( top() == '(' )
34     {
35         pop();
36         int rez = solveR();
37         pop();
38         return rez;
39     }
}
```

```

40     return 0;
41 }
42
43 int solveI()
44 {
45     int rez = solveP();
46     while( top() == '*' )
47     {
48         pop();
49         rez = min( rez , solveP() );
50     }
51     return rez;
52 }
53
54 int solveR()
55 {
56     int rez = solveI();
57     while( top() == '+' )
58     {
59         pop();
60         rez = max( rez , solveI() );
61     }
62     return rez;
63 }
64
65 int main()
66 {
67     fin>>n;
68     assert( n <= 16 );
69
70     fin.get();
71     for( int i = 1 ; i <= n ; i++ )
72     {
73         fin>>multimea>>tr>>valori[i]>>tr>>f[i]>>tr>>r[i];
74         norm[ multimea ] = 'A' + i - 1;
75         ev[ multimea ] = i;
76         assert( f[ i ] <= 1000000000 );
77     }
78
79     fin.get();
80     fin.get( exp , 10005 );
81     k = strlen( exp );
82
83     assert( k >= 3 );
84     assert( k <= 1000 );
85
86     for( int i = 0 ; i < k ; i++ )
87     {
88         if( exp[ i ] >= 'A' && exp[ i ] <= 'Z' )
89             exp[ i ] = norm[ exp[ i ] ];
90     }
91
92     val[ '+' ] = '+';
93     val[ '*' ] = '*';
94     val[ '(' ] = '(';
95     val[ ')' ] = ')';
96
97     for( int i = 0 ; i < ( 1<<(n) ) ; i++ )
98     {
99         for( int j = 0 ; j < n ; j++ )
100         {
101             val[ 'A' + j ] = '1';
102             if( ( i & (1<<(j)) ) == 0 )
103                 val[ 'A' + j ] = '0';
104         }
105
106         for( int j = 0 ; j < k ; j++ )
107             evexp[ j ] = val[ exp[ j ] ];
108         INDEX = 0;
109         bitmask[ i ] = solveR();
110     }
111
112     ok = 1;
113     while( ok == 1 )
114     {
115         ok = 0;

```

```

116         mini = 1000000009;
117         for( int i = 1 ; i <= n ; i++ )
118         {
119             if( valori[ i ] <= f[ i ] )
120             {
121                 ok = 1;
122                 mini = min( mini , valori[ i ] );
123             }
124         }
125
126         bmsk = 0;
127         for( int i = 1 ; i <= n ; i++ )
128         {
129             if( valori[ i ] == mini && valori[ i ] <= f[ i ] )
130             {
131                 bmsk += ( 1 <<(i-1) );
132                 valori[ i ] += r[ i ];
133             }
134         }
135         if( bitmask[ bmsk ] == 1 )
136         {
137             ans[ ++ans[ 0 ] ] = mini;
138         }
139     }
140
141     fout<<ans[ 0 ]<<' \n';
142     for( int i = 1 ; i <= ans[ 0 ] ; i++ )
143     {
144         fout<<ans[ i ]<<' ';
145     }
146
147     return 0;
148 }
```

Listing 20.2.3: multimi-eficientizat.cpp

```

1 #include <iostream>
2 #include <cstring>
3
4 using namespace std;
5
6 ifstream fin( "multimi.in" );
7 ofstream fout( "multimi.out" );
8
9 int i,j,n,m,k,a,bitmask[ (1<<16) ],s[20],f[20],r[20],ev[300],st[10010],
10    val[300],norm[300],ans[16 * 10010],ok,minim,bmsk;
11 char tr,multimea,exp[10010],evexp[10019];
12 int valori[ 20 ];
13
14 int INDEX = 0;
15
16 char top()
17 {
18     return evexp[ INDEX ];
19 }
20
21 char pop()
22 {
23     return evexp[ INDEX ++ ];
24 }
25
26 int solveR();
27
28 int solveP()
29 {
30     if( top() != '(' )
31         return pop()-'0';
32     else
33         if( top() == '(' )
34         {
35             pop();
36             int rez = solveR();
37             pop();
38             return rez;
39         }

```

```

40     return 0;
41 }
42
43 int solveI()
44 {
45     int rez = solveP();
46     while( top() == '*' )
47     {
48         pop();
49         rez = min( rez , solveP() );
50     }
51     return rez;
52 }
53
54 int solveR()
55 {
56     int rez = solveI();
57     while( top() == '+' )
58     {
59         pop();
60         rez = max( rez , solveI() );
61     }
62     return rez;
63 }
64
65 void eval( int bmsk )
66 {
67     for( int j = 0 ; j < n ; j++ )
68     {
69         val[ 'A' + j ] = '1';
70         if( ( bmsk & (1<<(j)) ) == 0 )
71             val[ 'A' + j ] = '0';
72     }
73
74     for( int j = 0 ; j < k ; j++ )
75         evexp[ j ] = val[ exp[ j ] ];
76     INDEX = 0;
77     bitmask[ bmsk ] = solveR() + 1;
78 }
79
80 int main()
81 {
82     fin>>n;
83     fin.get();
84     for( int i = 1 ; i <= n ; i++ )
85     {
86         fin>>multimea>>tr>>valori[i]>>tr>>f[i]>>tr>>r[i];
87         norm[ multimea ] = 'A' + i - 1;
88         ev[ multimea ] = i;
89     }
90     fin.get();
91     fin.get( exp , 10005 );
92     k = strlen( exp );
93
94     for( int i = 0 ; i < k ; i++ )
95         if( exp[ i ] >= 'A' && exp[ i ] <= 'Z' )
96             exp[ i ] = norm[ exp[ i ] ];
97
98     val[ '+' ] = '+';
99     val[ '*' ] = '*';
100    val[ '(' ] = '(';
101    val[ ')' ] = ')';
102
103    ok = 1;
104    while( ok == 1 )
105    {
106        ok = 0;
107        mini = 1000000009;
108        for( int i = 1 ; i <= n ; i++ )
109        {
110            if( valori[ i ] <= f[ i ] )
111            {
112                ok = 1;
113                mini = min( mini , valori[ i ] );
114            }
115        }

```

```

116     bmsk = 0;
117     for( int i = 1 ; i <= n ; i++ )
118     {
119         if( valori[ i ] == mini && valori[ i ] <= f[ i ] )
120         {
121             bmsk += ( 1 <<(i-1) );
122             valori[ i ] += r[ i ];
123         }
124     }
125 }
126
127 if( bitmask[ bmsk ] == 0 )
128     eval( bmsk );
129
130 if( bitmask[ bmsk ] == 2 )
131     ans[ ++ans[ 0 ] ] = mini;
132 }
133
134 fout<<ans[ 0 ]<<'\\n';
135 for( int i = 1 ; i <= ans[ 0 ] ; i++ )
136     fout<<ans[ i ]<<' ';
137
138 return 0;
139 }
```

Listing 20.2.4: multimi-nlog.cpp

```

1 #include <iostream>
2 #include <algorithm>
3 #include <cstring>
4 #include <cassert>
5
6 using namespace std;
7
8 #define f first
9 #define s second
10
11 ifstream fin( "multimi.in" );
12 ofstream fout( "multimi.out" );
13
14 int i,j,n,m,k,a,bitmask[ (1<<16) ],s[20],f[20],r[20],ev[300],st[10010],
15     val[300],norm[300],ans[16 * 10010],ok,mini,bmsk;
16 char tr,multimea,expx[1010],evexp[1019];
17 int valori[ 20 ];
18 pair<int,int> sortare[16 * 10010];
19
20 int INDEX = 0;
21
22 char top()
23 {
24     return evexp[ INDEX ];
25 }
26
27 char pop()
28 {
29     return evexp[ INDEX ++ ];
30 }
31
32 int solveR();
33
34 int solveP()
35 {
36     if( top() != '(' )
37         return pop()-'0';
38     else
39         if( top() == '(' )
40         {
41             pop();
42             int rez = solveR();
43             pop();
44             return rez;
45         }
46
47     return 0;
48 }
```

```

49
50 int solveI()
51 {
52     int rez = solveP();
53     while( top() == '*' )
54     {
55         pop();
56         rez = min( rez , solveP() );
57     }
58     return rez;
59 }
60
61 int solveR()
62 {
63     int rez = solveI();
64     while( top() == '+' )
65     {
66         pop();
67         rez = max( rez , solveI() );
68     }
69     return rez;
70 }
71
72 int main()
73 {
74     fin>>n;
75     assert( n <= 16 );
76
77     fin.get();
78     for( int i = 1 ; i <= n ; i++ )
79     {
80         fin>>multimea>>tr>>valori[i]>>tr>>f[i]>>tr>>r[i];
81         norm[ multimea ] = 'A' + i - 1;
82         ev[ multimea ] = i;
83         assert( f[ i ] <= 1000000000 );
84     }
85     fin.get();
86     fin.get( expx , 10005 );
87     k = strlen( expx );
88     assert( k >= 3 );
89     assert( k <= 1000 );
90
91     for( int i = 0 ; i < k ; i++ )
92         if( expx[ i ] >= 'A' && expx[ i ] <= 'Z' )
93             expx[ i ] = norm[ expx[ i ] ];
94
95     val[ '+' ] = '+';
96     val[ '*' ] = '*';
97     val[ '(' ] = '(';
98     val[ ')' ] = ')';
99
100    for( int i = 0 ; i < ( 1<<(n) ) ; i++ )
101    {
102        for( int j = 0 ; j < n ; j++ )
103        {
104            val[ 'A' + j ] = '1';
105            if( ( i & (1<<(j)) ) == 0 )
106                val[ 'A' + j ] = '0';
107        }
108
109        for( int j = 0 ; j < k ; j++ )
110            evexp[ j ] = val[ expx[ j ] ];
111
112        INDEX = 0;
113        bitmask[ i ] = solveR();
114    }
115
116    ok = 0;
117    for( int i = 1 ; i <= n ; i++ )
118        for( int j = valori[ i ] ; j <= f[ i ] ; j += r[ i ] )
119            sortare[ ++ok ] = { j , i };
120
121    sort( sortare + 1 , sortare + ok + 1 );
122
123    for( int i = 1 ; i <= ok ; i++ )
124    {

```

```

125         j = i;
126         bmsk = 0;
127         while( sortare[ i ].f == sortare[ j ].f )
128         {
129             bmsk += ( 1 << ( sortare[ j ].s - 1 ) );
130             j++;
131         }
132         i = j - 1;
133         if( bitmask[ bmsk ] == 1 )
134             ans[ ++ans[ 0 ] ] = sortare[ i ].f;
135     }
136
137     fout<<ans[ 0 ]<<'\n';
138     for( int i = 1 ; i <= ans[ 0 ] ; i++ )
139         fout<<ans[ i ]<<' ';
140
141     return 0;
142 }
```

Listing 20.2.5: multimi-turturica.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <set>
4 #include <cstring>
5
6 using namespace std;
7
8 ifstream fin( "multimi.in" );
9 ofstream fout( "multimi.out" );
10
11 int i,j,n,m,k,a,bitmask[ (1<<16)],s[20],f[20],r[20],ev[300],st[10010],
12     val[300],norm[300],ans[16 * 10010],ok,mini,bmsk;
13 char tr,multimea,exp[10010],evexp[10019];
14 int valori[ 20 ];
15
16 int INDEX = 0;
17
18 char top()
19 {
20     return evexp[ INDEX ];
21 }
22
23 char pop()
24 {
25     return evexp[ INDEX ++ ];
26 }
27
28 int solveR();
29
30 int solveP()
31 {
32     if( top() != '(' )
33         return pop()-'0';
34     else if( top() == '(' )
35     {
36         pop();
37         int rez = solveR();
38         pop();
39         return rez;
40     }
41
42     return 0;
43 }
44
45 int solveI()
46 {
47     int rez = solveP();
48     while( top() == '*' )
49     {
50         pop();
51         rez = min( rez , solveP() );
52     }
53     return rez;
54 }
```

```

55
56 int solveR()
57 {
58     int rez = solveI();
59     while( top() == '+' )
60     {
61         pop();
62         rez = max( rez , solveI() );
63     }
64     return rez;
65 }
66
67 int main()
68 {
69     fin>>n;
70     fin.get();
71     for( int i = 1 ; i <= n ; i++ )
72     {
73         fin>>multimea>>tr>>valori[i]>>tr>>f[i]>>tr>>r[i];
74         norm[ multimea ] = 'A' + i - 1;
75         ev[ multimea ] = i;
76     }
77
78     fin.get();
79     fin.get( exp , 10005 );
80     k = strlen( exp );
81
82     for( int i = 0 ; i < k ; i++ )
83     {
84         if( exp[ i ] >= 'A' && exp[ i ] <= 'Z' )
85             exp[ i ] = norm[ exp[ i ] ];
86
87     val[ '+' ] = '+';
88     val[ '*' ] = '*';
89     val[ '(' ] = '(';
90     val[ ')' ] = ')';
91
92     for( int i = 0 ; i < ( 1<<(n) ) ; i++ )
93     {
94         for( int j = 0 ; j < n ; j++ )
95         {
96             val[ 'A' + j ] = '1';
97             if( ( i & (1<<(j)) ) == 0 )
98                 val[ 'A' + j ] = '0';
99         }
100        evexp[ j ] = val[ exp[ j ] ];
101        INDEX = 0;
102        bitmask[ i ] = solveR();
103    }
104
105    ok = 1;
106    while( ok == 1 )
107    {
108        ok = 0;
109        mini = 1000000009;
110        for( int i = 1 ; i <= n ; i++ )
111        {
112            if( valori[ i ] <= f[ i ] )
113            {
114                ok = 1;
115                mini = min( mini , valori[ i ] );
116            }
117        }
118
119        bmsk = 0;
120        for( int i = 1 ; i <= n ; i++ )
121        {
122            if( valori[ i ] == mini && valori[ i ] <= f[ i ] )
123            {
124                bmsk += ( 1 <<(i-1) );
125                valori[ i ] += r[ i ];
126            }
127        }
128
129        if( bitmask[ bmsk ] == 1 )
130            ans[ ++ans[ 0 ] ] = mini;

```

```

131     }
132
133     fout<<ans[ 0 ]<<' \n';
134     for( int i = 1 ; i <= ans[ 0 ] ; i++ )
135         fout<<ans[ i ]<<' ';
136
137     return 0;
138 }
```

Listing 20.2.6: priv_final.cpp

```

1 #include <iostream>
2 #include <cstring>
3 #include <algorithm>
4
5 using namespace std;
6
7 ifstream fin("multimi.in");
8 ofstream fout("multimi.out");
9
10 int n,i,j,k,l,d,r,p,f[126],kso,ksx,stivax[1005],b[30][3];
11 char e[1005],m[30],ch, stivao[1005];
12
13 int x[1000005],nx,w[1000005];///nr de pozitii egal cu nr de valori distincte
14
15 char a[70005];///2^n pozitii
16
17 int evaluare(int v)
18 {
19     int w[30],i,p;
20     fill(w,w+30,0);
21     for(i=0;i<=n-1;i++)
22     {
23         p=(1<<i);
24         if(v&p)w[i]=1;
25     }
26
27     kso=0;
28     ksx=0;
29     for(i=0;e[i];i++)
30     {
31         if(e[i]=='('){kso++;stivao[kso]=e[i];}
32         else if(e[i]=='+') {kso++;stivao[kso]=e[i];}
33         else if(e[i]=='*') {kso++;stivao[kso]=e[i];}
34         else if('A'<=e[i] && e[i]<='Z')
35         {
36             ksx++;
37             stivax[ksx]=w[f[e[i]]];
38             if(stivao[kso]=='*')
39             {
40                 ///efectuez intersectia
41                 ksx--;
42                 stivax[ksx]=stivax[ksx+1] && stivax[ksx];
43                 stivao[kso]=0;
44                 kso--;
45             }
46         }
47         else if(e[i]==')')
48         {
49             while(stivao[kso]!='(')
50             {
51                 if(stivao[kso]=='+')
52                 {
53                     ///efectuez reuniunea
54                     ksx--;
55                     stivax[ksx]=stivax[ksx+1] || stivax[ksx];
56                     stivao[kso]=0;
57                     kso--;
58                 }
59             }
60             stivao[kso]=0;
61             kso--;
62             if(stivao[kso]=='*')
63             {
64
```

```

65          // efectuez intersectia
66          ksx--;
67          stivax[ksx]=stivax[ksx+1] && stivax[ksx];
68          stivao[kso]=0;
69          kso--;
70      }
71  }
72 }
73
74 return stivax[ksx];
75 }
76
77 int main()
78 {
79     fin>>n;
80     fin.get();
81     nx=0;
82     for(i=0;i<=n-1;i++)
83     {
84         fin.get(m[i]); // numele multimii
85         f[m[i]]=i; // pozitia din a[][] a multimii m[i]
86         fin.get(ch); // sare peste egal
87
88         j=0;
89         do
90         {
91             fin.get(ch); // cifra sau -
92             if('0'<=ch && ch<='9'){
93                 j=j*10+(ch-'0');
94             }
95             else{ // a fost minus
96                 break;
97             }
98         } while(1);
99
100        k=0;
101        do
102        {
103            fin.get(ch); // cifra sau /
104            if('0'<=ch && ch<='9'){
105                k=k*10+(ch-'0');
106            }
107            else{ // a fost /
108                break;
109            }
110        } while(1);
111
112        d=0;
113        do
114        {
115            fin.get(ch); // cifra sau enter
116            if('0'<=ch && ch<='9')
117            {
118                d=d*10+(ch-'0');
119            }
120            else
121            { // a fost enter
122                break;
123            }
124        } while(1);
125
126        b[i][0]=j;
127        b[i][1]=k;
128        b[i][2]=d;
129        for(r=j;r<=k;r=r+d)
130        {
131            nx++;
132            x[nx]=r;
133        }
134    }
135
136    sort(x+1,x+1+nx);
137
138    j=1;
139    for(i=2;i<=nx;i++)
140    {

```

```

141         if(x[j]<x[i])
142     {
143         j++;
144         x[j]=x[i];
145     }
146 }
147 nx=j;
148
149 fin.get(e+1,1005);
150 e[0]='(';
151 strcat(e,")");
152
153 p=(1<<n);
154 for(i=0;i<=p-1;i++)
155     a[i]=evaluare(i);
156
157 k=0;
158 for(i=1;i<=nx;i++)
159 {
160     p=0;
161
162     //daca exista s j+s*d==x[i], s*d==x[i]-j
163     for(j=n-1;j>=0;j--)
164     {
165         if(b[j][0]<=x[i] && x[i]<=b[j][1] && (x[i]-b[j][0])%b[j][2]==0)
166             p=p*2+1;
167         else
168             p=p*2+0;
169     }
170
171     if(a[p]==1)
172     {
173         k++;
174         w[k]=x[i];
175     }
176 }
177
178 fout<<k<<"\n";
179 for(i=1;i<=k;i++)
180     fout<<w[i]<<" ";
181
182 fout<<"\n";
183 fout.close();
184 fin.close();
185 return 0;
186 }

```

20.3 siruri

Problema 3 - siruri

100 de puncte

Fie M un tablou bidimensional (matrice), format din N linii și N coloane, având elemente din mulțimea $\{0, 1\}$. Liniile și coloanele acestui tablou sunt numerotate de 1 la N . Un drum de lungime K în acest tablou este un sir de perechi (x_i, y_i) , cu i de la 1 la K , cu următoarele proprietăți:

- x_i și y_i sunt valori numere naturale din intervalul $[1, N]$, pentru i de la 1 la K ;
- $(|x_i - x_{i+1}| = 1 \text{ și } y_i = y_{i+1})$ sau $(|y_i - y_{i+1}| = 1 \text{ și } x_i = x_{i+1})$, pentru i de la 1 la $K - 1$.

Altfel spus, un drum este o succesiune de poziții în tablou astfel încât două poziții consecutive sunt alăturate fie susjos, fie stânga-dreapta. Numim **valoare a unui drum** sirul de valori $M[x_i][y_i]$, cu i de la 1 la K .

Prin **anagramă** a unui sir de valori vom înțelege orice rearanjare a elementelor acestuia. De exemplu, dacă sirul inițial conține elementele $(1, 0, 0, 1)$, atunci sirurile $(1, 0, 0, 1)$, $(1, 0, 1, 0)$, $(1, 1, 0, 0)$, $(0, 1, 1, 0)$ sunt anagrame ale acestuia.

Cerințe

Cunoscând dimensiunea N a tabloului, elementele tabloului M și un sir S , compus din K elemente din mulțimea $\{0, 1\}$, determinați un drum în tabloul M , având ca valoare o anagramă a lui S , dacă acest drum există.

Date de intrare

Datele de intrare se citesc din fișierul **siruri.in**, care are următoarea structură:

- pe prima linie se află două numere naturale N și K , separate printr-un singur spațiu, reprezentând dimensiunea tabloului, respectiv lungimea șirului S ;
- pe fiecare din următoarele N linii se află câte N valori din mulțimea $\{0, 1\}$, separate prin câte un spațiu, reprezentând valorile tabloului M ;
- pe linia $N + 2$ se află K numere, din mulțimea $\{0, 1\}$, separate prin câte un spațiu, reprezentând valorile șirului S .

Date de ieșire

Datele de ieșire se vor scrie în fișierul **siruri.out**, astfel:

- Dacă nu există niciun drum care satisfacă cerința problemei, atunci se va scrie valoarea -1 ;
- Dacă există un astfel de drum, atunci se vor scrie, pe linii separate, K perechi de valori $x_i y_i$, din mulțimea $\{1, 2, \dots, N\}$, separate printr-un singur spațiu, reprezentând drumul determinat.

Restricții și precizări

- $1 \leq N \leq 100$
- $1 \leq K \leq 100\,000$
- Elementele tabloului M și ale șirului S sunt elemente din mulțimea $\{0, 1\}$
- Pentru teste în valoare de 20 de puncte, șirul S se găsește pe o linie sau pe o coloană a matricei, așa cum este dat în fișierul de intrare sau în ordine inversă;
- Soluția nu este unică, se acceptă orice soluție.

Exemple:

siruri.in	siruri.out	Explicații
3 10 0 0 0 0 1 1 1 0 1 1 1 1 0 1 1 1 0 1 1	2 2 2 3 2 2 2 1 2 2 2 3 2 2 2 3 3 3 3 2	Valoarea obținută pentru drumul din fișierul de ieșire este 1 1 1 0 1 1 1 1 1 0 și reprezintă o anagramă a șirului dat.
4 3 1 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 1 1 0	4 1 3 1 2 1	Șirul S se găsește pe coloana 1 a matricei.
2 4 0 0 0 0 1 0 1 0	-1	În tablou nu există valoarea 1, deci nu se poate obține un drum a cărui valoare să conțină 1.

Timp maxim de executare/test: **1.0** secunde sub Windows, **0.3** secunde sub Linux

Memorie: total **64 MB** din care pentru stivă **32 MB**

Dimensiune maximă a sursei: **30 KB**

20.3.1 Indicații de rezolvare

La această problemă nu este nevoie de foarte multe cunoștințe teoretice, doar de atenție la acoperirea tuturor posibilităților.

Vom lua câteva exemple:

I. Dacă sirul S e format dintr-un singur caracter (fie el 0 sau 1), trebuie doar să existe acel caracter în matricea M

II. Dacă sirul S e format din $K > 1$ caractere, toate egale (fie ele 0 sau 1), atunci avem nevoie ca în matrice să fie două elemente alăturate cu această valoare (0 sau 1). Dacă nu ar exista, atunci tabla ar arăta ca una de săh și, oricum am alege 2 elemente alăturate, ele ar avea valori diferite.

III. Dacă sirul S e format din $K > 1$ caractere, unde numărul de 0-uri este K_0 și numărul de 1-uri este K_1 cu $|K_1 - K_0| \leq 1$, și pe deasupra $K_0, K_1 > 0$. În acest caz, în matrice avem nevoie de două celule alăturate (x_0, y_0) și (x_1, y_1) , unde $M[x_0][y_0] = 0$ și $M[x_1][y_1] = 1$.

Aceste două elemente ne sunt suficiente să construim soluția: alternăm între ele de $K - 1$ ori și pornim de la cea care apare de mai multe ori (dacă avem $K_0 > K_1$, atunci pornim de la (x_0, y_0) , altfel pornim de la (x_1, y_1))

IV. $K_0 > K_1 + 1$, din *principiul cutiei* orice anagramă a lui S vom forma, la un moment dat va trebui să avem doi de 0 consecutivi, fie urmați, fie precedați de un 1. Deci avem nevoie ca în matrice să existe 3 celule (x_0, y_0) , (x_1, y_1) , (x_2, y_2) astfel încât (x_0, y_0) și (x_1, y_1) sunt alăturate, (x_1, y_1) și (x_2, y_2) sunt alăturate și $M[x_0][y_0] = M[x_1][y_1] = 0$ și $M[x_2][y_2] = 1$.

Ca și la cazul anterior, aceste 3 elemente sunt suficiente. Pornind de la (x_2, y_2) , alternăm între (x_2, y_2) și (x_1, y_1) de $2 * K_1 - 1$ ori, iar apoi alternăm între (x_1, y_1) și (x_0, y_0) de $K - 2 * K_1$ ori.

V. $K_1 > K_0 + 1$, similar cazului anterior

Problema se reduce acum la a găsi în matrice elemente alăturate, care au anumite valori, iar pentru asta se poate folosi orice metodă, intrucât limitele de timp sunt suficient de largi.

20.3.2 *Rezolvare detaliată

20.3.3 Cod sursă

Listing 20.3.1: siruri_1.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <algorithm>
5 #include <cassert>
6
7 using namespace std;
8
9 int abs(int x)
10 {
11     if (x < 0)
12         return -x;
13     return x;
14 }
15
16 static const int dx[4] = {-1, 0, 1, 0};
17 static const int dy[4] = {0, -1, 0, 1};
18
19 bool back(const vector<vector<int>> &V, int x, int y,
20           const vector<int>& values, int K, vector<pair<int, int>> &where)
21 {
22     if (x < 0 || y < 0 || x >= int(V.size()) || y >= int(V[x].size()))
23         return false;
24
25     if (V[x][y] != values[K])
26         return false;
27
28     where.emplace_back(x, y);
29
30     K += 1;

```

```

31     if (K == int(values.size()))
32         return true;
33
34     for (int i = 0; i < 4; ++i)
35     {
36         int new_x = x + dx[i];
37         int new_y = y + dy[i];
38         if (back(V, new_x, new_y, values, K, where))
39             return true;
40     }
41     where.pop_back();
42     return false;
43 }
44
45 vector< pair<int, int> > search(const vector< vector<int> > &V,
46                                     vector<int> values)
47 {
48     vector< pair<int, int> > where;
49     for (int i = 0; i < int(V.size()); ++i)
50         for (int j = 0; j < int(V.size()); ++j)
51             if (back(V, i, j, values, 0, where))
52                 return where;
53     return where;
54 }
55
56 int main()
57 {
58     ifstream cin("siruri.in");
59     ofstream cout("siruri.out");
60
61     int N, K;
62     assert(cin >> N >> K);
63     assert(1 <= N && N <= 100);
64     assert(1 <= K && K <= 100 * 1000);
65
66     vector< vector<int> > M(N, vector<int>(N));
67
68     for (int i = 0; i < N; ++i)
69         for (int j = 0; j < N; ++j)
70         {
71             assert(cin >> M[i][j]);
72             assert(0 <= M[i][j] && M[i][j] <= 1);
73         }
74
75     vector<int> S(K);
76     for (int i = 0; i < K; ++i)
77     {
78         assert(cin >> S[i]);
79         assert(0 <= S[i] && S[i] <= 1);
80     }
81
82     if (K == 1)
83     { // simple case
84         for (int i = 0; i < N; ++i)
85             for (int j = 0; j < N; ++j)
86                 if (S[0] == M[i][j])
87                 {
88                     cout << i + 1 << " " << j + 1 << "\n";
89                     return 0;
90                 }
91         cout << "-1\n";
92         return 0;
93     }
94
95     int zeros = count(S.begin(), S.end(), 0);
96     int ones = count(S.begin(), S.end(), 1);
97
98     if (zeros == 0 || ones == 0 || abs(zeros - ones) <= 1)
99     {
100         vector< pair<int, int> > how;
101         if (zeros == 0)
102             how = search(M, {1, 1});
103         else if (ones == 0)
104             how = search(M, {0, 0});
105         else if (zeros > ones)
106             how = search(M, {0, 1});

```

```

107         else
108             how = search(M, {1, 0});
109
110         if (how.empty())
111         {
112             cout << "-1\n";
113             return 0;
114         }
115
116         for (int i = 0; i < K; ++i)
117             cout << how[i%2].first + 1 << " " << how[i%2].second + 1 << "\n";
118         return 0;
119     }
120
121     vector< pair<int, int> > how;
122     int first_part;
123     if (zeros > ones)
124     {
125         how = search(M, {0, 0, 1});
126         first_part = zeros;
127     }
128     else
129     {
130         how = search(M, {1, 1, 0});
131         first_part = ones;
132     }
133
134     if (how.empty())
135     {
136         cout << "-1\n";
137         return 0;
138     }
139
140     int second_part = K - first_part;
141     for (int i = 0; i < second_part; ++i)
142     {
143         cout << how[2].first + 1 << " " << how[2].second + 1 << "\n";
144         cout << how[1].first + 1 << " " << how[1].second + 1 << "\n";
145     }
146
147     for (int i = 0; i < first_part - second_part; ++i)
148     {
149         cout << how[i % 2].first + 1 << " " << how[i % 2].second + 1 << "\n";
150     }
151 }
```

Listing 20.3.2: siruri_2.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 ifstream fin("siruri.in");
6 ofstream fout("siruri.out");
7
8 struct pereche
9 {
10     int L,C;
11 };
12
13 struct vpereche
14 {
15     pereche v[5];
16     int x;
17 };
18
19 int N, K, i, j, l, c, l1, cl, l2, c2, k, k1, k2,
20     M[105][105], S[100005], nv[2], nm[2], viz[1000], x, nr, r, ok;
21 vpereche a[20],v;
22
23 int dl[4]={-1,+1, 0, 0};
24 int dc[4]={ 0, 0,-1,+1};
25
26 int main()
27 {
```

```

28     fin>>N>>K;
29     for(i=1;i<=N;i++)
30     {
31         for(j=1;j<=N;j++)
32         {
33             fin>>M[i][j];
34             nm[M[i][j]]++;
35             if(M[i][j]==0 && nm[0]==1)
36             {
37                 v.v[0]={i,j};
38             }
39             if(M[i][j]==1 && nm[1]==1)
40             {
41                 v.v[1]={i,j};
42             }
43         }
44     }
45
46     for(i=1;i<=K;i++)
47     {
48         fin>>S[i];
49         nv[S[i]]++;
50     }
51
52     if(K==1)
53     {
54         if((nv[0]==1 && nm[0]==0) || (nv[1]==1 && nm[1]==0))
55         {
56             fout<<-1;
57             fout.close();
58             return 0;
59         }
60
61         if(nv[0]==1)
62         {
63             fout<<v.v[0].L<<" "<<v.v[0].C;
64             fout.close();
65             return 0;
66         }
67
68         fout<<v.v[1].L<<" "<<v.v[1].C;
69         fout.close();
70         return 0;
71     }
72
73     if(K>1 && N==1)
74     {
75         fout<<-1;
76         fout.close();
77         return 0;
78     }
79
80     for(i=0;i<=999;i++) viz[i]=-1;
81
82     for(i=1;i<=N;i++)
83     {
84         for(j=1;j<=N;j++)
85         {
86             l=i;
87             c=j;
88             r=1;
89             v.v[1]={i,j};
90             x=M[i][j];
91             for(k=0;k<=3;k++)
92             {
93                 l=i+dl[k];
94                 c=j+dc[k];
95                 if(l>=1 && l<=N && c>=1 && c<=N)
96                 {
97                     r++;
98                     v.v[r]={l,c};
99                     x=x*2+M[l][c];
100                    for(k1=0;k1<=3;k1++)
101                    {
102                        l1=l+dl[k1]; c1=c+dc[k1];
103                        if(l1>=1 && l1<=N && c1>=1 && c1<=N)

```

```

104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138 ok=0;
139 if(nv[0]==K && ok==0)
140 {
141     for(i=1;i<=nr;i++)
142     {
143         if(a[i].x/4==0)
144         {
145             // ok=1;
146             k=1;
147             for(j=1;j<=K;j++)
148             {
149                 fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
150                 k=3-k;
151             }
152             break;
153         }
154     }
155
156     if(a[i].x/2%4==0)
157     {
158         // ok=1;
159         k=2;
160         for(j=1;j<=K;j++)
161         {
162             fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
163             k=5-k;
164         }
165         break;
166     }
167
168     if(a[i].x%4==0)
169     {
170         // ok=1;
171         k=3;
172         for(j=1;j<=K;j++)
173         {
174             fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
175             k=7-k;
176         }
177         break;
178
179

```

```

180         }
181     }
182
183     if(ok==1)
184     {
185         fout.close();
186         return 0;
187     }
188 }
189
190 if(nv[1]==K && ok==0)
191 {
192     for(i=1;i<=nr;i++)
193     {
194         if(a[i].x/4==3)
195         {
196             //
197             ok=1;
198             k=1;
199             for(j=1;j<=K;j++)
200             {
201                 fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
202                 k=3-k;
203             }
204             break;
205         }
206
207         if(a[i].x/2%4==3)
208         {
209             //
210             ok=1;
211             k=2;
212             for(j=1;j<=K;j++)
213             {
214                 fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
215                 k=5-k;
216             }
217             break;
218         }
219
220         if(a[i].x%4==3)
221         {
222             //
223             ok=1;
224             k=3;
225             for(j=1;j<=K;j++)
226             {
227                 fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
228                 k=7-k;
229             }
230             break;
231         }
232     }
233
234     if(ok==1)
235     {
236         fout.close();
237         return 0;
238     }
239 }
240
241 if(nv[0]>=nv[1]+2 && ok==0)
242 {
243     int d=nv[0]-nv[1];
244     for(i=1;i<=nr;i++)
245     {
246         if(a[i].x/2==1)
247         {
248             ok=1;
249             if(d%2==0) k=2;
250             else k=1;
251             for(j=1;j<=d;j++)
252             {
253                 fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
254                 k=3-k;
255             }

```

```

256         for(j=1; j<=K-d; j++)
257         {
258             fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
259             k=5-k;
260         }
261         break;
262     }
263
264     if(a[i].x%8==1)
265     {
266         ok=1;
267         if(d%2==0) k=3;
268         else k=2;
269         for(j=1; j<=d; j++)
270         {
271             fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
272             k=5-k;
273         }
274         for(j=1; j<=K-d; j++)
275         {
276             fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
277             k=7-k;
278         }
279         break;
280     }
281
282     if(a[i].x/2==4)
283     {
284         //
285         ok=1;
286         if(d%2==0) k=2;
287         else k=3;
288         for(j=1; j<=d; j++)
289         {
290             fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
291             k=5-k;
292         }
293         for(j=1; j<=K-d; j++)
294         {
295             fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
296             k=3-k;
297         }
298         break;
299     }
300
301     if(a[i].x%8==4)
302     {
303         //
304         ok=1;
305         if(d%2==0) k=3;
306         else k=4;
307         for(j=1; j<=d; j++)
308         {
309             fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
310             k=7-k;
311         }
312         for(j=1; j<=K-d; j++)
313         {
314             fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
315             k=5-k;
316         }
317         break;
318     }
319 }
320
321     if(ok==1)
322     {
323         fout.close();
324         return 0;
325     }
326 }
327
328 if(nv[0]+2<=nv[1] && ok==0)
329 {
330     int d=nv[1]-nv[0];
331     for(i=1;i<=nr;i++)

```

```

332      {
333          if(a[i].x/2==6)
334          {
335              //
336              ok=1;
337              if(d%2==0) k=2;
338              else k=1;
339              for(j=1; j<=d; j++)
340              {
341                  fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
342                  k=3-k;
343              }
344              for(j=1; j<=K-d; j++)
345              {
346                  fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
347                  k=5-k;
348              }
349              break;
350          }
351
352          if(a[i].x%8==6)
353          {
354              //
355              ok=1;
356              if(d%2==0) k=3;
357              else k=2;
358              for(j=1; j<=d; j++)
359              {
360                  fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
361                  k=5-k;
362              }
363              for(j=1; j<=K-d; j++)
364              {
365                  fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
366                  k=7-k;
367              }
368              break;
369          }
370
371          if(a[i].x/2==3)
372          {
373              ok=1;
374              if(d%2==0)
375                  k=2;
376              else
377                  k=3;
378
379              for(j=1; j<=d; j++)
380              {
381                  fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
382                  k=5-k;
383              }
384              for(j=1; j<=K-d; j++)
385              {
386                  fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
387                  k=3-k;
388              }
389              break;
390          }
391
392          if(a[i].x%8==3)
393          {
394              //
395              ok=1;
396              if(d%2==0) k=3;
397              else k=4;
398              for(j=1; j<=d; j++)
399              {
400                  fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
401                  k=7-k;
402              }
403              for(j=1; j<=K-d; j++)
404              {
405                  fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
406                  k=5-k;
407              }

```

```

408             break;
409         }
410     }
411
412     if(ok==1)
413     {
414         fout.close();
415         return 0;
416     }
417 }
418
419 if((nv[0]==nv[1] || nv[0]==1+nv[1])&& ok==0)
420 {
421     for(i=1;i<=nr;i++)
422     {
423         if(a[i].x/4==1 || a[i].x/4==2)
424         {
425             //
426             ok=1;
427             k=a[i].x/4;
428             for(j=1;j<=K;j++)
429             {
430                 fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
431                 k=3-k;
432             }
433             break;
434         }
435
436         if(a[i].x/2%4==1 || a[i].x/2%4==2)
437         {
438             //
439             ok=1;
440             k=a[i].x/2%4+1;
441             for(j=1;j<=K;j++)
442             {
443                 fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
444                 k=5-k;
445             }
446             break;
447         }
448
449         if(a[i].x%4==1 || a[i].x%4==2)
450         {
451             ok=1;
452             k=a[i].x%4+2;
453             for(j=1;j<=K;j++)
454             {
455                 fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
456                 k=7-k;
457             }
458             break;
459         }
460     }
461
462     if(ok==1)
463     {
464         fout.close();
465         return 0;
466     }
467 }
468
469 if(nv[0]+1==nv[1] && ok==0)
470 {
471     for(i=1;i<=nr;i++)
472     {
473         if(a[i].x/4==2 || a[i].x/4==1)
474         {
475             //
476             ok=1;
477             k=3-a[i].x/4;
478             for(j=1;j<=K;j++)
479             {
480                 fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
481                 k=3-k;
482             }
483             break;

```

```

484         }
485
486         if(a[i].x/2%4==2 || a[i].x/2%4==1)
487         {
488             ok=1;
489             k=4-a[i].x/2%4==2;
490             for(j=1; j<=K; j++)
491             {
492                 fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
493                 k=5-k;
494             }
495             break;
496         }
497
498         if(a[i].x%4==2 || a[i].x%4==1)
499         {
500             ok=1;
501             k=5-a[i].x%4;
502             for(j=1; j<=K; j++)
503             {
504                 fout<<a[i].v[k].L<<" "<<a[i].v[k].C<<"\n";
505                 k=7-k;
506             }
507             break;
508         }
509     }
510
511     if(ok==1)
512     {
513         fout.close();
514         return 0;
515     }
516 }
517
518 if(ok==0) fout<<-1;
519 fout.close();
520 return 0;
521 }
```

Listing 20.3.3: siruri_turturica.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 ifstream fin( "siruri.in" );
6 ofstream fout( "siruri.out" );
7
8 #define f first
9 #define s second
10 #define X1 first.first
11 #define Y1 first.second
12 #define X2 second.first.first
13 #define Y2 second.first.second
14 #define X3 second.second.first
15 #define Y3 second.second.second
16
17 int v[110][110], i, j, n, m, k, o, a, b;
18
19 pair< pair<int,int> , pair<int,int> > poz;
20 pair< pair<int,int> , pair< pair<int,int> , pair<int,int> > > p;
21
22 int X[] = {0,1,-1,0,0};
23 int Y[] = {0,0,0,1,-1};
24
25 pair< pair<int,int> , pair<int,int> > find1()
26 {
27
28     for( int i = 1 ; i <= n ; i++ )
29     {
30         for( int j = 1 ; j <= n ; j++ )
31         {
32             if( v[ i ][ j ] == 0 )
33             {
34                 for( int k = 1 ; k <= 4 ; k++ )
35                 {
36                     if( X[k] == i && Y[k] == j )
37                     {
38                         if( poz.X1 == i && poz.Y1 == j )
39                         {
40                             if( poz.X2 == k && poz.Y2 == 0 )
41                             {
42                                 p.push_back( poz );
43                                 poz.X1 = i;
44                                 poz.Y1 = j;
45                                 poz.X2 = k;
46                                 poz.Y2 = 1;
47                             }
48                         }
49                     }
50                 }
51             }
52         }
53     }
54 }
```

```

35             {
36                 if( v[ i + X[ k ] ][ j + Y[ k ] ] == 1 )
37                     return { {i,j} , { i + X[ k ] , j + Y[ k ] } };
38             }
39         }
40     }
41 }
42 fout<<-1;
43 exit( 0 );
44 }
45
46 pair< pair<int,int> , pair<int,int> > find3()
47 {
48
49     for( int i = 1 ; i <= n ; i++ )
50     {
51         for( int j = 1 ; j <= n ; j++ )
52         {
53             if( v[ i ][ j ] == 0 )
54             {
55                 for( int k = 1 ; k <= 4 ; k++ )
56                 {
57                     if( v[ i + X[ k ] ][ j + Y[ k ] ] == 0 )
58                         return { {i,j} , { i + X[ k ] , j + Y[ k ] } };
59                 }
60             }
61         }
62     }
63     fout<<-1;
64     exit( 0 );
65 }
66
67 pair< pair<int,int> , pair< pair<int,int> , pair<int,int> > > find2()
68 {
69     for( int i = 1 ; i <= n ; i++ )
70     {
71         for( int j = 1 ; j <= n ; j++ )
72         {
73             if( v[ i ][ j ] == 0 )
74             {
75                 int x1 = i,y1 = j;
76                 for( int k = 1 ; k <= 4 ; k++ )
77                 {
78                     if( v[ x1 + X[ k ] ][ y1 + Y[ k ] ] == 0 )
79                     {
80                         int x2 = x1 + X[ k ];
81                         int y2 = y1 + Y[ k ];
82                         for( int k2 = 1 ; k2 <= 4 ; k2++ )
83                         {
84                             if( v[ x2 + X[ k2 ] ][ y2 + Y[ k2 ] ] == 1 )
85                             {
86                                 int x3 = x2 + X[ k2 ];
87                                 int y3 = y2 + Y[ k2 ];
88                                 return { { x1,y1 } , { {x2,y2} , {x3,y3} } };
89                             }
90                         }
91                     }
92                 }
93             }
94         }
95     }
96     fout<<-1;
97     exit( 0 );
98 }
99
100 int main()
101 {
102     fin>>n>>k;
103
104     for( i = 1 ; i <= n ; i++ )
105         for( j = 1 ; j <= n ; j++ )
106         {
107             fin>>v[ i ][ j ];
108         }
109     for( i = 1 ; i <= k ; i++ )
110     {

```

```

111         fin>>o;
112         if( o == 0 )
113             a++;
114         else
115             b++;
116     }
117
118     for( i = 0 ; i <= n + 1 ; i++ )
119         v[ i ][ 0 ] = v[ 0 ][ i ] = v[ i ][ n + 1 ] = v[ n + 1 ][ i ] = -1;
120
121     if( n == 1 )
122     {
123         if( k == 1 )
124         {
125             if( v[1][1] == o )
126                 fout<<"1 1";
127             else
128                 fout<<-1;
129         }
130         else
131         {
132             fout<<-1;
133         }
134     }
135
136     if( b > a )
137     {
138         swap( a , b );
139         for( i = 1 ; i <= n ; i++ )
140             for( j = 1 ; j <= n ; j++ )
141             {
142                 v[ i ][ j ] = 1 - v[ i ][ j ];
143             }
144     }
145
146
147     if( b == 0 )
148     {
149         poz = find3();
150         if( a % 2 )
151             fout<<poz.s.f<<' '<<poz.s.s<<' \n';
152         for( i = 1 ; i <= a / 2 ; i++ )
153             fout<<poz.f.f<<' '<<poz.f.s<<' \n'<<poz.s.f<<' '<<poz.s.s<<' \n';
154         return 0;
155     }
156
157
158     if( a - b <= 1 )
159     {
160         poz = find1();
161         for( i = 1 ; i <= b ; i++ )
162             fout<<poz.f.f<<' '<<poz.f.s<<' \n'<<poz.s.f<<' '<<poz.s.s<<' \n';
163         if( a > b )
164             fout<<poz.f.f<<' '<<poz.f.s<<' \n';
165     }
166     else
167     {
168         p = find2();
169         if( ( a - b ) % 2 == 1 )
170         {
171             fout<<p.X1<<' '<<p.Y1<<' \n';
172             a--;
173         }
174         while( a != b )
175         {
176             a -= 2;
177             fout<<p.X2<<' '<<p.Y2<<' \n'<<p.X1<<' '<<p.Y1<<' \n';
178         }
179         for( i = 1 ; i <= b ; i++ )
180         {
181             fout<<p.X2<<' '<<p.Y2<<' \n'<<p.X3<<' '<<p.Y3<<' \n';
182         }
183     }
184
185     return 0;
186 }
```

Listing 20.3.4: siruriscanf.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 FILE *fin,*fout;
6
7 #define f first
8 #define s second
9 #define X1 f.first.first
10 #define Y1 f.first.second
11 #define X2 f.second.first.first
12 #define Y2 f.second.first.second
13 #define X3 f.second.second.first
14 #define Y3 f.second.second.second
15
16 int v[110][110],i,j,n,m,k,o,a,b;
17
18 pair< pair<int,int> , pair<int,int> > poz;
19 pair< pair<int,int> , pair< pair<int,int> , pair<int,int> > > p;
20
21 int X[] = {0,1,-1,0,0};
22 int Y[] = {0,0,0,1,-1};
23
24 pair< pair<int,int> , pair<int,int> > find1()
25 {
26
27     for( int i = 1 ; i <= n ; i++ )
28     {
29         for( int j = 1 ; j <= n ; j++ )
30         {
31             if( v[ i ][ j ] == 0 )
32             {
33                 for( int k = 1 ; k <= 4 ; k++ )
34                 {
35                     if( v[ i + X[ k ] ][ j + Y[ k ] ] == 1 )
36                         return { {i,j} , { i + X[ k ] , j + Y[ k ] } };
37                 }
38             }
39         }
40     }
41
42     fprintf( fout , "-1" );
43     exit( 0 );
44 }
45
46 pair< pair<int,int> , pair<int,int> > find3()
47 {
48     for( int i = 1 ; i <= n ; i++ )
49     {
50         for( int j = 1 ; j <= n ; j++ )
51         {
52             if( v[ i ][ j ] == 0 )
53             {
54                 for( int k = 1 ; k <= 4 ; k++ )
55                 {
56                     if( v[ i + X[ k ] ][ j + Y[ k ] ] == 0 )
57                         return { {i,j} , { i + X[ k ] , j + Y[ k ] } };
58                 }
59             }
60         }
61     }
62
63     fprintf( fout , "-1" );
64     exit( 0 );
65 }
66
67 pair< pair<int,int> , pair< pair<int,int> , pair<int,int> > > find2()
68 {
69     for( int i = 1 ; i <= n ; i++ )
70     {
71         for( int j = 1 ; j <= n ; j++ )
72         {
73             if( v[ i ][ j ] == 0 )
74             {

```

```

75         int x1 = i,y1 = j;
76         for( int k = 1 ; k <= 4 ; k++ )
77         {
78             if( v[ x1 + X[ k ] ][ y1 + Y[ k ] ] == 0 )
79             {
80                 int x2 = x1 + X[ k ];
81                 int y2 = y1 + Y[ k ];
82                 for( int k2 = 1 ; k2 <= 4 ; k2++ )
83                 {
84                     if( v[ x2 + X[ k2 ] ][ y2 + Y[ k2 ] ] == 1 )
85                     {
86                         int x3 = x2 + X[ k2 ];
87                         int y3 = y2 + Y[ k2 ];
88                         return { { x1,y1 } , { {x2,y2} , {x3,y3} } };
89                     }
90                 }
91             }
92         }
93     }
94 }
95 }
96 fprintf( fout , "-1" );
97 exit( 0 );
98 }
99
100 int main()
101 {
102     fin = fopen( "siruri.in" , "r" );
103     fout = fopen( "siruri.out" , "w" );
104
105     fscanf( fin , "%d %d" , &n , &k );
106
107     for( i = 1 ; i <= n ; i++ )
108         for( j = 1 ; j <= n ; j++ )
109             fscanf( fin , "%d" , &v[ i ][ j ] );
110
111     for( i = 1 ; i <= k ; i++ )
112     {
113         fscanf( fin , "%d" , &o );
114         if( o == 0 )
115             a++;
116         else
117             b++;
118     }
119
120     for( i = 0 ; i <= n + 1 ; i++ )
121         v[ i ][ 0 ] = v[ 0 ][ i ] = v[ i ][ n + 1 ] = v[ n + 1 ][ i ] = -1;
122
123     if( n == 1 )
124     {
125         if( k == 1 )
126         {
127             if( v[1][1] == o )
128                 fprintf( fout , "1 1" );
129             else
130                 fprintf( fout , "-1" );
131         }
132         else
133             fprintf( fout , "-1" );
134     }
135
136     if( b > a )
137     {
138         swap( a , b );
139         for( i = 1 ; i <= n ; i++ )
140             for( j = 1 ; j <= n ; j++ )
141                 v[ i ][ j ] = 1 - v[ i ][ j ];
142     }
143
144     if( b == 0 )
145     {
146         poz = find3();
147         if( a % 2 )
148
149             fprintf( fout , "%d %d\n" , poz.s.f , poz.s.s );
150         for( i = 1 ; i <= a / 2 ; i++ )

```

```

151         fprintf(fout, "%d %d\n%d %d\n", poz.f.f, poz.f.s, poz.s.f, poz.s.s);
152         return 0;
153     }
154
155     if( a - b <= 1 )
156     {
157         poz = find1();
158         for( i = 1 ; i <= b ; i++ )
159             fprintf(fout, "%d %d\n%d %d\n", poz.f.f, poz.f.s, poz.s.f, poz.s.s);
160         if( a > b )
161             fprintf( fout , "%d %d\n" , poz.f.f , poz.f.s );
162     }
163 else
164 {
165     p = find2();
166     if( ( a - b ) % 2 == 1 )
167     {
168         fprintf( fout , "%d %d\n" , p.X1 , p.Y1 );
169         a--;
170     }
171     while( a != b )
172     {
173         a -= 2;
174         fprintf( fout , "%d %d\n%d %d\n" , p.X2 , p.Y2 , p.X1 , p.Y1 );
175     }
176     for( i = 1 ; i <= b ; i++ )
177     {
178         fprintf( fout , "%d %d\n%d %d\n" , p.X2,p.Y2,p.X3,p.Y3 );
179     }
180 }
181
182 return 0;
183 }
```

20.4 agora

Problema 4 - agora

100 de puncte

Prietenul nostru, Pit, se află în Grecia antică, în cea mai vestită piață publică. Considerăm că piață este un dreptunghi din plan, de dimensiuni X și Y . Dacă reprezentăm piață într-un reper cartezian xOy , aceasta are cele patru vârfuri în punctele de coordonate $(0,0)$, $(X,0)$, (X,Y) și $(0,Y)$. În fiecare punct (a,b) , cu $a \in \{1, \dots, X\}$ și $b \in \{1, \dots, Y\}$, se află câte o tarabă care vinde echere. Prietenul nostru este afacerist și vrea să închirieze o parcelă de teren dreptunghiulară, având laturile paralele cu laturile pieței, iar cele patru vârfuri de coordonate numere naturale. Vâfurile parcelei se află în interiorul pieței sau pe laturile acesteia. În această parcelă, Pit vrea să cuprindă cât mai multe tarabe speciale, care au următoarele proprietăți:

- distanța de la origine la tarabă este număr natural;
- nu există nici o altă tarabă pe segmentul dintre origine și tarabă.

Cerințe

Cunoscându-se valorile X , Y și coordonatele (S_{X_i}, S_{Y_i}) și (D_{X_i}, D_{Y_i}) pentru Q parcele, unde $1 \leq i \leq Q$, să se afle, pentru fiecare parcelă, care este numărul de tarabe speciale pe care le conține.

Date de intrare

Pe primul rând al fișierului **agora.in** se află trei numere naturale despărțite prin câte un spațiu, X , Y și Q , cu semnificația din enunț.

Pe următoarele Q rânduri se află câte 4 numere naturale nenule S_{x_i} , S_{y_i} , D_{x_i} , D_{y_i} , separate prin câte un spațiu, cu semnificația din enunț.

Date de ieșire

Pe fiecare dintre primele Q rânduri ale fișierului **agora.out** se va afla câte un număr natural, numărul de pe linia i reprezentând numărul tarabelor speciale conținute de către parcela i .

Restricții și precizări

- $2 \leq X \leq 7\ 000$
- $2 \leq Y \leq 7\ 000$
- $1 \leq Q \leq 100\ 000$
- o tarabă face parte dintr-o parcelă și dacă se află pe laturile ei;
- (S_{x_i}, S_{y_i}) și (D_{x_i}, D_{y_i}) nu se vor afla în afara dreptunghiului asociat pieței, dar se pot afla pe laturile lui;
 - Pentru teste în valoare de 10 puncte: $X, Y \leq 100$ și $Q \leq 100$
 - Pentru alte teste în valoare de 20 puncte: $X, Y \leq 2\ 000$ și $Q \leq 1\ 000$
 - Pentru alte teste în valoare de 10 puncte: $X, Y \leq 2\ 000$ și $Q \leq 100\ 000$

Exemple:

agora.in	agora.out	Explicații
5 5 2	1	Prima parcelă conține taraba specială de la punctul (3,4).
1 5 3 4	2	A doua parcelă conține tarabele speciale (3,4) și (4,3)
3 4 4 3		

Timp maxim de executare/test: **0.8** secunde pe Windows, **0.4** secunde pe linux

Memorie: total **64 MB** din care pentru stivă **32 MB**

Dimensiune maximă a sursei: **8 KB**

20.4.1 Indicații de rezolvare

Chiorean Tudor-Octavian - Universitatea Tehnică Cluj-Napoca

Observăm că toate punctele din plan care sunt considerate "speciale" sunt de fapt *punctele laticeale*, care determină triunghiuri dreptunghice cu catetele paralele cu axele de coordonate, având lungimile ipotenuzelor numere naturale.

Solutie O(X*Y*Q) - 10 puncte

Iterăm prin fiecare punct laticeal al planului și marcăm, într-o matrice de dimensiuni $X * Y$, cu valoarea 1, dacă este special sau cu 0, în caz contrar.

Pentru fiecare query, parcurgem dreptunghiul corespunzător și determinăm nuărul valorilor egale cu 1.

Solutie O(X*Y*log(X) + Q) - 40 de puncte

Aflăm, în același mod, toate punctele speciale, iar apoi precalculăm *sume parțiale* (pentru numărul valorilor egale cu 1) și răspundem la fiecare query în $O(1)$.

Solutie - 100 de puncte

Generarea punctelor speciale se poate face eficient în mai multe moduri. Unul dintre ele este folosirea *formulelor lui Euclid*:

Fixăm m, n , cu $m > n > 0$, m și n prime între ele și doar unul dintre ele impar.

Coordonatele punctului special determinate de acești parametrii vor fi:

$$x = m^2 - n^2, y = 2 * m * n$$

Luăm în considerare toate punctele și observăm că numărul de coordonate x și coordonate y distințe (pentru $X, Y \leq 7\ 000$) este mai mic decât aproximativ 2 500, astfel că vom putea *normaliza* coordonatele și precalcula *sume parțiale* pe matricea normalizată, la fel ca la punctul anterior.

Pentru fiecare query, vom determina dreptunghiul pe care il interogăm în matricea normalizată folosind *căutarea binară* după fiecare coordonată.

Răspunsul pe fiecare query va fi dat în timp logaritmic.

20.4.2 *Rezolvare detaliată

20.4.3 Cod sursă

Listing 20.4.1: agora.cpp

```

1  /**
2  Solutie cu generare folosind inmultire de matrici
3  */
4  #include <iostream>
5  #include <fstream>
6  #include <queue>
7  #include <iterator>
8  #include <set>
9  #include <vector>
10 #include <algorithm>
11
12 using namespace std;
13
14 #define x first
15 #define y second.first
16 #define z second.second
17 #define make_triple(a, b, c) make_pair(a, make_pair(b, c))
18
19 ifstream fin("agora.in");
20 ofstream fout("agora.out");
21
22 int x_1, y_1, x_0, y_0, ans;
23 int X, Y, K, i, j, xMax, yMax;
24 int mat[2500][2500];
25
26 vector<int> difX, difY;
27 vector<pair<int, int> > points;
28
29 queue<pair<int, pair<int, int> > Q;
30 pair<int, pair<int, int> > crt, nxt;
31
32 bool isBounded(int a, int b)
33 {
34     if (abs(a) > X || abs(b) > Y)
35         return false;
36     return true;
37 }
38
39 int main()
40 {
41     fin >> X >> Y >> K; // Plane is 0 to X,Y
42     Q.push(make_triple(3, 4, 5));
43
44     while (!Q.empty())
45     {
46         crt = Q.front(); Q.pop();
47
48         if (crt.x <= X && crt.y <= Y)
49         {
50             points.push_back(make_pair(crt.x, crt.y));
51             difX.push_back(crt.x);
52             difY.push_back(crt.y);
53         }
54
55         if (crt.y <= X && crt.x <= Y)
56         {
57             points.push_back(make_pair(crt.y, crt.x));
58             difX.push_back(crt.y);
59             difY.push_back(crt.x);
60         }
61
62         nxt.x = -crt.x + 2 * crt.y + 2 * crt.z;
63         nxt.y = -2 * crt.x + crt.y + 2 * crt.z;
64         nxt.z = -2 * crt.x + 2 * crt.y + 3 * crt.z;
65
66         if (isBounded(nxt.x, nxt.y))
67         {
68             Q.push(nxt);
69         }
70

```

```

71         nxt.x = crt.x + 2 * crt.y + 2 * crt.z;
72         nxt.y = 2 * crt.x + crt.y + 2 * crt.z;
73         nxt.z = 2 * crt.x + 2 * crt.y + 3 * crt.z;
74
75         if (isBounded(nxt.x, nxt.y))
76         {
77             Q.push(nxt);
78         }
79
80         nxt.x = crt.x + -2 * crt.y + 2 * crt.z;
81         nxt.y = 2 * crt.x + -crt.y + 2 * crt.z;
82         nxt.z = 2 * crt.x + -2 * crt.y + 3 * crt.z;
83
84         if (isBounded(nxt.x, nxt.y))
85         {
86             Q.push(nxt);
87         }
88     }
89
90     sort(difX.begin(), difX.end());
91     sort(difY.begin(), difY.end());
92
93     unique(difX.begin(), difX.end());
94     unique(difY.begin(), difY.end());
95
96     for(i = 0 ; i < points.size() ; i++)
97     {
98         mat[distance(difY.begin(), lower_bound(difY.begin(),
99                         difY.end(), points[i].second)) + 1]
100        [distance(difX.begin(), lower_bound(difX.begin(),
101                         difX.end(), points[i].first)) + 1] = 1;
102    }
103
104    xMax = difX.size();
105    yMax = difY.size();
106
107    for (i = 1 ; i <= xMax + 1 ; i++)
108        for (j = 1 ; j <= yMax + 1 ; j++)
109            mat[i][j] += mat[i-1][j] + mat[i][j-1] - mat[i-1][j-1];
110
111    for (i = 0 ; i < K ; i++)
112    {
113        fin >> x_1 >> y_1 >> x_0 >> y_0;
114        x_1=distance(difX.begin(),lower_bound(difX.begin(),difX.end(),x_1))+1;
115        x_0=distance(difX.begin(),upper_bound(difX.begin(),difX.end(),x_0));
116        y_1=distance(difY.begin(),upper_bound(difY.begin(),difY.end(),y_1));
117        y_0=distance(difY.begin(),lower_bound(difY.begin(),difY.end(),y_0))+1;
118
119        ans=mat[x_0][y_1]-mat[x_0][y_0-1]-mat[x_1-1][y_1]+mat[x_1-1][y_0-1];
120        fout << ans << '\n';
121    }
122
123    return 0;
124 }
```

20.5 grup

Problema 5 - grup

100 de puncte

În școală unde învață, Andrei și Bogdan cunosc alți N elevi, etichetați cu numerele 1, 2, ..., N . Dintre cei N elevi, o parte sunt prietenii lui Andrei. O parte dintre cei N elevi sunt dușmanii lui Bogdan. Se cunosc atât etichetele prietenilor lui Andrei, cât și etichetele dușmanilor lui Bogdan. Directorul școlii dorește să organizeze o excursie la care să participe Andrei, Bogdan și S dintre cunoșcuții acestora, astfel încât din grupul celor S elevi să facă parte cel puțin K_1 dintre prietenii lui Andrei și cel mult K_2 dintre dușmanii lui Bogdan. Dorind să evite evenimente neplăcute, directorul va alege cei S elevi astfel încât numărul total al absențelor acumulate de aceștia, notat Sm , să fie minim.

Cerințe

Cunoscând valorile N , S , K_1 , K_2 , etichetele prietenilor lui Andrei, etichetele dușmanilor

lui Bogdan, precum și numărul absențelor acumulate de fiecare dintre cei N elevi, determinăți valoarea Sm obținută pentru un grup ce satisfac condițiile de mai sus.

Date de intrare

Datele de intrare se citesc din fișierul text **grup.in**, cu structura următoare:

- pe prima linie se află valorile naturale $N, S, K1, K2$, separate prin câte un spațiu, cu semnificațiile din enunț;
- pe a doua linie se află valorile a_1, a_2, \dots, a_N , separate prin câte un spațiu, reprezentând numărul absențelor acumulate de fiecare dintre cei N elevi;
- pe a treia linie se află un sir compus din N caractere din mulțimea { '0', '1' }, neseparate prin spații. Dacă al i -lea caracter din sir este caracterul '1', atunci elevul cu eticheta i este prieten cu Andrei;
- pe a patra linie se află un sir compus din N caractere din mulțimea { '0', '1' }, neseparate prin spații. Dacă al i -lea caracter din sir este caracterul '1', atunci elevul cu eticheta i este dușmanul lui Bogdan.

Date de ieșire

Pe prima linie din fișierul text **grup.out** se va tipări valoarea Sm .

Restricții și precizări

- $2 \leq N \leq 100\,000$
- $1 \leq a_i \leq 1\,000\,000\,000, \forall i \in \{1, 2, \dots, N\}$
- Andrei și Bogdan nu fac parte din grupul celor S elevi selectați;

Exemplu:

grup.in	grup.out	Explicații
7 4 3 2		Elevii selectați în grup sunt cei cu etichetele 1, 3, 5, 6.
1 2 3 4 5 6 7	15	Numarul total de absențe $Sm = 1+3+5+6 = 15$. Prietenii lui Andrei, selectați în grup, sunt 3, 5 și 6. Dușmanii lui Bogdan, selectați în grup, sunt 3 și 6.
0010110		
0011010		

Timp maxim de executare/test: **2.0** secunde pe Windows, **0.2** secunde pe Linux

Memorie: total **128 MB** din care pentru stivă **32 MB**

Dimensiune maximă a sursei: **20 KB**

20.5.1 Indicații de rezolvare

Chiriac Andrei-Alexandru - Universitatea București, Facultatea de matematică și informatică
Adrian Budău - Universitatea București, Facultatea de matematică și informatică

Soluție $O(n^3)$

Elevii se împart în 4 grupe principale :

- Grupa Inamici, formată din elevii care sunt și inamicii lui Andrei, și inamicii lui Bogdan.
- Grupa A , formată din elevii care sunt simultan prietenii lui Andrei și inamicii lui Bogdan.
- Grupa B , formată din elevii care sunt simultan prietenii lui Bogdan și inamicii lui Andrei.
- Grupa AB , formată din elevii care sunt și prietenii lui Andrei, și prietenii lui Bogdan.

Împărțim cei N elevi în aceste patru grupe și sortăm crescător elevii fiecărei grupe după numărul de absențe și calculăm sumele parțiale ale acestor vectori.

Fixăm toate posibilitățile pentru căi elevi alegem din grupele A, B și AB . Numărul de elevi aleși din grupa *Inamici* va fi diferența de elevi până la S . Aflăm în $O(1)$ suma valorilor elevilor folosind sumele parțiale și actualizăm soluția.

Soluție $O(n^2)$

Vom fixa numărul elevilor aleși în soluție care fac parte din Grupa AB , notat cu x ($x \geq K1 - A.size()$ și $x \geq K2 - B.size()$).

Suntem obligați să alegem $K1 - x$ elevi din grupa A și $K2 - x$ elevi din grupa B , iar pe restul de $T = (S - K1 - K2 - x)$ îi putem alege din elementele încă neselectate din Grupe Inamici, A și B .

Astfel, menținem pointerii $posInamici$, $posA$ și $posB$, inițializați cu $posInamici = 0$, $posA = K1 - x$, $posB = K2 - x$ și la fiecare din cei T pași alegem valoarea minimă dintre $Inamici[posInamici + 1]$, $A[posA + 1]$ și $B[posB + 1]$, având grijă să nu se depășească dimensiunea grupelor.

Soluție $O(n * \log n * \log n)$

Similar soluției precedente, la ultimul pas, în loc să iterăm prin toți copiii rămași, putem căuta binar valoarea ultimului copil ales din cei T . La fiecare pas al căutării binare, pentru o valoare V , căutăm binar prin cele 3 grupe câte elemente mai mici sau egale decât V conține fiecare. Dacă în total avem cel puțin T elemente mai mici sau egale decât V , căutam un V mai mic, altfel, căutăm un V mai mare.

În acest fel, după aflarea V -ului minim, știm că în soluție vom alege toate valorile mai mici strict decât V și apoi surplusul de elemente până la T vor avea valoarea V , deci știm suma lor.

Soluție $O(n)$

Aflăm soluția optimă pentru cel mai mic x valid în timp liniar, ca în soluția $O(n^2)$, iar trecerea de la x la $x + 1$ o procesăm în $O(1)$ astfel:

Pentru x , soluția este descrisă de tupletul $(hA, hB, hInamici)$, care reprezintă câți elevi alegem din grupele A , B și $Inamici$. La trecerea de la x la $x + 1$, numărul total de elevi crește cu 1, deci va trebui să scoatem un elev din totalul primelor 3 grupe. Nu are sens să scoatem mai mult de 1 elev din fiecare grupă, pentru că altfel, am fi făcut acest lucru la pasul precedent, deci este de ajuns să scoatem ultimul elev din fiecare grupă, iar apoi să adăugăm la soluție minimul dintre pretendenți până ajungem la T elevi în total (maxim 2 adăugări).

20.5.2 *Rezolvare detaliată

20.5.3 Cod sursă

Listing 20.5.1: grup.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <algorithm>
5 #include <cassert>
6 #include <climits>
7
8 using namespace std;
9
10 int main()
11 {
12     ifstream cin("grup.in");
13     ofstream cout("grup.out");
14
15     int N, S, K1, K2;
16     assert(cin >> N >> S >> K1 >> K2);
17     assert(1 <= N && N <= 100 * 1000);
18     assert(0 <= K1 && K1 <= S);
19     assert(0 <= K2 && K2 <= S);
20     assert(1 <= S && S <= N);
21
22     vector<int> A(N);
23
24     for (int i = 0; i < N; ++i)
25     {
26         assert(cin >> A[i]);
27         assert(1 <= A[i] && A[i] <= 1000 * 1000 * 1000);
28     }
29
30     string friendsA, friendsB;
31     assert(cin >> friendsA >> friendsB);
32     assert(int(friendsA.size()) == N);
33     assert(int(friendsB.size()) == N);

```

```

34
35     for (auto &c : friendsA)
36         assert(c == '0' || c == '1');
37
38     for (auto &c : friendsB)
39     {
40         assert(c == '0' || c == '1');
41         c ^= '0' ^ '1';
42     }
43
44     K2 = S - K2;
45     vector<int> have[4];
46     for (int i = 0; i < N; ++i)
47     {
48         int mask = 0;
49         if (friendsA[i] == '1')
50             mask ^= 1;
51         if (friendsB[i] == '1')
52             mask ^= 2;
53         have[mask].push_back(A[i]);
54     }
55
56     for (int i = 0; i < 4; ++i)
57         sort(have[i].begin(), have[i].end());
58
59     int x[4] = {0, 0, 0, 0};
60     int64_t total_sum = 0;
61     int64_t answer = numeric_limits<int64_t>::max();
62
63     for (x[3] = -1; x[3] <= S && x[3] < int(have[3].size()); )
64     {
65         if (x[3] == -1)
66             ++x[3];
67         else
68             total_sum += have[3][x[3]++;
69
70         if (K1 - x[3] > int(have[1].size()))
71             continue; // not enough friends for A
72
73         if (K2 - x[3] > int(have[2].size()))
74             continue; // not enough friends for B
75
76         if (K1 - x[3] + K2 - x[3] + x[3] > S) // need to many people
77             continue;
78
79         if(x[3]+int(have[0].size())+int(have[1].size())+int(have[2].size())<S)
80             // not enough people to get S
81             continue;
82
83         for (int i = 0; i < 3; ++i)
84             if (x[i] > 0)
85                 total_sum -= have[i][--x[i]];
86
87         while (x[3] + x[1] < K1)
88             total_sum += have[1][x[1]++;
89
90         while (x[3] + x[2] < K2)
91             total_sum += have[2][x[2]++;
92
93         while (x[0] + x[1] + x[2] + x[3] < S)
94         {
95             int best = numeric_limits<int>::max();
96             int where = 0;
97             for (int j = 0; j < 3; ++j)
98                 if (x[j] < int(have[j].size()) && have[j][x[j]] < best)
99                 {
100                     best = have[j][x[j]];
101                     where = j;
102                 }
103             total_sum += have[where][x[where]++;
104         }
105
106         answer = min(answer, total_sum);
107     }
108
109     if (answer == numeric_limits<int64_t>::max())

```

```

110     answer = 1LL << 62;
111
112     cout << answer << "\n";
113 }
```

20.6 proiectoare

Problema 6 - proiectoare

100 de puncte

Primăria a montat, pe faleza din Mamaia, N proiectoare aşezate liniar, pentru fiecare cunoscându-se zona de faleză pe care o luminează, sub forma unui interval $[s, d]$, unde s și d ($s < d$) sunt numere naturale reprezentând distanțele față de punctul unde începe faleza. Pentru a verifica eficiența iluminării falezei, tehnicienii primăriei vor să determine intervalul de faleză de lungime maximă, iluminat de cel mult K proiectoare, conținut într-un interval $[X, Y]$ precizat. Pentru a fi siguri de corectitudinea rezultatelor obținute, tehnicienii realizează Q astfel de verificări.

Cerințe

Dându-se Q intervale de forma $[X_i, Y_i]$ determinați, pentru fiecare dintre acestea, câte un interval de lungime maximă iluminat de cel mult K proiectoare. Dacă nici un proiectoare nu luminează vreo porțiune din intervalul $[X_i, Y_i]$ se va afișa valoarea 0.

Date de intrare

Datele de intrare se citesc din fișierul text **proiectoare.in**, care are structura următoare:

- pe prima linie se află valorile naturale N , Q , K , separate prin câte un spațiu, cu semnificația din enunț;
- pe următoarele N linii se află câte o pereche de valori naturale S_i D_i , separate printr-un spațiu, reprezentând intervalele iluminate de fiecare proiectoare;
- pe următoarele Q linii se află câte o pereche de valori naturale X_i Y_i , separate printr-un spațiu, reprezentând intervalele pentru care se realizează verificările.

Date de ieșire

În fișierul text **proiectoare.out** se vor scrie Q linii; pe linia i ($1 \leq i \leq Q$) se va scrie un număr natural reprezentând lungimea intervalului obținut ca răspuns la verificarea efectuată pentru intervalul $[X_i, Y_i]$.

Restricții și precizări

- $0 \leq S < D \leq 1\,000\,000\,000$
- $1 \leq K \leq N \leq 100\,000$
- $1 \leq Q \leq 100\,000$
- Pentru teste în valoare de 11 puncte, $K = 1$ și $n \leq 2000$ și $Q \leq 2000$
- Pentru teste în valoare de alte 38 puncte, $K = 1$
- Pentru teste în valoare de alte 21 puncte, $K = 2$
- Pentru teste în valoare de alte 10 puncte, $K \leq 30$

Exemple:

proiectoare.in	proiectoare.out	Explicații
5 5 1	4	Pentru verificarea $[1,10]$ cel mai lung interval complet iluminat este $[4,8]$ cu lungimea 4.
1 4	2	Pentru verificarea $[2,5]$ cele mai lungi intervale complet iluminate sunt $[2,4]$ și $[3,5]$, ambele au lungimea 2.
2 3	1	Pentru verificarea $[3,4]$ cel mai lung interval complet iluminat este $[3,4]$ cu lungimea 1.
3 6	2	Pentru verificarea $[6,8]$ cel mai lung interval complet iluminat este $[6,8]$ cu lungimea 2.
4 7	0	Pentru verificarea $[8,9]$ se afișează valoarea 0.
4 8		
1 10		
2 5		
3 4		
6 8		
8 9		

Timp maxim de executare/test: **3.0** secunde pe Windows, **0.8** secunde pe Linux

Memorie: total **128 MB** din care pentru stivă **32 MB**

Dimensiune maximă a sursei: **20 KB**

20.6.1 Indicații de rezolvare

Adrian Budău - Universitatea București, Facultatea de matematică și informatică

În primul și în primul rând, trebuie observat că niciodată nu merită folosit un proiecto $[u, v]$ inclus într-un alt proiecto $[a, b]$, adică $a \leq u \leq v \leq b$ astfel, folosind o *stivă*, se pot obține toate intervalele neincluse în alte intervale mai lungi.

Cazul $K = 1$

Se poate observa că răspunsul pentru query-ul $[x, y]$ este fie un prefix de forma $[x, z]$, cu $z \leq y$, fie un sufix $[z, y]$, cu $x \leq z$, fie este complet inclus, adică $[u, v]$, cu $x < u < v < y$.

Pentru a determina răspunsul prefix (și analog sufix) se poate *cauta binar* în intervalele rămase după eliminarea de mai sus intervalului care are capătul din stânga cel mai mare, dar încă mai mic sau egal cu x , acest interval având indicele $i1$ după ordonare. Analog, pentru suffix, se obține indicele $i2$, evident cu $i1 < i2$.

Rămâne astfel de tratat cazul în care răspunsul este un interval complet inclus în $[x, y]$ și se poate observa foarte ușor că toate intervalele cu indici de la $i1 + 1$ la $i2 - 1$ respectă această proprietate și dintre acestea il vom alege pe cel mai lung. Problema se reduce la întrebări de forma: "care este valoarea maxima a unei valori din intervalul $(i1 + 1, i2 - 1)$, unde valorile sunt lungimile intervalelor ?"

Aceste întrebări se pot rezolva "offline" folosind *divide et impera*. La un pas (`begin, end`) cu $mid = (\text{begin} + \text{end}) / 2$ se rezolvă recursiv toate întrebările complet incluse în (begin, mid) și $(mid + 1, \text{end})$, iar apoi toate întrebările care conțin elemente din ambele jumătăți, folosind o preprocesare de maxim parțial.

Complexitatea acestei soluții este $O(N \log N)$ pentru *sortarea* intervalelor, plus $O(Q \log N)$ pentru determinarea întrebărilor ce trebuie puse, plus $O((Q + N) \log N)$ pentru determinarea răspunsurilor folosind *divide et impera*, deci complexitatea finală este $O((Q + N) \log N)$.

Cazul $K = 2$

Se poate observa că, odata stabilit că se folosește un proiecto $[u1, v1]$ ca soluție într-un query cu $K = 2$, iar acest proiecto este cel mai din stânga ales, atunci clar cel mai din dreapta ales este acel proiecto $[u2, v2]$ cu $u2 <= v1$, indiferent cum arată query-ul. Astfel, se poate transforma orice proiecto $[u1, v1]$ în $[u1, v2]$ cu $v2$ din definiția de mai sus. Pentru asta se folosesc indecsă care se incrementează în parallel, iar după această transformare rezolvarea este identică cu $K = 1$. Complexitatea este tot $O((Q + N) \log N)$.

Cazul $K \leq 30$

Se poate aplica rationamentul de la $K = 2$ de K ori și astfel complexitatea este $O(N * K + (Q + N) \log N)$

Cazul $K \leq N$

Se poate iarăși aplica raționamentul de la $K \leq 30$, dar cu ideea de la *exponențierea rapidă*. Astfel, dacă avem K impar, se extinde un interval obținut din $K - 1$ reuniuni, cu alt interval, altfel se extinde un interval obținut din $K / 2$ reuniuni tot cu unul obținut din $K / 2$ reuniuni. Complexitatea finală este $O(N \log K + (Q + N) \log N)$.

20.6.2 *Rezolvare detaliată

20.6.3 Cod sursă

Listing 20.6.1: proiectoare.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <algorithm>
5 #include <cassert>
6
7 using namespace std;
8

```

```

9  vector< pair<int, int> > remove_inner(vector< pair<int, int> > V)
10 {
11     vector< pair<int, int> > answer;
12     for (auto &p : V)
13     {
14         while (!answer.empty() &&
15             answer.back().first >= p.first &&
16             answer.back().second <= p.second)
17             answer.pop_back();
18         answer.push_back(p);
19     }
20     return answer;
21 }
22
23 template<class T>
24 const T& pick(T& first, T& second, bool condition)
25 {
26     if (condition)
27         return first;
28     else
29         return second;
30 }
31
32 int intersect(int x1, int y1, int x2, int y2)
33 {
34     int x = max(x1, x2);
35     int y = min(y1, y2);
36     if (x <= y)
37         return y - x;
38     return 0;
39 }
40
41 struct Query
42 {
43     int index, from, to;
44 };
45
46 template<class Iterator>
47 void divide_et_impera(const vector< pair<int, int> > &V,
48                         int begin,
49                         int end,
50                         Iterator query_begin,
51                         Iterator query_end,
52                         vector<int> &buffer,
53                         vector<int> &answer)
54 {
55     if (end - begin == 1)
56     { // we done
57         for (auto it = query_begin; it != query_end; ++it)
58             answer[it->index] = max(answer[it->index],
59                                     V[begin].second - V[begin].first);
60     }
61 }
62
63     int mid = (begin + end) / 2;
64
65     auto to_answer=partition(query_begin, query_end, [&](const Query& query) {
66         return query.to < mid || query.from >= mid;
67     });
68
69     // precalculate
70     for (int i = mid - 1; i >= begin; --i)
71     {
72         buffer[i] = V[i].second - V[i].first;
73         if (i + 1 < mid)
74             buffer[i] = max(buffer[i], buffer[i + 1]);
75     }
76
77     for (int i = mid; i < end; ++i)
78     {
79         buffer[i] = V[i].second - V[i].first;
80         if (i > mid)
81             buffer[i] = max(buffer[i], buffer[i - 1]);
82     }
83
84     for (auto it = to_answer; it != query_end; ++it)

```

```

85         answer[it->index] = max(answer[it->index],
86                                 max(buffer[it->from], buffer[it->to]));
87
88     // partition again
89     auto query_mid=partition(query_begin, to_answer, [&](const Query& query) {
90         return query.to < mid;
91     });
92
93     divide_et_impera(V, begin, mid, query_begin, query_mid, buffer, answer);
94     divide_et_impera(V, mid, end, query_mid, to_answer, buffer, answer);
95 }
96
97 int main()
98 {
99     ifstream cin("proyectoare.in");
100    ofstream cout("proyectoare.out");
101
102    int N, Q, K;
103    assert(cin >> N >> Q >> K);
104    assert(1 <= N && N <= 100000);
105    assert(1 <= Q && Q <= 100000);
106    assert(1 <= K && K <= N);
107
108    vector< pair<int, int> > V(N);
109    for (int i = 0; i < N; ++i)
110    {
111        assert(cin >> V[i].first >> V[i].second);
112        assert(0 <= V[i].first &&
113              V[i].first <= V[i].second &&
114              V[i].second <= 1000 * 1000 * 1000);
115    }
116
117    sort(V.begin(), V.end(), [&](pair<int, int> a, pair<int, int> b) {
118        if (a.second != b.second)
119            return a.second < b.second;
120        return a.first > b.first;
121    });
122
123    V = remove_inner(V);
124
125    vector<int> step1(V.size());
126
127    int index = 0;
128    for (int i = 0; i < int(V.size()); ++i)
129    {
130        while (index < int(V.size()) && V[index].first <= V[i].second)
131            ++index;
132        step1[i] = index;
133    }
134
135    auto current = step1;
136
137    // recompute the intervals using K at a time
138
139    int oldK = K--;
140    vector<bool> to_double;
141
142    while (K > 1)
143    {
144        if (K % 2)
145        {
146            to_double.push_back(false);
147            K--;
148        }
149        else
150        {
151            to_double.push_back(true);
152            K /= 2;
153        }
154    }
155
156    reverse(to_double.begin(), to_double.end());
157
158    for (auto op : to_double)
159    {
160        auto& extend = pick(current, step1, op);

```

```

161         vector<int> next;
162
163     for (int i = 0; i < int(V.size()); ++i)
164         next.emplace_back(extend[current[i] - 1]);
165
166     current = next;
167 }
168
169 if (oldK > 1)
170 {
171     vector< pair<int, int> > newV;
172
173     for (int i = 0; i < int(V.size()); ++i)
174         newV.emplace_back(V[i].first, V[current[i] - 1].second);
175
176     V = newV;
177 }
178
179 // now we good, we can finally solve problem
180
181 vector<int> answer(Q, 0);
182 vector<Query> need;
183
184 for (int i = 0; i < Q; ++i)
185 {
186     int x, y; assert(cin >> x >> y);
187     assert(0 <= x && x <= y && y <= 1000 * 1000 * 1000);
188
189     // find best prefix, best suffix
190     int prefix=lower_bound(V.begin(),V.end(),x,[&](pair<int,int> p,int x){
191         return p.first <= x;
192     }) - V.begin();
193     --prefix;
194
195     int suffix=lower_bound(V.begin(),V.end(),y,[&](pair<int,int> p,int y){
196         return p.second < y;
197     }) - V.begin();
198
199     if (prefix >= 0)
200         answer[i] = max(answer[i],
201                           intersect(V[prefix].first,V[prefix].second,x,y));
202     if (suffix < int(V.size()))
203         answer[i] = max(answer[i],
204                           intersect(V[suffix].first,V[suffix].second,x,y));
205     if (prefix + 1 < suffix)
206         need.emplace_back(Query{i, prefix + 1, suffix - 1});
207 }
208
209 vector<int> buffer(V.size(), 0);
210 divide_et_impera(V, 0, V.size(), need.begin(), need.end(), buffer, answer);
211
212 for (int i = 0; i < Q; ++i)
213     cout << answer[i] << "\n";
214 }
```

Capitolul 21

ONI 2017

21.1 multisum

Problema 1 - multisum

100 de puncte

Orice număr natural mai mare decât 2 poate fi scris ca sumă de numere naturale nenule **aflate în ordine strict crescătoare**, astfel încât orice termen al sumei, cu excepția primului termen, este un multiplu al termenului precedent din sumă. De exemplu, $27=3+6+18$, unde 6 este multiplul lui 3, iar 18 este multiplul lui 6. Cum se dorește o descompunere formată dintr-un număr cât mai mare de termeni, vom obține și descompunerii cu 4 termeni: $27=1+2+8+16$, $27=1+2+4+20$, $27=1+2+6+18$. Dintre cele trei descompunerii cu 4 termeni, descompunerea $27=1+2+4+20$ este minimă din punct de vedere lexicografic (1 și 2 sunt la fel în cele 3 descompunerii, dar $4 < 6$ și $4 < 8$). Numărul 30 poate fi descompus $30=2+4+8+16$. El are o descompunere tot de lungime 4, dar este mai mare din punct de vedere lexicografic decât oricare dintre descompunerile cu 4 termeni ale lui 27 ($2 > 1$).

Cerințe

Pentru mai multe seturi de date formate din câte două numere naturale A și B , $A \leq B$, se cere să se determine, pentru fiecare set una dintre următoarele cerințe:

- 1) numărul maxim de termeni în care pot fi descompuse numerele din intervalul $[A, B]$ după regula descrisă în enunț;
- 2) numărul de numere din intervalul $[A, B]$ care pot fi descompuse cu un număr maxim de termeni;
- 3) numărul din intervalul $[A, B]$ care admite o descompunere cu un număr maxim de termeni, minimă din punct de vedere lexicografic.

Date de intrare

Din fișierul **multisum.in** se citesc, de pe prima linie, două numere N și C , despărțite printr-un spațiu, reprezentând numărul de seturi de date și tipul cerinței: $C = 1$ pentru cerința 1, $C = 2$ pentru cerința 2 și $C = 3$ pentru cerința 3. De pe următoarele N linii ale fișierului se citește câte o pereche de numere A și B , separate printr-un spațiu.

Date de ieșire

În fișierul **multisum.out** se va scrie câte o linie pentru fiecare pereche A B din fișierul de intrare, linie care va contine numărul cerut, conform cerinței: dacă $C = 1$, numărul va reprezenta numărul maxim de termeni dintr-o descompunere, dacă $C = 2$, numărul va reprezenta numărul de valori din intervalul respectiv care pot fi descompuse cu un număr maxim de termeni, iar dacă $C = 3$, numărul va reprezenta acea valoare din intervalul respectiv care admite o descompunere cu un număr maxim de termeni, minimă din punct de vedere lexicografic.

Restricții și precizări

- $0 < N \leq 1000$
- $2 < A \leq B \leq 100\ 000$
- Suma lungimilor intervalelor din toate seturile unui test nu va depăși 100 000.

- Pentru rezolvarea corectă a cerinței 1, se acordă 20% din punctaj, pentru cerința 2 se acordă 40% din punctaj, iar pentru cerința 3 se acordă 40% din punctaj.
- Pentru teste în valoare de 30 de puncte, $N \leq 50$, $B \leq 1000$ și suma lungimilor intervalelor din teste nu va depăși 10 000.

Exemple:

multisum.in	multisum.out	Explicații
1 1 50 60	5	Există un singur set de date. Se rezolvă cerința 1. Descompunerile maximale ale numerelor din interval au unele 4 termeni, altele 5 termeni. Deci cel mai mare număr de termeni este 5 (și acesta se obține pentru numerele 55, 57, 58 și 59).
1 2 50 60	4	Există un singur set de date. Se rezolvă cerința 2. Numerele care se pot descompune într-un număr maxim de termeni sunt 55, 57, 58 și 59. Deci sunt 4 numere care admit o descompunere maximală.
1 3 50 60	55	Există un singur set de date. Se rezolvă cerința 3. Cele 4 numere care admit descompuneri maximale sunt: $55=1+2+4+16+32$, $55=1+2+4+12+36$, $55=1+2+4+8+40$ $57=1+2+6+12+36$, $58=1+3+6+12+36$, $59=1+2+8+16+32$ Cea mai mică sumă din punct de vedere lexicografic este $1+2+4+8+40$ și ea corespunde numărului 55.
3 3 50 50 10 13 16 17	50 11 17	Sunt 3 seturi de date. Se rezolvă cerința 3: Intervalul $[50, 50]$ conține doar numărul 50 care admite o singură descompunere maximală (cu 4 termeni) și aceasta este minimă din punct de vedere lexicografic. Dintre numerele din intervalul $[10, 13]$, numerele 10, 11 și 13 admit o descompunere cu un număr maxim de termeni (3 termeni), dar o descompunere maximală a lui 11 ($1+2+8$) este minimă din punct de vedere lexicografic. În intervalul $[16, 17]$ numerele admit câte două descompuneri maximale : $16=1+3+12$, $16=1+5+10$, $17=1+2+14$, $17=1+4+12$, iar descompunerea minimă din punct de vedere lexicografic este $1+2+14$ și corespunde valorii 17.

Timp maxim de executare/test: **1.0** secunde pe Windows, **0.4** secunde pe Linux

Memorie: total **16 MB**

Dimensiune maximă a sursei: **10 KB**

21.1.1 Indicații de rezolvare

Rodica Pintea

Putem scrie orice descompunere pentru un număr N:

$$a + a*k1 + a*k1*k2 + a*k1*k2*k3 + a*k1*k2*k3*k4 + \dots = \\ a * (1 + k1 + k1*k2 + k1*k2*k3 + k1*k2*k3*k4 + \dots)$$

Se observă că

$$1 + k1 + k1*k2 + k1*k2*k3 + k1*k2*k3*k4 + \dots$$

este și ea o descompunere validă pentru numărul N/a , unde a este un divizor al lui N .

În același timp, orice descompunere de forma

$$M = 1 + k1 + k1*k2 + k1*k2*k3 + k1*k2*k3*k4 + \dots$$

se poate scrie

$$1 + k1 * (k2 + k2*k3 + k2*k3*k4 + \dots).$$

Se observă că expresia

$$k2 + k2*k3 + k2*k3*k4 + \dots$$

este și ea o descompunere validă a numărului $(M - 1)/k1$, unde $k1$ este un divizor al lui $M - 1$.

Pentru a rezolva toate seturile de date, vom calcula iterativ lungimile maxime ale tuturor descompunerilor numerelor N de la 3 până la $BMAX = 100 000$ cu formula:

$$LMAX[i] = \max \begin{cases} \max\{LMAX[N/a] \text{ pentru orice divizor } a \text{ al lui } N\} \\ \max\{LMAX[(N-1)/b] \text{ pentru orice divizor } b \text{ al lui } N-1\} \end{cases}$$

Soluția de circa 70 puncte (complexitate $O(BMAX * \sqrt{BMAX})$)

Se determină divizorii lui N în complexitate \sqrt{N} .

Pentru cerința 3, putem determina recursiv un sir de elemente de maximum Log(i) valori, sir a cărui sumă este i și respectă proprietatea din enunț. Fie $Sol(i)$ funcția recursivă corespunzătoare. O soluție optimizată va sesiza dacă $Sol(j)$ necesar la un moment dat a mai fost apelat anterior și va refolosi răspunsul calculat anterior, în locul apelului recursiv. Complexitatea de memorie în acest caz este $O(BMAX * \log(BMAX))$.

Soluția de 100 puncte (complexitate $O(BMAX * \sqrt{BMAX} / \log(BMAX) + BMAX * \log(BMAX))$) - Patrick Sava)

Se poate optimiza soluția precedentă, observând că nu este nevoie de toți divizorii lui N , fiind necesari doar cei primi care se pot precalcula folosind *ciurul lui Eratostene*.

Soluția de 100 puncte (complexitate $O(BMAX * \log(BMAX))$) - Maria Smaranda Pandele/Adrian Budău)

Pentru orice i , se actualizează valorile $LMAX[i * k]$ și $LMAX[i * k + 1]$, $i * k$ reprezentând toți multiplii lui i mai mici sau egali cu $BMAX$.

Determinarea răspunsului la cerința 3 presupune memorarea pentru fiecare număr N , împreună cu lungimea maximă, a divizorului optim și a indicelui în sortarea lexicografică a tuturor numerelor care au aceeași lungime ca și N . Este nevoie de o implementare atentă întrucât divizorul optim trebuie întâi calculat doar pentru descompunerile de forma

$$1 + k_1 * (1 + k_2 * \dots (1 + k_x)),$$

și apoi, dacă nu există, să se calculeze pentru descompunerile de forma

$$k_1 * (1 + k_2 * (1 + k_3 * \dots (1 + k_x))).$$

Determinarea răspunsurilor la cerințele 1 și 2 se face parcurgând intervalele $[a, b]$ ale lui $LMAX$.

21.1.2 *Rezolvare detaliată

21.1.3 Cod sursă

Listing 21.1.1: multisum_100p_1.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <algorithm>
5 #include <tuple>
6
7 using namespace std;
8
9 int main()
10 {
11     ifstream cin("multisum.in");
12     ofstream cout("multisum.out");
13
14     int N, TYPE; cin >> N >> TYPE;
15     vector< pair<int, int> > V(N);
16
17     int maxB = 3;
18     for (int i = 0; i < N; ++i)
19     {
20         cin >> V[i].first >> V[i].second;
21         maxB = max(maxB, V[i].second);
22     }
23 }
```

```

24     vector<int> from(maxB + 1, 0), max_chain(maxB + 1, 0);
25     vector<int> first(maxB + 1, 0);
26
27     max_chain[1] = 1;
28     from[1] = 1;
29     first[1] = 1;
30
31     int best = 0;
32     for (int i = 1; i <= maxB; ++i)
33     {
34         if (max_chain[i] == 0)
35             continue;
36         for (int j = i * 2; j < maxB; j += i)
37         {
38             if (max_chain[i] + 1 >= max_chain[j + 1])
39             {
40                 max_chain[j + 1] = max_chain[i] + 1;
41                 from[j + 1] = j / i;
42                 first[j + 1] = 1;
43             }
44         }
45     }
46
47     for (int i = maxB; i > 0; --i)
48     {
49         best = max(best, max_chain[i]);
50         for (int j = i * 2; j <= maxB; j += i)
51         {
52             if (max_chain[i] > max_chain[j])
53             {
54                 max_chain[j] = max_chain[i];
55                 from[j] = j / i;
56                 first[j] = j / i;
57             }
58         }
59     }
60
61     vector<int> index(maxB + 1, -1);
62
63     index[1] = 0;
64     index[2] = 1;
65     for (int i = 2; i <= best; ++i)
66     {
67         vector<int> aux;
68         for (int j = 1; j <= maxB; ++j)
69             if (max_chain[j] == i)
70                 aux.push_back(j);
71
72         sort(aux.begin(), aux.end(),
73               [&](int pos1, int pos2)
74               {
75                 if (first[pos1] != first[pos2])
76                     return first[pos1] < first[pos2];
77                 if (from[pos1] != from[pos2])
78                     return from[pos1] < from[pos2];
79                 return index[pos1 / from[pos1]] < index[pos2 / from[pos2]];
80             });
81
82         for (int i = 0; i < int(aux.size()); ++i)
83             index[aux[i]] = i;
84     }
85
86     for (auto &q : V)
87     {
88         int a, b; tie(a, b) = q;
89
90         int best = 0;
91         int count = 0;
92         int lexicographic_minimum = a;
93         for (int i = a; i <= b; ++i)
94         {
95             if (max_chain[i] > best)
96             {
97                 best = max_chain[i];
98                 count = 1;
99                 lexicographic_minimum = i;

```

```

100         }
101     else
102         if (max_chain[i] == best)
103         {
104             ++count;
105             if (index[i] < index[lexicographic_minimum])
106                 lexicographic_minimum = i;
107         }
108     }
109
110     if (TYPE == 1)
111         cout << best << "\n";
112     else
113         if (TYPE == 2)
114             cout << count << "\n";
115         else
116             cout << lexicographic_minimum << "\n";
117     }
118 }
```

Listing 21.1.2: multisum_100p_2.cpp

```

1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4 #include <assert>
5
6
7 using namespace std;
8
9 const int N = 1e5 + 4;
10
11 int len[N], dp[N], m[N], pos[N];
12 vector<pair<int, int>> v[20];
13
14
15 inline int last(const pair<int, int> &A)
16 {
17     return (A.second) / m[A.second];
18 }
19
20 bool cmp (const pair<int, int> &A, const pair<int, int> &B)
21 {
22     if (A.first > B.first)
23         return true;
24     if (A.first < B.first)
25         return false;
26     if (m[A.second] < m[B.second])
27         return true;
28     if (m[A.second] > m[B.second])
29         return false;
30
31     return pos[last(A)] < pos[last(B)];
32 }
33
34 int main()
35 {
36     int a, b, x, maxim, startl, lim = 0, type;
37
38     ifstream cin ("multisum.in");
39     ofstream cout ("multisum.out");
40
41     b = 1e5;
42     len[1] = 1;
43     dp[1] = 1;
44     m[1] = 1;
45     for (int i = 1; i <= b; i++)
46     {
47         for (int j = 2 * i; j <= b; j = j + i)
48         {
49             if (len[j + 1] < 1 + len[i])
50             {
51                 len[j + 1] = 1 + len[i];
52                 dp[j + 1] = dp[i];
53                 m[j + 1] = j / i;
54             }
55         }
56     }
57 }
```

```

54         }
55     else
56         if (len[j + 1] == 1 + len[i])
57         {
58             dp[j + 1] = dp[j + 1] + dp[i];
59             m[j + 1] = min(m[j + 1], j / i);
60         }
61     }
62 }
63
64 for (int i = b; i >= 1; i--)
65 {
66     for (int j = 2 * i; j <= b; j = j + i)
67         if (len[j] < len[i])
68         {
69             len[j] = len[i];
70             dp[j] = dp[i];
71             m[j] = j / i;
72         }
73 }
74
75 for (int i = 1; i <= b; i++)
76 {
77     if (i % m[i])
78         start1 = 1;
79     else
80         start1 = 0;
81     v[len[i]].push_back(make_pair(start1, i));
82     lim = max(lim, len[i]);
83 }
84
85 v[1].push_back(make_pair(1, 1));
86 v[1].push_back(make_pair(0, 2));
87
88 pos[1] = 0;
89 pos[2] = 1;
90
91 for (int i = 2; i <= lim; i++)
92 {
93     sort(v[i].begin(), v[i].end(), cmp);
94     for (int j = 0; j < v[i].size(); j++)
95         pos[v[i][j].second] = j;
96 }
97
98 int T;
99 long long s = 0;
100 cin >> T >> type;
101
102 assert(T >= 1 && T <= 1000);
103
104 for (int t = 0; t < T; t++)
105 {
106     cin >> a >> b;
107
108     assert(3 <= a && a <= b && b <= 100000);
109     s = s + b - a + 1;
110     assert(s <= 100000);
111
112     int maxim = 0, cnt = 0, ans, minim = 0;
113     for (int i = a; i <= b; i++)
114     {
115         if (maxim < len[i])
116         {
117             maxim = len[i];
118             cnt = 1;
119             ans = i;
120             minim = pos[i];
121         }
122     else
123         if (maxim == len[i])
124         {
125             ++cnt;
126             if (pos[i] < minim)
127             {
128                 minim = pos[i];
129                 ans = i;
130             }
131         }
132     }
133 }
```

```

130             }
131         }
132     }
133
134     if (type == 1)
135         cout << maxim << "\n";
136     else
137         if (type == 2)
138             cout << cnt << "\n";
139         else
140             cout << ans << "\n";
141     }
142
143     assert(s <= 100000);
144     return 0;
145 }
```

Listing 21.1.3: multisum_100p_3.cpp

```

1 /*Patrick Catalin Alexandru Sava
2 Atoom Industry and Resources SRL
3 University of Bucharest
4 Expected : 100 points
5 */
6
7 #include <iostream>
8 #include <vector>
9 #include <algorithm>
10 #include <cassert>
11
12 using namespace std ;
13
14 ifstream cin ("multisum.in") ;
15 ofstream cout ("multisum.out") ;
16
17 vector <int> Ciur (int N)
18 {
19     vector <int> prim (N + 1, 1) ;
20     vector <int> primes ;
21
22     for (int i = 2 ; i <= N ; ++ i)
23         if (prim [i] == 1)
24         {
25             for (int j = 2 * i ; j <= N ; j += i)
26                 prim [j] = 0 ;
27             primes.push_back (i) ;
28         }
29
30     return primes ;
31 }
32
33 vector <int> Sol (int N, vector <int> &primes, vector <int> *&gr8)
34 {
35     if (gr8 [N].size()) return gr8 [N] ;
36     if (N == 1) gr8 [N] = vector <int> ({1}) ;
37     if (N == 2) gr8 [N] = vector <int> ({2}) ;
38     if (N == 3) gr8 [N] = vector <int> ({1, 2}) ;
39     if (gr8 [N].size()) return gr8 [N] ;
40
41     vector <int> best ;
42
43     for (auto x : primes)
44     {
45         if (x * x > N) break ;
46         if (N % x) continue ;
47
48         vector <int> temp = Sol (N / x, primes, gr8) ;
49
50         if (best.size() < temp.size())
51         {
52             best.clear() ;
53             for (auto y : temp)
54                 best.push_back (y * x) ;
55         }
56     }
57 }
```

```

57         if (best.size() == temp.size())
58             for (int i = 0 ; i < (int)best.size() ; ++ i)
59                 if (best [i] > x * temp [i])
60                 {
61                     best.clear() ;
62                     for (auto y : temp)
63                         best.push_back (y * x) ;
64                     break ;
65                 }
66             else
67                 if (best [i] < x * temp [i])
68                     break ;
69     }
70
71     N -- ;
72     if (best.size() <= 2)
73         best = vector <int> ({1, N}) ;
74
75
76     for (auto x : primes)
77     {
78         if (x * x > N) break ;
79         if (N % x) continue ;
80
81         vector <int> temp = Sol (N / x, primes, gr8) ;
82         reverse (temp.begin(), temp.end());
83
84         for (auto &y : temp)
85             y *= x ;
86
87         temp.push_back (1) ;
88         reverse (temp.begin(), temp.end());
89
90         if (best.size() < temp.size())
91             best = temp ;
92         else
93             if (best.size() == temp.size())
94                 for (int i = 0 ; i < (int)best.size() ; ++ i)
95                     if (best [i] > temp [i])
96                     {
97                         best = temp ;
98                         break ;
99                     }
100                else
101                    if (best [i] < temp [i])
102                        break ;
103    }
104
105    gr8 [N + 1] = best ;
106    return best ;
107 }
108
109 void Solve123 (vector <pair<int, int>> &Intervals, int type)
110 {
111     vector <int> primes = Ciur (1e5) ;
112
113     vector <int> *gr8 = new vector <int> [100001] ;
114
115     for (auto x : Intervals)
116     {
117         int best = 0 ;
118         int cate = 0 ;
119         for (int i = x.first ; i <= x.second ; ++ i)
120         {
121             if (Sol (i, primes, gr8).size() > best)
122             {
123                 best = Sol (i, primes, gr8).size() ;
124                 cate = 1 ;
125             }
126         else
127             if (Sol (i, primes, gr8).size() == best)
128                 cate += 1 ;
129     }
130
131     if (type == 1)
132         cout << best << '\n' ;

```

```

133     else
134     {
135         if (type == 2)
136             cout << cate << '\n' ;
137         else
138         {
139             vector <int> lexc (1, 1e9) ;
140             for (int i = x.first ; i <= x.second ; ++ i)
141                 if (gr8[i].size() == best)
142                     if (lexc.size() < gr8 [i].size())
143                         lexc = gr8 [i] ;
144                     else
145                         for (int j = 0 ; j < (int)lexc.size() ; ++ j)
146                             if (lexc [j] > gr8 [i][j])
147                             {
148                                 lexc = gr8 [i] ;
149                                 break ;
150                             }
151                         else
152                             if (lexc [j] < gr8 [i][j])
153                                 break ;
154
155             int s = 0 ;
156             for (auto y : lexc)
157                 s += y ;
158
159             cout << s << '\n' ;
160         }
161     }
162 }
163 }
164
165 int main(int argc, char const *argv[])
166 {
167     int n , c ;
168     assert(cin >> n >> c) ;
169     assert (0 < n and n <= 1000) ;
170     assert (c == 1 or c == 2 or c == 3) ;
171
172     vector <pair<int, int>> Intervals ;
173     int verif = 0 ;
174
175     for (int i = 1 ; i <= n ; ++ i)
176     {
177         int x, y ;
178         assert(cin >> x >> y) ;
179         verif += y - x + 1 ;
180         assert (x <= y) ;
181         assert (2 < x and x <= 100000) ;
182         assert (2 < y and y <= 100000) ;
183         Intervals.push_back ({x, y}) ;
184     }
185
186     assert (verif >= 1 and verif <= 100000) ;
187     Solve123 (Intervals, c) ;
188     return 0;
189 }
```

21.2 puzzle

Problema 2 - puzzle

100 de puncte

Definim un **puzzle de numere** ca fiind adunarea a **două numere naturale**, în care o parte dintre cifre au fost înlocuite cu caracterul *.

De exemplu, pentru adunarea:

9334

789

10123

unele dintre puzzle-urile corespunzătoare pot fi:

*3*4 9**4 ****

78* **9 ***

10123 ***** *****

Cerințe

Să se scrie un program care determină o adunare din care provine un puzzle dat.

Date de intrare

Fișierul de intrare **puzzle.in** va conține mai multe teste. Pe prima linie se va găsi un număr natural T reprezentând numărul de puzzle-uri din fișier. Pe următoarele $3 * T$ linii se vor găsi T triplete, fiecare triplet reprezentând un puzzle format din caractere * și eventual cifre.

Date de ieșire

Fișierul de ieșire **puzzle.out** va conține exact $3 * T$ linii cu numere naturale, câte trei linii pentru fiecare puzzle din fișierul de intrare. Prima și cea de-a doua linie a unui puzzle vor conține numerele care urmează să fie adunate, iar a treia linie va conține suma acestora, în ordinea citirii din fișierul de intrare.

Restricții și precizări

- $1 \leq T \leq 10$
- Toate numerele fiecărui puzzle nu pot avea prima cifră 0.
- Dacă există mai multe adunări corecte corespunzătoare unui puzzle, se va accepta oricare dintre acestea.
 - Lungimea oricărei linii a unui puzzle nu depășește 100.000 de caractere.
 - Se garantează existența unei soluții pentru toate testele de intrare.
 - Pentru teste în valoare de 15 puncte lungimea oricărui număr din fiecare puzzle va fi mai mică sau egal cu 18
 - Pentru teste în valoare de încă 25 de puncte lungimea oricărui număr din fiecare puzzle va fi mai mică sau egal cu 1 000
 - Pentru teste în valoare de încă 25 puncte lungimea oricărui număr din fiecare puzzle va fi mai mică sau egal cu 20 000

Exemple:

puzzle.in	puzzle.out	Explicații
1 *3*4 78* 10123	9334 789 10123	Fișierul de intrare conține un puzzle: *3*4 78* 10123 O adunare corectă corespunzătoare acestui puzzle este: 9334 789 10123
2 ** * ***7 75 * *6	98 9 107 75 1 76	Fișierul de intrare conține 2 puzzle-uri. Pentru primul puzzle o adunare corectă este: 98 9 107 Pentru al doilea puzzle o adunare corectă este: 75 1 76

Timp maxim de executare/test: **1.2** secunde sub Windows, **0.2** secunde sub Linux

Memorie: total **16 MB**

Dimensiune maximă a sursei: **10 KB**

21.2.1 Indicații de rezolvare

Cristina Anton, Adrian Budău

O observație necesară oricărei soluții mai eficiente este că odată aliniate la dreapta cele trei numere dintr-un puzzle, fiecare coloană este oarecum independentă de restul. Tot ce contează pentru o coloană este dacă s-a făcut sau nu transport de la coloana anterioară și dacă se face sau nu transport pe coloana următoare.

De exemplu pentru:

2*

7*

10*

Se știe că trebuie să se facă transport de pe a doua coloană din dreapta spre stânga spre a treia din dreapta spre stânga, și din acest motiv are nevoie de transport de la prima coloană din dreapta. Nu este important ce combinație de cifre s-a ales ca rezultat pentru prima coloană din dreapta, e important doar să aibă transport.

Pentru 40 de puncte $N \leq 1.000 - O(N * \sum^2 2 + N^2)$, $\sum = 10$ în această problemă, deoarece baza în care se lucrează este 10

Se poate menține parcurgând de la dreapta la stânga coloanele, o soluție (dacă există) pentru care se face transport la coloana următoare, și una pentru care nu se face transport (tot dacă există).

Pentru exemplul de mai sus o soluție pentru prima coloană care face transport este

9

5

4

și una care nu face transport este

1

2

3

Pentru a doua coloană o soluție în care se face transport se poate obține doar din una care face transport de pe prima coloană și deci este

29

75

04

iar una care nu face transport nu există.

Pentru ce-a de-a treia coloană, o soluție care face transport nu există, iar una care nu face transport este:

29

75

104

După acest procedeu se poate rezolva orice puzzle, și deci complexitatea finală este:

$O(N * \sum^2 + N^2)$

- $N * \sum^2$ pentru că pe fiecare coloană se încercă orice cifră posibilă pentru primul număr (daca e *), orice cifră posibilă pentru al doilea număr (tot dacă e *) și dacă este sau nu transport de la coloana precedenta și se verifică dacă se potrivește peste puzzle
- N^2 pentru că la fiecare coloană, avem o soluție de lungime cel mult N în fața careia se adaugă o coloană nouă

Pentru 65 de puncte $N \leq 20.000 - O(N * \sum^2)$, $\sum = 10$ în această problema, deoarece baza în care se lucrează este 10

Se poate scăpa de bucata de N^2 observând că în loc să se mențină efectiv soluția la fiecare pas, se poate ține minte doar cum s-a obținut soluția până la coloana respectivă (ce cifră s-a ales pentru fiecare număr și cu ce transport din coloana anterioară s-a obținut), și nu soluția însăși și la final să se plece de pe prima coloană și să se reconstruiască recursiv soluția.

Pentru 100 de puncte $O(N + \sum^3)$

Observația care duce la soluția de 100 de puncte este că pentru majoritatea coloanelor (mai puțin cele în care se află prima cifră a unuia din cele 3 numere) se poate precalcula exact cum să se obțină soluția pe acea coloană cu sau fără transport de la coloana anterioară, și cu sau fără transport spre coloana următoare. Precalcularea se poate face în $O(\sum^3)$, iar apoi complexitatea soluție este $O(N)$.

21.2.2 *Rezolvare detaliată

21.2.3 Cod sursă

Listing 21.2.1: puzzle_100p.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <array>
5 #include <algorithm>
6
7 using namespace std;
8
9 bool match(char x, char wanted, bool is_first)
10 {
11     if (is_first && x == 0)
12         return false;
13     return x == wanted || wanted == '*' ;
14 }
15
16 struct triplet
17 {
18     int a;
19     int b;
20     int r;
21 };
22
23 int main()
24 {
25     ifstream cin("puzzle.in");
26     ofstream cout("puzzle.out");
27
28     int T;
29     cin >> T;
30     while (T--)
31     {
32         string A, B, C; cin >> A >> B >> C;
33
34         // A + B = C
35         reverse(A.begin(), A.end());
36         reverse(B.begin(), B.end());
37         reverse(C.begin(), C.end());
38
39         int N = A.size();
40         int M = B.size();
41         int K = C.size();
42
43         vector<triplet> how[2];
44         how[0] = how[1] = vector<triplet>(K, triplet{-1, -1, -1});
45
46         array< array< array< array< triplet, 11>, 11>, 11>, 2>, 2>
47             precalc_how;
48
49         for (int r1 = 0; r1 < 2; ++r1)
50             for (int r2 = 0; r2 < 2; ++r2)
51                 for (int a = 0; a <= 10; ++a)
52                     for (int b = 0; b <= 10; ++b)
53                         for (int c = 0; c <= 10; ++c)
54                         {
55                             precalc_how[r1][r2][a][b][c] = triplet{-1,-1,-1};
56                         }
57
58         for (int remainder = 0; remainder < 2; ++remainder)
59             for (int a = 0; a < 10; ++a)
60                 for (int b = 0; b < 10; ++b)
61                 {
62                     int c = (remainder + a + b) % 10;
63                     int new_rem = (remainder + a + b) / 10;
64                     for (auto x : {a, 10})
65                         for (auto y : {b, 10})

```

```

66             for (auto z : {c, 10})
67                 precalc_how[remainder][new_rem][x][y][z] =
68                     triplet{a, b, remainder};
69         }
70
71     for (int i = 0; i < K; ++i)
72     {
73         if (i == N - 1 || i == M - 1 || i == K - 1)
74         {
75             int startA = 0, endA = 9, startB = 0, endB = 9;
76             if (i >= N)
77                 startA = endA = 0;
78             else
79                 if (A[i] != '*')
80                     startA = endA = A[i] - '0';
81
82             if (i == N - 1)
83                 startA = max(startA, 1);
84
85             if (i >= M)
86                 startB = endB = 0;
87             else
88                 if (B[i] != '*')
89                     startB = endB = B[i] - '0';
90
91             if (i == M - 1)
92                 startB = max(startB, 1);
93
94             triplet prevhow[2] = {triplet{0, 0, 0}, triplet{-1, -1, -1}};
95
96             if (i > 0)
97             {
98                 prevhow[0] = how[0][i - 1];
99                 prevhow[1] = how[1][i - 1];
100            }
101
102            for (int a = startA; a <= endA; ++a)
103                for (int b = startB; b <= endB; ++b)
104                    for (int remainder = 0; remainder < 2; ++remainder)
105                        if (match((a + b + remainder) % 10 + '0',
106                                  C[i], i == K - 1))
107                            if (prevhow[remainder].r != -1)
108                                how[(a + b + remainder) / 10][i] =
109                                    triplet{a, b, remainder};
110            }
111        else
112        {
113            // we have it all precalculated
114            int a = 0;
115            if (i < N)
116            {
117                a = A[i] - '0';
118                if (A[i] == '*')
119                    a = 10;
120            }
121
122            int b = 0;
123            if (i < M)
124            {
125                b = B[i] - '0';
126                if (B[i] == '*')
127                    b = 10;
128            }
129
130            int c = C[i] - '0';
131            if (C[i] == '*')
132                c = 10;
133
134            triplet prevhow[2] = {triplet{0, 0, 0}, triplet{-1, -1, -1}};
135
136            if (i > 0) {
137                prevhow[0] = how[0][i - 1];
138                prevhow[1] = how[1][i - 1];
139            }
140
141            for (int remainder = 0; remainder < 2; ++remainder)

```

```

142         for (int new_rem = 0; new_rem < 2; ++new_rem)
143             if(precalc_how[remainder][new_rem][a][b][c].a != -1 &&
144                 prevhow[remainder].a != -1)
145                 how[new_rem][i] =
146                     precalc_how[remainder][new_rem][a][b][c];
147     }
148 }
149
150 int remainder = 0;
151 for (int i = K - 1; i >= 0; --i)
152 {
153     if (i < N)
154         A[i] = how[remainder][i].a + '0';
155     if (i < M)
156         B[i] = how[remainder][i].b + '0';
157
158     C[i] = (how[remainder][i].a +
159             how[remainder][i].b +
160             how[remainder][i].r) % 10 + '0';
161
162     remainder = how[remainder][i].r;
163 }
164
165 reverse(A.begin(), A.end());
166 reverse(B.begin(), B.end());
167 reverse(C.begin(), C.end());
168
169 cout << A << "\n" << B << "\n" << C << "\n";
170 }
171 }
```

21.3 tăietura

Problema 3 - tăietura

100 de puncte

Fiind dat un sir V format din N numere întregi V_1, \dots, V_N , definim o tăietură în poziția pos ca fiind o subsecvență care conține elementul de pe poziția pos . Formal, tăieturile în poziția pos sunt de forma $V_k, V_{k+1}, \dots, V_{pos}, \dots, V_{r-1}, V_r$ pentru orice k , $1 \leq k \leq pos$ și orice r , $pos \leq r \leq N$.

Valoarea unei tăieturi este suma tuturor elementelor care fac parte din tăietura respectivă.

Definim funcția $MulT(pos)$ ca fiind numărul de tăieturi în poziția pos care au valoarea 0.

Cerințe

Ioana, fiind foarte curioasă din fire, dar și foarte fascinată de această funcție numită $MulT$, este foarte interesată în a afla rezultatul pentru $MulT(i)$, unde $1 \leq i \leq N$.

Date de intrare

Fișierul de intrare **tăietura.in** conține pe prima linie un număr natural N , reprezentând numărul de elemente din sirul V . Următoarea linie va conține exact N valori întregi despărțite prin câte un spațiu, și anume elementele sirului V .

Date de ieșire

Fișierul de ieșire **tăietura.out** va conține pe prima linie N numere naturale separate prin câte un spațiu, și anume valorile funcției $MulT(i)$, unde $1 \leq i \leq N$.

Restricții și precizări

- $1 \leq N \leq 100\,000$;
- Orice element al sirului V este mai mic sau egal în valoare absolută cu 10^9 .
- Pentru teste în valoare de 20 de puncte $N \leq 100$
- Pentru teste în valoare de încă 20 de puncte $N \leq 1000$

Exemple:

taietura.in	taietura.out	Explicații
3 0 1 0	1 0 1	Rezultatul pentru $\text{MulT}(1)$ este 1 deoarece există o singură tăietură, și anume (0) care are valoarea 0. Pentru $\text{MulT}(2)$ rezultatul este 0 deoarece nu există nicio tăietură aplicată pe poziția 2 care să aibă valoarea 0. Rezultatul pentru $\text{MulT}(3)$ este 1 deoarece există o unică tăietură, și anume (0) care are valoarea 0.
6 2 -2 0 0 1 -1	4 4 6 6 4 4	De exemplu, rezultatul pentru $\text{MulT}(2)$ este 4 deoarece tăieturile formate din subsecvențele (2, -2), (2, -2, 0), (2, -2, 0, 0), (2, -2, 0, 0, 1, -1) au valoarea 0.

Timp maxim de executare/test: **1.0** secunde pe Windows **0.2** secunde pe Linux

Memorie: total **16 MB**

Dimensiune maximă a sursei: **10 KB**

21.3.1 Indicații de rezolvare

Patrick Sava, Universitatea din Bucuresti;
Adrian Budău, Universitatea din București

Pentru 20 de puncte:

Calculăm *sumele parțiale* pe sirul dat iar apoi iterăm pentru fiecare poziție cât de mult ne extindem în stânga și cât de mult ne extindem în dreapta. Odată fixate aceste trei valori (poziția, capătul stâng, respectiv capătul drept) putem determina în timp constant dacă tăietura trebuie numărată sau nu. Complexitate $O(N^3)$, unde N se referă la numărul de elemente din input.

Pentru 40 de puncte:

Tot având *sirul sumelor parțiale* calculat, putem ca pentru fiecare poziție i să introducem într-un sir toate sumele parțiale care se termină pe poziția i și pentru fiecare sumă parțială S care începe pe poziția $i+1$, să se caute în acest sir de câte ori există S (de exemplu, cu căutare binară). Complexitate $O(N^2)$ sau $O(N^2 \log^2 N)$.

Pentru 40 - 50 de puncte:

Calculăm sirul sumelor parțiale, și îl *normalizăm* (atribuim fiecarui element indicele său în ordine sortată în loc de valoarea inițială). Acum, valoarea maximă din sirul sumelor parțiale va fi cel mult N . Dacă privim mai atent sirul sumelor, două poziții diferite din acesta care au aceeași valoare înseamnă că produc o subsecvență de sumă 0. Această subsecvență va incrementa valoarea funcției MulT pentru fiecare element din intervalul cuprins între cele două poziții alese. Pentru a "salva" un ordin de mărime al complexității, putem să actualizăm într-un sir T doar în capetele intervalului (adăugăm 1 pe poziția de început a intervalului și scădem 1 de pe poziția succesoare a sfârșitului intervalului). La final, dacă facem sumele parțiale pe T unde doar am incrementat și decrementat, vom obține cele N rezultate cerute. Complexitate $O(N^2)$. Această soluție se comportă mai bine în practică.

Pentru 100 de puncte:

Se poate optimiza soluția anterioară, observând că pentru o sumă parțială care se găsește pe poziția i în sirul T , vom incrementa de x ori și vom decrementa de y ori, unde y reprezintă numărul aparițiilor acelei sume parțiale înainte de poziția i și x reprezintă numărul aparițiilor acelei sume parțiale după poziția i . Complexitate $O(N \log N)$.

21.3.2 *Rezolvare detaliată

21.3.3 Cod sursă

Listing 21.3.1: taietura_100p_1.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <algorithm>
5 #include <cassert>
6
7 using namespace std;
8
9 int main()
10 {
11     ifstream cin("taietura.in");
12     ofstream cout("taietura.out");
13
14     int N;
15     assert(cin >> N);
16     assert(1 <= N && N <= 200 * 1000);
17
18     vector<int> V(N + 1, 0);
19
20     for (int i = 1; i <= N; ++i)
21     {
22         assert(cin >> V[i]);
23         assert(-1000 * 1000 * 1000 <= V[i] && V[i] <= 1000 * 1000 * 1000);
24     }
25
26     vector<int64_t> partial_sum(N + 1, 0);
27
28     for (int i = 1; i <= N; ++i)
29         partial_sum[i] = partial_sum[i - 1] + V[i];
30
31     auto sums = partial_sum;
32
33     sort(sums.begin(), sums.end());
34
35     sums.erase(unique(sums.begin(), sums.end()), sums.end());
36
37     vector<int64_t> mars(N + 1, 0);
38     vector<int> count(sums.size(), 0);
39
40     for (int i = 0; i <= N; ++i)
41     {
42         int sum_index = lower_bound(sums.begin(), sums.end(), partial_sum[i]) -
43                         sums.begin();
44         int to_the_left = count[sum_index];
45         if (i < N)
46             mars[i + 1] -= to_the_left;
47         ++count[sum_index];
48     }
49
50     for (int i = 0; i < N; ++i)
51     {
52         int sum_index = lower_bound(sums.begin(), sums.end(), partial_sum[i]) -
53                         sums.begin();
54         --count[sum_index];
55         int to_the_right = count[sum_index];
56         mars[i + 1] += to_the_right;
57     }
58
59     int64_t answer = 0;
60     for (int i = 1; i <= N; ++i)
61     {
62         answer += mars[i];
63         cout << answer << " ";
64     }
65     cout << "\n";
66 }
```

Listing 21.3.2: taietura_100p_2.cpp

```

1 /*Patrick Catalin Alexandru Sava
2 Atooom Industry and Resources SRL
3 University of Bucharest
4 Expected : 100
```

```

5  O (N log N)
6  */
7
8 #include <iostream>
9 #include <cassert>
10 #include <vector>
11 #include <unordered_map>
12 #include <algorithm>
13
14 using namespace std ;
15
16 ifstream cin ("taietura.in") ;
17 ofstream cout ("taietura.out") ;
18
19 const int MAX = 2e5 ;
20 const int VALMAX = 2e9 ;
21
22 int main(int argc, char const *argv[])
23 {
24     int n ;
25     cin >> n ;
26     assert (n >= 1 and n <= MAX) ;
27
28     vector <long long> sp (n + 1, 0) ;
29     vector <int> v (n + 1, 0) ;
30
31     for (int i = 1 ; i <= n ; ++ i)
32     {
33         cin >> v [i] ;
34         assert (-VALMAX <= v [i] and v [i] <= VALMAX) ;
35         sp [i] = v [i] ;
36         sp [i] += sp [i - 1] ;
37     }
38
39     sort (sp.begin(), sp.end()) ;
40
41     unordered_map <long long, int> H ;
42
43     int e = 1 ;
44     long long last = sp [0] ;
45     for (auto x: sp)
46     {
47         if (x == last)
48             H [x] = e ;
49         else
50         {
51             ++ e ;
52             H [x] = e ;
53             last = x ;
54         }
55     }
56
57     sp [0] = 0 ;
58     for (int i = 1 ; i <= n ; ++ i)
59     {
60         sp [i] = v [i] ;
61         sp [i] += sp [i - 1] ;
62     }
63
64     vector <int> freqv (e + 1, 0) ;
65     vector <int> dr (e + 1, 0) ;
66
67     for (int i = 0; i <= n ; ++ i)
68     {
69         freqv [H[sp[i]]] += 1 ;
70         dr [H[sp[i]]] += 1 ;
71     }
72
73     vector <long long> mars (n + 1, 0) ;
74
75     for (int i = 0 ; i < n; ++ i)
76     {
77         dr [H[sp[i]]] -= 1 ;
78         mars [i + 1] += dr [H[sp[i]]] ;
79         mars [i + 1] -= freqv [H[sp[i]]] - dr [H[sp[i]]] - 1 ;
80     }

```

```

81
82     for (int i = 1 ; i <= n ; ++ i)
83     {
84         mars [i] += mars [i - 1] ;
85         cout << mars [i] << ' ' ;
86     }
87
88     return 0;
89 }
```

21.4 100m

Problema 4 - 100m

Proba de 100 metri plat este una dintre cele mai populare și prestigioase probe din cadrul oricărui concurs de atletism. Recor-dul mondial al acestei probe este deținut în prezent de sportivul jamaican Usain Bolt cu timpul de 9.58 secunde.

Uneori lupta dintre sportivi este atât de strânsă încât diferen-țierea dintre atleți se poate face doar cu ajutorul camerelor de luat vederi ce surprind finish-ul atleților.

Au existat cazuri când doi sau mai mulți atleți au fost declarati la egalitate.

100 de puncte



Cerințe

Considerând N atleți, ce participă la o cursă de 100 metri plat, identificați prin numerele 1, 2, ..., N , să se scrie un program care determină numărul P al clasamentelor distincte care pot fi obținute după finalizarea cursei.

De exemplu, pentru $N = 2$, se pot obține 3 clasamente distincte: (1,2), (2,1),(1=2); unde (1=2) reprezintă situația când ambii atleți s-au clasat la egalitate.

Date de intrare

Fișierul de intrare **100m.in** conține pe prima linie numărul natural N , cu semnificația de mai sus.

Date de ieșire

Fișierul de ieșire **100m.out** va conține pe prima linie restul împărțirii numărului P la 666013.

Restricții și precizări

- $2 \leq N \leq 5\ 000$;
- Două clasamente se consideră distincte dacă diferă prin cel puțin o poziție;
- Pentru teste în valoare de 32 de puncte $N \leq 500$

Exemplu:

100m.in	100m.out	Explicații
3	13	$N = 3$ atleți. Numerotând atleții cu 1, 2 și 3 există 13 clasamente distincte: (1, 2, 3) ; (1, 3, 2) ; (2, 1, 3) ; (2, 3, 1) ; (3, 1, 2) ; (3, 2, 1) (1 și (2=3)) ; (2 și (1=3)) ; (3 și (1=2)) ; ((2=3) și 1) ; ((1=3) și 2) ; ((1=2) și 3) ; (1=2=3). Prin $(i = j)$ am notat posibilitatea ca atleții i și j să termine cursa în același timp. Prin $(i = j = k)$ am notat posibilitatea ca atleții i , j și k să termine cursa în același timp.
1771	74140	$N = 1771$ atleți. Numărul de clasamente distincte în care atleții pot termina cursa, modulo 666013, este 74140.

Timp maxim de executare/test: **0.8** secunde pe Windows, **0.6** secunde pe Linux

Memorie: total **2 MB**

Dimensiune maximă a sursei: **10 KB**

21.4.1 Indicații de rezolvare

prof. Cheșcă Ciprian, Liceul Tehnologic "Grigore C. Moisil" Buzău

Varianta 1

Fără egalități între atleți răspunsul este $n!$

Fie H_n răspunsul corespunzător cazului în care putem avea egalități.

Aveam $H_1 = 1$ și $H_2 = 3$.

Să analizăm modul de calcul al lui H_3 .

Rezultatele pot fi de forma $3, 2 + 1, 1 + 1 + 1$.

Acestea sunt toate *partițiiile numărului 3*.

Primul element înseamnă că cei 3 atleți ajung simultan, $2 + 1$ înseamnă ca doi atleți ajung simultan iar al treilea înainte sau după cei doi și $1 + 1 + 1$ înseamnă că toți atleții ajung la momente diferite.

- Grupul de 3 poate ajunge într-o singură modalitate.
- Grupurile de 2 din $2 + 1$ pot ajunge în două moduri iar atletul singur poate fi ales în 3 moduri.
- Grupul $1 + 1 + 1$ poate fi ales în $3!$ moduri

Așadar conform *regului sumă-produs* avem: $H_3 = 1 + 2 \times 3 + 3! = 13$

Pentru a calcula H_4 considerăm toate partițiiile numărului 4 și studiem ordinea diferitelor grupuri. Obținem $H_4 = 1 + 4 \times 2 + 3 \times 2 + 6 \times 3! + 4! = 75$

Generalizând, dacă vom nota $H_0 = 1$, găsim următoarea relație de recurență :

$$H_n = \sum_{k=1}^n C_n^k \cdot H_{n-k} \quad (1)$$

Varianta 2

Fie $S(n, k)$ numărul de partiții ale unei mulțimi cu n elemente în grupuri de k submulțimi. $S(n, k)$ este cunoscut ca *numărul lui Stirling de speță a II-a*.

Se poate demonstra că

$$H_n = \sum_{k=0}^n S(n, k) \cdot k! \quad (2)$$

O relație de recurență pentru $S(n, k)$ este $S(n+1, k) = S(n, k-1) + k \cdot S(n, k)$, cu $S(n, 1) = S(n, n) = 1$ și este asemănătoare cu *relația de recurență a combinărilor*.

Rezolvarea implementează relațiile de recurență (1) sau (2), cu menținerea că trebuie gestionat atent modul de utilizare al memoriei deoarece limitele de memorie vor impune *utilizarea a 2 vectori* în locul unei matrice.

Varianta 3

Se determină o formulă recursivă de calcul de genul:

fie $f(N, K)$ numărul de așezări a N concurenți pe K nivele

$f(N, 1) = 1$ //evident

$f(N, N) = N!$ //evident

$f(N, k) = k * f(N - 1, k) + k * f(N - 1, k - 1)$

Explicație: orice final pentru N concurenți pe k nivele se obține

a) dintr-o așezare a $N - 1$ concurenți pe K nivele și punând pe concurrentul N pe oricare din cele k nivele

sau

b) dintr-o așezare pe $k - 1$ nivele a $N - 1$ candidați și un nivel suplimentar (al k -lea) pe care se află doar concurrentul N , iar aici avem k posibilități pentru poziția acestui nivel suplimentar printre cele $k - 1$ nivele existente

La final numărul căutat este

$$s = f(N, 1) + f(N, 2) + \dots + f(N, N)$$

Apare dificultatea lucrului cu 2 vectori sau cu ultimele 2 linii ale matricei pentru încadrare în memorie.

21.4.2 *Rezolvare detaliată

21.4.3 Cod sursă

Listing 21.4.1: 100m_40p.cpp

```

1 /*Patrick Catalin Alexandru Sava
2 Atooom Industry and Resources SRL
3 University of Bucharest
4 Expected : 40 points */
5
6 #include <iostream>
7 #include <vector>
8
9 using namespace std ;
10
11 ifstream cin ("100m.in") ;
12 ofstream cout ("100m.out") ;
13
14 const int MOD = 666013 ;
15
16 int main(int argc, char const *argv[])
17 {
18     int n ;
19     cin >> n ;
20     vector <int> *dp = new vector <int> [n + 2] ;
21
22     for (int i = 0 ; i <= n + 1 ; ++ i)
23         dp [i].resize(n + 2) ;
24
25     dp [0][0] = 1 ;
26     for (int i = 1 ; i <= n ; ++ i)
27     {
28         for (int j = 1 ; j <= i ; ++ j)
29         {
30             dp [i][j] = 1LL * dp [i - 1][j - 1] * j % MOD +
31                         1LL * dp [i - 1][j] * j % MOD ;
32             dp [i][j] %= MOD ;
33         }
34     }
35
36     int s = 0 ;
37     for (int i = 1 ; i <= n ; ++ i)
38     {
39         s += dp [n][i] ;
40         s %= MOD ;
41     }
42
43     cout << s << '\n' ;
44     return 0;
45 }
```

Listing 21.4.2: 100m_100p_1.cpp

```

1 // Sursa cu numere Stirling de speta a doua - prof. Chesca Ciprian
2 // H(n) = S(n,1)! + S(n,2)! + ... + S(n,n-1)(n-1)! + S(n,n)n!
3
4 #include <iostream>
5 #define M 666013
6 #define nmax 5001
7
8 using namespace std;
9
10 long long n,H=0,a[nmax+1],b[nmax+1],f=1;
11
12 ifstream fin("100m.in");
13 ofstream fout("100m.out");
14
15 int main()
16 {
```

```

17     int i,j;
18     fin>>n;
19
20     a[0]=1;b[0]=1;
21
22     for(i=1;i<n;i++)
23     {
24         for(j=1;j<=i;j++)
25             b[j]=((j+1)*a[j]+a[j-1])%M;
26
27         for(j=0;j<=i;j++)
28             a[j]=b[j];
29     }
30
31     for(i=1;i<=n;i++)
32     {
33         H=(H+b[i-1]*f)%M;
34         f=f*(i+1)%M;
35     }
36
37     fout<<H<<"\n";
38
39     fin.close();
40     fout.close();
41
42     return 0;
43 }
```

Listing 21.4.3: 100m_100p_2.cpp

```

1 #include<iostream>
2
3 using namespace std;
4
5 ifstream fin("100m.in");
6 ofstream fout("100m.out");
7
8 long long f[2][5002],fact,s;
9 int N,k,n1,n0,n;
10
11 int main()
12 {
13     fin>>N;
14
15     f[1][1]=1;
16     fact=1;
17     for(n=2;n<=N;n++)
18     {
19         fact=fact*n%666013;
20         n1=n%2;//nou
21         n0=1-n1;//vechi
22         f[n1][1]=1;
23         f[n1][n]=fact;
24         for(k=2;k<=n-1;k++)
25             f[n1][k]=(f[n0][k]+f[n0][k-1])*k%666013;
26     }
27
28     s=0;
29     for(k=1;k<=N;k++)
30     {
31         s=s+f[N%2][k];
32         s=s%666013;
33     }
34
35     fout<<s;
36     fout.close();
37     return 0;
38 }
```

Listing 21.4.4: 100m_100p_3.cpp

```

1 #include <iostream>
2
3 using namespace std;
```

```

4
5 int dp[2][5003];
6 const int MOD = 666013;
7
8 ifstream cin ("100m.in");
9 ofstream cout ("100m.out");
10
11 int main()
12 {
13     int n, l1, l2;
14
15     cin >> n;
16     dp[1][1] = 1;
17     for (int i = 2; i <= n; i++)
18     {
19         l1 = i % 2;
20         l2 = (i - 1) % 2;
21         dp[l1][1] = 1;
22         for (int j = 2; j <= i; j++)
23         {
24             dp[l1][j] = dp[l2][j - 1] + dp[l2][j];
25             if (dp[l1][j] >= MOD)
26                 dp[l1][j] -= MOD;
27             dp[l1][j] = 111 * dp[l1][j] * j % MOD;
28         }
29     }
30
31     int ans = 0;
32     l1 = n % 2;
33     for (int i = 1; i <= n; i++)
34     {
35         ans = ans + dp[l1][i];
36         if (ans >= MOD)
37             ans -= MOD;
38     }
39
40     cout << ans << "\n";
41 }
```

Listing 21.4.5: 100m_100p_4.cpp

```

1 /*Patrick Catalin Alexandru Sava
2 Atoom Industry and Resources SRL
3 University of Bucharest
4 Expected : 100 points */
5
6 #include <iostream>
7 #include <vector>
8
9 using namespace std ;
10
11 ifstream cin ("100m.in") ;
12 ofstream cout ("100m.out") ;
13
14 const int MOD = 666013 ;
15
16 int main(int argc, char const *argv[])
17 {
18     int n ;
19     cin >> n ;
20
21     vector <int> *dp = new vector <int> [2] ;
22
23     for (int i = 0 ; i <= 1 ; ++ i)
24     {
25         dp [i].resize(n + 2) ;
26     }
27
28     int lin = 1 ;
29     dp [0][0] = 1 ;
30     for (int i = 1 ; i <= n ; ++ i)
31     {
32         for (auto &x : dp[lin])
33             x = 0 ;
```

```

35     for (int j = 1 ; j <= i ; ++ j)
36     {
37         dp [lin][j] = 1LL * dp [1 - lin][j - 1] * j % MOD +
38                     1LL * dp [1 - lin][j] * j % MOD ;
39         dp [lin][j] %= MOD ;
40     }
41
42     lin = 1 - lin ;
43 }
44
45 int s = 0 ;
46 for (int i = 1 ; i <= n ; ++ i)
47 {
48     s += dp [1 - lin][i] ;
49     s %= MOD ;
50 }
51
52 cout << s << '\n' ;
53 return 0;
54 }
```

21.5 camp

Problema 5 - camp

100 de puncte

Maria petrece ultimele zile din vacanța de Crăciun la Brașov, în aprilie. Aceasta este atât de nerăbdătoare ca să se topească încât deja își imaginează cum se joacă într-un spațiu verde, de forma unui poligon cu N vârfuri. Maria, pasionată de matematică de altfel, reprezintă acest poligon în sistemul de coordonate carteziene. Aceasta este foarte curioasă câte puncte laticeale se află în interiorul poligonului, cât și pe marginile acestuia.

Un punct laticeal este un punct (x, y) cu proprietatea că x și y sunt numere naturale. Considerând această problemă mult prea ușoară pentru nivelul ei, aceasta vrea să o complice, atribuindu-i fiecărui punct laticeal de forma (x, y) un cost egal cu suma coordonatelor sale, $x + y$. Acum, mulțumită de ceea ce a realizat, vă roagă să aflați care este suma costurilor punctelor laticeale aflate în interiorul și pe marginile poligonului.

Cerințe

Având un poligon cu N vârfuri date în ordine trigonometrică, vi se cere să aflați care este suma coordonatelor punctelor laticeale aflate în interiorul și pe marginea acestuia.

Date de intrare

Fișierul de intrare **camp.in** conține pe prima linie un număr natural N , reprezentând numărul de vârfuri ale poligonului. Următoarele N linii vor conține exact câte două valori naturale separate prin câte un spațiu, reprezentând vâfurile poligonului, date în ordine trigonometrică.

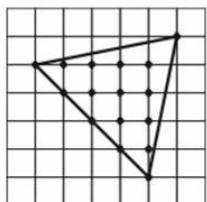
Date de ieșire

Fișierul de ieșire **camp.out** va conține pe prima linie numărul natural cerut.

Restricții și precizări

- $3 \leq N \leq 100.000$
- Pentru orice punct (x, y) din fișierul de intrare se garantează faptul că $1 \leq x, y \leq 100.000$
- Se garantează faptul că poligonul din fișierul de intrare este convex.
- Pentru teste în valoare de 25 de puncte $N \leq 50$ și $x, y \leq 100$
- Pentru teste în valoare de încă 25 de puncte $N \leq 500$ și $x, y \leq 1000$

Exemplu:

camp.in	camp.out	Explicații
3 5 1 6 6 1 5	122	<p>Suma costurilor celor 16 puncte laticeale care se află în interiorul sau pe marginile triunghiului cu vâfurile în coordonatele $(5,1)$, $(6,6)$ și $(1,5)$ este 122.</p> 

Timp maxim de executare/test: **1.0** secunde pe Windows **0.2** secunde pe Linux

Memorie: total **32 MB**

Dimensiune maximă a sursei: **10 KB**

21.5.1 Indicații de rezolvare

Adrian Budau

Patrick Sava

Maria Pandele

Observăm faptul că punctele au coordonatele numere naturale mai mici sau egale cu 100 000. Pentru un x fixat, avem o dreaptă verticală ce se intersectează cu poligonul prin maxim 2 puncte (poligonul este *convex*). Trebuie să determinăm rapid cele două laturi cu care se intersectează o astfel de dreaptă.

Pornim cu cea mai mică coordonată x_{min} care există în setul de intrare și ne fixăm prin 2 indici i_1 și i_2 cele 2 laturi ce se intersectează cu verticala x_{min} . Cu un indice ne vom muta în sens trigonometric și cu unul în sens orar. Apoi fixăm o coordonată x de la x_{min} la x_{max} , verificăm dacă trebuie să mutăm indicii și calculăm cele două puncte de intersecție ce pot avea coordonate reale (x, y_1) și (x, y_2) cu $y_1 \leq y_2$.

Notam

$a =$ cel mai mic număr natural $\geq y_1$

$b =$ cel mai mare număr natural $\leq y_2$.

Punctele lacticeale al căror cost trebuie adunat vor avea coordonatele (x, a) , $(x, a+1)$, $(x, a+2)$, ..., (x, b) . Deci adăugam la răspuns $x * (b - a + 1) + b * (b + 1)/2 - a * (a - 1)/2$.

Complexitate: O(coordonata maxima)

21.5.2 *Rezolvare detaliată

21.5.3 Cod sursă

Listing 21.5.1: camp_30p.cpp

```

1  /*Patrick Catalin Alexandru Sava
2   Atooom Industry and Resources SRL
3   University of Bucharest
4   Expected : 30*/
5
6  #include <fstream>
7  #include <vector>
8
9  using namespace std ;
10
11 ifstream cin ("camp.in") ;
12 ofstream cout ("camp.out") ;
13
14 int sgn (pair <int, int> a, pair <int, int> b, pair <int, int> c)
15 {
16     int res = a.first * b.second + b.first * c.second + c.first * a.second
17         - b.second * c.first - c.second * a.first - a.second * b.first ;
18
19     if (res == 0)    return 0 ;
20     if (res > 0)    return 1 ;
21     if (res < 0)    return -1 ;
22
23     return 123; // !!!
24 }
25
26 bool verif (pair<int, int> point, vector <pair<int, int>> &points, int n)
27 {
28     int maxim = -5 ;
29     int minim = 5 ;
30     for (int i = 0 ; i < n ; ++ i)

```

```

31     {
32         maxim = max (maxim , sgn (point, points[i], points[i + 1])) ;
33         minim = min (minim , sgn (point, points[i], points[i + 1])) ;
34     }
35     if (maxim - minim == 2)
36         return 0 ;
37     return 1 ;
38 }
39
40 int main(int argc, char const *argv[])
41 {
42     vector <pair<int, int>> points ;
43     int n ;
44     cin >> n ;
45
46     int min_x = 1e6 ;
47     int min_y = 1e6 ;
48     int max_x = 0 ;
49     int max_y = 0 ;
50
51     for (int i = 1 ; i <= n ; ++ i)
52     {
53         int x, y ;
54         cin >> x >> y ;
55         points.push_back ({x, y}) ;
56         min_x = min (min_x, x) ;
57         min_y = min (min_y, y) ;
58         max_x = max (max_x, x) ;
59         max_y = max (max_y, y) ;
60     }
61
62     points.push_back (points[0]) ;
63     long long sol = 0 ;
64     for (int i = min_x ; i <= max_x; ++ i)
65     {
66         bool been = false ;
67         for (int j = min_y; j <= max_y; ++ j)
68             if (verif ({i, j}, points, n))
69             {
70                 sol = sol + 1LL * i + 1LL * j ;
71                 been = true ;
72             }
73             else if (been == true)
74                 break ;
75         }
76         cout << sol << '\n' ;
77     }
78 }

```

Listing 21.5.2: camp_45p.cpp

```

1 /*Patrick Catalin Alexandru Sava
2 Attoom Industry and Resources SRL
3 University of Bucharest
4 Expected : 30*/
5
6 #include <iostream>
7 #include <vector>
8
9 using namespace std ;
10
11 ifstream cin ("camp.in") ;
12 ofstream cout ("camp.out") ;
13
14 int sgn (pair <int, int> a, pair <int, int> b, pair <int, int> c)
15 {
16     int res = a.first * b.second + b.first * c.second + c.first * a.second
17             - b.second * c.first - c.second * a.first - a.second * b.first ;
18     if (res == 0) return 0 ;
19     if (res > 0) return 1 ;
20     if (res < 0) return -1 ;
21
22     return 123; // !!!
23 }
24

```

```

25  bool verif (pair<int, int> point, vector <pair<int, int>> &points, int n)
26  {
27      int maxim = -5 ;
28      int minim = 5 ;
29      for (int i = 0 ; i < n ; ++ i)
30      {
31          maxim = max (maxim , sgn (point, points[i], points[i + 1])) ;
32          minim = min (minim , sgn (point, points[i], points[i + 1])) ;
33          if (maxim - minim == 2)
34              return 0 ;
35      }
36      return 1 ;
37  }
38 }
39
40 int main(int argc, char const *argv[])
41 {
42     vector <pair<int, int>> points ;
43     int n ;
44     cin >> n ;
45
46     int min_x = 1e6 ;
47     int min_y = 1e6 ;
48     int max_x = 0 ;
49     int max_y = 0 ;
50     for (int i = 1 ; i <= n ; ++ i)
51     {
52         int x, y ;
53
54         cin >> x >> y ;
55         points.push_back ({x, y}) ;
56
57         min_x = min (min_x, x) ;
58         min_y = min (min_y, y) ;
59         max_x = max (max_x, x) ;
60         max_y = max (max_y, y) ;
61     }
62
63     points.push_back (points[0]) ;
64     long long sol = 0 ;
65     for (int i = min_x ; i <= max_x; ++ i)
66     {
67         int a, b ;
68         for (int j = min_y; j <= max_y; ++ j)
69         {
70             if (verif ({i, j}, points, n))
71             {
72                 a = j ;
73                 break ;
74             }
75         }
76
77         for (int j = max_y; j >= min_y; -- j)
78         {
79             if (verif ({i, j}, points, n))
80             {
81                 b = j ;
82                 break ;
83             }
84         }
85
86         if (a > b)
87             swap (a, b) ;
88
89         sol += 1LL * (b - a + 1) * i ;
90         sol += 1LL * (b + 1) * b / 2LL ;
91         sol -= 1LL * (a - 1) * a / 2LL ;
92     }
93
94     cout << sol << '\n' ;
95     return 0;
96 }
```

Listing 21.5.3: camp_100p_1.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <algorithm>
5 #include <cassert>
6
7 using namespace std;
8
9 int64_t area(pair<int, int> A, pair<int, int> B, pair<int, int> C)
10 {
11     return 1LL * A.first * (B.second - C.second) +
12        1LL * B.first * (C.second - A.second) +
13        1LL * C.first * (A.second - B.second);
14 }
15
16 int sign(int x)
17 {
18     if (x > 0)
19         return 1;
20     return -1;
21 }
22
23 int64_t partial_sum(int to)
24 {
25     return 1LL * to * (to + 1) / 2;
26 }
27
28 int64_t partial_sum(int from, int to)
29 {
30     return partial_sum(to) - partial_sum(from - 1);
31 }
32
33 int main()
34 {
35     ifstream cin("camp.in");
36     ofstream cout("camp.out");
37
38     int N;
39     assert(cin >> N);
40     assert(1 <= N && N <= 100 * 1000);
41
42     vector< pair<int, int> > V(N);
43     for (int i = 0; i < N; ++i)
44     {
45         assert(cin >> V[i].first >> V[i].second);
46         assert(1 <= V[i].first && V[i].first <= 100 * 1000);
47         assert(1 <= V[i].second && V[i].second <= 100 * 1000);
48     }
49
50     for (int i = 0; i < N; ++i)
51         assert(area(V[i], V[(i + 1) % N], V[(i + 2) % N]) >= 0);
52
53     int max_x = V[0].first;
54     for (auto &p : V)
55         max_x = max(max_x, p.first);
56
57     vector< vector< pair<int, int> > > intersect(max_x + 1);
58
59     for (int i = 0; i < N; ++i)
60     {
61         auto now = V[i];
62         auto next = V[(i + 1) % N];
63
64         if (now.first == next.first)
65             continue;
66
67         int from = now.first;
68         int to = next.first;
69         auto after_next = V[(i + 2) % N];
70
71         if (next.first == after_next.first)
72             to += sign(to - from);
73
74         int anum = (now.second - next.second);
75         int den = (now.first - next.first);
76         int64_t bnum = (1LL * next.second * now.first

```

```

77             - 1LL * now.second * next.first);
78
79         if (den < 0)
80     {
81             anum *= -1;
82             den *= -1;
83             bnum *= -1;
84         }
85
86         for (int x = from; x != to; x += sign(to - from))
87     {
88             int64_t ynum = 1LL * anum * x + bnum;
89             assert(1 <= ynum / den && ynum / den <= 100 * 1000);
90             if (ynum % den == 0)
91                 intersect[x].emplace_back(ynum / den, ynum / den);
92             else
93                 intersect[x].emplace_back(ynum / den, ynum / den + 1);
94         }
95     }
96
97     int64_t answer = 0;
98     for (int i = 0; i <= max_x; ++i)
99     {
100         if (intersect[i].empty())
101             continue;
102         if (intersect[i].size() == 1)
103         {
104             assert(intersect[i][0].first == intersect[i][0].second);
105             answer += i + intersect[i][0].first;
106             continue;
107         }
108
109         assert(intersect[i].size() == 2);
110         if (intersect[i][0] > intersect[i][1])
111             swap(intersect[i][0], intersect[i][1]);
112
113         int from = intersect[i][0].second;
114         int to = intersect[i][1].first;
115
116         answer += 1LL * i * (to - from + 1);
117         answer += partial_sum(from, to);
118     }
119
120     cout << answer << "\n";
121 }
```

Listing 21.5.4: camp_100p_2.cpp

```

1 #include <fstream>
2 #include <iostream>
3 #include <vector>
4
5 using namespace std;
6
7 struct Point
8 {
9     int x, y;
10 };
11
12 vector <Point> v;
13
14 inline long long sum(int n)
15 {
16     return 111 * n * (n + 1) / 2;
17 }
18
19 bool InSegment(Point A, Point B, int x)
20 {
21     int minx = min(A.x, B.x);
22     int maxx = A.x + B.x - minx;
23
24     return (minx <= x && x <= maxx);
25 }
26
27 inline bool Vertical(Point A, Point B)
```

```

28 {
29     return A.x == B.x;
30 }
31
32 int intersect(int x, int &i, int sens)
33 {
34     while (!InSegment(v[i], v[i + 1], x) || Vertical(v[i], v[i + 1]))
35     {
36         i = i + sens;
37         if (i == v.size() - 1) i = 0;
38         if (i < 0) i = v.size() - 2;
39     }
40
41     int a = v[i].y - v[i + 1].y;
42     int b = v[i + 1].x - v[i].x;
43     long long c = 111 * v[i].x * v[i + 1].y - 111 * v[i].y * v[i + 1].x;
44
45     double long y = (-1 * (double long)c - (double long)x * a) / b;
46     if (sens > 0)
47     {
48         if ((y - (int)y) > 1e-14)
49             return (int)(y + 1);
50         return (int)y;
51     }
52
53     return (int)y;
54 }
55
56 int before(int i)
57 {
58     i--;
59     if (i < 0)
60         i = v.size() - 2;
61
62     return i;
63 }
64
65 int main()
66 {
67     Point P;
68     int n, x, y, i1, i2, left, right;
69
70     ifstream cin("camp.in");
71     ofstream cout("camp.out");
72
73     cin >> n;
74
75     int limx = 0;
76     for (int i = 0; i < n; i++)
77     {
78         cin >> x >> y;
79         P.x = x;
80         P.y = y;
81         v.push_back(P);
82         limx = max(x, limx);
83     }
84
85     v.push_back(v[0]);
86
87     x = 2e5 + 10;
88     for (int i = 0; i < n; i++)
89         if (v[i].x < x)
90             x = v[i].x;
91
92     i1 = i2 = -1;
93     for (int i = 0; i < n; i++)
94         if (v[i].x == x)
95         {
96             if (i1 == -1)
97                 i1 = before(i);
98             else
99                 if (v[i].y > v[i1].y)
100                     i1 = before(i);
101
102             if (i2 == -1)
103                 i2 = i;

```

```

104         else
105             if (v[i].y < v[i2].y)
106                 i2 = i;
107         }
108
109     long long ans = 0;
110     for (int i = x; i <= limx; i++)
111     {
112         left = intersect(i, i2, 1);
113         right = intersect(i, il, -1);
114
115 //         cerr << left << " " << right << "\n";
116         ans = ans + 111 * i * (right - left + 1) + sum(right) - sum(left - 1);
117     }
118
119     cout << ans << "\n";
120
121     return 0;
122 }
```

21.6 identice

Problema 6 - identice

100 de puncte

Mihai a construit o matrice pătratică A de dimensiune N cu valori în mulțimea $\{0,1\}$. El preferă acele matrice care au toate elementele identice și de aceea a calculat pentru matricea A , numărul K de submatrice care au toate elementele identice. Acum, Mihai vrea să transforme matricea A într-o matrice cu toate elementele identice. Pentru aceasta, el a selectat un număr natural nenul D , și definește operația ZET care constă în alegerea unei submatrice pătratice de dimensiunea D din matricea precedentă în care schimbă toate elementele 0 în 1 și invers. El vrea să aplice operația ZET inițial pentru matricea A , apoi repetă operația pentru matricea obținută la momentul anterior, de un număr minim de ori, notat R , până când matricea obținută are toate elementele identice, sau dacă nu este posibil, R va avea valoarea -1.

Cerințe

Mihai vă roagă să calculați valorile K și R . Pentru a preciza tipul cerinței, Mihai folosește un cod T care dacă are valoarea 1, atunci solicită calcularea valorii K , iar dacă T are valoarea 2, atunci solicită calcularea valorii R .

Date de intrare

Pe prima linie a fișierului **identice.in** se vor afla numerele naturale T , N și D , cu semnificația de mai sus, separate prin câte un spațiu.

Pe următoarele N linii se vor afla câte N valori de 0 și 1, elementele liniilor matricei A , fără spații între ele.

Date de ieșire

Pe prima linie a fișierului **identice.out** se va afla un număr natural, respectiv valoarea K pentru $T = 1$ sau valoarea R pentru $T = 2$.

Restricții și precizări

- $1 < D < N \leq 1000$.
- Pentru calcularea valorii K , submatricele pot fi pătratice sau dreptunghiulare, cu diferite dimensiuni (inclusiv 1), cu elementele identice.
- Se acordă 40% din punctaj pentru determinarea corectă a lui K iar pentru determinarea corectă a lui R se acordă 60% din punctajul total.

Exemple:

identice.in	identice.out	Explicații
1 4 2 0011 0011 1100 1100	36	$T = 1$, deci se calculează $K = 36$ Sunt 18 submatrice cu toate elementele 0 și 18 cu toate elementele 1.
2 4 2 0011 0011 1100 1100	2	$T = 2$, deci se calculează $R = 2$, deoarece sunt necesare 2 aplicări ale operației ZET .

Timp maxim de executare/test: **0.5** secunde pe Windows **0.15** secunde pe Linux

Memorie: total **32 MB**

Dimensiune maximă a sursei: **10 KB**

21.6.1 Indicații de rezolvare

prof. Gheorghe Manolache, Colegiul Național de Informatică. Piatra Neamț

Se va studia separat cazul pentru valoarea 0 și separat pentru valoarea 1.

Vom analiza modul de calcul al numărului de submatrice K ce conțin o singură valoare.

Fie de exemplu, $s[i][j]$ secvență de 0 pe linia i de lungime maximă ce se termină în coloana j . Atunci numărul de submatrici ce conțin doar 0 cu colțul dreapta-jos în linia i , coloana j , va fi $s[i][j] + \min(s[i][j], s[i-1][j]) + \dots + \min(s[i][j], s[i-1][j], \dots, s[1][j])$, deci se adună după înălțimea de la 1 la j .

O astfel de implementare are complexitate $O(N^3)$ și obține 10-20 puncte.

Se obține complexitate $O(N^2)$ dacă parcurgem elementele pe coloană și menținem o *stivă* ordonată crescător cu elementele din s , iar pentru fiecare element asociem numărul de termeni din suma precedentă pentru care este minim. Atunci când introducem $s[i][j]$ în stivă, eliminăm toate elementele mai mari și vom adăuga la numărul asociat lui numerele asociate celor eliminate. Vom adăuga mereu la rezultat produsul dintre elementele stivei și valorile asociate lor.

Pentru a calcula valoarea R , se constată că un *brut* se poate face în N^4 , observând că atunci când ajungem la o valoare 1 (și vrem să obținem 0) suntem stânga-sus la un pătrat și aplicăm ZET dacă este posibil, altfel nu am soluție.

Se optimizează metoda calculând în $O(1)$ pentru fiecare element numărul de modificări.

Se poate folosi *șmenul lui Mars* pe o matrice pentru a optimiza operația ZET sau folosind N *cozi* pentru linii și alte N *cozi* pentru coloane. Verificăm pentru fiecare element paritatea sumei dimensiunii cozilor conform poziției. Dacă obținem 1, se schimbă în 0 și poziția se reține în cozi sortate iar dacă diferența dintre indicii primului și cel curent depășește D , se scoate elementul din coadă.

Dacă trebuie aplicată operația ZET în poziția curentă dar nu există spațiu (submatrice de latura D) atunci nu avem soluție, deci $R=-1$.

21.6.2 *Rezolvare detaliată

21.6.3 Cod sursă

Listing 21.6.1: identice_100p_1.cpp

```

1 #include<bits/stdc++.h>
2
3 using namespace std;
4
5 ifstream in("identice.in");
6 ofstream out("identice.out");
7
8 #define Nmax 1005
9

```

```

10 char a[Nmax][Nmax],a1[Nmax][Nmax];
11 char opus[]="10";
12 int t0[Nmax][Nmax],st0[Nmax],val0[Nmax],vf0,n,d;
13 int t1[Nmax][Nmax],st1[Nmax],val1[Nmax],vf1;
14 long long sum0,sum1;
15
16 int sol0,p0[Nmax],M0[Nmax][Nmax];
17 int sol1,p1[Nmax],M1[Nmax][Nmax];
18 int test;
19
20 void read()
21 {
22     in>>test>>n>>d; in.get();
23     for(int i=1;i<=n;++i)
24     {
25         in.getline(a[i]+1,n+2);
26         for(int j=1;j<=n;++j)
27             a1[i][j]=opus[a[i][j]-'0'];
28     }
29     in.close();
30 }
31
32 void suma0()
33 {
34     for(int i=1;i<vf0;i++)
35         sum0+=val0[i]*st0[i];
36 }
37 void suma1()
38 {
39     for(int i=1;i<vf1;i++)
40         sum1+=val1[i]*st1[i];
41 }
42
43 void solve0()
44 {
45     int m;
46     for(int j=1;j<=n;j++)
47     {
48         vf0=0;
49         for(int i=1;i<=n;i++)
50         {
51             if(a[i][j]=='0')
52                 t0[i][j+1]=1+t0[i][j];
53             m=1;
54             while(vf0>0&&st0[vf0]>=t0[i][j+1])
55             {
56                 m+=val0[vf0];
57                 vf0--;
58             }
59             vf0++;
60             st0[vf0]=t0[i][j+1];
61             val0[vf0]=m;
62             sum0+=val0[vf0]*st0[vf0];
63             suma0();
64         }
65     }
66     //out<<sum0;
67 }
68 void solve1()
69 {
70     int m;
71     for(int j=1;j<=n;j++)
72     {
73         vf1=0;
74         for(int i=1;i<=n;i++)
75         {
76             if(a[i][j]=='1')
77                 t1[i][j+1]=1+t1[i][j];
78             m=1;
79             while(vf1>0&&st1[vf1]>=t1[i][j+1])
80             {
81                 m+=val1[vf1];
82                 vf1--;
83             }
84         }
85         vf1++;

```

```

86             st1[vf1]=t1[i][j+1];
87             val1[vf1]=m;
88             sum1+=val1[vf1]*st1[vf1];
89             sumal();
90         }
91     }
92     //out<<sum1;
93 }
94
95 void R0()
96 {
97     int ok=1;
98     for(int i=1;i<=n;++i)
99         for(int j=1;j<=n;++j)
100     {
101         M0[i][j]+=M0[i-1][j];
102         p0[j]=p0[j-1]+M0[i][j];
103
104         int t=(p0[j]&1);
105
106         if(i>n-d+1||j>n-d+1)
107             if(t!=(a[i][j]-'0'))
108                 ok=0;
109             else
110                 if(t!=(a[i][j]-'0'))
111                 {
112                     ++sol0;
113                     M0[i][j]++;
114                     M0[i+d][j]--;
115                     M0[i][j+d]--;
116                     M0[i+d][j+d]++;
117                     ++p0[j];
118                 }
119     }
120
121     if(!ok) sol0=-1;
122 }
123
124 void R1()
125 {
126     int ok=1;
127     for(int i=1;i<=n;++i)
128         for(int j=1;j<=n;++j)
129     {
130         M1[i][j]+=M1[i-1][j];
131         p1[j]=p1[j-1]+M1[i][j];
132
133         int t=(p1[j]&1);
134
135         if(i>n-d+1||j>n-d+1)
136         {
137             if(t!=(a1[i][j]-'0'))
138                 ok=0;
139         }
140         else
141             if(t!=(a1[i][j]-'0'))
142             {
143                 ++sol1;
144                 M1[i][j]++;
145                 M1[i+d][j]--;
146                 M1[i][j+d]--;
147                 M1[i+d][j+d]++;
148                 ++p1[j];
149             }
150     }
151
152     if(!ok)
153         sol1=-1;
154 }
155
156 int main()
157 {
158     read();
159     if(test==1)
160     {
161         solve0();

```

```

162         solve1();
163         out<<sum0+sum1<<' \n';
164     }
165     else
166     {
167         R0();
168         R1();
169         int rx=min(sol0,sol1),ry=max(sol0,sol1);
170         if(rx== -1)
171             rx=ry;
172         out<<rx<<' \n';
173     }
174     out.close();
175     return 0;
176 }

```

Listing 21.6.2: identice_100p_2.cpp

```

1 #include<bits/stdc++.h>
2
3 using namespace std;
4
5 #define Nmax 1005
6 #define pb push_back
7 #define pf pop_front
8 #define mp make_pair
9
10 char a[Nmax][Nmax],a1[Nmax][Nmax];
11 char opus[]="10";
12 int t0[Nmax][Nmax],st0[Nmax],val0[Nmax],vf0,n,d;
13 int t1[Nmax][Nmax],st1[Nmax],val1[Nmax],vf1;
14 long long sum0,sum1;
15 int test;
16 int sol0,sol1;
17
18 deque <pair <int,int> > q10;
19 deque <pair <int,int> > q11;
20
21 deque <int> qc0[Nmax];
22 deque <int> qc1[Nmax];
23
24 void read ()
25 {
26     int i;
27
28     scanf ("%d%d%d\n",&test,&n,&d);
29     for (i=1; i<=n; ++i)
30     {
31         fgets(a[i]+1,Nmax,stdin);
32         for(int j=1;j<=n;++j)
33             a1[i][j]=opus[a[i][j]-'0'];
34     }
35 }
36
37 void suma0 ()
38 {
39     for(int i=1;i<vf0;i++)
40         sum0+=val0[i]*st0[i];
41 }
42
43 void suma1 ()
44 {
45     for(int i=1;i<vf1;i++)
46         sum1+=val1[i]*st1[i];
47 }
48
49 void solve0 ()
50 {
51     int m;
52     for(int j=1;j<=n;j++)
53     {
54         vf0=0;
55         for(int i=1;i<=n;i++)
56         {

```

```

57         if(a[i][j]=='0')
58             t0[i][j+1]=1+t0[i][j];
59
60         m=1;
61         while(vf0>0&&st0[vf0]>=t0[i][j+1])
62         {
63             m+=val0[vf0];
64             vf0--;
65         }
66
67         vf0++;
68         st0[vf0]=t0[i][j+1];
69         val0[vf0]=m;
70         sum0+=val0[vf0]*st0[vf0];
71         suma0();
72     }
73 }
//out<<sum0;
75 }
76 void solve1()
77 {
78     int m;
79     for(int j=1; j<=n; j++)
80     {
81         vf1=0;
82         for(int i=1; i<=n; i++)
83         {
84             if(a[i][j]=='1')
85                 t1[i][j+1]=1+t1[i][j];
86
87             m=1;
88             while(vf1>0&&st1[vf1]>=t1[i][j+1])
89             {
90                 m+=val1[vf1];
91                 vf1--;
92             }
93
94             vf1++;
95             st1[vf1]=t1[i][j+1];
96             val1[vf1]=m;
97             sum1+=val1[vf1]*st1[vf1];
98             sumal();
99         }
100    }
101   //out<<sum1;
102 }
103
104 void R0()
105 {
106     int i,j,nr;
107
108     for (i=1; i<=n; ++i)
109     {
110         nr=0;
111         ql0.clear ();
112         for (j=1; j<=n; ++j)
113         {
114             while (!qc0[j].empty () && i-qc0[j].front ()>=d)
115                 qc0[j].pf ();
116
117             if (!qc0[j].empty ())
118             {
119                 nr+=qc0[j].size ();
120                 ql0.pb (mp (j,qc0[j].size ()));
121             }
122
123             while (!ql0.empty () && j-ql0.front ().first>=d)
124             {
125                 nr-=ql0.front ().second;
126                 ql0.pf ();
127             }
128
129             if ((nr%2==1 && a[i][j]=='0') || (nr%2==0 && a[i][j]=='1'))
130             {
131                 if (i>n-d+1 || j>n-d+1)
132                 {

```

```

133             sol0=-1;
134             return ;
135         }
136         ++sol0;
137         qc0[j].pb (i);
138         ++nr;
139         if (!ql0.empty () && ql0.back ().first==j)
140             ++ql0.back ().second;
141         else
142             ql0.pb (mp (j,1));
143     }
144 }
145 }
146 }
147
148 void R1()
149 {
150     int i,j,nr;
151
152     for (i=1; i<=n; ++i)
153     {
154         nr=0;
155         qll.clear ();
156         for (j=1; j<=n; ++j)
157         {
158             while (!qc1[j].empty () && i-qc1[j].front ()>=d)
159                 qc1[j].pf ();
160
161             if (!qc1[j].empty ())
162             {
163                 nr+=qc1[j].size ();
164                 qll.pb (mp (j,qc1[j].size ()));
165             }
166             while (!qll.empty () && j-qll.front ().first>=d)
167             {
168                 nr-=qll.front ().second;
169                 qll.pf ();
170             }
171
172             if ((nr%2==1 && a1[i][j]=='0') || (nr%2==0 && a1[i][j]=='1'))
173             {
174                 if (i>n-d+1 || j>n-d+1)
175                 {
176                     sol1=-1;
177                     return ;
178                 }
179                 ++sol1;
180                 qc1[j].pb (i);
181                 ++nr;
182                 if (!qll.empty () && qll.back ().first==j)
183                     ++qll.back ().second;
184                 else
185                     qll.pb (mp (j,1));
186             }
187         }
188     }
189 }
190
191 int main()
192 {
193     freopen("identice.in","r",stdin);
194     ofstream out("identice.out");
195
196     read();
197     if(test==1)
198     {
199         solve0();
200         solve1();
201         out<<sum0+sum1<<'\\n';
202     }
203     else
204     {
205         R0();
206         R1();
207         int rx=min(sol0,sol1),ry=max(sol0,sol1);
208         if(rx==-1)

```

```

209         rx=ry;
210         out<<rx<<' \n' ;
211     }
212
213     out.close();
214     return 0;
215 }
```

Listing 21.6.3: identice_100p_3.cpp

```

1  /*Patrick Catalin Alexandru Sava
2  Atoom Industry and Resources SRL
3  University of Bucharest
4  Expected : 100*/
5
6 #include <iostream>
7 #include <cassert>
8 #include <cstring>
9 #include <stack>
10 #include <vector>
11
12 using namespace std ;
13
14 ifstream cin ("identice.in") ;
15 ofstream cout ("identice.out") ;
16
17 const int MAX = 1e3 ;
18
19 long long compute (char mat[] [MAX + 3], char target, int n)
20 {
21     vector <int> st (n + 1) ;
22     vector <int> dr (n + 1) ;
23     vector <long long> ans (n + 1) ;
24     stack <int> s ;
25     vector <int> height (n + 2, 0) ;
26
27     for (int i = 1 ; i <= n ; ++ i)
28     {
29         while (!s.empty())
30             s.pop() ;
31
32         for (int j = 1 ; j <= n ; ++ j)
33             if (mat [i] [j] == target)
34                 height [j] += 1 ;
35             else
36                 height [j] = 0 ;
37
38         height [0] = -1 ;
39         s.push (0) ;
40         for (int j = 1 ; j <= n ; ++ j)
41         {
42             while (!s.empty() and height [s.top()] >= height [j])
43                 s.pop () ;
44
45             st [j] = s.top() ;
46             s.push(j) ;
47         }
48
49         while (!s.empty())
50             s.pop() ;
51
52         height [n + 1] = -1 ;
53         s.push (n + 1) ;
54         for (int j = n ; j >= 1 ; -- j)
55         {
56             while (!s.empty() and height [s.top()] > height [j])
57                 s.pop () ;
58
59             dr [j] = s.top() ;
60             s.push(j) ;
61         }
62
63         for (int j = 1 ; j <= n ; ++ j)
64             ans [height [j]] += 1LL * (dr [j] - j) * (j - st [j]) ;
65 }
```

```

66
67     for (int j = n ; j >= 1; -- j)
68         ans [j - 1] += ans [j] ;
69
70     long long Sum = 0 ;
71     for (int j = n ; j >= 1; -- j)
72         Sum += ans [j] ;
73
74     return Sum ;
75 }
76
77 long long Solve1 (char mat[][][MAX + 3], int n)
78 {
79     return compute (mat, '0', n) + compute (mat, '1', n) ;
80 }
81
82 int ZET (char mat[][][MAX + 3], int n, int d)
83 {
84     vector <int> *mars = new vector <int> [n + 4] ;
85
86     for (int i = 0 ; i <= n + 3 ; ++ i)
87         mars [i].resize(n + 3) ;
88
89     int sol = 0 ;
90     for (int i = 1 ; i <= n ; ++ i)
91         for (int j = 1 ; j <= n ; ++ j)
92         {
93             mars [i][j] += mars [i - 1][j] ;
94             mars [i][j] += mars [i][j - 1] ;
95             mars [i][j] -= mars [i - 1][j - 1] ;
96             if (mat [i][j] == '1' and mars [i][j] % 2 == 0)
97             {
98                 if (i + d - 1 > n or j + d - 1 > n)
99                     return -1 ;
100                mars [i][j] += 1 ;
101                mars [i][j + d] -= 1 ;
102                mars [i + d][j] -= 1 ;
103                mars [i + d][j + d] += 1 ;
104                sol += 1 ;
105            }
106
107            if (mat [i][j] == '0' and mars [i][j] % 2 == 1)
108            {
109                if (i + d - 1 > n or j + d - 1 > n)
110                    return -1 ;
111                mars [i][j] += 1 ;
112                mars [i][j + d] -= 1 ;
113                mars [i + d][j] -= 1 ;
114                mars [i + d][j + d] += 1 ;
115                sol += 1 ;
116            }
117        }
118
119     return sol ;
120 }
121
122 int Solve2 (char mat[][][MAX + 3], int n, int dfixed)
123 {
124     int A = ZET (mat, n, dfixed) ;
125
126     for (int i = 1 ; i <= n ; ++ i)
127         for (int j = 1 ; j <= n ; ++ j)
128             if (mat [i][j] == '0')
129                 mat [i][j] = '1' ;
130             else
131                 mat [i][j] = '0' ;
132
133     int B = ZET (mat, n, dfixed) ;
134
135     if (A == -1 and B == -1) return -1 ;
136     if (A == -1) return B ;
137     if (B == -1) return A ;
138
139     return min (A, B) ;
140 }
141

```

```
142 int main(int argc, char const *argv[])
143 {
144     char mat [MAX + 3][MAX + 3] ;
145     int tip ;
146     cin >> tip ;
147     int n, d;
148     cin >> n >> d ;
149     assert (n >= 1 and n <= MAX) ;
150     assert (d >= 1 and d <= MAX) ;
151
152     for (int i = 1 ; i <= n ; ++ i)
153     {
154         cin >> (mat[i] + 1) ;
155         assert (strlen (mat[i] + 1) == n) ;
156     }
157
158     if (tip == 1)
159         cout << Solve1(mat, n) << '\n' ;
160     else
161         cout << Solve2(mat, n, d) << '\n' ;
162
163 }
```

Capitolul 22

ONI 2016

22.1 calc

Problema 1 - calc

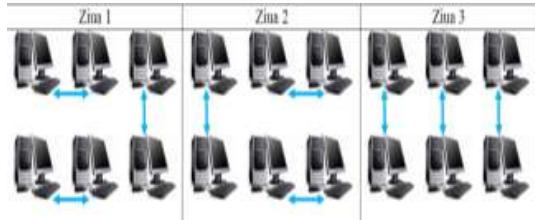
100 de puncte

La un concurs de informatică participă $2 \cdot N$ elevi împărțiți în N echipe de câte 2. Echipa poate lucra în comun la problemele propuse doar dacă au calculatoarele în rețea. Laboratorul de informatică e mai special: are $2 \cdot N$ calculatoare, distribuite pe două rânduri la distanță de un metru între ele (vertical și orizontal) și N cabluri de rețea de lungime un metru. Concursul se desfășoară pe mai multe zile și nu există două zile de concurs cu aceeași configurație a rețelei.

Exemplu: pentru $N = 3$, cei 6 elevi au fost împărțiți în 3 echipe, iar aranjarea rețelei în cele 3 zile de concurs este cea din figura alăturată.

Administratorul laboratorului vrea să memoreze în ordine lexicografică toate configurațiile folosite în zilele de concurs. Cablul orizontal se notează prin 0, iar cel vertical prin 1. Lucrândordonat și eficient, pentru cele trei zile el își va nota valorile: 001, 100, respectiv 111.

Se observă că o reprezentare de genul 000, 010, 011, 101 nu poate fi realizată.



Cerințe

Cunoscând N , să se determine:

1. Numărul de zile modulo 1 000 000 007 în care se desfășoară concursul.
2. Configurațiile laboratorului în ziua $X - 1$ și ziua $X + 1$, cunoscând configurația zilei X .

Date de intrare

Fișierul de intrare **calc.in** conține pe prima linie un număr natural p . Pentru toate testele de intrare, numărul p poate avea doar valoarea 1 sau valoarea 2.

Pe linia a doua vom avea numărul natural N .

Pe linia a treia se va găsi un sir de N cifre binare, fără spații între ele, reprezentând configurația corectă realizată de administrator în ziua X .

Date de ieșire

Dacă valoarea lui p este 1, se va rezolva numai punctul 1) din cerință.

În acest caz, în fișierul de ieșire **calc.out** se va scrie un singur număr natural Z reprezentând numărul de zile în care se desfășoară concursul pentru cele N echipe.

Dacă valoarea lui p este 2, se va rezolva numai punctul 2) din cerință.

În acest caz, fișierul de ieșire **calc.out** va conține două linii. Pe prima linie se vor scrie N cifre binare, fără spații între ele, reprezentând configurația rețelei din ziua precedentă, iar pe a doua linie N cifre binare, fără spații între ele, reprezentând configurația din ziua următoare. Dacă în ziua precedentă nu există o configurație conform cerințelor problemei, se va scrie pe prima linie valoarea -1. Dacă în ziua următoare nu există o configurație conform cerințelor problemei, se va scrie pe a doua linie valoarea -1.

Restricții și precizări

- $1 \leq N \leq 100\ 000$
- Pentru rezolvarea corectă a primei cerințe se acordă 20 de puncte, iar pentru cerința a două se acordă 80 de puncte.

Exemple:

calc.in	calc.out	Explicații
1 3 001	3	
2 3 001	-1 100	

Timp maxim de executare/test: **0.1** secunde

Memorie: total **2 MB**

Dimensiune maximă a sursei: **10 KB**

22.1.1 Indicații de rezolvare

prof. Gorea Claudiu-Cristian - Colegiul Național "Al. Papiu Ilarian", Tg-Mureș

Cerință 1 (20 puncte)

Numărul de zile de concurs este:

$\text{Fib}(n) = (\text{Fib}(n-1) + \text{Fib}(n-2))$ modulo 1.000.000.007, unde $\text{Fib}(1)=1$, $\text{Fib}(2)=2$

Cerință 2 (30 puncte)

Pentru obținerea configurației zilei precedente și zilei următoare se scade, respectiv se adună câte un bit de 1 la numărul scris în baza 2 (configurația zilei curente X), până la obținerea unei soluții valide (sevențe de 0 de lungime pară).

Descrierea soluției - 100 puncte, complexitate $O(n)$

student Budău Adrian

Cerință 2 (80 puncte)

Există mai multe posibilități de terminare a finalului configurației din ziua X.

În funcție de numărul de valori de 1 respectiv valori de 0 din finalul configurației zilei X, putem să obținem diverse cazuri:

	Ziua curentă X	Ziua precedentă (X-1)	Ziua următoare (X+1)
Cazul 1 - Impar de 1 la final 00111 00100 10000
Cazul 2 - Par de 1 la final 001111 001100 100001
Cazul 3 - Par de 0 la final 110000 100111 110011

E important sa se trateze cu atenție sirurile de forma 11111 ... 1111, respectiv 00000 ... 0000 (sau 0000 ... 0001 pentru lungimi impare) întrucât acestea nu au succesor respectiv predecesor.

22.1.2 *Rezolvare detaliată

22.1.3 Cod sursă

Listing 22.1.1: calc2.cpp

```

1 // Mircea Popoveniuc - 100 de puncte - Complexitate O(N)
2 #include<bits/stdc++.h>
3
4 using namespace std;
5

```

```

6  typedef long long int lld;
7  const int INF = (1LL << 30) - 1;
8  const lld LINF = (1LL << 62) - 1;
9  const int MOD = (int)1e9 + 7;
10 const int NMAX = 1e5;
11
12 int cerinta, N;
13 char numar[NMAX + 5];
14 char prevNr[NMAX + 5];
15 char nextNr[NMAX + 5];
16
17 void solve_a();
18 void solve_b();
19 void do_prev();
20 void do_next();
21
22 int main()
23 {
24     freopen("calc.in", "r", stdin);
25     freopen("calc.out", "w", stdout);
26
27     scanf("%d", &cerinta);
28     scanf("%d", &N);
29     scanf("%s", numar + 1);
30
31     if (cerinta == 1) solve_a();
32     else solve_b();
33
34     return 0;
35 }
36
37 void solve_a()
38 {
39     int a = 0, b = 1;
40
41     for (int i = 1, c; i <= N; i++)
42     {
43         c = (a + b) % MOD;
44         a = b;
45         b = c;
46     }
47
48     printf("%d\n", b);
49 }
50
51 void solve_b()
52 {
53     do_prev();
54     do_next();
55 }
56
57 void do_prev()
58 {
59     strcpy(nextNr + 1, numar + 1);
60
61     int zero = 0;
62     for (int i = N; i >= 1; i--)
63         if (numar[i] == '0')
64             zero++;
65         else
66             break;
67
68     if (zero == N)
69         strcpy(nextNr + 1, "-1");
70     else
71         if (zero)
72         {
73             for (int i = N; i >= N - zero + 2; i--)
74                 nextNr[i] = '1';
75
76             nextNr[N - zero] = nextNr[N - zero + 1] = '0';
77         }
78     else
79         if (numar[N] == '1' && numar[N - 1] == '1')
80             nextNr[N] = nextNr[N - 1] = '0';
81         else

```

```

82         {
83             zero = 0;
84             for (int i = N - 1; i >= 1; i--)
85                 if (numar[i] == '0')
86                     zero++;
87                 else
88                     break;
89
90             if (zero == N - 1)
91                 strcpy(nextNr + 1, "-1");
92             else
93             {
94                 for (int i = N; i >= N - zero + 1; i--)
95                     nextNr[i] = '1';
96
97                 nextNr[N - zero - 1] = nextNr[N - zero] = '0';
98             }
99         }
100
101     printf("%s\n", nextNr + 1);
102 }
103
104 void do_next()
105 {
106     strcpy(prevNr + 1, numar + 1);
107
108     int unu = 0;
109     for (int i = N; i >= 1; i--)
110         if (numar[i] == '1')
111             unu++;
112         else
113             break;
114
115     if (unu == N)
116         strcpy(prevNr + 1, "-1");
117     else
118         if (unu % 2 == 0 && unu)
119         {
120             for (int i = N - 1; i >= N - unu; i--)
121                 prevNr[i] = '0';
122
123             prevNr[N - unu - 1] = '1';
124         }
125     else
126         if (unu % 2 == 1 && unu)
127         {
128             for (int i = N; i >= N - unu; i--)
129                 prevNr[i] = '0';
130
131             prevNr[N - unu - 1] = '1';
132         }
133     else
134         if (numar[N] == '0' && numar[N - 1] == '0')
135             prevNr[N] = prevNr[N - 1] = '1';
136
137     printf("%s\n", prevNr + 1);
138 }
```

Listing 22.1.2: calc3.cpp

```

1 // Adrian Budau - 100 de puncte - Complexitate O(N)
2 #include <iostream>
3 #include <fstream>
4
5 using namespace std;
6
7 static const int kModulo = 1000 * 1000 * 1000 + 7;
8
9 int main()
10 {
11     ifstream cin("calc.in");
12     ofstream cout("calc.out");
13
14     int tip; cin >> tip;
15     int N; cin >> N;
```

```

16     string S; cin >> S;
17
18     if (tip == 1)
19     {
20         int A = 1, B = 1;
21         for (int i = 2; i <= N; ++i)
22         {
23             int C = (A + B) % kModulo;
24             A = B; B = C;
25         }
26
27         cout << B << "\n";
28         return 0;
29     }
30
31     if (N == 1)
32     {
33         cout << -1 << "\n" << -1 << "\n";
34         return 0;
35     }
36
37     string prev = S, next = S;
38
39     // predecessor
40     if (S[N - 1] == '1' && S[N - 2] == '1')
41         prev[N - 1] = prev[N - 2] = '0';
42     else
43     {
44         int i;
45         for (i = N - 2; i >= 0 && S[i] == '0'; --i);
46         if (i < 0)
47             prev = "-1";
48         else
49         {
50             prev[i] = '0';
51             for (i += 2; i < N; ++i)
52                 prev[i] = '1';
53         }
54     }
55
56     // successor
57     if (S[N - 1] == '0' && S[N - 2] == '0')
58         next[N - 1] = next[N - 2] = '1';
59     else
60     {
61         int i;
62         for (i = N - 1; i >= 0 && S[i] == '1'; --i);
63
64         if (i < 0)
65             next = "-1";
66         else
67         {
68             next[i - 1] = '1';
69             next[i] = '0';
70             int oldi = i;
71             for (++i; i < N; ++i)
72                 next[i] = '0';
73             if ((N - 1 - oldi) % 2 == 0)
74                 next[N - 1] = '1';
75         }
76     }
77
78     cout << prev << "\n" << next << "\n";
79 }
```

Listing 22.1.3: calc4.cpp

```

1 //prof.Gorea Claudiu-Cristian
2 //Colegiul National Al. Papiu Ilarian Tg-Mures
3 //complexitate 2^N = 50p
4 #include <iostream>
5
6 using namespace std;
7
8 #define modulo 1000000007
```

```

9
10 ifstream fin ("calc.in");
11 ofstream fout("calc.out");
12
13 int p,n,i,j,l,fib[100002], a[100002],b[100002],sol,imp,ok,k0;
14 char c;
15
16 int main()
17 {
18     fin>>p;
19     fin>>n;
20     if (p==1)
21     {
22         fib[1]=1;
23         fib[2]=2;
24         for(i=3;i<=n;i++)
25             fib[i]=(fib[i-1]+fib[i-2])%modulo;
26         fout<<fib[n];    fout<<endl;
27     }
28     else
29     {
30         ///cerinta 2;
31         for(i=1;i<=n;i++)
32         {
33             fin>>c;
34             a[i]=c-'0';
35             b[i]=a[i];
36         }
37
38         sol=0;
39         while(sol==0)
40         {
41             //scaderea din a
42             if(a[n]==1) a[n]--, imp=0;
43             else a[n]=1, imp=1;
44             if (imp==1)
45             {
46                 i=n-1;
47                 while(a[i]==0 && i>=1)
48                 {
49                     a[i]=1; i--;
50                 }
51                 if (i>0) a[i]--;
52                 else sol=-1; //nu mai am din ce sa scad... -1 ca solutie
53             }
54
55             ok=1; //presupun secenta ok...
56             i=1;
57             while(i<=n && ok)
58             {
59                 k0=0;
60                 while(a[i]==0 && i<=n)
61                     i++, k0++;
62
63                 if (k0%2==1)
64                     ok=0; //secenta impara de 0-uri
65
66                 while(a[i]==1 && i<=n)
67                     i++;
68             }
69
70             if (sol==0 && ok==1)
71                 sol=1;
72         }
73
74         if (sol==-1)
75             fout<<"-1\n";
76         else
77         {
78             for(i=1;i<=n;i++)
79                 fout<<a[i];
80             fout<<endl;
81         }
82
83         //ziua urmatoare.....
84         sol=0;

```

```

85         while(sol==0)
86     {
87         b[n]++;
88         i=n;
89         while(b[i]==2)
90     {
91             b[i]=0;
92             b[i-1]++;
93             i--;
94         }
95         if(b[0]>0)
96             sol=-1; //nu mai am ce aduna....
97         else
98     {
99         ok=1; //presupun secventa ok...
100        i=1;
101        while(i<=n && ok)
102    {
103            k0=0;
104            while(b[i]==0 && i<=n)      i++, k0++;
105            if(k0%2==1) ok=0; //secventa impara de 0-uri
106            while(b[i]==1 && i<=n)      i++;
107        }
108        if(ok) sol=1;
109    }
110    }
111    if(sol==1)
112        fout<<"1\n";
113    else
114    {
115        for(i=1;i<=n;i++)
116            fout<<b[i];
117        fout<<endl;
118    }
119}
120 return 0;
121 }
```

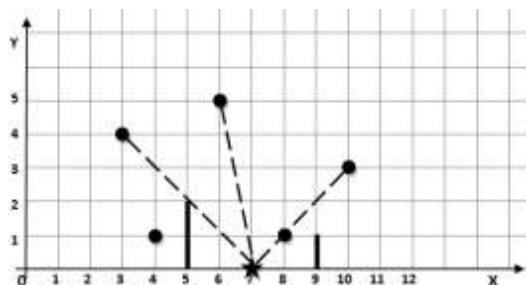
22.2 elmer

Problema 2 - elmer

100 de puncte

În antrenamentul său intens pentru prinderea lui Daffy Duck, celebrul vânător Elmer Fudd a început să vâneze rațe în orașul său preferat, Craiova. Se știe că există N rațe reprezentate prin puncte în planul de coordonate xOy , având coordonatele (x, y) și M ziduri sub forma unor segmente verticale având un capăt pe axa Ox și o anumită înălțime fiecare.

Vânătorul Elmer dorește să împuște cât mai multe rațe. El poate fi poziționat în orice punct de abscisă număr natural nenul, pe axa Ox . O rață poate fi ochită de vânător dacă zidul nu blochează glonțul vânătorului, adică segmentul imaginar delimitat de rață și de vânător nu se intersectează cu niciun zid.



Cerințe

Să se afle numărul maxim de rațe care pot fi ochite de vânătorul Elmer.

Date de intrare

Fișierul de intrare **elmer.in** conține pe prima linie numărul natural N , reprezentând numărul de rațe. Pe următoarele N linii se află perechi de numere naturale, reprezentând coordonatele rațelor. Pe următoarea linie se află numărul natural M , reprezentând numărul de ziduri, iar pe următoarele M linii, perechi de numere naturale, reprezentând abscisa și înălțimea fiecărui zid.

Date de ieșire

Fișierul de ieșire **elmer.out** va conține pe prima linie numărul maxim de rațe care pot fi ochite de Elmer.

Restricții și precizări

- $1 \leq N, M \leq 1\,000$
- Coordonatele rațelor și ale zidurilor, precum și înălțimile zidurilor sunt din intervalul $[1, 1\,000\,000\,000]$
- Se consideră numai coordonate întregi pozitive pentru vânător care nu corespund cu coordonatele niciunui zid.
 - Dacă glonțul trece prin vârful unui zid, se consideră că poate trece de el.
 - Se garantează că nu există ziduri cu aceeași abscisa, nici rațe aflate la aceeași coordonate și nici rațe care să fie "în zid" (adică nici o rață nu se află pe segmentul închis delimitat de capetele unui zid).
 - Pentru 15% din teste, se garantează faptul că $1 \leq N, M \leq 50$ și vânătorului îi este suficient să se poționeze în intervalul $[1, 1\,000]$ pentru a putea împușca numărul maxim de rațe.
 - Pentru alte 25% din teste, se garantează doar faptul că $1 \leq N, M \leq 50$.

Exemple:

elmer.in	elmer.out	Explicații
5 4 1 3 4 6 5 8 1 10 3 2 5 2 9 1	4	
6 5 4 10 10 1 9 7 5 10 2 5 1 1 8 3	5	

Timp maxim de executare/test: **1.0** secunde pe Windows **0.3** secunde pe Linux

Memorie: total **32 MB**

Dimensiune maximă a sursei: **10 KB**

22.2.1 Indicații de rezolvare

Soluție 15 puncte - $O(N * M * VMAX)$

stud. Popoveniuc Mircea

$VMAX$ = poziția maximă unde îi este suficient vânătorului să fie pozitionat pentru a putea împușca numărul maxim de rață

Se fixează fiecare poziție de pe axa Ox între 1 și $VMAX$ pentru vânător și, pentru fiecare poziție fixată, se verifică pentru fiecare rață dacă poate fi sau nu ochită. Pentru verificare, se ia fiecare zid la rând și se verifică dacă segmentul delimitat de poziția vânătorului și rață se intersectează cu segmentul delimitat de capetele zidului. Complexitatea totală a soluției este $O(N * M * VMAX)$.

Soluție 40 puncte - $O((N * M)^2)$

stud. Budău Adrian

Pentru fiecare rață, se caută pentru zidurile din dreapta raței cea mai din stânga poziție din care rața poate fi ochită, iar pentru zidurile din stânga raței cea mai din dreapta poziție din care rața poate fi ochită. Pentru pozițiile găsite, se află câte rațe pot fi împușcate, la fel ca la soluția de 15 puncte. Se obțin $N*M$ poziții "speciale" ce trebuie verificate, iar complexitatea verificării unei anumite poziții este $N*M$, aşadar complexitatea totală a soluției este $O((N * M)^2)$.

Soluție 100 puncte - $O((N * M) * \log(N * M))$

stud. Popoveniuc Mircea

Pentru fiecare rață și pereche de ziduri consecutive, se află subintervalul în care rața este vizibilă.

Asemănător soluției de 40 puncte, pentru aflarea subintervalului se utilizează pozițiile "speciale", dar la aflarea lor se mai acordă atenție cazurilor în care un zid mai apropiat de rață ar bloca mai tare vizibilitatea raței față de un zid mai depărtat, ținându-se o stivă pentru zidurile din stânga raței și una pentru cele din dreapta raței.

Se obțin maxim $N*M$ subintervale care ar crește numărul de rațe vizibile cu 1. Se aplică un *algoritm de baleiere* în care capetele subintervalelor reprezintă evenimente, capătul stâng incrementând numărul de rațe curente, iar capătul drept decrementându-l. Evenimentele trebuie sortate și apoi se iterează prin ele, obținându-se o complexitate de $O((N * M) * \log(N * M))$.

22.2.2 *Rezolvare detaliată

22.2.3 Cod sursă

Listing 22.2.1: elmer1.cpp

```

1 // Popoveniuc Mircea - O((N*M)*log(N*M)) - 100p
2 #include<bits/stdc++.h>
3
4 using namespace std;
5
6 typedef long long int lld;
7 typedef pair<lld, lld> Point;
8
9 const lld INF = (1LL << 60) - 1;
10 const lld NMAX = 1e3;
11 const lld MMAX = 1e3;
12
13 void read();
14 void solve();
15 void print();
16
17 int N, M, sol;
18
19 vector<Point> ducks;
20 vector<Point> walls;
21 vector<Point> E;
22
23 struct eventCmp
24 {
25     bool operator() (const Point& e1, const Point &e2) const
26     {
27         if (e1.first == e2.first)
28             return e1.second > e2.second;
29         return e1.first < e2.first;
30     }
31 }
```

```

31  };
32
33 lld cp(Point& A, Point& B, Point& C)
34 {
35     return (A.first - C.first) * 1LL * (B.second - C.second)
36         - (A.second - C.second) * 1LL * (B.first - C.first);
37 }
38
39 lld find_min_pos(Point& duck, vector<Point>& S)
40 {
41     if (duck.second <= S[0].second)
42         return INF;
43     double x = (S[0].first * 1LL * duck.second
44                 - duck.first * 1LL * S[0].second) *
45                 1.0 / (duck.second - S[0].second);
46     return (fabs(x) - (lld)x <= 1e-6) ? x : x + 1.0;
47 }
48
49 lld find_max_pos(Point& duck, vector<Point>& S)
50 {
51     if (duck.second <= S[0].second)
52         return -INF;
53     double x = (S[0].first * 1LL * duck.second -
54                 duck.first * 1LL * S[0].second)
55                 * 1.0 / (duck.second - S[0].second);
56     return x;
57 }
58
59 int main() {
60     freopen("elmer.in", "r", stdin);
61     freopen("elmer.out", "w", stdout);
62
63     read();
64     solve();
65     print();
66
67     return 0;
68 }
69
70 void read()
71 {
72     scanf("%d", &N);
73
74     assert(1 <= N && N <= 1000);
75
76     for (int i = 1, x, y; i <= N; i++)
77     {
78         scanf("%d%d", &x, &y);
79         assert(1 <= x && x <= 1000000000);
80         assert(1 <= y && y <= 1000000000);
81         ducks.push_back(make_pair(x, y));
82     }
83
84     walls.push_back(make_pair(0, 0));
85
86     scanf("%d", &M);
87
88     assert(1 <= M && M <= 1000);
89
90     for (int i = 1, x, h; i <= M; i++)
91     {
92         scanf("%d%d", &x, &h);
93         assert(1 <= x && x <= 1000000000);
94         assert(1 <= h && h <= 1000000000);
95         walls.push_back(make_pair(x, h));
96     }
97
98     M += 2;
99
100    walls.push_back(make_pair(INF, 0));
101 }
102
103 void solve()
104 {
105     lld L[NMAX + 5];
106     lld R[NMAX + 5];

```

```

107
108     sort(ducks.begin(), ducks.end());
109     sort(walls.begin(), walls.end());
110
111     for (int i = 0; i < N; i++)
112     {
113         Point duck = ducks[i];
114
115         int lb = (int)(lower_bound(walls.begin(), walls.end(), duck)
116                         - walls.begin());
117         vector<Point> S;
118
119         for (int j = lb; j < M; j++)
120         {
121             Point wall = walls[j];
122
123             while (!S.empty() && cp(duck, S.back(), wall) >= 0)
124                 S.pop_back();
125
126             S.push_back(wall);
127
128             L[j] = find_min_pos(duck, S);
129             R[j] = INF - 1;
130         }
131
132         S.clear();
133
134         for (int j = lb - 1; j >= 0; j--)
135         {
136             Point wall = walls[j];
137
138             while (!S.empty() && cp(duck, S.back(), wall) <= 0)
139                 S.pop_back();
140
141             S.push_back(wall);
142
143             L[j] = -INF + 1;
144             R[j] = find_max_pos(duck, S);
145         }
146
147         for (int j = 0; j + 1 < M; j++)
148         {
149             lld l, r;
150
151             l = max(walls[j].first + 1LL, L[j]);
152             r = min(walls[j + 1].first - 1LL, R[j + 1]);
153
154             if (l <= r)
155             {
156                 E.push_back(make_pair(l, 1));
157                 E.push_back(make_pair(r, -1));
158             }
159         }
160     }
161
162     sort(E.begin(), E.end(), eventCmp());
163
164     int hit = 0;
165
166     for (auto event = E.begin(); event != E.end(); event++)
167     {
168         hit += event->second;
169         sol = max(sol, hit);
170     }
171 }
172
173 void print()
174 {
175     printf("%d\n", sol);
176 }
```

Listing 22.2.2: elmer2.cpp

```

1 // Budau Adrian - O((N*M)*log(N*M)) - 100p
2 #include <iostream>
```

```

3 #include <fstream>
4 #include <vector>
5 #include <cassert>
6 #include <vector>
7 #include <limits>
8 #include <algorithm>
9
10 using namespace std;
11
12 static const int64_t kInfinite = numeric_limits<int64_t>::max() - 5;
13
14 int64_t intersect(pair<int, int> duck, pair<int, int> wall, bool right=false)
15 {
16     if (wall.first == duck.first)
17         return wall.first + 1;
18
19     int64_t prod = 1LL * (wall.first - duck.first) * duck.second;
20     int64_t when = prod / (duck.second - wall.second);
21
22     if (prod % (duck.second - wall.second))
23     {
24         if (right && prod > 0)
25             ++when;
26         else
27             if (!right && prod < 0)
28                 --when;
29     }
30
31     return when + duck.first;
32 }
33
34 int main()
35 {
36     ifstream cin("elmer.in");
37     ofstream cout("elmer.out");
38
39     int N;
40     cin >> N;
41
42     assert(1 <= N && N <= 1000);
43
44     vector< pair<int, int> > duck(N);
45     for (int i = 0; i < N; ++i)
46         cin >> duck[i].first >> duck[i].second;
47
48     int M;
49     cin >> M;
50
51     vector< pair<int, int> > wall(M);
52     for (int j = 0; j < M; ++j)
53         cin >> wall[j].first >> wall[j].second;
54
55     sort(wall.begin(), wall.end());
56
57     vector< pair<int64_t, int> > moments;
58     moments.reserve(N * M);
59     for (int i = 0; i < N; ++i)
60     {
61         int left = lower_bound
62             (
63                 wall.begin(), wall.end(), duck[i],
64                 [&](pair<int, int> wall, pair<int, int> duck)
65                 {
66                     return wall.first < duck.first;
67                 })
68             - wall.begin();
69
70         int right = left;
71
72         int64_t to = kInfinite;
73         if (left < M)
74             to = wall[left].first - 1;
75
76         while (left > 0)
77         {
78             if (to > wall[left - 1].first)

```

```

79         {
80             moments.emplace_back(wall[left - 1].first + 1, 1);
81             moments.emplace_back(to + 1, -1);
82         }
83
84         if (duck[i].second <= wall[left - 1].second)
85             break;
86
87         to = min(to, intersect(duck[i], wall[left - 1]));
88         --left;
89     }
90
91     if (left == 0 && to > 0)
92     {
93         moments.emplace_back(1, 1);
94         moments.emplace_back(to + 1, -1);
95     }
96
97     if (right == M || duck[i].second <= wall[right].second)
98         continue;
99
100    int64_t from = intersect(duck[i], wall[right], true);
101    ++right;
102    while (right < M)
103    {
104        if (from < wall[right].first)
105        {
106            moments.emplace_back(from, 1);
107            moments.emplace_back(wall[right].first, -1);
108        }
109
110        if (duck[i].second <= wall[right].second)
111            break;
112
113        from = max(from, intersect(duck[i], wall[right], true));
114        ++right;
115    }
116
117    if (right == M)
118        moments.emplace_back(from, 1);
119    }
120
121    int answer = 0;
122
123    sort(moments.begin(), moments.end());
124
125    int aux = 0;
126    for (int i = 0, j; i < int(moments.size()); i = j)
127    {
128        for (j = i;
129            j < int(moments.size()) && moments[j].first == moments[i].first;
130            ++j)
131            aux += moments[j].second;
132
133        answer = max(answer, aux);
134    }
135
136    cout << answer << "\n";
137}

```

Listing 22.2.3: elmer3.cpp

```

1 // Popoveniuc Mircea - O((N*M)^2) - 40p
2 #include<bits/stdc++.h>
3
4 using namespace std;
5
6 #define dbg(x) (cout<<#x<<" = "<<(x)<<' \n' )
7
8 typedef long long int lld;
9 typedef pair<lld, lld> Point;
10 const lld INF = (1LL << 60) - 1;
11 const lld NMAX = 1e3;
12 const lld MMAX = 1e3;
13

```

```

14 void read();
15 void solve();
16 void print();
17
18 int N, M, sol;
19 vector<Point> ducks;
20 vector<Point> walls;
21 vector<Point> E;
22
23 struct eventCmp
24 {
25     bool operator()(const Point& e1, const Point &e2) const {
26         if (e1.first == e2.first)
27             return e1.second > e2.second;
28         return e1.first < e2.first;
29     }
30 };
31
32 lld cp(Point& A, Point& B, Point& C)
33 {
34     return (A.first - C.first) * 1LL * (B.second - C.second)
35         - (A.second - C.second) * 1LL * (B.first - C.first);
36 }
37
38 lld find_min_pos(Point& duck, Point& wall)
39 {
40     if (duck.second <= wall.second)
41         return INF;
42     double x = (wall.first * 1LL * duck.second
43                 - duck.first * 1LL * wall.second)*1.0/(duck.second-wall.second);
44
45     return (fabs(x) - (lld)x <= 1e-6) ? x : x + 1.0;
46 }
47
48 lld find_max_pos(Point& duck, Point& wall)
49 {
50     if (duck.second <= wall.second)
51         return -INF;
52     double x = (wall.first * 1LL * duck.second
53                 - duck.first * 1LL * wall.second)*1.0/(duck.second-wall.second);
54
55     return x;
56 }
57
58 bool intersect(Point wall, Point duck, Point hunter)
59 {
60     if ((hunter <= wall && wall <= duck) || (hunter >= wall && wall >= duck))
61     {
62         if (hunter >= duck)
63             return (cp(hunter, duck, wall) < 0);
64         else
65             return (cp(hunter, duck, wall) > 0);
66     }
67
68     return false;
69 }
70
71 int main()
72 {
73     freopen("elmer.in", "r", stdin);
74     freopen("elmer.out", "w", stdout);
75
76     read();
77     solve();
78     print();
79
80     return 0;
81 }
82
83 void read()
84 {
85     scanf("%d", &N);
86
87     assert(1 <= N && N <= 1000);
88
89     for (int i = 1, x, y; i <= N; i++)

```

```

90     {
91         scanf("%d%d", &x, &y);
92         assert(1 <= x && x <= 1000000000);
93         assert(1 <= y && y <= 1000000000);
94         ducks.push_back(make_pair(x, y));
95     }
96
97     scanf("%d", &M);
98
99     assert(1 <= M && M <= 1000);
100
101    for (int i = 1, x, h; i <= M; i++)
102    {
103        scanf("%d%d", &x, &h);
104        assert(1 <= x && x <= 1000000000);
105        assert(1 <= h && h <= 1000000000);
106        walls.push_back(make_pair(x, h));
107    }
108 }
109
110 int solutie_pe_pozitie(lld x)
111 {
112     if (x <= 0) return -1;
113     int hit = 0;
114
115     for (auto duck = ducks.begin(); duck != ducks.end(); duck++)
116     {
117         int w = 0;
118
119         for (auto wall = walls.begin(); wall != walls.end(); wall++)
120         {
121             if (wall->first == x) w++;
122             else if (intersect(*wall, *duck, make_pair(x, 0))) w++;
123         }
124
125         if (!w) hit++;
126     }
127
128     return hit;
129 }
130
131 void solve()
132 {
133     sort(ducks.begin(), ducks.end());
134     sort(walls.begin(), walls.end());
135
136     for (lld i = 0; i < N; i++)
137     {
138         Point duck = ducks[i];
139
140         lld lb = (lld)(lower_bound(walls.begin(),
141                                     walls.end(), duck) - walls.begin());
142
143         for (lld j = lb; j < M; j++)
144         {
145             lld pozitie = find_min_pos(duck, walls[j]);
146             sol = max(sol, solutie_pe_pozitie(pozitie));
147         }
148
149         for (lld j = lb - 1; j >= 0; j--)
150         {
151             lld pozitie = find_max_pos(duck, walls[j]);
152             sol = max(sol, solutie_pe_pozitie(pozitie));
153         }
154     }
155 }
156
157 void print()
158 {
159     printf("%d\n", sol);
160 }
```

Listing 22.2.4: elmer4.cpp

```

2 #include<bits/stdc++.h>
3
4 using namespace std;
5
6 const int VMAX = 1000;
7 const int NMAX = 50;
8 const int MMAX = 50;
9
10 void read();
11 void solve();
12 void print();
13
14 int N, M, sol;
15 vector<pair<int, int>> ducks, walls;
16
17 int cp(pair<int, int> A, pair<int, int> B, pair<int, int> C)
18 {
19     return (A.first - C.first) * 1LL * (B.second - C.second)
20     - (A.second - C.second) * 1LL * (B.first - C.first);
21 }
22
23 bool intersect(pair<int,int> wall, pair<int,int> duck, pair<int,int> hunter)
24 {
25     if ((hunter <= wall && wall <= duck) || (hunter >= wall && wall >= duck))
26     {
27         if (hunter >= duck)
28             return (cp(hunter, duck, wall) < 0);
29         else
30             return (cp(hunter, duck, wall) > 0);
31     }
32
33     return false;
34 }
35
36 int main()
37 {
38     freopen("elmer.in", "r", stdin);
39     freopen("elmer.out", "w", stdout);
40
41     read();
42     solve();
43     print();
44
45     return 0;
46 }
47
48 void read()
49 {
50     scanf("%d", &N);
51
52     assert(1 <= N && N <= 1000);
53
54     for (int i = 1, x, y; i <= N; i++)
55     {
56         scanf("%d%d", &x, &y);
57         assert(1 <= x && x <= 1e9);
58         assert(1 <= y && y <= 1e9);
59         ducks.push_back(make_pair(x, y));
60     }
61
62     scanf("%d", &M);
63
64     assert(1 <= M && M <= 1000);
65
66     for (int i = 1, x, h; i <= M; i++)
67     {
68         scanf("%d%d", &x, &h);
69         assert(1 <= x && x <= 1e9);
70         assert(1 <= h && h <= 1e9);
71         walls.push_back(make_pair(x, h));
72     }
73 }
74
75 void solve()
76 {
77     for (int x = 1; x <= VMAX; x++)

```

```

78     {
79         int hit = 0;
80
81         for (auto duck = ducks.begin(); duck != ducks.end(); duck++)
82         {
83             int w = 0;
84
85             for (auto wall = walls.begin(); wall != walls.end(); wall++)
86             {
87                 if (wall->first == x)
88                     w++;
89                 else
90                     if (intersect( *wall, *duck, make_pair(x, 0)))
91                         w++;
92             }
93
94             if (!w) hit++;
95         }
96
97         sol = max(sol, hit);
98     }
99 }
100 void print()
101 {
102     printf("%d\n", sol);
103 }
```

22.3 tort

Problema 3 - tort

100 de puncte

Pentru că s-a calificat la Olimpiada Națională de Informatică de la Craiova, NN îi pregătește lui $XORin$ un tort. Tortul este dreptunghiular, format din linii și coloane numerotate de la 1 la N pentru linii și de la 1 la M pentru coloane. Tortul este format din bucăți de dimensiune 1×1 , fiecare fiind acoperită cu un alt tip de glazură. În fiecare zi NN îi taie lui $XORin$ câte o felie, alegând cel mai mare pătrat care conține bucăți acoperite cu același tip de glazură. În cazul în care există mai multe astfel de felii, NN o alege pe cea care are colțul din dreapta jos situat pe linia cu indicele cel mai mic. Dacă și în acest caz există mai multe posibilități, el o va alege pe cea cu colțul din dreapta jos situat în coloana cu indicele cel mai mic.

Cerințe

Precizați latura și coordonatele colțului din dreapta jos pentru fiecare felie de tort primită, în ordinea specificată mai sus.

Date de intrare

Fisierul **tort.in** conține pe prima linie numerele naturale N și M , separate printr-un spațiu, reprezentând lungimea și lățimea tortului. Pe următoarele N linii se vor afla câte M caractere din mulțimea $\{0', ..., '9\}$ reprezentând tipul de glazură cu care este acoperită bucata de pe linia i și coloana j a tortului. Liniile și coloanele sunt numerotate de la 1 la N , respectiv de la 1 la M . Pe linii nu există spațiu între oricare două caractere alăturate.

Date de ieșire

În fisierul de ieșire **tort.out** se vor afișa feliile de tort în ordinea în care $XORin$ le va primi. Pentru fiecare felie se va afișa latura feliei, precum și coordonatele colțului din dreapta jos, valori separate prin câte un singur spațiu.

Restricții și precizări

- $1 \leq N, M \leq 500$
- Numerotarea liniilor și coloanelor nu se schimbă în urma operațiilor de eliminare.
- Pentru 30% din teste se garantează că $1 \leq N, M \leq 35$

Exemplu:

tort.in	tort.out	Explicații
4 7	3 3 3	Prima felie primită de <i>XORin</i> va fi cea care are colțul din dreapta jos (3,3) și latura 3.
1111111	3 4 7	A doua felie va fi cea cu colțul din dreapta jos (4,7) și latura 3.
1112333	1 1 4	Următoarea felie va fi cea cu colțul din dreapta jos (1,4) și latura 1. s.a.m.d.
1112333	1 1 5	
4444333	1 1 6	
	1 1 7	
	1 2 4	
	1 3 4	
	1 4 1	
	1 4 2	
	1 4 3	
	1 4 4	

Timp maxim de executare/test: **1.2** secunde pe Windows **0.5** secunde pe Linux

Memorie: total **32 MB**

Dimensiune maximă a sursei: **10 KB**

22.3.1 Indicații de rezolvare

stud. Petru Trîmbițaș, Universitatea Babeș-Bolyai, Cluj-Napoca

Soluție O(N2*M2) - 30 puncte

Se calculează dinamica $lm[i][j]$ reprezentând latura maximă a unui pătrat cu colțul din dreapta jos (i, j) .

La fiecare pas căutăm elementul maxim din lm , îl ștergem și actualizăm toată matricea calculată anterior.

Soluție O(N*M*min(N,M)) - 100 puncte

Adrian Budău

Latura maximă a unui pătrat poate fi $\min(N, M)$. La fiecare pas fixăm câte o latură în ordine descrescătoare, căutăm toate pătratele de acea latură și verificăm dacă nu au fost șterse încă celule din el. În locul lor completăm matricea inițială cu un caracter care nu e prezent în input.

Numim pătrat invalid un pătrat care are o parte dintre celule eliminate. Observăm că dacă pentru un pătrat de latură 1 pentru care dorim să verificăm dacă este invalid, este suficient să verificăm dacă colțurile sale nu au fost eliminate. Deoarece pătratele au fost eliminate în ordine descrescătoare a laturii înseamnă că dacă avem o intersecție, ea va intersecta o latură întreagă a noului pătrat sau minim un colț. Deoarece avem maxim $\min(N, M)$ laturi posibile, iar căutarea laturilor de o anumită lungime ia $N * M$ complexitatea finală e $O(N * M * \min(N, M))$.

Solutie $O(N * M * \log(N * M))$ - 100 puncte

Adrian Budău, Petru Trîmbițaș

Deoarece avem maxim N lungimi de laturi distincte, putem reține pentru fiecare lungime, posibilele celule unde se termină un pătrat la momentul calculării matricii de la prima soluție.

Pentru fiecare pătrat eliminat dorim să actualizăm matricea lm .

Pentru un pătrat cu colțurile în $(x-l+1, y-l+1)$ și (x, y) observăm că elementele din stânga și susul pătratului nu trebuie eliminate.

Deoarece știm că pătratul eliminat are latura maximă, observăm că nu are sens să actualizăm decât zona delimitată de colțurile $(x-l+1, y-l+1)$ și $(x+l-1, y+l-1)$.

În cazul în care eliminăm un pătrat de latură 1 avem de actualizat 3 elemente. Pentru latura 2 vom avea 5, iar pentru N avem 0.

Se observă că pentru fiecare element șters, actualizăm 3 elemente. Cum fiecare element este șters o singură dată, complexitatea totală pentru ștergeri și actualizări $O(N * M)$. Sortăm fiecare listă de celule candidate pentru a le găsi pe cele de sus.

22.3.2 *Rezolvare detaliată

22.3.3 Cod sursă

Listing 22.3.1: tort1.cpp

```

1 // Petru Trimbitas - N*M*log(N*M) - 100 puncte
2 #include <iostream>
3 #include <fstream>
4 #include <vector>
5 #include <algorithm>
6 #include <cassert>
7 #define DN 1005
8 #define x first
9 #define y second
10 using namespace std;
11
12 typedef pair<int,int> per;
13
14 int n,m,lgm,lm[DN][DN],sz,q,w;
15 char mt[DN][DN];
16 vector<per> pt[DN];
17 per el[DN*DN];
18
19 int main() {
20     ifstream f("tort.in");
21     ofstream g("tort.out");
22     f>>n>>m;
23     for(int i=1; i<=n; ++i) f>>(mt[i]+1);
24     for(int i=1; i<=n; ++i) {
25         for(int j=1; j<=m; ++j) {
26             lm[i][j]=1;
27             if(mt[i][j]==mt[i-1][j]&&mt[i][j]==mt[i][j-1]&&mt[i][j]==mt[i-1][j-1])
28                 lm[i][j]=max(lm[i][j],1+min(lm[i-1][j-1],min(lm[i-1][j],lm[i][j-1])));
29             pt[lm[i][j]].push_back(make_pair(i,j));
30             lgm=max(lm[i][j],lgm);
31         }
32     }
33     for(int lgc=lgm; lgc>0; --lgc) {
34         sort(pt[lgc].begin(), pt[lgc].end());
35         for(auto j:pt[lgc]) {
36             int x=j.x,y=j.y;
37             if(lm[x][y]<lgc) continue;
38             g<<lgc<<' '<<x<<' '<<y<<' \n';
39             for(int i=-lgc+1; i<lgc; ++i) for(int j=-lgc+1; j<lgc; ++j) {
40                 int l=x+i,c=y+j;
41                 if(l>n || c>m) continue;
42                 if(i<=0 && j<=0) {
43                     lm[l][c]=0;
44                     mt[l][c]='-';
45                 } else if(mt[l][c]!='-') {
46                     lm[l][c]=1;
47                     if(mt[l][c]==mt[l-1][c]&&
48                         mt[l][c]==mt[l][c-1]&&mt[l][c]==mt[l-1][c-1])
49                         lm[l][c]=max(lm[l][c],1+min(lm[l-1][c-1],
50                                         min(lm[l-1][c],lm[l][c-1])));
51                     pt[lm[l][c]].push_back(make_pair(l,c));
52                 }
53             }
54         }
55     }
56 }
```

Listing 22.3.2: tort2.cpp

```

1 // Petru Trimbitas - N*M*min(N,M) - 100 puncte
2 #include <iostream>
3 #include <fstream>
4 #include <vector>
5 #include <algorithm>
6 #include <cassert>
7
8 #define DN 1005
9 #define x first
10 #define y second
```

```

11
12  using namespace std;
13
14  int n,m,lm[DN][DN],lgm;
15  char mt[DN][DN];
16
17  int main()
18  {
19      ifstream f("tort.in");
20      ofstream g("tort2.out");
21      f>>n>>m;
22      for(int i=1; i<=n; ++i) f>>(mt[i]+1);
23      for(int i=1; i<=n; ++i) {
24          for(int j=1; j<=m; ++j) {
25              lm[i][j]=1;
26              if(mt[i][j]==mt[i-1][j]&&mt[i][j]==mt[i][j-1]&&mt[i][j]==mt[i-1][j-1])
27                  lm[i][j]=max(lm[i][j],1+min(lm[i-1][j-1],min(lm[i-1][j],lm[i][j-1])));
28              lgm=max(lgm,lm[i][j]);
29          }
30      }
31      for(int lgc=lgm; lgc; --lgc) {
32          int ok=0;
33          for(int i=1; i<=n; ++i) for(int j=1; j<=m; ++j)
34              if(lm[i][j]>=lgc && mt[i-lgc+1][j-lgc+1]!='-' && mt[i-lgc+1][j]!=='-' &&
35                  mt[i][j-lgc+1]!=='-' && mt[i][j]!=='-')
36                  {
37                      g<<lgc<<' '<<i<<' '<<j<<'\n';
38                      for(int x=i-lgc+1; x<=i; ++x)
39                          for(int y=j-lgc+1; y<=j; ++y)
40                              mt[x][y]='-';
41                      ok=1;
42                  }
43          lgc+=ok;
44      }
45  }

```

Listing 22.3.3: tort3.cpp

```

1  /**
2   * Solutie de complexitate O(N * M * log(N))
3   *
4   * @author Adrian Budau
5   */
6  #include <iostream>
7  #include <fstream>
8  #include <cassert>
9  #include <vector>
10 #include <algorithm>
11 #include <queue>
12
13 using namespace std;
14
15 int main()
16 {
17     ifstream cin("tort.in");
18     ofstream cout("tort.out");
19
20     int N, M;
21     assert(cin >> N >> M);
22     assert(1 <= N && N <= 1500);
23     vector<string> S(N);
24     for (int i = 0; i < N; ++i)
25     {
26         assert(cin >> S[i]);
27         assert(int(S[i].size()) == M);
28         for (auto &c : S[i])
29             assert(c >= '0' && c <= '9');
30     }
31
32     vector<vector<int> > dp(N, vector<int>(M, 1));
33
34     for (int i = N - 2; i >= 0; --i)
35         for (int j = M - 2; j >= 0; --j)
36             if (S[i][j] == S[i + 1][j] &&
37                 S[i][j] == S[i][j + 1] &&

```

```

38         S[i][j] == S[i + 1][j + 1])
39         dp[i][j] = min(dp[i + 1][j],
40                           min(dp[i][j + 1], dp[i + 1][j + 1])) + 1;
41
42
43     vector< vector< pair<int, int> > > has(min(N, M) + 1);
44     for (int i = 0; i < N; ++i)
45         for (int j = 0; j < M; ++j)
46             has[dp[i][j]].emplace_back(i, j);
47
48     for (int L = min(N, M); L > 0; --L)
49     {
50         sort(has[L].begin(), has[L].end());
51         for (auto &p : has[L])
52         {
53             int i = p.first;
54             int j = p.second;
55
56             if (S[i][j] == '-')
57                 continue;
58
59             if (dp[i][j] == L && S[i + L - 1][j] != '-' &&
60                 S[i][j + L - 1] != '-' && S[i + L - 1][j + L - 1] != '-')
61             {
62                 cout << L << " " << i + L << " " << j + L << "\n";
63                 for (int k = 0; k < L; ++k)
64                     for (int p = 0; p < L; ++p)
65                         S[i + k][j + p] = '-';
66             }
67             else
68             {
69                 int step;
70                 for (step = 1; step <= L; step <= 1);
71                 int newL;
72                 for (newL = 0; step; step >= 1)
73                     if (newL + step <= L &&
74                         i + newL + step <= N && j + newL + step <= M)
75                         if (S[i + newL + step - 1][j] != '-' &&
76                             S[i][j + newL + step - 1] != '-' &&
77                             S[i + newL + step - 1][j + newL + step - 1] != '-')
78                             newL += step;
79
80                 dp[i][j] = newL;
81                 has[newL].emplace_back(i, j);
82             }
83         }
84     }

```

Listing 22.3.4: tort4.cpp

```

1 //Petru Trimbitas - N^2*M^2 - 30 puncte
2 #include <iostream>
3 #include <fstream>
4 #include <vector>
5 #include <algorithm>
6 #include <cassert>
7
8 #define DN 1005
9 #define x first
10 #define y second
11
12 using namespace std;
13
14 int n,m,lm[DN][DN];
15 char mt[DN][DN];
16
17 int main()
18 {
19     ifstream f("tort.in");
20     ofstream g("tort.out");
21     f>>n>>m;
22     for(int i=1; i<=n; ++i) f>>(mt[i]+1);
23     int lgm=-1,x,y;
24     while(lgm) {
25         lgm=0;

```

```

26     for(int i=1; i<=n; ++i)
27     {
28         for(int j=1; j<=m; ++j)
29         {
30             if(mt[i][j]!='-')
31             {
32                 lm[i][j]=1;
33                 if(mt[i][j]==mt[i-1][j] &&
34                     mt[i][j]==mt[i][j-1] && mt[i][j]==mt[i-1][j-1])
35                     lm[i][j]=max(lm[i][j], 1+min(lm[i-1][j-1],
36                                         min(lm[i-1][j], lm[i][j-1])));
37                 if(lm[i][j]>lqm)
38                 {
39                     lqm=lm[i][j];
40                     x=i;
41                     y=j;
42                 }
43             }
44             else
45                 lm[i][j]=0;
46         }
47     }
48     if(lqm)
49         g<<lqm<<' '<<x<<' '<<y<<'\n';
50     for(int i=x-lqm+1; i<=x; ++i)
51         for(int j=y-lqm+1; j<=y; ++j)
52             mt[i][j]='-';
53 }
54 }
```

22.4 intrus

Problema 4 - intrus

100 de puncte

Terminalul unui aeroport este o sală foarte mare având formă unui dreptunghi împărțit în patrate cu latură unitară. Aici se află mai multe persoane, care trebuie să poarte la vedere un ecuson cu un cod de bare care poate fi citit în orice moment de camerele de supraveghere și decodificat de calculatoarele serviciului de protecție și pază. Într-un patrat cu latură unitară poate să se afle doar o singură persoană la un moment dat. Sala este reprezentată printr-o matrice cu R linii și C coloane, elementele sale fiind numere naturale de cel mult 6 cifre cu valorile: 0 - pentru spațiu neocupat, respectiv numere naturale nenule, care reprezintă identificatorul (*ID*-ul) persoanelor. Printre aceste persoane există persoane infiltrate (intruși) care au *ID*-uri cu valori identice cu ale altor persoane. Dacă există două sau mai multe persoane cu același *ID*, acestea sunt considerate toate suspecte.

Intrușii vor să ajungă în apropierea unor VIP-uri (persoane importante), pentru a le înregistra discuțiile cu un microfon care poate înregistra sunete în interiorul unui patrat cu latura D , în centrul căruia se află chiar el. Acest patrat nu este cuprins neapărat integral în matricea sălii (vedeți figura alăturată)!

Prin convenție, *ID*-urile VIP-urilor sunt numere prime distincte. În plus, și un *ID* al unui VIP poate fi copiat, crescând astfel numărul suspectilor. Un VIP se caracterizează printr-un nivel de importanță: cu cât *ID*-ul este un număr mai mare, cu atât nivelul de importanță este mai mare (este "mai importantă").

Persoanele suspecte au asociat un "grad de periculozitate". Aceasta este cu atât mai mare cu cât numărul de VIP-uri aflate în interiorul patratului de latură D , în centrul căruia se află suspectul, este mai mare. Dacă există doi suspecți cu același grad de periculozitate, se consideră "mai periculoasă" persoana care are în patratul său VIP-ul cu *ID*-ul cel mai mare. În caz de egalitate, se consideră "mai periculoasă" persoana care este așezată pe o linie cu un indice mai mic, iar la egalitate de indici de linii, pe o coloană cu indice mai mic. Există și persoane suspecte cu gradul de periculozitate 0, dacă în interiorul patratului în centrul căruia se plasează nu există niciun număr prim.

Cerințe

0	0	0	0	0	10
0	82	0	0	24	25
11	0	7	17	0	0
1	31	8	0	4	0
0	0	0	23	3	0
11	0	0	0	15	0
81	4	5	0	0	0
0	30	0	0	0	0

Persoana cu ID-ul 4 din celula (4, 5) este „mai periculoasă” decât cea cu ID-ul 4 din (7, 2).
Observați că aici D este egal cu 3.

- 1) Să se determine numărul persoanelor suspecte aflate în sala de aşteptare.
 2) Să se determine ID -ul și coordonatele persoanelor suspecte, (RS_i -linia suspectului i , CS_i -coloana suspectului i) în ordinea descrescătoare a "gradului de pericolozitate".

Date de intrare

Fișierul de intrare **intrus.in** va conține pe prima linie valoarea p , care poate fi doar 1 sau 2. Linia a doua va conține valorile R , C și D , separate prin câte un spațiu. Pe următoarele R linii, câte C numere naturale de cel mult 6 cifre, separate prin câte un spațiu, reprezentând elementele matricei descrise în enunț.

Date de ieșire

Dacă $p = 1$, se cere doar rezolvarea primei cerințe. În acest caz, fișierul de ieșire **intrus.out** va conține o singură valoare T (care poate fi și 0), reprezentând numărul persoanelor suspecte.

Dacă $p = 2$, se va rezolva numai a doua cerință. În acest caz fișierul de ieșire **intrus.out** va conține pe fiecare linie câte 3 numere naturale nenule: ID_i (ID -ul intrusului i), R_i , C_i (linia, respectiv coloana în care se află intrusul), separate prin câte un spațiu. Dacă nu există niciun suspect, în prima linie a fișierului de ieșire **intrus.out** se va scrie -1.

Restricții și precizări

- $0 < R, C \leq 1000$
- $3 \leq D \leq 9$, D număr impar.
- Pentru $p = 2$ se garantează că numărul suspectilor nu depășește 10% din totalul persoanelor aflate în sală.

Exemple:

intrus.in	intrus.out	Explicații
1 3 4 3 1 0 7 3 5 2 3 0 3 2 0 1	7	$p = 1$, se rezolvă doar cerința 1. Există 2 ID -uri egale cu 2 și 3 ID -uri egale cu 3, deci avem 5 suspecti.
2 3 4 3 1 0 7 8 5 2 3 0 3 2 0 9	2 2 2 2 3 2 3 2 3 3 3 1	$p = 2$, se rezolvă doar cerința 2. Persoana cu ID -ul 2, aflată pe linia 2 și coloana 2 are cel mai mare grad de pericolozitate. Urmează ID -ul 2 din (3, 2), 3 din (2, 3) și 3 din (3, 1), care reprezintă o persoană suspectă, deși zona sa de latură D nu este cuprinsă în întregime în matricea sălii!

Timp maxim de executare/test: **1.2** secunde pe Windows, **0.5** secunde pe Linux

Memorie: total **32 MB**

Dimensiune maximă a sursei: **10 KB**

22.4.1 Indicații de rezolvare

prof. István Budai, Liceul Teoretic "Nagy Mózes" Tg. Secuiesc, jud. Covasna

Se declară

constantele de tip întreg $Rmax$ și $Cmax$ cu valoarea 1001,
 apoi o matrice X cu $Rmax$ linii și $Cmax$ coloane, cu elemente întregi,
 un vector W cu 1 000 000 de elemente numere întregi,
 o structură cu patru câmpuri:
 $elem \{int r; int c; int niv; int maxpri;\}$,
 unde r și c sunt coordonatele unui element în matrice (linie, coloană), niv este gradul de pericolozitate al elementului, $maxpri$ este numărul prim cel mai mare aflat în pătratul de latură D al elementului, descris în enunț;
 un tablou Y cu $Rmax \times Cmax$ elemente de tip $elem$.

Se implementează subprogramele:

$boolprim(int k)$ - care verifică dacă k este sau nu este prim;

intnivel(intM[][Cmax], intis, intjs, intr, intc, intd, int&mx) - care calculează gradul de pericolozitate al persoanei aflate în punctul de coordonate (is, js) , în centrul pătratului de latură d , rezultatul fiind furnizat prin parametrul mx ;

voidseek(intM[][Cmax], intnr, intii, intjj, int&i0, int&j0) - care caută poziția anterioară a unui număr din matrice care apare a doua oară pe coordonatele (ii, jj) , coordonatele punctului precedent fiind furnizate prin parametrii $i0$ și $j0$.

În programul principal se citesc datele problemei din fișierul de intrare, și se memorează în variabilele corespunzătoare.

Dacă $p = 1$, se rezolvă doar prima cerință. Pe parcursul citirii datelor, se incrementează elementul de pe poziția $X[i][j]$ al *vectorului de frecvențe* W , la fiecare citire a unui element nenul al matricei X . Tot aici se calculează valoarea maximă a indicelui vectorului W pentru care s-a realizat o incrementare. În final se face suma frecvențelor pentru elementele care au frecvența de cel puțin 2, rezultând numărul persoanelor suspecte, cerut în enunț.

Pentru $p = 2$ se rezolvă numai a doua cerință. În acest caz se citesc elementele matricei. Pentru fiecare element se calculează frecvența de apariție. Când un element apare a doua oară, se caută coordonatele pe care a apărut prima dată și ambele coordonate se adaugă la lista Y , cu toate câmpurile calculate (coordonate, grad de pericolozitate, valoarea celui mai mare număr prim din pătratul de latură D). Apoi se sortează vectorul Y , cu mare grijă la respectarea tuturor condițiilor formulate în enunț, în ordinea dată. Dacă Y nu conține elemente, în fișierul de ieșire se înregistrează valoarea -1. Dacă Y conține elemente, se trec valorile cerute în fișierul de ieșire, din câmpurile corespunzătoare ale vectorului Y , vector care în prealabil a fost sortat corespunzător cerințelor.

Complexitate: $O(Rmax * Cmax * D^2)$

22.4.2 *Rezolvare detaliată

22.4.3 Cod sursă

Listing 22.4.1: intrus1.cpp

```

1 // Istvan Budai - 100 de puncte - O(N * M * D^2)
2 #include <iostream>
3 #include <fstream>
4
5 using namespace std;
6
7 const int Rmax=1001;
8 const int Cmax=1001;
9
10 ifstream f("intrus.in");
11 ofstream g("intrus.out");
12
13 int X[Rmax][Cmax];
14 int W[1000001];
15
16 struct elem
17 //typedef struct elem
18 {
19     int r;
20     int c;
21     int niv;
22     int maxpri;
23 };
24
25 int pt=0;
26 elem Y[Rmax*Cmax+1], Temp;
27
28 bool prim(int k)
29 {
30     if (k<2) return false;
31     if (k==2) return true;
32     if (k>2 && k%2==0) return false;
33     for (int i=3; i*i<=k; i=i+2)
34         if (k%i==0)

```

```

35         return false;
36     return true;
37 }
38
39 int nivel(int M[][][Cmax], int is, int js, int r, int c, int d, int &mx)
40 {
41     int i1, j1, i2, j2, v=0, ii, jj, ns=0;
42
43     if (is-d/2>=1) i1=is-d/2; else i1=1;
44     if (js-d/2>=1) j1=js-d/2; else j1=1;
45     if (is+d/2<=r) i2=is+d/2; else i2=r;
46     if (js+d/2<=c) j2=js+d/2; else j2=c;
47
48     mx=0;
49     for (ii=i1; ii<=i2; ii++)
50         for (jj=j1; jj<=j2; jj++)
51             if (prim(M[ii][jj]) && (ii!=is || jj!=js))
52             {
53                 ns++;
54                 if (M[ii][jj]>mx) mx=M[ii][jj];
55             }
56
57     return ns;
58 }
59
60 void seek(int M[][][Cmax], int nr, int ii, int jj, int &i0, int &j0)
61 {
62     for (int ir=1; ir<=ii; ir++)
63         for (int ic=1; ic<=jj; ic++)
64             if (M[ir][ic]==nr)
65             {
66                 i0=ir;
67                 j0=ic;
68                 return;
69             }
70 }
71
72 int main()
73 {
74     int p, R, C, D, S=0, t, i0, j0, si, oi;
75     f>>p;
76     f>>R>>C>>D;
77
78     if (p==2)
79     {
80         for (si=1; si<=R; si++)
81             for (oi=1; oi<=C; oi++)
82                 f>>X[si][oi];
83
84         for (si=1; si<=R; si++)
85             for (oi=1; oi<=C; oi++)
86             {
87                 if (X[si][oi]!=0)
88                 {
89                     W[X[si][oi]]=W[X[si][oi]]+1;
90                     t=W[X[si][oi]];
91                     if (t>=2)
92                     {
93                         if (t==2)
94                         {
95                             seek(X, X[si][oi], R, C, i0, j0);
96                             pt++;
97                             Y[pt].r=i0; Y[pt].c=j0;
98                             Y[pt].niv=nivel(X, i0, j0, R, C, D, Y[pt].maxpri);
99                             pt++;
100                            Y[pt].r=si; Y[pt].c=oi;
101                            Y[pt].niv=nivel(X, si, oi, R, C, D, Y[pt].maxpri);
102                        }
103                     else
104                     {
105                         pt++;
106                         Y[pt].r=si; Y[pt].c=oi;
107                         Y[pt].niv=nivel(X, si, oi, R, C, D, Y[pt].maxpri);
108                     }
109                 }
110             }
}

```

```

111         }
112
113     for (si=1; si<pt; si++)
114         for (oi=si+1; oi<=pt; oi++)
115             if (Y[si].niv<=Y[oi].niv)
116                 {
117                     if (Y[si].niv==Y[oi].niv)
118                     {
119                         if (Y[si].maxpri<=Y[oi].maxpri)
120                         {
121                             if (Y[si].maxpri<Y[oi].maxpri)
122                             {
123                                 Temp=Y[si];
124                                 Y[si]=Y[oi];
125                                 Y[oi]=Temp;
126                             }
127                         else
128                             if (Y[si].r>=Y[oi].r)
129                             {
130                                 if (Y[si].r>Y[oi].r)
131                                     {
132                                         Temp=Y[si];
133                                         Y[si]=Y[oi];
134                                         Y[oi]=Temp;
135                                     }
136                                 else
137                                     if (Y[si].c>Y[oi].c)
138                                     {
139                                         Temp=Y[si];
140                                         Y[si]=Y[oi];
141                                         Y[oi]=Temp;
142                                     }
143                             }
144                         }
145                     else
146                     {
147                         {
148                             Temp=Y[si];
149                             Y[si]=Y[oi];
150                             Y[oi]=Temp;
151                         }
152                     }
153
154     if (pt>0)
155         for (si=1; si<=pt; si++)
156             g<<X[Y[si].r][Y[si].c]<<" "<<Y[si].r<<" "<<Y[si].c<<endl;
157     else
158     g<<-1;
159 }
160
161 else//p==1
162
163 {
164     int mm=0;
165     for (int i=1; i<=R; i++)
166         for (int j=1; j<=C; j++)
167         {
168             f>>X[i][j];
169             if (X[i][j])
170             {
171                 ++W[X[i][j]];
172                 if (X[i][j]>mm)
173                     mm=X[i][j];
174             }
175         }
176
177     for (int i=1; i<=mm; i++) if (W[i]>1) S=S+W[i];
178     g<<S;
179 }
180
181 f.close();
182 g.close();
183
184 return 0;
185 }
```

Listing 22.4.2: intrus2.cpp

```

1 // Mircea Popoveniuc - 100 de puncte - O(N * M * D^2)
2 #include<bits/stdc++.h>
3
4 using namespace std;
5
6 #define dbg(x) (cout<<#x<<" = "<<(x)<<'\\n' )
7
8 const int NMAX = 1e3;
9 const int VMAX = 1e6;
10
11 int cerinta, R, C, D;
12
13 bitset < VMAX + 5 > neprim;
14
15 int grid[NMAX + 5][NMAX + 5];
16 int S[NMAX + 5][NMAX + 5];
17 int freq[VMAX + 5];
18
19 vector<pair<pair<int, int>, pair<int, int> > > suspecti;
20
21 bool cmp(pair<pair<int, int>, pair<int, int> > A,
22           pair<int, int> > B)
23 {
24     if (A.first == B.first)
25         return A.second < B.second;
26     return A.first > B.first;
27 }
28
29 void ciur()
30 {
31     neprim[0] = neprim[1] = 1;
32
33     for (int i = 4; i <= 1e6; i += 2)
34         neprim[i] = 1;
35
36     for (int i = 3; i * i <= 1e6; i += 2)
37         if (!neprim[i])
38             for (int j = 2 * i; j <= 1e6; j += i)
39                 neprim[j] = 1;
40 }
41
42 void sume_partiale()
43 {
44     for (int i = 1; i <= R; i++)
45         for (int j = 1; j <= C; j++)
46             S[i][j] = S[i - 1][j] + S[i][j - 1]
47             - S[i - 1][j - 1] + (!neprim[grid[i][j]]);
48 }
49
50 int vipmax(int x, int y, int xmin, int ymin, int xmax, int ymax)
51 {
52     int val = 0;
53     for (int i = xmin; i <= xmax; i++)
54         for (int j = ymin; j <= ymax; j++)
55             if ((i != x || j != y) && !neprim[grid[i][j]])
56                 val = max(val, grid[i][j]);
57
58     return val;
59 }
60
61 int main()
62 {
63     cin.sync_with_stdio(false);
64
65     #ifndef ONLINE_JUDGE
66     freopen("intrus.in", "r", stdin);
67     freopen("intrus.out", "w", stdout);
68     #endif
69
70     scanf("%d", &cerinta);
71     scanf("%d%d%d", &R, &C, &D);
72
73     assert(1 <= R && R <= 1000);
74 }
```

```

75     assert(1 <= C && C <= 1000);
76     assert(3 <= D && D <= 9 && (D % 2));
77
78     for (int i = 1; i <= R; i++)
79         for (int j = 1; j <= C; j++)
80         {
81             scanf("%d", &grid[i][j]);
82             assert(0 <= grid[i][j] && grid[i][j] <= 999999);
83             if (grid[i][j]) freq[grid[i][j]]++;
84         }
85
86     ciur();
87     sume_partiale();
88
89     for (int i = 1; i <= R; i++)
90     {
91         for (int j = 1; j <= C; j++)
92             if (freq[grid[i][j]] > 1) {
93                 int xmin = max(i - D / 2, 1);
94                 int ymin = max(j - D / 2, 1);
95                 int xmax = min(i + D / 2, R);
96                 int ymax = min(j + D / 2, C);
97                 int vip = S[xmax][ymax] - S[xmin - 1][ymax]
98                     - S[xmax][ymin - 1] + S[xmin - 1][ymin - 1]
99                     - !(neprim[grid[i][j]]);
100                suspecti.push_back(
101                    make_pair(
102                        make_pair(vip, vipmax(i, j, xmin, ymin, xmax, ymax)),
103                        make_pair(i, j)));
104            }
105        }
106
107        if (cerinta == 1)
108        {
109            printf("%d\n", (int)suspecti.size());
110        }
111        else
112        {
113            sort(suspecti.begin(), suspecti.end(), cmp);
114
115            for (int i = 0, x, y; i < (int)suspecti.size(); i++)
116            {
117                x = suspecti[i].second.first;
118                y = suspecti[i].second.second;
119                printf("%d %d %d\n", grid[x][y], x, y);
120            }
121
122            if (suspecti.empty())
123                printf("-1\n");
124        }
125
126        return 0;
127    }

```

Listing 22.4.3: intrus3.cpp

```

1 // Profesor Gorea Claudiu-Cristian - 100 de puncte - O(N * M * D^2)
2 //Colegiul National "Al. Papiu Ilarian" Tg-Mures
3 #include <iostream>
4 #include <cstring>
5 #include <algorithm>
6
7 using namespace std;
8
9 struct intrus
10 {
11     int info,x,y,id,vip;
12 } v[1000002];
13
14 ifstream fin ("intrus.in");
15 ofstream fout("intrus.out");
16
17 int a[1002][1002],viz[1000002];
18 int n,m,i,j,l,c,cerinta,d,suspect,maxid, lim,li,lf,ci,cf;
19 bool ciur[1000002];

```

```

20
21 int ordonare(intrus a, intrus b)
22 {
23     if (a.vip<b.vip) return 0;
24
25     if ((a.vip==b.vip) and (a.id<b.id)) return 0;
26
27     if ((a.vip==b.vip) and (a.id==b.id) and ((a.x*m+a.y)>(b.x*m+b.y))) )
28         return 0;
29
30     return 1;
31 }
32
33 int main()
34 {
35     fin>>cerinta>>n>>m>>d;      //cerinta,n,m lungimile
36     maxid=0;
37     suspect=0;
38     for(i=1; i<=n; i++)
39         for(j=1; j<=m; j++)
40         {
41             fin>>a[i][j];
42             //if(a[i][j])
43             viz[a[i][j]]++;
44             maxid=max(maxid, a[i][j]);
45         }
46
47     viz[0]=0;
48     //ciurul false initial
49     ciur[0]=ciur[1]=true;
50     for(i=1; i<=maxid; i++)
51         if (ciur[i]==false)
52             for(j=i+i; j<=maxid; j+=i)
53                 ciur[j]=true; // nu mai e prim
54
55     if (cerinta==1)
56     {
57         suspect=0;
58         for(i=1; i<=maxid; i++)
59             if (viz[i]>1) suspect+=viz[i];
60         fout<<suspect<<'\\n';
61     }
62     else
63     {
64         suspect=0;
65         lim=(d-1)/2; //d-impar
66         for(i=1; i<=n; i++)
67             for(j=1; j<=m; j++)
68                 if (viz[a[i][j]]>1) // are id multiplicat
69                 {
70                     suspect++;
71                     v[suspect].info=a[i][j];
72                     v[suspect].x=i;
73                     v[suspect].y=j;
74                     v[suspect].id=0;
75                     li=max(i-lim,1);
76                     lf=min(i+lim,n);
77                     ci=max(j-lim,1);
78                     cf=min(j+lim,m);
79                     for(l=li; l<=lf; l++)
80                         for(c=ci; c<=cf; c++)
81                             if (ciur[a[l][c]]==false)
82                             {
83                                 if (l!=i or c!=j)
84                                     v[suspect].vip++;
85                                     v[suspect].id=max(v[suspect].id,a[l][c]);
86                             }
87                         }
88                     sort(v+1,v+1+suspect,ordonare);
89                     if (suspect<1)
90                         fout<<-1<<endl;
91                     else
92                         for (i=1; i<=suspect; i++)
93                             fout<<v[i].info<<' ' <<v[i].x<<' '<<v[i].y<<'\\n';
94                 }
95 }
```

22.5 movedel

Problema 5 - movedel

100 de puncte

Se consideră două siruri de caractere A și B , ambele siruri având același număr de caractere. Asupra sirurilor se aplică următorul algoritm:

- sirul A se permutează circular cu k_i poziții spre stânga
- din cele două siruri se elimină caracterele care coincid din punct de vedere al poziției și valorilor

Algoritmul se oprește când fie ambele siruri devin vide, fie sirurile nu mai au caractere comune. Valoarea k_i pentru fiecare pas i reprezintă al i -lea număr prim din mulțimea numerelor prime.

Cerințe

Dându-se N și M , să se genereze sirurile A și B , ambele având lungimea N , astfel încât numărul de repetări ale algoritmului aplicat celor două siruri să fie M .

Date de intrare

Fisierul de intrare **movedel.in** conține pe prima linie valorile N și M .

Date de ieșire

În fisierul de ieșire **movedel.out** se vor scrie sirurile de caractere A și B de lungime N , fiecare pe câte un rând.

Restricții și precizări

- sirurile trebuie să conțină doar litere mici ale alfabetului englez.
- în cazul în care algoritmul efectuează cel puțin M repetări pentru sirurile afișate, se va obține punctajul maxim pentru test. În caz contrar se vor obține $[X/M * 10]$ puncte pe test, unde X este numărul de repetări ale algoritmului (prin $[X/M]$ se înțelege partea întreagă a numărului X/M).
- Se garantează că există soluție pentru datele de test:

Testul	0	1	2	3	4	5	6	7	8	9
N	23	23	50	100	50	100	500	1 000	1 550	2 000
M	50	107	250	160	100	700	1 500	8 000	12 000	16 000

Exemple:

movedel.in	movedel.out	Explicații
3 5	abc cba	Prima aplicare a algoritmului: <i>cab</i> - după permutarea spre stânga cu 2 poziții (2 - primul număr prim), după eliminarea caracterelor comune, cele două siruri vor fi: <i>ab</i> <i>ba</i> A doua aplicare a algoritmului: <i>ba</i> - după permutarea spre stânga cu 3 poziții (3 - al doilea număr prim), după eliminarea caracterelor comune, cele două siruri devin vide, algoritmul încheindu-se. Astfel se obțin $[2/5*10]=4$ puncte pentru acest test
5 5	abcde edabc	Pentru sirurile găsite, algoritmul se încheie după 20 de etape. Astfel se obțin 10 puncte

Timp maxim de executare/test: **0.2** secunde pe Windows, **0.1** secunde pe Linux

Memorie: total **4 MB**

Dimensiune maximă a sursei: **10 KB**

22.5.1 Indicații de rezolvare

profesor Eugen Nodea, Colegiul Național "Tudor Vladimirescu" - Târgu Jiu
stud. Adrian Budău, Universitatea București

Soluție 1 - aproximativ 20 de puncte

Pentru un sir generat aleator, numărul de aplicări ale algoritmului este în medie egal cu lungimea sirului. Astă înseamnă că un sir aleatoriu este suficient de bun pentru a obține câteva puncte pe fiecare test.

Soluție 2 - aproximativ 50 de puncte

O observație ce poate îmbunătăți soluția anterioară este că creșterea numărului de aplicări ale algoritmului până la prima eliminare a unui caracter comun crește numărul total de aplicări ale algoritmului. O idee bazată pe această observație este folosirea unui singur caracter comun între cele două siruri.

Exemplu:

$N = 5$

babb

acc

Cele două soluții de mai sus și alte euristică puteau obține punctaje mai mari dacă erau repeteate pentru fiecare test și se alegea cea mai bună pereche de siruri. Această idee obține aproximativ 30 respectiv 65 de puncte.

Soluție 3 - 100 puncte

Plecând de la soluția doi se va căuta cea mai bună poziție pentru caracterul comun în primul sir. Astfel se obține numărul maxim de aplicări pentru siruri de acest fel.

Se pune caracterul comun în al doilea sir pe prima poziție. Apoi se simulează iterațiile urmărind la fiecare aplicare ce poziție din sirul de dinaintea rotațiilor se suprapune cu prima poziție din al doilea sir. Se marchează pozițiile în ordine până când toate ajung să fie marcate și se alege ultima dintre acestea.

Exemplu:

$N = 3$

După prima aplicare se marchează poziția 2.

După a doua aplicare se marchează poziția $(2+3) \bmod 3 = 2$.

După a treia aplicare se marchează poziția $(2+3+5) \bmod 3 = 1$.

După a patra aplicare se marchează poziția $(2+3+5+7) \bmod 3 = 2$.

...

După cea de a zecea aplicare se marchează poziția $(2+3+5+\dots+29) \bmod 3 = 0$.

Deci poziția optimă pentru caracterul comun este 0. O soluție care execută 10 aplicări este:

abb

acc

22.5.2 *Rezolvare detaliată

22.5.3 Cod sursă

Listing 22.5.1: movedel1.cpp

```

1 // Adrian Budau - 100 de puncte - O(M)
2 #include <iostream>
3 #include <vector>
4 #include <bits/stdc++.h>
5
6 using namespace std;
7
8 static const int kMaxV = 1000 * 1000 + 7;
9
10 int main()

```

```

11 {
12     freopen("movedel.in", "r", stdin);
13     freopen("movedel.out", "w", stdout);
14
15     vector<bool> sieve(kMaxV, false);
16     vector<int> primes;
17
18     for (int i = 2; i < kMaxV; ++i)
19         if (!sieve[i])
20         {
21             primes.push_back(i);
22             for (int j = i + i; j < kMaxV; j += i)
23                 sieve[j] = true;
24         }
25
26     reverse(primes.begin(), primes.end());
27
28     int N; cin >> N;
29
30     int covered = 0;
31     vector<bool> cover(N, 0);
32     int last = -1;
33     int lastlast = 0;
34     int now = 0;
35     int iterations = 0;
36
37     while (covered < N)
38     {
39         now = now + primes.back();
40         primes.pop_back();
41         now %= N;
42         if (!cover[now])
43         {
44             cover[now] = true;
45             ++covered;
46             lastlast = last;
47             last = now;
48         }
49         ++iterations;
50     }
51
52     string A(N, 'b'), B(N, 'c');
53     B[0] = 'a';
54     A[last] = 'a';
55
56     cerr << "I think i did " << iterations << " iterations\n";
57     cerr << "With last " << last << " and last last " << lastlast << "\n";
58     cout << A << "\n" << B << "\n";
59 }
```

Listing 22.5.2: movedel2.cpp

```

1 // Adrian Budau - intre 40 si 56 de puncte - O(N)
2 #include <iostream>
3 #include <algorithm>
4 #include <bits/stdc++.h>
5
6 using namespace std;
7
8 int main()
9 {
10     srand(time(NULL));
11
12     freopen("movedel.in", "r", stdin);
13     freopen("movedel.out", "w", stdout);
14
15     int N; cin >> N;
16     string A(N, 'b');
17     string B(N, 'c');
18     B[0] = 'a';
19     A[rand() % N] = 'a';
20
21     cout << A << "\n" << B << "\n";
22 }
```

Listing 22.5.3: movede3.cpp

```

1 // Adrian Budau - intre 18 si 26 de puncte - O(N)
2 #include <iostream>
3 #include <cstdlib>
4 #include <cstring>
5 #include <ctime>
6
7 using namespace std;
8
9 int main()
10 {
11     srand(time(NULL));
12
13     freopen("movedel.in", "r", stdin);
14     freopen("movedel.out", "w", stdout);
15
16     int N;
17     cin >> N;
18
19     for (int i = 0; i < N; ++i)
20         cout << char(rand() % 26 + 'a');
21
22     cout << "\n";
23
24     for (int i = 0; i < N; ++i)
25         cout << char(rand() % 26 + 'a');
26
27     cout << "\n";
28 }
```

22.6 undo

Problema 6 - undo

100 de puncte

XORin este nemulțumit de problemele primite în prima zi de concurs de la Olimpiada Națională de Informatică și decide astfel să se implice în comisie. În scurt timp devine specialistul comisiei în generarea de teste formate din siruri de numere. Din când în când el trebuie să adauge sau să șteargă elemente din sir. Câteodată el decide să readauge dintre elemente șterse anterior. Fie sirul de numere $a = (a_1, a_2, \dots, a_N)$ și N numărul de elemente din sir după fiecare operație.

Astfel el are de realizat următoarele operații pornind de la un sir vid:

- Inserează la sfârșitul sirului o valoare x ;
- șterge ultimele x elemente din sir;
- Readaugă la sfârșitul sirului primele x elemente șterse. Dacă, de exemplu, în operația anterioară de ștergere a unui număr y de elemente, am șters elementele $a_{N-y+1}, a_{N-y+2}, \dots, a_N$, iar acum urmează o operație de readăugare a x elemente, vor fi adăugate în ordine elementele $a_{N-y+1}, a_{N-y+2}, \dots, a_{N-y+x}$ la sfârșitul sirului.

Din când în când *XORin* își pune următoarea întrebare: de câte ori există valoarea x în sir?

Cerințe

Date de intrare

Pe prima linie a fișierului **undo.in** se va afla M ce reprezintă numărul de operații.

Pe următoarele M linii se vor afla operațiile codificate astfel:

- 1 x - se inserează elementul x la sfârșitul sirului
- 2 y - se șterge ultimele y elemente adăugate în sir
- 3 z - se adaugă înapoi la sfârșitul sirului primele z elemente șterse
- 4 t - se afișează numărul de elemente cu valoarea t din sir

Date de ieșire

Pe fiecare linie a fișierului **undo.out** se scriu răspunsurile la întrebările lui *XORin*, fiecare răspuns pe câte o linie.

Restricții și precizări

- Toate numerele din fișierul de intrare sunt cuprinse între 1 și 200 000;
- Pentru 20% din teste se garantează $M \leq 1\ 000$, pentru alte 40% din teste, se garantează că numerele inserate vor fi distințe;
- între o operație de ștergere și una de readăugare sau între două de readăugare nu se vor afla alte operații de inserare
- Numărul de elemente readăugate nu va fi mai mare decât numărul de elemente șterse la ultima operație.
- între două operații de readăugare va exista cel puțin o operație de ștergere.

Exemple:

undo.in	undo.out	Explicații
16	1	Inițial sirul este vid.
1 1	0	După primele 4 operații de inserare sirul este 1, 2, 3, 4.
1 2	0	Operația 4 4 va afișa 1.
1 3	1	Operația 2 2 va șterge ultimele două elemente sirul devinind 1,
1 4	3	2.
4 4	1	Din cauză că elementul 3 a fost șters, a șaptea operație va afișa 0.
2 2		
4 3		Operația 3 1 va readăuga la sfârșitul sirului elementul 3.
3 1		în urma acestei operații sirul devine 1, 2, 3.
4 4		Operația 4 4 va afișa 0, iar operația 4 3 va afișa 1.
4 3		§.a.m.d.
1 7		
1 7		
1 7		
4 7		
2 2		
4 7		

Timp maxim de executare/test: **1.0** secunde pe Windows, **0.3** secunde pe Linux

Memorie: total **16 MB**

Dimensiune maximă a sursei: **10 KB**

22.6.1 Indicații de rezolvare

stud. Petru Trîmbițăș, Universitatea Babes-Bolyai, Cluj-Napoca

Soluție 1 - $O(M^2)$ - 20 puncte

Simulăm soluția folosind o *stivă*.

Soluție 2 - $O(N)$ pentru cazul în care numerele sunt unice - 40 puncte

Pe lângă stivă, vom menține un sir poz_i - poziția din sir pe care se află numărul i . Pentru o inserare vom actualiza sirul poz și stiva. La ștergere vom scădea dimensiunea sirului fără a șterge efectiv numerele, iar la readăugare o vom crește. Pentru a afla dacă un număr x există în sir este suficient să verificăm dacă elementul de pe poziția poz_x din stivă este egal cu x și că poziția x este în stivă.

Solutie 3 - $O(N * \log N)$ - 100 puncte

Adrian Budău

În soluția anterioară în loc să reținem o singură poziție, vom reține toate pozițiile pe care apare elementul x în sir. Pentru fiecare operație de ștergere, vom executa operația propriu-zisă (ștergerea pozițiilor) la următoarea operație de inserare sau ștergere. În rest procedăm ca și la soluția anterioară. Pentru a afla numărul de apariții vom căuta binar în lista de poziții către care sunt în stivă.

22.6.2 *Rezolvare detaliată

22.6.3 Cod sursă

Listing 22.6.1: undol_1.cpp

```

1 // Mircea Popoveniuc - 100 de puncte - O(M log M)
2 #include<bits/stdc++.h>
3
4 using namespace std;
5
6 const int NMAX = 1e6;
7 const int VMAX = 1e6;
8
9 void insert(int);
10 void erase(int);
11 void undo(int);
12 int count(int);
13
14 int N, top;
15 vector<int> S;
16 vector<int> poz[VMAX + 5];
17
18 int main()
19 {
20     cin.sync_with_stdio(false);
21
22     #ifndef ONLINE_JUDGE
23         freopen("undo.in", "r", stdin);
24         freopen("undo.out", "w", stdout);
25     #endif
26
27     scanf("%d", &N);
28
29     for(int op, x; N--; )
30     {
31         scanf("%d%d", &op, &x);
32         if (op == 1) insert(x);
33         else if (op == 2) erase(x);
34         else if (op == 3) undo(x);
35         else if (op == 4) printf("%d\n", count(x));
36     }
37
38     return 0;
39 }
40
41 void insert(int x)
42 {
43     while (top < (int)S.size())
44     {
45         poz[S.back()].pop_back();
46         S.pop_back();
47     }
48
49     poz[x].push_back(++top);
50     S.push_back(x);
51 }
52
53 void erase(int x)
54 {
55     top -= x;
56 }
57
58 void undo(int x)
59 {
60     top += x;
61 }
62
63 int count(int x)
64 {
65     return (int)(upper_bound(poz[x].begin(), poz[x].end(), top)
66                  - poz[x].begin());
67 }
```

Listing 22.6.2: undol_2.cpp

```

1 // Petru Trimbitas - MlogM - 100 puncte
2 #include <iostream>
3 #include <fstream>
4 #include <vector>
5 #include <algorithm>
6
7 #define DN 200005
8 #define LL long long
9
10 using namespace std;
11
12 int q,n,nn,v[DN];
13 vector<int> poz[DN];
14
15 void ups()
16 {
17     for(int i=max(nn,n); i>min(nn,n); --i)
18         poz[v[i]].pop_back();
19 }
20
21 int main ()
22 {
23     ifstream f("undo.in");
24     ofstream g("undo.out");
25
26     int op,x,m;
27     f>>m;
28
29     while(f>>op>>x) {
30         if(op==1) {
31             ups();
32             v[++n]=x;
33             nn=n;
34             poz[x].push_back(n);
35         }else if(op==2) {
36             ups();
37             nn=n;
38             n-=x;
39         }else if(op==3) n+=x;
40         else if(op==4) {
41             int p=upper_bound(poz[x].begin(), poz[x].end(), n)-poz[x].begin();
42             g<<p<<'n';
43         }
44     }
45 }
```

Listing 22.6.3: undo2.cpp

```

1 // Adrian Budau - 100 de puncte - O(M log M)
2 #include <iostream>
3 #include <fstream>
4 #include <vector>
5 #include <algorithm>
6 #include <cassert>
7
8 using namespace std;
9
10 int main()
11 {
12     ifstream cin("undo.in");
13     ofstream cout("undo.out");
14
15     int N;
16     assert(cin >> N);
17     assert(1 <= N && N <= 1000 * 1000);
18
19     vector< vector<int> > list;
20     int lastClear = 0;
21     vector<int> stack;
22
23     for (int i = 0; i < N; ++i)
24     {
25         int type, x;
26         assert(cin >> type >> x);

```

```

27         assert(1 <= type && type <= 4);
28         assert(1 <= x && x <= 200 * 1000);
29
30         // do the last clear
31         if (type == 3)
32         {
33             assert(1 <= x && x <= lastClear);
34             lastClear -= x;
35         }
36
37         if (type <= 3)
38         {
39             while (lastClear > 0)
40             {
41                 --lastClear;
42                 list[stack.back()].pop_back();
43                 stack.pop_back();
44             }
45         }
46
47         if (type == 1)
48         {
49             if (x >= int(list.size()))
50                 list.resize(x + 1);
51             list[x].push_back(stack.size());
52             stack.push_back(x);
53             continue;
54         }
55
56         if (type == 2)
57         {
58             assert(1 <= x && x <= int(stack.size()));
59             lastClear = x;
60             continue;
61         }
62
63         if (type == 3)
64             continue;
65
66         if (x >= int(list.size()))
67             cout << 0 << "\n";
68         else
69             cout << lower_bound(
70                 list[x].begin(),
71                 list[x].end(),
72                 stack.size() - lastClear) - list[x].begin() << "\n";
73     }
74 }
```

Listing 22.6.4: undo3.cpp

```

1 // Petru Trimbitas - 40 de puncte - O(M) doar elemente unice
2 #include <iostream>
3 #include <algorithm>
4
5 #define DN 1000005
6
7 using namespace std;
8
9 int n,x,v[DN],poz[DN], op;
10
11 int main()
12 {
13     ifstream f("undo.in");
14     ofstream g("undo.out");
15     f>>n;
16     while(f>>op>>x) {
17         if(op==1) {
18             v[+n]=x;
19             poz[x]=n;
20         }
21         else if(op==2) n-=x;
22         else if(op==3) n+=x;
23         else if(op==4) {
24             g<<(poz[x]<=n && v[poz[x]]==x)<<'\n';
25         }
26     }
27 }
```

```
25     }
26   }
27 }
```

Listing 22.6.5: undo4.cpp

```
1 // Petru Trimbitas - 20 de puncte - O(M^2)
2 #include <iostream>
3 #include <fstream>
4 #define DN 1000005
5 using namespace std;
6
7 int n,v[DN];
8
9 int main()
10 {
11   ifstream f("undo.in");
12   ofstream g("undo.out");
13
14   int n,op,x;
15   f>>n;
16
17   while(f>>op>>x) {
18     if(op==1) v[++n]=x;
19     else if(op==2) n-=x;
20     else if(op==3) n+=x;
21     else if(op==4) {
22       int p=0;
23       for(int i=1; i<=n; ++i) p+=(v[i]==x);
24       g<<p<<'\'n';
25     }
26   }
27 }
```

Capitolul 23

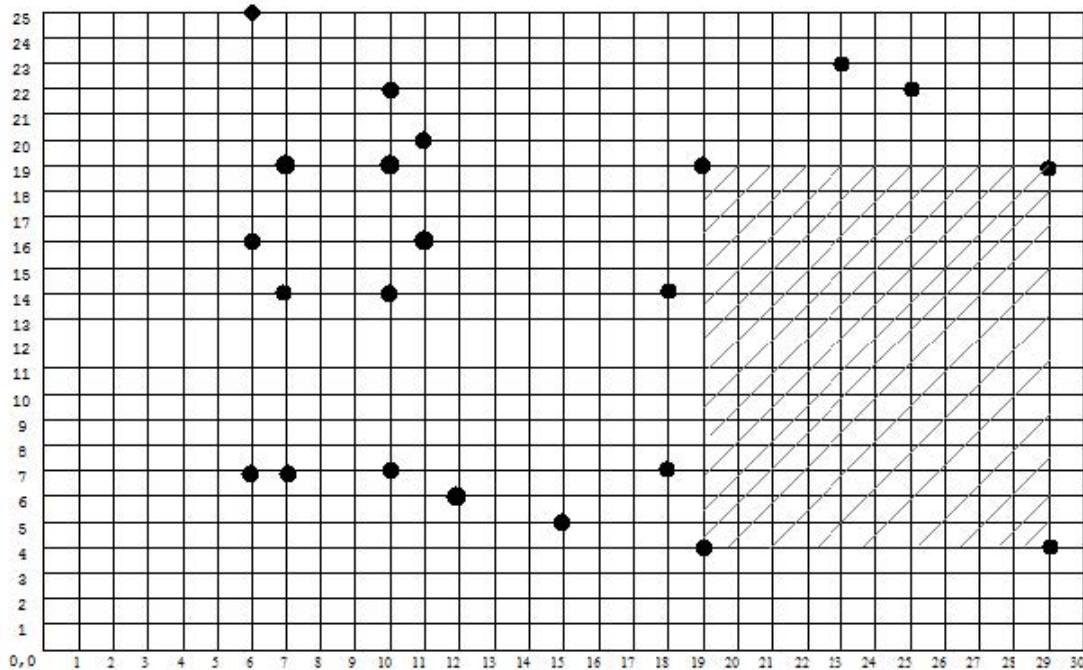
ONI 2015

23.1 cabana

Problema 1 - cabana

100 de puncte

Ben are un teren pe care se află o pădure cu arbori seculari. Acolo vrea să-și construiască o cabană, însă el fiind ecologist nu vrea să taie niciun arbore, ci vrea să găsească cea mai mare suprafață dreptunghiulară fără arbori. El caută o suprafață dreptunghiulară străjuită doar în colțuri de arbori și cu laturile paralele cu axele de coordonate. Ben cunoaște coordonatele tuturor arborilor din pădure și vă roagă să-l ajutați să găsească aria dreptunghiului cu suprafață maximă care are arbori doar în cele patru colțuri.



Cerințe

Cunoscând numărul arborilor din pădure și coordonatele acestora, se cere să se determine aria dreptunghiului de suprafață maximă cu copaci doar în cele 4 colțuri, unde Ben intenționează să-și construiască cabana.

Date de intrare

Fișierul de intrare **cabana.in** conține pe prima linie un număr natural n , reprezentând numărul de arbori din pădure.

Pe fiecare dintre următoarele n linii se află două numere întregi, separate printr-un spațiu, ce reprezintă abscisa și ordonata unui arbore.

Date de ieșire

În fișierul de ieșire **cabana.out** se scrie pe prima linie numărul natural a , reprezentând aria dreptunghiului de suprafață maximă.

Restricții și precizări

- Pentru 10% din teste: $1 \leq n \leq 10$, $-10^3 \leq x \leq 10^3$, $-10^3 \leq y \leq 10^3$
- Pentru 30% din teste: $1 \leq n \leq 500$, $-10^3 \leq x \leq 10^3$, $-1063 \leq y \leq 10^3$
- Pentru 50% din teste: $1 \leq n \leq 500$, $-10^6 \leq x \leq 10^6$, $-10^6 \leq y \leq 10^6$
- Pentru 70% din teste: $1 \leq n \leq 3000$, $-10^9 \leq x \leq 10^9$, $-10^9 \leq y \leq 10^9$
- Pentru 100% din teste: $1 \leq n \leq 50000$, $-10^9 \leq x \leq 10^9$, $-10^9 \leq y \leq 10^9$
- Nu există doi arbori așezați pe aceeași poziție.

Exemple:

cabana.in	cabana.out	Explicații
22		
6 25		
25 22		
15 5	19 19	
23 23	29 19	
6 7	29 4	
11 16	19 4	
11 20		deci aria maximă este 150
10 22		
6 16		
12 6		
7 19		
10 19		
10 14		
7 14		
7 7		
18 14		
18 7		
10 7		
19 19		
29 19		
29 4		
19 4		

Timp maxim de executare/test: **1.2** secunde pe Windows, **0.5** secunde pe

Memorie: total **16 MB** din care pentru stivă **6 MB**

Dimensiune maximă a sursei: **10 KB**

23.1.1 Indicații de rezolvare

prof. Liliana Șchipu - C.N. Frații Buzești, Craiova

Varianta 1

- student Mihai Nițu, Universitatea București

Solutie 50 de puncte: Se iterează prin fiecare pereche de 2 puncte și se consideră dacă această pereche poate constitui vârfurile diagonale opuse ale unui dreptunghi. Pentru a verifica acest lucru se poate parcurge din nou vectorul. Trebuie în această parcurgere să depistăm celelalte două vârfuri (care sunt unic determinate) și să ne asigurăm că nu există alte puncte în interior. Complexitate: $O(N^3)$.

Varianta 2

- prof. Liliana Șchipu, C.N. Frații Buzești, Craiova

Solutie 70 de puncte: Se rețin coordonatele arborilor într-un vector de structuri. Se sortează coordonatele arborilor de două ori:

- o dată după ordonate descrescător și pentru ordonate egale se sortează crescător după abscise și se rețin rezultatele într-un vector de structuri;
- și încă o dată după abscise crescător și pentru abscise egale se sortează descrescător după ordonate și se rețin rezultatele într-un alt vector de struct-uri.

Se caută pentru coordonatele consecutive a doi căte doi arboridin primul vector, care au aceeași ordonată, dacă există în cel de-al doilea vector doi arbori cu care pot forma un dreptunghi. Se va folosi căutarea binară. Odată găsite 4 puncte care pot forma un dreptunghi, se verifică să nu existe alți arbori în interiorul sau pe conturul acestuia parcurgând din nou vectorul de puncte, excepție fac colțurile dreptunghiului unde trebuie să existe arbori. Complexitate: $O(N^2)$

Varianta 3 - student Mihai Nițu, student Alexandru Murtaza, Universitatea București

Solutie 70 de puncte: Se grupează punctele pe fâșii orizontale și verticale (o fâșie orizontală reprezintă un șir de puncte care au aceeași ordonată; o fâșie verticală este un șir de puncte care au aceeași abscisă).

O fâșie orizontală o ținem sortată după x , o fâșie verticală, sortată după y . Cu ajutorul acestor fâșii putem să identificăm în $O(1)$, pentru un punct P , punctul imediat de la stânga și punctul imediat de jos.

Acestea sunt singurele puncte candidate pentru a fi colțurile dreapta-sus și stânga-jos pentru dreptunghiul care are punctul P ca vîrf stânga-sus.

Analog, putem afla singurul candidat pentru punctul dreapta-jos. Dacă aceste puncte formează un dreptunghi valid, mai rămâne să verificăm dacă mai există puncte strict în interior iar pentru a verifica asta, parcurgem lista de puncte. Complexitate: $O(N^2)$

Varianta 4 - student Mihai Nițu, student Alexandru Murtaza, Universitatea București

Solutie 100 de puncte: Vom proceda asemănător cu soluția precedentă, formând mai întâi fâșile verticale și orizontale sortate. Pentru a îmbunătăți complexitatea soluției precedente, ne vom lega de ordinea în care evaluăm fiecare punct ca posibil colț stânga-sus al unui dreptunghi.

Puteam, de exemplu, să inițializăm o serie de indici (câte unul pentru fiecare fâșie verticală) care să indice fiecare către primul punct (cel mai de sus) din fiecare fâșie verticală.

Alegem să evaluăm mai întâi cel mai de sus din aceste puncte. După ce l-am evaluat, vom incrementa indicele de pe această fâșie pentru ca acesta să arate spre următorul punct de pe fâșie. Repetăm procedeul până când am evaluat toate punctele. Dar cum facem acum evaluarea?

Procedăm ca la soluția anterioară pentru a afla care sunt cele 3 puncte cu care punctul pe care îl evaluăm poate forma un dreptunghi. Fie x fâșia verticală careia îi aparține punctul stânga-sus. Fie y fâșia verticală careia îi aparține punctul dreapta-sus.

Dacă va exista un punct în interiorul dreptunghiului format din aceste 4 puncte, atunci el sigur va fi "cel mai de sus" punct de pe fâșile dintre fâșia x și fâșia y .

Dacă acesta nu este în interiorul dreptunghiului, atunci nici un punct nu este în interiorul dreptunghiului și putem actualiza soluția cu aria acestuia.

Puteam folosi o descompunere în bucăți de \sqrt{N} pentru a implementa mai eficient modificări și interogări de maxim pe intervalul de fâșii verticale. Complexitatea devine astfel $O(N\sqrt{N})$.

23.1.2 *Rezolvare detaliată

23.1.3 Cod sursă

Listing 23.1.1: 1_cabana.cpp

```

1 // student Mihai Nitu - Universitatea Bucuresti
2
3 #include <fstream>
4 #include <iostream>
5 #include <cmath>
6 #include <vector>
7 #include <algorithm>
8
9 #define maxn 150010
10 #define inf 1000000001

```

```

11
12 using namespace std;
13
14 ifstream fin ("cabana.in");
15 ofstream fout("cabana.out");
16
17 struct point
18 {
19     int x,y,i;
20 } v[maxn], v1[maxn], v2[maxn];
21
22 bool cmp1 (point a, point b)
23 {
24     if (a.x == b.x)
25         return a.y > b.y;
26     return a.x < b.x;
27 }
28
29 bool cmp2 (point a, point b)
30 {
31     if (a.y == b.y)
32         return a.x < b.x;
33     return a.y > b.y;
34 }
35
36 int bit[maxn], whichL[maxn], whichC[maxn],
37 posL[maxn], posC[maxn], pointer[maxn];
38 int sq, p, nrl, nr2, t, n;
39 vector<int> C[maxn], L[maxn];
40
41 void create_structure()
42 {
43     sq = sqrt(nrl);
44
45     v[0].y = -inf;
46     int maxpoint = 0;
47     t = 0;
48
49     for (int i = 1; i <= nrl; ++i)
50     {
51         if (v[C[i][0]].y > v[maxpoint].y)
52             maxpoint = C[i][0];
53
54         if (i % sq == 0)
55         {
56             ++t;
57             bit[t] = maxpoint;
58             maxpoint = 0;
59         }
60     }
61 }
62
63 void update (int pos)
64 {
65     int whbit = (pos-1)/sq+1;
66
67     if (whbit > t)
68         return;
69
70     int maxpoint = 0;
71
72     for (int i = (whbit-1)*sq + 1; i <= whbit*sq; ++i)
73         if (v[C[i][pointer[i]]].y > v[maxpoint].y)
74             maxpoint = C[i][pointer[i]];
75
76     bit[whbit] = maxpoint;
77 }
78
79 int query (int left, int right)
80 {
81     int maxpoint = 0;
82
83     if (right - left + 1 <= sq)
84     {
85         for (int i = left; i <= right; ++i)
86             if (v[C[i][pointer[i]]].y > v[maxpoint].y)

```

```

87             maxpoint = C[i][pointer[i]];
88
89         p = whichC[maxpoint];
90         return v[maxpoint].y;
91     }
92
93     int whleft = (left-1)/sq+1;
94     int whright = (right-1)/sq+1;
95
96     for (int i = whleft+1; i <= whright-1; ++i)
97         if (v[bit[i]].y > v[maxpoint].y)
98             maxpoint = bit[i];
99
100    for (int i = left; i <= whleft*sq; ++i)
101        if (v[C[i][pointer[i]]].y > v[maxpoint].y)
102            maxpoint = C[i][pointer[i]];
103
104    for (int i = (whright-1)*sq +1; i <= right; ++i)
105        if (v[C[i][pointer[i]]].y > v[maxpoint].y)
106            maxpoint = C[i][pointer[i]];
107
108    p = whichC[maxpoint];
109    return v[maxpoint].y;
110 }
111
112 int main()
113 {
114     fin >> n;
115
116     for (int i = 1; i <= n; ++i)
117     {
118         fin >> v[i].x >> v[i].y;
119         v1[i].x = v[i].x;
120         v1[i].y = v[i].y;
121         v2[i].x = v[i].x;
122         v2[i].y = v[i].y;
123         v1[i].i = i;
124         v2[i].i = i;
125     }
126
127     sort(v1+1, v1+n+1, cmp1);
128     sort(v2+1, v2+n+1, cmp2);
129
130     v1[0].x = -inf;
131
132     for (int i=1 ; i <= n; ++i)
133     {
134         if (v1[i].x != v1[i-1].x)
135             ++nrl;
136
137         C[nrl].push_back(v1[i].i);
138         whichC[v1[i].i] = nrl;
139         posC[v1[i].i] = C[nrl].size()-1;
140     }
141
142     v2[0].y = inf;
143
144     for (int i =1 ; i <= n; ++i)
145     {
146         if (v2[i].y != v2[i-1].y)
147             ++nr2;
148
149         L[nr2].push_back(v2[i].i);
150         whichL[v2[i].i] = nr2;
151         posL[v2[i].i] = L[nr2].size()-1;
152     }
153
154     for (int i = 1; i <= nrl; ++i)
155     {
156         pointer[i] = 0;
157         C[i].push_back(0);
158     }
159
160     create_structure();
161     long long answer = 0;
162

```

```

163     int cnt = 0;
164
165     while (query(1, nr1) != -inf)
166     {
167         ++cnt;
168         point LU = v[C[p][pointer[p]]];
169         int whcol = p;
170         int poscol = pointer[p];
171         int whline = whichL[C[p][pointer[p]]];
172         int posline = posL[C[p][pointer[p]]];
173
174         if (posline + 1 < L[whline].size() && poscol + 1 < C[whcol].size()-1)
175         {
176             int rightp = L[whline][posline+1];
177             int downp = C[whcol][poscol+1];
178
179             point RU = v[rightp];
180             point LD = v[downp];
181
182             int newcol = whichC[rightp];
183             int posnewcol = posC[rightp];
184             int newline = whichL[downp];
185             int posnewline = posL[downp];
186
187             if (posnewcol + 1 < C[newcol].size()-1 &&
188                 posnewline + 1 < L[newline].size() &&
189                 C[newcol][posnewcol+1] == L[newline][posnewline+1])
190             {
191                 point RD = v[C[newcol][posnewcol+1]];
192
193                 int rez = query(whcol+1, newcol-1);
194
195                 if (rez < RD.y)
196                     answer = max(answer, 1LL*(RD.x - LD.x)*(RU.y - RD.y));
197             }
198         }
199         ++pointer[whcol];
200         update(whcol);
201     }
202
203     fout << answer;
204 }

```

Listing 23.1.2: 2_cabana.cpp

```

1 //student Mihai Nitu - Universitatea Bucuresti
2
3 #include <fstream>
4 #include <iostream>
5 #include <cmath>
6 #include <vector>
7 #include <algorithm>
8
9 #define maxn 150010
10 #define inf 1000000001
11
12 using namespace std;
13
14 ifstream fin ("cabana.in");
15 ofstream fout("cabana.out");
16
17 struct point
18 {
19     int x,y,i;
20 } v[maxn], v1[maxn], v2[maxn];
21
22 bool cmp1 (point a, point b)
23 {
24     if (a.x == b.x)
25         return a.y > b.y;
26     return a.x < b.x;
27 }
28
29 bool cmp2 (point a, point b)

```

```

30  {
31      if (a.y == b.y)
32          return a.x < b.x;
33      return a.y > b.y;
34  }
35
36 int bit[maxn], whichL[maxn], whichC[maxn],
37 posL[maxn], posC[maxn], pointer[maxn];
38 int sq, p, nr1, nr2, t, n;
39 vector<int> C[maxn], L[maxn];
40
41 int main()
42 {
43     fin >> n;
44
45     for (int i = 1; i <= n; ++i)
46     {
47         fin >> v[i].x >> v[i].y;
48         v1[i].x = v[i].x;
49         v1[i].y = v[i].y;
50         v2[i].x = v[i].x;
51         v2[i].y = v[i].y;
52         v1[i].i = i;
53         v2[i].i = i;
54     }
55
56     sort(v1+1, v1+n+1, cmp1);
57     sort(v2+1, v2+n+1, cmp2);
58
59     v1[0].x = -inf;
60
61     for (int i=1 ; i <= n; ++i)
62     {
63         if (v1[i].x != v1[i-1].x)
64             ++nr1;
65
66         C[nr1].push_back(v1[i].i);
67         whichC[v1[i].i] = nr1;
68         posC[v1[i].i] = C[nr1].size()-1;
69     }
70
71     v2[0].y = inf;
72
73     for (int i =1 ; i <= n; ++i)
74     {
75         if (v2[i].y != v2[i-1].y)
76             ++nr2;
77
78         L[nr2].push_back(v2[i].i);
79         whichL[v2[i].i] = nr2;
80         posL[v2[i].i] = L[nr2].size()-1;
81     }
82
83     for (int i = 1; i <= nr1; ++i)
84     {
85         pointer[i] = 0;
86         C[i].push_back(0);
87     }
88
89     long long answer = 0;
90
91     int cnt = 0;
92
93     for (int i = 1; i <= n; ++i)
94     {
95         point LU = v[i];
96         int whcol = whichC[i];
97         int poscol = posC[i];
98         int whline = whichL[i];
99         int posline = posL[i];
100
101        if (posline + 1 < L[whline].size() && poscol + 1 < C[whcol].size()-1)
102        {
103            int rightp = L[whline][posline+1];
104            int downp = C[whcol][poscol+1];
105

```

```

106         point RU = v[rightp];
107         point LD = v[downp];
108
109         int newcol = whichC[rightp];
110         int posnewcol = posC[rightp];
111         int newline = whichL[downp];
112         int posnewline = posL[downp];
113
114         if (posnewcol + 1 < C[newcol].size() - 1 &&
115             posnewline + 1 < L[newline].size() &&
116             C[newcol][posnewcol+1] == L[newline][posnewline+1])
117         {
118             point RD = v[C[newcol][posnewcol+1]];
119
120             bool ok = 1;
121
122             for (int j = 1; j <= n; ++j)
123             {
124                 if (LD.x < v[j].x && v[j].x < RD.x && RD.y < v[j].y && v[j].y < RU.y)
125                     ok = 0;
126             }
127
128             if (ok)
129                 answer = max(answer, 1LL * (RD.x - LD.x) * (RU.y - RD.y));
130         }
131     }
132 }
133
134 fout << answer;
135 }
```

Listing 23.1.3: 3_cabana.cpp

```

1 // student Mihai Nitu - Universitatea Bucuresti
2
3 #include <iostream>
4 #include <vector>
5 #include <algorithm>
6
7 #define maxn 150010
8 #define inf 1000000001
9
10 using namespace std;
11
12 ifstream fin("cabana.in");
13 ofstream fout("cabana.out");
14
15 struct point
16 {
17     int x, y, i;
18 } v[maxn], v1[maxn], v2[maxn];
19
20 bool cmp1(point a, point b)
21 {
22     if (a.x == b.x)
23         return a.y > b.y;
24     return a.x < b.x;
25 }
26
27 bool cmp2(point a, point b)
28 {
29     if (a.y == b.y)
30         return a.x < b.x;
31     return a.y > b.y;
32 }
33
34
35 int bit[maxn], whichL[maxn], whichC[maxn],
36 posL[maxn], posC[maxn], pointer[maxn];
37 int sq, p, nr1, nr2, t, n;
38 vector<int> C[maxn], L[maxn];
39
40 int main()
41 {
```

```

43     fin >> n;
44
45     for (int i = 1; i <= n; ++i)
46     {
47         fin >> v[i].x >> v[i].y;
48         v1[i].x = v[i].x;
49         v1[i].y = v[i].y;
50         v2[i].x = v[i].x;
51         v2[i].y = v[i].y;
52         v1[i].i = i;
53         v2[i].i = i;
54     }
55
56     sort(v1 + 1, v1 + n + 1, cmp1);
57     sort(v2 + 1, v2 + n + 1, cmp2);
58
59     v1[0].x = -inf;
60
61     for (int i = 1; i <= n; ++i)
62     {
63         if (v1[i].x != v1[i - 1].x)
64             ++nr1;
65
66         C[nr1].push_back(v1[i].i);
67         whichC[v1[i].i] = nr1;
68         posC[v1[i].i] = C[nr1].size() - 1;
69     }
70
71     v2[0].y = inf;
72
73     for (int i = 1; i <= n; ++i)
74     {
75         if (v2[i].y != v2[i - 1].y)
76             ++nr2;
77
78         L[nr2].push_back(v2[i].i);
79         whichL[v2[i].i] = nr2;
80         posL[v2[i].i] = L[nr2].size() - 1;
81     }
82
83     for (int i = 1; i <= nr1; ++i)
84     {
85         pointer[i] = 0;
86         C[i].push_back(0);
87     }
88
89     long long answer = 0;
90
91     int cnt = 0;
92
93     for (int i = 1; i <= n; ++i)
94     {
95         point LU = v[i];
96         int whcol = whichC[i];
97         int poscol = posC[i];
98         int whline = whichL[i];
99         int posline = posL[i];
100
101         if (posline+1 < L[whline].size() && poscol+1 < C[whcol].size() - 1)
102         {
103             int rightp = L[whline][posline + 1];
104             int downp = C[whcol][poscol + 1];
105
106             point RU = v[rightp];
107             point LD = v[downp];
108
109             int newcol = whichC[rightp];
110             int posnewcol = posC[rightp];
111             int newline = whichL[downp];
112             int posnewline = posL[downp];
113
114             if (posnewcol + 1 < C[newcol].size() - 1 &&
115                 posnewline + 1 < L[newline].size() &&
116                 C[newcol][posnewcol + 1] == L[newline][posnewline + 1])
117             {
118                 point RD = v[C[newcol][posnewcol + 1]];

```

```

119         if (answer < 1LL * (RD.x - LD.x)*(RU.y - RD.y))
120     {
121         bool ok = 1;
122
123         for (int j = 1; j <= n; ++j)
124         {
125             if (LD.x < v[j].x && v[j].x < RD.x &&
126                 RD.y < v[j].y && v[j].y < RU.y)
127             {
128                 ok = 0;
129                 break;
130             }
131         }
132
133         if (ok)
134             answer = max(answer, 1LL * (RD.x-LD.x)*(RU.y-RD.y));
135     }
136 }
137 }
138 }
139 }
140
141 fout << answer;
142 }
```

Listing 23.1.4: 4_cabana.cpp

```

1 // student Mihai Nitu - Universitatea Bucuresti
2
3 #include <iostream>
4
5 #define maxn 150010
6
7 using namespace std;
8
9 ifstream fin("cabana.in");
10 ofstream fout("cabana.out");
11
12 struct point
13 {
14     int x,y;
15 } v[maxn];
16
17 int n;
18
19 int main()
20 {
21     fin >> n;
22
23     for (int i = 1; i <= n; ++i)
24         fin >> v[i].x >> v[i].y;
25
26     long long answer = 0;
27
28     for (int i = 1; i <= n; ++i)
29     {
30         for (int j = 1; j <= n; ++j)
31         {
32             if (i == j || v[i].x > v[j].x || v[i].y > v[j].y)
33                 continue;
34
35             bool ok1 = 0, ok2 = 0, ok3 = 1;
36
37             if(answer >= 1LL * (v[j].y - v[i].y)*(v[j].x - v[i].x))
38                 continue;
39
40             for (int k = 1; k <= n; ++k)
41             {
42                 if (i == k || j == k)
43                     continue;
44
45                 if (v[k].x == v[i].x && v[k].y == v[j].y)
46                     ok1 = 1;
47
48                 else if (v[k].x == v[j].x && v[k].y == v[i].y)
```

```

49             ok2 = 1;
50
51         else if (v[i].x <= v[k].x && v[k].x <= v[j].x &&
52                 v[i].y <= v[k].y && v[k].y <= v[j].y)
53         {
54             ok3 = 0;
55             break;
56         }
57     }
58
59     if (ok1 && ok2 && ok3)
60         answer = max (answer, 1LL*(v[j].y-v[i].y)*(v[j].x-v[i].x));
61     }
62 }
63
64 fout << answer;
65 }
```

Listing 23.1.5: 5_cabana.cpp

```

1 //prof. Liliana Schiopu, CNFB-Craiova
2 #include <algorithm>
3 #include <fstream>
4
5 using namespace std;
6
7 ifstream f("cabana.in");
8 ofstream g("cabana.out");
9
10 struct punct
11 {
12     long long x,y;
13 };
14
15 punct v1[30001],v2[30001];
16 long long n,i,j;
17 long long dx,dy,dxM,dyM,poz1,poz2;
18 int ok;
19 long long aria,ariaM;
20
21 int cmp(punct A,punct B)
22 {
23     if(A.x==B.x)
24         return A.y>B.y;
25     return A.x<B.x;
26 }
27
28 int cmp1(punct A,punct B)
29 {
30     if(A.y==B.y)
31         return A.x<B.x;
32     return A.y>B.y;
33 }
34
35 int caut(long long st,long long dr,long long x1,long long y1)
36 {
37     long long mij;
38     mij=(st+dr)/2;
39     while(st<=dr)
40     {
41         if(v2[mij].x==x1&&v2[mij].y==y1)
42             return mij;
43         else
44             if((x1<=v2[mij].x&&v2[mij+1].x!=x1) ||
45                 (x1<=v2[mij].x&&v2[mij+1].x==x1&&y1>=v2[mij].y))
46                 dr=mij-1;
47             else
48                 st=mij+1;
49
50         mij=(st+dr)/2;
51     }
52
53     return 0;
54 }
```

```

56 int main()
57 {
58     f>>n;
59     for(i=1; i<=n; i++)
60     {
61         f>>v1[i].x>>v1[i].y;
62         v2[i].x=v1[i].x;
63         v2[i].y=v1[i].y;
64     }
65
66     sort(v1+1, v1+n+1, cmp1);
67     sort(v2+1, v2+n+1, cmp);
68
69     for(i=1; i<n; i++)
70     {
71         if(v1[i].y==v1[i+1].y)
72         {
73             poz1=caut(1,n,v1[i].x,v1[i].y);
74             poz2=caut(1,n,v1[i+1].x,v1[i+1].y);
75             if(poz1!=0&&poz2!=0&&poz1<n&&poz2<n)
76                 if(v2[poz1+1].x==v2[poz1].x&&
77                     v2[poz2+1].x==v2[poz2].x&&
78                     v2[poz1+1].y==v2[poz2+1].y)
79                 {
80                     aria=(v1[i+1].x-v1[i].x)*(v2[poz1].y-v2[poz1+1].y);
81                     ok=0;
82                     for(j=poz1+2; j<=poz2-1&&!ok; j++)
83                         if(v2[j].x>=v2[poz1+1].x&&v2[j].x<=v2[poz2].x&&
84                             v2[j].y>=v2[poz2+1].y&&v2[j].y<=v2[poz2].y)
85                             ok=1;
86                     if(aria>ariaM&&ok==0)
87                         ariaM=aria;
88                 }
89             }
90         }
91     g<<ariaM;
92     return 0;
93 }

```

23.2 fence

Problema 2 - fence

100 de puncte

Un proprietar vinde un teren de formă dreptunghiulară împărțit în $M \times N$ parcele de formă patrată cu lungimea laturii de o unitate. Fiecare parcelă costă V lei. Vlad s-a interesat și aflat pentru fiecare din parcelele terenului care este valoarea de revânzare. El constată că unele parcele i-ar putea aduce profit, iar altele i-ar aduce pierdere. Fiind îșteț, negociază cu proprietarul să cumpere atâtea parcele de teren câte pot fi împrejmuite cu un singur gard de lungime egală cu $2M + 2N$ unități. Terenul are pe fiecare din cele patru laturi acces la drumul exterior, pe o porțiune de lungime egală cu o unitate. Vlad negociază astfel încât terenul achiziționat să conțină și cele patru parcele de acces la exterior.

Cerințe

Cunoscând M și N - dimensiunile terenului, V - prețul de cumpărare al fiecărei parcele, x_nord , x_sud , y_vest și y_est - pozițiile parcelelor cu acces la drumul exterior și $A[i][j]$, $1 \leq i \leq M$ și $1 \leq j \leq N$ - valorile de revânzare pentru fiecare parcelă, să se determine:

- Profitul $P_arie_minimă$ pe care-l poate obține Vlad după cumpărarea și apoi revânzarea suprafetei de teren de arie minimă, împrejmuită conform condițiilor negociate.
- Profitul maxim P_max pe care-l poate obține Vlad după cumpărarea și apoi revânzarea unei suprafete de teren împrejmuită conform condițiilor negociate.

Date de intrare

Fișierul **fence.in** conține pe prima linie numărul t .

Pentru toate testelete de intrare numărul t poate avea doar valoarea 1 sau valoarea 2.

Pe linia a doua se găsesc numerele M , N , V , x_nord , x_sud , y_vest și y_est separate prin câte un spatiu, iar pe următoarele M linii se află câte N numere naturale separate prin câte un spatiu, reprezentând valorile de revânzare ale celor $M \times N$ parcele de teren.

Date de ieșire

Dacă valoarea lui t este 1, atunci se va rezolva numai punctul a) din cerință.

În acest caz în fișierul de ieșire **fence.out** se va scrie pe prima linie numărul $P_arie_minimă$.

Dacă valoarea lui t este 2, atunci se va rezolva numai punctul b) din cerință.

În acest caz în fișierul de ieșire **fence.out** se va scrie pe prima linie numărul P_max .

Restricții și precizări

- $3 \leq M \leq 1\,000$
- $3 \leq N \leq 1\,000$
- $1\,000 \leq V \leq 10\,000$
- $2 \leq x_nord \leq N - 1, 2 \leq x_sud \leq N - 1, 2 \leq y_vest \leq M - 1, 2 \leq y_est \leq M - 1$
- $(x_nord - x_sud) \cdot (y_est - y_vest) \geq 0$
- $1 \leq A[i][j] \leq 20\,000$
- Prin profit se înțelege suma valorilor de revânzare corespunzătoare parcelelor din suprafața împrejmuită din care se scade produsul dintre prețul de cumpărare V și numărul parcelelor împrejmuite, care poate fi și negativ.
- Pentru rezolvarea corectă a primei cerințe se va obține 20% din punctaj.

Exemple:

fence.in	fence.out	Explicații
<pre> 1 5 7 6 3 5 3 2 3 5 8 4 9 8 7 9 3 7 6 4 5 9 6 6 8 2 5 4 8 3 3 4 7 7 2 1 8 7 9 2 8 4 2 </pre>	3	<p>$M=5, N=7, V=6, x_nord=3, x_sud=5, y_vest=3, y_est=2$</p> <p>$P_arie_minimă = (8+7+6+4+5+9+6+6+8+2+5+7+8)-6 \cdot 13 = 81-78 = 3$</p>
<pre> 2 5 7 6 3 5 3 2 3 5 8 4 9 8 7 9 3 7 6 4 5 9 6 6 8 2 5 4 8 3 3 4 7 7 2 1 8 7 9 2 8 4 2 </pre>	8	<p>$M=5, N=7, V=6, x_nord=3, x_sud=5, y_vest=3, y_est=2$</p> <p>$P_max = (8+4+9+8+7+7+6+4+5+9+6+6+8+2+5+7+7+8)-6 \cdot 18 = 116 - 108 = 8$</p>

Timp maxim de executare/test: **1.2** secunde pe Windows, **0.3** secunde pe Linux

Memorie: total **32 MB** din care pentru stivă **8 MB**

Dimensiune maximă a sursei: **10 KB**

23.2.1 Indicații de rezolvare

prof. Ionel-Vasile Pit-Rada, Colegiul Național "TRAIAN", Drobeta Turnu Severin

Varianta 1

Inițial, din fiecare valoare $a[i][j]$, se scade costul V

a) P_arie_minima se calculează ca suma a elementelor din matrice care nu aparțin zonelor cu colțurile diagonale:

Zona NV : (1 , 1) - (y_vest - 1, x_nord - 1)

Zona NE: (1, x_nord + 1) - (y_est - 1, N)

Zona SV: (y_vest + 1, 1) - (M, x_sud-1)

Zona SE: (y_est + 1, x_sud + 1) - (M,N)

Complexitate $O(M * N)$

b) Se calculează, pentru fiecare din cele patru zone, câte un profit maximal care, dacă va fi pozitiv, atunci se va adăuga la P_arie_minima, calculat ca la punctul a).

Prin transformări, precum translații, simetrii și rotații, studiul celor patru tipuri de zone se poate reduce la studiul unui singur tip.

Să analizăm de exemplu zona NE pentru care folosim notațiile

$mx = y_{est} - 1$, $nx = N - x_{nord}$

și presupunem că datele se află în matricea x cu mx linii și nx coloane.

Dorim să găsim un "gard" descrescător care pornește din dreptul pozitiei (1,1) și se oprește în dreptul poziției (mx, nx), pentru care suma elementelor cuprinse între "gard", linia mx și limitate în stânga și respectiv dreapta de coloanele 1 și respectiv nx , este maximă. Este posibil ca suma maximă să fie negativă sau zero, adică zona studiată să nu aducă profit.

Proprietatea "gardului" de a fi descrescător va asigura conservarea perimetrului total egal cu $2M+2N$.

Perimetru oricărui gard descrescător care unește (1,1) cu (mx, nx) va avea lungimea egală cu $mx+nx$ unități.

Profitul maxim specific zonei se poate calcula după cum urmează:

în linia $mx+1$ inițializăm cu 0.

Pentru fiecare $1 <= j <= nx$

Dacă $j == 1$, atunci

Pentru fiecare $1 <= i <= mx+1$, începând cu $i=mx+1$

//sume parțiale pentru prima coloană

calculăm $y[i] = x[i][1] + x[i+1][1] + \dots + x[mx][1] + x[mx+1][1]$

și

//profiturile corespunzătoare coloanei 1

$z[i][1] = y[i];$

Dacă $2 <= j <= nx$, atunci

Pentru fiecare $1 <= i <= mx+1$, începând cu $i=mx+1$

//sume parțiale pentru coloana j

calculăm $y[i] = x[i][1] + x[i+1][1] + \dots + x[mx][1] + x[mx+1][1]$

//se calculează profiturile, combinând sumele parțiale din coloana j

//cu profiturile corespunzătoare coloanelor 1,2, ..., j-1, care se găsesc în

//coloana j-1 a matricei z

//ideea este că pentru fiecare sumă parțială $y[i]$ să găsim cea mai bună

//completare din stânga, care să conserve descreșterea traseului

//traseul propriu-zis nu va interesa ci doar suma maximă posibilă

Pentru fiecare $1 <= i <= mx+1$, începând cu $i=1$

calculăm $z[i][j] = y[i] + \max(z[1][j-1], z[2][j-1], \dots, z[i][j-1])$

La final profitul maximal corespunzător zonei studiate este egal cu

$\max(z[1][nx], z[2][nx], \dots, z[mx+1][nx])$

Dacă profitul maximal găsit pentru zona studiată va fi negativ, atunci acesta se va neglija în suma finală.

Complexitate: $O(M * N)$

Varianța 2

O soluție cu *backtracking* ar putea lua maxim 30 puncte.

23.2.2 *Rezolvare detaliată

23.2.3 Cod sursă

Listing 23.2.1: 1_fence.cpp

```

1 // prof. Vasile Ionel Pit-Rada
2
3 #include<stdio.h>
4
5 int m,n,v,xn,xs,yv,ye;
6 long long pmin,pmax;
7 short int a[1002][1002], x[1002][1002];
8 short int mx,nx;
9
10 int solve()
11 {
12     short int i,j;
13     long long b[1002],c[1002],vmax,s;
14     for(j=1;j<=nx;j++)
15         c[j]=0;
16
17     vmax=(long long)-10000000000;
18     for(i=mx;i>=1;i--)
19     {
20         s=0;
21         for(j=1;j<=nx;j++)
22         {
23             s=s+x[i][j];
24             b[j]=s+c[j];
25         }
26         c[nx]=b[nx];
27         for(j=nx-1;j>=1;j--)
28         {
29             c[j]=c[j+1];
30             if(b[j]>c[j])
31                 c[j]=b[j];
32         }
33         if(c[1]>vmax)vmax=c[1];
34     }
35     return vmax;
36 }
37
38 int main()
39 {
40     long long tot,s,s1,snv,sne,ssv,sse;
41     int z,i,j,p;
42     freopen("fence.in","r",stdin);
43     freopen("fence.out","w",stdout);
44     scanf("%d %d %d %d %d %d %d %d\n", &p, &m, &n, &v, &xn, &xs, &yv, &ye);
45
46     tot=0;
47     for(i=1;i<=m;i++)
48     {
49         for(j=1;j<=n;j++)
50         {
51             scanf("%d ",&z);
52             a[i][j]=z-v;
53             tot=tot+a[i][j];
54         }
55     }
56     pmin=tot;
57     s=0;
58     s1=0;
59
60     mx=yv-1;
61     nx=xn-1;
62     for(i=1;i<=mx;i++)
63     {
64         for(j=1;j<=nx;j++)
65         {
66             x[i][j]=a[i][nx+1-j];
67             s1=s1+x[i][j];
68         }
69     }
70     pmin=pmin-s1;

```

```

71     snv=solve();
72     if(snv>0)s=s+snv;
73
74     s1=0;
75     mx=ye-1;
76     nx=n-xn;
77     for(i=1;i<=mx;i++)
78     {
79         for(j=1;j<=nx;j++)
80         {
81             x[i][j]=a[i][j+xn];
82             s1=s1+x[i][j];
83         }
84     }
85     pmin=pmin-s1;
86     sne=solve();
87     if(sne>0)s=s+sne;
88
89     s1=0;
90     mx=m-yv;
91     nx=xs-1;
92     for(i=1;i<=mx;i++)
93     {
94         for(j=1;j<=nx;j++)
95         {
96             z=a[i+yv][nx+1-j];
97             x[mx+1-i][j]=z;
98             s1=s1+z;
99         }
100    }
101
102    pmin=pmin-s1;
103    ssv=solve();
104    if(ssv>0)s=s+ssv;
105
106    s1=0;
107    mx=m-ye;
108    nx=n-xs;
109    for(i=1;i<=mx;i++)
110    {
111        for(j=1;j<=nx;j++)
112        {
113            z=a[i+ye][j+xs];
114            x[mx+1-i][j]=z;
115            s1=s1+z;
116        }
117    }
118
119    pmin=pmin-s1;
120    sse=solve();
121    if(sse>0)s=s+sse;
122
123    pmax=s+pmin;
124    if(p==1)
125        printf("%I64d\n",pmin);
126    else
127        printf("%I64d\n",pmax);
128
129    return 0;
130 }
```

Listing 23.2.2: 2_fence.cpp

```

1 // prof. Ionel Vasile Pit-Rada
2 // backtracking
3
4 #include<stdio.h>
5
6 int m,n,v,xn,xs,yv,ye;
7 long long pmin,pmax;
8 short int a[1002][1002];
9 int x[1002][1002];
10 short int mx,nx;
11
12 void solve(int k, int p, long long s, long long &smax)
```

```

13  {
14      short int i;
15      if(k==mx+1)
16      {
17          if(s>smax) smax=s;
18      }
19      else
20          for(i=p;i<=nx;i++)
21              solve(k+1,i,s+x[k][i],smax);
22  }
23
24 int main()
25 {
26     long long tot,s,s1,snv,sne,ssv,sse;
27     int z,i,j,p;
28     freopen("fence.in","r",stdin);
29     freopen("fence.out","w",stdout);
30
31     scanf("%d %d %d %d %d %d %d %d\n", &p, &m, &n, &v, &xn, &xs, &yv, &ye);
32
33     tot=0;
34     for(i=1;i<=m;i++)
35     {
36         for(j=1;j<=n;j++)
37         {
38             scanf("%d ",&z);
39             a[i][j]=z-v;
40             tot=tot+a[i][j];
41         }
42     }
43     pmin=tot;
44     s=0;
45     s1=0;
46
47     mx=yv-1;
48     nx=xn-1;
49     for(i=1;i<=mx;i++)
50     {
51         x[i][0]=0;
52         for(j=1;j<=nx;j++)
53         {
54             z=a[i][nx+1-j];
55             s1=s1+z;
56             x[i][j]=x[i][j-1]+z;
57         }
58     }
59     pmin=pmin-s1;
60     snv=(long long)-20000000000;
61
62     if(p==2)solve(1,0,0,snv);
63     if(snv>0)s=s+snv;
64
65     s1=0;
66     mx=ye-1;
67     nx=n-xn;
68     for(i=1;i<=mx;i++)
69     {
70         x[i][0]=0;
71         for(j=1;j<=nx;j++)
72         {
73             z=a[i][j+xn];
74             s1=s1+z;
75             x[i][j]=x[i][j-1]+z;
76         }
77     }
78     pmin=pmin-s1;
79     sne=(long long)-20000000000;
80     if(p==2)solve(1,0,0,sne);
81     if(sne>0)s=s+sne;
82
83     s1=0;
84     mx=m-yv;
85     nx=xs-1;
86     for(i=1;i<=mx;i++)
87     {
88         x[mx+1-i][0]=0;

```

```

89         for(j=1; j<=nx; j++)
90     {
91         z=a[i+yv][nx+1-j];
92         x[mx+1-i][j]=x[mx+1-i][j-1]+z;
93         s1=s1+z;
94     }
95 }
96
97 pmin=pmin-s1;
98 ssv=(long long)-200000000000;
99 if(p==2)solve(1,0,0,ssv);
100 if(ssv>0)s=s+ssv;
101
102 s1=0;
103 mx=m-ye;
104 nx=n-xs;
105 for(i=1; i<=mx; i++)
106 {
107     x[mx+1-i][0]=0;
108     for(j=1; j<=nx; j++)
109     {
110         z=a[i+ye][j+xs];
111         x[mx+1-i][j]=x[mx+1-i][j-1]+z;
112         s1=s1+z;
113     }
114 }
115
116 pmin=pmin-s1;
117 sse=(long long)-200000000000;
118 if(p==2)solve(1,0,0,sse);
119 if(sse>0)s=s+sse;
120
121 pmax=s+pmin;
122 if(p==1)
123     printf("%I64d\n",pmin);
124 else
125     printf("%I64d\n",pmax);
126
127 return 0;
128 }
```

Listing 23.2.3: 3_fence.cpp

```

1 // prof. Ionel Vasile Pit-Rada
2
3 #include <iostream>
4 using namespace std;
5
6 long long a[1000][1000], b[1000][1000], best[1000][1000];
7 int m, n, v;
8 int col_top, col_bottom, row_left, row_right;
9
10 long long sum(long long (*a)[1000], int p, int q, int m, int n)
11 {
12     long long s = 0L;
13
14     for (int i = 0; i < m; i++)
15         for (int j = 0; j < n; j++)
16             s += a[p + i][q + j];
17
18     return s;
19 }
20
21 void reflect_x(long long (*a)[1000], int p, int q, int m, int n)
22 {
23     for (int i = 0; i < m; i++)
24         for (int j = 0; j < n/2; j++)
25             swap(a[p + i][q + j], a[p + i][q + n - j - 1]);
26 }
27
28 void reflect_y(long long (*a)[1000], int p, int q, int m, int n)
29 {
30     for (int i = 0; i < m/2; i++)
31         for (int j = 0; j < n; j++)
32             swap(a[p + i][q + j], a[p + m - i - 1][q + j]);
```

```

33 }
34
35
36 long long bestGain(long long (*a)[1000], int p, int q, int m, int n)
37 {
38     long long maxLeft, gain;
39
40     for (int j = 0; j <= n; j++)
41         for (int i = m; i >= 0; i--)
42             b[i][j] = ((i == m) || (j == 0)) ? 0 : b[i+1][j] + a[p+i][q+j-1];
43
44     for (int j = 1; j <= n; j++)
45     {
46         maxLeft = best[0][j-1];
47         for (int i = 0; i <= m; i++)
48         {
49             maxLeft = max<long long>(maxLeft, best[i][j-1]);
50             best[i][j] = maxLeft + b[i][j];
51         }
52     }
53
54     gain = best[0][n];
55     for (int i = 0; i <= m; i++)
56         gain = max<long long>(gain, best[i][n]);
57
58     return gain;
59 }
60
61 int main()
62 {
63     ifstream f("fence.in");
64     ofstream g("fence.out");
65     long long gain;
66     int p;
67     f >> p;
68     f >> m >> n >> v;
69     f >> col_top >> col_bottom >> row_left >> row_right;
70     col_top--; col_bottom--; row_left--; row_right--;
71     for (int i = 0; i < m; i++)
72         for (int j = 0; j < n; j++)
73         {
74             f >> a[i][j];
75             a[i][j] -= v;
76         }
77
78     gain = sum(a, 0, 0, m, n)
79         - sum(a, 0, 0, row_left, col_top)
80         - sum(a, 0, col_top + 1, row_right, n - col_top - 1)
81         - sum(a, row_left + 1, 0, m - row_left - 1, col_bottom)
82         - sum(a, row_right + 1, col_bottom + 1, m - row_right - 1, n - col_bottom - 1);
83     if(p == 1)
84         g << gain << endl;
85     else
86     {
87         reflect_x(a, 0, 0, row_left, col_top);
88         reflect_x(a, row_left + 1, 0, m - row_left - 1, col_bottom);
89         reflect_y(a, row_left + 1, 0, m - row_left - 1, col_bottom);
90         reflect_y(a, row_right + 1, col_bottom + 1,
91                     m - row_right - 1, n - col_bottom - 1);
92
93         gain += bestGain(a, 0, 0, row_left, col_top)
94             + bestGain(a, 0, col_top + 1, row_right, n - col_top - 1)
95             + bestGain(a, row_left + 1, 0, m - row_left - 1, col_bottom)
96             + bestGain(a, row_right + 1, col_bottom + 1,
97                         m - row_right - 1, n - col_bottom - 1);
98         g << gain;
99     }
100    g.close();
101
102    return 0;
103 }
```

Listing 23.2.4: 4_fence.cpp

```

2
3 #include <fstream>
4
5 #define maxn 1010
6
7 using namespace std;
8
9 ifstream fin("fence.in");
10 ofstream fout("fence.out");
11
12 int a[maxn][maxn], aux[maxn][maxn], viz[maxn][maxn];
13 long long dp[maxn][maxn];
14 int n, m, N, M, p, v, xu, xd, yl, yr;
15
16 long long calc()
17 {
18     for (int i = 1; i <= m; ++i)
19         dp[0][i] = 0;
20
21     for (int i = 1; i <= n; ++i)
22     {
23         long long sum = 0;
24         for (int j = 1; j <= m; ++j)
25             sum += aux[i][j];
26
27         long long val = dp[i-1][m];
28         for (int j = m; j >= 1; --j)
29         {
30             dp[i][j] = val + sum;
31             sum -= aux[i][j];
32             val = max(val, dp[i-1][j-1]);
33         }
34     }
35
36     long long ans = 0;
37
38     for (int i = 1; i <= n; ++i)
39         for (int j = 1; j <= m; ++j)
40             ans = max(ans, dp[i][j]);
41
42     return ans;
43 }
44
45 int main()
46 {
47     fin >> p >> N >> M >> v >> xu >> xd >> yl >> yr;
48
49     for (int i = 1; i <= N; ++i)
50     {
51         for (int j = 1; j <= M; ++j)
52         {
53             fin >> a[i][j];
54             a[i][j] -= v;
55         }
56     }
57
58     long long cost = 0;
59
60     for (int i = min(yl, yr); i <= max(yl, yr); ++i)
61     {
62         for (int j = min(xu, xd); j <= max(xu, xd); ++j)
63         {
64             cost += a[i][j];
65             viz[i][j] = 1;
66         }
67     }
68
69     for (int i = 1; i < min(yl, yr); ++i)
70     {
71         cost += a[i][xu];
72         viz[i][xu] = 1;
73     }
74
75     for (int i = N; i > max(yl, yr); --i)
76     {
77         cost += a[i][xd];

```

```

78         viz[i][xd] = 1;
79     }
80
81     for (int i = 1; i < min(xd,xu); ++i)
82     {
83         cost += a[y1][i];
84         viz[y1][i] = 1;
85     }
86
87     for (int i = M; i > max(xd,xu); --i)
88     {
89         cost += a[yr][i];
90         viz[yr][i] = 1;
91     }
92
93     if (p == 1)
94     {
95         fout << cost;
96     }
97     else
98     {
99         int k = xu-1;
100        int h = 1;
101        while (!viz[h][k])
102            ++h;
103        --h;
104        n = 0;
105        for (int j = k; j >= 1; --j)
106        {
107            ++n;
108            m = 0;
109            for (int i = h; i >= 1; --i)
110                aux[n][++m] = a[i][j];
111        }
112        cost += calc();
113
114        k = xu+1;
115        h = 1;
116        while (!viz[h][k])
117            ++h;
118        --h;
119        n = 0;
120        for (int j = k; j <= M; ++j)
121        {
122            ++n;
123            m = 0;
124            for (int i = h; i >= 1; --i)
125                aux[n][++m] = a[i][j];
126        }
127        cost += calc();
128
129        k = xd+1;
130        h = N;
131        while (!viz[h][k])
132            --h;
133        ++h;
134        n = 0;
135        for (int j = k; j <= M; ++j)
136        {
137            ++n;
138            m = 0;
139            for (int i = h; i <= N; ++i)
140                aux[n][++m] = a[i][j];
141        }
142        cost += calc();
143
144        k = xd-1;
145        h = N;
146        while (!viz[h][k])
147            --h;
148        ++h;
149        n = 0;
150        for (int j = k; j >= 1; --j)
151        {
152            ++n;
153            m = 0;

```

```

154         for (int i = h; i <= N; ++i)
155             aux[n][++m] = a[i][j];
156     }
157     cost += calc();
158
159     fout << cost;
160 }

```

Listing 23.2.5: 5_fence.cpp

```

1 // prof. Radu Visinescu,Colegiul National "I.L.Caragiale" Ploiesti
2 // backtracking
3 #include <iostream>
4 #include <fstream>
5
6 using namespace std;
7
8 ifstream fin("fence.in");
9 ofstream fout("fence.out");
10
11 int m,n,v,x_nord,x_sud,y_vest,y_est,p;
12 int a[1002][1002],w[1002][1002];
13 int st[2000];
14
15 int profit(int y1, int y2,int x1, int x2,int v)
16 {
17     int i,j;int profitul=0;
18     for(i=y1;i<=y2;i++)
19         for(j=x1;j<=x2;j++)
20             if(w[i][j]==1) profitul+=a[i][j]-v;
21     return profitul;
22 }
23
24 int continua(int p,int n,int k)
25 {int i,nr=0;
26     for(i=1; i<=p;i++)
27         if(st[i]==1) nr++;
28     if((p==n)&&(nr<=k))return 1;
29     else return 0;
30 }
31
32 int solutie(int p,int n,int k)
33 {int i,nr=0;
34     for(i=1; i<=p;i++)
35         if(st[i]==1) nr++;
36     if ((p==n)&&(nr==k))return 1;
37     else return 0;
38 }
39
40 void tipar(int p)
41 {
42     int i;
43     for (i=1;i<=p;i++)
44         cout<<st[i]<<" ";
45     cout<<endl;
46 }
47
48 int back_nord_vest(int n2, int k)
49 {int p,profitul,profitul_maxim;int i,j,s,lin,col;
50 profitul_maxim=0;
51 p=1;
52 st[p]=-1;
53 while (p>0)
54 { while (st[p]<1)
55     { st[p]++;
56         if (continua(p,n2,k))
57             {if (solutie(p,n2,k))
58                 { for (i=1;i<=y_vest-1;i++)
59                     for(j=1;j<=x_nord-1;j++)
60                         w[i][j]=0;
61                     lin=y_vest-1;col=1;
62                     for(s=1;s<=n2;s++)
63                         if (st[s]==0)col++;
64                         else {for(j=col;j<=x_nord-1;j++)w[lin][j]=1;

```

```

65             lin--;
66         profitul=profit(1,y_vest-1,1,x_nord-1,v);
67         if (profitul>profitul_maxim) profitul_maxim=profitul;
68     }
69
70     else { p++;st[p]=-1; }
71 }
72 p--;
73 }
74 return profitul_maxim;
75 }
76
77 int back_sud_vest(int n2, int k)
78 {int p,profitul,profitul_maxim;int i,j,s,lin,col;
79 profitul_maxim=0;
80 p=1;
81 st[p]=-1;
82 while (p>0)
83 { while (st[p]<1)
84 { st[p]++;
85   if (continua(p,n2,k))
86   {if (solutie(p,n2,k))
87   { for (i=y_vest+1;i<=m;i++)
88     for(j=1;j<=x_sud-1;j++)
89       w[i][j]=0;
90     lin=y_vest+1;col=1;
91     for(s=1;s<=n2;s++)
92       if (st[s]==0)col++;
93       else {for(j=col;j<=x_sud-1;j++)w[lin][j]=1;
94         lin++; }
95     profitul=profit(y_vest+1,m,1,x_sud-1,v);
96     if (profitul>profitul_maxim) profitul_maxim=profitul;
97   }
98   else { p++;st[p]=-1; }
99 }
100 }
101 p--;
102 }
103 return profitul_maxim;
104 }
105
106 int back_nord_est(int n2, int k)
107 {int p,profitul,profitul_maxim;int i,j,s,lin,col;
108 profitul_maxim=0;
109 p=1;
110 st[p]=-1;
111 while (p>0)
112 { while (st[p]<1)
113 { st[p]++;
114   if (continua(p,n2,k))
115   {if (solutie(p,n2,k))
116   {
117     for (i=1;i<=y_est-1;i++)
118       for(j=x_nord+1;j<=n;j++)
119         w[i][j]=0;
120       lin=y_est-1;col=n;
121       for(s=1;s<=n2;s++)
122         if (st[s]==0)col--;
123         else {for(j=x_nord+1;j<=col;j++)
124           w[lin][j]=1;
125           lin--; }
126       profitul=profit(1,y_est-1,x_nord+1,n,v);
127       if (profitul>profitul_maxim) profitul_maxim=profitul;
128     }
129   }
130   else { p++;st[p]=-1; }
131 }
132 p--;
133 }
134 }
135 return profitul_maxim;
136 }
137
138 int back_sud_est(int n2, int k)
139 {int p,profitul,profitul_maxim;int i,j,s,lin,col;
140 profitul_maxim=0;

```

```

141     p=1;
142     st[p]=-1;
143     while (p>0)
144     {
145         while (st[p]<1)
146         {
147             if (continua(p,n2,k))
148             {
149                 for (i=y_est+1;i<=m;i++)
150                     for(j=x_sud+1;j<=n;j++)
151                         w[i][j]=0;
152                 lin=y_est+1;col=n;
153                 for(s=1;s<=n2;s++)
154                     if (st[s]==0) col--;
155                     else {for(j=x_sud+1;j<=col;j++) w[lin][j]=1;
156                             lin++;}
157                     profitul=profit(y_est+1,m,x_sud+1,n,v);
158                     if (profitul>profitul_maxim) profitul_maxim=profitul;
159                 }
160             else { p++;st[p]=-1; }
161         }
162         p--;
163     }
164     return profitul_maxim;
165 }
166
167 void citire()
168 {int i,j;
169     fin>>p>>m>>n>>v>>x_nord>>x_sud>>y_vest>>y_est;
170     for (i=1;i<=m;i++)
171         for (j=1;j<=n;j++)
172             fin>>a[i][j];
173     fin.close();
174 }
175
176 void aria_minima()
177 {int i,j,int minx,maxx,miny,maxy;
178     minx=min(x_nord,x_sud);
179     maxx=max(x_nord,x_sud);
180     miny=min(y_vest,y_est);
181     maxy=max(y_vest,y_est);
182     for(j=1;j<minx;j++) w[y_vest][j]=1;
183     for(j=n;j>maxx;j--) w[y_est][j]=1;
184     for(i=1;i<miny;i++) w[i][x_nord]=1;
185     for(i=m;i>maxy;i--) w[i][x_sud]=1;
186
187     for(i=miny;i<=maxy;i++)
188         for(j=minx;j<=maxx;j++)
189             w[i][j]=1;
190 }
191
192 int main()
193 {int p2;
194     citire();
195     aria_minima();
196     if(p==1) fout<<profit(1,m,1,n,v)<<' \n';
197
198     if(p==2)
199     {
200         p2=profit(1,m,1,n,v);
201         p2+=back_nord_vest(x_nord-1+y_vest-1,y_vest-1);
202         p2+=back_sud_vest(x_sud-1+m-y_vest,m-y_vest);
203         p2+=back_nord_est(n-x_nord+y_est-1,y_est-1);
204         p2+=back_sud_est(n-x_sud+m-y_est,m-y_est);
205         fout<<p2<<' \n';
206     }
207     fout.close();
208     return 0;
209 }

```

23.3 nmult

Se consideră trei numere naturale nenule n, k și w .

Cerințe

Să se scrie un program care determină numărul m al mulțimilor de forma $\{x_1, x_2, \dots, x_k\}$ având ca elemente numere naturale nenule, ce satisfac simultan condițiile:

- $1 \leq x_1 < x_2 < \dots < x_k \leq n$
- $x_{i+1} - x_i \geq w, 1 \leq i \leq k - 1$

Date de intrare

Fișierul de intrare **nmult.in** conține pe prima linie trei numere naturale nenule n, k, w separate prin câte un spațiu, cu semnificația de mai sus.

Date de ieșire

Fișierul de ieșire **nmult.out** va conține pe prima linie restul împărțirii numărului m la 666013.

Restricții și precizări

- $1 \leq n, k, w \leq 1\ 000\ 000$;

Exemplu:

nmult.in	nmult.out	Explicații
5 2 2	6	n=5, k=2, w=2 Există 6 mulțimi cu 2 elemente, astfel încât diferența între oricare 2 termeni consecutivi să fie cel puțin 2: {1,3}; {1,4}; {1,5}; {2,4}; {2,5}; {3,5}
10 3 4	4	n=10, k=3, w=4 Există 4 mulțimi cu 3 elemente, astfel încât diferența între oricare 2 termeni consecutivi să fie cel puțin 4: {1,5,9}; {1,5,10}, {1,6,10}; {2,6,10};
10 4 4	0	n=10, k=4, w=4 Nu există nicio mulțime de 4 elemente în care condițiile să fie îndeplinite.

Timp maxim de executare/test: **0.** secunde

Memorie: total **4 MB** din care pentru stivă **2 MB**

Dimensiune maximă a sursei: **5 KB**

23.3.1 Indicații de rezolvare

prof. Cheșcă Ciprian, Liceul Tehnologic "Costin Nenițescu" Buzău

Varianta 1 (backtracking)

Se pot genera toate mulțimile de cardinal k cu elemente din mulțimea $\{1, 2, 3, \dots, n\}$ cu proprietatea că diferența dintre două elemente consecutive este de cel puțin w , utilizând metoda backtracking. Această varianta obține aproximativ 30% din punctaj.

Varianta 2 (recursivitate)

Fie $N(k, n, w)$ numărul căutat. Pentru $w = 1$, numărul $N(k, n, 1)$ este numărul mulțimilor de k elemente distincte ce se pot forma cu numerele $1, 2, \dots, n$.

Deci $N(k, n, 1) = \text{comb}(n, k)$ adică combinări de n luate câte k , cu convenția obișnuită că pentru $n < k$ această valoare este 0.

Mai departe vom obține o relație de recurență pentru numerele $N(k, n, w)$.

Fie transformarea

$$y_i = x_i - i + 1 \text{ cu } 1 \leq i \leq k \quad (1)$$

Numerele y_i satisfac restricțiile

$1 \leq y_1 < y_2 < \dots < y_k \leq n - k + 1$ și $y_{i+1} - y_i = x_{i+1} - x_i - 1 \geq w - 1$ cu $1 \leq i \leq k - 1$ (2)
 dacă x_1, x_2, \dots, x_k satisfac restricțiile din enunț.

Reciproc, având întregii y_1, y_2, \dots, y_k ce îndeplinesc relația (2) din (1) se obține $x_i = y_i + i - 1$ cu $1 \leq i \leq k$ iar întregii x_1, x_2, \dots, x_k satisfac condițiile din enunț.

Prin urmare

$$N(k, n, w) = N(k, n - k + 1, w - 1), (\forall) n, k, w \in \mathbb{N}^* \quad (3)$$

relație ce poate fi utilizată pentru determinarea prin recurență a valorii cerute de program.

Pentru calculul combinărilor se poate utiliza *teorema Legendre* sau *invers modular*.

în funcție de implementare această variantă poate obține între 75% și 100% din punctaj.

23.3.2 *Rezolvare detaliată

23.3.3 Cod sursă

Listing 23.3.1: 1_nmult.cpp

```

1 // prof. Pit-Rada Vasile Ionel
2 #include<stdio.h>
3 long n,k,w,H=666013,r;
4 long long m,z,p,x,y,t,xx,yy,i;
5 char a[1000010];
6
7 int main()
8 {
9     freopen("nmult.in","r",stdin);
10    freopen("nmult.out","w",stdout);
11
12    scanf("%ld%ld%ld",&n,&k,&w);
13    m=n-(k-1)*(w-1);
14    xx=k;
15    yy=m-k;
16
17    if (xx<yy) {xx=m-k; yy=k;}
18    z=1;
19    for (p=2;p<=m;p++)
20    {
21        if (a[p]==0)
22        {
23            for (i=p*p;i<=m;i=i+p)
24                a[i]=1;
25            x=xx;
26            y=yy;
27            t=0;
28            while(x)
29            {
30                t=(t+x%p+y%p)/p;
31                if(t!=0) z=(z*p)%H;
32                x=x/p;
33                y=y/p;
34            }
35        }
36    }
37
38    r=z;
39    if (m<k)
40        printf("0\n");
41    else
42        printf("%ld\n",r);
43    fclose(stdout);
44    return 0;
45 }
```

Listing 23.3.2: 2_nmult.cpp

```

1 // prof. Daniel Popa
2 // backtracking
3 #include <iostream>
```

```

4 #include <fstream>
5
6 using namespace std;
7
8 int n,k,w,t=0,m=0,i=0;
9 int v[100];
10
11 void bk(int x,int a)
12 {int i;
13 if(x==k){t++; t%=666013;}
14 else
15 for(i=w;i<=n-a-(k-x-1)*w;i++)
16     bk(x+1,a+i);
17 }
18
19 int main()
20 {
21 ifstream fin("nmult.in");
22 ofstream fout("nmult.out");
23 fin>>n>>k>>w;
24
25 do
26 {t=0;
27 i++;
28 bk(1,i);
29 m+=t;
30 m%=666013;
31 }while(t!=0);
32
33 fout<<m;
34
35     return 0;
36 }
```

Listing 23.3.3: 3_nmultiplication.cpp

```

1 // sursa cu combinari + Legendre
2 // prof. Chesca Ciprian
3 #include <fstream>
4 #define M 666013
5 #define nmax 1000001
6
7 using namespace std;
8
9 int n,k,w;
10 ifstream f("nmult.in");
11 ofstream g("nmult.out");
12
13 char v[nmax];
14 int p[nmax/10],desc[nmax/10],nr=0;
15
16 // Ciurul lui Eratostene
17 void ciur(int n)
18 {
19     int i, j;
20     for (i = 2; i <= n; ++i)
21     {
22         if (v[i] == 0)
23         {
24             p[++nr]=i;
25             for (j = i + i; j <= n; j += i)
26             {
27                 v[j] = 1;
28             }
29         }
30     }
31 }
32
33 // descompunere in factori primi a lui x! cu Legendre
34 void dsc(int x,int op)
35 {
36     long long i,s,pp,ok;
37
38     for(i=1;i<=nr;i++)
39     {
```

```

40         s=0;
41         pp=p[i];
42         ok=1;
43         while (ok)
44         {
45             s=s+x/pp;
46             if (x/pp==0) ok=0;
47             pp=pp*p[i];
48
49         }
50
51         if (op==1)
52             desc[i]+=s;
53         else
54             desc[i]-=s;
55     }
56 }
57
58 int mdl()
59 {
60     long long i,j,pp=1,r=1;
61     for(i=1;i<=nr;i++)
62     {
63         pp = 1;
64         for(j=1;j<=desc[i];j++)
65             pp = (pp*p[i]) % M;
66
67         r = (r*pp) % M;
68     }
69
70     return r;
71 }
72
73 int main()
74 {
75     f>>n>>k>>w;
76     ciur(n-(w-1)*(k-1));
77
78     dsc(n-(w-1)*(k-1),1);
79
80     dsc(k,0);
81     dsc(n-(w-1)*(k-1)-k,0);
82
83     if (n-(k-1)*(w-1)<k)
84         g<<"0"<<"\n";
85     else
86         g<<mdl();
87
88     f.close();
89     g.close();
90
91     return 0;
92 }
```

Listing 23.3.4: 4_nmult.cpp

```

1 // sursa cu formula de recurenta + combinari recursiv
2 // prof. Chesca Ciprian
3 #include <fstream>
4 #include <math.h>
5
6 #define M 666013
7
8 using namespace std;
9
10 int n,k,w;
11 ifstream f("nmult.in");
12 ofstream g("nmult.out");
13
14 long long comb(int n, int k)
15 {
16     if (k==0)
17         return 1;
18     else
19         return (n*comb(n-1,k-1)/k)%M;
```

```

20 }
21
22 int nmult(int k, int n, int w)
23 {
24     if (n<k)
25         return 0;
26     else
27         if (w==1)
28             return comb(n,k);
29         else
30             return nmult(k,n-k+1,w-1);
31 }
32
33 int main()
34 {
35     f>>n>>k>>w;
36
37     g<<nmult(k,n,w);
38
39     f.close();
40     g.close();
41
42     return 0;
43 }
```

Listing 23.3.5: 5_nmult.cpp

```

1 // sursa cu formula de recurenta + Legendre
2 // prof. Chesca Ciprian
3 #include <iostream>
4 #define M 666013
5 #define nmax 1000001
6
7 using namespace std;
8
9 int n,k,w;
10 ifstream f("nmult.in");
11 ofstream g("nmult.out");
12
13 char v[nmax];
14 int p[nmax/10],desc[nmax/10],nr=0;
15
16 // Ciurul lui Eratostene
17 void ciur(int n)
18 {
19     int i, j;
20     for (i = 2; i <= n; ++i)
21     {
22         if (v[i] == 0)
23         {
24             p[++nr]=i;
25             for (j = i + i; j <= n; j += i)
26             {
27                 v[j] = 1;
28             }
29         }
30     }
31 }
32
33 // descompunere in factori primi a lui x! cu Legendre
34 void dsc(int x,int op)
35 {
36     long long i,s,pp,ok;
37
38     for(i=1;i<=nr;i++)
39     {
40         s=0;
41         pp=p[i];
42         ok=1;
43         while (ok)
44         {
45             s=s+x/pp;
46             if (x/pp==0)
47                 ok=0;
48             pp=pp*p[i];
```

```

49         }
50
51     if (op==1)
52         desc[i]+=s;
53     else
54         desc[i]-=s;
55     }
56 }
57
58 int mdl()
59 {
60     long long i,j,pp=1,r=1;
61     for(i=1;i<=nr;i++)
62     {
63         pp = 1;
64         for(j=1;j<=desc[i];j++)
65             pp = (pp*p[i]) % M;
66
67         r = (r*pp) % M;
68     }
69
70     return r;
71 }
72
73 int nmult(int k, int n, int w)
74 {
75     if (n<k)
76         return 0;
77     else
78         if (w==1)
79         {
80             // calculez combinari de n luate cate k modulo M
81             dsc(n,1);
82             dsc(k,0);
83             dsc(n-k,0);
84             return mdl();
85         }
86         else
87             return nmult(k,n-k+1,w-1);
88 }
89
90 int main()
91 {
92     f>>n>>k>>w;
93     ciur(n);
94
95     g<<nmult(k,n,w);
96
97     f.close();
98     g.close();
99
100    return 0;
101 }
```

Listing 23.3.6: 6_nmultiplication.cpp

```

1 // solutie invers modular C(n-(k-1)*(w-1), k)
2 // prof. Eugen Nodea
3 # include <fstream>
4 # define M 666013
5
6 using namespace std;
7
8 ifstream f("nmult.in");
9 ofstream g("nmult.out");
10
11 long long n, k, w;
12
13 void GCD (long long A, long long B, long long &X, long long &Y)
14 {
15     long long X0, Y0;
16     if (B ==0 )
17     {
18         X = 1LL, Y = 0;
19         return;
```

```

20     }
21
22     GCD (B, A % B, X0, Y0);
23     X = Y0;
24     Y = X0 - Y0 * (A/B);
25 }
26
27 long long inversm(long long N)
28 {
29     long long X, Y;
30
31     GCD (N, M, X, Y);
32     if (X<=0)
33     {
34         X = M + X % M;
35     }
36     return X;
37 }
38
39 long long comb(long long N, long long K)
40 {
41     long long A, B, i;
42     A = B = 1;
43
44     for(i=K+1; i<=N; ++i)
45         if (i % M) A = ((A % M) * (i % M)) % M;
46
47     for(i=1; i<=N-K; ++i)
48         if (i % M) B = ((B % M) * (i % M)) % M;
49
50     A = ((A % M) * (inversm(B) % M)) % M;
51     return A;
52 }
53
54 int main()
55 {
56     f >> n >> k >> w;
57     // C(n-(w-1)*(k-1), k)
58     g << comb(n - (k-1) * (w-1), k);
59     return 0;
60 }
```

Listing 23.3.7: 7_nmultiplication.cpp

```

1 // solutie triunghiul lui Pascal C(n-(k-1)*(w-1), k)
2 // prof. Eugen Nodea
3 # include <iostream>
4 # include <cstring>
5 # define M 666013
6
7 using namespace std;
8
9 ifstream f("nmult.in");
10 ofstream g("nmult.out");
11
12 int n, k, w;
13 int L0[100001], L[100001];
14
15 void comb(int N)
16 {
17     int i, j;
18     L0[0] = L0[1] = 1; // linia 1
19     for (i=2; i<=N; ++i)
20     {
21         L[0] = L[i] = 1;
22         for (j=1; j<i; ++j)
23             L[j] = (L0[j-1] % M + L0[j] % M) % M;
24
25         memcpy(L0, L, sizeof(L));
26     }
27 }
28
29 int main()
30 {
31     f >> n >> k >> w;
```

```

32     comb(n - (k-1) * (w-1));
33     g << L[k];
34
35     return 0;
36 }

```

Listing 23.3.8: 8_nmult.cpp

```

1 // prof. Tucu Galatan
2 #include <iostream>
3
4 using namespace std;
5
6 #define MOD 666013
7
8 ifstream fin("nmult.in");
9 ofstream fout("nmult.out");
10
11 // nr[i][v] - in cate moduri pot avea cel mult valoarea v la poz i
12 int nr[2][100001];
13 int n, k, w, p, c, i;
14
15 int main()
16 {
17     fin >> n >> k >> w;
18
19     for ( int v = 1; v <= n; ++v )
20         nr[0][v] = v;
21
22     for ( i = 2, c = 1, p = 0; i <= k; p = !p, c = !c, ++i )
23     {
24         nr[c][(i - 1)* w] = 0;
25         for ( int v = (i - 1)* w + 1; v <= n - (k - i) * w; ++v )
26             nr[c][v] = (nr[c][v - 1] + nr[p][v - w]) % MOD;
27     }
28
29     fout << nr[p][n] << '\n';
30     fin.close();
31     fout.close();
32     return 0;
33 }

```

Listing 23.3.9: 9_nmult.cpp

```

1 // student Alexandru Murtaza - Universitatea Bucuresti
2 #include <iostream>
3
4 using namespace std;
5
6 const char InFile[] = "nmult.in";
7 const char OutFile[] = "nmult.out";
8 const int MOD = 666013;
9
10 ifstream fin(InFile);
11 ofstream fout(OutFile);
12
13 int N, K, W;
14
15 inline int mypow(int A, int B)
16 {
17     int ans=1;
18     for ( ; B; B >>= 1)
19     {
20         if (B & 1)
21             ans = (1LL*ans*A) %MOD;
22
23         A = (1LL*A*A) % MOD;
24     }
25     return ans;
26 }
27
28 inline int invmod(int A)
29 {

```

```

30     return mypow(A,MOD-2);
31 }
32
33 int main()
34 {
35     fin >> N >> K >> W;
36     fin.close();
37
38     long long CN = N - (W-1)*(K-1);
39     int CK = K;
40
41     if (CN < CK )
42     {
43         fout << "0\n";
44         fout.close();
45         return 0;
46     }
47
48     int st = 1;
49     int dr = CN;
50     int other = CK;
51     if (CK < CN - CK)
52     {
53         st = CN - CK + 1;
54         other = CK;
55     }
56     else
57     {
58         st = CK + 1;
59         other = CN - CK;
60     }
61
62     int sol = 1;
63     for (int i = st; i <= dr; ++i)
64         sol = (1LL * sol*i) % MOD;
65
66     int other_p = 1;
67     for (int i = 1; i <= other; ++i)
68         other_p = (1LL * other_p*i) % MOD;
69
70     sol = (1LL * sol*invmod(other_p)) % MOD;
71
72     fout << sol << "\n";
73     fout.close();
74     return 0;
75 }
```

23.4 procente

Problema 4 - procente

100 de puncte

Definim o modificare procentuală de preț ca fiind o pereche $(c \ p)$ formată dintr-un caracter $c \in \{'+', '-' \}$ și un număr natural p . Dacă $c = '+'$ atunci are loc o scumpire iar dacă $c = '-'$ atunci are loc o ieftinire a unui preț, iar numărul p reprezintă procentul de modificare a prețului.

Exemple de modificări procentuale de preț:

- $(+ \ 35)$ - reprezintă scumpirea unui preț cu 35% ;
- $(- \ 50)$ - reprezintă ieftinirea unui preț cu 50%

Unui preț inițial î se poate aplica o succesiune de n modificări procentuale de preț obținându-se un preț final.

Numim ciclu de preț de lungime n o succesiune de n modificări procentuale de preț, cu proprietatea că prețul final este egal cu prețul inițial.

Exemple de cicluri de preț :

- de lungime $n = 2$: $(- \ 20)(+ \ 25)$
- de lungime $n = 3$: $(- \ 50)(+ \ 25)(+ \ 60)$

Cerințe

Să se scrie un program care citește un număr natural n și determină numărul de cicluri de preț de lungime n distințe ce conțin cel puțin o dată, o modificare procentuală cunoscută $(C \ P)$.

Date de intrare

Fișierul de intrare **procente.in** conține pe prima linie numărul natural n și pe a doua linie un caracter $C \in \{'+', '-' \}$, urmat de un număr natural P , despărțite printr-un spațiu, cu semnificația de mai sus.

Date de ieșire

Fișierul de ieșire **procente.out** va conține pe prima linie numărul căutat.

Restricții și precizări

- $2 \leq n \leq 80$
- $C \in \{'+', '-' \}$
- Valoarea procentului p în caz de scumpire este cuprinsă între 0 și 100 inclusiv.
- Valoarea procentului p în caz de ieftinire este cuprinsă între 1 și 99 inclusiv.
- Două modificări procentuale de preț $(c_1 p_1), (c_2 p_2)$ sunt diferite dacă $c_1 \neq c_2$ sau $p_1 \neq p_2$
- Două cicluri de preț de lungime n sunt distințe, dacă diferă prin cel puțin o modificare procentuală de preț.
- Două cicluri de preț de lungime n ce conțin aceleași modificări procentuale, dar în altă ordine, sunt identice.
- Pentru 28% din punctaj $n \leq 20$, pentru 60% din punctaj $n \leq 40$, pentru 100% din punctaj $n \leq 80$

Exemple:

procente.in	procente.out	Explicații
2 - 20	1	Există o singură succesiune de 2 modificări procentuale de preț ce conține și o ieftinire cu 20% care are prețul final egal cu prețul inițial. Această succesiune este : (- 20)(+ 25).
3 + 25	4	Există patru succesiuni distințe de 3 modificări procentuale de preț ce conțin cel puțin o scumpire cu 25% care au prețul final egal cu prețul inițial. Aceste succesiuni sunt: (- 50)(+ 25)(+ 60); (- 36)(+ 25)(+ 25); (- 60)(+ 25)(+ 100); (- 20)(+ 25)(+ 0).

Timp maxim de executare/test: **1.0** secunde pe Windows, **0.4** secunde pe Linux

Memorie: total **24 MB** din care pentru stivă **6 MB**

Dimensiune maximă a sursei: **5 KB**

23.4.1 Indicații de rezolvare

1. prof. Cheșcă Ciprian, Liceul Tehnologic "Costin Nenițescu" Buzău,
2. student Murtaza Alexandru, Universitatea București,
3. student Nițu Mihai, Universitatea București,

Varianta 1 (Prof. Cheșcă Ciprian)

Se poate demonstra ușor formula de calcul a prețului final pornind de la un preț inițial și aplicând n modificări procentuale de preț.

Această formulă este:

$$P_i = P_f \cdot \frac{100 \pm p_1}{100} \cdot \frac{100 \pm p_2}{100} \cdots \cdot \frac{100 \pm p_n}{100}$$

unde

P_i = prețul inițial,

P_f = prețul final

p_i = procentele de modificare a prețului ($1 \leq i \leq n$)

Se utilizează + când se face o scumpire și se utilizează - când se face o ieftinire.

Dacă $P_i = P_f$ atunci relația de mai sus se scrie astfel:

$$(100 \pm p_1) \cdot (100 \pm p_2) \cdot \dots \cdot (100 \pm p_n) = (100)^n = 2^{2n} \cdot 5^{2n}$$

și înținând cont că toate numere din această ecuație sunt naturale, deducem că parantezele sunt divizori ai lui $(100)^n$ cuprinși între 1 și 200.

Dedecem de aici că parantezele pot lua următoarele 19 valori: {1, 2, 4, 5, 8, 10, 16, 20, 25, 32, 40, 50, 64, 80, 100, 125, 128, 160, 200}

Cu aceste observații se poate face un *backtracking* punând pe stivă puterile lui 2 și 5 ale fiecărui divizor de mai sus. Soluția obține punctaj minim.

Varianta 2 (student - Mihai Nițu) 60 de puncte

Potem utiliza o relație de recurență de forma $\text{solve}(x,y,k,i)$ = numărul de moduri în care se poate descompune un număr format din x factori de 2 și y factori de 5 în k dintre numerele numai cu factori de 2 și 5, folosind numai numere dintre acestea de la poziția i mai departe. Pentru a afla $\text{solve}(x,y,k,i)$ se va parurge vectorul de numere de la i la sfârșit și pentru fiecare j (de la i la sfârșitul vectorului) se adună valoarea lui $\text{solve}(x - \text{fatori de 2 din } v[j], y - \text{fatori de 5 din } v[j], k-1, j)$.

Cu memorarea valorilor într-un tablou de 4 dimensiuni, complexitatea va fi (N^3) cu o constantă de 19^2 .

Varianta 3 (student - Alexandru Murtaza) 100 de puncte

Se poate proceda ca la *problema rucsacului* pentru a scapa de un factor de 19. Adăugăm pe rând cele 19 elemente în matricea $\text{rucsac}[x][y][k] =$ numărul de moduri în care se descompune un număr cu x factori de 2 și y factori de 5, cu k dintre cele 19 numere. Adăugarea unui element (a,b) din cele 19 (cu a factori de 2 și b factori de 5) în matrice implica actualizarea $\text{rucsac}[x+a][y+b][k+1] += \text{rucsac}[x][y][k]$.

Complexitate: $O(N^3)$ cu o constantă de 19

23.4.2 *Rezolvare detaliată

23.4.3 Cod sursă

Listing 23.4.1: 1_procente.cpp

```

1 // student Alexandru Murtaza - Universitatea Bucuresti
2 #include <fstream>
3
4 using namespace std;
5
6 const char InFile[] = "procente.in";
7 const char OutFile[] = "procente.out";
8 const int MaxN = 82;
9 const int MaxVal = 205;
10
11 ifstream fin(InFile);
12 ofstream fout(OutFile);
13
14 int N,p,v[MaxVal];
15 int e2[MaxVal];
16 int e5[MaxVal];
17 char ch;
18 long long D[MaxN*2][MaxN*2][MaxN];
19
20 int main()
21 {
22     fin >> N;
23     fin >> ch >> p;
24     fin.close();
25
26     if (ch == '+')
27         p += 100;
28     else
29         p = 100 - p;
30
31     for (int i = 1; i <= 200; ++i)
32     {
33         int e2t = 0;

```

```

34         int e5t = 0;
35
36         int t = i;
37         while (t % 2 == 0)
38         {
39             t /= 2;
40             ++e2t;
41         }
42         while (t % 5 == 0)
43         {
44             t /= 5;
45             ++e5t;
46         }
47
48         if (t == 1)
49         {
50             v[+v[0]] = i;
51             e2[v[0]] = e2t;
52             e5[v[0]] = e5t;
53         }
54     }
55
56     int N2 = N*2;
57
58     D[0][0][0] = 1;
59     for (int nr = 1; nr <= v[0]; ++nr)
60     {
61         int e2t = e2[nr];
62         int e5t = e5[nr];
63
64         for (int i = 0; i <= N2 - e2t; ++i)
65             for (int j = 0; j <= N2 - e5t; ++j)
66                 for (int k = 0; k < N; ++k)
67                     D[i + e2t][j + e5t][k + 1] += D[i][j][k];
68     }
69
70     int A = N2;
71     int B = N2;
72
73     while (p % 2 == 0)
74     {
75         --A;
76         p /= 2;
77     }
78     while (p % 5 == 0)
79     {
80         --B;
81         p /= 5;
82     }
83
84     if (p != 1)
85         fout << "0";
86     else
87         fout << D[A][B][N-1];
88
89     fout.close();
90     return 0;
91 }
```

Listing 23.4.2: 2_procente.cpp

```

1 // Vasile - Ionel Pit-Rada
2 #include<iostream>
3 #include<iomanip>
4
5 using namespace std;
6
7 ifstream fin("procente.in");
8 ofstream fout("procente.out");
9
10 struct dc
11 {
12     int c2,c5;
13 };
14
```

```

15 long long a[2][161][161][19],s;
16 int n, p, d2=0, d5=0, n2, i, j, k, l, h, hh, h1, h2;
17 char C;
18 dc aux, v[19]={{0,0},{1,0},{2,0},{0,1},{3,0},{1,1},{4,0},{2,1},{0,2},{5,0},
19 {3,1},{1,2},{6,0},{4,1},{0,3},{7,0},{5,1},{3,2},{2,2}};
20
21 int main()
22 {
23     fin>>n>>C>>p;
24
25     n2=2*n;
26     if(C=='-') p=-p;
27     p=p+100;
28
29     while(p%2==0)
30     {
31         d2++;
32         p=p/2;
33     }
34
35     while(p%5==0)
36     {
37         d5++;
38         p=p/5;
39     }
40
41     if(p!=1)
42     {
43         fout<<0;
44         fout.close();
45         return 0;
46     }
47
48     for(i=0;i<=18;i++)
49         if(v[i].c2==d2 && v[i].c5==d5)
50             break;
51
52     aux=v[i];
53     v[i]=v[0];
54     v[0]=aux;
55
56     a[0][v[0].c2][v[0].c5][0]=1;
57     for(h=1;h<=n-1;h++)
58     {
59         h1=h%2;
60         h2=1-h1;
61         for(i=0;i<=n2;i++)
62             for(j=0;j<=n2;j++)
63                 for(k=0;k<=18;k++)
64                 {
65                     hh=a[h2][i][j][k];
66                     if(hh!=0)
67                         for(l=k;l<=18;l++)
68                             if(i+v[l].c2<=n2 && j+v[l].c5<=n2)
69                                 a[h1][i+v[l].c2][j+v[l].c5][l]+=hh;
70
71                     a[h2][i][j][k]=0;
72                 }
73     }
74
75     s=0;
76     for(i=0;i<=18;i++)
77         s=s+a[h1][n2][n2][i];
78
79     fout<<s<<"\n";
80
81     fout.close();
82     fin.close();
83     return 0;
84 }
```

Listing 23.4.3: 3_procente.cpp

```

1 // Vasile Ionel Pit-Rada
2 #include<fstream>
```

```

3 #include<iomanip>
4
5 using namespace std;
6
7 ifstream fin("procente.in");
8 ofstream fout("procente.out");
9
10 struct dc
11 {
12     int c2,c5;
13 };
14
15 long long a[2][161][161][19];
16 int n, p, d2=0, d5=0, n2, i, j, k, l, h, hh, h1, h2;
17 char C;
18 dc aux, v[19]={{0,0},{1,0},{2,0},{0,1},{3,0},{1,1},{4,0},{2,1},{0,2},
19             {5,0},{3,1},{1,2},{6,0},{4,1},{0,3},{7,0},{5,1},{3,2},{2,2}};
20
21 int main()
22 {
23     fin>>n>>C>>p;
24
25     n2=2*n;
26     if(C=='-') p=-p;
27     p=p+100;
28
29     while(p%2==0)
30     {
31         d2++;
32         p=p/2;
33     }
34
35     while(p%5==0)
36     {
37         d5++;
38         p=p/5;
39     }
40
41     if(p!=1)
42     {
43         fout<<0;
44         fout.close();
45         return 0;
46     }
47
48     for(i=0;i<=18;i++)
49         if(v[i].c2==d2 && v[i].c5==d5)
50             break;
51
52     aux=v[i];
53     v[i]=v[0];
54     v[0]=aux;
55
56     a[0][v[0].c2][v[0].c5][0]=1;
57     for(h=1;h<=n-1;h++)
58     {
59         h1=h%2;
60         h2=1-h1;
61         for(i=0;i<=n2;i++)
62             for(j=0;j<=n2;j++)
63                 for(k=0;k<=18;k++)
64                 {
65                     hh=a[h2][i][j][k];
66                     if(hh!=0)
67                         for(l=k;l<=18;l++)
68                             if(i+v[1].c2<=n2 && j+v[1].c5<=n2)
69                             a[h1][i+v[1].c2][j+v[1].c5][l]+=hh;
70
71                     a[h2][i][j][k]=0;
72                 }
73     }
74
75     s=0;
76     for(i=0;i<=18;i++)
77         s=s+a[h1][n2][n2][i];
78

```

```

79     fout<<s<<"\n";
80
81     fout.close();
82     fin.close();
83     return 0;
84 }
```

Listing 23.4.4: 4_procente.cpp

```

1 // Vasile Ionel Pit-Rada
2 #include<iostream>
3 #include<iomanip>
4
5 using namespace std;
6
7 ifstream fin("procente.in");
8 ofstream fout("procente.out");
9
10 struct nod
11 {
12     short int c2,c5;
13     char k;
14 };
15
16 nod c[2][170*170*19];
17 int pr[2], ul[2];
18
19 struct dc
20 {
21     int c2,c5;
22 };
23
24 long long a[2][170][170][19], s, hh;
25 int n, p, d2=0, d5=0, n2, i, j, k, l, h, h1, h2;
26 char C;
27 dc aux, v[19]={{0,0},{1,0},{2,0},{0,1},{3,0},{1,1},{4,0},{2,1},{0,2},{5,0},
28                 {3,1},{1,2},{6,0},{4,1},{0,3},{7,0},{5,1},{3,2},{2,2}};
29
30 int main()
31 {
32     fin>>n>>C>>p;
33
34     n2=2*n;
35     if(C=='-') p=-p;
36     p=p+100;
37
38     while(p%2==0)
39     {
40         d2++;
41         p=p/2;
42     }
43
44     while(p%5==0)
45     {
46         d5++;
47         p=p/5;
48     }
49
50     if(p!=1)
51     {
52         fout<<0;
53         fout.close();
54         return 0;
55     }
56
57     for(i=0;i<=18;i++)
58         if(v[i].c2==d2 && v[i].c5==d5)
59             break;
60
61     aux=v[i];
62     v[i]=v[0];
63     v[0]=aux;
64
65     a[0][v[0].c2][v[0].c5][0]=1;
66     pr[0]=0;
```

```

67     ul[0]=0;
68     c[0][ul[0]].c2=v[0].c2;
69     c[0][ul[0]].c5=v[0].c5;
70     c[0][ul[0]].k=0;
71     ul[0]++;
72     for(h=1; h<=n-1; h++)
73     {
74         h1=h%2;
75         h2=1-h1;
76         pr[h1]=0;
77         ul[h1]=0;
78         while(pr[h2]<ul[h2])
79         {
80             i=c[h2][pr[h2]].c2;
81             j=c[h2][pr[h2]].c5;
82             k=c[h2][pr[h2]].k;
83             hh=a[h2][i][j][k];
84             for(l=k; l<=18; l++)
85                 if(i+v[l].c2<=n2 && j+v[l].c5<=n2)
86                 {
87                     if(a[h1][i+v[l].c2][j+v[l].c5][l]==0)
88                     {
89                         c[h1][ul[h1]].c2=i+v[l].c2;
90                         c[h1][ul[h1]].c5=j+v[l].c5;
91                         c[h1][ul[h1]].k=l;
92                         ul[h1]++;
93                     }
94                     a[h1][i+v[l].c2][j+v[l].c5][l]+=hh;
95                 }
96             a[h2][i][j][k]=0;
97             pr[h2]++;
98         }
99     }
100    }
101
102    s=0;
103    for(i=0; i<=18; i++)
104        s=s+a[h1][n2][n2][i];
105
106    fout<<s<<"\n";
107
108    fout.close();
109    fin.close();
110    return 0;
111 }
112 }
```

23.5 robotics

Problema 5 - robotics

100 de puncte

Ne aflăm în secția de vopsitorie a uzinei Toyota Motor unde inginerii japonezi prezintă ultimul tip de robot industrial de vopsire. În dorința de a evidenția calitatea și viteza de execuție a roboților, inginerii folosesc pentru demonstrație o tablă de dimensiunea $n \times n$, împărțită în pătrate cu latura egală cu 1, reprezentată sub forma unui tablou bidimensional cu n linii și n coloane.



Un robot utilizat pentru vopsire are două brațe telescopice care se deplasează de-a lungul unei axe. Fiecare braț poate vopsi într-o unitate de timp un singur pătrat. La momentul de timp $t = 0$ robotul primește comanda de a se poziționa într-un pătrat specificat prin coordonatele (x, y) .

În funcție de traiectoria de deplasare roboții folosesc sunt de două tipuri. La momentul de timp t robotul de tip 1 vopsește pătratele aflate la coordonatele: $(x-t, y+t)$ și $(x+t, y-t)$, iar robotul de tip 2 vopsește pătratele aflate la coordonatele: $(x+t, y+t)$ și $(x-t, y-t)$. Pentru vopsirea unui pătrat se consumă 1 litru de vopsea.

Pe tablă sunt așezăți m roboți.

Cerințe

Cunoscând pentru cei m roboți coordonatele inițiale (x_i, y_i) , $i = 1, \dots, m$, se cere să se determine:

- a) Cantitatea totală de vopsea care a fost folosită de roboți după t unități de timp
 b) Numărul minim de unități de timp necesare formării primului dreptunghi cu arie nenulă.
 Un dreptunghi corect format este rezultatul intersecției a două traiectorii paralele a doi roboți de tip 1 cu două traiectorii paralele a doi roboți de tip 2, iar colțurile dreptunghiului sunt 4 pătrate care au fost vopsite de doi roboți de tipuri diferite.

Date de intrare

Fișierul de intrare **robotics.in** conține pe prima linie trei valori naturale nenule n , m și t , cu semnificațiile din enunț, despărțite prin câte un singur spațiu.

Pe fiecare dintre următoarele m linii se află câte trei valori naturale nenule x_i , y_i și z_i , despărțite prin câte un spațiu, unde: x_i , y_i reprezintă coordonatele inițiale unde se poziționează robotul i , iar z_i reprezintă tipul robotului.

Date de ieșire

Fișierul de ieșire **robotics.out** va avea două linii.

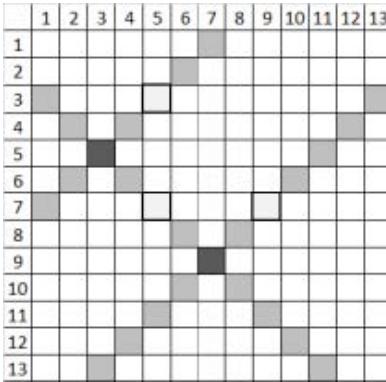
Prima linie conține un număr natural C nenul ce reprezintă cantitatea totală de vopsea care este folosită de roboți după t unități de timp.

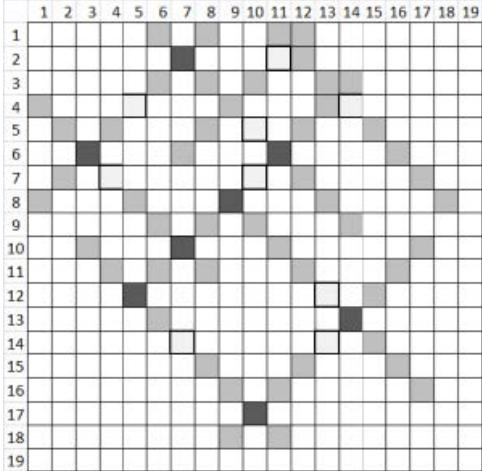
A doua linie conține un număr natural T_{min} ce reprezintă numărul minim de unități de timp necesare formării primului dreptunghi de arie nenulă.

Restricții și precizări

- $1 \leq t < n \leq 1\,000$
- $1 \leq m \leq 2 * n$
- $1 \leq x, y \leq n$
- Coordonatele celor m roboți sunt distincte două câte două.
- Doi roboți nu se pot afla pe aceeași traiectorie la un moment dat.
- La momentul $t = 0$ robotul se poziționează în pătratul specificat prin coordonatele (x, y) și vopsește o singură dată acest pătrat.
- Doi roboți de tipuri diferite care ajung în același timp pe un pătrat pot vopsi simultan pătratul.
 - Dacă brațul unui robot părăsește tabla dreptunghiulară, brațul își încetează activitatea.
 - Pentru rezolvarea corectă a primei cerințe se acordă 20 de puncte, iar pentru cerința a două se acordă 80 de puncte.

Exemple:

robotics.in	robotics.out	Explicații
13 3 6 3 5 1 7 5 2 7 9 1	29 0	 Cantitatea de vopsea care este folosită de roboți după t unități de timp este 29. Nu se pot forma dreptunghiuri.

19 9 4 4 5 1 4 14 2 2 11 1 14 7 2 5 10 2 14 13 1 7 4 2 7 10 1 12 13 2	75 3	 <p>Cantitatea de vopsea care este folosită de roboți după t unități de timp este 75.</p> <p>Sigurele dreptunghiuri corect formate după $t=4$ au colțurile în pătratele de coordonate:</p> <ul style="list-style-type: none"> (2,7),(6,11),(10,7),(6,3), respectiv (8,9),(13,14),(17,10),(12,5). <p>Observăm faptul că primul dreptunghi se formează după $t = 3$ (timpul minim)</p>
--	-------------	---

Timp maxim de executare/test: **0.1** secunde

Memorie: total **4 MB** din care pentru stivă **2 MB**

Dimensiune maximă a sursei: **10 KB**

23.5.1 Indicații de rezolvare

prof. Eugen Nodea, Colegiul "Tudor Vladimirescu" Tg. Jiu.

Varianta 1 - prof. Eugen Nodea - 100 p

Calculăm coordonatele capetelor segmentelor descrise de traectoria fiecărui robot după t unități de timp:

$(R[i].x1, R[i].y1)$, respectiv $(R[i].x2, R[i].y2)$, $1 \leq i \leq m$.

a) Numărul de pătrate vopsite de roboți după t unități de timp este egal cu:

$$\sum_{i=1}^m (|R[i].x1 - R[i].x2| + 1) = \sum_{i=1}^m (|R[i].y1 - R[i].y2| + 1)$$

b) Pentru determinarea numărului minim de unități de timp necesare formării primului dreptunghi vom folosi *căutarea binară*.

Pentru a determina cu ușurință unui dreptunghi corect format vom roti segmentele tuturor roboților astfel încât traectoriile generate să devină verticale pentru roboții de tip 1, respectiv orizontale pentru roboții de tip 2.

Puteam folosi transformare $(x, y) \rightarrow (x + y, x - y)$.

Dacă notăm cu n_1 numărul roboților de tip 1, iar cu n_2 numărul roboților de tip 2 ($n_1 + n_2 = m$) complexitatea algoritmului care verifică formarea unui dreptunghi este $O(n_1 * n_2)$.

Varianta 2 - prof. Vasile Ionel Pit-Rada - 55 p

Pentru cerința 2, se sortează roboții astfel:

Cei de tip 1 aflați pe poziții cu suma pară

Cei de tip 2 aflați pe poziții cu suma pară

Cei de tip 1 aflați pe poziții cu suma impară

Cei de tip 2 aflați pe poziții cu suma impară

Se parcurg perechile de roboți (i, j) de tip 1 aflați pe poziții cu suma pară și pentru fiecare pereche se parcurg roboții k de tip 2 aflați pe poziții de suma pară.

Pentru fiecare astfel de triplet de roboți se calculează cele două puncte de intersecție și timpul minim necesar pentru a ajunge cu vopsirea în ambele puncte. Pentru fiecare pereche (i, j) , temporar fixată, parcurgerea roboților k produce un sir de valori ale timpilor din care trebuie păstrate cele mai mici două valori $t_{min1} \leq t_{min2}$. La finalul parcurgerii celor k roboți, pentru perechea (i, j) se asociază timpul t_{min2} . Bineînțeles că dintre toate tripletele (i, j, k) se vor evita acelea pentru care unele din cele patru puncte de intersecție ies din suprafața de lucru.

Dintre toți timpii asociați perechilor (i, j) se va păstra cel mai mic.

Analog se procedează cu roboții de tip 1 și 2 aflați pe poziții cu suma impară.

Timpul cel mai mic va fi afișat.

Complexitate $O(n^3)$.

23.5.2 *Rezolvare detaliată

23.5.3 Cod sursă

Listing 23.5.1: 1_robots.cpp

```

1 // prof. Eugen Nodea
2 # include <fstream>
3 # include <cstring>
4
5 using namespace std;
6
7 ifstream f("robotics.in");
8 ofstream g("robotics.out");
9
10 struct patrat
11 {
12     short x, y;
13 } R1[2001], R2[2001];
14
15 struct traiectorie
16 {
17     short x1,x2,y1,y2;
18 } r1[2001], r2[2001];
19
20 int n, m, t, i, j, tip, cant, x, y, n1, n2;
21 bool a[1000][1000];
22 int nr[1001];
23
24 inline int abs(int a)
25 {
26     if (a < 0) return -a;
27     return a;
28 }
29
30 bool cmp1(patrat a, patrat b)
31 {
32     if (a.x == b.x) return (a.y < b.y);
33     return (a.x < b.x);
34 }
35
36 bool cmp2(patrat a, patrat b)
37 {
38     if (a.y == b.y) return (a.x < b.x);
39     return (a.y < b.y);
40 }
41
42 bool verif(int t)
43 {
44     int i, j, k, l, c, x, y, Max = 0, d1, d2, x1, x2, y1, y2;
45
46     for(i=1; i<=n1; ++i)
47     {
48         x = R1[i].x;
49         y = R1[i].y;
50         d1 = abs(x - r1[i].x1) + abs(y - r1[i].y1);
51         d2 = abs(x - r1[i].x2) + abs(y - r1[i].y2);
52         if (d1 > d2) swap(d1, d2);
53         if (d1 > t) continue;
54         if (d2 > t) continue;
55         if (a[x][y] > 0) continue;
56         if (a[x][y] < 0) a[x][y] = -1;
57         else a[x][y] = 1;
58         if (d1 < Max) Max = d1;
59         if (d2 < Max) Max = d2;
60     }
61
62     for(j=1; j<=n2; ++j)
63     {
64         x = R2[j].x;
65         y = R2[j].y;
66         d1 = abs(x - r2[j].x1) + abs(y - r2[j].y1);
67         d2 = abs(x - r2[j].x2) + abs(y - r2[j].y2);
68         if (d1 > d2) swap(d1, d2);
69         if (d1 > t) continue;
70         if (d2 > t) continue;
71         if (a[x][y] > 0) continue;
72         if (a[x][y] < 0) a[x][y] = -1;
73         else a[x][y] = 1;
74         if (d1 < Max) Max = d1;
75         if (d2 < Max) Max = d2;
76     }
77
78     for(k=1; k<=m; ++k)
79     {
80         x = R1[k].x;
81         y = R1[k].y;
82         d1 = abs(x - r1[k].x1) + abs(y - r1[k].y1);
83         d2 = abs(x - r1[k].x2) + abs(y - r1[k].y2);
84         if (d1 > d2) swap(d1, d2);
85         if (d1 > t) continue;
86         if (d2 > t) continue;
87         if (a[x][y] > 0) continue;
88         if (a[x][y] < 0) a[x][y] = -1;
89         else a[x][y] = 1;
90         if (d1 < Max) Max = d1;
91         if (d2 < Max) Max = d2;
92     }
93
94     for(l=1; l<=n; ++l)
95     {
96         x = R2[l].x;
97         y = R2[l].y;
98         d1 = abs(x - r2[l].x1) + abs(y - r2[l].y1);
99         d2 = abs(x - r2[l].x2) + abs(y - r2[l].y2);
100        if (d1 > d2) swap(d1, d2);
101        if (d1 > t) continue;
102        if (d2 > t) continue;
103        if (a[x][y] > 0) continue;
104        if (a[x][y] < 0) a[x][y] = -1;
105        else a[x][y] = 1;
106        if (d1 < Max) Max = d1;
107        if (d2 < Max) Max = d2;
108    }
109
110    if (nr[t] < 0) nr[t] = Max;
111    else if (nr[t] > Max) nr[t] = Max;
112    else if (nr[t] < Max) nr[t] = -1;
113 }
114
115 void write()
116 {
117     for(i=1; i<=t; ++i)
118     {
119         if (nr[i] < 0) g << " ";
120         else g << nr[i];
121     }
122 }
123
124 int main()
125 {
126     f >> n >> m >> t;
127     for(i=1; i<=n1; ++i)
128     {
129         R1[i].x = f.get();
130         R1[i].y = f.get();
131     }
132     for(j=1; j<=n2; ++j)
133     {
134         R2[j].x = f.get();
135         R2[j].y = f.get();
136     }
137     for(k=1; k<=m; ++k)
138     {
139         r1[k].x1 = f.get();
140         r1[k].x2 = f.get();
141         r1[k].y1 = f.get();
142         r1[k].y2 = f.get();
143     }
144     for(l=1; l<=n; ++l)
145     {
146         r2[l].x1 = f.get();
147         r2[l].x2 = f.get();
148         r2[l].y1 = f.get();
149         r2[l].y2 = f.get();
150     }
151     write();
152 }
```

```

49         y = R1[i].y;
50         d1 = (x - t > 0) ? 0 : abs(x - t) + 1;
51         d2 = (y + t <= n)? 0 : (y + t) - n;
52
53         Max = max(d1, d2);
54         x1 = x - (t - Max); y1 = y + (t - Max);
55         d1 = (x + t <= n)? 0 : (x + t) - n;
56         d2 = (y - t > 0) ? 0 : abs(y - t) + 1;
57
58         Max = max(d1, d2);
59         x2 = x + (t - Max);
60         y2 = y - (t - Max);
61         r1[i].x1 = x1 + y1;
62         r1[i].y1 = x1 - y1;
63         r1[i].x2 = x2 + y2;
64         r1[i].y2 = x2 - y2;
65     }
66
67     for(i=1; i<=n2; ++i)
68     {
69         x = R2[i].x;
70         y = R2[i].y;
71         d1 = (x + t <= n)? 0 : (x + t) - n;
72         d2 = (y + t <= n)? 0 : (y + t) - n;
73
74         Max = max(d1, d2);
75         x2 = x + (t - Max);
76         y2 = y + (t - Max);
77         d1 = (x - t > 0) ? 0 : abs(x - t) + 1;
78         d2 = (y - t > 0) ? 0 : abs(y - t) + 1;
79
80         Max = max(d1, d2);
81         x1 = x - (t - Max);
82         y1 = y - (t - Max);
83         r2[i].x1 = x1 + y1;
84         r2[i].y1 = x1 - y1;
85         r2[i].x2 = x2 + y2;
86         r2[i].y2 = x2 - y2;
87     }
88
89     memset( a, 0, sizeof(a));
90     memset(nr, 0, sizeof(nr));
91
92     for(i=1; i<=n1; i++)
93     {
94         k = 0; // primul tip de intersectie
95         if ((R1[i].x + R1[i].y) % 2 == 0)
96             for(j=1; j<=n2; j++)
97             {
98                 if ((R2[j].x + R2[j].y) % 2 == 0)
99                     if (r2[j].x1 <= r1[i].x1 && r1[i].x1 <= r2[j].x2)
100                         if (r1[i].y1 <= r2[j].y1 && r2[j].y1 <= r1[i].y2)
101                             nr[+k] = j;
102             }
103         else
104             for(j=1; j<=n2; j++) // al doilea tip de intersectie
105             {
106                 if ((R2[j].x + R2[j].y) % 2 == 1)
107                     if (r2[j].x1 <= r1[i].x1 && r1[i].x1 <= r2[j].x2)
108                         if (r1[i].y1 <= r2[j].y1 && r2[j].y1 <= r1[i].y2)
109                             nr[+k] = j;
110             }
111
112         for(l=1; l<k; ++l)
113             for(c=l+1; c<=k; ++c)
114                 if (a[nr[l]][nr[c]])
115                     return 1;
116                 else
117                     a[nr[l]][nr[c]] = a[nr[c]][nr[l]] = 1;
118     }
119
120     return 0;
121 }
122
123 int main()
124 {

```

```

125     f >> n >> m >> t;
126
127     for(i=1; i<=m; ++i)
128     {
129         f >> x >> y >> tip;
130         ++cant;
131         if (tip == 1)
132         {
133             cant += min(t, min(x-1, n-y));
134             cant += min(t, min(n-x, y-1));
135             R1[++n1].x = x;
136             R1[n1].y = y;
137         }
138         else
139         {
140             cant += min(t, min(x-1, y-1));
141             cant += min(t, min(n-x, n-y));
142             R2[++n2].x = x;
143             R2[n2].y = y;
144         }
145     }
146     g << cant << "\n";
147
148 //b)
149 i = 1;
150 j = n;
151 while (i <= j)
152 {
153     t = (i+j) / 2;
154     if (verif(t))
155         j = t - 1;
156     else
157         i = t + 1;
158 }
159 if ( i <= n)
160     g << i << "\n";
161 else
162     g << "0\n";
163
164 return 0;
165 }
```

Listing 23.5.2: 2_robots.cpp

```

1 // prof. Vasile Ionel Pit-Rada
2 #include<iostream>
3 #define shint short int
4
5 using namespace std;
6
7 ifstream fin("robotics.in");
8 ofstream fout("robotics.out");
9
10 struct pozitie
11 {
12     shint x,y;
13 };
14
15 shint n,m,t,tmin;
16 pozitie ril[2001],ri2[2001],rp1[2001],rp2[2001];
17 shint ni1,ni2,np1,np2;
18
19 int tot;
20
21 void sortare(pozitie v[], shint p, shint q, shint k)
22 {
23     shint i,j;
24     pozitie aux;
25     for(i=p;i<q;i++)
26     {
27         for(j=i+1;j<=q;j++)
28         {
29             if(v[i].x+k*v[i].y>v[j].x+k*v[j].y)
30             {
31                 aux=v[i];

```

```

32             v[i]=v[j];
33             v[j]=aux;
34         }
35     }
36 }
38
39 shint minim(shint a, shint b)
40 {
41     if(a<b) return a;
42     return b;
43 }
44
45 shint min3(shint a, shint b, shint c)
46 {
47     if(a<=b && a<=c) return a;
48     if(b<=a && b<=c) return b;
49     return c;
50 }
51
52 shint maxim(shint a, shint b)
53 {
54     if(a>b) return a;
55     return b;
56 }
57
58 int main()
59 {
60     shint i,u,v,w,ul,v1,j,k;
61     shint di,dj,dk,tiu,tju1,tku,tku1,t1,tij1,tij2;
62     fin>>n>>m>>t;
63     tot=0;
64
65     ni1=0;
66     ni2=0;
67     np1=0;
68     np2=0;
69     for(i=1;i<=m; i++)
70     {
71         fin>>u>>v>>w;
72         if(w==2)
73             tot=tot+min3(u-1,v-1, t)+min3(n-u, n-v, t);
74         else
75             tot=tot+min3(u-1,n-v, t)+min3(n-u,v-1, t);
76
77         if((u+v)%2==0)
78         {
79             if(w==2)
80             {
81                 np2++;
82                 rp2[np2].x=u;
83                 rp2[np2].y=v;
84             }
85             else
86             {
87                 np1++;
88                 rp1[np1].x=u;
89                 rp1[np1].y=v;
90             }
91         }
92         else
93         {
94             if(w==2)
95             {
96                 ni2++;
97                 ri2[ni2].x=u;
98                 ri2[ni2].y=v;
99             }
100            else
101            {
102                ni1++;
103                ril[ni1].x=u;
104                ril[ni1].y=v;
105            }
106        }
107    }

```

```

108
109     //ordonez cele de tip 1
110     sortare(ril,1,nil,1);
111     sortare(rp1,1,np1,1);
112
113     //ordonez cele de tip 2
114     sortare(ri2,1,ni2,-1);
115     sortare(rp2,1,np2,-1);
116
117     tmin=1001;
118     for(i=1;i<=np1-1;i++)
119     {
120         di=rp1[i].x+rp1[i].y;
121         for(j=i+1;j<=np1;j++)
122         {
123             dj=rp1[j].x+rp1[j].y;
124             tij1=1001;
125             tij2=1001;
126             for(k=1;k<=np2;k++)
127             {
128                 dk=rp2[k].x-rp2[k].y;
129                 u=(di+dk)/2;
130                 v=(di-dk)/2;
131                 u1=(dj+dk)/2;
132                 v1=(dj-dk)/2;
133                 if(1<=u && u<=n && 1<=v && v<=n &&
134                     1<=u1 && u1<=n && 1<=v1 && v1<=n)
135                 {
136                     tiu=u-rp1[i].x;
137                     if(tiu<0)tiu=-tiu;
138                     tju1=u1-rp1[j].x;
139                     if(tju1<0)tju1=-tju1;
140                     tku=u-rp2[k].x;
141                     if(tku<0)tku=-tku;
142                     tkul=u1-rp2[k].x;
143                     if(tkul<0)tkul=-tkul;
144                     t1=maxim(tiu,tju1);
145                     t1=maxim(t1,tku);
146                     t1=maxim(t1,tkul);
147                     if(t1<=tij1)
148                     {
149                         tij2=tij1;
150                         tij1=t1;
151                     }
152                     else
153                     {
154                         if(t1<tij2)
155                             tij2=t1;
156                     }
157                 }
158             }
159             if(tij2<tmin)
160                 tmin=tij2;
161         }
162     }
163 }
164
165     for(i=1;i<=nil-1;i++)
166     {
167         di=ril[i].x+ril[i].y;
168         for(j=i+1;j<=nil;j++)
169         {
170             dj=ril[j].x+ril[j].y;
171             tij1=1001;
172             tij2=1001;
173             for(k=1;k<=ni2;k++)
174             {
175                 dk=ri2[k].x-ri2[k].y;
176                 u=(di+dk)/2;
177                 v=(di-dk)/2;
178                 u1=(dj+dk)/2;
179                 v1=(dj-dk)/2;
180                 if(1<=u && u<=n && 1<=v && v<=n &&
181                     1<=u1 && u1<=n && 1<=v1 && v1<=n)
182                 {
183                     tiu=u-ril[i].x;

```

```

184             if(tiu<0)tiu=-tiu;
185             t jul=u1-ri1[j].x;
186             if(t jul<0)t jul=-t jul;
187             t ku=u-ri2[k].x;
188             if(t ku<0)tku=-tku;
189             tkul=u1-ri2[k].x;
190             if(tkul<0)tkul=-tkul;
191             t1=maxim(tiu,t jul);
192             t1=maxim(t1,tku);
193             t1=maxim(t1,tkul);
194             if(t1<=tij1)
195             {
196                 tij2=tij1;
197                 tij1=t1;
198             }
199             else
200                 if(t1<tij2)
201                     tij2=t1;
202             }
203         }
204     }
205     if(tij2<tmin)
206         tmin=tij2;
207     }
208 }
209
210 fout<<tot+m<<"\n"<<tmin<<"\n";
211 fout.close();
212 fin.close();
213 return 0;
214 }
```

Listing 23.5.3: 3_robots.cpp

```

1 // prof. Eugen Nodea
2 # include <cstdio>
3 # include <algorithm>
4 # include <cstring>
5
6 using namespace std;
7
8 FILE *f = fopen("robotics.in", "r");
9 FILE *g = fopen("robotics.out", "w");
10
11 struct patrat
12 {
13     short x, y;
14 } R1[2001], R2[2001];
15
16 struct traiectorie
17 {
18     short x1,x2,y1,y2;
19 } r1[2001], r2[2001];
20
21 int n, m, t, i, j, tip, cant, x, y, n1, n2;
22 bool a[1000][1000];
23 int nr[1001];
24
25 inline int abs(int a)
26 {
27     if (a < 0) return -a;
28     return a;
29 }
30
31 bool cmp1(patrat a, patrat b)
32 {
33     if (a.x == b.x) return (a.y < b.y);
34     return (a.x < b.x);
35 }
36
37 bool cmp2(patrat a, patrat b)
38 {
39     if (a.y == b.y) return (a.x < b.x);
40     return (a.y < b.y);
41 }
```

```

42
43 bool verif(int t)
44 {
45     int i, j, k, l, c, x, y, Max = 0, d1, d2, x1, x2, y1, y2;
46
47     for(i=1; i<=n1; ++i)
48     {
49         // calculam capetele segmentelor
50         x = R1[i].x;
51         y = R1[i].y;
52         d1 = (x - t > 0) ? 0 : abs(x - t) + 1;
53         d2 = (y + t <= n)? 0 : (y + t) - n;
54
55         Max = max(d1, d2);
56         x1 = x - (t - Max); y1 = y + (t - Max);
57         d1 = (x + t <= n)? 0 : (x + t) - n;
58         d2 = (y - t > 0) ? 0 : abs(y - t) + 1;
59
60         Max = max(d1, d2);
61         x2 = x + (t - Max);
62         y2 = y - (t - Max);
63
64         //rotim cu 45 grade segmentul
65         r1[i].x1 = x1 + y1;
66         r1[i].y1 = x1 - y1;
67         r1[i].x2 = x2 + y2;
68         r1[i].y2 = x2 - y2;
69     }
70
71     for(i=1; i<=n2; ++i)
72     {
73         x = R2[i].x; y = R2[i].y;
74         d1 = (x + t <= n)? 0 : (x + t) - n;
75         d2 = (y + t <= n)? 0 : (y + t) - n;
76
77         Max = max(d1, d2);
78         x2 = x + (t - Max);
79         y2 = y + (t - Max);
80         d1 = (x - t > 0) ? 0 : abs(x - t) + 1;
81         d2 = (y - t > 0) ? 0 : abs(y - t) + 1;
82
83         Max = max(d1, d2);
84         x1 = x - (t - Max);
85         y1 = y - (t - Max);
86         r2[i].x1 = x1 + y1;
87         r2[i].y1 = x1 - y1;
88         r2[i].x2 = x2 + y2;
89         r2[i].y2 = x2 - y2;
90     }
91
92     memset( a, 0, sizeof(a));
93     memset(nr, 0, sizeof(nr));
94
95     for(i=1; i<=n1; i++)
96     {
97         k = 0; // primul tip de intersectie
98         if ((R1[i].x + R1[i].y) % 2 == 0)
99             for(j=1; j<=n2; j++)
100             {
101                 if ((R2[j].x + R2[j].y) % 2 == 0)
102                     if (r2[j].x1 <= r1[i].x1 && r1[i].x1 <= r2[j].x2)
103                         if (r1[i].y1 <= r2[j].y1 && r2[j].y1 <= r1[i].y2)
104                             nr[++k] = j;
105             }
106         else
107             for(j=1; j<=n2; j++) // al doilea tip de intersectie
108             {
109                 if ((R2[j].x + R2[j].y) % 2 == 1)
110                     if (r2[j].x1 <= r1[i].x1 && r1[i].x1 <= r2[j].x2)
111                         if (r1[i].y1 <= r2[j].y1 && r2[j].y1 <= r1[i].y2)
112                             nr[++k] = j;
113             }
114
115         for(l=1; l<k; ++l)
116             for(c=l+1; c<=k; ++c)
117                 if (a[nr[l]][nr[c]]) return 1;

```

```

118         else a[nr[1]][nr[c]] = a[nr[c]][nr[1]] = 1;
119     }
120
121     return 0;
122 }
123
124 int main()
125 {
126     fscanf(f, "%hd%hd%hd", &n, &m, &t);
127     for(i=1; i<=m; ++i)
128     {
129         fscanf(f, "%hd%hd%hd", &x, &y, &tip);
130         ++cant;
131         if (tip == 1)
132         {
133             cant += min(t, min(x-1, n-y));
134             cant += min(t, min(n-x, y-1));
135             R1[++n1].x = x; R1[n1].y = y;
136         }
137         else
138         {
139             cant += min(t, min(x-1, y-1));
140             cant += min(t, min(n-x, n-y));
141             R2[++n2].x = x; R2[n2].y = y;
142         }
143     }
144
145     fprintf(g, "%d\n", cant);
146
147 //b)
148     i = 1;
149     j = n;
150     while (i <= j)
151     {
152         t = (i+j) / 2;
153         if (verif(t))
154             j = t - 1;
155         else
156             i = t + 1;
157     }
158
159     if ( i <= n)
160         fprintf(g, "%d\n", i);
161     else
162         fprintf(g, "0\n");
163
164     return 0;
165 }
```

23.6 sablon

Problema 6 - sablon

100 de puncte

Se consideră alfabetul englez compus din literele mici, de la a la z.

Se numește **cuvânt** un sir finit, eventual vid, de litere din acest alfabet.

Se numește **expresie sablon** un sir de caractere din alfabet în care pot apărea și caracterele ? și *.

Un cuvânt se potrivește cu o expresie sablon dacă se poate obține din aceasta astfel:

- caracterul ? se înlocuiește cu o singură literă din alfabet;
- caracterul * se înlocuiește cu un cuvânt oarecare, eventual vid;
- din expresia sablon se poate elimina sau nu, înainte de a efectua înlocuirea caracterelor ?, si *, un singur caracter de tip literă.

Cerințe

Considerându-se o expresie sablon și un sir de cuvinte, să se determine, pentru fiecare cuvânt în parte, dacă se potrivește sau nu cu expresia sablon dată.

Date de intrare

Fișierul de intrare **sablon.in** conține:

Pe prima linie se găsește o expresie şablon E .

Pe a doua linie se găsește un număr natural N , ce reprezintă numărul de cuvinte din sir.

Pe fiecare din următoarele N linii se găsește câte un cuvânt S_i ($1 \leq i \leq N$).

Date de ieșire

Fișierul de ieșire **sablon.out** va conține pe fiecare din primele N linii valoarea 1 sau 0, după cum cuvântul S_i ($1 \leq i \leq N$) se potrivește cu expresia şablon E .

Restricții și precizări

- $1 \leq N \leq 10$
- Pentru 16% din teste expresia şablon E nu conține caracterul * iar lungimea lui E și a oricărui cuvânt S este între 1 și 1 000 de caractere.
 - Pentru alte 16% din teste lungimile lui E și S sunt cuprinse între 1 și 20 de caractere.
 - Pentru alte 32% din teste lungimile lui E și S sunt cuprinse între 1 și 100 de caractere.
 - Pentru restul de 36% din teste lungimile lui E și S sunt cuprinse între 1 și 1 500 de caractere.

Exemple:

sablon.in	sablon.out	Explicații
a*a?b	7	Cuvintele <i>ababb</i> , <i>aabb</i> și <i>abcaab</i> se potrivesc cu expresia şablon.
ababb	aab	Cuvântul <i>aab</i> se potrivesc cu expresia şablon obținută prin eliminarea în prealabil a unuia din caracterele 'a'.
aabb	bac	Cuvintele <i>bac</i> și <i>ababa</i> nu se pot obține în nicio situație din expresia şablon.
caab	abcaab	
ababa		

Timp maxim de executare/test: **1.2** secunde pe Windows, **0.7** secunde pe Linux

Memorie: total **8 MB** din care pentru stivă **2 MB**

Dimensiune maximă a sursei: **5 KB**

23.6.1 Indicații de rezolvare

prof. Radu Vișinescu, C.N. "A.I.J.I.L. Caragiale" Ploiești

Soluție 32 puncte (prof. Radu Vișinescu):

Se programează o funcție recursivă care decide dacă un sir de caractere S se potrivesc cu o expresie şablon E . Cazurile se separă în funcție de primul caracter al expresiei E respectiv al sirului S . Există și un caz în care se execută mai multe apeluri recursive și anume la întâlnirea caracterului * în expresia şablon E . Funcția se apelează din programul principal pentru fiecare valoare din sirul de cuvinte.

Soluție 48 puncte (prof. Daniel Popa):

Se transformă şablonul într-o *expresie regulată* care este folosită pentru căutare folosind libraria *regex*.

Avantaj: Cod scurt.

Dezavantaj: funcționează doar sub Linux, se consumă multă memorie, algoritm implementat în librărie nu este foarte performant.

Soluție 64 puncte (student - Mihai Nițu):

Pentru a verifica dacă un cuvânt se potrivesc cu un şablon dat, se poate utiliza un *algoritm greedy* sau o *relație de recurență*.

Algoritmul greedy constă în verificarea dacă capetele şablonului (prefixul și sufixul până la '*') coincide cu capetele cuvântului, și apoi căutarea pe rând a fiecărei subsecvențe din şablon cuprinsă între '**' în cuvânt. Dacă acestea apar ca subsecvențe în cuvânt, în ordinea în care apar și în şablon și nu se intersectează cu bucățile de care am stabilit că vor corespunde capetelor şablonului, atunci cuvântul se potrivesc cu şablonul.

Soluția alternativă este data de următoarea relație de recurență:

$Match[i][j] = 1$ dacă sufixul $[j..m]$ din c se potrivesc cu sufixul $[i..n]$ din şablonul s .

$Match[n+1][m+1] = 1$
 $Match[i][j] = Match[i+1][j+1]$ dacă $s[i] == '?'$ sau $s[i]$ - literă și $s[i] == c[j]$
 $= 0$, dacă $s[i]$ - literă și $s[i] != c[j]$
 $= Match[i+1][j] \parallel Match[i+1][j+1] \parallel \dots Match[i+1][m+1]$, dacă
 $s[i] == '*'$, unde \parallel este 'sau' logic. \rightarrow se pot ține 'sau'-uri parțiale pe fiecare linie a matricei $Match$ pentru a obține valoarea în $O(1)$.

Pentru a ține cont de posibilitatea eliminării unei litere, putem să aplicăm algoritmul, pentru şablonurile obținute prin eliminarea fiecărei litere. Complexitatea finală va fi $O(nr.Cuvinte * N^3)$ cu ambele abordări.

Pentru algoritmul greedy, nu putem folosi *KMP* din cauza caracterelor ?, deci vom folosi algoritmul de potrivire brut. Această soluție poate fi dificil de implementat dar poate obține chiar 100 puncte deoarece *algoritmul brut de potrivire* se comportă foarte bine în practică.

Soluție 100 puncte (student - Mihai Nițu):

Putem să îmbunătățim, spre exemplu, soluția prin recurență, pentru a nu mai fi nevoie să o recalculem de fiecare dată când eliminăm o literă. Astfel, vom calcula aceeași recurență de 2 ori, *Match1* și *Match2*. Diferența dintre ele este că *Match1* va verifica potrivirea între un sufix al şablonului și un sufix al cuvântului iar *Match2* va verifica potrivirea între un prefix al şablonului și un prefix al cuvântului.

Potem, apoi, să verificăm pentru fiecare poziție din şablon, pe care există o literă, dacă $ok2[i-1][j] = 1$ și $ok1[i+1][j+1] = 1$, pentru oricare j de la 0 la m . Dacă cel puțin una se evaluează la 1, atunci avem o potrivire.

Complexitate: $O(nr.Cuvinte * N^2)$.

23.6.2 *Rezolvare detaliată

23.6.3 Cod sursă

Listing 23.6.1: 1_sablon.cpp

```

1 // student Mihai Nitu - Universitatea Bucuresti
2 #include <iostream>
3 #include <fstream>
4 #include <cstring>
5
6 #define maxn 1510
7
8 using namespace std;
9
10 ifstream fin("sablon.in");
11 ofstream fout("sablon.out");
12
13 bool ok1[maxn][maxn], ok2[maxn][maxn], part[maxn][maxn];
14 int n,m;
15
16 char s[maxn], c[maxn];
17
18 void gol (int n, int m, bool dp[][][maxn])
19 {
20     memset(part, 0, sizeof(part));
21
22     dp[n+1][m+1] = 1;
23     part[n+1][m+1] = 1;
24
25     for (int i = 1; i <= m; ++i)
26         part[n+1][i] = 1;
27
28     for (int i = n; i >= 1 && s[i] == '*' ; --i)
29     {
30         dp[i][m+1] = 1;
31         part[i][m+1] = 1;
32     }
33
34     for (int i = n; i >= 1; --i)

```

```

35         for (int j = m; j >= 1; --j)
36     {
37         if (s[i] == c[j])
38             dp[i][j] = dp[i+1][j+1];
39         else
40             if (s[i] == '?')
41                 dp[i][j] = dp[i+1][j+1];
42             else
43                 if (s[i] == '*')
44                     dp[i][j] = part[i+1][j];
45             else
46                 dp[i][j] = 0;
47
48         part[i][j] = part[i][j+1] || dp[i][j];
49     }
50 }
51
52 void go2 (int n, int m, bool dp[] [maxn])
53 {
54     memset(part, 0, sizeof(part));
55
56     dp[0][0] = 1;
57     part[0][0] = 1;
58
59     for (int i = 1; i <= m; ++i)
60         part[0][i] = 1;
61
62     for (int i = 1; i <= n && s[i] == '*'; ++i)
63     {
64         dp[i][0] = 1;
65         part[i][0] = 1;
66     }
67
68     for (int i = 1; i <= n; ++i)
69         for (int j = 1; j <= m; ++j)
70         {
71             if (s[i] == c[j])
72                 dp[i][j] = dp[i-1][j-1];
73             else
74                 if (s[i] == '?')
75                     dp[i][j] = dp[i-1][j-1];
76                 else
77                     if (s[i] == '*')
78                         dp[i][j] = part[i-1][j];
79                 else
80                     dp[i][j] = 0;
81
82             part[i][j] = part[i][j-1] || dp[i][j];
83         }
84     }
85
86 string getString(int x)
87 {
88     if (x == 0)
89         return "0";
90
91     string s;
92
93     while (x != 0)
94     {
95         char c = x%10 + '0';
96         s = c + s;
97         x /= 10;
98     }
99
100    return s;
101 }
102
103 int main()
104 {
105     fin >> s+1;
106     fin >> n;
107
108     for (int i = 1; i <= n; ++i)
109     {
110         fin >> c+1;

```

```

111     int n = strlen(s+1);
112     int m = strlen(c+1);
113
114     memset(ok1, 0, sizeof(ok1));
115     memset(ok2, 0, sizeof(ok2));
116
117     go1(n,m,ok1);
118     go2(n,m,ok2);
119
120     bool ok = ok1[1][1];
121
122     for (int i = 1; i <= n; ++i)
123         if ('a' <= s[i] && s[i] <= 'z')
124             for (int j = 0; j <= m; ++j)
125                 if (ok2[i-1][j] && ok1[i+1][j+1])
126                     ok = 1;
127
128     fout << ok << "\n";
129 }
130 }
```

Listing 23.6.2: 2_sablon.cpp

```

1 // student Mihai Nitu - Universitatea Bucuresti
2 // backtracking
3 #include <iostream>
4 #include <iostream>
5 #include <cstring>
6
7 #define maxn 1510
8
9 using namespace std;
10
11 ifstream fin("sablon.in");
12 ofstream fout("sablon.out");
13
14 bool ok1[maxn][maxn], ok2[maxn][maxn], part[maxn][maxn];
15 int n,m;
16
17 char s[maxn], c[maxn];
18 int N,M;
19
20 bool back (int i, int j, bool can)
21 {
22     if (i == N+1)
23     {
24         if (j == M+1)
25             return 1;
26         return 0;
27     }
28
29     if (j == M+1)
30     {
31         if (s[i] == '*')
32             return back(i+1, j, can);
33         else if (can)
34             return back(i+1, j, 0);
35         else return 0;
36     }
37
38     if (s[i] == '?')
39         return back(i+1, j+1, can);
40     else if (s[i] == '*')
41     {
42         for (int k = j; k <= M+1; ++k)
43         {
44             if(back(i+1,k,can))
45                 return 1;
46         }
47
48         return 0;
49     }
50     else
51     {
```

```

52         if (s[i] == c[j])
53     {
54         return back(i+1, j+1, can);
55     }
56     else
57     {
58         if (can)
59             return back(i+1, j, 0);
60         return 0;
61     }
62 }
63 }
64
65 int main()
66 {
67     fin >> s+1;
68     fin >> n;
69     for (int i = 1; i <= n; ++i)
70     {
71         fin >> c+1;
72
73         N = strlen(s+1);
74         M = strlen(c+1);
75
76         fout << back(1,1,1) << "\n";
77     }
78 }
```

Listing 23.6.3: 3_sablon.cpp

```

1 // student Mihai Nitu - Universitatea Bucuresti
2 // O(N3)
3 #include <fstream>
4 #include <iostream>
5 #include <cstring>
6
7 #define maxn 1510
8
9 using namespace std;
10
11 ifstream fin("sablon.in");
12 ofstream fout("sablon.out");
13
14 bool ok1[maxn][maxn], ok2[maxn][maxn], part[maxn][maxn];
15 int n,m;
16
17 char s[maxn], c[maxn], ss[maxn];
18
19 void go1 (int n, int m, bool dp[][][maxn ])
20 {
21     memset(part, 0, sizeof(part));
22
23     dp[n+1][m+1] = 1;
24     part[n+1][m+1] = 1;
25
26     for (int i = 1; i <= m; ++i)
27     {
28         part[n+1][i] = 1;
29     }
30     for (int i = n; i >= 1 && s[i] == '*' ; --i)
31     {
32         dp[i][m+1] = 1;
33         part[i][m+1] = 1;
34     }
35
36     for (int i = n; i >= 1; --i)
37     {
38         for (int j = m; j >= 1; --j)
39         {
40             if (s[i] == c[j])
41                 dp[i][j] = dp[i+1][j+1];
42             else
43                 if (s[i] == '?')
44                     dp[i][j] = dp[i+1][j+1];
45             else
```

```

46             if (s[i] == '*')
47                 dp[i][j] = part[i+1][j];
48             else
49                 dp[i][j] = 0;
50
51         part[i][j] = part[i][j+1] || dp[i][j];
52     }
53 }
54 }
55
56 int main()
57 {
58     fin >> s+1;
59     fin >> n;
60     for (int i = 1; i <= n; ++i)
61     {
62         fin >> c+1;
63
64         int n = strlen(s+1);
65         int m = strlen(c+1);
66
67         bool ok = 0;
68
69         for (int k = 1; k <= n; ++k)
70         {
71             if ('a' > s[k] || s[k] > 'z')
72                 continue;
73
74             for (int j = 1; j <= n; ++j)
75                 ss[j] = s[j];
76
77             for (int j = k; j < n; ++j)
78                 s[j] = s[j+1];
79
80             n = n-1;
81             s[n+1] = '\0';
82
83             memset(ok1, 0, sizeof(ok1));
84
85             go1(n,m,ok1);
86
87             ok = ok1[1][1];
88
89             n = n+1;
90             for (int j = 1; j <= n; ++j)
91                 s[j] = ss[j];
92
93             ok = ok1[1][1];
94             if (ok)
95                 break;
96         }
97
98         fout << ok << "\n";
99     }
100 }
```

Listing 23.6.4: 6_sablon.cpp

```

1 // prof. Vasile Ionel Pit-Rada
2 #include<iostream>
3 #include<cstring>
4
5 using namespace std;
6
7 ifstream fin("sablon.in");
8 ofstream fout("sablon.out");
9
10 int N,aL,sL,nrx;
11 char a[1509],sablon[1509];
12 int i,j,k;
13
14 int solve(int nrx, int nrd, int a1, int a2, int s1, int s2)
15 {
16     int i,j,r,ok;
17     if(nrd==1 && a2-a1+1<s2-s1+1-nrx) return 0;
```

```

18     if(nrd==0 && a2-a1+1<s2-s1-nrx) return 0;
19     if(a1>a2 && s1>s2) return 1;
20     if(s1==s2 && sablon[s1]=='*') return 1;
21     if(s1==s2 && sablon[s1]=='?' && a1==a2) return 1;
22
23     if(s1<=s2&&a1<=a2&&sablon[s1]!='*' &&
24         sablon[s1]!='?'&&sablon[s1]!=a[a1]&&nrd==1)
25     {
26         return 0;
27     }
28
29     if(sablon[s1]!='*' && sablon[s1]!='?'&&sablon[s1]!=a[a1]&&nrd==0)
30     {
31         return solve(nrx,1,a1,a2,s1+1,s2);
32     }
33
34     if(s1<=s2&&a1<=a2&&sablon[s1]==a[a1])
35     {
36         return solve(nrx,nrd,a1+1,a2,s1+1,s2);
37     }
38
39     if(s1<=s2&&a1<=a2&&sablon[s1]== '?')
40     {
41         return solve(nrx,nrd,a1+1,a2,s1+1,s2);
42     }
43
44     if(sablon[s1]=='*')
45     {
46         ok=0;
47         for(j=a1;j+s2-s1-nrx-1-nrd<=a2;j++)
48         {
49             r=solve(nrx-1,nrd,j,a2,s1+1,s2);
50             if(r==1)
51             {
52                 ok=1;
53                 break;
54             }
55         }
56         return ok;
57     }
58     return 1;
59 }
60
61 int main()
62 {
63     fin>>sablon;
64     i=0;
65     j=0;
66     nrx=0;
67     while(sablon[i] !=0)
68     {
69         if(sablon[i]!='*')
70         {
71             sablon[j]=sablon[i];
72             j++;
73             i++;
74         }
75         else
76         {
77             if(i>0 && sablon[i-1]=='*')
78                 i++;
79             else
80             {
81                 sablon[j]=sablon[i];
82                 i++;
83                 j++;
84                 nrx++;
85             }
86         }
87     }
88
89     sablon[j]=0;
90     sL=j;
91     fin>>N;
92     for(k=1;k<=N;k++)
93     {

```

```
94         fin>>a;
95         aL=strlen(a);
96         j=solve(nrx,0,0,aL-1,0,sL-1);
97         fout<<j<<"\n";
98     }
99
100    fout.close();
101    return 0;
102 }
```

Capitolul 24

ONI 2014

24.1 joc

Problema 1 - joc

100 de puncte

Costel are o mare pasiune pentru rezolvarea cubului Rubik, atât de mare încât a început să facă cercetări și calcule diverse pornind de la acest joc. Ultima lui idee, inspirată de cubul Rubik, folosește un cub de latură 2 unități, compus din 8 cuburi cu latura de o unitate (cub unitate), având fețele exterioare colorate. Fiecare cub unitate are 3 fețe exterioare și fiecare dintre acestea este colorată cu una din cele 10 culori disponibile, codificate prin cifrele de la 0 la 9.

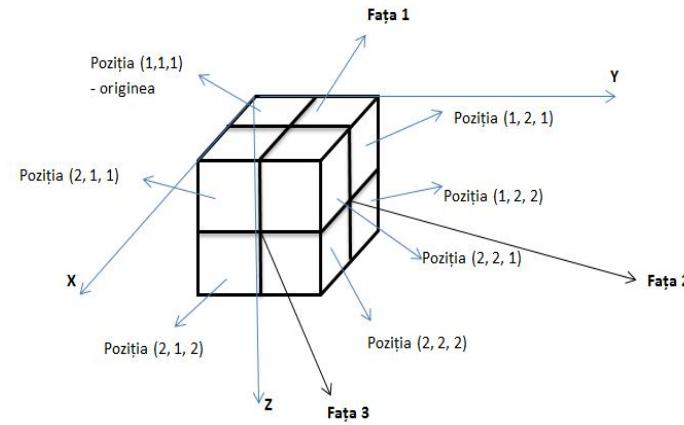


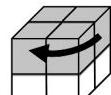
Figura 1

Fața 4		Fața 1		Fața 2		Fața 6		Fața 5	
(1,1,2)	(1,2,2)								
(1,1,1)	(1,2,1)								
(1,1,1)	(1,2,1)	(1,2,1)	(1,2,2)	(1,2,2)	(1,1,2)	(1,1,2)	(1,1,1)		
(2,1,1)	(2,2,1)	(2,2,1)	(2,2,2)	(2,2,2)	(2,1,2)	(2,1,2)	(2,1,1)		
(2,1,1)	(2,2,1)								
(2,1,2)	(2,2,2)								

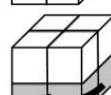
Figura 2

Identificarea cuburilor unitate se face conform specificațiilor din Figura 1. Cubul care nu este vizibil în Figura 1 are coordonatele (1, 1, 2). Cubul lui Costel permite efectuarea următoarelor tipuri de mutări, asemănătoare cu cele din cubul Rubik:

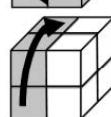
M1: Paralelipipedul 1 conține cuburile unitate de coordonate: (1, 1, 1); (1, 2, 1); (2, 1, 1); (2, 2, 1). Acesta este un disc așezat orizontal și poate fi rotit cu 90 de grade către dreapta, în sens acelor de ceasornic.



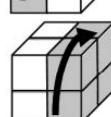
M2: Paralelipipedul 2 conține cuburile unitate de coordonate: (1, 1, 2); (1, 2, 2); (2, 1, 2); (2, 2, 2). Acesta este un disc așezat orizontal și poate fi rotit cu 90 de grade către dreapta, în sens invers acelor de ceasornic.



M3: Paralelipipedul 3 conține cuburile unitate de coordonate: (1, 1, 1); (2, 1, 1); (1, 1, 2); (2, 1, 2). Acesta este un disc așezat vertical și poate fi rotit cu 90 de grade către planul îndepărtat, în sens invers acelor de ceasornic.



M4: Paralelipipedul 4 conține cuburile unitate de coordonate: (1, 2, 1); (2, 2, 1); (1, 2, 2); (2, 2, 2). Acesta este un disc așezat vertical și poate fi rotit cu 90 de grade către planul îndepărtat, în sens acelor de ceasornic.



Prin configurație se înțelege memorarea culorii fiecărei fețe exterioare a celor 8 cuburi unitate, deci culorile celor 24 de fețe exterioare. Aplicând o succesiune validă de mutări se obține o altă configurație.

Pentru ușurința memorării unei configurații, Costel utilizează desfășurarea în plan a celor 6 fețe ale cubului său după modelul din Figura 2, care ilustrează modul în care sunt dispuse fețele

în desfășurare. Fiecare față a cubului conține patru fețe exterioare ale cuburilor unitate având, în ordine, coordonatele specificate în figură.

Cerințe

Fiind date o configurație inițială și o configurație finală ale jocului, determinați numărul minim de mutări prin care se poate ajunge de la configurația initială la configurația finală și succesiunea corespunzătoare de mutări prin care se poate obține configurația finală.

Date de intrare

Fișierul de intrare **joc.in** conține:

- 12 linii corespunzătoare configurației inițiale - câte două linii pentru fiecare din cele șase fețe; pe fiecare linie sunt memorate câte două cifre, separate prin exact un spațiu (pe primele două linii se află elementele feței 1, pe următoarele două linii se află elementele feței 2, ... , pe liniile 11 și 12 se află elementele feței 6).
- Pe următoarele 12 linii se află elementele configurației finale - câte două linii pentru fiecare din cele șase fețe; pe fiecare linie sunt memorate câte două cifre, separate prin exact un spațiu.

Date de ieșire

Fișierul de ieșire **joc.out** va conține:

- Pe prima linie, un număr natural MIN, reprezentând numărul minim de mutări determinat.
- Pe următoarele MIN linii succesiunea de mutări care transformă configurația inițială în cea finală, pe fiecare linie fiind scris un număr natural cuprins între 1 și 4 ce reprezintă numărul asociat unei mutări.

Restricții și precizări

- Se garantează că pentru toate datele de test există soluție, aceasta având cel mult 11 mutări.
- Orice soluție cu număr minim de mutări care conduce la obținerea configurației finale va obține punctajul maxim.

Exemple:

joc.in	joc.out	Explicații																																																																																																
1 2 3 1 2 7 5 4 2 9 9 4 2 9 4 5 5 8 2 3 6 4 1 7 2 2 9 1 2 7 5 4 7 9 4 4 1 9 3 5 8 3 5 2 6 4 1 2	1 3	<p>Se efectuează mutarea discului 3, către planul îndepărtat, adică mutarea 3.</p> <p style="text-align: center;">Configurația inițială</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td colspan="2" style="text-align: center;">Față 4</td> <td colspan="4" style="text-align: center;">Față 2 Față 6 Față 5</td> </tr> <tr> <td>2</td><td>9</td> <td></td><td></td><td></td><td></td> </tr> <tr> <td>4</td><td>5</td> <td></td><td></td><td></td><td></td> </tr> <tr> <td>1</td><td>2</td> <td>2</td><td>7</td> <td>6</td><td>4</td> </tr> <tr> <td>3</td><td>1</td> <td>5</td><td>4</td> <td>1</td><td>7</td> </tr> <tr> <td></td><td></td> <td>2</td><td>3</td> <td>2</td><td>8</td> </tr> <tr> <td></td><td></td> <td>9</td><td>4</td> <td>7</td><td>3</td> </tr> <tr> <td></td><td></td> <td></td><td></td> <td>4</td><td>2</td> </tr> <tr> <td></td><td></td> <td></td><td></td> <td>4</td><td>4</td> </tr> </table> <p style="text-align: center;">Configurația finală</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td colspan="2" style="text-align: center;">Față 4</td> <td colspan="4" style="text-align: center;">Față 2 Față 6 Față 5</td> </tr> <tr> <td>1</td><td>9</td> <td></td><td></td><td></td><td></td> </tr> <tr> <td>3</td><td>5</td> <td></td><td></td><td></td><td></td> </tr> <tr> <td>2</td><td>2</td> <td>2</td><td>7</td> <td>6</td><td>4</td> </tr> <tr> <td>9</td><td>1</td> <td>5</td><td>4</td> <td>1</td><td>2</td> </tr> <tr> <td>2</td><td>5</td> <td>7</td><td>3</td> <td>8</td><td>3</td> </tr> <tr> <td>4</td><td>4</td> <td></td><td></td> <td></td><td></td> </tr> </table> <p style="text-align: center;">Față 1 Față 3 Față 1 Față 3 Față 2 Față 6 Față 5</p>	Față 4		Față 2 Față 6 Față 5				2	9					4	5					1	2	2	7	6	4	3	1	5	4	1	7			2	3	2	8			9	4	7	3					4	2					4	4	Față 4		Față 2 Față 6 Față 5				1	9					3	5					2	2	2	7	6	4	9	1	5	4	1	2	2	5	7	3	8	3	4	4				
Față 4		Față 2 Față 6 Față 5																																																																																																
2	9																																																																																																	
4	5																																																																																																	
1	2	2	7	6	4																																																																																													
3	1	5	4	1	7																																																																																													
		2	3	2	8																																																																																													
		9	4	7	3																																																																																													
				4	2																																																																																													
				4	4																																																																																													
Față 4		Față 2 Față 6 Față 5																																																																																																
1	9																																																																																																	
3	5																																																																																																	
2	2	2	7	6	4																																																																																													
9	1	5	4	1	2																																																																																													
2	5	7	3	8	3																																																																																													
4	4																																																																																																	

Timp maxim de executare/test: **1.0** secunde pe Windows, **0.2** secunde pe Linux

Memorie: total **16 MB** din care pentru stivă **8 MB**

Dimensiune maximă a sursei: **20 KB**

24.1.1 Indicații de rezolvare

prof. Alin Burța, C.N. "B.P. Hasdeu" Buzău

Algoritmul de rezolvare se bazează pe expandarea unei cozi. Aceasta va conține configurațiile posibile, numărul minim de mutări în care s-au obținut acestea și indicele din coadă al configurației din care au fost obținute. Algoritmul are următorii pași:

1. Se introduce configurația inițială în coadă. Aceasta se obține în 0 mutări.
2. Se extrage o configurație din coadă.
 - a. Dacă s-a extras configurația finală, atunci numărul minim de mutări este cel asociat (memorat în coadă), iar succesiunea de mutări se reconstituie pe baza indicilor configurațiilor precedente.
 - b. Dacă configurația extrasă nu este cea finală, atunci se introduc în coadă configurațiile obținute din aceasta aplicând cele 4 mutări posibile. Numărul minim de mutări prin care se obține configurația introdusă în coadă este cu o unitate mai mare decât numărul minim de mutări în care s-a obținut configurația precedentă (cea din care s-a obținut configurația introdusă în coadă). Dacă una dintre acestea este configurația finală se aplică procedeul descris la punctul a). Se pot face verificări pentru a evita introducerea în coadă a unor configurații deja obținute.
3. Se repetă pasul 2 până la obținerea configurației finale.

Din cauza numărului mare de configurații posibile, coada poate atinge dimensiuni mari și de aceea este necesar ca o configurație să poată fi memorată într-un mod compact, folosind o codificare în baza 10 a fețelor cubului.

24.1.2 *Rezolvare detaliată

24.1.3 Cod sursă

Listing 24.1.1: joc.cpp

```

1 #include <fstream>
2 #include <cmath>
3
4 #define FIN "joc.in"
5 #define FOU "joc.out"
6 #define Cmax 500000
7 #define init 999999
8
9 using namespace std;
10
11 int in1, in2, in3, in4;           //configuratie initiala
12 int fil, fi2, fi3, fi4;         //configuratie finala
13 int cr1, cr2, cr3, cr4;        //configuratie curenta
14 int C1[Cmax], C2[Cmax], C3[Cmax], C4[Cmax]; //coada
15 int prim, ultim;               //indicii necesari cozii
16 short Nr[Cmax];                //numarul minim de mutari
17 int Pr[Cmax];                  //indicele din C al mutarii precedente
18 short Mv[Cmax];                //numarul mutarii
19
20 int putere(int e)
21 {
22     int i, rez = 1;
23     for(i=1;i<=e;i++) rez *= 10;

```

```

24     return rez;
25 }
26
27 void SetCif(int &x, int c, int i)
28 {
29     int p = putere(7-i);
30     x = ((x / p) * 10 + c) * (p / 10) + x % (p / 10);
31 }
32
33 int GetCif(int x, int i)
34 {
35     int p = putere(6-i);
36     return (x / p) % 10;
37 }
38
39 int E_Final(int n1, int n2, int n3, int n4)
40 {
41     return n1 == fil && n2 == fi2 && n3 == fi3 && n4 == fi4;
42 }
43
44 int E_Egal(int n1, int n2, int n3, int n4, int poz)
45 {
46     return n1 == C1[poz] && n2 == C2[poz] && n3 == C3[poz] && n4 == C4[poz];
47 }
48
49 int Apare(int n1, int n2, int n3, int n4)
50 {
51     int i , cat;
52     cat = max(1, ultim - 50);
53     for(i = ultim - 50 ; i <= ultim; i++)
54         if(C1[i] == n1 && C2[i] == n2 &&
55             C3[i] == n3 && C4[i] == n4 && i != prim)
56             return 1;
57     return 0;
58 }
59
60 void read()
61 {
62     ifstream in(FIN);
63     int x, y, z, t;
64
65     in1 = in2 = in3 = in4 = init;
66
67     //citesc configuratia initiala
68
69     in>>x>>y>>z>>t;
70     SetCif(in1, x, 3);
71     SetCif(in2, y, 3);
72     SetCif(in1, z, 4);
73     SetCif(in2, t, 4); //fata 1
74
75     in>>x>>y>>z>>t;
76     SetCif(in3, x, 1);
77     SetCif(in3, y, 2);
78     SetCif(in4, z, 1);
79     SetCif(in4, t, 2); //fata 2
80
81     in>>x>>y>>z>>t;
82     SetCif(in1, x, 5);
83     SetCif(in2, y, 5);
84     SetCif(in1, z, 6);
85     SetCif(in2, t, 6); //fata 3
86
87     in>>x>>y>>z>>t;
88     SetCif(in1, x, 1);
89     SetCif(in2, y, 1);
90     SetCif(in1, z, 2);
91     SetCif(in2, t, 2); //fata 4
92
93     in>>x>>y>>z>>t;
94     SetCif(in3, x, 5);
95     SetCif(in3, y, 6);
96     SetCif(in4, z, 5);
97     SetCif(in4, t, 6); //fata 5
98
99     in>>x>>y>>z>>t;

```

```

100     SetCif(in3, x, 3);
101     SetCif(in3, y, 4);
102     SetCif(in4, z, 3);
103     SetCif(in4, t, 4); //fata 6
104
105     fil = fi2 = fi3 = fi4 = init;
106
107 //citesc configuratia finala
108
109     in>>x>>y>>z>>t;
110     SetCif(fil, x, 3);
111     SetCif(fi2, y, 3);
112     SetCif(fil, z, 4);
113     SetCif(fi2, t, 4); //fata 1
114
115     in>>x>>y>>z>>t;
116     SetCif(fi3, x, 1);
117     SetCif(fi3, y, 2);
118     SetCif(fi4, z, 1);
119     SetCif(fi4, t, 2); //fata 2
120
121     in>>x>>y>>z>>t;
122     SetCif(fil, x, 5);
123     SetCif(fi2, y, 5);
124     SetCif(fil, z, 6);
125     SetCif(fi2, t, 6); //fata 3
126
127     in>>x>>y>>z>>t;
128     SetCif(fil, x, 1);
129     SetCif(fi2, y, 1);
130     SetCif(fil, z, 2);
131     SetCif(fi2, t, 2); //fata 4
132
133     in>>x>>y>>z>>t;
134     SetCif(fi3, x, 5);
135     SetCif(fi3, y, 6);
136     SetCif(fi4, z, 5);
137     SetCif(fi4, t, 6); //fata 5
138
139     in>>x>>y>>z>>t;
140     SetCif(fi3, x, 3);
141     SetCif(fi3, y, 4);
142     SetCif(fi4, z, 3);
143     SetCif(fi4, t, 4); //fata 6
144     in.close();
145 }
146
147 void move3(int &n1, int &n2, int &n3, int &n4) //vd1 -> Nord
148 {
149     int a1, a2;
150     a1 = GetCif(n1, 1);
151     a2 = GetCif(n1, 2);
152     SetCif(n1, GetCif(n1, 3), 1);
153     SetCif(n1, GetCif(n1, 4), 2);
154     SetCif(n1, GetCif(n1, 5), 3);
155     SetCif(n1, GetCif(n1, 6), 4);
156     SetCif(n1, GetCif(n4, 4), 5);
157     SetCif(n1, GetCif(n3, 4), 6);
158     SetCif(n4, a1, 4);
159     SetCif(n3, a2, 4);
160     a1 = GetCif(n3, 5);
161     SetCif(n3, GetCif(n3, 6), 5);
162     SetCif(n3, GetCif(n4, 6), 6);
163     SetCif(n4, GetCif(n4, 5), 6);
164     SetCif(n4, a1, 5);
165 }
166
167 void move4(int &n1, int &n2, int &n3, int &n4) //vd2 -> Nord
168 {
169     int a1, a2;
170     a1 = GetCif(n2, 1);
171     a2 = GetCif(n2, 2);
172     SetCif(n2, GetCif(n2, 3), 1);
173     SetCif(n2, GetCif(n2, 4), 2);
174     SetCif(n2, GetCif(n2, 5), 3);
175     SetCif(n2, GetCif(n2, 6), 4);

```

```

176     SetCif(n2, GetCif(n4, 3), 5);
177     SetCif(n2, GetCif(n3, 3), 6);
178     SetCif(n4, a1, 3);
179     SetCif(n3, a2, 3);
180     a1 = GetCif(n3, 1);
181     SetCif(n3, GetCif(n4, 1), 1);
182     SetCif(n4, GetCif(n4, 2), 1);
183     SetCif(n4, GetCif(n3, 2), 2);
184     SetCif(n3, a1, 2);
185 }
186
187 void move1(int &n1, int &n2, int &n3, int &n4) //od3 -> Est
188 {
189     int a1, a2;
190     a1 = GetCif(n1, 3);
191     SetCif(n1, GetCif(n1, 4), 3);
192     SetCif(n1, GetCif(n2, 4), 4);
193     SetCif(n2, GetCif(n2, 3), 4);
194     SetCif(n2, a1, 3);
195     a1 = GetCif(n1, 2);
196     a2 = GetCif(n2, 2);
197     SetCif(n1, GetCif(n4, 6), 2);
198     SetCif(n2, GetCif(n3, 6), 2);
199     SetCif(n4, GetCif(n2, 5), 6);
200     SetCif(n3, GetCif(n1, 5), 6);
201     SetCif(n2, GetCif(n3, 1), 5);
202     SetCif(n1, GetCif(n4, 1), 5);
203     SetCif(n3, a1, 1);
204     SetCif(n4, a2, 1);
205 }
206
207 void move2(int &n1, int &n2, int &n3, int &n4) //od4 -> Est
208 {
209     int a1, a2;
210     a1 = GetCif(n3, 3);
211     SetCif(n3, GetCif(n3, 4), 3);
212     SetCif(n3, GetCif(n4, 4), 4);
213     SetCif(n4, GetCif(n4, 3), 4);
214     SetCif(n4, a1, 3);
215     a1 = GetCif(n1, 1);
216     a2 = GetCif(n2, 1);
217     SetCif(n1, GetCif(n4, 5), 1);
218     SetCif(n2, GetCif(n3, 5), 1);
219     SetCif(n4, GetCif(n2, 6), 5);
220     SetCif(n3, GetCif(n1, 6), 5);
221     SetCif(n2, GetCif(n3, 2), 6);
222     SetCif(n1, GetCif(n4, 2), 6);
223     SetCif(n3, a1, 2);
224     SetCif(n4, a2, 2);
225 }
226
227 void solve_and_print()
228 {
229     int rez = init;
230     int tm1, tm2, tm3, tm4;
231     short R[100], Lm;
232     int i, j, poz;
233
234     //initialize
235
236     prim = ultim = 1;
237     C1[prim] = in1;
238     C2[prim] = in2;
239     C3[prim] = in3;
240     C4[prim] = in4;
241     Nr[1] = 0;
242     Pr[1] = 0;
243     Mv[1] = 0;
244     while( prim <= ultim )
245     {
246         cr1 = C1[prim];
247         cr2 = C2[prim];
248         cr3 = C3[prim];
249         cr4 = C4[prim];
250         if( E_Final(cr1, cr2, cr3, cr4) )
251     {

```

```

252         rez = Nr[prim];
253         poz = prim;
254         Lm = 0;
255         break;
256     }
257
258     //move 1
259
260     tm1 = cr1;
261     tm2 = cr2;
262     tm3 = cr3;
263     tm4 = cr4;
264     move1(tm1, tm2, tm3, tm4);
265     if( E_Final(tm1, tm2, tm3, tm4) )
266     {
267         rez = Nr[prim] + 1;
268         poz = prim;
269         Lm = 1;
270         break;
271     }
272     if( !Apare(tm1, tm2, tm3, tm4) )
273     {
274         ultim++;
275         C1[ultim] = tm1;
276         C2[ultim] = tm2;
277         C3[ultim] = tm3;
278         C4[ultim] = tm4;
279         Nr[ultim] = 1 + Nr[prim];
280         Pr[ultim] = prim;
281         Mv[ultim] = 1;
282     }
283
284     //move 2
285
286     tm1 = cr1;
287     tm2 = cr2;
288     tm3 = cr3;
289     tm4 = cr4;
290     move2(tm1, tm2, tm3, tm4);
291     if( E_Final(tm1, tm2, tm3, tm4) )
292     {
293         rez = Nr[prim] + 1;
294         poz = prim;
295         Lm = 2;
296         break;
297     }
298     if( !Apare(tm1, tm2, tm3, tm4) )
299     {
300         ultim++;
301         C1[ultim] = tm1;
302         C2[ultim] = tm2;
303         C3[ultim] = tm3;
304         C4[ultim] = tm4;
305         Nr[ultim] = 1 + Nr[prim];
306         Pr[ultim] = prim;
307         Mv[ultim] = 2;
308     }
309
310     //move 3
311
312     tm1 = cr1;
313     tm2 = cr2;
314     tm3 = cr3;
315     tm4 = cr4;
316     move3(tm1, tm2, tm3, tm4);
317     if( E_Final(tm1, tm2, tm3, tm4) )
318     {
319         rez = Nr[prim] + 1;
320         poz = prim;
321         Lm = 3;
322         break;
323     }
324     if( !Apare(tm1, tm2, tm3, tm4) )
325     {
326         ultim++;
327         C1[ultim] = tm1;

```

```

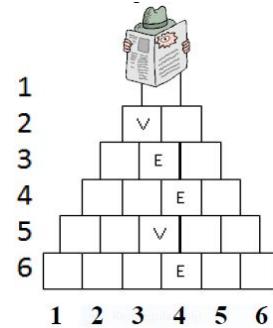
328         C2[ultim] = tm2;
329         C3[ultim] = tm3;
330         C4[ultim] = tm4;
331         Nr[ultim] = 1 + Nr[prim];
332         Pr[ultim] = prim;
333         Mv[ultim] = 3;
334     }
335
336     //move 4
337
338     tm1 = cr1;
339     tm2 = cr2;
340     tm3 = cr3;
341     tm4 = cr4;
342     move4(tm1, tm2, tm3, tm4);
343     if( E_Final(tm1, tm2, tm3, tm4) )
344     {
345         rez = Nr[prim] + 1;
346         poz = prim;
347         Lm = 4;
348         break;
349     }
350     if( !Apare(tm1, tm2, tm3, tm4) )
351     {
352         ultim++;
353         C1[ultim] = tm1;
354         C2[ultim] = tm2;
355         C3[ultim] = tm3;
356         C4[ultim] = tm4;
357         Nr[ultim] = 1 + Nr[prim];
358         Pr[ultim] = prim;
359         Mv[ultim] = 4;
360     }
361     prim++;
362 }
363
364 ofstream OUT(FOU);
365 OUT << rez << '\n';
366
367 //determin mutarile
368
369 i = rez;
370 if(Lm)
371 {
372     R[i] = Lm;
373     i--;
374 }
375
376 while(Pr[poz])
377 {
378     R[i--] = Mv[poz];
379     poz = Pr[poz];
380 }
381
382 for(j = 1; j <= rez ; j++)
383     OUT << R[j] << '\n';
384 OUT.close();
385
386 }
387
388 int main()
389 {
390     read();
391     solve_and_print();
392
393     return 0;
394 }
```

24.2 spion

Problema 2 - spion

100 de puncte

Spironul 008 vrea să găsească o locație secretă în junglă, având asupra lui un dispozitiv de localizare. Inițial spionul se află la intrarea în junglă pe nivelul 1 și cu fiecare pas, el avansează de la nivelul i la nivelul $i + 1$, ajungând la locația secretă, aflată pe ultimul nivel, în poziția u față de marginea stângă a nivelului curent. Pentru a ajunge în locația secretă, el poate să se deplaseze cu o poziție spre Sud-Est (codificat cu caracterul E) sau spre Sud-Vest (codificat cu caracterul V), trecând de pe nivelul i pe nivelul $i + 1$ cu viteza constantă. Numărul de poziții de pe un nivel crește cu unu față de nivelul anterior, conform imaginii alăturate. Numim traseu o succesiune formată din caracterele E sau V, corespunzătoare deplasării spionului de pe nivelul 1 la locația secretă. Pentru exemplul din figura alăturată succesiunea de caractere VEEVE reprezintă un traseu ce corespunde locației secrete din poziția 4 a nivelului 6.



Cerințe

Cunoscând succesiunea de caracter corespunzătoare unui traseu, determinați:

- poziția locației secrete de pe ultimul nivel;
- numărul de trasee distințe pe care le poate urma spionul plecând din poziția inițială pentru a ajunge în locația secretă corespunzătoare traseului dat. Două trasee se consideră distințe dacă diferă prin cel puțin o poziție.

Date de intrare

Fișierul de intrare **spion.in** conține pe prima linie un număr natural $p \in \{1, 2\}$, iar pe a doua linie o succesiune de caracter corespunzătoare unui traseu.

Date de ieșire

Dacă valoarea lui p este 1, atunci se va rezolva numai punctul a) din cerință. În acest caz, fișierul de ieșire **spion.out** va conține pe prima linie un număr natural ce reprezintă poziția de pe nivelul final a locației secrete.

Dacă valoarea lui p este 2, atunci se va rezolva numai punctul b) din cerință. În acest caz, fișierul de ieșire **spion.out** va conține pe prima linie un număr natural ce reprezintă numărul de trasee distințe modulo 100 003.

Restricții și precizări

- $2 \leq$ lungimea sirului pașilor $\leq 100\ 000$;
- pentru 20% din teste valoarea lui $p=1$;
- pentru alte 10% din teste valoarea lui $p = 2$ și lungimea secvenței de caracter ≤ 255 ;
- pentru alte 10% din teste valoarea lui $p = 2$ și $300 \leq$ lungimea secvenței de caracter $\leq 1\ 900$;
- pentru alte 10% din teste valoarea lui $p = 2$ și $3\ 000 \leq$ lungimea secvenței de caracter $\leq 5\ 000$.

Exemple:

spion.in	spion.out	Explicații
1 VEEVE	4	Locația secretă este în poziția 4 de pe nivelul 6.
2 VEV	3	Există trei trasee: VVE, VEV, EVV.
2 EVEVVEVEEE	210	

Timp maxim de executare/test: **0.3** secunde

Memorie: total **16 MB** din care pentru stivă **8 MB**

Dimensiune maximă a sursei: **20 KB**

24.2.1 Indicații de rezolvare

Cerință a) - 20p

Pentru determinarea poziției se verifică de câte ori agentul se deplasează spre Est. Valoarea astfel obținută se incrementează cu 1 și reprezintă rezultatul cerinței a).

Cerință b) - 80p

Pentru determinarea numărului de drumuri se observă că acesta reprezintă numărul din *triunghiul lui Pascal* aflat pe poziția locației secrete.

Soluția 1 - 20p (lungimea secvenței $\leq 1\ 900$)

Se generează triunghiul lui Pascal folosind un tablou bidimensional, dar pentru că numerele sunt foarte mari se calculează modulo 100003. Memoria nu permite dimensiuni mai mari decât 1 900.

Soluția 2 - 30p (lungimea secvenței $\leq 5\ 000$)

În locul tabloului bidimensional se folosesc două tablouri unidimensionale, care rețin ultimele două linii ale triunghiului lui Pascal.

Soluția 3 - 80p

Se observă că numărul traseelor distincte este egal cu

$$C_{lungime_sir}^{numarul_caracterelor_E}$$

Această valoare se poate calcula optim *invers modular*.

24.2.2 *Rezolvare detaliată

24.2.3 Cod sursă

Listing 24.2.1: spion_1.cpp

```

1 //Nicu Vlad-Laurentiu Liceul Mihail Kogalniceanu Vaslui
2 #include <iostream>
3 #include <cstring>
4
5 using namespace std;
6
7 const int N=100005, MOD=100003;
8
9 ifstream fin("spion.in");
10 ofstream fout("spion.out");
11
12 int fact[N], p;
13 char a[N];
14
15 int pw(int x, int y)
16 {
17     int ret=1;
18     for(;y;y>>=1)
19     {
20         if(y&1)
21         {
22             ret=(1LL*ret*x)%MOD;
23         }
24         x=(1LL*x*x)%MOD;
25     }
26     return ret;
27 }
28
29 void init(int n)
30 {
31     int i;
32     fact[1]=1;
33     for(i=2;i<=n;i++) fact[i]=(1LL*fact[i-1]*i)%MOD;
34 }
```

```

36 int comb(int n, int k)
37 {
38     if(!n||!k) return 1;
39     return 1LL*fact[n]*pw(1LL*fact[k]*fact[n-k]%MOD, MOD-2)%MOD;
40 }
41
42 int main()
43 {
44     int i, n, x=0, y=0;
45     fin>>p;fin.get();
46     fin.getline(a,N);
47     n=strlen(a);
48     init(n);
49     for(i=0;i<n;i++)
50     {
51         if(a[i]=='E') y++;
52         x++;
53     }
54     if(p==1)
55         fout<<y+1<<"\n";
56     else
57         fout<<comb(x, y)<<"\n";
58 }
```

24.3 zimeria

Problema 3 - zimeria

100 de puncte

Olimpia D'Info a găsit o placă gravată ce conține mai multe cuvinte scrise cu semne grafice necunoscute, fiecare cuvânt fiind format din exact 5 semne grafice. Studiind cu atenție cuvintele, a dedus că în scrierea acestora sunt utilizate 12 semne grafice distințe și a asociat căte o literă mică din alfabetul englez fiecarui semn. După asociere, a stabilit pentru fiecare semn o complexitate, scriind literele în ordinea crescătoare a complexităților pe care le-a stabilit anterior. Olimpia consideră că această "complexitate" este cel mai potrivit criteriu de ordonare lexicografică.

Cerințe

Cunoșcând ordinea semnelor și cuvintele de pe placă determinați:

- Numărul de cuvinte distințe existente pe placă.
- șirul de cuvinte ordonat lexicografic, conform criteriului formulat de Olimpia.

Date de intrare

Fișierul de intrare **zimeria.in** conține:

- pe prima linie un număr natural $p \in \{1, 2\}$, reprezentând varianta cerinței de rezolvare;
- pe a doua linie un număr natural n reprezentând numărul de cuvintede pe placă;
- pe a treia linie 12 caractere, litere mici ale alfabetului englez, care reprezintă semnele codificate, în ordinea lexicografică a semnelor;
- pe fiecare din următoarele n linii câte un cuvânt.

Date de ieșire

- Dacă valoarea lui p este 1, atunci se va rezolva numai punctul a) din cerință. În acest caz, fișierul de ieșire **zimeria.out** va conține pe prima linie numărul de cuvinte distințe de pe placă.
- Dacă valoarea lui p este 2, atunci se va rezolva numai punctul b) din cerință. În acest caz, fișierul de ieșire **zimeria.out** va conține n linii, pe fiecare linie câte un cuvânt în ordine lexicografică, conform complexității stabilite de către Olimpia.

Restricții și precizări

- $n < 400\ 000$;
- 30% din teste vor avea pe prima linie valoarea 1, iar restul de 70% din teste vor avea pe prima linie valoarea 2.

Exemplu:

zimeria.in	zimeria.out	Explicații
1 5 qwertyuiopas reeet wyuty reeet oiopp oiopp	3	Placa conține 3 cuvinte distincte.
2 5 qwertyuiopas oiopp reeet wyuty reeet oiopp	wyuty reeet reeet oiopp oiopp	Ordonăm cuvintele și obținem wyuty, reeet, reeet, oiopp, oiopp.

Timp maxim de executare/test: **0.6** secunde pe Windows, **0.2** secunde pe Linux

Memorie: total **4 MB** din care pentru stivă **2 MB**

Dimensiune maximă a sursei: **20 KB**

24.3.1 Indicații de rezolvare

Popa Daniel, Colegiul Național "Aurel Vlaicu" Orăștie

Soluție oficială

Solutia de 100 puncte presupune transformarea cuvintelor în numere (litera primului semn are pondere 0, urmatoarea 1, s.a.m.d), ultima literă a cuvantului se înmulțește cu 1, următoarea cu 12 s.a.m.d). Valorile obținute se sortează folosind *QuickSort* sau *CountSort*. La scrierea valorilor ordonate acestea se "convertesc" în text.

Soluție de maxim 70 puncte

O primă soluție ar fi să înlocuim literele "amestecate" cu literele a, b, c, d, ..., l, să facem substituția în fiecare cuvânt, să sortăm iar apoi să verificăm căte sunt distincte. Aceasta abordare aduce maxim 70 puncte, în funcție de sortarea folosită.

24.3.2 *Rezolvare detaliată

24.3.3 Cod sursă

Listing 24.3.1: zimeria.cpp

```

1 #include <iostream>
2 #include <cstdio>
3 #include <cstring>
4
5 using namespace std;
6
7 const int kc=12,nmax=1+kc*kc*kc*kc*kc;
8
9 int a[nmax]; //vector de frecventa
10 char o[13], // literele ordonate
11     cmin[6], // cuvantul minim(primul)
12     cmax[6], // cuvantul maxim(ultimul)
13     cu[6],s[6];
14
15 int nrcuv=0, //nr de cuvinte distincte

```

```

16     l[30], // codul coresponzator fiecarei litere
17     vmin=nmax, //valoarea cuvantului minim
18     vmax=-1, //valoarea cuvantului maxim
19     n, //nr de cuvinte
20     i,vc,p,j;
21
22 //determina valoarea unui cuvant
23
24 int val(char *c)
25 {
26     int s=0,p=1,i;
27     for(i=4;i>=0;i--)
28     {
29         s+=p*l[c[i]-'a'];
30         p*=kc;
31     }
32     return s;
33 }
34
35 char * cuv(int n)
36 {
37     char i;
38     s[5]=0;
39     for(i=4;i>=0;i--)
40     {
41         s[i]=o[n%12];
42         n/=12;
43     }
44     return s;
45 }
46
47 int main()
48 {
49     FILE *fi=fopen("zimeria.in","r");
50     FILE *fo=fopen("zimeria.out","w");
51
52     fscanf(fi,"%d\n%d\n",&p,&n);
53     fscanf(fi,"%s\n",o);
54
55     for(i=0;i<12;i++)
56         l[o[i]-'a']=i;//pregatim vectorul de calcul
57
58     for(i=1;i<=n;i++)
59     {
60         fscanf(fi,"%s\n",cu);
61         vc=val(cu);
62         if(vc>vmax) vmax=vc;
63         if(vc<vmin) vmin=vc;
64         if(a[vc]==0) nrcuv++; //numaram cuvintele distincte
65         a[vc]++;
66     }
67
68     if(p==1)
69         fprintf(fo,"%d",nrcuv);
70     else
71     {
72         cu[5]=0;
73         for(i=vmin;i<=vmax;i++)
74             if(a[i]!=0)
75             {
76                 strcpy(cu,cuv(i));
77                 for(j=1;j<=a[i];j++)
78                     fprintf(fo,"%s\n",cu);
79             }
80     }
81
82     fclose(fi);
83     fclose(fo);
84     return 0;
85 }

```

24.4 puteri

Problema 4 - puteri

100 de puncte

Nu e un secret pentru nimeni faptul că Mireluș se antrenează în timpul liber cu probleme de algoritmică.

De curând a aflat că un număr natural N , pentru care există două numere naturale nenule A și B ($B > 1$) astfel încât $N = A^B$, se numește putere. Mireluș și-a propus să determine numărul de puteri din intervalul $[X, Y]$, unde X și Y sunt numere naturale nenule.

Cum probabil v-ați imaginat deja, Mireluș nu a reușit să rezolve această problemă și a decis să ceară ajutorul Olimpiei D'Info. Pentru a fi sigur că nici ea nu greșește, i-a dat un set de intervale și i-a cerut să determine pentru fiecare interval numărul de puteri corespunzător.

Cerințe

Dându-se numărul de intervale T și pentru fiecare din cele T intervale cele două extremități, determinați numărul de puteri corespunzător fiecărui interval dat de Mireluș Olimpiei.

Date de intrare

Fișierul de intrare **puteri.in** conține pe prima linie numărul de intervale T , iar pe fiecare din următoarele T linii câte 2 numere naturale nenule X și Y , separate prin exact un spațiu, reprezentând extremitățile intervalelor.

Date de ieșire

Fișierul de ieșire **puteri.out** conține T linii. Fiecare linie va conține numărul de puteri care aparțin intervalului corespunzător din fișierul de intrare.

Restricții și precizări

- $1 \leq T \leq 131$.
- $1 \leq X \leq Y \leq 1018$.
- Intervalul $[X, Y]$ conține și numerele X și Y .
- Pentru 10% din teste $Y \leq 5000$.
- Pentru alte 25% din teste $Y \leq 100\,000$.
- Pentru alte 20% din teste $Y \leq 10\,000\,000$.

Exemple:

puteri.in	puteri.out	Explicații
1	9	Cele 9 numere sunt:
1 36		1, 4, 8, 9, 16, 25, 27, 32, 36

Timp maxim de executare/test: **0.5** secunde pe Windows, **0.1** secunde pe Linux

Memorie: total **64 MB** din care pentru stivă **32 MB**

Dimensiune maximă a sursei: **20 KB**

24.4.1 Indicații de rezolvare

stud. Cosmin-Mihai Tutunaru, Universitatea "Babeș Bolyai" Cluj

Fie următoarele observații:

Fie $f(x)$ o funcție care calculează răspunsul pentru intervalul $[1, x]$

Răspunsul pentru un interval $[x, y]$ este $f(y) - f(x - 1)$

Cel mai mare exponent B care ar putea fi este 60, deoarece $2^{60} > 10^{18}$.

Acum trebuie să vedem cum putem calcula $f(x)$.

Folosindu-ne de ultima observație, putem să calculăm răspunsul pentru fiecare B în parte. Răspunsul este evident partea întreagă a radicalului de ordinul B din x , radical ce poate fi calculat fără probleme cu o *căutare binară* (trebuie acordată mare atenție la înmulțiri, deoarece rezultatul produsului poate depăși 10^{18}).

Folosind această tactică numărăm unele numere de mai multe ori (ex: 2^4 și 4^2 , care ambele sunt 16). Pentru a evita acest lucru are sens să numărăm doar pentru acele valori B care sunt prime, astfel numărul 16 ar fi numărat doar ca 4^2 .

În continuare avem unele numere pe care le numărăm de mai multe ori: 8^2 și 4^3 . Putem evita acest lucru eliminând aceste baze care sunt la rândul lor puteri, dar cu un exponent strict mai mic. Astfel am număra $(2^3)^2$, dar nu am număra $(2^2)^3$.

Cumulând toate aceste observații, putem transforma funcția $f(x)$ în $f(x, e)$ care returnează răspunsul pentru intervalul $[1, x]$ cu restricția că $B < e$, iar un algoritm pseudocod ar fi următorul:

```

f(x, e) =
    • ret = 1 # îl numărăm mereu pe 1
    • pentru k de la 2 la e - 1
        – dacă k este număr prim
            * r = radical de ordinul k din x
            * dacă r > 1 # există cel puțin 2 numere
                · ret = ret + r
            · ret = ret - f(r, k) # eliminăm bazele care la rândul lor sunt puteri
    • return ret

```

Complexitate: $O(T * \log(Y)^3)$.

24.4.2 *Rezolvare detaliată

24.4.3 Cod sursă

Listing 24.4.1: puteri.cpp

```

1 #define TuTTY "Cosmin-Mihai Tutunaru"
2 #include <cstdio>
3 #include <cmath>
4
5 #define infile "puteri.in"
6 #define outfile "puteri.out"
7 #define ll long long
8 #define eMax 60
9
10 using namespace std;
11
12 bool prime[eMax];
13
14 void initialization()
15 {
16
17     prime[0] = prime[1] = true;
18
19     for (int i = 2; i < eMax; ++i)
20         if (prime[i] == false)
21             for (int j = i+i; j < eMax; j += i)
22                 prime[j] = true;
23 }
24
25 bool isPrime(int x)
26 {
27     return prime[x] == false;
28 }
29
30 ll ppw(ll x, int p)
31 {
32     ll ret = 1;
33     ll prv = 1;
34
35     while (p-- && ret > 0)
36     {
37         prv = ret;
38         ret *= x;
39
40         if (ret / x != prv)
41             return -1;

```

```

42     }
43
44     return ret;
45 }
46
47 ll root(ll x, int e)
48 {
49     ll le = 1, ri = x, sol = 1;
50
51     while (le <= ri)
52     {
53         ll me = (le+ri) / 2;
54         ll p = ppw(me, e);
55         if (p <= x && p > 0)
56             sol = me, le = me+1;
57         else
58             ri = me-1;
59     }
60
61     return sol;
62 }
63
64 ll f(ll x, int e = eMax)
65 {
66
67     if (x == 0)
68         return 0;
69
70     ll ret = 1;
71
72     for (int k = 2; k < e; ++k)
73     {
74         if (isPrime(k) == false)
75             continue;
76         ll r = root(x, k);
77         if (r == 1)
78             break;
79         ret += r - f(r, k);
80     }
81
82     return ret;
83 }
84
85 int main()
86 {
87     freopen(infile, "r", stdin);
88     freopen(outfile, "w", stdout);
89
90     int t;
91     scanf("%d\n", &t);
92
93     initialization();
94
95     while (t--)
96     {
97         ll x, y;
98         scanf("%lld %lld\n", &x, &y);
99         printf("%lld\n", f(y) - f(x-1));
100    }
101
102    fclose(stdin);
103    fclose(stdout);
104    return 0;
105 }
```

24.5 rascoala

Problema 5 - rascoala

100 de puncte

Suleiman I s-a confruntat în anul 1548 cu mari probleme interne. În acel an, el a primit vestea că într-o din regiunile Imperiului se pregătește o răscoală. Harta Imperiului este realizată sub forma unui tablou bidimensional cu n linii și m coloane, iar fiecare element al tabloului corespunde unei regiuni a Imperiului. În fiecare regiune erau deja cantonați soldați, dar pentru a preîntâmpina

răscoala sultanul decide ca toți cei k soldați din Garda Imperială să fie trimiși în regiuni, întărindu-le pe cele păzite de mai puțini soldați. Distribuirea lor respectă următoarele reguli:

- Dacă există o singură regiune cu număr de soldați mai mic decât al tuturor celorlalte regiuni, trimite un soldat în această regiune.
- Dacă există mai multe regiuni cu același număr minim de soldați, trimite un soldat în regiunea care inițial avea un număr mai mic de soldați. Dacă mai multe regiuni aveau același număr inițial de soldați, se trimite un soldat în regiunea cu indicele liniei mai mic, iar dacă regiunile sunt pe aceeași linie, în regiunea cu indicele coloanei mai mic.

Suleiman continuă distribuirea soldaților din garda imperială în regiuni conform celor precizate anterior, până la epuizarea soldaților din Garda Imperială.

Cerințe

Cunoscându-se n , m și k reprezentând numărul de linii, numărul de coloane, respectiv numărul de soldați din Garda Imperială, precum și numărul de soldați existent deja în regiunile Imperiului, să se determine:

- numărul de regiuni din Imperiu în care vor fi trimiși soldații din Garda Imperială, respectiv numărul minim de soldați care se vor găsi într-o regiune, după trimitera soldaților din Garda Imperială;
- distanța maximă între două regiuni în care au fost trimiși soldați ai Gărzii Imperiale. Distanța între o regiune A și o regiune B se calculează folosind formula $|LA - LB| + |CA - CB|$, unde (LA, CA) reprezintă coordonatele regiunii A , precizate prin numărul liniei și coloanei, respectiv (LB, CB) reprezintă coordonatele regiunii B , precizate prin numărul liniei și coloanei.

Date de intrare

Fișierul de intrare **rascoala.in** conține pe prima linie un număr natural $p \in \{1, 2\}$.

Pe a doua linie a fișierului se găsesc trei valori naturale n , m și k , despărțite printr-un spațiu, cu semnificația din enunț.

Pe fiecare dintre următoarele n linii se află câte m numere naturale, separate prin câte un spațiu, reprezentând numărul de soldați aflați inițial în fiecare regiune.

Date de ieșire

Dacă valoarea lui p este 1, atunci se va rezolva numai punctul a) din cerință. În acest caz, fișierul de ieșire **rascoala.out** va conține două valori naturale, fiecare pe câte un rând, reprezentând în ordine, numărul de regiuni în care a trimis Suleiman soldații din Garda Imperială, respectiv, numărul minim de soldați care se află într-o regiune după trimitera soldaților din Garda Imperială.

Dacă valoarea lui p este 2, atunci se va rezolva numai punctul b) din cerință. În acest caz, fișierul de ieșire **rascoala.out** va conține un singur număr natural, reprezentând distanța maximă între două regiuni în care au fost trimiși soldați ai Gărzii Imperiale.

Restricții și precizări

- $1 \leq n, m \leq 500$;
- $1 \leq k \leq 1\,000\,000\,000$;
- numărul inițial de soldați din orice regiune este un număr natural nenul ce nu depășește 1 000 000;
 - 40% din teste vor avea pe prima linie valoarea 1, iar restul de 60% din teste vor avea pe prima linie valoarea 2.

Exemple:

rascoala.in	rascoala.out	Explicații
1 3 4 6 5 3 4 6 5 5 8 5 9 6 8 7	3 5	Se trimit un soldat în regiunea (1,2), obținând două regiuni cu căte 4 soldați. Se trimit un soldat în regiunea (1,2), (număr inițial de soldați minim), apoi un soldat în regiunea (1,3). Cei trei soldați rămași vor fi trimiși astfel: primul în regiunea (1,2), al doilea în regiunea (1,3), iar al treilea în regiunea (1,1).
2 3 4 6 5 3 4 6 5 5 8 5 9 6 8 7	2	Distanța maximă va fi 2 între regiunile (1, 1) și (1,3).

Timp maxim de executare/test: **0.6** secunde pe Windows, **0.6** secunde pe linux

Memorie: total **32 MB** din care pentru stivă **16 MB**

24.5.1 Indicații de rezolvare

prof. Lukács Sándor, Liceul Teoretic "Onisifor Ghibu" Oradea

Memorăm numărul inițial de soldați și coordonatele lor într-un tablou unidimensional cu elemente de tip structură/înregistrare sau în 3 tablouri unidimensionale separate. Tabloul/-tablourile se sortează crescător, în funcție de cele 3 criterii enunțate (complexitate acceptată $O(n*m*log(n*m))$ sau $O(m*n+1\ 000\ 000)$).

Începând cu prima regiune, cât timp mai există soldați, numărul soldaților din regiunea/regiunile cu număr minim de soldați se va suplimenta, astfel încât să se ajungă la următorul nivel valoric de soldați (complexitate $O(n * m)$).

Pentru a afla distanța maximă între două regiuni, în care există soldați din Garda Imperială, se determină punctele care sunt maxime sau minime pe linii și pe coloane. Numărul acestor puncte poate fi cel mult $2 * (m + n)$. Rezultatul căutat va fi distanța maximă între două dintre aceste puncte.

24.5.2 *Rezolvare detaliată

24.5.3 Cod sursă

Listing 24.5.1: rascoala.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <cmath>
4 #include <algorithm>
5
6 using namespace std;
7
8 ifstream f("rascoala.in");
9 ofstream g("rascoala.out");
10
11
12 int n,p,k,r,min_s,m,xmin[505],xmax[505],ymin[505],ymax[505];
13
14 struct cetate
15 {
16     int x, y, s;
17 } a[250005],PX[250005],aux;
18
19 void citire()
20 {
21     int i,j;
22     f>>p>>n>>m>>k;
23
24     for(i=1;i<=n;i++)
25         for(j=1;j<=m;j++)
26             f>>a[(i-1)*m+j].s;

```

```

27         a[(i-1)*m+j].x = i;
28         a[(i-1)*m+j].y = j;
29     }
30 }
31 a[n*m+1].s=2000000005;// sa nu mai pot adauga soldati dupa regiunea n*m
32 }
33 }
34
35 int smaller(int i, long pivot, long pivotx, long pivoty)
36 {
37     if(a[i].s<pivot)
38         return 1;// daca numarul de soldati din regiunea i e mai mic decat
39                     // bpivotul elementul i e mai mic decat pivotul
40     else
41         if(a[i].s>pivot)
42             return 0;
43     else
44         if(a[i].x<pivotx)
45             return 1;// daca numarul de soldati e egal verific
46                     // indicele liniei
47     else
48         if(a[i].x>pivotx)
49             return 0;
50     else
51         if(a[i].y<pivoty)
52             return 1;// daca numarul de soldati si indicele liniei
53                     // sunt egale, verific indicele coloanei
54     else
55         return 0;
56 }
57
58 int bigger(int i, long pivot, long pivotx, long pivoty)
59 {
60     if(a[i].s>pivot) return 1;
61     else if(a[i].s<pivot) return 0;
62     else if(a[i].x>pivotx) return 1;
63     else if(a[i].x<pivotx) return 0;
64     else if(a[i].y>pivoty) return 1;
65     else return 0;
66 }
67 }
68
69 void quickSort(struct cetate *a, int left, int right)
70 {
71     int i = left, j = right;
72     long pivot = a[(left + right) / 2].s;
73     long pivotx = a[(left + right) / 2].x;
74     long pivoty = a[(left + right) / 2].y;
75     while (i <= j)
76     {
77         while (smaller(i,pivot,pivotx,pivoty)&&i<=right)
78             i++;
79         while (bigger(j,pivot,pivotx,pivoty))
80             j--;
81         if (i <= j)
82         {
83             aux=a[i];
84             a[i]=a[j];
85             a[j]=aux;
86             i++;
87             j--;
88         }
89     };
90     if (left < j)
91         quickSort(a, left, j);
92     if (i < right)
93         quickSort(a, i, right);
94 }
95 }
96
97 void solutie()
98 {
99     int i,sldt;
100    long num;
101    sldt=0;
102    r=1;//am utilizat 0 soldati

```

```

103     num=k;
104     min_s=a[1].s;
105     i=1;
106     while(r<n*m && sldt<k)
107     { //cat timp mai am regiuni si soldati
108
109         while(i<n*m && a[i+1].s==a[i].s)
110             i++; // numaram cate regiuni egale sunt
111
112         if(i<n*m)
113             sldt=sldt+i*(a[i+1].s-a[i].s); // reg regiuni X nr de soldati din
114                                         // fiecare regiune cu numar minim
115         else
116             sldt=k;
117
118         if(sldt<k)
119             { //daca poate pune in toate regiunile
120                 r=i;
121
122                 num=k-sldt; // trec la urmatoarea regiune, retin cati soldati
123                                         // mai raman
124                 i++;
125                 min_s=a[i].s; // retin numarul minim de soldati
126             }
127         }
128
129         if(i <= num)
130             r = i;
131         else
132             if(r<num)
133                 r=num;
134
135         min_s = min_s + num/i;
136     }
137
138 void dist_max()
139 {
140     //retinem toate elementele de pe chenar
141     //xmin[i] - retine coloana minima de pe linia i in care au fost
142                                         // trimisi soldati
143     //xmax[i] - retine coloana maxima afectata de pe linia i
144
145     int i,j,dmax,dx,dy;
146     int ax,ay,bx,by;
147
148     //initializari
149     for(i=1;i<=n;i++)
150     {
151         xmin[i] = 505;
152         xmax[i] = 0;
153     }
154
155     for(i=1;i<=m;i++)
156     {
157         ymin[i] = 505;
158         ymax[i] = 0;
159     }
160
161     for(i=1;i<=r;i++) //consideram regiunile in care au fost trimisi soldati
162     {
163         if(xmin[a[i].x] > a[i].y ) xmin[a[i].x] = a[i].y;
164         if(xmax[a[i].x] < a[i].y ) xmax[a[i].x] = a[i].y;
165         if(ymin[a[i].y] > a[i].x ) ymin[a[i].y] = a[i].x;
166         if(ymax[a[i].y] < a[i].x ) ymax[a[i].y] = a[i].x;
167     }
168
169     //distanta maxima va fi intre 2 elemente de pe chenar
170     //copiez elentele de pe chenar in vectorul PX(nu e neaparat nevoie)
171
172     int nr=0;
173     for(i=1;i<=m;i++)
174         if(ymin[i]<=n)
175             { //daca pe coloana i exista regiuni afectate
176                 nr++;
177                 PX[nr].x = ymin[i];
178                 PX[nr].y = i;

```

```

179         }
180
181     for(i=1;i<=n;i++)
182     {
183         if(xmax[i]>0)
184         {
185             //daca pe linia i exista regiuni afectate
186             nr++;
187             PX[nr].x = i;
188             PX[nr].y = xmax[i];
189         }
190
191     for(i=m;i>=1;i--)
192     {
193         if(ymax[i]>0)
194         {
195             nr++;
196             PX[nr].x = ymax[i]; PX[nr].y = i;
197         }
198
199     for(i=n;i>=1;i--)
200     {
201         if(xmin[i]<=n)
202         {
203             nr++;
204             PX[nr].x = i;
205             PX[nr].y = xmin[i];
206         }
207
208         //determinam maximul distantei intre 2 elemente de pe chenar
209         //fiind maxim 2*n+2*m elemente se poate aplica brute-force
210
211         dmax = 0;
212         for(i=1;i<nr;i++)
213         {
214             for(j=i+1;j<=nr;j++)
215             {
216                 if(PX[i].x > PX[j].x)
217                     dx = PX[i].x - PX[j].x;
218                 else
219                     dx = PX[j].x - PX[i].x;
220
221                 if(PX[i].y > PX[j].y)
222                     dy = PX[i].y - PX[j].y;
223                 else
224                     dy = PX[j].y - PX[i].y;
225
226                 if(dmax < dx + dy)
227                 {
228                     dmax = dx + dy;
229                     ax = PX[i].x;ay=PX[i].y;
230                     bx = PX[j].x;by = PX[j].y;
231
232                     g<<dmax;
233                 }
234             }
235         }
236     int main()
237     {
238         citire();
239         quickSort(a,1,n*m);
240         solutie(); //determina numarul de regiuni si numarul minim de soldati
241         if(p==1)
242             g<<r<<"\n"<<min_s<<"\n";
243         else
244             dist_max();
245
246         return 0;
247     }

```

24.6 stiva

Problema 6 - stiva

100 de puncte

Olivius d'Info a primit de ziua lui o stivă și s-a bucurat foarte tare. S-a tot gândit ce să facă cu ea și a inventat un joc de logică pentru colegii lui de clasă.

În prima fază el a scris mai multe biletele, conținând fiecare câte o permutare a primelor n

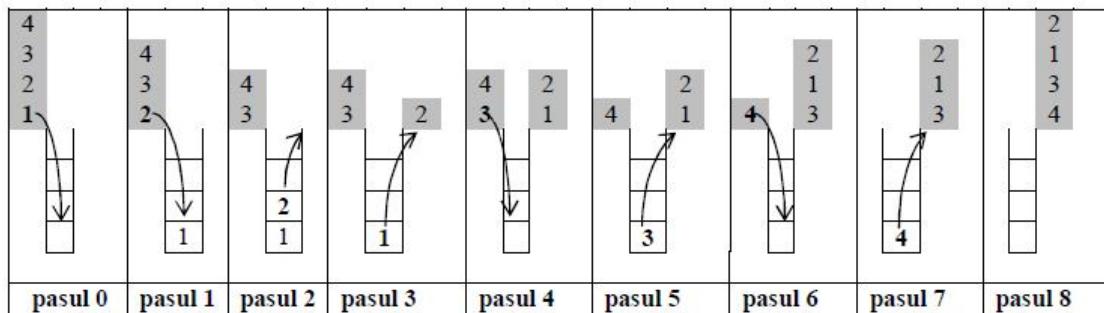
numere naturale nenule: $1, 2, 3, \dots, n$. Biletelele scrise conțin permutări pentru diferite valori ale lui n .

A clasificat aceste permutări în *permute stivuite* și *permute nestivuite*.

O *permute* este *stivuită* dacă se poate obține pe parcursul introducerii în stivă a numerelor $1, 2, 3, \dots, n$ în această ordine, prin extragerea elementelor, în ordinea indicată în permutare.

O *permute* *nestivuită* este o permutare care NU se poate obține prin procedeul de mai sus.

Respectând procedeul lui Olivius, pentru $n=4$, *permutea stivuită* $(2,1,3,4)$ se obține astfel:



Succesiunile $(3,1,2,4)$ și $(4,2,1,3)$ sunt permutări nestivuite.

În faza a doua, unele biletele au fost scurte din stânga și/sau din dreapta. Astfel, din permutarea stivuită $(2,1,3,4)$ se pot obține succesiuni de lungime mai mică: $(1,3,4)$, $(2,1,3)$, $(1,3)$, (3) etc.

Orice succesiune care aparține unei permutări stivuite, poate aparține și unei permutări nestivuite.

De exemplu, succesiunea $(2,1,3)$ aparține atât permutării stivuite $(2,1,3,4)$, cât și permutării nestivuite $(4,2,1,3)$.

Cerinte

Dându-se mai multe succesiuni de numere naturale distincte, determinați, pentru fiecare dintre acestea, dacă aparțin cel puțin unei *permute stivuite*.

Date de intrare

Fisierul **stiva.in** conține un set de cinci succesiuni de elemente, după cum urmează:

- pe prima linie un număr natural k , reprezentând numărul de elemente al fiecareia dintre cele cinci succesiuni;
- pe fiecare din următoarele cinci linii câte k numere naturale nenule, separate prin câte un spațiu, reprezentând elementele unei succesiuni.

Date de ieșire

Fisierul **stiva.out** va conține 5 linii, pe fiecare linie câte un număr natural astfel:

1 - dacă succesiunea curentă aparține unei permutări stivuite;

0 - dacă succesiunea curentă nu aparține unei permutări stivuite.

Răspunsurile se scriu pe câte o linie, în ordinea aparițiilor succesiunilor în fisierul de intrare.

Restricții și precizări

- $1 \leq$ valoarea elementelor din succesiune $\leq 2.000.000.000$;
- diferența dintre cel mai mare și cel mai mic element al succesiunii nu depășește 50.000;
- pentru 50% din teste, elementele din succesiune nu depășesc 50.000;
- elementele dintr-o succesiune sunt distincte două căte două.

Exemplu:

stiva.in	stiva.out	Explicații
3	1	$n = 3$, avem cinci succesiuni de numere, fiecare de lungime 3.
1 3 4	1	- succesiunea (1,3,4) aparține unei permutări stivuite (răspuns corect 1);
2 1 3	0	- succesiunea (2,1,3) aparține unei permutări stivuite (răspuns corect 1);
3 1 2	1	- succesiunea (3,1,2) nu aparține niciunei permutări stivuite (răspuns corect 0);
1 2 4	0	- succesiunea (1,2,4) aparține unei permutări stivuite (răspuns corect 1);
1 4 2		- succesiunea (1,4,2) nu aparține niciunei permutări stivuite (răspuns corect 0).

Timp maxim de executare/test: **0.3** secunde pe Windows, **0.1** secunde pe linux

Memorie: total **32 MB** din care pentru stivă **16 MB**

Dimensiune maximă a sursei: **20 KB**

24.6.1 Indicații de rezolvare

prof. Zoltan Szabo, IȘJ Mureș

Enumerăm câteva proprietăți pe care le respectă o permutare/secvență stivuită:

1. O secvență *stivuită compactă* ce conține toate numerele naturale dintr-un interval $[a,b]$, adică toate numerele $a, a+1, a+2, \dots, b-1, b$, este echivalentă cu o *permuteare stivuită*, ce se obține prin micșorarea tuturor elementelor cu $a-1$, respectiv $1, 2, 3, \dots, b-a+1$.

în acest mod, orice *secvență compactă* de intrare se poate transforma într-o *permuteare stivuită*.

Analog, orice secvență (nu neapărat compactă) pe intervalul $[a,b]$, se poate transforma într-o secvență (nu neapărat compactă) pe intervalul $[1,n]$, unde $n \leq 50\,000$.

2. Într-o permutare stivuită, poziția valorii 1 împarte permutarea în două secvențe compacte stivuite adică, dacă în permutarea stivuită de lungime n , elementul 1 se află pe poziția k , $p_1 p_2 p_3 \dots p_{k-1} p_k = 1 p_{k+1} \dots p_n$, atunci secvențele $p_1 p_2 p_3 \dots p_{k-1}$ și $p_{k+1} \dots p_n$ vor fi la rândul lor secvențe compacte ce se pot transforma în permutare stivuită.

3. O secvență stivuită de lungime n care îl conține pe 1 pe o poziție k , $s_1 s_2 s_3 \dots s_{k-1} s_{k+1} \dots s_n$, se poate transforma într-o permutare stivuită.

Notăm cu $\max = \max(s_1 \dots s_{k-1})$. Adăugăm în stânga toate numerele lipsă ale multșimii $\{1, 2, 3, \dots, \max\}$ în ordine crescătoare, iar numerele lipsă ale multșimii $\{\max + 1, \max + 2, \dots, n\}$ se adaugă la dreapta în ordine descrescătoare. Astfel se obține o permutare stivuită.

O secvență transformată într-o permutare cu metoda de mai sus va fi stivuită dacă și numai dacă și permutarea este stivuită.

Algoritmul optim de rezolvare constă în a obține pentru secvențele de intrare câte o permutare folosind proprietățile 1, 2, 3 de mai sus, iar apoi se verifică cu ajutorul unei stive dacă permutarea se poate obține cu procedeul enunțat în problemă (introducând elementele 1, 2, 3, ... în acestă ordine).

Complexitate: $O(n)$.

24.6.2 *Rezolvare detaliată

24.6.3 Cod sursă

Listing 24.6.1: stiva_90p.cpp

```

1 # include <fstream>
2 # include <iostream>
3 # include <algorithm>
4 # include <bitset>
5 # include <cstring>
6
7 # define DIM 50003
8 # define MAX 2147000000
9
10 # define INF "stiva.in"
11 # define OUTF "stiva.out"
```

```

12
13 using namespace std;
14 int n, w[DIM], v[DIM], ax[4*DIM], an[4*DIM], m, M, poz[DIM];
15 bitset<DIM>bs;
16
17 void substr()
18 {
19     for(int i=1;i<=n;++i)
20         w[i] -= m-1;
21 }
22
23 void complet()
24 {
25     int mst=0, g1=0, mdr=0;
26     for(int i=1;i<=n;++i)
27     {
28         bs.set(w[i], true);
29
30         if (w[i]==1)
31             g1=1;
32         else
33             if (!g1 && w[i] > mst)
34                 mst = w[i];
35             else
36                 if (g1 && w[i] > mdr)
37                     mdr = w[i];
38     }
39
40     for(int i=1;i<=mst;++i)
41         if (!bs.test(i))
42             v[++v[0]] = i;
43
44     for(int i=1;i<=n;++i)
45         v[++v[0]] = w[i];
46
47     for(int i=mdr;i>mst;--i)
48         if (!bs.test(i))
49             v[++v[0]] = i;
50 }
51
52 void upd(int k, int st, int dr, int p)
53 {
54     if (st==dr)
55     {
56         ax[k]=v[p];
57         an[k]=v[p];
58
59         return;
60     }
61
62     int mij = (st+dr)/2;
63
64     if (p<=mij)
65         upd(2*k, st, mij, p);
66     else
67         upd(2*k+1, mij+1, dr, p);
68
69     ax[k] = max(ax[2*k], ax[2*k+1]);
70     an[k] = min(an[2*k], an[2*k+1]);
71 }
72
73 int getMin(int k, int st, int dr, int i, int j)
74 {
75     if (st>=i && dr<=j)
76         return an[k];
77
78     int mij = (st+dr)/2, x=MAX, y=MAX;
79
80     if (i<=mij)
81         x = getMin(2*k, st, mij, i, j);
82     if (mij<j)
83         y = getMin(2*k+1, mij+1, dr, i, j);
84
85     return min(x,y);
86 }
87

```

```

88 int getMax(int k, int st, int dr, int i, int j)
89 {
90     if (i<=st && dr<=j)
91         return ax[k];
92
93     int mij = (st+dr)/2, x=0, y=0;
94
95     if (i<=mij)
96         x = getMax(2*k, st, mij, i, j);
97     if (mij<j)
98         y = getMax(2*k+1, mij+1, dr, i, j);
99
100    return max(x, y);
101 }
102
103 int dei(int st, int dr, int q)
104 {
105     if (st>=dr) return 1;
106
107     int m = getMin(1, 1, n, st, dr);
108     int M = getMax(1, 1, n, st, dr);
109     int Mst=getMax(1, 1, n, st, poz[m]);
110
111     if (q != m || M-m != dr-st || Mst-m != poz[m]-st)
112         return 0;
113
114     int rst = dei(st, poz[m]-1, m+1);
115     int rdr = dei(poz[m]+1, dr, Mst+1);
116
117     return rst && rdr;
118 }
119
120 int solve()
121 {
122     if (m != 1)
123         substr();
124
125     if (M-m+1 != n)
126         complet();
127     else
128         for(int i=1;i<=n;++i)
129             v[i]=w[i];
130
131     n = M-m+1;
132
133     for(int i=1;i<=n;++i)
134     {
135         upd(1, 1, n, i);
136         poz[v[i]]=i;
137     }
138
139     return dei(1, n, 1);
140 }
141
142 int main()
143 {
144     ifstream fin (INF);
145     ofstream fout (OUTF);
146
147     int nn;
148     fin>>nn;
149     for(int tt=1;tt<=5;++tt)
150     {
151         n=nn;
152         m=MAX;M=0;
153         bs.reset();
154
155         for(int i=1;i<=n;++i)
156         {
157             fin>>w[i];
158             m = min(m,w[i]);
159             M = max(M,w[i]);
160         }
161
162         memset(&v, 0, sizeof(v));
163         memset(&an, MAX, sizeof(an));

```

```

164         memset(&ax, 0, sizeof(ax));
165
166     fout<<solve()<<"\n";
167 }
168
169 return 0;
170 }
```

Listing 24.6.2: stiva_100p.cpp

```

1 // profesor Szabo Zoltan - Liceul Tehnologic "Petru Maior" Reghin
2 #include <iostream>
3 #include <fstream>
4
5 using namespace std;
6
7 int main()
8 {
9     int n,a[50003],i,j,st[50003],k,max,min,p,b[50003],maxst,maxdr,start;
10    bool ok1;
11
12    ifstream fin("stiva.in");
13    ofstream fout("stiva.out");
14
15    fin>>n;
16    for (i=1;i<=5;++i)
17    {
18        for (j=1;j<=50000;++j)
19            b[j]=0;           // b va fi vectorul de frecventa pentru elemente
20
21        max=0;
22        min=2000000000;
23        for(j=1;j<=n;++j)
24        {
25            fin>>a[j];           // citesc elementele din fisier
26            if (a[j]>max)          // calculez maximul elementelor
27                max=a[j];
28            if (a[j]<min)          // calculez minimul elementelor
29                min=a[j];
30        }
31
32        maxst=0;
33        maxdr=0;
34        ok1=false;
35        for (j=1;j<=n;++j)
36        {
37            a[j]=a[j]-min+1; // micsorez elementele sa le transform in 1,2,3,..
38            b[a[j]]=1;
39            if (a[j]==1)
40                ok1=true;
41            if (not ok1)
42                if (a[j]>maxst) // in stanga lui 1 orice numar existent
43                    // in stiva va trebui scos
44                maxst=a[j]; // sa il putem scoata si pe 1. de aceea
45                // calculam maxst
46        }
47
48        max=max-min+1;
49        start=1;
50        for (j=maxst;j>1;j--)
51        {
52            if (b[j]==0)
53            {
54                start=(start+49999)%50000; // numerele mai mici decat maxst
55                // le pun in stanga sirului
56                a[start]=j;
57            }
58
59            p=n; // astfel intervalul efectiv este [1,max] din care mai lipsesc
60            // cateva numere
61            for (j=max;j>=maxst;--j)
62            {
63                if (b[j]==0)
64                {
65                    a[++p]=j; // completez cu elementele lipsa din permutare
66                }
67            }
68        }
69    }
70 }
```

```
66      k=0;
67      p=start;
68      for(j=1; j<=max; ++j)
69      {
70          st[++k]=j;
71
72          while(k>0 and st[k]==a[p])
73          {
74              p=(p+1)%50000;
75              --k;
76          }
77      }
78
79      if (k==0)
80          fout<<1<<"\n";
81      else
82          fout<<0<<"\n";
83  }
84
85  fout.close();
86  fin.close();
87  return 0;
88 }
```

Capitolul 25

ONI 2013

25.1 cumpănit

Problema 1 - cumpănit

100 de puncte

Un număr natural nenul n se numește cumpănit dacă în descompunerea sa în factori primi suma bazelor este egală cu suma exponentilor.

De exemplu, numerele $72 = 2^3 \cdot 3^2$, $5760 = 2^7 \cdot 3^2 \cdot 5^1$ sunt cumpănite.

Cerințe

Să se scrie un program care citește două numere naturale nenule a și b și determină toate numerele cumpănite din intervalul închis $[a, b]$.

De exemplu, dacă $a = 2$ și $b = 99$, numerele cumpănite cuprinse între 2 și 99 sunt 4, 27, 48 și 72.

Date de intrare

Fișierul de intrare **cumpănit.in** conține pe prima linie numerele naturale nenule a și b despărțite prin exact un spațiu, cu semnificația de mai sus.

Date de ieșire

Fișierul de ieșire **cumpănit.out** va conține numerele căutate, scrise în ordine crescătoare, câte unul pe fiecare linie.

Restricții și precizări

- $2 \leq a \leq b \leq 10^{14}$
- Pentru 25% din teste se garantează că $2 \leq a \leq b \leq 10^6$

Exemple:

cumpănit.in	cumpănit.out	Explicații
2 99	4 27 48 72	$a = 2$, $b = 99$ Numerele cumpănite cuprinse între 2 și 99 sunt 4, 27, 48, 72

Timp maxim de executare/test: **1.0** secunde

Memorie: total **8 MB** din care pentru stivă **4 MB**

Dimensiune maximă a sursei: **5 KB**

25.1.1 Indicații de rezolvare

Varianta 1

Se poate alege o variantă "brute force", adică generarea numerelor prime cu *ciurul lui Eratostene* apoi descompunerea numerelor în factori primi și verificarea condiției de număr cumpănit, adică suma bazelor egală cu suma exponentilor.

Această variantă obține aproximativ 30 puncte.

Varianta 2 (recursivitate + matematică)

Soluție propusă de Prof. Cheșcă Ciprian

Să demonstrăm pentru început că în descompunerea unui număr cumpănat, cu restricțiile impuse de problemă nu pot intra mai mult de 5 factori primi.

Să presupunem prin absurd că în descompunere intră 6 factori primi. Atunci cel mai mic număr cumpănat, scris cu 6 factori primi ar conține ca factori primi pe 2, 3, 5, 7, 11, 13 având suma 41 și ar fi $2^{36} \cdot 3^1 \cdot 5^1 \cdot 7^1 \cdot 11^1 \cdot 13^1$ care depășește restricțiile impuse de problemă.

La pasul următor deducem că suma maximă a bazelor nu depășește 50, ceea ce duce la utilizarea doar a primelor 11 numere prime $\{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47\}$.

Cu aceste valori limită generăm recursiv toate combinațiile de numere prime a căror sumă este maxim 50 (în ordine strict crescătoare), ce vor reprezenta factorii primi din descompunerea numărului cumpănat.

Când am generat o astfel de variantă, generăm prin alt mecanism recursiv toate posibilitățile de scriere (nu neapărat crescătoare) a exponentilor, ca sumă de numere naturale nenule. În final sortăm rezultatele.

Această variantă obține 100 puncte.

Varianta 3 (recursivitate + matematică)

Soluție propusă de dr. Csaba Pătcaș

Pentru a îmbunătăți timpul de rulare a variantei precedente, putem face următoarele observații:

- În primul procedeu recursiv, în care generăm factorii primi, contorizăm suma acestora, fie acesta S. Dacă $2S$ depășește limita superioară din fișierul de intrare, înseamnă că nu există nicio modalitate de a distribui exponentii în aşa fel, încât suma lor să fie cel puțin S și numărul format să fie în intervalul specificat. În acest caz ne putem întoarce din recursivitate.

- Pe baza observației de mai sus, putem precalcula cea mai mare putere a lui 2, care este mai mic decât 10^{14} , care este 46. Când ajungem cu S la o valoare mai mare decât 46, iarăși ne putem întoarce din recursivitate.

- Pentru a verifica cât mai rapid, dacă o soluție este corectă (face parte din intervalul dat), precalculăm puterile numerelor prime și folosim valorile precalculate în loc să facem înmulțirile de fiecare dată.

Varianta 4 (recursivitate + matematică) - 100 de puncte, Soluție Andrei Ciocan, Student Universitatea POLITEHNICA BUCURESTI

Se poate observa că numerele din soluție în descompunerea în factori primi nu pot conține mai mulți de 5 termeni (cu 6 termeni, am fi avut suma minima a bazei $2+3+5+7+11+13 = 33$; un astfel de număr ar fi depasit restricțiile problemei).

Se mai observă de altfel că suma factorilor primi nu poate depăși 50, astfel nu ne vor interesa numerele prime mai mari decât 50.

In continuare, cu *backtracking pe biti*, incercam să generăm submultimi de maxim 5 elemente de numere prime mai mici decât 50.

Astfel, vom stabili factorii primi, și vom încerca să generăm exponentii acestora. Aceasta o putem face tot printr-un *backtracking*, dar la fiecare nivel în recursivitate verificăm dacă suma exponentilor depășește suma bazelor. În acest caz ne oprim la acel pas și ne întoarcem în recursivitate.

25.1.2 *Rezolvare detaliată

25.1.3 Cod sursă

Listing 25.1.1: AC_cumpanit.cpp

```

1 #include <iostream>
2 #include<fstream>
3 #include<algorithm>
4

```

```

5  using namespace std;
6
7  int prim[100];
8  int nrprim= 0 ;
9  long long n ,a,b;
10 long long sol[100000];
11 int soln=0,v[10000];
12 long long x;
13 int sumbase,sumexp ;
14
15 void getprimenumbers ()
16 {
17     int i, j;
18     bool boolk;
19
20     for (i=2; i<=47 ;i++ )
21     {
22         boolk = true;
23         for (j=2 ;j<=i/2;j++)
24             if ( i % j == 0 )
25                 boolk = false;
26
27         if ( boolk )
28             prim[++nrprim]=i;
29     }
30 }
31
32 int check (long long val )
33 {
34     int i, j=1;
35     sumbase = n =0;
36     for ( i = nrprim ; i>0; i--)
37     {
38         if (val%2 == 1)
39         {
40             if (b/j>=prim[i])
41                 j*=prim[i];
42             else
43                 return 0 ;
44             n ++ ;
45             v[n]=prim[i];
46             sumbase+=prim[i];
47         }
48
49         val/=2;
50     }
51     return 1;
52 }
53
54 void back(int k)
55 {
56     if ( k == n+1 )
57     {
58         if (sumexp==sumbase)
59             sol[++soln]=x;
60
61         return ;
62     }
63
64     int i ;
65     for (i=1; b/x>=v[k] && sumexp+i<=sumbase;++i)
66     {
67         sumexp+=i;
68         x*=v[k];
69         back(k+1);
70         sumexp-=i;
71     }
72
73     for ( ; i>1 ; i--, x/=v[k]);
74 }
75
76 bool cmp (long long x, long long y ) { return x<y; }
77
78 int main()
79 {
80     int i,j;

```

```

81     ifstream f("cumpanit.in");
82     f>>a >> b;
83     getprimenumbers();
84
85     long long total= (1<<nrprim) ;
86     for (i=1;i<total ;i++)
87         if (check(i) )
88         {
89             x=1;
90             sumbase=0;
91             for (j =1 ; j<= n ;j++ )
92                 sumbase+=v[j];
93             sumexp=0;
94             if (n<=5 )
95                 back(1);
96         }
97
98     sort (sol+1 , sol + soln+1 , cmp ) ;
99
100    ofstream g("cumpanit.out");
101
102    for (i=1;i<=soln;i++)
103        if (sol[i]>=a && sol[i]<=b )
104            g <<sol[i]<<'\n';
105
106    for (i=1;i<=soln;i++)
107    {
108        sumexp=0;
109        sumbase=0;
110        for (j=1;j<=nrprim;j++)
111            if (sol[i]%prim[j]==0)
112            {
113                sumbase+=prim[j];
114                while (sol[i]%prim[j]==0)
115                {
116                    sumexp ++ ;
117                    sol[i]/=prim[j];
118                }
119            }
120
121        if ( sumexp!=sumbase)
122            g<<"sursa gresita !! " ;
123    }
124
125    g.close();
126    f.close();
127    return 0;
128 }
```

Listing 25.1.2: CC_cumpanit.cpp

```

1 // sursa Ciprian Chesca
2
3 #include <iostream>
4 #include <cmath>
5 #include <algorithm>
6
7 #define nmax 100001
8 #define smax 50
9 #define nmaxf 5
10
11 using namespace std;
12
13 ifstream f("cumpanit.in");
14 ofstream g("cumpanit.out");
15
16 int p,x,y,sumafact=0,sumaexp=0,k,ca,cb;
17 int prime[]={15,2,3,5,7,11,13,17,19,23,29,31,37,41,43,47};
18 double w=0;
19 long long cump[40000],a,b;
20 int st[nmax],fact[nmax];
21
22 int cifre(long long q)
23 {
24     int nc=0;
```

```

25     while (q>0)
26     {
27         nc++;
28         q=q/10;
29     }
30     return nc;
31 }
32
33 bool cmp (long long x, long long y) { return x<y; }
34
35 void afisare(int y)
36 {
37     int ncifre;
38     if (sumafact==sumaexp)
39     {
40         ncifre=ceil(w);
41         if (ncifre>=ca&&ncifre<=cb)
42         {
43             // calculez numarul
44             int i,j;
45             long long nr=1;
46             for(i=1;i<y;i++)
47                 for(j=1;j<=st[i];j++)
48                     nr=nr*fact[i];
49             if (nr>=a&&nr<=b) cump[++k]=nr;
50         }
51     }
52 }
53
54 void back(int y,int p)
55 {
56     int i;
57     bool ok;
58     if (y==p+1)
59         afisare(y);
60     else
61         for(i=1;i<=sumafact-sumaexp;i++)
62         {
63             st[y]=i;
64             sumaexp+=st[y];
65             w+=st[y]*log10((float)fact[y]);
66
67             ok=true;
68             if (w>cb) ok=false;
69             if (ok) back(y+1,p);
70
71             sumaexp-=st[y];
72             w-=st[y]*log10((float)fact[y]);
73         }
74 }
75
76 void factori(int x)
77 {
78     if (sumafact<=smax)
79     {
80         sumaexp=0;
81         back(1,x);
82     }
83 }
84
85 void generare(int x)
86 {
87     int i;
88     bool ok;
89     if (x<=nmaxf)
90     {
91         for(i=1;i<=prime[0];i++)
92         {
93             fact[x]=prime[i];
94             sumaexp+=fact[x];
95             ok=true;
96             if (fact[x]<=fact[x-1]&&x>1) ok=false;
97             if (ok)
98             {
99                 factori(x);
100                generare(x+1);

```

```

101         }
102         sumafact-=fact[x];
103     }
104 }
105 }
106
107 int main()
108 {
109     f>>a>>b;
110
111     ca=cifre(a);
112     cb=cifre(b);
113
114     generare(1);
115
116     sort (cump+1 , cump+k+1 , cmp) ;
117
118     for(p=1;p<=k;p++)
119         g<<cump[p]<<"\n";
120
121     f.close();
122     g.close();
123     return 0;
124 }
```

Listing 25.1.3: CP1_cumpanit.cpp

```

1 //Code by Patcas Csaba
2 //Method: Double backtracking
3 //Implementation time: 1 h 30 min
4
5 #include <vector>
6 #include <string>
7 #include <set>
8 #include <map>
9 #include <queue>
10 #include <bitset>
11 #include <stack>
12 #include <list>
13
14 #include <numeric>
15 #include <algorithm>
16
17 #include <cstdio>
18 #include <fstream>
19 #include <iostream>
20 #include <sstream>
21 #include <iomanip>
22
23 #include <cctype>
24 #include <cmath>
25 #include <ctime>
26 #include <cassert>
27
28 using namespace std;
29
30 #define LL long long
31 #define PII pair <int, int>
32 #define VB vector <bool>
33 #define VI vector <int>
34 #define VD vector <double>
35 #define VS vector <string>
36 #define VPII vector <pair <int, int> >
37 #define VVI vector < VI >
38 #define VVB vector < VB >
39
40 #define FORN(i, n) for(int i = 0; i < (n); ++i)
41 #define FOR(i, a, b) for(int i = (a); i <= (b); ++i)
42 #define FORD(i, a, b) for(int i = (a); i >= (b); --i)
43 #define FORI(it, X) for(__typeof((X).begin()) it = (X).begin(); it !=(X).end(); ++it)
44 #define REPEAT do{
45 #define UNTIL(x) }while(! (x));
46
47 #define SZ size()
48 #define BG begin()
```

```

49 #define EN end()
50 #define CL clear()
51 #define X first
52 #define Y second
53 #define RS resize
54 #define PB push_back
55 #define MP make_pair
56 #define ALL(x) x.begin(), x.end()
57
58 #define IN_FILE "cumpant.in"
59 #define OUT_FILE "cumpant.out"
60
61 ifstream fin(IN_FILE);
62 ofstream fout(OUT_FILE);
63
64 int nrPrimes;
65 LL n, m;
66 VI primes, ss, ss2;
67 vector < vector <LL> > primePowers;
68 vector <LL> solution;
69
70 void buildPrimes()
71 {
72     primes.PB(2);
73     primes.PB(3);
74     primes.PB(5);
75     primes.PB(7);
76     primes.PB(11);
77     primes.PB(13);
78     primes.PB(17);
79     primes.PB(19);
80     primes.PB(23);
81     primes.PB(29);
82     primes.PB(31);
83     primes.PB(37);
84     primes.PB(41);
85     primes.PB(43);
86     nrPrimes = primes.SZ;
87 }
88
89 void buildPrimePowers()
90 {
91     primePowers.RS(nrPrimes, vector <LL> (47));
92     FORN(i, nrPrimes)
93     {
94         LL aux = 1;
95         FOR(j, 1, 46)
96         {
97             aux *= primes[i];
98             if (aux > 1000000000000000) break;
99             primePowers[i][j] = aux;
100        }
101    }
102 }
103
104 void check(int sp)
105 {
106     LL p = 1;
107     FOR(i, 1, sp)
108     {
109         if (primePowers[ss[i]][ss2[i]] == 0) return;
110         if (primePowers[ss[i]][ss2[i]] <= (1000000000000000 / p))
111             p *= primePowers[ss[i]][ss2[i]];
112         else
113             return;
114         if (p > m) return;
115     }
116     if (p >= n)
117         solution.PB(p);
118 }
119
120
121 void doBack2(int sp2, int sp, int sum)
122 {
123     if (sp2 == sp)
124     {

```

```

125         ss2[sp2] = sum;
126         check(sp);
127         return;
128     }
129
130     FOR(i, 1, sum - sp + sp2)
131     {
132         ss2[sp2] = i;
133         doBack2(sp2 + 1, sp, sum - i);
134     }
135 }
136
137 void doBack(int sp, int sum)
138 {
139     if (sum > 46 || primePowers[0][sum] > m) return;
140     FOR(i, ss[sp - 1] + 1, nrPrimes - 1)
141     {
142         ss[sp] = i;
143         if (sum + primes[i] <= 46) doBack2(1, sp, sum + primes[i]);
144         doBack(sp + 1, sum + primes[i]);
145     }
146 }
147
148 int main()
149 {
150     //Read data
151     fin >> n >> m;
152     fin.close();
153
154     //Solve
155     buildPrimes();
156     buildPrimePowers();
157
158     //Write data
159     ss.RS(nrPrimes + 1);
160     ss2.RS(nrPrimes + 1);
161     ss[0] = -1;
162     doBack(1, 0);
163
164     sort(ALL(solution));
165     FORN(i, solution.SZ) fout << solution[i] << endl;
166     fout.close();
167
168     return 0;
169 }

```

25.2 romb

Problema 2 - romb

Noul împărat INFO al țării ONI2013 a decis să împartă țara în regiuni codificate după un algoritm stabilit prin decret. țara are formă de romb, având centrul în punctul de coordonate $(0,0)$ și lungimile semi-diagonalelor dx și dy (ca în figura 1).

împăratul alege un număr k , reprezentând numărul de etape de parcurs, astfel:

- în prima etapă, rombul inițial este împărțit în patru regiuni egale, în formă de romb, fiecare latură fiind jumătate din latura rombului inițial;
- în fiecare din celelalte $k - 1$ etape, orice romb rezultat la etapa precedentă este împărțit în alte patru romburi egale, aşa cum este descris în prima etapă.

Astfel, după k etape vom avea în total $4k$ regiuni egale, în formă de romb.

Codificarea regiunilor este făcută astfel:

- în prima etapă, rombul inițial se împarte în patru regiuni, codificate în sens trigonometric cu valorile 1, 2, 3 și 4 (ca în figura 2);
- în fiecare din celelalte etape, se reface codificarea, astfel: dacă rombul anterior avea la etapa precedentă codul X , cele patru romburi obținute

100 de puncte

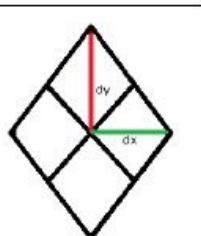


Figura 1

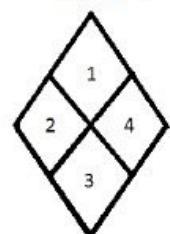


Figura 2



după divizarea curentă vor avea acum codurile $4 * X - 3$, $4 * X - 2$, $4 * X - 1$, $4 * X$ (figura 3).

Cerințe

împăratul dorește să știe după cele k etape, care este codul regiunii unde se află un oraș dat prin coordonatele (Cx, Cy) .

Date de intrare

Pe prima linie a fișierului **romb.in** se află numărul T de întrebări (seturi de date de test). Pe fiecare din următoarele T linii se află câte un set de date de test cu valorile dx , dy , k , Cx , Cy , cu semnificația anterioară, separate prin câte un spațiu.

Date de ieșire

Fișierul **romb.out** va conține T linii, pe fiecare linie i fiind răspunsul la întrebarea i , un număr natural reprezentând codul regiunii în care se află orașul de coordonate date (pentru testul i).

Restricții și precizări

- $-20\ 000 < dx, dy, Cx, Cy < 20\ 000$; $0 < k < 20$; $0 < T < 10$;
- dx și dy sunt numere naturale iar Cx și Cy sunt numere întregi;
- Se garantează că punctul de coordonate (Cx, Cy) nu se află la distanță mai mică de 10^{-7} față de latura unui romb obținut în ultima etapă.

Exemplu:

romb.in	romb.out	Explicații
2	15	Numarul de teste este $T = 2$.
10 8 2 6 -2	10	Orașul de coordonate $(6, 2)$, se află în regiunea codificată cu 15
12 16 3 -2 4		Orașul de coordonate $(-2, 4)$, se află în regiunea codificată cu 10

Timp maxim de executare/test: **0.1** secunde

Memorie: total **32 MB** din care pentru stivă **16 MB**

Dimensiune maximă a sursei: **5 KB**

25.2.1 Indicații de rezolvare

prof. Gheorghe Manolache, Colegiul Național de Informatică, Piatra-Neamț

Se poate observa că o abordare prin metoda *divide et impera* conduce la obținerea soluției de 100 de puncte.

```
inlinevoid divide(punct A, punct B, intniv_k)
{
    if (niv_k > k) return ;
    zona = 0;

    C.init_punct( A.x , B.y + (B.y - A.y) ), 
    D.init_punct( A.x + (A.x - B.x) , B.y );

    ab.media(A, B), bc.media(B, C), cd.media(C, D),
    da.media(D, A), O.media(A, C);

    v1 = verif(da, bc, p), v2 = verif(A, B, p), v3 = verif(D, C, p);
    v4 = verif(cd, ab, p), v5 = verif(D, A, p), v6 = verif(C, B, p);
    if (v1 * v2 < 0 && v4 * v5 < 0) zona = 1;
    else
        if (v1 * v3 < 0 && v4 * v5 < 0) zona = 2;
    else
```

```

    if (v1 * v3 < 0 && v4 * v6 < 0) zona = 3;
    else
        if (v1 * v2 < 0 && v4 * v6 < 0) zona = 4;
        switch (zona)
    {
        case 1LL: cod = cod * 4LL - 3LL, divide(A, ab, niv_k + 1); break;
        case 2LL: cod = cod * 4LL - 2LL, divide(da, O, niv_k + 1); break;
        case 3LL: cod = cod * 4LL - 1LL, divide(O, bc, niv_k + 1); break;
        case 4LL: cod = cod * 4LL, divide(ab, B, niv_k + 1);
    }
}

```

25.2.2 *Rezolvare detaliată

25.2.3 Cod sursă

Listing 25.2.1: romb.cpp

```

1 #include <iostream>
2 #include <cmath>
3 #include <iostream>
4 #include <cstdlib>
5
6 using namespace std;
7
8 ifstream f("romb.in");
9 ofstream g("romb.out");
10
11 typedef long double ld;
12
13 struct punct
14 {
15     ld x, y;
16     inline void init_punct(ld xx, ld yy)
17     {
18         this->x = xx; this->y = yy;
19     }
20     inline void media(punct a, punct b)
21     {
22         this->x = (a.x + b.x) / 2.0;
23         this->y = (a.y + b.y) / 2.0;
24     }
25 };
26
27 int k, v1, v2, v3, v4, v5, v6, zona, T;
28 long long cod;
29 ld eps = 1.0e-7, ok;
30 punct d, p, p1, p2;
31 punct C, D, ab, bc, cd, da, O;
32
33 inline int verif(punct a, punct b, punct h)
34 {
35     ok = a.x*b.y + b.x*h.y + h.x*a.y - b.y*h.x - h.y*a.x - a.y*b.x;
36     if (abs(ok) < eps)
37     {
38         cout << "testul nu e bun ", exit(0);
39         g << "testul nu e bun ", g.close(), exit(0);
40     }
41     return (ok > 0 ? 1 : -1 );
42 }
43
44 inline void divide(punct A, punct B, int niv_k)
45 {
46     if (niv_k > k) return ;
47     zona = 0;
48     C.init_punct( A.x , B.y + (B.y - A.y) ),
49     D.init_punct( A.x + (A.x - B.x) , B.y );
50

```

```

51     ab.media(A, B), bc.media(B, C),
52     cd.media(C, D), da.media(D, A), O.media(A, C);
53
54     v1 = verif(da, bc, p), v2 = verif(A, B, p), v3 = verif(D, C, p);
55     v4 = verif(cd, ab, p), v5 = verif(D, A, p), v6 = verif(C, B, p);
56
57     if (v1 * v2 < 0 && v4 * v5 < 0) zona = 1;
58     else
59         if (v1 * v3 < 0 && v4 * v6 < 0) zona = 2;
60         else
61             if (v1 * v3 < 0 && v4 * v6 < 0) zona = 3;
62             else
63                 if (v1 * v2 < 0 && v4 * v6 < 0) zona = 4;
64
65     switch (zona)
66     {
67         case 1LL: cod = cod * 4LL - 3LL, divide(A, ab, niv_k + 1); break;
68         case 2LL: cod = cod * 4LL - 2LL, divide(da, O, niv_k + 1); break;
69         case 3LL: cod = cod * 4LL - 1LL, divide(O, bc, niv_k + 1); break;
70         case 4LL: cod = cod * 4LL,           divide(ab, B, niv_k + 1);
71     }
72 }
73
74 int main()
75 {
76     f >> T;
77     while (T)
78     {
79         f >> d.x >> d.y >> k;
80         f >> p.x >> p.y;
81         cod = 1;
82         p1.init_punct(0.0, d.y), p2.init_punct(d.x, 0.0);
83         divide(p1, p2, 1);
84         g << cod << '\n';
85         --T;
86     }
87     g.close();
88 }
```

Listing 25.2.2: rombMN.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 int T, k, i, pminim;
6 double qx, qy, cx, cy, c1x, c1y, c2x, c2y, c3x, c3y, c4x, c4y,
7       dx, dy, d1, d2, d3, d4, minim;
8 long long sol, start;
9
10 double dist(double x1, double y1, double x2, double y2, double x3, double y3)
11 {
12     double a = y2-y1;
13     double b = x1-x2;
14     double c = -x1*(y2-y1) + y1*(x2-x1);
15     double aux = a*x3+b*y3+c;
16     return aux*aux/(a*a + b*b);
17 }
18
19 int main()
20 {
21     ifstream fin("romb.in");
22     ofstream fout("romb.out");
23
24     fin>>T;
25
26     while (T)
27     {
28         fin>>dx>>dy>>k>>qx>>qy;
29         start = 1LL << (2*(k-1));
30         cx = 0;
31         cy = 0;
32         sol = 0;
33         for (i=1;i<=k;i++)
34         {
```

```

35
36     dx /= 2; dy /= 2;
37     c1x = cx; c1y = cy + dy;
38     c2x = cx - dx; c2y = cy;
39     c3x = cx; c3y = cy - dy;
40     c4x = cx + dx; c4y = cy;
41     d1 = dist(c1x, c1y, c2x, c2y, qx, qy);
42     d2 = dist(c2x, c2y, c3x, c3y, qx, qy);
43     d3 = dist(c3x, c3y, c4x, c4y, qx, qy);
44     d4 = dist(c4x, c4y, c1x, c1y, qx, qy);
45
46     minim = d1 + d4;
47     pminim = 1;
48
49     if (minim > d1 + d2)
50     {
51         minim = d1 + d2;
52         pminim = 2;
53     }
54
55     if (minim > d2 + d3)
56     {
57         minim = d2 + d3;
58         pminim = 3;
59     }
60
61     if (minim > d3 + d4)
62     {
63         minim = d3 + d4;
64         pminim = 4;
65     }
66
67     if (pminim == 1)
68     {
69         cx = c1x;
70         cy = c1y;
71         sol += 0*start;
72     }
73
74     if (pminim == 2)
75     {
76         cx = c2x;
77         cy = c2y;
78         sol += 1*start;
79     }
80
81     if (pminim == 3)
82     {
83         cx = c3x;
84         cy = c3y;
85         sol += 2*start;
86     }
87
88     if (pminim == 4)
89     {
90         cx = c4x;
91         cy = c4y;
92         sol += 3*start;
93     }
94
95     start /= 4LL;
96 }
97
98     fout<<sol+1<<"\n";
99     T--;
100 }
101 return 0;
102 }
```

25.3 showroom

Problema 3 - showroom

100 de puncte

Un showroom din Strasbourg comercializează o gamă foarte mare de modele de autoturisme,

așezate pe n linii. Pe căte o linie se găsesc numai modele de autoturisme comercializate de același dealer. Un dealer poate avea modele pe mai multe linii. Parlamentul European dorește să-și înnoiască parcoul auto și trimite responsabilul cu activitatea de transport la showroom pentru a se informa cu privire la variantele pe care le are pentru rezolvarea acestei probleme de achiziție. Responsabilul trebuie să aleagă de la primul dealer f_1 modele, de la al doilea dealer f_2 modele, etc.

Șirul de numere f_1, f_2, f_3, \dots reprezintă termenii modulo k ai unei progresii aritmetice cu primul termen a și rația r . Dacă valoarea din șirul de numere este mai mare decât numărul de modele al dealerului corespunzător, atunci responsabilul nu mai alege nici un model al dealerului. Primul dealer este cel care are modelele pe prima linie și, eventual, pe alte linii care urmează primei linii (dar nu neapărat consecutive!), al doilea dealer este cel care are modelele pe prima linie ce conține modele diferite de cele ale primului dealer etc.

Cerințe

Să se scrie un program care determină:

- Numărul de dealeri prezenți în showroom;
- Numărul de modalități de achiziție al modelelor de către Parlamentul European, modulo 9001.

Date de intrare

Fișierul de intrare **showroom.in** conține pe prima linie numerele n, a, r, k separate prin exact un spațiu, cu semnificația de mai sus, iar pe următoarele n linii se află denumirile modelelor din enunț, separate prin unul sau mai multe spații. Fiecare linie se termină cu caracterul sfârșit de linie.

Date de ieșire

Fișierul de ieșire **showroom.out** va conține pe prima linie numărul cerut la subpunctul a), iar pe a doua linie numărul cerut la subpunctul b).

Restricții și precizări

- $1 \leq n \leq 500$;
- $1 \leq a, r, k \leq 10\,000$;
- Denumirea unui model are cel mult 20 de litere mici și/sau cifre;
- Pe o linie sunt cel mult 100 de denumiri de modele și nu pot exista mai mult de 10 spații între două modele;
- Pot exista linii cu numerele de ordine i_1, i_2, \dots, i_p cu modele ale aceluiaș dealer, astfel încât perechile de linii $(i_1, i_2), \dots, (i_{p-1}, i_p)$ au cel puțin un model de mașină în comun;
- Pentru rezolvarea corectă a fiecărei cerințe se acordă 50% din punctaj;
- Acordarea punctajului pentru a doua cerință se face numai dacă în fișierul de ieșire există un răspuns pentru prima cerință, indiferent de corectitudinea acestuia.
- Pentru 60% din teste se garantează că valoarea k este mai mică sau egală decât 10.

Exemplu:

showroom.in	showroom.out	Explicații
6 1 2 3 logan duster logan peugeot207 peugeot307 sandero sandero opelcorsa opelastra opelcorsa peugeot207 sandero duster	3 3	La showroom sunt 3 dealeri. Dealerul 1: logan, duster, sandero. Dealerul 2: peugeot207, peugeot307. Dealerul 3: opelcorsa, opelastra. Primii trei termeni din progresia aritmetică sunt 1, 3, 5. $f_1 = 1$ modulo 3 = 1; $f_2 = 3$ modulo 3 = 0; $f_3 = 5$ modulo 3 = 2. Modalitățile de achiziție ale modelelor de la dealerul 1, a niciunui model de la dealerul 2 și a două modele de la dealerul 3 sunt următoarele: logan, opelcorsa, opelastra; duster, opelcorsa, opelastra; sandero, opelcorsa, opelastra.

Timp maxim de executare/test: **0.8** secunde

Memorie: total **32 MB** din care pentru stivă **16 MB**

Dimensiune maximă a sursei: **5 KB**

25.3.1 Indicații de rezolvare

Prof. Dr. Doru Anastasiu Popescu, C.N."Radu Greceanu", Slatina

Varianta 1

Soluție propusă de Prof. Dr. Doru Anastasiu Popescu

Trebuie să construim o structură de date cu denumirile modelelor pentru fiecare dealer.

Se citește pe rand câte o linie din fișier, se identifică cuvintele distincte și se determină dealerul căreia îi aparțin modelele (dacă există). În caz afirmativ se completează dealerul cu noile modele, altfel se adaugă o un nou dealer cu aceste modele.

În felul acesta se obțin:

p dealeri cu câte x_1, x_2, \dots, x_p modele fiecare.

La punctul a) trebuie să afișăm p .

Notând cu f_1, f_2, \dots, f_p primii p termeni modulo 9001 din progresie, la punctul b) trebuie să afișăm valoarea expresiei:

$$C_{x[1]}^{f[1]} \cdot C_{x[2]}^{f[2]} \cdot \dots \cdot C_{x[p]}^{f[p]} \text{ modulo } 9001.$$

In funcție de algoritmul folosit pentru rezolvarea cerinței a) și de modul de calcul al numărului de combinări de la cerința b) se pot obține punctaje diferite.

Varianta 2

Soluție propusă de dr. Csaba Pătcaș

Pentru a îmbunătăți timpul de rulare, putem înlocui fiecare model de mașină cu un număr unic deja din fază citirii datelor. Pentru o soluție eficientă și o implementare simplă se pot folosi tipurile *set* și *map* din biblioteca *STL*.

Pentru a determina modelele care aparțin fiecărui dealer fără a parcurge liniile din fișierul de intrare de mai multe ori, putem aplica următoarea idee. Fie m numărul modelelor distincte, numerotate de la 1 la m , valori determinate în pasul precedent.

Numerotăm de la $m + 1$ la $m + n$ liniile în care sunt amplasate modelele. Pentru fiecare dintre cele $m+n$ entități obținute astfel, reținem două valori: *parent[i]*, semnificând indicele unei entități de care aparține entitatea i (sau 0, dacă nu aparține de nicio entitate), respectiv *size[i]*, numărul modelelor distincte ce aparțin entității i . Un model poate aparține unei linii, iar o linie la rândul ei poate aparține altor linii.

Când parcurgem linia j , reținem valoarea *this*, care semnifică numărul entității de care aparțin modelele din linia actuală. La început inițializăm *this* cu $m + j$. Când întâlnim un model i cu *parent[i] = 0*, asta înseamnă că este vorba de un model pe care nu l-am mai întâlnit. Setăm *parent[i] = this* și incrementăm *size[this]*.

În cazul în care *parent[i]* este diferit de zero, trebuie să unim entitățile ce aparțin de i cu entitățile ce aparțin de *this*. Urmărim valorile din *parent* pornind din i și din *this*, pentru a ajunge la entitățile cele mai importante care le aparțin (cele care au valoarea din *parent* egală cu zero), fie acestea x și y . Dacă x este diferit de y , trebuie să creăm o legătură între ele prin intermediul sirului *parent*. Pentru a obține un timp de rulare cât mai mic, comparăm *size[x]* cu *size[y]* și modificăm valoarea *parent* corespunzător celui mai mic dintre cele două.

Subpunctul a) îl putem rezolva prin iterarea peste cele n linii și găsirea entității cea mai importantă de care aparține prima mașină din linia actuală. Între timp reținem într-o mulțime entitățile la care am ajuns deja. Dacă este vorba de o entitate nouă, înseamnă că am găsit un dealer nou, introducem entitatea în mulțime și citim numărul modelelor pe care le are dealer-ul din sirul *size*.

Există mai multe abordări pentru rezolvarea subpunctului b). Ideile bazate pe construcția *triunghiului lui Pascal* depășesc timpul și / sau memoria pentru testele mari. Abordarea cu *invers modular* iarăși întâmpină probleme, din cauză că numărul 9001 nu este suficient de mare pentru a acoperi valorile din numitor la formulele de combinări pentru fiecare test.

Pentru obținerea punctajului maxim se poate folosi următoare abordare. Fie C_n^k unul dintre combinări ce trebuie calculate. Vom determina numerele prime până la cea mai mare valoare n pe care o putem întâlni (practic numărul maxim de mașini, care aparțin unui dealer), folosind metoda *ciorului lui Erastotene*. Pe urmă în sirul *power* la poziția i reținem puterea la care se află numărul prim i în valoarea C_n^k .

Pentru a determina valorile corecte din *power*, ne folosim de formula $n!/k!(n-k)!$. Prima dată iterăm cu j de la 1 la n , îl descompunem pe j în factori primi și incrementăm valoarea *power[p]* pentru fiecare factor prim p , cu exponentul la care acesta apare în descompunere. Pe urmă repetăm procedeul de la 1 la k și de la 1 la $n-k$, cu diferența că de această dată decrementăm valorile din *power*.

La sfârșit ne rămâne se iterăm de la 1 la n încă o dată și pentru fiecare i să înmulțim soluția cu $i^{power[i]}$.

25.3.2 *Rezolvare detaliată

25.3.3 Cod sursă

Listing 25.3.1: showroom_powers.cpp

```

1 //Code by Patcas Csaba
2 //Time complexity: O(nrCars * log(nrCars) + nrCompanies * maxCarsInACompany)
3 //Space complexity: O(nrCars + maxCarsInACompany)
4 //Method: Combinations with prime factors
5 //Implementation time: 1 h 30 min
6
7 #include <vector>
8 #include <string>
9 #include <set>
10 #include <map>
11 #include <queue>
12 #include <bitset>
13 #include <stack>
14 #include <list>
15
16 #include <numeric>
17 #include <algorithm>
18
19 #include <cstdio>
20 #include <fstream>
21 #include <iostream>
22 #include <sstream>
23 #include <iomanip>
24
25 #include <cctype>
26 #include <cmath>
27 #include <ctime>
28 #include <cassert>
29
30 using namespace std;
31
32 #define LL long long
33 #define PII pair <int, int>
34 #define VB vector <bool>
35 #define VI vector <int>
36 #define VD vector <double>
37 #define VS vector <string>
38 #define VPII vector <pair <int, int> >
39 #define VVI vector < VI >
40 #define VVB vector < VB >
41
42 #define FORN(i, n) for(int i = 0; i < (n); ++i)
43 #define FOR(i, a, b) for(int i = (a); i <= (b); ++i)
44 #define FORD(i, a, b) for(int i = (a); i >= (b); --i)
45 #define FORI(it, X) for(__typeof((X).begin()) it=(X).begin(); it!=(X).end(); ++it)
46 #define REPEAT do{
47 #define UNTIL(x) }while(!(x));
48
49 #define SZ size()
50 #define BG begin()
51 #define EN end()
52 #define CL clear()
53 #define X first
54 #define Y second
55 #define RS resize
56 #define PB push_back
57 #define MP make_pair
58 #define ALL(x) x.begin(), x.end()
59
60 #define IN_FILE "showroom.in"
61 #define OUT_FILE "showroom.out"

```

```

62
63 #define BASE 9001
64
65 ifstream fin(IN_FILE);
66 ofstream fout(OUT_FILE);
67
68 VVI lines, c;
69 int n, a, r, k, nrIDs;
70 map <string, int> id;
71 VI father, treeSize, companySizes, power;
72 VB isPrime;
73
74 void read()
75 {
76     nrIDs = 0;
77     fin >> n >> a >> r >> k;
78     lines.RS(n + 1);
79     FOR(i, 0, n)
80     {
81         string s;
82         getline(fin, s);
83
84         stringstream ss(s);
85         string model;
86         while (ss >> model)
87         {
88             if (!id.count(model))
89             {
90                 ++nrIDs;
91                 id[model] = nrIDs;
92             }
93             lines[i].PB(id[model]);
94         }
95     }
96     fin.close();
97
98 }
99
100 int getRoot(int node)
101 {
102     if (father[node] == 0) return node;
103     return father[node] = getRoot(father[node]);
104 }
105
106 void unite(int node1, int node2)
107 {
108     node1 = getRoot(node1);
109     node2 = getRoot(node2);
110     if (node1 == node2) return;
111     if (treeSize[node1] > treeSize[node2])
112     {
113         father[node1] = node2;
114         treeSize[node2] += treeSize[node1];
115     }
116     else
117     {
118         father[node2] = node1;
119         treeSize[node1] += treeSize[node2];
120     }
121 }
122
123 void getCompanies()
124 {
125     father.RS(nrIDs + n + 1);
126     treeSize.RS(nrIDs + n + 1);
127     FOR(i, 1, n)
128     {
129         int thisRoot = nrIDs + i;
130         FORN(j, lines[i].SZ)
131         {
132             if (father[lines[i][j]] == 0)
133             {
134                 father[lines[i][j]] = thisRoot;
135                 ++treeSize[thisRoot];
136             }
137             else

```

```

138         {
139             unite(thisRoot, lines[i][j]);
140             thisRoot = getRoot(thisRoot);
141         }
142     }
143 }
144
145 set <int> usedRoots;
146 FOR(i, 1, n)
147 {
148     int root = getRoot(lines[i][0]);
149     if (!usedRoots.count(root))
150     {
151         companySizes.PB(treeSize[root]);
152         usedRoots.insert(root);
153     }
154 }
155 }
156
157 void doErast(int limit)
158 {
159     isPrime.RS(limit + 1, true);
160     int outerLimit = sqrt(double(limit));
161     FOR(i, 2, outerLimit)
162     {
163         if (!isPrime[i]) continue;
164         for(int j = i + i; j <= limit; j += i) isPrime[j] = false;
165     }
166 }
167
168 void addPowers(int x)
169 {
170     int factor = 2;
171     while (!isPrime[x])
172     {
173         while ((x % factor) == 0)
174         {
175             ++power[factor];
176             x /= factor;
177         }
178         ++factor;
179     }
180     ++power[x];
181 }
182
183 void subPowers(int x)
184 {
185     int factor = 2;
186     while (!isPrime[x])
187     {
188         while ((x % factor) == 0)
189         {
190             --power[factor];
191             x /= factor;
192         }
193         ++factor;
194     }
195     --power[x];
196 }
197
198 int getComb(int nn, int kk)
199 {
200     power.CL, power.RS(nn + 1);
201     FOR(i, 2, nn) addPowers(i);
202     FOR(i, 2, kk) subPowers(i);
203     FOR(i, 2, nn - kk) subPowers(i);
204
205     int ans = 1;
206     FOR(i, 2, nn)
207         FORN(j, power[i]) ans = (ans * i) % BASE;
208     return ans;
209 }
210
211 int solve()
212 {
213     int solution = 1, f = a;

```

```

214     FORN(i, companySizes.SZ)
215     {
216         if (f <= companySizes[i]) solution = (solution * getComb(companySizes[i], f)) %
217             BASE;
218         f = (f + r) % k;
219     }
220     return solution;
221 }
222 int main()
223 {
224     read();
225
226     //Solve
227     getCompanies();
228     doErast( *max_element(ALL(companySizes)));
229
230     //Write data
231     fout << companySizes.SZ << endl;
232     fout << solve();
233     fout.close();
234
235     return 0;
236 }
```

25.4 flori

Problema 4 - flori

100 de puncte

Lalelele din Parcul Soarelui au fost numerotate de la 1 la n . Se dorește formarea unui buchet, care să conțină cel puțin o floare, iar două flori numerotate consecutiv să nu aparțină buchetului.

Cerințe

Fiind dat n , numărul de flori, să se determine în câte moduri se poate forma buchetul.

Date de intrare

Fișierul de intrare **flori.in** conține pe prima linie un număr natural n , reprezentând numărul de flori.

Date de ieșire

Fișierul de ieșire **flori.out** conține pe prima linie un număr natural ce reprezintă numărul de buchete modulo 9001.

Restricții și precizări

- $1 \leq n \leq 10\ 000$; • Pentru 30% din teste n este mai mic sau egal decât 26; • Pentru 60% din teste n este mai mic sau egal decât 1 000.

Exemplu:

flori.in	flori.out	Explicații
5	12	Se pot forma: {1}, {2}, {3}, {4}, {5}, {1, 3}, {1,4}, {1,5}, {1, 3, 5}, {2, 4}, {2, 5}, {3, 5}
7	33	Se pot forma : {1}, {2}, {3}, {4}, {5}, {6}, {7}, {1, 3}, {1,4}, {1,5}, {1,6}, {1,7}, {1,3,5}, {1,3,6}, {1,3,7}, {1,4,6}, {1,4,7}, {1,5,7}, {1,3,5,7}, {2, 4}, {2, 5}, {2,6}, {2,7}, {2,4,6}, {2,4,7}, {2,5,7}, {3, 5}, {3,6}, {3,7}, {3,5,7}, {4,6}, {4,7}, {5,7}

Timp maxim de executare/test: **1.0** secunde

Memorie: total **8 MB** din care pentru stivă **4 MB**

Dimensiune maximă a sursei: **5 KB**

25.4.1 Indicații de rezolvare

prof. Georgie Vlad, Liceul de Informatică, Suceava

Soluția 1, prof. Georgie Vlad, Liceul de Informatică, Suceava

Fie $T(n, k)$ numărul de submulțimi cu k elemente, ce nu conțin numere consecutive, ale mulțimii $\{1, 2, 3, \dots, n\}$.

Se demonstrează prin inducție formula de recurență:

$$T(1, 1) = 1; T(2, 1) = 2; T(2, 2) = 0;$$

$$\text{Oricare ar fi } n > 2: T(n, 1) = n; T(n, k) = T(n - 1, k) + T(n - 2, k - 1); 1 < k \leq n$$

Valorile T se rețin și se calculează utilizând 3 vectori. Se rețin resturile mod 9001.

Numărul cerut, pentru n dat, se calculează ca suma (mod 9001)

Soluția 2, prof. dr. Popescu Doru Anastasiu, Colegiul Național "Radu Greceanu", Slatina

Pentru 30 puncte se poate scrie un subprogram recursiv care determină toate buchete și le numără, modulo 9001.

Pentru 100 puncte putem să folosim o relație de recurență astfel:

$a[i]$ = numărul de buchete care se pot forma având cel mai mare număr pentru o floare egal cu i .

$$a[1] = a[2] = 1;$$

$a[i]$ = numărul de buchete care se pot forma cu cel mai mare număr pentru o floare cel mult egal cu $i - 2$. Acest lucru se scrie astfel:

$$a[i] = 1 + j = 1i - 2a[j], \text{ pentru } i = 3, \dots, n. \text{ Suma se calculeaza modulo 9001.}$$

Numărul căutat este $a[1] + a[2] + \dots + a[n]$ modulo 9001.

Soluția 3, prof. Cheșcă Ciprian, Liceul Tehnologic "Costin Nenițescu" Buzău

Se observă că numărul căutat este termenul $n + 2$ al sirului Fibonacci din care se scade 1.

Acest rezultat poate fi demonstrat cu ajutorul relației:

$F_n = \text{comb}(n, 0) + \text{comb}(n - 1, 1) + \text{comb}(n - 2, 2) + \dots$ unde F_n reprezintă termenul n al sirului Fibonacci.

25.4.2 *Rezolvare detaliată

25.4.3 Cod sursă

Listing 25.4.1: CC_flori.cpp

```

1 // sursa cu Fibonacci - Chesca Ciprian
2
3 #include <iostream>
4
5 #define M 9001
6 #define nmax 1000001
7
8 using namespace std;
9
10 int n;
11 int fibo[nmax];
12
13 ifstream f("flori.in");
14 ofstream g("flori.out");
15
16 int main()
17 {
18     int i;
19
20     f>>n;
21     fibo[1]=1;fibo[2]=1;
22     for(i=3;i<=n+2;i++)
23         fibo[i]=(fibo[i-1]%M+fibo[i-2]%M)%M;
24

```

```

25     g<<fibonacci[n+2]-1<<"\n";
26
27     f.close();
28     g.close();
29
30     return 0;
31 }
```

Listing 25.4.2: DPA1_flori.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 ifstream fin("flori.in");
6 ofstream fout("flori.out");
7
8 int x[1005], n, nr;
9
10 void gen(int k)
11 {
12     int i;
13     nr=(nr+1)%9001;
14     if(k<=n)
15         for(i=x[k-1]+2; i<=n; i++)
16         {
17             x[k]=i;
18             gen(k+1);
19         }
20 }
21
22 void cit()
23 {
24     fin>>n;
25     fin.close();
26 }
27
28 int main()
29 {
30     x[0]=-1;
31     nr=-1;
32     cit();
33     gen(1);
34     fout<<nr;
35     fout.close();
36     return 0;
37 }
```

Listing 25.4.3: DPA2_flori.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 ifstream fin("flori.in");
6 ofstream fout("flori.out");
7
8 int a[10005], n, nr;
9
10 void cit()
11 {
12     fin>>n;
13     fin.close();
14 }
15
16 int main()
17 {
18     cit();
19     a[1]=1;
20     a[2]=1;
21     int i, j;
22     for(i=3; i<=n; i++)
23     {
24         a[i]=1;
```

```

25         for(j=1;j<=i-2;j++)
26             a[i]=(a[i]+a[j])%9001;
27     }
28
29     nr=0;
30     for(i=1;i<=n;i++)
31         nr=(nr+a[i])%9001;
32
33     fout<<nr;
34     fout.close();
35     return 0;
36 }
```

Listing 25.4.4: flori_MN.cpp

```

1 #include <iostream>
2 #include <string.h>
3
4 using namespace std;
5
6 #define DIM 10100
7 #define MOD 9001
8
9 int A[DIM];
10 int S[DIM];
11 int SA[DIM];
12
13 int n, i, j, sol;
14
15 int main()
16 {
17     ifstream fin("flori.in");
18     ofstream fout("flori.out");
19
20     fin>>n;
21     for (i=1;i<=n;i++)
22     {
23         A[i] = 1;
24         sol+=A[i];
25         sol %= MOD;
26         SA[i] = (SA[i-1] + A[i]) % MOD;
27     }
28
29     for (i=2;i<=n;i++)
30     {
31         A[1] = 0;
32         for (j=2;j<=n;j++)
33         {
34             A[j] = SA[j-2];
35             sol+=A[j];
36             sol %= MOD;
37             S[j] = (S[j-1] + A[j]) % MOD;
38         }
39
40         memcpy(SA, S, sizeof(S));
41     }
42
43     fout<<sol<<"\n";
44     return 0;
45 }
```

Listing 25.4.5: flori_VG.cpp

```

1 #include <iostream>
2 #include<iostream>
3 #include<algorithm>
4 #include<vector>
5
6 int a1[10000],a2[10000],a3[10000];
7
8 using namespace std;
9
10 ifstream f("flori.in");
11 ofstream g("flori.out");
```

```

12 int sum(int n)
13 {
14     int i,s=0;
15     for(i=1;i<=n;i++)
16         s=(s+a3[i])%9001;
17     return s;
18 }
19 }
20
21 int main()
22 {
23     int n,i,k;
24     f>>n;
25     a1[1]=1;
26     a2[1]=2;
27     a2[2]=0;
28     for(i=3;i<=n;i++)
29     {
30         a3[i]=0;
31         a3[1]=i;
32         for(k=i-1;k>1;k--)
33             a3[k]=(a2[k]+a1[k-1])%9001;
34
35         for(k=1;k<=i;k++)
36             a1[k]=a2[k], a2[k]=a3[k];
37     }
38
39     g<<sum(n)<<' \n';
40     return 0;
41 }
```

25.5 taxa

Problema 5 - taxa

100 de puncte

Miruna se pregătește de vacanța de vară. Ea a hotărât deja că împreună cu un grup de colegi să facă o excursie în regatul INFO unde moneda locală se numește BOSS. A studiat deja harta acestei zone și a aflat multe lucruri interesante. Ea știe că regatul se află pe o insulă cu suprafața uscatului sub forma dreptunghiulară ce poate fi reprezentată ca o matrice cu N linii și M coloane în care fiecare element este un cod pentru un tip de obiectiv turistic ce poate fi vizitat. Deoarece sosirea și plecarea de pe insulă se face cu avionul, ea cunoaște poziția (l_0, c_0) unde va fi debarcată și poziția (l_f, c_f) unde va fi plecarea de pe insulă. Ea se poate deplasa pentru vizitarea obiectivelor turistice doar în celule vecine pe cele opt direcții (N, S, E, V, NE, NV, SE, SV), iar dacă noua poziție are alt cod decât cel din care venise la pasul precedent, atunci trebuie să plătească o taxa de vizitare egală cu produsul codurilor celor două zone (exprimată tot în moneda locală, BOSS!!!). Miruna ar dori să afle care ar fi suma minimă necesară pentru a se deplasa până la locul de plecare de pe insulă.

Cerințe

Dându-se configurația regatului și pozițiile de plecare și sosire, să se determine suma minimă necesară deplasării.

Date de intrare

Pe prima linie a fișierului **taxa.in** se află valorile naturale N, M, l_0, c_0, l_f, c_f . Pe următoarele N linii se află câte M elemente, codurile fiecărei zone, numere naturale separate prin câte un spațiu.

Date de ieșire

Fișierul **taxa.out** va conține un număr natural B , reprezentând suma minimă necesară deplasării.

Restricții și precizări

- $0 < N, M < 1001$
- Obiectivele au coduri numere naturale nenule mai mici sau egale cu 5, iar poziția inițială și finală sunt distințe;
- Pentru 30% din teste vom avea $N, M \leq 100$;
- Pentru 20% din teste matricea conține numai 2 valori.

Exemple:

taxa.in	taxa.out	Explicații
<pre>5 5 1 1 4 5 1 1 2 2 2 1 2 3 3 3 1 1 3 3 3 2 2 2 2 2 1 1 1 2 1</pre>	2	Suma minimă necesară deplasării din (1,1) în (4,5) este de 2 BOSSI.

Timp maxim de executare/test: **0.5** secunde

Memorie: total **36 MB** din care pentru stivă **18 MB**

Dimensiune maximă a sursei: **5 KB**

25.5.1 Indicații de rezolvare

prof. Gheorghe Manolache, Colegiul Național de Informatică, Piatra-Neamț

O soluție care obține 100 de puncte se poate obține cu unele optimizări.

Să definim o "grupa" ca fiind o zonă care are același cod, și care are proprietatea că aici se poate ajunge plecând din grupa de start. Având în vedere că toate codurile sunt numere până în 5, deducem că în cel mai rău caz avem costul de trecere de la o grupă la alta cu valoarea 20. Deci costul maxim de a ajunge de pe start la final va fi $20*(N+M)$, adică 40 000.

Reținem maxim 40 000 de cozi, în a i -a coadă punând toate pozițiile care au costul i .

Când am ajuns pe o poziție din $coada[cost_{actual}]$, și nu a fost calculat costul ei, atunci evident că pentru acea poziție costul ei va fi indicele actualei cozi. Deci facem *fill* din poziția actuală, umplem toată "grupa" cu costul actual, și punem toate pozițiile care nu au același cod cu poziția actuală în $coada[cost_{actual} + cost_{intrepozitiaactuală_i_pozitie nouă}]$.

Complexitatea finală este $O(n * m)$ cu o constantă în jur de 5-6.

Se poate aplica și *algoritmul lui LEE* cu costuri dar această implementare nu obține punctajul maxim.

25.5.2 *Rezolvare detaliată

25.5.3 Cod sursă

Listing 25.5.1: taxa_GM1.cpp

```

1 #include <iostream>
2 #include <queue>
3
4 using namespace std;
5
6 ifstream fin("taxa.in");
7 ofstream fout("taxa.out");
8
9 struct nod{ int a,b;};
10
11 #define CMAX 40010
12
13 const short dx[] = {0, 1, 0, -1, 1, 1, -1, -1};
14 const short dy[] = {1, 0, -1, 0, 1, -1, 1, -1};
15 short d[1010][1010];
16 short c[1010][1010];
17 int N, M, ii, xx;
18 nod p1, p2, bb, w, t;
19
20 queue<nod> Q[CMAX];
21
22 inline short prod(short A, short B)
23 {
    if (A == B) return 0;

```

```

25     return A * B;
26 }
27
28 inline bool este(short i, short j)
29 {
30     if (1 <= i && i <= N && 1 <= j && j <= M) return true;
31     return false;
32 }
33
34 inline void Fill(short i, short j, short cost)
35 {
36     if (este(i, j) && c[i][j] == -1)
37     {
38         if (d[i][j] == xx)
39         {
40             c[i][j] = cost;
41             for (int k = 0; k < 8; k++)
42                 Fill(i + dx[k], j + dy[k], cost);
43         }
44         else
45             bb.a = i, bb.b = j, Q[cost + prod(d[i][j], xx)].push(bb);
46     }
47 }
48
49 int main()
50 {
51     fin >> N >> M >> p1.a >> p1.b >> p2.a >> p2.b;
52     for (int i = 1; i <= N; i++)
53         for (int j = 1; j <= M; j++)
54             fin >> d[i][j], c[i][j] = -1;
55
56     Q[0].push(p1);
57     for (ii = 0; ii < CMAX; ii++)
58     {
59         for (; !Q[ii].empty(); Q[ii].pop())
60         {
61             w = Q[ii].front(); xx = d[w.a][w.b];
62             if (c[w.a][w.b] == -1)
63                 Fill(w.a, w.b, ii);
64             if (c[p2.a][p2.b] != -1)
65             {
66                 fout << c[p2.a][p2.b] << '\n';
67                 fout.close();
68                 return 0;
69             }
70         }
71     }
72     return 0;
73 }
```

Listing 25.5.2: taxa_GM2.cpp

```

1 #include <cstdio>
2 #include <queue>
3
4 using namespace std;
5
6 struct nod{ int a,b; };
7
8 #define CMAX 40010
9 const short dx[] = {0, 1, 0, -1, 1, 1, -1, -1};
10 const short dy[] = {1, 0, -1, 0, 1, -1, 1, -1};
11 int d[1010][1010];
12 int c[1010][1010];
13 int N, M, ii, xx;
14 nod p1, p2, bb, w, t;
15 queue<nod> Q[CMAX];
16
17 inline short prod(short A, short B)
18 {
19     if (A == B) return 0;
20     return A * B;
21 }
22
23 inline bool este(short i, short j)
```

```

24  {
25      if (1 <= i && i <= N && 1 <= j && j <= M) return true;
26      return false;
27  }
28
29  inline void Fill(short i, short j, short cost)
30  {
31      if (este(i, j) && c[i][j] == -1)
32      {
33          if (d[i][j] == xx)
34          {
35              c[i][j] = cost;
36              for (int k = 0; k < 8; k++)
37                  Fill(i + dx[k], j + dy[k], cost);
38          }
39          else
40              bb.a = i, bb.b = j, Q[cost + prod(d[i][j], xx)].push(bb);
41      }
42  }
43
44  int main()
45  {
46      freopen("taxa.out", "w", stdout);
47      freopen("taxa.in", "r", stdin);
48
49      scanf("%d%d%d%d%d", &N, &M, &p1.a, &p1.b, &p2.a, &p2.b);
50      for (int i = 1; i <= N; i++)
51          for (int j = 1; j <= M; j++)
52              scanf("%d", &d[i][j]), c[i][j] = -1;
53
54      Q[0].push(p1);
55      for (ii = 0; ii < CMAX; ii++)
56      {
57          for (; !Q[ii].empty(); Q[ii].pop())
58          {
59              w = Q[ii].front(); xx = d[w.a][w.b];
60              if (c[w.a][w.b] == -1)
61                  Fill(w.a, w.b, ii);
62              if (c[p2.a][p2.b] != -1)
63              {
64                  printf("%d\n", c[p2.a][p2.b]);
65                  return 0;
66              }
67          }
68      }
69      return 0;
70  }

```

Listing 25.5.3: taxa_GM3.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <queue>
4
5 using namespace std;
6
7 ifstream fin("taxa.in");
8 ofstream fout("taxa.out");
9
10 #define INF 0x3f3f3f3f
11
12 struct nod{int a,b;};
13
14 const short dx[] = {0, 1, 0, -1, 1, 1, -1, -1};
15 const short dy[] = {1, 0, -1, 0, 1, -1, 1, -1};
16
17 int N, M;
18 int d[1010][1010];
19 int c[1010][1010];
20
21 queue< nod > Q;
22 nod p1, p2, w, t;
23
24 inline int prod(int A, int B)
25 {

```

```

26     if (A == B) return 0;
27     return A * B;
28 }
29
30 int main()
31 {
32     fin >> N >> M >> p1.a >> p1.b >> p2.a >> p2.b;
33     for (int i = 1; i <= N; i++)
34         for (int j = 1; j <= M; j++)
35             fin >> d[i][j], c[i][j] = INF;
36
37     c[p1.a][p1.b] = 0;
38     Q.push(p1);
39     for (; !Q.empty(); Q.pop())
40     {
41         w = Q.front();
42         for (int k = 0; k < 8; k++)
43         {
44             t = w;
45             t.a += dx[k]; t.b += dy[k];
46             if (c[t.a][t.b] > c[w.a][w.b] + prod(d[w.a][w.b], d[t.a][t.b]))
47                 (c[t.a][t.b] = c[w.a][w.b] + prod(d[w.a][w.b], d[t.a][t.b])), 
48                 Q.push(t);
49         }
50     }
51
52     fout << c[p2.a][p2.b] << '\n';
53     fout.close();
54     return 0;
55 }
```

Listing 25.5.4: taxa_MN1.cpp

```

43             d[i][j] > d[iv][jv] + a[iv][jv]*a[i][j])
44         {
45             d[i][j] = d[iv][jv] + a[iv][jv]*a[i][j];
46             ok = 0;
47         }
48
49         if (iv>0 && iv<=n && jv > 0 && jv<=m &&
50             a[iv][jv]==a[i][j] && d[i][j] > d[iv][jv])
51         {
52             d[i][j] = d[iv][jv];
53             ok = 0;
54         }
55     }
56 } while (!ok);
57
58 fout<<d[i2][j2]<<"\n";
59
60 return 0;
}

```

Listing 25.5.5: taxa_MN2.cpp

```

1 #include <iostream>
2
3 #define DIM 1010
4 #define INF 40040
5
6 using namespace std;
7
8 int n, m, i1, i2, j1, j2, i, j, ok, dir, iv, jv, u, u1, p;
9
10 int di[] = {0,0,-1,1,-1,1,-1,1};
11 int dj[] = {-1,1,0,0,1,1,-1,-1};
12
13 int a[DIM][DIM];
14 int d[DIM][DIM];
15 int v[DIM][DIM];
16
17 int c[2][DIM*DIM];
18 int c1[2][DIM*DIM];
19
20 int main()
21 {
22     ifstream fin("taxa.in");
23     ofstream fout("taxa.out");
24     fin>>n>>m>>i1>>j1>>i2>>j2;
25     for (i=1;i<=n;i++) {
26         for (j=1;j<=m;j++) {
27             fin>>a[i][j];
28             d[i][j] = INF;
29         }
30     }
31     d[i1][j1] = 0;
32     c[0][1] = i1;
33     c[1][1] = j1;
34 //    v[i1][j1] = 0;
35     u = 1;
36     do
37     {
38         ok = 1;
39         u1 = 0;
40         for (p=1;p<=u;p++)
41         {
42             i = c[0][p];
43             j = c[1][p];
44             for (dir=0;dir<=7;dir++)
45             {
46                 iv = i+di[dir];
47                 jv = j+dj[dir];
48                 if (iv>0 && iv<=n && jv > 0 && jv<=m &&
49                     a[iv][jv]!=a[i][j] &&
50                     d[iv][jv] > d[i][j] + a[iv][jv]*a[i][j])
51                 {
52                     d[iv][jv] = d[i][j] + a[iv][jv]*a[i][j];
53                     if (v[iv][jv] == 0)
54                     {

```

```

55             v[iv][jv] = 1;
56             c1[0][++u1] = iv;
57             c1[1][u1] = jv;
58         }
59         ok = 0;
60     }
61
62     if (iv>0 && iv<=n && jv > 0 && jv<=m &&
63         a[iv][jv]==a[i][j] && d[iv][jv] > d[i][j])
64     {
65         d[iv][jv] = d[i][j];
66         if (v[iv][jv] == 0)
67         {
68             v[iv][jv] = 1;
69             c1[0][++u1] = iv;
70             c1[1][u1] = jv;
71         }
72         ok = 0;
73     }
74 }
75
76
77     for (p=1;p<=u1;p++)
78     {
79         c[0][p] = c1[0][p];
80         c[1][p] = c1[1][p];
81         v[c[0][p]][c[1][p]] = 0;
82 //         c[0][p] = 0;
83 //         c[1][p] = 0;
84     }
85     u = u1;
86 } while (!ok);
87
88 fout<<d[i2][j2]<<"\n";
89 return 0;
90 }
```

Listing 25.5.6: taxa_PC1.cpp

```

1 //Code by Patcas Csaba
2 //Time complexity: O(n * m * log(n * m))
3 //Space complexity: O(n * m)
4 //Method: Heap
5 //Implementation time: 30 minutes
6
7 #include <vector>
8 #include <string>
9 #include <set>
10 #include <map>
11 #include <queue>
12 #include <bitset>
13 #include <stack>
14 #include <list>
15
16 #include <numeric>
17 #include <algorithm>
18
19 #include <cstdio>
20 #include <fstream>
21 #include <iostream>
22 #include <sstream>
23 #include <iomanip>
24
25 #include <cctype>
26 #include <cmath>
27 #include <ctime>
28 #include <cassert>
29
30 using namespace std;
31
32 #define LL long long
33 #define PII pair <int, int>
34 #define VB vector <bool>
35 #define VI vector <int>
36 #define VD vector <double>
```

```

37 #define VS vector <string>
38 #define VPII vector <pair <int, int> >
39 #define VVI vector < VI >
40 #define VVB vector < VB >
41
42 #define FORN(i, n) for(int i = 0; i < (n); ++i)
43 #define FOR(i, a, b) for(int i = (a); i <= (b); ++i)
44 #define FORD(i, a, b) for(int i = (a); i >= (b); --i)
45 #define FORI(it, X) for(__typeof((X).begin())it =(X).begin();it!=(X).end();++it)
46 #define REPEAT do{
47 #define UNTIL(x) }while(!(x));
48
49 #define SZ size()
50 #define BG begin()
51 #define EN end()
52 #define CL clear()
53 #define X first
54 #define Y second
55 #define RS resize
56 #define PB push_back
57 #define MP make_pair
58 #define ALL(x) x.begin(), x.end()
59
60 #define IN_FILE "taxa.in"
61 #define OUT_FILE "taxa.out"
62
63 ifstream fin(IN_FILE);
64 ofstream fout(OUT_FILE);
65
66 int n, m, xs, xf, ys, yf;
67 int dx[8] = {-1, -1, -1, 0, 0, 1, 1, 1};
68 int dy[8] = {-1, 0, 1, -1, 1, -1, 0, 1};
69 VVI a;
70 VVB was;
71 priority_queue < pair <int, PII>,
72                     vector <pair <int, PII> >,
73                     greater<pair <int, PII> > > heap;
74
75 int solve()
76 {
77     was.RS(n + 1, VB(m + 1));
78     heap.push(MP(0, MP(xs, ys)));
79     was[xs][ys] = true;
80     while (1)
81     {
82         int cost = heap.top().X;
83         PII coord = heap.top().Y;
84         if (coord.X == xf && coord.Y == yf) return cost;
85         heap.pop();
86         FORN(i, 8)
87         {
88             int newX = coord.X + dx[i];
89             int newY = coord.Y + dy[i];
90             if (newX < 1 || newX > n || newY < 1 || newY > m) continue;
91             if (was[newX][newY]) continue;
92             was[newX][newY] = true;
93             int newCost;
94             if (a[coord.X][coord.Y] != a[newX][newY])
95                 newCost = cost + a[coord.X][coord.Y] * a[newX][newY];
96             else newCost = cost;
97             heap.push(MP(newCost, MP(newX, newY)));
98         }
99     }
100 }
101
102 int main()
103 {
104     //Read data
105     fin >> n >> m >> xs >> ys >> xf >> yf;
106     a.RS(n + 1, VI(m + 1));
107     FOR(i, 1, n)
108         FOR(j, 1, m) fin >> a[i][j];
109     fin.close();
110
111     //Solve
112

```

```

113     //Write data
114     fout << solve();
115     fout.close();
116
117     return 0;
118 }
```

Listing 25.5.7: taxa_PC2.cpp

```

1 //Code by Patcas Csaba
2 //Time complexity: O(n * m)
3 //Space complexity: O(n * m)
4 //Method: Fill + BF
5 //Implementation time: 40 minutes
6 //Only works for the two color case
7
8 #include <vector>
9 #include <string>
10 #include <set>
11 #include <map>
12 #include <queue>
13 #include <bitset>
14 #include <stack>
15 #include <list>
16
17 #include <numeric>
18 #include <algorithm>
19
20 #include <cstdio>
21 #include <fstream>
22 #include <iostream>
23 #include <sstream>
24 #include <iomanip>
25
26 #include <cctype>
27 #include <cmath>
28 #include <ctime>
29 #include <cassert>
30
31 using namespace std;
32
33 #define LL long long
34 #define PII pair <int, int>
35 #define VB vector <bool>
36 #define VI vector <int>
37 #define VD vector <double>
38 #define VS vector <string>
39 #define VPII vector <pair <int, int> >
40 #define VVI vector < VI >
41 #define VVB vector < VB >
42
43 #define FORN(i, n) for(int i = 0; i < (n); ++i)
44 #define FOR(i, a, b) for(int i = (a); i <= (b); ++i)
45 #define FORD(i, a, b) for(int i = (a); i >= (b); --i)
46 #define FORI(it, X) for(__typeof((X).begin()) it=(X).begin(); it!=(X).end(); ++it)
47 #define REPEAT do{
48 #define UNTIL(x) }while(!(x));
49
50 #define SZ size()
51 #define BG begin()
52 #define EN end()
53 #define CL clear()
54 #define X first
55 #define Y second
56 #define RS resize
57 #define PB push_back
58 #define MP make_pair
59 #define ALL(x) x.begin(), x.end()
60
61 #define IN_FILE "taxa.in"
62 #define OUT_FILE "taxa.out"
63
64 ifstream fin(IN_FILE);
65 ofstream fout(OUT_FILE);
66
```

```

67 int n, m, xs, xf, ys, yf, nrZones, color, color1 = -1, color2 = -1;
68 VVI a, b;
69 vector <set <int> > g;
70
71 void addEdge(int node1, int node2)
72 {
73     if (node1 == node2) return;
74     g[node1].insert(node2);
75     g[node2].insert(node1);
76 }
77
78 void fill(int x, int y)
79 {
80     if (x < 1 || x > n || y < 1 || y > m) return;
81     if (b[x][y] != 0)
82     {
83         addEdge(nrZones, b[x][y]);
84         return;
85     }
86     if (a[x][y] != color) return;
87     b[x][y] = nrZones;
88     fill(x - 1, y - 1);
89     fill(x - 1, y);
90     fill(x - 1, y + 1);
91     fill(x, y - 1);
92     fill(x, y + 1);
93     fill(x + 1, y - 1);
94     fill(x + 1, y);
95     fill(x + 1, y + 1);
96 }
97
98 void getZones()
99 {
100    b.RS(n + 1, VI(m + 1));
101    nrZones = 0;
102    FOR(i, 1, n)
103        FOR(j, 1, m)
104            if (b[i][j] == 0)
105            {
106                ++nrZones;
107                g.RS(nrZones + 1);
108                color = a[i][j];
109                fill(i, j);
110            }
111    }
112
113 int doBF(int startNode, int endNode)
114 {
115     queue <int> fifo;
116     VI dist(nrZones + 1, -1);
117     dist[startNode] = 0;
118     fifo.push(startNode);
119     while (!fifo.empty() && dist[endNode] == -1)
120     {
121         int head = fifo.front();
122         for (set<int>::iterator it = g[head].BG; it != g[head].EN; ++it)
123         {
124             int node = (*it);
125             if (dist[node] != -1) continue;
126             dist[node] = dist[head] + 1;
127             fifo.push(node);
128         }
129         fifo.pop();
130     }
131     return dist[endNode];
132 }
133
134 int main()
135 {
136     //Read data
137     fin >> n >> m >> xs >> ys >> xf >> yf;
138     a.RS(n + 1, VI(m + 1));
139     FOR(i, 1, n)
140         FOR(j, 1, m)
141         {
142             fin >> a[i][j];

```

```

143         if (color1 == -1) color1 = a[i][j];
144         else
145             if (color1 != a[i][j]) color2 = a[i][j];
146     }
147     fin.close();
148
149     //Solve
150     getZones();
151
152     //Write data
153     fout << doBF(b[xs][ys], b[xf][yf]) * color1 * color2;
154     fout.close();
155
156     return 0;
157 }
```

Listing 25.5.8: taxa_PC3.cpp

```

1 //Code by Patcas Csaba
2 //Time complexity: O(n * m * log(n * m))
3 //Space complexity: O(n * m)
4 //Method: Heap
5 //Implementation time: 30 minutes
6
7 #include <vector>
8 #include <string>
9 #include <set>
10 #include <map>
11 #include <queue>
12 #include <bitset>
13 #include <stack>
14 #include <list>
15
16 #include <numeric>
17 #include <algorithm>
18
19 #include <cstdio>
20 #include <fstream>
21 #include <iostream>
22 #include <sstream>
23 #include <iomanip>
24
25 #include <cctype>
26 #include <cmath>
27 #include <ctime>
28 #include <cassert>
29
30 using namespace std;
31
32 #define LL long long
33 #define PII pair <int, int>
34 #define VB vector <bool>
35 #define VI vector <int>
36 #define VD vector <double>
37 #define VS vector <string>
38 #define VPII vector <pair <int, int> >
39 #define VVI vector < VI >
40 #define VVB vector < VB >
41
42 #define FORN(i, n) for(int i = 0; i < (n); ++i)
43 #define FOR(i, a, b) for(int i = (a); i <= (b); ++i)
44 #define FORD(i, a, b) for(int i = (a); i >= (b); --i)
45 #define FORI(it, X) for(__typeof((X).begin()) it=(X).begin(); it!=(X).end(); ++it)
46 #define REPEAT do{
47 #define UNTIL(x) }while(!(x));
48
49 #define SZ size()
50 #define BG begin()
51 #define EN end()
52 #define CL clear()
53 #define X first
54 #define Y second
55 #define RS resize
56 #define PB push_back
57 #define MP make_pair
```

```

58 #define ALL(x) x.begin(), x.end()
59
60 #define IN_FILE "taxa.in"
61 #define OUT_FILE "taxa.out"
62
63 ifstream fin(IN_FILE);
64 ofstream fout(OUT_FILE);
65
66 int n, m, xs, xf, ys, yf;
67 int dx[8] = {-1, -1, -1, 0, 0, 1, 1, 1};
68 int dy[8] = {-1, 0, 1, -1, 1, -1, 0, 1};
69 VVI a, b;
70 VVB was;
71 priority_queue < pair <int, PII>,
72                 vector <pair <int, PII> >,
73                 greater<pair <int, PII> > > heap;
74
75 int solve()
76 {
77     was.RS(n + 1, VB(m + 1));
78     b.RS(n + 1, VI(m + 1, -1));
79     heap.push(MP(0, MP(xs, ys)));
80     was[xs][ys] = true;
81     while (1)
82     {
83         int cost = heap.top().X;
84         PII coord = heap.top().Y;
85         heap.pop();
86         if (coord.X == xf && coord.Y == yf) return cost;
87         FORN(i, 8)
88         {
89             int newX = coord.X + dx[i];
90             int newY = coord.Y + dy[i];
91             if (newX < 1 || newX > n || newY < 1 || newY > m) continue;
92             //if (was[newX][newY]) continue;
93             //was[newX][newY] = true;
94             int newCost;
95             if (a[coord.X][coord.Y] != a[newX][newY])
96                 newCost = cost + a[coord.X][coord.Y] * a[newX][newY];
97             else newCost = cost;
98             if (newCost < b[newX][newY] || b[newX][newY] == -1)
99             {
100                 b[newX][newY] = newCost;
101                 heap.push(MP(newCost, MP(newX, newY)));
102             }
103         }
104     }
105 }
106
107 int main()
108 {
109     //Read data
110     fin >> n >> m >> xs >> ys >> xf >> yf;
111     a.RS(n + 1, VI(m + 1));
112     FOR(i, 1, n)
113         FOR(j, 1, m) fin >> a[i][j];
114     fin.close();
115
116     //Solve
117
118     //Write data
119     fout << solve();
120     fout.close();
121
122     return 0;
123 }
```

25.6 zone

Problema 6 - zone

100 de puncte

Scooby-Doo, celebrul personaj de desene animate, a intrat iar în bucluc. Acesta se găsește acum într-o cameră dreptunghiulară de dimensiuni $n \times m$ alcătuită din celule pătratice de latură

1, divers colorate.

Se definesc :

- Zonă - ca fiind un grup cu număr maxim de celule de aceeași culoare, adiacente pe linie sau pe coloană.
- SUPERZONA - ca fiind o mulțime A de zone, cu proprietatea că fiecare are cel puțin k zone vecine aflate în aceeași mulțime A .

Pentru a fi în siguranță, Scooby-Doo trebuie să se adăpostească într-o SUPERZONĂ cu număr maxim de celule din matrice.

Cerințe

Părăsit de prietenii săi, Scooby-Doo nu se descurcă de unul singur și vă roagă pe voi să rezolvați misterul și să calculați numărul de celule al celei mai mari SUPERZONE din matrice.

Date de intrare

Fișierul de intrare **zone.in** conține pe prima linie numerele naturale n , m și k separate prin câte un spațiu. Pe următoarele n linii se află câte m caractere, fără spații între ele, reprezentând culoarea respectivei celule din matrice.

Date de ieșire

Fișierul de ieșire **zone.out** va conține pe prima linie un număr natural ce reprezintă numărul de celule al celei mai mari SUPERZONE din matrice.

Restricții și precizări

- $2 \leq n, m, k \leq 300$
- Culorile sunt reprezentate prin litere mici ale alfabetului englez ('a' - 'z');
- Două celule se consideră adiacente dacă au o latură comună;
- Două zone se consideră vecine dacă au cel puțin câte o celulă adiacentă.

Exemple:

zone.in	zone.out	Explicații
4 3 2 aaa bad baa cda	11	Este selectată toată matricea fără zona formată din litera "d" de pe ultima coloană.

Timp maxim de executare/test: **0.3** secunde

Memorie: total **16 MB** din care pentru stivă **8 MB**

Dimensiune maximă a sursei: **5 KB**

25.6.1 Indicații de rezolvare

Andrei Ciocan, Student Universitatea POLITEHNICA București

Prima etapă a rezolvării constă în delimitarea zonelor și etichetarea lor cu numere naturale printr-un *algoritm de fill*.

În continuare, vom calcula pentru fiecare zonă numărul de zone vecine. Intuitiv, vom elibera din matrice zonele care nu au k vecini, dar numai acest pas nu duce la rezolvarea corectă a problemei.

După eliminarea unei zone, trebuie verificat pentru fiecare zonă vecină cu ea (încă neeliminată) dacă numărul de vecini a scăzut sub k . În acest caz, acea zonă trebuie eliminată ulterior.

Algoritm de rezolvare: inițial toate zonele cu numărul de vecini mai mic decât k se adaugă la o *coadă*. Apoi se parcurge coada, iar pentru fiecare zonă X procesată, se adaugă la coadă toate zonele care prin eliminarea zonei X vor avea mai puțin de k vecini.

Algoritmul continuă cât timp coada nu este goală.

În final, trebuie să alegem din superzonele rămase, cea de dimensiune maximă (pot fi mai multe).

Complexitate oficială : $O(n^2)$ timp și spațiu.

Soluții de complexitate mai mari, $O(n^2 \log n)$ folosind *heapuri* sau $O(n^3)$ de asemenea se încadrează în timp, cu o implementare grijulie.

25.6.2 *Rezolvare detaliată

25.6.3 Cod sursă

Listing 25.6.1: zone_AC1.cpp

```

1 #include<stdio.h>
2 #include<vector>
3 #include<string.h>
4 #include<iostream>
5 #include<fstream>
6
7 #define fin "zone.in"
8 #define fout "zone.out"
9
10 using namespace std;
11
12 int n ,m,k,thissol;
13 char ch[400][400];
14 int sw[400][400], val[400][400];
15 int vecini[100005];
16 int nrlev[100005];
17 int coada[100005];
18 int sz[100005];
19 vector<int> v[100005];
20
21 int comp;
22
23 int dir[7][4] = { {1,0} , {0 ,1 } , { -1,0 } , { 0 , -1 } } ;
24
25 int fill(int x, int y )
26 {
27     sw[x][y]=1;
28     sz[comp]++;
29     for (int i = 0 ;i<4 ;i++)
30     {
31         if (x+dir[i][0]>0 && x+dir[i][0]<=n && y+dir[i][1] >0 &&
32             y+dir[i][1] <=m && ch[x][y]==ch[x+dir[i][0]][y+dir[i][1]] &&
33             sw[x+dir[i][0]][y+dir[i][1]]==0)
34         {
35             val[x+dir[i][0]][y+dir[i][1]]=val[x][y];
36             fill(x+dir[i][0],y+dir[i][1]);
37         }
38     }
39 }
40
41 void go(int x, int y )
42 {
43     sw[x][y]=1;
44     int valc, valnext;
45     for (int i =0;i<4;i++)
46         if (x+dir[i][0]>0 && x+dir[i][0]<=n &&
47             y+dir[i][1]>0 && y+dir[i][1]<=m)
48         {
49             valnext=val[x+dir[i][0]][y+dir[i][1]];
50             valc=val[x][y];
51             if (sw[x+dir[i][0]][y+dir[i][1]]==0 && valnext==valc)
52                 go(x+dir[i][0],y+dir[i][1]);
53             else
54                 if (valnext!=valc && vecini[valnext]!=valc)
55                 {
56                     vecini[valnext]=valc;
57                     v[valc].push_back(valnext);
58                 }
59         }
60     }
61
62 void numarasol2(int x)
63 {
64     thissol+=sz[x];
65     vecini[x]=1;

```

```

66     for (int i= 0; i<v[x].size();i++ )
67         if (vecini[v[x][i]]==0 && nrlev[v[x][i]]>=k)
68             numarasol2(v[x][i]);
69 }
70
71 int main()
72 {
73     int i,j;
74     FILE *f =fopen(fin,"r");
75     fscanf(f,"%d%d", &n , &m , &k );
76     for (i=1;i<=n;i++)
77         fscanf(f,"%s", &ch[i][1]);
78
79     int nr =0;
80
81     for ( i=1;i<=n;i++)
82         for (j=1;j<=m;j++)
83             if (sw[i][j]==0)
84             {
85                 val[i][j]=++nr;
86                 comp=nr;
87                 fill(i,j);
88             }
89
90     memset(sw,0,sizeof(sw));
91
92     for (i=1;i<=n;i++)
93         for (j=1;j<=m;j++)
94             if (sw[i][j]==0)
95             {
96                 go(i,j);
97             }
98
99     int u=0,p=0;
100
101    for (i=1;i<=nr;i++)
102    {
103        nrlev[i]=v[i].size();
104        if (nrlev[i]<k)
105            coada[++u]=i;
106    }
107
108    while (u>=p)
109    {
110        int nod = coada[p];
111        for (i = 0; i<v[nod].size();i++)
112        {
113            nrlev[v[nod][i]]--;
114            if(nrlev[v[nod][i]]==k-1)
115                coada[++u]=v[nod][i];
116        }
117        ++p;
118    }
119
120    for (i=1;i<=n;i++)
121        for(j=1;j<=m;j++)
122            if((nrlev[val[i][j]]<k))
123                val[i][j]= -1;
124
125    int maxsol = 0;
126    memset(sw,0,sizeof(sw));
127    memset(vecini,0,sizeof(vecini));
128
129    for (i =1; i<=n;i++)
130        for (j=1;j<=m;j++)
131            if(val[i][j]<=0)
132                sw[i][j]=1;
133
134    ofstream g(fout);
135
136    for (i=1;i<=n;i++)
137        for (j=1;j<=m;j++)
138        {
139            if (nrlev[val[i][j]]>=k && vecini[val[i][j]]==0)
140            {
141                thissol=0;

```

```

142             numarasol2(val[i][j]);
143             maxsol=max(maxsol,thissol);
144         }
145     }
146
147     cout <<maxsol;
148     g << maxsol ;
149     return 0 ;
150 }
```

Listing 25.6.2: zone_AC2.cpp

```

1 #include<iostream>
2 #include<fstream>
3 #include<string.h>
4
5 #define fin "zone.in"
6
7 using namespace std;
8
9 char s[405][405];
10
11 int n, m ,k,nr ;
12 int cs;
13 int vecini[100005],nrv[100005],number[405][405], sw[405][405];
14 int dir [4][2] = {{ 1 ,0} ,{ 0 , 1 } ,{ -1 ,0 } ,{ 0 , -1 } } ;
15 int vec[100005],pozx[100005],pozy[100005],coada[100005];
16
17 void fill (int x, int y ) {
18     int i,dirx,diry;
19     sw[x][y]=1;
20     for (i=0;i<4;i++) {
21         dirx=x+dir[i][0];
22         diry=y+dir[i][1];
23         if (dirx<=n && dirx>=1 && diry<=m && diry>=1 &&
24             sw[dirx][diry]==0 && s[dirx][diry]==s[x][y])
25         {
26             number[dirx][diry]=number[x][y];
27             fill(dirx,diry);
28         }
29     }
30 }
31
32 void numara (int x, int y )
33 {
34     int i, dirx, diry ;
35     sw[x][y]=1;
36     for (i=0;i<4;i++)
37     {
38         dirx= x+dir[i][0];
39         diry= y+dir[i][1];
40         if (dirx>0 && dirx<=n && diry<=m && diry>0)
41         {
42             if (sw[dirx][diry]==0 && number[x][y]==number[dirx][diry])
43             {
44                 numara(dirx,diry);
45             }
46             if (number[x][y]!=number[dirx][diry] &&
47                 vecini[number[dirx][diry]]!=number[x][y])
48             {
49                 vecini[number[dirx][diry]]=number[x][y];
50                 nrv[number[x][y]]++;
51             }
52         }
53     }
54 }
55
56 void getvecini(int x, int y)
57 {
58     int i, dirx, diry ;
59     sw[x][y]=1;
60     for (i=0;i<4;i++)
61     {
62         dirx=x+dir[i][0];
63         diry=y+dir[i][1];

```

```

64         if (dirx>0 && dirx<=n && diry>0 && diry<=m )
65     {
66         if (sw[dirx][diry]==0 && number[x][y]==number[dirx][diry])
67             getvecini(dirx,diry);
68         if (number[x][y]!=number[dirx][diry] &&
69             vecini[number[dirx][diry]]!=number[x][y])
70         {
71             vecini[number[dirx][diry]]=number[x][y];
72             vec[++vec[0]]=number[dirx][diry];
73         }
74     }
75 }
76 }
77
78 int getsol (int x ,int y )
79 {
80     int dirx,diry, i ;
81     cs++;
82     sw[x][y]=1;
83     for (i=0;i<4;i++)
84     {
85         dirx=x+dir[i][0];
86         diry=y+dir[i][1];
87         if (dirx>0 && diry>0 && dirx<=n && diry<=m &&
88             sw[dirx][diry]==0 && nrw[number[dirx][diry]]>=k )
89         {
90             getsol(dirx,diry);
91         }
92     }
93 }
94
95 int main()
96 {
97     int i,j ;
98     ifstream f(fin);
99     f>>n>>m>>k;
100    for (i=1;i<=n;i++)
101    {
102        f>>&s[i][1];
103    }
104
105    memset(sw,0,sizeof(sw));
106
107    for ( i=1 ;i<=n;i++)
108        for ( j=1;j<=m;j++)
109        {
110            if (sw[i][j]==0)
111            {
112                number[i][j]=++nr;
113                pozx[nr]=i;
114                pozy[nr]=j;
115                fill(i,j);
116            }
117        }
118        memset(sw,0,sizeof(sw));
119        for(i=1;i<=n;i++)
120            for(j=1;j<=m;j++)
121            {
122                if (sw[i][j]==0)
123                {
124                    numara(i,j);
125                }
126            }
127            memset(sw,0,sizeof(sw));
128            int p=0,u=0;
129
130            for (i=1;i<=nr;i++)
131            {
132                if (nrw[i]<k)
133                    coada[++u]=i;
134            }
135
136            memset(sw,0,sizeof(sw));
137            memset(vecini,0,sizeof(vecini));
138
139            for (p=u;)

```

```

140      {
141          vec[0]=0;
142          getvecini(pozx[coada[p]],pozy[coada[p]]);
143          for (i=1;i<=vec[0];i++)
144          {
145              nrw[vec[i]]--;
146              if(nrw[vec[i]]==k-1)
147                  coada[++u]=vec[i];
148          }
149          ++p;
150      }
151
152      int finalrez =0;
153      memset(sw,0,sizeof(sw));
154
155      for (i=1;i<=n;i++)
156          for (j=1;j<=m;j++)
157              if ( sw[i][j]==0 && nrw[number[i][j]]>=k)
158              {
159                  cs=0;
160                  getsol(i,j);
161                  finalrez=max(finalrez,cs);
162              }
163
164      ofstream g("zone.out");
165      g<<finalrez;
166      cout << finalrez;
167      return 0 ;
168 }
```

Listing 25.6.3: zone_MN.cpp

```

1 #include <fstream>
2 #define DIM 302
3 #include <set>
4 using namespace std;
5
6 int di[] = {-1,1,0,0};
7 int dj[] = {0,0,1,-1};
8
9 set<int> L[DIM*DIM];
10 int v[DIM*DIM];
11 int c[DIM*DIM];
12 int w[DIM*DIM];
13 char a[DIM][DIM];
14 int b[DIM][DIM];
15
16 int n,m,i,j,maxim,x,k,cc,d,iv,jv,p,u;
17
18 void fill(int i, int j, int k)
19 {
20     b[i][j] = k;
21     w[k]++;
22     int iv,jv;
23     for (int d=0;d<=3;d++)
24     {
25         iv = i+di[d];
26         jv = j+dj[d];
27         if (b[iv][jv]==0 && a[iv][jv]!=0 && a[iv][jv] == a[i][j])
28             fill(iv,jv,k);
29     }
30 }
31
32 void df(int i)
33 {
34     x+=w[i];
35     v[i] = 1;
36     for (set<int>::iterator it = L[i].begin();it!=L[i].end();it++)
37     {
38         j = *it;
39         if (v[j] == 0)
40         {
41             v[j] = 1;
42             df(j);
43         }
44     }
45 }
```

```

44      }
45  }
46
47 int main()
48 {
49     ifstream fin("zone.in");
50     ofstream fout("zone.out");
51
52     fin>>n>>m>>k;
53     for (i=1;i<=n;i++)
54         fin>>a[i]+1;
55
56     for (i=1;i<=n;i++)
57         for (j=1;j<=m;j++)
58         {
59             if (b[i][j] == 0)
60                 fill(i,j,++cc);
61         }
62
63     for (i=1;i<=n;i++)
64         for (j=1;j<=m;j++)
65             for (d=0;d<=3;d++)
66             {
67                 iv = i+di[d];
68                 jv = j+dj[d];
69                 if (iv>0 && iv<=n && jv>0 && jv<=m&&a[i][j]!=a[iv][jv])
70                 {
71                     L[b[i][j]].insert(b[iv][jv]);
72                     L[b[iv][jv]].insert(b[i][j]);
73                 }
74             }
75
76     for (i=1;i<=cc;i++)
77     {
78         if (L[i].size() < k)
79         {
80             u++;
81             c[u] = i;
82             v[i] = 1;
83         }
84     }
85
86     p=1;
87     while (p<=u)
88     {
89         i = c[p];
90         for (set<int>::iterator it = L[i].begin(); it!=L[i].end();it++)
91         {
92             j = *it;
93             L[j].erase(i);
94             if (L[j].size() < k && v[j] == 0)
95             {
96                 u++;
97                 c[u] = j;
98                 v[j] = 1;
99             }
100        }
101    p++;
102 }
103
104 for (i=1;i<=cc;i++)
105     if (v[i] == 0)
106     {
107         x = 0;
108         df(i);
109         if (x > maxim)
110         {
111             maxim = x;
112         }
113     }
114
115     fout<<maxim<<"\n";
116     return 0;
117 }
```

Listing 25.6.4: zone_PC.cpp

```

1 //Code by Patcas Csaba
2
3 #include <vector>
4 #include <string>
5 #include <set>
6 #include <map>
7 #include <queue>
8 #include <bitset>
9 #include <stack>
10 #include <list>
11
12 #include <numeric>
13 #include <algorithm>
14
15 #include <cstdio>
16 #include <fstream>
17 #include <iostream>
18 #include <sstream>
19 #include <iomanip>
20
21 #include <cctype>
22 #include <cmath>
23 #include <ctime>
24 #include <cassert>
25
26 using namespace std;
27
28 #define LL long long
29 #define PII pair <int, int>
30 #define VB vector <bool>
31 #define VI vector <int>
32 #define VD vector <double>
33 #define VS vector <string>
34 #define VPII vector <pair <int, int> >
35 #define VVI vector < VI >
36 #define VVB vector < VB >
37
38 #define FORN(i, n) for(int i = 0; i < (n); ++i)
39 #define FOR(i, a, b) for(int i = (a); i <= (b); ++i)
40 #define FORD(i, a, b) for(int i = (a); i >= (b); --i)
41 #define FORI(it, X) for(__typeof((X).begin())it=(X).begin();it!=(X).end();++it)
42 #define REPEAT do{
43 #define UNTIL(x) }while(!(x));
44
45 #define SZ size()
46 #define BG begin()
47 #define EN end()
48 #define CL clear()
49 #define X first
50 #define Y second
51 #define RS resize
52 #define PB push_back
53 #define MP make_pair
54 #define ALL(x) x.begin(), x.end()
55
56 #define IN_FILE "zone.in"
57 #define OUT_FILE "zone.out"
58
59 ifstream fin(IN_FILE);
60 ofstream fout(OUT_FILE);
61
62 int dx[4] = {-1, 0, 0, 1};
63 int dy[4] = {0, -1, 1, 0};
64 int n, m, k, nrZones, nrSuperzones;
65 VS a;
66 VVI b;
67 vector <set <int> > g;
68 VI area, indSuperzone, areaSuperzone;
69
70 void addEdge(int node1, int node2)
71 {
72     g[node1].insert(node2);
73     g[node2].insert(node1);
74 }
75

```

```

76 void doBF(PII p)
77 {
78     queue <PII> fifo;
79     fifo.push(p);
80     b[p.X][p.Y] = nrZones;
81     area[nrZones] = 0;
82     while (!fifo.empty())
83     {
84         ++area[nrZones];
85         PII head = fifo.front();
86         FORN(i, 4)
87         {
88             PII newP = MP(head.X + dx[i], head.Y + dy[i]);
89             if (newP.X < 0 || newP.X >= n ||
90                 newP.Y < 0 || newP.Y >= m) continue;
91             if (a[newP.X][newP.Y] != a[p.X][p.Y])
92             {
93                 if (b[newP.X][newP.Y]) addEdge(nrZones, b[newP.X][newP.Y]);
94                 continue;
95             }
96             if (b[newP.X][newP.Y]) continue;
97             b[newP.X][newP.Y] = nrZones;
98             fifo.push(newP);
99         }
100        fifo.pop();
101    }
102 }
103
104 void findZones()
105 {
106     b.RS(n, VI(m));
107     nrZones = 0;
108     FORN(i, n)
109     {
110         FORN(j, m)
111         {
112             if (!b[i][j])
113             {
114                 ++nrZones;
115                 g.RS(nrZones + 1);
116                 area.RS(nrZones + 1);
117                 doBF(MP(i, j));
118             }
119     }
120     void removeZone(int z)
121     {
122         for (set<int>::iterator it = g[z].BG; it != g[z].EN; ++it)
123         {
124             int aux = *it;
125             g[aux].erase(z);
126         }
127         g[z].clear();
128     }
129     void solve()
130     {
131         bool found;
132         do
133         {
134             found = false;
135             FOR(i, 1, nrZones)
136             {
137                 if (g[i].SZ && g[i].SZ < k)
138                 {
139                     found = true;
140                     removeZone(i);
141                 }
142             }
143         }
144         while (found);
145     }
146     void doBF2(int indZone)
147     {
148         queue <int> fifo;
149         fifo.push(indZone);
150         indSuperzone[indZone] = nrSuperzones;
151     }

```

```

152     int head = fifo.front();
153     areaSuperzone[nrSuperzones] += area[head];
154     for (set <int>::iterator it = g[head].BG; it != g[head].EN; ++it)
155     {
156         int aux = (*it);
157         if (indSuperzone[aux] == 0)
158         {
159             indSuperzone[aux] = nrSuperzones;
160             fifo.push(aux);
161         }
162     }
163     fifo.pop();
164 }
165 }
166
167 void findSuperzones()
168 {
169     nrSuperzones = 0;
170     indSuperzone.RS(nrZones + 1);
171     FOR(i, 1, nrZones)
172     {
173         if (indSuperzone[i] == 0)
174         {
175             ++nrSuperzones;
176             areaSuperzone.RS(nrSuperzones + 1);
177             doBF2(i);
178         }
179     }
180
181 int main()
182 {
183     //Read data
184     fin >> n >> m >> k;
185     a.RS(n);
186     FORN(i, n) fin >> a[i];
187     fin.close();
188
189     //Solve
190     findZones();
191     solve();
192     findSuperzones();
193
194     //Write data
195     //FORN(i, n)
196     //{
197     //    FORN(j, m) fout << ((indSuperzone[b[i][j]] == 1) ? "X" : " ");
198     //    fout << endl;
199     //}
200     fout << *max_element(ALL(areaSuperzone)) << endl;
201
202     fout.close();
203
204     return 0;
205 }
```

Capitolul 26

ONI 2012

26.1 cutie

Problema 1 - cutie

100 de puncte

După ce au vizitat toate obiectivele turistice din municipiul Iași, Ioana și Maria au inventat un joc. Ele au la dispoziție un număr de n cutii aranjate în linie dreaptă, numerotate în ordine de la 1 la n , și un număr de m bile ce pot fi așezate în unele dintre aceste cutii. Unele cutii sunt deteriorate, astfel că bilele dispar dacă sunt puse în acele cutii.

O mutare constă în alegerea unei bile și poziționarea ei în una din cutiile învecinate (precedenta sau următoarea cutie).

Bilele pot fi mutate după următoarea regulă: când o bilă a fost mutată pentru prima dată într-o anumită direcție, atunci bila își păstrează direcția de deplasare la următoarele mutări (de exemplu, dacă o bilă a fost mutată pentru prima dată spre stânga atunci orice mutări ulterioare ale acestei bile se pot face doar spre stânga).

Jocul se termină atunci când nici un jucător nu mai poate face nici o mutare. Pierde primul jucător care nu mai poate face nici o mutare.

Fetele joacă un număr de T astfel de jocuri.

Știind că Ioana este prima care face o mutare, iar apoi fetele mută alternativ, se cere să se stabilească pentru fiecare din cele T jocuri dacă ea are sau nu o strategie sigură de câștig.

Cerințe

Date de intrare

Fișierul **cutie.in** conține pe prima linie un număr natural T , care reprezintă numărul de jocuri pe care le joacă cele două fete.

Pe următoarele linii fișierul conține, în ordine, descrierea celor T jocuri.

Fiecare joc este descris prin câte 3 linii: prima linie va conține, în ordine, trei numere naturale n , k , m separate prin câte un singur spațiu (n reprezintă numărul de cutii, k reprezintă numărul de cutii deteriorate și m reprezintă numărul de bile), a doua linie va conține k numere naturale, separate prin câte un singur spațiu, reprezentând numerele de ordine ale cutiilor deteriorate, iar a treia linie va conține m numere naturale reprezentând numerele de ordine ale cutiilor care conțin bile la începutul jocului.

Date de ieșire

Fișierul **cutie.out** va conține pe prima linie un sir de T valori de 0 și 1 nesperate prin spații, având următoarea semnificație: valoarea de pe poziția i din sir ($i \in \{1, 2, \dots, T\}$) este 1 dacă jocul i este câștigat de Ioana sau 0 dacă jocul i este pierdut de Ioana.

Restricții și precizări

- $1 \leq T \leq 10$;
- $1 \leq n, m \leq 10\,000$;
- $0 \leq k \leq 10\,000$;
- se consideră că din prima cutie nu poate fi mutată o bilă spre stânga, iar din ultima cutie nu se poate muta o bilă spre dreapta;

- inițial nicio bilă nu se află în prima cutie, în ultima cutie sau într-o cutie deteriorată;
- în fișierul de intrare pozițiile bilelor și cele ale cutiilor deteriorate sunt sortate crescător;
- într-o cutie se pot afla mai multe bile;
- pentru 20% dintre teste k are valoarea 0 sau 1.

Exemplu:

cutie.in	cutie.out	Explicații
2	10	Pentru primul joc, Ioana are o strategie sigură de câștig (pentru a câștiga va muta bila la dreapta), iar pentru al doilea joc nu are o strategie sigură de câștig.
10 1 1		
6		
3		
10 1 2		
5		
4 4		

Timp maxim de execuție/test: **0.1** secunde

Memorie: total **4 MB** din care pentru stivă **4 MB**

Dimensiune maximă a sursei: **10 KB**

26.1.1 Indicații de rezolvare

Andrei Ciocan, Universitatea Politehnica București

Ideea pe care se bazează soluția este că primul jucător ar trebui să aibă o strategie de orientare a bilelor astfel încât numărul total de mutări să fie impar.

Se observă că fiecare bilă este caracterizată de una din cele trei stări:

- 1) atât în stânga cât și în dreapta distanța până la o cutie deteriorată este pară;
- 2) în stânga și în dreapta distanța până la o cutie deteriorată este impară;
- 3) avem o distanță pară și cealaltă impară.

Se observă că dacă ultima stare nu este prezentă în joc (distanța pară într-o direcție și impară în cealaltă), atunci rezultatul este dat de paritatea numărului de cazuri de tip 2 (astfel numărul de mutări este impar, ceea ce duce la câștigarea jocului de primul jucător).

Tratarea acestui caz particular aduce 20 de puncte. Astfel jocul format doar din primele două tipuri de bile reprezintă de fapt o constantă.

În cazul în care avem doar bile de tipul 3, atunci se observă că ultimul jucător care poate muta o astfel de bilă câștigă, astfel primul jucător câștigă dacă aceste bile sunt în număr impar.

Dacă acestea ar fi în număr par, atunci al doilea jucător alege exact aceleași mutări ca și primul, ajungându-se în final la un număr par de mutări pe care le fac jucătorii, de unde rezultă că primul jucător pierde.

Pe același principiu se merge și în cazul când pe lângă bilele de tipul 3, apar și bile de tipul 1 sau tipul 2.

Celelalte bile formează de fapt o constantă, se comportă ca o singură bilă (conținândă dacă această constantă este pară sau impară), care impune jocului încă un număr fix de pași.

Dacă numărul de bile de tipul 3 este impar, atunci primul jucător câștigă, indiferent de constantă. Dacă numărul de bile de tipul 3 este par, atunci primul jucător ar avea strategie de câștig în cazul în care constanta ar fi impară.

Pentru a câștiga, ar trebui să mute constanta, pentru ca al doilea jucător să aibă prima mutare pentru jocul doar cu bile de tip 3 în număr par (poziție pierzătoare).

26.1.2 *Rezolvare detaliată

26.1.3 Cod sursă

Listing 26.1.1: cutie.cpp

```

1 #include<iostream>
2
3 using namespace std;
4
5 int b[10005],c[10005];
6
7 int main()
8 {
9     int t,i,nrp,nri,ok,soli,solip,solp,m,n,k,indb,indc;
10    ifstream f("cutie.in");
11    ofstream g("cutie.ok");
12    f>>t;
13    for (;t;--t)
14    {
15        f>>n>>k>>m;
16        for (i=1;i<=k;++i)
17            f>>c[i];
18
19        c[0]=1;
20        c[k+1]=n;
21
22        for (i=1;i<=m;i++)
23            f>>b[i];
24
25        solp=soli=solip=0;
26        nrp=nri=0;
27
28        indb=1;
29
30        for (i=1;i<=k+1;i++)
31        {
32            nrp=nri=0;
33            while (b[indb]<=c[i] && indb <=m)
34            {
35                if ((b[indb]-c[i-1])%2==0)
36                    nrp++;
37                else
38                    nri++;
39                indb++;
40            }
41
42            if ((c[i]-c[i-1]+1)%2==0)
43                solip+=(nrp+nri);
44            else
45            {
46                solp+=nrp;
47                soli+=nri;
48            }
49        }
50
51        if (solip%2==1)
52            g<<1;
53        else
54            if ((solip%2==0) && (soli%2==1))
55                g<<1;
56            else
57                g<<0;
58    }
59
60    g.close();
61    f.close();
62    return 0;
63 }
```

26.2 gheizere

Problema 2 - gheizere

100 de puncte

Într-o zonă vulcanică de formă dreptunghiulară de dimensiuni întregi $N \times M$, împărțită în celule patrate de latura 1, există P gheizere. Gheizerul este o izvoare intermitentă de origine vulcanică, care emite în atmosferă la intervale regulate de timp jeturi de apă fierbinte și/sau vapori, datorită

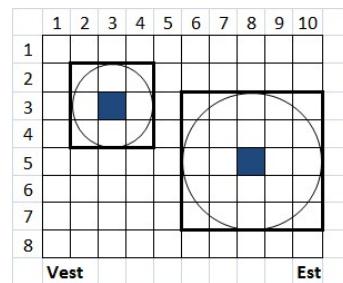
încălzirii rapide a apelor din golarile subterane.

Erupția unui gheizer produce stropirea cu apă fierbinte a unei zone circulare cu centrul situat în punctul (x, y) aflat pe linia x și coloana y , și raza r , zona afectată fiind considerată zona delimitată de pătratul în care se înscrie cercul.

În figura alăturată avem 2 gheizeri ce au coordonatele $(3,3)$ și $(5,8)$, primul gheizer cu raza egală cu 1, iar cel de-al doilea gheizer cu raza egală cu 2.

Pentru fiecare gheizer se cunoaște intervalul de unități de timp t consumat între eruptii, precum și durata d a unei eruptii, valori exprimate în secunde.

Misiunea unui cercetător este de a găsi un traseu ce pleacă dintr-un punct dat al terenului situat pe latura de vest $(v, 1)$ și traversează zona fără să o părăsească și fără să treacă de două ori prin aceeași poziție până la un alt punct (e, M) situat pe latura de est. Cercetătorul se poate deplasa în oricare dintre direcțiile **nord**, **est**, **sud**, iar timpul consumat pentru parcurgerea unei celule neafectate la un moment dat este de o secundă. Parcurgerea unei zone în care acționează **cel puțin** un gheizer **pe perioada de eruptie** a acestuia implică riscuri majore pentru cercetător.



Cerințe

Să se determine timpul minim necesar traversării de la vest la est a zonei vulcanice de către cercetător fără riscuri majore.

Date de intrare

Fișierul **gheizere.in** conține:

- pe prima linie trei valori naturale N , M și P , unde N reprezintă numărul de linii și M numărul de coloane ale zonei vulcanice, iar P numărul de gheizeri;
- pe a două linie două valori naturale v respectiv e , unde v reprezintă linia punctului de plecare din vest, iar e linia punctului de sosire din est;
- pe următoarele P linii un set de cinci valori x , y , r , t , d ce reprezintă în ordine, pentru fiecare gheizer coordonatele centrului (x, y) , raza cercului de eruptie, t timpul dintre eruptii, iar d durata unei eruptii.

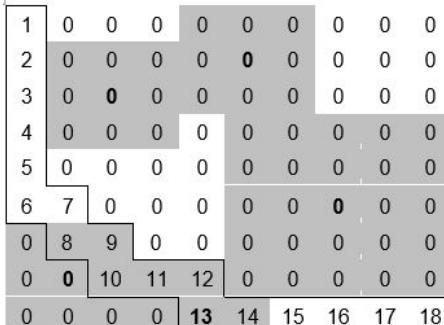
Date de ieșire

Fișierul **gheizere.out** va conține o singură valoare naturală T ce reprezintă timpul minim necesar cercetătorului pentru a traversa zona vulcanică de la vest la est fără riscuri majore.

Restricții și precizări

- $2 \leq N, M \leq 250$
- $1 \leq e, v \leq N$
- $1 \leq x \leq N; 1 \leq y \leq M$.
- $1 \leq P \leq 1000$
- $1 \leq r \leq 5$
- $1 \leq d \leq 5$
- $2 \leq t < 10$
- Două gheizeri nu pot avea aceleași coordonate pentru centrul cercului.
- Inițial toate gheizerele sunt inactive.
- Cercetătorul parcurge traseul fără să se oprească.
- Pentru datele de intrare există întotdeauna soluție.
- Durata maximă a unui traseu parcurs de către cercetător este ≤ 1000 secunde.

Exemplu:

gheizere.in	gheizere.out	Explicații
9 10 5 1 9 2 6 1 2 2 3 3 1 3 3 6 8 2 6 1 8 2 1 4 2 9 5 1 4 1	18	<p>Unul dintre traseele posibile care pornește din (1,1) și ajunge în (9,10) în timp minim este marcat, secundă cu secundă, în figura următoare.</p> <p>Zonele influențate de gheizer sunt marcate cu pătrate gri, iar centrul fiecărui gheizer este evidențiat (bold).</p> 

Timp maxim de executare/test: **1.0** secunde

Memorie: total **16 MB** din care pentru stivă **16 MB**

Dimensiune maximă a sursei: **10 KB**

26.2.1 Indicații de rezolvare

Soluție 80 p

prof. Eugen Nodea, C.N. "Tudor Vladimirescu", Tg.Jiu

Se folosește *algoritmul Lee*

Structuri de date utilizate:

- matrice în care vom marca toate zonele atinse de erupțiile celor p gheizer;
- matrice ce va reține timpul necesar erupției pentru gheizerul de coordonate (i, j) ;
- matrice pentru determinarea timpului necesar pentru atingerea celulei (i, j) .

Soluție 100 p

Marius Stroe, Universitatea București

Vom folosi o variantă îmbunătățită a algoritmului anterior datorită restricției ca drumul nu poate depăși 1 000 de secunde. Identificăm o stare a personajului ca fiind (k, i, j) , unde (i, j) reprezintă celula, iar k cât timp a trecut de la începutul expediției.

În starea (k, i, j) putem ajunge din $(k - 1, i - 1, j - 1)$, $(k - 1, i, j - 1)$ sau $(k - 1, i + 1, j - 1)$. Astfel, găsim că $(1, v, 1)$ este starea initială iar o stare finală este de forma (L, e, M) . Soluția este cel mai mic L dintre toate stările finale.

26.2.2 *Rezolvare detaliată

26.2.3 Cod sursă

Listing 26.2.1: gheizere.cpp

```

1 /*
2      Solutie 80 p
3      prof. Eugen Nodea
4 */
5 #include <cstdio>
6 #include <queue>
7 #include <algorithm>
8
9 #define nmax 253
10 #define mmax 253
11 #define pmax 10001
12 #define inf 253*253+2
13
14 using namespace std;
15
16 int A[nmax][mmax], M[nmax][mmax], Timp[nmax][mmax];

```

```

17 int dx[] = {-1, 0, 1};
18 int dy[] = { 0, 1, 0};
19 int n, m, p, d, T, i, j, k, l, c, e, v, r, t, x;
20
21 struct cel
22 {
23     int l,c;
24 };
25
26 queue <cel> q;
27
28 inline int caut(int l, int c, int x, int ts)
29 {
30     int i, j, t, d;
31     //caut existenta centrului unui gheizer unei am memorate date despre durata
32     for (i=l-x; i<=l+x; i++)
33         for (j=c-x; j<=c+x; j++)
34             if (A[i][j] && i>0 && j>0 && i<=n && j<=m)
35             {
36                 //calculați dacă este afectată celula (i,j)
37                 t = A[i][j] / 100;    // timpul între eruptii
38
39                 d = A[i][j] % 100;    // durata unei eruptii
40                 if (ts % (t + d) >= t) return 0;
41             }
42     return 1;
43 }
44
45 void Lee(int l, int c)
46 {
47     int ic, jc, iv, jv, k, ts, x, ok;
48     cel p,u;
49
50     p.l=l; p.c=c;
51     q.push(p);
52     Timp[l][c] = 1;
53
54     while (!q.empty())
55     {
56         p = q.front(); q.pop();
57         ic = p.l; jc = p.c;
58         if (Timp[ic][jc] != inf && ic == e && jc == m)
59         {
60             break;
61         }
62         // timpul scurs
63         ts = Timp[ic][jc];
64         for (k=0; k<3; ++k)
65         {
66             iv = ic + dx[k];
67             jv = jc + dy[k];
68
69             if (iv>0 && iv<=n && jv>0 && jv<=m)
70             {
71                 //determină dacă celula este afectată
72                 x = M[iv][jv];
73                 ok = 1;
74                 if (x && caut(iv,jv,x,ts) == 0) ok=0;
75                 if (Timp[iv][jv] > Timp[ic][jc] + 1 && ok)
76                 {
77                     Timp[iv][jv] = Timp[ic][jc] + 1;
78                     u.l = iv; u.c = jv;
79                     q.push(u);
80                 }
81                 else if (Timp[iv][jv]<=Timp[ic][jc] && ok)
82                 {
83                     Timp[iv][jv] = Timp[ic][jc] + 1;
84                     u.l = iv; u.c = jv;
85                     q.push(u);
86                 }
87             }
88         }
89     }
90 }
91
92 int main()

```

```

93  {
94      freopen("gheizere.in", "r", stdin);
95      freopen("gheizere.out", "w", stdout);
96
97      scanf("%d %d %d", &n, &m, &p);
98      scanf("%d %d", &v, &e);
99
100     for (k=1; k<=p; ++k)
101    {
102        scanf("%d %d %d %d", &i, &j, &r, &t, &d);
103
104        //retin pentru fiecare gheizer atat timpul intre eruptii cat si
105        // durata unei eruptii
106        A[i][j] = t * 100 + d;
107
108        //retin in matrice M zonele afectate de eruptie (raza maxima)
109        for (l=1; l<=2 * r + 1; ++l)
110            for (c=1; c<=2 * r + 1; ++c)
111                if (l>0 && l<=n && c>0 && c<=m)
112                    M[i-r+l-1][j-r+c-1] = max(r, M[i-r+l-1][j-r+c-1]);
113    }
114
115    //initializare
116    for (i=1; i<=n; i++)
117        for (j=1; j<=m; j++)
118            Timp[i][j] = inf;
119
120    Lee(v,1);
121
122    if (Timp[e][m] == inf) printf("0\n"); //nu este cazul
123    else printf("%d\n", Timp[e][m]);
124
125    return 0;
126}

```

Listing 26.2.2: gheizereM.cpp

```

1 #include <cstdio>
2 #include <cassert>
3 #include <memory.h>
4 #include <algorithm>
5
6 using namespace std;
7
8 const char iname[] = "gheizere.in";
9 const char oname[] = "gheizere.out";
10
11 const int MAX_N = 250;
12 const int MAX_GHEIZERS = 1005;
13 const int MAX_TIME = MAX_N * 2;
14 const int infinity = 0x3F3F3F3F;
15
16 typedef struct
17 {
18     int x, y, r, t, d;
19 } gheizer;
20
21 int rows, cols, gheizers, start_line, end_line;
22
23 gheizer g[MAX_GHEIZERS];
24
25 int C[2][MAX_N][MAX_N];
26
27
28 void print(int C[][MAX_N])
29 {
30     for (int i = 0; i < rows; ++ i)
31     {
32         for (int j = 0; j < cols; ++ j)
33         {
34             if (C[i][j] < infinity)
35                 printf("%2d ", C[i][j]);
36             else
37                 printf(" 0 ");
38         }
39     }
40 }

```

```

39         printf("\n");
40     }
41     getchar();
42 }
43
44 void volcanoes_activity(int time, int C[][][MAX_N])
45 {
46     for (int k = 0; k < gheizers; ++ k)
47     {
48         for (int i = max(0, g[k].x - g[k].r);
49             i <= min(g[k].x + g[k].r, rows - 1);
50             ++ i)
51         {
52             for (int j = max(0, g[k].y - g[k].r);
53                 j <= min(g[k].y + g[k].r, cols - 1);
54                 ++ j)
55             {
56                 // Check if the volcano erupted over this cell.
57                 if (time % (g[k].t + g[k].d) >= g[k].t)
58                     C[i][j] = infinity;
59             }
60         }
61     }
62 }
63
64 int solve(void) {
65     /* Solution:
66      *   C(k, i, j) is either 0 or 1 if we can be in (i, j) cell at time k
67      *   and
68      *   C(k, i, j) = max(C(k-1, i-1, j), C(k-1,i,j-1), C(k-1,i+1,j-1))
69      *
70      * Time complexity: O(L * (M * N + P * R^2))
71      * Memory complexity: O(M * N)
72     */
73
74     memset(C[0], infinity, sizeof C[0]);
75     C[0][start_line][0] = 1;
76     volcanoes_activity(0, C[0]);
77
78     int time;
79     for (time = 1; time < MAX_TIME; ++ time)
80     {
81         int stp = time % 2;
82         memset(C[stp], infinity, sizeof C[stp]);
83
84         for (int i = 0; i < rows; ++ i)
85         {
86             for (int j = 0; j < cols; ++ j)
87             {
88                 int step = infinity;
89                 if (j > 0)
90                     step = min(step, C[stp ^ 1][i][j - 1]);
91                 if (i > 0)
92                     step = min(step, C[stp ^ 1][i - 1][j]);
93                 if (i < rows - 1)
94                     step = min(step, C[stp ^ 1][i + 1][j]);
95                 if (step < infinity)
96                     C[stp][i][j] = step + 1;
97                 else
98                     C[stp][i][j] = infinity;
99             }
100        }
101        volcanoes_activity(time, C[stp]);
102
103        if (C[stp][end_line][cols - 1] < infinity)
104            return C[stp][end_line][cols - 1];
105    }
106    return -1;
107 }
108
109 int main(void)
110 {
111     FILE *fi = fopen(iname, "r");
112     assert(fscanf(fi, "%d %d %d", &rows, &cols, &gheizers) == 3);
113     assert(fscanf(fi, "%d %d", &start_line, &end_line) == 2);
114     start_line --, end_line --;

```

```

115 // TODO: Assert limits.
116 for (int i = 0; i < gheizers; ++ i)
117 {
118     assert(fscanf(fi, "%d %d %d %d %d",
119                 &g[i].x, &g[i].y, &g[i].r, &g[i].t, &g[i].d)
120             == 5);
121     g[i].x --, g[i].y --;
122     assert(0 <= g[i].x && g[i].x < rows);
123     assert(0 <= g[i].y && g[i].y < cols);
124     assert(1 <= g[i].r && g[i].r <= 5);
125     assert(1 <= g[i].t && g[i].t < 10);
126     assert(1 <= g[i].d && g[i].d <= 5);
127 }
128
129 fprintf(fopen(oname, "w"), "%d", solve());
130
131 fclose(fi);
132 return 0;
133 }

```

26.3 plus

Problema 3 - plus

100 de puncte

Locuitorii planetei Aritmo au hotărât ca în celebrul an 2012 să le explice pământenilor metoda ”plus” de adunare a numerelor naturale pe planeta lor. La fel ca și planetele, înainte de adunare, numerele se aliniază astfel încât să se obțină cât mai multe cifre egale pe aceleași poziții. Cifrele egale, astfel obținute, se elimină din cele două numere. Pentru a obține rezultatul final, se adună cele două numerele deplasate, obținute după eliminare, ca în exemplu.

Exemplu: Numerele 18935 și 85352 se aliniază ca în figura alăturată. După eliminare se obțin numerele 19 și 52 care se adună deplasate, pentru a obține rezultatul final. Așadar 18935 plus 85352=242.

$$\begin{array}{r} 18935 \text{ plus} \\ 85352 \\ \hline 242 \end{array}$$

Dacă există mai multe posibilități de a alinia numerele astfel încât să se eliminate același număr maxim de cifre, atunci numerele sunt aliniate astfel încât, după eliminare și adunarea numerelor după metoda descrisă, să se obțină o valoare cât mai mare.

$$\begin{array}{r} 22331 \text{ plus} \\ 3322 \\ \hline 2232 \end{array}$$

Exemplu: 22331 plus 3322 = 33331 (există două moduri în care cele două numere pot fi aliniate astfel încât să se eliminate un număr maxim de cifre, valoarea maximă obținându-se atunci când se elimină cele două cifre 2).

$$\begin{array}{r} 22331 \text{ plus} \\ 3322 \\ \hline 33 \end{array}$$

Dacă două numere a și b sunt identice sau nu au cifre comune atunci a plus $b = 0$.

Dacă se elimină toate cifrele unui număr atunci rezultatul este dat de cifrele rămase în celălalt număr.

Exemplu: 23 plus 523=5, 562 plus 56=2.

Adunarea mai multor numere se face de la stânga la dreapta: se adună primele două numere conform metodei descrise mai sus, apoi rezultatul se adună cu al treilea, și aşa mai departe.

Într-o expresie în care se adună mai multe numere pot să apară paranteze rotunde. În evaluarea unei asemenea expresii, numită expresie parantezată, se efectuează mai întâi adunările din paranteze conform metodei descrise mai sus, parantezele fiind apoi înlocuite cu rezultatul adunărilor din paranteze.

Se definește **adâncimea** A_E corespunzătoare unei expresii parantezate E astfel:

- dacă expresia E nu conține paranteze, atunci adâncimea acesteia este 0;
- dacă expresia E este de forma (F) , atunci $A_E = 1 + A_F$;
- dacă expresia E este de forma E_1 plus E_2 ... plus E_k , atunci $A_E = \max(A_{E_1}, A_{E_2}, \dots, A_{E_k})$.

Cerințe

Pentru a-i ajuta pe pămânenii care doresc să învețe acest nou mod de adunare, scrieți un program care citește o expresie parantezată și determină:

- adâncimea expresiei date;
- valoarea acestei expresii.

Date de intrare

Fișierul **plus.in** conține pe prima linie un număr natural n . Pe următoarele n linii se află descrierea expresiei parantezate. Pe fiecare dintre aceste linii se află un număr natural sau una dintre valorile -1 sau -2. Valoarea -1 reprezintă o paranteză rotundă deschisă iar valoarea -2 reprezintă o paranteză rotundă închisă.

Date de ieșire

Fișierul de ieșire **plus.out** va conține:

- pe prima linie numărul natural ce reprezintă adâncimea expresiei date;
- pe a doua linie se va scrie numărul natural ce reprezintă rezultatul evaluării expresiei date, adunarea numerelor făcându-se conform metodei descrise.

Restricții și precizări

- $1 < n \leq 2000$;
- fiecare dintre celelalte numere naturale din fișier are cel mult 9 cifre;
- în fiecare paranteză se află cel puțin un număr natural;
- dacă într-o paranteză se află un singur număr natural, atunci valoarea expresiei este egală cu valoarea numărului din paranteză;
- pentru rezolvarea corectă a cerinței a) se acordă 20% din punctaj, iar pentru rezolvarea corectă a ambelor cerințe se acordă punctajul integral.

Exemple:

plus.in	plus.out	Explicații
12	2	Expresia parantezată care trebuie evaluată este:
-1	4639	$(1343 \text{ plus } (234 \text{ plus } 4532)) \text{ plus } 14091 \text{ plus } (21 \text{ plus } 2) =$ $(1343 \text{ plus } 45334) \text{ plus } 14091 \text{ plus } (21 \text{ plus } 2) =$ $4543 \text{ plus } 14091 \text{ plus } (21 \text{ plus } 2) =$ $4543 \text{ plus } 14091 \text{ plus } 1 =$ $46391 \text{ plus } 1 = 4639$
1343		
-1		Valoarea expresiei este 4639.
234		Adâncimea expresiei este 2, deoarece
4532		$\bullet A_{(1343 \text{ plus } (234 \text{ plus } 4532)) \text{ plus } 14091 \text{ plus } (21 \text{ plus } 2)} =$ $\max(A_{(1343 \text{ plus } (234 \text{ plus } 4532))}, A_{14091}, A_{(21 \text{ plus } 2)}) = \max(2, 0, 1) = 2$
-2		$\bullet A_{(1343 \text{ plus } (234 \text{ plus } 4532))} = 1 + \max(A_{1343}, A_{(234 \text{ plus } 4532)}) =$ $1 + \max(0, 1) = 2$
14091		$A_{1343} = 0;$
-1		$\bullet A_{(234 \text{ plus } 4532)} = 1 + A_{234 \text{ plus } 4532} = 1 + 0 = 1$
21		
2		
-2		

Timp maxim de executare/test: **0.2** secunde

Memorie: total **10 MB** din care pentru stivă **10 MB**

Dimensiune maximă a sursei: **10 KB**

26.3.1 Indicații de rezolvare

prof. Cristina Sichim, C.N. "Ferdinand I" Bacău

Pentru rezolvarea problemei se utilizează o *stivă* în care se memorează rezultatele parțiale obținute pe parcursul evaluării expresiei parantezate.

Valoarea -1, corespunzătoare unei paranteze deschise se adaugă în stivă și indică prezența unei noi subexpresii.

Valoarea -2, corespunzătoare unei paranteze închise determină eliminarea elementelor din stivă până la prima paranteză deschisă și adăugarea în stivă a elementului care conține valoarea subexpresiei.

Numărul de paranteze imbricate se actualizează de fiecare dată când se întâlnește o valoare corespunzătoare unei paranteze.

Pentru adunarea conform metodei plus, se utilizează *numere mari* și pentru obținerea rezultatului se tratează fiecare caz descris în enunț.

26.3.2 *Rezolvare detaliată

26.3.3 Cod sursă

Listing 26.3.1: plusCS.cpp

```

1 #include <fstream>
2 #include <iostream>
3
4 using namespace std;
5
6 int m,n,p[1000],j,k,M,i,sterse,l,h,d,adancime;
7 int a[1000], b[15],rez[1000],c[1000],s[1000][1000];
8 ifstream f("plus.in");
9 ofstream g("plus.out");
10
11 void copie(int a[], int b[]) { for(int i=0;i<=b[0];i++) a[i]=b[i]; }
12
13 void aduna(int a[], int b[],int k)
14 {int i,t=0,r,l=b[0],ia=1,ib=1,rez[0]=0;
15
16 if(k<1) for(ib=1;ib<=l-k;ib++) rez[0]++,rez[rez[0]]=b[ib];
17 if(k>1) for(ia=1;ia<=k-1;ia++) rez[0]++,rez[rez[0]]=a[ia];
18 while(ib<=l && ia<=a[0])
19 {
20     if(b[ib]!=a[ia])
21         {r=a[ia]+b[ib]+t; rez[0]++;rez[rez[0]]=(r>9?r-10:r); t=r>9;}
22     ia++;
23     ib++;
24 }
25 while(ia<=a[0]) {r=a[ia]+t; rez[0]++;rez[rez[0]]=(r>9?r-10:r); t=r>9;
26     ia++;}
27 while(ib<=l) {r=b[ib]+t; rez[0]++;rez[rez[0]]=(r>9?r-10:r); t=r>9;
28     ib++;}
29 if(t) {rez[0]++; rez[rez[0]]=1;}
30 while(rez[0] && rez[rez[0]]==0) rez[0]--;
31 if(rez[0]==0) rez[0]=1,rez[1]=0;
32 }
33
34 int cate(int a[],int b[],int &k)
35 { int nr,l=b[0],i,j,M=0; p[0]=0;k=0;
36
37     for(i=1;i<=a[0]+b[0]-1;i++)
38     { // cate au in comun cele doua siruri daca ultimul element al sirului b
39         // este pe poz i in a
40         nr=0;
41         for(j=1;j<=b[0];j++)
42             if(i+j-1>0 && i+j-1<=a[0]) nr=nr+(b[j]==a[i+j-1]);
43         if(nr>M) {M=nr;k=1;p[k]=i;
44             else if(nr==M) k++,p[k]=i;
45         }
46     return M;
47 }
48
49 int mai_mare()
50 { if(rez[0]>c[0]) return 1;
51 if(rez[0]<c[0]) return 0;
52 int i=rez[0];
53 while(i && rez[i]==c[i]) i--;
54 return (rez[i]>c[i]);
55 }
56
57
58 void Plus(int a[], int b[])
59 { int r,j,k;c[0]=0;
60 r=cate(a,b,k);
61 if(r==0) {c[0]=1;c[1]=0;}
62 else
63     for(j=1;j<=k;j++)
64     {
65         aduna(a,b,p[j]); //r=a+b dar b este deplasat, ultimul element fiind

```

```

66             // pe locul p[j]
67         if(mai_mare()) copie(c,rez); // ce am in rez mai mare decat c
68     }
69 }
70
71 int main()
72 {
73     int x;
74     f>>n;
75     s[0][0]=-1;
76     while(n--)
77     { f>>x;
78         if(x==-1) {h++;s[h][0]=-1;d++; if(d>adancime)adancime=d;}
79         else
80             if(x!=-2) { //numarul se aduna cu ultimul introdus daca nu e primul
81                 // din paranteza
82                 b[0]=0;
83                 do{b[++b[0]]=x%10;x=x/10;}while(x);
84                 if(s[h][0]==-1) {h++;copie(s[h],b);}
85                 else {Plus(s[h],b);copie(s[h],c);}
86             }
87             else{d--;
88                 if(s[h-1][0]==-1) {h--; copie(s[h],s[h+1]);
89                     if(s[h-1][0]!=-1)
90                         {Plus(s[h],s[h-1]); h--;copie(s[h],c);}
91                 }
92                 else {Plus(s[h],s[h-1]); h--;copie(s[h],c);}
93             }
94         }
95         g<<adancime<<'\\n';
96
97         if(s[h-1][0]!=-1) {Plus(s[h],s[h-1]); h--;copie(s[h],c);}
98         for(i=s[h][0];i>=1;i--)g<<s[h][i];
99
100        g<<'\\n';
101
102        return 0;
103    }

```

Listing 26.3.2: plusSC.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <cassert>
4
5 #define NN 1500
6 #define MM 64
7
8 using namespace std;
9
10 ifstream fin("plus.in");
11 ofstream fout("plus.out");
12
13 typedef unsigned char NrMare[MM];
14
15 NrMare stiva[NN];
16 int n, nrs;
17
18 int NrMareCmp(NrMare a, NrMare b)
19 {
20     // daca a<b => -1
21     // daca a>b => 1
22     // daca a=b => 0
23     if( a[0] > b[0] )
24         return 1;
25     if( a[0] < b[0] )
26         return -1;
27     int i=a[0];
28     while( i>0 && a[i]==b[i] )
29         --i;
30     if( i == 0 )
31         return 0;
32     return ( a[i]<b[i] ) ? -1 : 1;
33 }
34

```

```

35 void Atrib(NrMare x, int a)
36 {
37     x[0] = 0;
38     do
39         x[++x[0]] = a % 10 , a /= 10;
40     while(a);
41 }
42
43 void Atrib(NrMare d, NrMare s)
44 {
45     for( int i=0 ; i<=s[0] ; ++i )
46         d[i] = s[i];
47 }
48
49
50 void Afis(ostream &out, NrMare x, bool ENDL=true)
51 {
52     for(int i = x[0] ; i ; --i)
53         out << (int)x[i] ;
54     if(ENDL)
55         out << endl;
56     else
57         out << " ";
58 }
59
60 void Plus(NrMare nm1, NrMare nm2, NrMare sol)
61 {
62     sol[ sol[0]=1 ] = 0;
63     int pc = 0 ; // numarul de pozitii comune
64     NrMare c;
65     unsigned char *x , *y;
66     for( int ttt=0 ; ttt<2 ; ++ttt )
67     {
68         if( ttt )
69             x = nm2, y = nm1;
70         else
71             x = nm1, y = nm2;
72
73         for( int i=y[0] ; i>0 ; --i )
74         {
75             int nrpc = 0;
76             for( int j=1, k=i ; j<=x[0] && k<=y[0] ; ++j, ++k )
77             {
78                 if( x[j]==y[k] )
79                     ++ nrpc;
80             }
81
82             if( nrpc > 0 && nrpc >= pc )
83             {
84                 int t = 0 , tmp;
85                 c[0]=0;
86                 for( int j=1 ; j<i ; ++j )
87                 {
88                     tmp = y[j]+t;
89                     ++c[0];
90                     assert(c[0] < MM);
91                     c[c[0]]= tmp % 10;
92                     t = tmp /10;
93                 }
94
95                 int j , k;
96                 for( j=1 , k=i; j<=x[0] && k<=y[0] ; ++j , ++k )
97                 if(x[j]!=y[k])
98                 {
99                     tmp = x[j]+y[k]+t;
100                    ++c[0];
101                    assert(c[0] < MM);
102                    c[c[0]]= tmp % 10;
103                    t = tmp /10;
104                }
105
106                while( j<=x[0] )
107                {
108                    tmp = x[j]+t;
109                    ++c[0];
110                    assert(c[0] < MM);

```

```

111                     c[c[0]]= tmp % 10;
112                     t = tmp /10;
113                     j++;
114                 }
115
116             while( k<=y[0] )
117             {
118                 tmp = y[k]+t;
119                 ++c[0];
120                 assert(c[0] < MM);
121                 c[c[0]]= tmp % 10;
122                 t = tmp /10;
123                 k++;
124             }
125
126             while( t )
127             {
128                 ++c[0];
129                 assert(c[0] < MM);
130                 c[c[0]]= t % 10 , t = t /10;
131             }
132
133         if( nrpc>pc )
134             Atrib(sol,c), pc = nrpc;
135         else
136             if(NrMareCmp(sol,c)<0)
137                 Atrib(sol,c);
138         }
139     }
140 }
141
142 if(pc==0)
143     Atrib(sol,0);
144
145 }
146
147 int main()
148 {
149     fin >> n;
150     assert( 1 < n && n <= 2000 );
151     int adancime_maxima=0;
152     NrMare a,b;
153     nrs = 0;
154     for(int i=1;i<=n;++i)
155     {
156         int x;
157         fin >> x;
158         assert(x>=-2 && x<=1000000000);
159         if(x== -1)
160         {
161             nrs++;
162             if(nrs > adancime_maxima)
163                 adancime_maxima = nrs;
164             stiva[nrs][0]=0;
165         }
166
167         if(x>=0)
168         {
169             if(stiva[nrs][0]==0)
170                 Atrib(stiva[nrs],x);
171             else
172             {
173                 Atrib(a,stiva[nrs]);
174                 Atrib(b,x);
175                 Plus(a,b,stiva[nrs]);
176             }
177         }
178
179         if(x== -2)
180         {
181             Atrib(a,stiva[nrs]);
182             nrs--;
183             assert(nrs >= 0);
184             if(stiva[nrs][0]==0)
185                 Atrib(stiva[nrs],a);
186             else

```

```

187         {
188             Atrib(b,stiva[nrs]);
189             Plus(a,b,stiva[nrs]);
190         }
191     }
192 }
193
194     fout << adancime_maxima << "\n";
195     Afis(fout, stiva[nrs]);
196     return 0;
197 }
```

26.4 amedie

Problema 4 - amedie

100 de puncte

Pentru o matrice A cu n linii și m coloane, ce conține numere naturale, se definește **amedia matricei** A ca fiind valoarea situată la mijlocul sirului ordonat crescător format din toate elementele matricei A , dacă numărul de elemente din acest sir este impar, respectiv cea mai mică valoare dintre cele două valori situate la mijloc, dacă numărul de elemente din acest sir este par.

Se definesc trei tipuri de operații ce pot fi aplicate matricei A :

- operația de tip 1, notată cu $L x$, ce constă în eliminarea liniei cu indicele x din matrice;
- operația de tip 2, notată cu $C y$, ce constă în eliminarea coloanei cu indicele y din matrice;
- operația de tip 3, notată cu Q , ce are ca rezultat determinarea valorii amediei matricei.

Ami este pasionată de matematică și trebuie să rezolve următoarea problemă: pentru o matrice A asupra căreia se aplică un sir de astfel de operații, să se stabilească răspunsurile la toate operațiile de tip 3.

Cerințe

Scrieți un program care determină valorile obținute în urma efectuării operațiilor de tip 3 din sirul de operații aplicate matricei A .

Date de intrare

Fisierul de intrare **amedie.in** conține pe prima linie trei numere naturale n , m și q despărțite prin câte un spațiu, ce reprezintă, în ordine, numărul de linii din matricea A , numărul de coloane din matricea A și numărul de operații ce se aplică matricei A .

Pe fiecare dintre următoarele n linii din fișier se află câte m numere naturale, separate prin câte un spațiu, ce reprezintă, în ordine, elementele aflate pe liniile matricei A .

Fiecare dintre următoarele q linii din fișier conține un sir de caractere ce reprezintă o operație ce se va aplica matricei A .

Date de ieșire

Fisierul de ieșire **amedie.out** va conține valorile obținute, în ordine, la toate operațiile Q din sirul de operații aplicate matricei A . Fiecare dintre valorile obținute se va afișa pe câte o linie din fișier, în ordinea stabilită în sirul de operații.

Restricții și precizări

- $2 \leq n, m \leq 800$;
- $0 < q \leq 2000$;
- elementele matricei A sunt numere naturale nenule, strict mai mici decât 100 001;
- indicii de linie și coloană din matrice sunt notați începând cu 1;
- operațiile de tipul 1 și 2 se aplică utilizând indicii liniilor și coloanelor din matricea A inițială;
- se garantează că orice operație din fișierul de intrare poate fi efectuată.

Exemplu:

amedie.in	amedie.out	Explicații																
4 4 4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 L 2 Q C 1 Q	10 11	L 2 Prin aplicarea operației de tip 3 obținem valoarea 10 . C 1 Prin aplicarea operației de tip 3 obținem valoarea 11 .																
		<table border="1"><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>5</td><td>6</td><td>7</td><td>8</td></tr><tr><td>9</td><td>10</td><td>11</td><td>12</td></tr><tr><td>13</td><td>14</td><td>15</td><td>16</td></tr></table>	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4															
5	6	7	8															
9	10	11	12															
13	14	15	16															
		<table border="1"><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>5</td><td>6</td><td>7</td><td>8</td></tr><tr><td>9</td><td>10</td><td>11</td><td>12</td></tr><tr><td>13</td><td>14</td><td>15</td><td>16</td></tr></table>	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4															
5	6	7	8															
9	10	11	12															
13	14	15	16															

Timp maxim de executare/test: **1.8** secunde pe Windows, **0.2** secunde pe Linux

Memorie: total **2 MB** din care pentru stivă **2 MB**

Dimensiune maximă a sursei: **15 KB**

26.4.1 Indicații de rezolvare

Andrei Ciocan, Universitatea Politehnica București

Se rețin elementele matricei în ordine crescătoare într-un vector V , mediana aflându-se inițial pe poziția $k/2 + k\%2$.

Pentru a putea efectua operațiile de tip1 și 2 în $O(n)$ și cea de tip 3 în $O(1)$, vom folosi vectorii $left[]$ și $right[]$; $right[i]$ pointând către cel mai din dreapta element din vectorul sortat care încă nu a fost eliminat, iar $left[]$ către cel din stânga.

Se mai folosește un vector $poz[valmax]$, $poz[i]$ reprezentând poziția valorii i în vectorul V .

Când eliminăm un element din vector, doar actualizăm valorile $left$ și $right$ ale vecinilor la momentul respectiv.

Dacă mediana este mai mică decât elementul ce a fost eliminat, atunci mediana trebuie mutată cu o poziție mai la dreapta ($median = right[median]$). Dacă este mai mare, atunci mediana trebuie mutată la stânga (trebuie ținut cont și de paritatea lui n).

Trebuie avut grijă să fie tratat cazul când chiar mediana este eliminată.

Complexitate timp $O(n^2 + valmax)$ și spațiu $O(n^2 + valmax)$.

26.4.2 *Rezolvare detaliată

26.4.3 Cod sursă

Listing 26.4.1: amedie.cpp

```

1 #include<iostream>
2 #include<fstream>
3 #include<algorithm>
4
5 using namespace std;
6
7 ifstream f("amedie.in");
8 ofstream g("amedie.out");
9
10 inline int cmp ( int x, int y ) { return x<y; }
11
12 int r,val,p,l,c,n,m,nr,median,valmax,q;
13
14 int poz[100005],w[700005],left1[700005],right1[700005],a[806][806],v[700005];
15
16 void elimin ()
17 {
18     int i;
```

```

19     r=poz[a[1][c]];
20     if (v[r+1]!=a[1][c]) poz[a[1][c]]=0;
21     else poz[a[1][c]]++;
22
23     left1[right1[r]]=left1[r];
24     right1[left1[r]]=right1[r];
25
26     if (nr%2==0)
27         if (median>=r)
28             median=right1[median];
29     if (nr%2==1)
30         if (median<=r)
31             median=left1[median];
32     nr--;
33 }
34
35 void solve()
36 {
37     int i,j;
38     char op;
39     nr=n*m;
40     median=nr/2 + nr%2;
41     for (;q;--q)
42     {
43         f>>op;
44         if (op=='L')
45         {
46             f>>l;
47             for(i=1;i<=m;i++)
48                 if (a[l][i]>-1)
49                 {
50                     c=i;
51                     elimin();
52                     a[l][c]=-1;
53                 }
54         }
55         if (op=='C')
56         {
57             f>>c;
58             for(i=1;i<=n;i++)
59                 if (a[i][c]>-1)
60                 {
61                     l=i;
62                     elimin();
63                     a[l][c]=-1;
64                 }
65         }
66         if (op=='Q')
67         {
68             g<<v[median]<<endl;
69         }
70     }
71 }
72
73 void preproc()
74 {
75     int i,j,ind;
76     for (i=1;i<=n;i++)
77         for(j=1;j<=m;j++)
78             { valmax=max(valmax,a[i][j]);
79               w[a[i][j]]++;
80             }
81     p=0;
82     for (i=1;i<=valmax;i++)
83         for (;w[i];w[i]--)
84             v[++p]=i;
85
86     ind=1;
87     for (i=1;i<=valmax;i++)
88     {
89         while (v[ind]<i)
90             ind++;
91         if (v[ind]==i)
92             poz[i]=ind; // poz , pozitia valorii val in vectorul v
93     }
94 }
```

```

95     for (i=1; i<=p; i++)
96     {
97         left1[i]=i-1;
98         right1[i]=i+1;
99     }
100 }
101
102 void citire()
103 {
104     int i, j;
105     f>>n>>m>>q;
106     for (i=1; i<=n; i++)
107         for (j=1; j<=m; j++)
108             f>>a[i][j];
109 }
110
111 int main ()
112 {
113     citire();
114     preproc();
115     solve();
116     f.close();
117     g.close();
118     return 0;
119 }
```

26.5 drept

Problema 5 - drept

100 de puncte

Se consideră n cartoane albastre de formă pătrată. Aceste cartoane se așează pe o foaie albă de hârtie, având formă dreptunghiulară, suficient de mare, astfel încât cartoanele să încapă în întregime pe foaie. Laturile cartoanelor sunt paralele cu laturile foii. Coordonatele vîrfurilor cartoanelor sunt numere naturale. Cartoanele pot fi suprapuse integral sau parțial. Considerăm că foaia alba are colțul stânga-jos de coordonate $(0, 0)$.

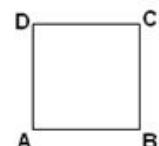
Cerințe

Să se determine:

- aria totală a suprafeței albastre care se va vedea, privind foaia de hârtie de sus;
- numărul maxim de cartoane care au cel puțin o suprapunere comună.

Date de intrare

Pe prima linie a fișierului de intrare **drept.in** se află un număr natural n , reprezentând numărul cartoanelor. Pe fiecare dintre următoarele n linii sunt câte trei numere naturale x, y și d , despărțite prin căte un spațiu, numere care descriu cartoanele. Cele trei numere x, y și d corespund unui carton pătrat $ABCD$ având următoarele coordonate: $A(x, y), B(x + d, y), C(x + d, y + d), D(x, y + d)$.



Date de ieșire

Pe prima linie a fișierului de ieșire **drept.out** se va scrie o valoare un număr natural reprezentând aria totală ocupată de cartoanele albastre pe foaia albă, iar pe a doua linie un număr natural ce reprezintă numărul maxim de cartoane care au cel puțin o suprapunere comună.

Restricții și precizări

- $1 \leq n \leq 10\ 000$
- $0 \leq x, y, d \leq 5\ 000$
- aria oricărei suprafețe albastre este strict mai mică decât 2^{31} ;
- Pentru determinarea corectă a valorii de la cerința a) se acordă 40% din punctajul acordat testului respectiv, iar pentru determinarea corectă a valorii de la cerința b) se acordă 60% din punctajul acordat testului respectiv.

Exemple:

drept.in	drept.out	Explicații
4 1 1 2 2 0 2 3 1 2 5 3 1	11 2	
4 1 1 4 2 2 3 3 3 2 4 4 1	16 4	Cele 4 cartoane se suprapun în zona delimitată de punctele $(4, 4)$, $(4, 5)$, $(5, 5)$, $(5, 4)$.

Timp maxim de executare/test: **0.2** secunde

Memorie: total **6 MB** din care pentru stivă **6 MB**

Dimensiune maximă a sursei: **10 KB**

26.5.1 Indicații de rezolvare

prof. Adrian Pintea, Inspectoratul școlar al Județului Cluj

Se poate folosi un *algoritm de baleiere*. Coordonatele a și $a+c$ reprezintă evenimente orizontale, iar b și $b+c$ evenimente verticale. Folosim 2 vectori pentru a memora aceste evenimente, după care le sortăm. Menținem un vector de 0/1, pentru a marca zonele active.

Cu o dreaptă de baleiere parcurgem pe rând evenimentele orizontale și calculăm lungimea pe verticală a zonelor active, folosind de vectorul de zone active și evenimentele orizontale. Aceasta o înmulțim cu diferența pe orizontală dintre ultimele două evenimente orizontale. Numărul maxim de cartoane suprapuse se găsește similar (verificăm zonele active).

Ordin de complexitate: $O(n^2)$

26.5.2 *Rezolvare detaliată

26.5.3 Cod sursă

Listing 26.5.1: drept.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <set>
5 #include <algorithm>
6
7 using namespace std;
8
9 ifstream fin("drept.in");
10 ofstream fout("drept.out");
11
12 #define MAXN 10005
13
14 struct drept

```

```

15  {
16      int x1,y1,x2,y2;
17  };
18
19 struct ev
20 {
21     int val,nr;
22     int tip;
23 };
24
25 bool compev(ev a, ev b)
26 {
27     if (a.val!=b.val) return (a.val<b.val);
28     else return(a.tip>b.tip);
29 }
30
31 int i,j,n,m,nra,linie,lx,ly,dreact,liniey,arie,maxact,nrevx,nrevy;
32 drept d[MAXN];
33 ev evx[20003],evy[20003];
34 ev evaux;
35
36 bitset<MAXN> activ;
37
38
39 int main()
40 {
41     fin>>n;
42     maxact=0; arie=0;
43     for(i=1;i<=n;i++)
44     {
45         fin>>d[i].x1>>d[i].y1>>nra;
46         d[i].x2=d[i].x1+nra;
47         d[i].y2=d[i].y1+nra;
48         nrevx++; evx[nrevx].val=d[i].x1; evx[nrevx].tip=0; evx[nrevx].nr=i;
49         nrevx++; evx[nrevx].val=d[i].x2; evx[nrevx].tip=1; evx[nrevx].nr=i;
50         nrevy++; evy[nrevy].val=d[i].y1; evy[nrevy].tip=0; evy[nrevy].nr=i;
51         nrevy++; evy[nrevy].val=d[i].y2; evy[nrevy].tip=1; evy[nrevy].nr=i;
52     }
53
54     sort (evx+1, evx+nrevx+1, compev);
55     sort (evy+1, evy+nrevy+1, compev);
56
57 //BALEIERE
58 linie=0;
59 for(i=1;i<=nrevx;i++)
60 {
61     lx=evx[i].val-linie;
62     ly=0;
63     dreact=0;
64     if (lx>0)
65         for(j=1;j<=nrevy;j++)
66         {
67             if (activ[evy[j].nr])
68             {
69                 if (dreact>0) ly+=evy[j].val-liniey;
70                 if (evy[j].tip==0) dreact++;
71                 else dreact--;
72                 if (dreact>maxact) maxact=dreact;
73                 liniey=evy[j].val;
74             }
75         }
76
77     arie+=lx*ly;
78     if (evx[i].tip==0) activ[evx[i].nr]=1;
79     else activ[evx[i].nr]=0;
80     linie=evx[i].val;
81 }
82
83 //SFARSIT
84 fout<<arie<<"\n"<<maxact<<"\n";
85 fout.close();
86 return 0;
87 }
```

Listing 26.5.2: M_Stroe.cpp

```

1 #include <cstdio>
2 #include <cassert>
3 #include <vector>
4 #include <algorithm>
5
6 using namespace std;
7
8 const char iname[] = "drept.in";
9 const char oname[] = "drept.out";
10
11 const int MAX_N = 10005;
12 const int MAX_COORD = 10005;
13
14 typedef struct
15 {
16     int x, y1, y2, sgn;
17 } edge;
18
19 bool edge_cmp(edge a, edge b)
20 {
21     if (a.x != b.x) return a.x < b.x;
22     return a.sgn < b.sgn;
23 }
24
25 vector <edge> edges; int n, ys[MAX_COORD];
26
27 int main(void)
28 {
29     FILE *fi = fopen(iname, "r");
30     assert(fscanf(fi, "%d", &n) == 1);
31     assert(1 <= n && n <= 10000);
32     for (int i = 0; i < n; ++ i)
33     {
34         int x, y, d;
35         assert(fscanf(fi, "%d %d %d", &x, &y, &d) == 3);
36         assert(0 <= x && x <= 5000);
37         assert(0 <= y && y <= 5000);
38         assert(1 <= d && d <= 5000);
39         edge e;
40         e.x = x, e.y1 = y, e.y2 = y + d - 1, e.sgn = +1;
41         edges.push_back(e);
42         e.x = x + d, e.y1 = y, e.y2 = y + d - 1, e.sgn = -1;
43         edges.push_back(e);
44     }
45
46     assert((int) edges.size() == n * 2);
47     sort(edges.begin(), edges.end(), edge_cmp);
48
49     int min_y = MAX_COORD, max_y = 0, prev_x = 0;
50     int area = 0, overlaps = 0;
51     for (vector <edge>::iterator it = edges.begin(); it != edges.end(); ++ it)
52     {
53         edge e = (*it);
54         int count = 0;
55         min_y = min(min_y, e.y1);
56         max_y = max(max_y, e.y2);
57         for (int j = min_y; j <= max_y; ++ j) if (ys[j] > 0)
58         {
59             count++;
60         }
61         area += count * (e.x - prev_x);
62
63         for (int j = e.y1; j <= e.y2; ++ j)
64             ys[j] += e.sgn;
65
66         for (int j = min_y; j <= max_y; ++ j) if (ys[j] > 0)
67         {
68             overlaps = max(overlaps, ys[j]);
69         }
70
71         prev_x = e.x;
72     }
73     fprintf(fopen(oname, "w"), "%d\n%d\n", area, overlaps);
74
75 fclose(fi);

```

```

76     return 0;
77 }

```

26.6 poly

Problema 6 - poly

100 de puncte

Un polyomino este o figură geometrică plană compactă formată din una sau mai multe piese de domino, pătrate egale de latură 1. Două piese se consideră alăturate dacă au o latură comună.

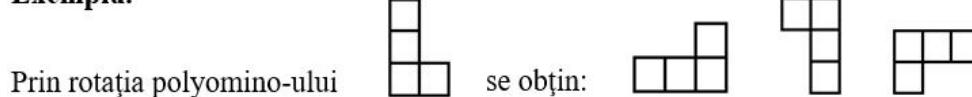
Exemplu de Polyominouri cu n piese:	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px;">$n=1$</td><td style="padding: 5px;"></td><td style="padding: 5px;">$n=3$</td><td style="padding: 5px;"></td><td style="padding: 5px;">$n=4$</td><td style="padding: 5px;"></td></tr> <tr> <td style="padding: 5px;">$n=2$</td><td style="padding: 5px;"></td><td style="padding: 5px;">$n=3$</td><td style="padding: 5px;"></td><td style="padding: 5px;">$n=4$</td><td style="padding: 5px;"></td></tr> </table>	$n=1$		$n=3$		$n=4$		$n=2$		$n=3$		$n=4$	
$n=1$		$n=3$		$n=4$									
$n=2$		$n=3$		$n=4$									

Două Polyominouri se consideră identice dacă sunt formate din același număr de piese și au aceeași configurație sau configurația unuia se poate obține prin oglindirea celuilalt. În caz contrar cele două Polyominouri se consideră distincte.

Polyominouri distincte		Polyominouri care nu sunt distincte	
------------------------	--	-------------------------------------	--

Un polyomino poate fi rotit cu 900, 1800 și 2700, în sens trigonometric. Prin rotație se obțin alte Polyominouri, nu neapărat identice cu cel inițial.

Exemplu.



Niciunul dintre aceste Polyominouri nu este identic cu cel inițial.

Niciunul dintre aceste Polyominouri nu este identic cu cel inițial. Un polyomino este convex dacă prin parcurgerea succesivă a linilor sau coloanelor nu se întâlnesc găuri.

Polyomino convex		Polyominouri neconvexe	
------------------	--	------------------------	--

Un polyomino este oblic convex (skew polyomino) dacă este convex și prin parcurgerea succesivă a coloanelor de la stânga la dreapta acestea nu descresc în înălțime. Altfel spus, partea de jos a coloanei din stânga este întotdeauna mai mică sau egală ca înălțime cu partea de jos a coloanei din dreapta. În mod similar, partea de sus a coloanei din stânga este întotdeauna mai mică sau egală cu partea de sus a coloanei din dreapta.

Exemplu:

Polyominouri oblice		Polyominouri care nu sunt oblice	
---------------------	--	----------------------------------	--

Cerințe

Să se determine numărul de Polyominouri oblice distincte care au perimetru egal cu $2 * N + 2$. Acest număr poate să fie mare, de aceea ne interesează rezultatul modulo 30103.

Date de intrare

Fișierul de intrare **poly.in** conține pe prima linie un număr natural N .

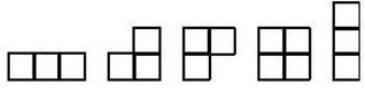
Date de ieșire

Fișierul de ieșire **poly.out** conține pe prima linie un număr natural ce reprezintă numărul de Polyominouri oblice care au perimetru egal cu $2 * N + 2$, număr afișat modulo 30103.

Restricții și precizări

- $2 \leq N \leq 500$

Exemplu:

poly.in	poly.out	Explicații
3	5	Sunt 5 Polyominouri care respectă cerința: 

Timp maxim de executare/test: **0.2** secunde

Memorie: total **8 MB** din care pentru stivă **8 MB**

Dimensiune maximă a sursei: **10 KB**

26.6.1 Indicații de rezolvare

prof. Eugen Nodea, C.N. "Tudor Vladimirescu", Tg.Jiu

Numărul de polyominoes oblice = C_n numărul lui Catalan

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$$

Numărul Catalan se poate obține :

1. folosind *triunghiul combinărilor al lui Pascal*. Astfel, C_n se obține din diferența dintre termenul n de pe linia $2n$ a triunghiului lui Pascal și termenul din stânga sa.

$$C(n) = \text{Comb}(2n, n) - \text{Comb}(2n, n-1)$$

Soluție care obține (60 p). Sursă: Eugen Nodea

2. folosind formula recursivă

$$C(n) = C(0) * C(n-1) + C(1) * C(n-2) + C(2) * C(n-3) + \dots + C(n-1) * C(0)$$

Soluție care obține (100 p).

Surse : Eugen Nodea, Marius Stroe

3. folosind *backtracking*

Soluție care obține (50 p).

Sursă : Marius Stroe

26.6.2 *Rezolvare detaliată

26.6.3 Cod sursă

Listing 26.6.1: poly_bkt.cpp

```

1  /*
2   *      Solutie 50 p
3   *      backtracking
4   *      stud. Marius Stroe
5  */
6  #include <cstdio>
7  #include <cassert>
8  #include <algorithm>
9
10 using namespace std;
11
12 const char iname[] = "poly.in";
13 const char oname[] = "poly.out";
14
15 const int MAX_N = 1 << 16;
16 const int Modulo = 30103;
17
18

```

```

19 int headers[MAX_N][2], n, answer = 0;
20
21
22 void bkt(int i, int perimeter)
23 {
24     if (perimeter == 2 * n + 2)
25         answer++;
26     else
27     {
28         if (i == 0)
29         {
30             for (int end = 1; end <= n; ++ end)
31             {
32                 headers[i][0] = 1, headers[i][1] = end;
33                 bkt(i + 1, 2 * end + 2);
34             }
35         }
36     else
37     {
38         int low = headers[i - 1][0], high = headers[i - 1][1];
39         for (int start = low; start <= high; ++ start)
40         {
41             for (int end = high; end <= n; ++ end)
42             {
43                 int new_perimeter = perimeter - (high - start + 1) +
44                             (end - high) + (end - start + 1) + 2;
45                 if (new_perimeter > 2 * n + 2)
46                     break;
47
48                 headers[i][0] = start, headers[i][1] = end;
49                 bkt(i + 1, new_perimeter);
50             }
51         }
52     }
53 }
54
55
56 int main(void)
57 {
58     FILE *fi = fopen(iname, "r");
59     assert(fscanf(fi, "%d", &n) == 1);
60     assert(2 <= n && n <= 15000);
61     fclose(fi);
62
63     bkt(0, 0);
64     fprintf(fopen(oname, "w"), "%d\n", answer % Modulo);
65
66     return 0;
67 }
```

Listing 26.6.2: poly_n2.cpp

```

1 /*
2     Solutie 50 p
3     recursiv
4     stud. Marius Stroe
5 */
6 #include <cstdio>
7 #include <cassert>
8 #include <algorithm>
9
10 using namespace std;
11
12 const char iname[] = "poly.in";
13 const char oname[] = "poly.out";
14
15 typedef long long i64;
16
17 const int MAX_N = 1 << 16;
18 const i64 Modulo = 30103;
19
20 int n;
21
22 int main(void)
23 {
```

```

24     FILE *fi = fopen(iname, "r");
25     assert(fscanf(fi, "%d", &n) == 1);
26     assert(2 <= n && n <= 15000);
27     fclose(fi);
28
29     i64 C[MAX_N];
30
31     C[0] = 1;
32
33     for (int i = 0; i < n; ++ i)
34     {
35         C[i + 1] = 0;
36         for (int j = 0; j <= i; ++ j)
37             C[i + 1] = (C[i + 1] + C[j] * C[i - j]) % Modulo;
38     }
39
40     fprintf(fopen(oname, "w"), "%lld\n", C[n]);
41
42     return 0;
43 }
```

Listing 26.6.3: poly_recursiv.cpp

```

1 # include <cstdio>
2 # include <cstring>
3 # define M 30103
4
5 int N, i;
6 unsigned long long C[15001];
7
8 /*
9 Catalan(n) = Catalan(0)*Catalan(n-1) +
10           Catalan(1)*Catalan(n-2) +
11           Catalan(2)*Catalan(n-3) +
12           ...
13           int catalan(int N)
14 */
15 int catalan(int N)
16 {
17     int i, j;
18     memset(C, 0, sizeof(C));
19     C[0] = (unsigned long long) 1;
20     for (i=1; i<=N; ++i)
21         for (j=0; j<i; ++j)
22             C[i] = (C[i] % M + (C[j]*C[i-j-1]) % M) % M;
23     return C[N];
24 }
25
26 int main()
27 {
28     freopen("poly.in", "r", stdin);
29     freopen("poly.out", "w", stdout);
30
31     scanf("%d", &N);
32     printf("%d\n", catalan(N));
33
34     return 0;
35 }
```

Listing 26.6.4: poly_triunghi.cpp

```

1 /*
2  Solutie 60 p
3
4  calc.combinari folosind triunghiul lui Pascal
5
6  prof. Eugen Nodea
7 */
8
9 # include <cstdio>
10 # include <cstring>
11
12 # define M 30103
13 # define nmax 1001
```

```
14
15 int c[nmax][nmax], N;
16
17 int catalan_(int N)
18 {
19     int i,j;
20     //C(n) = Comb(2n, n) - Comb(2n, n-1)
21     // calcul combinari folosind triunghiul lui Pascal
22     c[0][0] = 1;
23     c[1][1] = 1;
24     for (i = 2; i<=2 * N; ++i)
25         for (j=1; j<= N; ++j)
26             c[i][j] = (c[i-1][j] % M + c[i-1][j-1] % M);
27     return (int)(c[2 * N][N] % M - c[2 * N][N-1] % M) % M ;
28 }
29
30 int main()
31 {
32     freopen("poly.in", "r", stdin);
33     freopen("poly.out", "w", stdout);
34
35     scanf("%d", &N);
36     printf("%d\n", catalan_(N));
37
38     return 0;
39 }
```

Capitolul 27

ONI 2011

27.1 mxl

Problema 1 - mxl

100 de puncte

Firma MicroPhone, producător de aplicații software pentru telefoane mobile, dezvoltă programul de calcul tabelar MicroXL pentru a-l introduce în software-ul de bază al viitoarei generații de telefoane inteligente. În stadiul actual, aplicația permite lucrul pe o foaie de calcul compusă din $N \times N$ celule, în fiecare celulă putându-se introduce un număr natural nenul sau o formulă.

O formulă este precedată de semnul egal și conține un număr arbitrar, posibil zero, de operații de adunare. Termenii unei formule pot fi constante naturale nenule sau referințe la alte celule, așa cum se poate observa în figura de mai jos. Referința la o celulă se realizează specificând numărul liniei și al coloanei corespunzătoare, separate prin caracterul ":". Formula nu conține spații.

	1	2	3	4	5
1	=1:2+5	=1:3+4	=3+1:4	=1:5+2	1
2					
3	=13				
4			=1+2+3		
5					

Cerințe

Scrieți un program care determină valoarea fiecărei celule dintr-o foaie de calcul dată.

Date de intrare

Fișierul **mxl.in** conține pe prima linie numerele naturale nenule N și K , reprezentând numărul de linii și coloane ale foii de calcul, respectiv numărul celulelor ce conțin date, iar pe următoarele K linii câte două numere întregi lin , col și un sir de caractere s , cu semnificația: lin și col reprezintă linia, respectiv coloana corespunzătoare unei celule, iar s este conținutul celulei respective (număr natural nenul sau formulă).

Date de ieșire

Fișierul **mxl.out** va conține N linii. Pe fiecare linie i se vor afla câte N numere naturale, separate prin câte un spătiu, reprezentând valorile calculate pentru fiecare celulă de pe linia i a foii de calcul. Dacă o celulă nu conține date sau formule se va afișa valoarea 0.

Restricții și precizări

- $0 < N \leq 40$
- $0 < K \leq NN$
- Constantele utilizate sunt numere naturale nenule mai mici sau egale cu 75
- Formulele au o lungime de cel mult 255 caractere și pot fi calculate întotdeauna
- Nu există referințe circulare

Exemplu:

mxl.in	mxl.out	Explicații
5 7	15 10 6 3 1	
1 1 =1:2+5	0 0 0 0 0	
1 2 =1:3+4	13 0 0 0 0	
1 3 =3+1:4	0 0 6 0 0	
1 4 =1:5+2	0 0 0 0 0	
1 5 1		
4 3 =1+2+3		
3 1 =13		

Timp maxim de executare/test: **0.1** secunde

Memorie: total **6 MB**

Dimensiune maximă a sursei: **5 KB**

27.1.1 Indicații de rezolvare

prof. Alin Burța, C.N. "B.P. Hasdeu" Buzău

Utilizăm tablourile bidimensionale:

S - memorează conținutul celulelor foii de calcul sub forma unui sir de caractere

Tab - memorează valorile întregi calculate la un moment dat (rezultatele formulelor pentru care calculul s-a finalizat ori valorile constante aflate în celulele foii de calcul),

Determinarea valorilor formulelor se realizează prin parcurgeri succesive ale foii de calcul și rezolvarea tuturor formulelor ce pot fi calculate la acel moment, până când sunt determinate valorile tuturor formulelor.

Formulele care nu conțin referințe sau care conțin referinte la celule cu valori constante sunt calculate de la prima parcurgere.

Complexitatea soluției este $O(N * N * M)$, unde M reprezintă numărul maxim de celule ce pot fi referite într-o formulă.

27.1.2 *Rezolvare detaliată

27.1.3 Cod sursă

Listing 27.1.1: mxl.cpp

```

1 // implementare - prof. Alin Burta
2 #include <fstream>
3 #include <cstring>
4
5 #define Fin "mxl.in"
6 #define Fou "mxl.out"
7 #define Max 255
8 #define NMax 41
9
10 using namespace std;
11
12 int N, K;
13 long long Tab[NMax][NMax];
14 char *S[NMax][NMax];
15
16 char *expand(char *Sir)
17 {
18     char *left, *right;
19     left = new char[Max];
20     right = new char[Max];
21     right = strchr(Sir, ':')+1;
22     strncpy(left, Sir, strchr(Sir, ':')-Sir);
23     left[strchr(Sir, ':')-Sir]='\0';
24     return S[atoi(left)][atoi(right)];
25 }
26

```

```

27 void lgtoa(long long x,char *s)
28 {
29     int i=0;
30     char ch;
31     strcpy(s,"");
32 // s = new char[24];
33     while(x)
34     {
35         s[i++] = x % 10 + 48;
36         x/=10;
37     }
38     s[i] = '\0';
39     for(i=0;i<strlen(s)/2;i++)
40     {
41         ch = s[strlen(s)-i-1];
42         s[strlen(s)-i-1] = s[i];
43         s[i] = ch;
44     }
45 }
46
47 long long ValCelula(char *Sir)
48 {
49     //determina valoarea din celula specificata
50     char *left, *right;
51     left = new char[Max];
52     right = new char[Max];
53     right = strchr(Sir,':')+1;
54     strncpy(left, Sir, strchr(Sir,':')-Sir);
55     left[strchr(Sir,':')-Sir]='\0';
56     return Tab[atol(left)][atol(right)];
57 }
58
59 int rezolva()
60 {
61     char *tmp = new char[Max];
62     char *termen;
63     char *Copy = new char[Max];
64     int i,j, ok, OKT=0;
65     long long sum;
66     for(i=1;i<=N;i++)
67     {
68         for(j=1;j<=N;j++)
69         {
70             if(strlen(S[i][j]) && strchr(S[i][j],'='))
71             {
72                 OKT = 1;
73                 sum = 0; ok = 1; //pp ca se poate calcula
74                 strcpy(Copy, S[i][j]+1);
75                 termen = strtok(Copy, "+");
76                 while (termen)
77                 {
78                     if(strchr(termen,:))
79                         if(!strchr(expand(termen),'='))
80                             sum+=ValCelula(termen);
81                         else ok=0;
82                     else sum+=atoll(termen);
83                     termen = strtok(NULL,"+");
84                 }
85             }
86             if(ok) {Tab[i][j]=sum; lgtoa(sum, tmp); S[i][j]=strupr(tmp);}
87         }
88     }
89     return OKT;
90 }
91
92 int main()
93 {
94     ifstream f(Fin);
95     ofstream g(Fou);
96     int i, j, Lin, Col;
97     char Val[Max];
98     //citesc datele de intrare
99     f>>N>>K; f.get();
100    for(i=1;i<=N;i++)
101        for(j=1;j<=N;j++) { Tab[i][j]=0; S[i][j]=strupr(""); }
102    for(i=1;i<=K;i++)
103    {
104        f>>Lin; f.get();
105        f>>Col; f.get();

```

```

103     f.get(Val, Max); f.get();
104     if(!strchr(Val, '='))
105     {
106         Tab[Lin][Col]=atoll(Val);
107         S[Lin][Col] = strdup(Val);
108     }
109     else
110     {
111         S[Lin][Col]=strdup(Val);
112     }
113 }
114
115 while (rezolva());
116
117
118 for(i=1;i<=N;i++)
119 {
120     for(j=1;j<N;j++) g<<Tab[i][j]<<' ';
121     g<<Tab[i][N]<<'\n';
122 }
123
124 f.close();
125 g.close();
126 return 0;
127 }
```

Listing 27.1.2: mxl1.cpp

```

1 // 100 puncte
2 // Implementare - Constantin Galatan
3 #include <iomanip>
4 #include <iostream>
5 #include <fstream>
6 #include <string>
7 #include <cstdlib>
8 #include <vector>
9 #include <map>
10
11 using namespace std;
12
13 #define DIM 50
14 #define pb push_back
15 #define mp make_pair
16
17 ifstream fin("mxl.in");
18 ofstream fout("mxl.out");
19
20 int N, K;
21 string Tab[DIM][DIM];
22
23 struct State
24 {
25     int i, j;
26     string expr;
27     State(int _i,int _j, string _expr)
28         : i(_i), j(_j), expr(_expr)
29     {
30     }
31     bool operator< (const State& st) const
32     {
33         return expr < st.expr;
34     }
35 };
36
37 map<State, long long> memo;
38 vector<pair<int, int> > cells;
39
40 void Read();
41 long long Value(int i, int j, string expr);
42 void Solve();
43 void Write();
44
45 int main()
46 {
47     Read();
```

```

48     Solve();
49     Write();
50
51     return 0;
52 }
53
54 void Write()
55 {
56     for ( int i = 1; i <= N; ++i)
57     {
58         for ( int j = 1; j <= N; ++j )
59             fout << memo[State(i, j, Tab[i][j])] << ' ';
60         fout << '\n';
61     }
62     fout.close();
63 }
64
65 void Read()
66 {
67     fin >> N >> K;
68     int i, j;
69     string expr;
70     for ( int k = 0; k < K; ++k )
71     {
72         fin >> i >> j; fin.get();
73         cells.pb(mp(i, j));
74         getline(fin, expr);
75         Tab[i][j] = expr;
76     }
77     fin.close();
78 }
79
80 void Solve()
81 {
82     int i, j;
83     for (int c = 0; c < (int)cells.size(); c++ )
84     {
85         i = cells[c].first; j = cells[c].second;
86         Value(i, j, Tab[i][j]);
87     }
88 }
89
90 long long Value(int i, int j, string expr)
91 {
92     State s(i, j, expr);
93     if ( memo.find(s) != memo.end() )
94         return memo[s];
95
96     if ( expr[0] == '=' )
97         return memo[s] = Value(i, j, expr.substr(1));
98
99     if ( expr.find_first_of("+:") == string::npos ) // nu apare + sau :
100    {
101        int x = atoi(expr.c_str()); // deci e numar
102        return memo[s] = x;
103    }
104
105 // daca expresia nu incepe cu = , caut +
106 int poz_plus = expr.find('+');
107 if ( poz_plus != (int)string::npos ) // am gasit un +
108 {
109     string t = expr.substr(0, poz_plus);
110
111     if(t.find_first_of("+:") == string::npos) // daca primul termenul e numar
112     {
113         long x = atol(t.c_str());
114         memo[s] = x;
115     }
116     else // primul termen e referinta
117     {
118         int poz_doua_puncte = t.find(':');
119         int ii = atoi(t.substr(0, poz_doua_puncte).c_str());
120         int jj = atoi(t.substr(poz_doua_puncte + 1,
121                               poz_plus - poz_doua_puncte - 1).c_str());
122         memo[s] = Value(ii, jj, Tab[ii][jj]);
123     }
}

```

```

124     memo[s] += Value(i, j, expr.substr(poz_plus + 1));
125 }
126 else // expr e referinta (numar nu poate fi - s-a tratat cazul la intrare)
127 {
128     int poz_doua_puncte = expr.find(':');
129     int ii = atoi(expr.substr(0, poz_doua_puncte).c_str());
130     int jj = atoi(expr.substr(poz_doua_puncte + 1,
131                             poz_plus - poz_doua_puncte - 1).c_str());
132     memo[s] = Value(ii, jj, Tab[ii][jj]);
133 }
134
135
136 return memo[s];
137 }
```

27.2 segmente

Problema 2 - segmente

100 de puncte

Considerăm N segmente în planul XOY . Segmentele sunt fie orizontale (paralele cu axa OX), fie verticale (paralele cu axa OY) și au capetele în puncte de coordonate întregi. De asemenea, nu există două segmente orizontale situate pe aceeași dreaptă și nici două segmente verticale situate pe aceeași dreaptă. Tuturor segmentelor li se poate aplica simultan o operație de prelungire la ambele capete cu o valoare întreagă D . Pentru orice segment, dacă lungimea este L , atunci după prelungire dimensiunea sa este $L + 2 * D$.

Cerințe

Să se determine valoarea minimă D cu care trebuie prelungite segmentele astfel încât după prelungire segmentele să închidă cel puțin un dreptunghi.

Date de intrare

Pe prima linie a fișierului **segmente.in** se află un număr natural N reprezentând numărul de segmente. Pe fiecare din următoarele N linii se găsesc câte patru numere întregi $x_1 \ y_1 \ x_2 \ y_2$. Primele două reprezintă un capăt al segmentului, iar ultimele două celălalt capăt.

Date de ieșire

Pe prima linie a fișierului **segmente.out**, se va scrie un număr natural D reprezentând dimensiunea minimă necesară prelungirii tuturor segmentelor pentru a se forma cel puțin un dreptunghi.

Restricții și precizări

- $4 \leq N \leq 1000$
- capetele segmentelor sunt numere întregi din intervalul $[-500\ 000\ 000, 500\ 000\ 000]$
- Orice segment are lungimea inițială de cel puțin 1
- Pentru datele de intrare, nu există inițial niciun dreptunghi deja format; de asemenea, vor exista cel puțin 2 segmente verticale și cel puțin două orizontale
- Se garantează că există o soluție cu $1 \leq D \leq 1\ 000\ 000\ 000$
- Pentru 50% din teste, $N \leq 200$

Exemplu:

segmente.in	segmente.out	Explicații
5 1 2 1 3 3 2 4 2 2 3 2 5 5 2 5 7 5 6 7 6	3	<p>Segmentele inițiale sunt cele îngroșate. Cu linie punctată sunt extinderile cu 3 unități la ambele capete ale tuturor segmentelor, iar hașurat este marcat dreptunghiul care s-a format după prelungirea cu $D = 3$.</p>

Timp maxim de executare/test: **2.0** secunde

Memorie: total **16 MB**

Dimensiune maximă a sursei: **5 KB**

27.2.1 Indicații de rezolvare

Dan Pracsu

Soluția 1 (100 puncte - Stelian Ciurea)

Se construiește o matrice matov în care se determină pentru fiecare pereche de segmente orizontal/vertical distanța până la punctul de intersecție. Se vor depune în matrice 4 valori:

1 - se intersectează $\Rightarrow 0$,

2, 3 - prelungirea unuia îl intersectează pe celălalt (două cazuri aici) \Rightarrow distanța de la unul din capete la segmentul celălalt

4 - ambele trebuie alungite ca să se intersecteze \Rightarrow maximul dintre cele două bucați cu care trebuie alungite

Să presupunem că cele orizontale sunt mai puține. Atunci se ia fiecare pereche de două segmente orizontale și se determină pentru această pereche și fiecare segment vertical care e alungirea ca să se intersecteze ambele cu respectivul (maximul dintre cele două valori corespunzătoare din matov). Se rețin cele mai mici două valori care reprezintă alungirea cea mai mică astfel încât să obținem un dreptunghi cu cele două orizontale fixate și două verticale. Minimul dintre alungirile astea reprezintă soluția.

Dacă numărul de verticale este mai mic, atunci se ia fiecare pereche de verticale cu fiecare orizontal.

Deci complexitatea este $O(n_o * n_o * n_v)$ sau $O(n_v * n_v * n_o)$

Soluția 2 (70 puncte - Dan Pracsu)

Separăm segmentele verticale (vectorul V) de cele orizontale (vectorul H). Sortăm apoi segmentele verticale după x .

Trebuie mai întâi să determinăm pentru o valoare dată D dacă din segmentele lungite la ambele capete cu D se formează măcar un dreptunghi.

Pentru aceasta luăm oricare două segmente verticale și stabilim pentru ele valoarea maximă a capetelor lor de jos și valoarea minimă a capetelor lor de sus (aceste două valori ne folosesc pentru a verifica dacă segmentele orizontale intersectează ambele segmente).

Parcurgem apoi lista H a segmentelor orizontale și verificăm câte dintre acestea intersectează ambele segmente verticale. Dacă sunt cel puțin două, atunci s-a format un dreptunghi. Complexitatea operației este deci în total $O(nV * nV * nH)$, unde nV este numărul segmentelor verticale, iar nH numărul celor orizontale.

Pentru a găsi valoarea D minimă facem acum o căutare binară în intervalul $[1..1\ 000\ 000\ 000]$ a lui D și verificăm formarea dreptunghiului.

Căutarea binară are cel mult 30 de pași.

Complexitatea finală va fi deci $O(nV * nV * nH * \log(1\ 000\ 000\ 000))$, iar memorie $O(N)$.

27.2.2 *Rezolvare detaliată

27.2.3 Cod sursă

Listing 27.2.1: segmente.cpp

```

1 // segmente - implementare Stelian Ciurea
2 // complexitate: O(nH*nH*nV) sau O(nV*nV*nH) unde
3 // nV - nr segm verticale, nH - nr segm. orizontale
4 #include <fstream>
5 #include <iostream>
6 #include <vector>
7
8 #define nmax 1000
9
10 using namespace std;
11
12 int distoov,min1,min2,mini;
13 unsigned i,i1,i2,n,j,nv,nh;
14
15 struct segment { int x1,y1,x2,y2;};
16
17 segment seg;
18 segment h[nmax+1],v[nmax+1];
19
20 int matov[nmax+1][nmax+1];
21
22 ifstream f("segmente.in");
23 ofstream g("segmente.out");
24
25 int distintersectie(segment a, segment b) //a vertical, b orizontal
26 {
27     int rez;
28     if (b.x1<=a.x1 && a.x1 <= b.x2 &&
29         a.y1 <= b.y1 && b.y1 <= a.y2) //se intersecteaza
30         return 0;
31
32     if (b.x1<=a.x1 && a.x1 <= b.x2)           //prelungirea lui a taie b
33     {
34         rez = min(abs(b.y1 - a.y1), abs(b.y1-a.y2));
35         return rez;
36     }
37
38     if (a.y1 <= b.y1 && b.y1 <= a.y2)           //prelungirea lui b taie a
39     {
40         rez = min(abs(b.x1 - a.x1), abs(b.x2 - a.x1));
41         return rez;
42     }
43
44     int rez1 = min(abs(b.y1 - a.y1), abs(b.y1 - a.y2));
45     int rez2 = min(abs(b.x1 - a.x1), abs(b.x2 - a.x1));
46     rez = max(rez1,rez2);
47
48     return rez;
49 }
50
51 int main()
52 {
53     f >> n;
54     for (i=1;i<=n;i++)
55     {
56         f >> seg.x1 >> seg.y1 >> seg.x2 >> seg.y2;
57         if (seg.x1 == seg.x2)
58             v[nv++] = seg;
59         else
60             h[nh++] = seg;
61     }
62
63     for (i=0;i<nh;i++)
64         for (j=0;j<nv;j++)
65         {

```

```

66         matov[i][j] = distintersectie(v[j], h[i]);
67     }
68
69     mini = INT_MAX;
70     if (nh <= nv)
71         for (i1=0;i1<nh;i1++)
72             for (i2=i1+1;i2<nh;i2++)
73             {
74                 min1=min2=INT_MAX;
75                 for (j=0;j<nv;j++)
76                 {
77                     distoov = max(matov[i1][j],matov[i2][j]);
78                     if (distoov < min1)
79                         min2 = min1, min1 = distoov;
80                     else
81                         if (distoov<min2)
82                             min2 = distoov;
83                 }
84                 if (mini > min2)
85                     mini = min2;
86             }
87         else
88             for (i1=0;i1<nv;i1++)
89                 for (i2=i1+1;i2<nv;i2++)
90                 {
91                     min1=min2=INT_MAX;
92                     for (j=0;j<nh;j++)
93                     {
94                         distoov = max(matov[j][i1],matov[j][i2]);
95                         if (distoov < min1)
96                             min2 = min1, min1 = distoov;
97                         else
98                             if (distoov<min2)
99                                 min2 = distoov;
100                }
101                if (mini > min2)
102                    mini = min2;
103            }
104
105 cout << mini << endl;
106 g << mini << endl;
107 return 0;
108 }
```

Listing 27.2.2: segmentel.cpp

```

1 /*
2      implementare Dan Pracsiu (70 puncte)
3      O(nV * nV * nH * log 1000 000 000)
4 */
5 #include<iostream>
6
7 #define InFile "segmente.in"
8 #define OutFile "segmente.out"
9
10 using namespace std ;
11
12 struct segment
13 {
14     int x1, y1, x2, y2 ;
15 };
16
17 segment V[1001] ; // memorez segmentele verticale
18 segment H[1001] ; // memorez segmentele orizontale
19 int nV, nH ; // numar segmente verticale si orizontale
20 int lungimeMinima ;
21
22 void Citire()
23 {
24     int i, n, a, b, c, d, aux;
25
26     ifstream fin(InFile) ;
27     fin >> n ;
28     for (i = 1 ; i <= n ; i++)
29     {
```

```

30         fin >> a >> b >> c >> d ;
31         if (b == d) // segment orizontal
32         {
33             if (a > c) {aux = a; a = c; c = aux;}
34             H[nH].y1 = H[nH].y2 = b ;
35             H[nH].x1 = a ;
36             H[nH].x2 = c ;
37             nH++ ;
38         }
39     else // segment vertical
40     {
41         if (b > d) {aux = b; b = d; d = aux;}
42         V[nV].x1 = V[nV].x2 = a ;
43         V[nV].y1 = b ;
44         V[nV].y2 = d ;
45         nV++ ;
46     }
47 }
48 fin.close() ;
49 }
50
51 // sortez segmentele verticale crescator dupa abscisa
52 void SortV()
53 {
54     int i, j ;
55     segment s ;
56     for (i = 0 ; i < nV - 1 ; i++)
57         for (j = i + 1 ; j < nV ; j++)
58             if (V[i].x1 > V[j].x1)
59             {
60                 s = V[i] ;
61                 V[i] = V[j] ;
62                 V[j] = s ;
63             }
64 }
65
66 inline int Maxim(int a, int b)
67 {
68     if (a > b) return a ;
69     return b ;
70 }
71
72 inline int Minim(int a, int b)
73 {
74     if (a < b) return a ;
75     return b ;
76 }
77
78 // caut daca se formeaza un dreptunghi daca lungesc segmentele cu valoarea val
79 int Cauta(int val)
80 {
81     int i, j, p, jos, sus, contor ;
82     for (i = 0 ; i < nV - 1 ; i++)
83         for (j = i + 1 ; j < nV ; j++)
84         {
85             jos = Maxim(V[i].y1 - val, V[j].y1 - val) ;
86             sus = Minim(V[i].y2 + val, V[j].y2 + val) ;
87             if (jos < sus)
88             {
89                 contor = 0 ;
90                 for (p = 0 ; (p < nH) && (contor < 2) ; p++)
91                     if((H[p].x1 - val <= V[i].x1)
92                         && (H[p].x2 + val >= V[j].x1)
93                         && (H[p].y1 >= jos) && (H[p].y1 <= sus) )
94                         contor++ ;
95                 if (contor >= 2)
96                     return 1 ;
97             }
98         }
99     return 0 ;
100 }
101
102 // caut binar valoarea minima de lungit segmentele
103 void CautareBinara()
104 {
105     int st, dr, mij ;

```

```

106     st = 1 ; dr = 1000000000 ;
107     while (st <= dr)
108     {
109         mij = (st + dr) / 2 ;
110         if (Cauta(mij) == 1)
111         {
112             lungimeMinima = mij ;
113             dr = mij - 1 ;
114         }
115         else st = mij + 1 ;
116     }
117 }
118
119 void Solutie()
120 {
121     ofstream fout(OutFile) ;
122     fout << lungimeMinima << "\n" ;
123     fout.close() ;
124 }
125
126 int main()
127 {
128     Citire() ;
129     SortV() ;
130     CautareBinara() ;
131     Solutie() ;
132
133     return 0 ;
134 }
```

27.3 tsunami

Problema 3 - tsunami

100 de puncte

Tsunamiul este *valul mareic* ce se propagă prin apă oceanelor/mărilor, ca urmare a producerii unor erupții subacvatice sau/și a unor cutremure submarine sau de coastă foarte puternice.

Cercetătorii doresc să preîntâmpine efectele unor posibile valuri mareice prin marcarea și clasicarea zonelor al căror risc de inundare este ridicat.

Teritoriul studiat a fost împărțit în $n \times m$ pătrate identice (zone) rezultând o hartă digitizată, reprezentată sub forma unui tablou bidimensional cu n linii și m coloane, fiecare element al tabloului memorând cota (înălțimea) terenului din pătratul unitate corespunzător. Zonele de apă au cota 0, iar zonele de uscat au cote mai mari decât 0.

Orice tsunami este clasificat în funcție de înălțimea valului mareic, pe o scară de la 1 la 10. Cercetătorii doresc să marcheze zonele de risc ce pot fi afectate de un potențial tsunami.

Initial, valul mareic apare în toate zonele de cotă 0 vecine cu cel puțin o zonă de uscat. O zonă teritorială poate fi afectată dacă are cota strict mai mică decât înălțimea valului mareic și se află în vecinătatea unei zone afectate. Două pătrate unitate se învecinează dacă au o latură comună.

Cerințe

Dață fiind harta digitizată a zonelor monitorizate, să se determine numărul zonelor de uscat afectate de un tsunami de înălțime h .

Date de intrare

Fișierul de intrare **tsunami.in** conține pe prima linie trei numere naturale n , m și h separate prin câte un spațiu, reprezentând dimensiunile hărții, respectiv înălțimea valului mareic. Pe următoarele n linii sunt scrise câte m numere naturale separate prin câte un spațiu reprezentând, în ordine, cotele din cele $n \times m$ pătrate teritoriale unitate ale hărții.

Date de ieșire

Fișierul de ieșire **tsunami.out** conține o singură valoare ce reprezintă numărul pătratelor unitate afectate de un tsunami de înălțime h .

Restricții și precizări

- $2 \leq n, m \leq 1000$
- $1 \leq h \leq 10$
- Zona monitorizată nu conține lacuri interioare (pătrate unitate învecinate, având cota 0, înconjurate complet de pătrate unitate având cote strict mai mari decât 0)
- Cotele sunt numere naturale $\leq 1\,000$

Exemplu:

tsunami.in	tsunami.out	Explicații
6 7 3	6	Zonele inundate sunt reprezentate îngroșat în tabloul bidimensional:
0 0 4 2 5 0 0		0 0 4 2 5 0 0
1 0 0 7 3 6 0		1 0 0 7 3 6 0
2 3 0 5 2 2 0		2 3 0 5 2 2 0
0 7 5 4 0 0 0		0 7 5 4 0 0 0
0 5 2 3 0 2 0		0 5 2 3 0 2 0
0 4 4 8 0 2 0		0 4 4 8 0 2 0

Timp maxim de executare/test: **1.0** secunde

Memorie: total **16 MB**

Dimensiune maximă a sursei: **5 KB**

27.3.1 Indicații de rezolvare

prof. Eugen Nodea

Se caută puncte care au ieșire la ocean/mare și a căror cotă poate fi inundată.

Se aplică un *algoritm de fill*.

Orice punct întâlnit a cărui cotă poate fi inundată, punct vecin cu teritoriul actual inundat se pună într-o *coadă* (*queue*).

Cât timp coada nu este vidă se caută puncte posibile care pot fi inundate.

Trebuie avut în vedere că pot fi zone teritoriale cu cotă mai mică decât h , dar care nu pot fi inundate (afectate).

27.3.2 *Rezolvare detaliată

27.3.3 Cod sursă

Listing 27.3.1: tsunami.cpp

```

1  /*
2   *          Eugen Nodea
3   *          Implementare - fill pe coada
4   */
5  # include <cstdio>
6  # include <queue>
7
8  # define nmax 1002
9  # define mmax 1002
10
11 using namespace std;
12
13 // evident ca se poate renunta la o matrice
14 int n,m,h,A[nmax][mmax],i,j,M[nmax][mmax],ic,jc,sol,x,y,k;
15
16 int dx[] = {0, 1, 0,-1};
17 int dy[] = {1, 0,-1, 0};
18
19 typedef pair<short int, short int> celula;
20
21 queue <celula> q;
```

```

22 //Verific daca un patrat este adiacent cu o zona de apa
23 inline int vecini(int i, int j)
24 {
25     int k,x,y;
26     for (k=0; k<4; ++k)
27     {
28         x = i + dx[k];
29         y = j + dy[k];
30         if (A[x][y]==0) return 1;
31     }
32     return 0;
33 }
34 }
35
36 //construim coada cu patrate adiacente cu patrate inundate
37 void add(int i, int j)
38 {
39     int k,x,y;
40     for (k=0; k<4; ++k)
41     {
42         x = i + dx[k];
43         y = j + dy[k];
44         if (A[x][y] != A[i][j] && A[x][y] > 0 &&
45             A[x][y] < h && M[x][y] == 0)
46         {
47             q.push(make_pair(x, y));
48         }
49     }
50 }
51
52 void fill(int i, int j)
53 {
54     int k,x,y;
55     M[i][j] = h - A[i][j];
56     ++sol;
57     add(i,j);
58     for (k=0; k<4; ++k)
59     {
60         x = i + dx[k];
61         y = j + dy[k];
62         if (A[i][j] == A[x][y] && M[x][y] == 0) fill(x,y);
63     }
64 }
65
66 int main()
67 {
68     freopen("tsunami.in", "r", stdin);
69     freopen("tsunami.out", "w", stdout);
70
71     scanf("%d %d %d", &n, &m, &h);
72     for (i=1; i<=n; ++i)
73         for (j=1; j<=m; ++j)
74             scanf ("%d ",&A[i][j]);
75
76     //bordez matricea
77     for (i=0; i<=n+1; ++i)
78         A[i][0] = A[i][m+1] = -1;
79
80     for (i=0; i<=m+1; ++i)
81         A[0][i] = A[n+1][i] = -1;
82
83     for (i=1; i<=n; ++i)
84         for (j=1; j<=m; ++j)
85             if (A[i][j] < h && M[i][j] == 0 && A[i][j] > 0)
86                 if (vecini(i,j)) fill(i,j);
87
88     while (!q.empty())
89     {
90         ic = q.front().first; jc = q.front().second;
91         if (M[ic][jc] == 0) fill(ic,jc);
92         q.pop();
93     }
94
95     printf("%d\n", sol);
96     return 0;
97 }

```

Listing 27.3.2: sunami1.cpp

```

1  /* 100 puncte
2   Implementare -Dan Pracsiu
3   Complexitate O(mxn)
4 */
5 #include<fstream>
6
7 #define InFile "tsunami.in"
8 #define OutFile "tsunami.out"
9
10 using namespace std;
11
12 struct coord
13 {
14     short x, y;
15 };
16
17 short a[1009][1009], b[1009][1009], L, C, h;
18 coord q[1000001];
19 int top, sol;
20
21 int main()
22 {
23     int i, j;
24
25     ifstream fin(InFile);
26     fin >> L >> C >> h;
27     for (i = 1; i <= L; i++)
28         for (j = 1; j <= C; j++)
29             fin >> a[i][j];
30
31     fin.close();
32
33     // bordare
34     for (i = 0; i <= C + 1; i++)
35         a[0][i] = a[L + 1][i] = 1001;
36     for (i = 0; i <= L + 1; i++)
37         a[i][0] = a[i][C + 1] = 1001;
38
39     //init fill
40     sol = 0;
41     top = -1;
42     for (i = 1; i <= C; i++)
43         if (a[1][i] == 0)
44         {
45             b[1][i] = 1;
46             top++;
47             q[top].x = 1;
48             q[top].y = i;
49         }
50     for (i = 1; i <= C; i++)
51         if (a[L][i] == 0)
52         {
53             b[L][i] = 1;
54             top++;
55             q[top].x = L;
56             q[top].y = i;
57         }
58     for (i = 1; i <= L; i++)
59         if (a[i][1] == 0)
60         {
61             b[i][1] = 1;
62             top++;
63             q[top].x = i;
64             q[top].y = 1;
65         }
66     for (i = 1; i <= L; i++)
67         if (a[i][C] == 0)
68         {
69             b[i][C] = 1;
70             top++;
71             q[top].x = i;
72             q[top].y = C;
73         }
74

```

```

75     // fill
76     while (top >= 0)
77     {
78         i = q[top].x;
79         j = q[top].y;
80         top--;
81         if (b[i - 1][j] != 1 && a[i - 1][j] < h)
82         {
83             b[i - 1][j] = 1;
84             top++;
85             q[top].x = i - 1;
86             q[top].y = j;
87             if (a[i - 1][j] > 0) sol++;
88         }
89         if (b[i + 1][j] != 1 && a[i + 1][j] < h)
90         {
91             b[i + 1][j] = 1;
92             top++;
93             q[top].x = i + 1;
94             q[top].y = j;
95             if (a[i + 1][j] > 0) sol++;
96         }
97         if (b[i][j - 1] != 1 && a[i][j - 1] < h)
98         {
99             b[i][j - 1] = 1;
100            top++;
101            q[top].x = i;
102            q[top].y = j - 1;
103            if (a[i][j - 1] > 0) sol++;
104        }
105        if (b[i][j + 1] != 1 && a[i][j + 1] < h)
106        {
107            b[i][j + 1] = 1;
108            top++;
109            q[top].x = i;
110            q[top].y = j + 1;
111            if (a[i][j + 1] > 0) sol++;
112        }
113    }
114
115    ofstream fout (OutFile);
116    fout << sol << "\n";
117    fout.close();
118
119    return 0;
120}

```

27.4 acces

Problema 4 - acces

100 de puncte

Considerăm o matrice cu L linii (numerotate de sus în jos de la 1 la L) și C coloane (numerotate de la stânga la dreapta de la 1 la C) care memorează doar valori 0 și 1. Mai mult, valorile egale cu 1 sunt grupate în mai multe dreptunghiuri pline, care nu se învecinează nici pe linii, nici pe coloane, nici pe diagonale. În exemplul din fig. 1 matricea este corectă deoarece cele 4 dreptunghiuri de 1 nu se învecinează. În schimb în fig. 2 există 2 dreptunghiuri de 1 învecinate pe coloană și două învecinate pe diagonală, deci matricea este incorectă.

$\begin{array}{ccccccc} 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} \end{array}$	$\begin{array}{ccccccc} 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\ 1 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \end{array}$
<i>fig. 1</i>	<i>fig. 2</i>

În această matrice se pot face deplasări doar pe direcțiile *Vest* și *Nord* în elemente egale cu 0, deci din poziția (i, j) se poate ajunge doar într-o din pozițiile $(i, j - 1)$ și $(i - 1, j)$, marcate cu 0. În acest fel, pornind de la o anumită poziție, prin deplasări succesive, pot fi accesate un anumit

număr de elemente ale matricei egale cu 0. De exemplu, în fig. 1, din poziția (2,4) pot fi accesate 5 componente egale cu 0, iar din poziția (5,4) pot fi accesate 14 componente egale cu 0.

Trebuie să răspundeți la Q întrebări, fiecare întrebare fiind de forma: "Câte din elementele egale cu zero ale matricei pot fi accesate din poziția (i, j) ?"

Cerințe

Scrieți un program care să determine, pentru fiecare întrebare, câte elemente egale cu 0 din matrice pot fi accesate din poziția precizată în cadrul întrebării.

Date de intrare

Pe prima linie a fișierului **acces.in** se află două numere naturale L și C separate printr-un spațiu, reprezentând numărul liniilor, respectiv numărul coloanelor matricei. Pe următoarele L linii se găsesc câte C cifre binare, separate prin câte un spațiu, reprezentând elementele matricei. Pe linia următoare se află numărul natural Q , reprezentând numărul întrebărilor. Pe următoarele Q linii se găsesc câte două numere naturale i și j , separate prin câte un spațiu, reprezentând poziția corespunzătoare unei întrebări.

Date de ieșire

Fișierul **acces.out** conține Q linii. Pe linia p ($1 \leq p \leq Q$) se află un număr natural k_p reprezentând răspunsul la cea de-a p -a întrebare.

Restricții și precizări

- $4 \leq L, C \leq 1000$
- $3 \leq Q \leq 500\,000$
- Pentru orice întrebare i, j se garantează că valoarea corespunzătoare din matrice este 0.
- pentru toate teste, dreptunghiurile formate din valori de 1 nu se încinează

Exemple:

acces.in	acces.out	Explicații
5 7	5	Pentru prima întrebare, cele 5 componente egale cu 0 care pot fi accesate sunt cele din pozițiile (1,1), (1, 2), (1,3), (1,4), (2,4)
0 0 0 0 1 1 1	14	
0 1 1 0 1 1 1	11	
0 1 1 0 0 0 0	3	
0 1 1 0 1 0 0		
0 0 0 0 1 0 1		
4		
2 4		
5 4		
4 7		
3 1		

Timp maxim de executare/test: **1.5** secunde

Memorie: total **16 MB**

Dimensiune maximă a sursei: **5 KB**

27.4.1 Indicații de rezolvare

Dan Pracsu

Soluția 1 (40 - 45 puncte).

Pentru fiecare interogare (x, y) să numărăm printr-un *algoritm de tip Fill* numărul componentelor accesibile din (x, y) . Astfel complexitatea ar fi $O(Q \times L \times C)$.

Soluția 2 (100 puncte).

Se construiește o matrice b cu L linii și C coloane în care $b[i, j]$ va memora numărul de componente egale cu 0 accesibile în matricea inițială a pornind din poziția (i, j) . Această matrice o

construim în $O(L \times C)$, urmând ca apoi la fiecare interogare să răspundem în $O(1)$. Complexitatea totală va fi deci $O(L \times C + Q)$.

Inițial construim prima linie și prima coloană din matricea b . Restul componentelor din b le obținem astfel:

- componentele din a marcate cu 1 care fac parte din eventualele dreptunghiuri de pe frontiera de *Nord* și *Vest* le marcăm cu 0 în b .

- Celelalte poziții din b care corespund unor poziții din a de valoare 1 vor avea câte o valoare negativă. Astfel, dacă (i, j) este o poziție care face parte dintr-un dreptunghi de 1 care are colțul din stânga-sus în (p, q) atunci în $b[i, j]$ se memorează valoarea din $a[p - 1, q - 1]$, dar cu minus.

- Pentru fiecare poziție (x, y) cu $a[x, y] = 0$ vom avea cazurile:

1. $a[x - 1, y] = 1$: atunci $b[x, y] = 1 + b[x, y - 1]$
2. $a[x, y - 1] = 1$: atunci $b[x, y] = 1 + b[x - 1, y]$
3. $a[x - 1, y] = 0$ și $a[x, y - 1] = 0$:

atunci $b[x, y] = 1 + b[x, y - 1] + b[x - 1, y] - b[x - 1, y - 1]$, dacă $b[x - 1, y - 1] > 0$
sau $b[x, y] = 1 + b[x, y - 1] + b[x - 1, y] + b[x - 1, y - 1]$, dacă $b[x - 1, y - 1] \leq 0$

Ca exemplu, considerăm primele 4 coloane din matricea a din exemplul enunțului:

```
0 0 0 0
0 1 1 0
0 1 1 0
0 1 1 0
0 0 0 0
```

în matricea b valorile vor fi:

```
1 2 3 4
2 -1 -1 5
3 -1 -1 6
4 -1 -1 7
5 6 7 14 (14 = 1 + 7 + 7 - 1, conform cazului 3)
```

Valoarea din dreapta-jos va fi completată conform cazului 3 de mai sus. Am reținut în b valori negative pentru a nu face confuzia cu cele pozitive care sunt corespunzătoare valorilor de 0 din a . Acea valoare este păstrată aşa deoarece în cazul poziției (5,4) apar două trasee distincte din matricea a care pleacă din (5, 4) și care se pot întâlni (în acest caz în poziția (1, 1)). De fapt se scade astfel numărul componentelor de 0 comune celor două trasee.

27.4.2 *Rezolvare detaliată

27.4.3 Cod sursă

Listing 27.4.1: acces.cpp

```

1 /*
2     implementare Dan Pracsiu
3     complexitate O( L x C + Q )
4 */
5 #include<cstdio>
6
7 #define Dim 1002
8 #define InFile "acces.in"
9 #define OutFile "acces.out"
10
11 using namespace std ;
12
13 int a[Dim][Dim], b[Dim][Dim], L, C ;
14
15 inline int Minim(int a, int b)
16 {
17     return a < b ? a : b;
18 }
19
20 void ConstructieB()
21 {
```

```

22     int i, j ;
23     // prima linie
24     for (i=1 ; i<=C ; i++)
25         if (a[1][i] == 1) b[1][i] = 0;
26         else b[1][i] = b[1][i-1] + 1;
27
28     // prima coloana:
29     for (i=1 ; i <= L ; i++)
30         if (a[i][1] == 1) b[i][1] = 0 ;
31         else b[i][1] = b[i-1][1] + 1 ;
32
33     // construiesc restul matricei
34     for (i=2 ; i <= L ; i++)
35         for (j=2 ; j<=C ; j++)
36             if (a[i][j] == 1) // perete
37             {
38                 if ( (b[i-1][j] == 0) || (b[i][j-1] == 0) )
39                     b[i][j] = 0 ;
40                 else if ( (b[i-1][j] > 0) && (b[i][j-1] > 0) )
41                     b[i][j] = -b[i-1][j-1] ;
42                 else b[i][j] = Minim(b[i-1][j], b[i][j-1]);
43             }
44             else // liber!
45             {
46                 if ( (b[i-1][j] > 0) && (b[i][j-1] > 0) )
47                 {
48                     b[i][j] = 1 + b[i-1][j] + b[i][j-1] ;
49                     if (b[i-1][j-1] < 0)
50                         b[i][j] = b[i][j] + b[i-1][j-1] ;
51                     else
52                         b[i][j] = b[i][j] - b[i-1][j-1] ;
53                 }
54                 else
55                     if (b[i-1][j] > 0) b[i][j] = 1 + b[i-1][j] ;
56                     else if (b[i][j-1] > 0) b[i][j] = 1 + b[i][j-1] ;
57                     else b[i][j] = 1 ;
58             }
59 }
60
61 int main()
62 {
63     int i, j, K, p, q;
64     freopen(InFile, "r", stdin);
65     freopen(OutFile, "w", stdout);
66
67     scanf("%d %d", &L, &C) ;
68     for (i=1 ; i<=L ; i++)
69         for (j=1 ; j<=C ; j++)
70             scanf("%d", &a[i][j]);
71     ConstructieB();
72
73     scanf("%d", &K);
74
75     for (i=1 ; i <= K ; i++)
76     {
77         scanf("%d %d", &p, &q) ;
78         printf("%d\n", b[p][q]);
79     }
80
81     return 0 ;
82 }
```

Listing 27.4.2: acces1.cpp

```

1 /*
2     implementare Dan Pracsiu (45 puncte)
3     Complexitate O ( L x C x Q )
4 */
5 #include<iostream>
6
7 #define dim 1002
8
9 #define InFile "acces.in"
10 #define OutFile "acces.out"
11
```

```

12  using namespace std ;
13
14  struct coord
15  {
16      short lin, col;
17  };
18
19  short a[dim][dim];
20  int b[dim][dim], L, C;
21  int nr;
22
23  void Bordare()
24  {
25      int i;
26      for (i = 0; i <= C + 1; i++)
27          a[0][i] = 1;
28      for (i = 0; i <= L + 1; i++)
29          a[i][0] = 1;
30  }
31
32  void Fill1(int x, int y)
33  {
34      coord q[dim], e;
35      int top;
36      top = 0;
37      q[0].lin = x;
38      q[0].col = y;
39      nr = 1;
40      while (top >= 0)
41      {
42          e = q[top--];
43          a[e.lin][e.col] = 2;
44          if (a[e.lin - 1][e.col] == 0)
45          {
46              nr++;
47              top++;
48              q[top].lin = e.lin - 1;
49              q[top].col = e.col;
50          }
51          if (a[e.lin][e.col - 1] == 0)
52          {
53              nr++;
54              top++;
55              q[top].lin = e.lin;
56              q[top].col = e.col - 1;
57          }
58      }
59  }
60
61  void Fill2(int x, int y)
62  {
63      coord q[dim], e;
64      int top;
65      top = 0;
66      q[0].lin = x;
67      q[0].col = y;
68      while (top >= 0)
69      {
70          e = q[top--];
71          a[e.lin][e.col] = 0;
72          if (a[e.lin - 1][e.col] == 2)
73          {
74              top++;
75              q[top].lin = e.lin - 1;
76              q[top].col = e.col;
77          }
78          if (a[e.lin][e.col - 1] == 2)
79          {
80              top++;
81              q[top].lin = e.lin;
82              q[top].col = e.col - 1;
83          }
84      }
85  }
86
87  int main()

```

```

88  {
89      int i, j, K, p, q;
90      ifstream fin(InFile);
91      fin >> L >> C ;
92      for (i=1 ; i<=L ; i++)
93          for (j=1 ; j<=C ; j++)
94              fin >> a[i][j] ;
95      Bordare();
96      fin >> K ;
97
98      ofstream fout(OutFile);
99      for (i=1 ; i <= K ; i++)
100     {
101         fin >> p >> q ;
102         nr = 0;
103         Fill1(p, q);
104         fout << nr << "\n" ;
105         Fill2(p, q);
106     }
107
108     fout.close();
109     fin.close();
110     return 0 ;
111 }
```

Listing 27.4.3: acces2.cpp

```

1 #include<iostream>
2
3 #define Dim 1002
4
5 using namespace std ;
6
7 int a[Dim][Dim], b[Dim][Dim], L, C ;
8 int nr;
9
10 void Bordare()
11 {
12     int i;
13     for (i = 0; i <= C + 1; i++)
14         a[0][i] = 1;
15     for (i = 0; i <= L + 1; i++)
16         a[i][0] = 1;
17 }
18
19 void Fill1(int x, int y)
20 {
21     if (a[x][y] == 0)
22     {
23         nr++;
24         a[x][y] = 2;
25         Fill1(x - 1, y);
26         Fill1(x, y - 1);
27     }
28 }
29
30 void Fill2(int x, int y)
31 {
32     if (a[x][y] == 2)
33     {
34         a[x][y] = 0;
35         Fill2(x - 1, y);
36         Fill2(x, y - 1);
37     }
38 }
39
40 int main()
41 {
42     int i, j, K, p, q;
43     ifstream fin("acces.in");
44     fin >> L >> C ;
45     for (i=1 ; i<=L ; i++)
46         for (j=1 ; j<=C ; j++)
47             fin >> a[i][j] ;
48     Bordare();
```

```

49     fin >> K ;
50
51     ofstream fout ("acces.out");
52     for (i=1 ; i <= K ; i++)
53     {
54         fin >> p >> q ;
55         nr = 0;
56         Fill1(p, q);
57         fout << nr << "\n" ;
58         Fill2(p, q);
59     }
60
61     fout.close();
62     fin.close();
63     return 0 ;
64 }
```

27.5 expresie

Problema 5 - expresie

100 de puncte

Se consideră o expresie, care poate să conțină:

- operanzi, care sunt litere mici ale alfabetului englez;
- paranteze rotunde;
- operatorul / care simbolizează împărțirea;
- operatorul * care simbolizează înmulțirea.

Regulile după care se evaluatează o astfel de expresie sunt cele din matematică.

Ne propunem să rescriem această expresie sub forma unui produs în care factorii pot să apară la puteri pozitive sau negative fără să mai folosim parantezele rotunde și în care folosim notația xy pentru $x * y$.

Astfel, a/b este echivalentă cu a^1b^{-1} , $a * (c/b) \Leftrightarrow a^1c^1b^{-1}$, $a/(b * c) * (a * b/c) \Leftrightarrow a^2c^{-2}$

Cerințe

Cunoscând expresia inițială să se determine expresia echivalentă scrisă sub formă de produs.

Date de intrare

Pe prima linie a fișierului **expresie.in** se află un sir de caractere ce reprezintă expresia dată.

Date de ieșire

Fiecare linie a fișierului **expresie.out** va conține un operand urmat de exact un spațiu și de un număr întreg ce reprezintă puterea la care acest operand apare în expresia scrisă sub formă de produs echivalentă. Operanzii vor apărea în fișier în ordine alfabetică, iar puterile pozitive sau nule nu vor fi precedate de semn.

Restricții și precizări

- Expresia dată are cel mult 20 000 de caractere
- Expresia dată este corectă și nu conține caractere spațiu
- 10% dintre teste vor conține doar operatorul *.

Exemple:

expresie.in	expresie.out	Explicații
a/b	a 1 b -1	
a/(b*c)*(a*b/c)	a 2 b 0 c -2	
(p/x)/((b/h/(x/x)))/(p/(b/(x/(h/(p)))))	b 0 h 2 p -1 x -2	

Timp maxim de executare/test: **0.5** secunde

Memorie: total **2 MB**

Dimensiune maximă a sursei: **5 KB**

27.5.1 Indicații de rezolvare

Stelian Ciurea

Varianta 1

Transformăm expresia în *forma poloneză* ($x/y \Rightarrow xy^*$, $x^*y \Rightarrow xy^*$)

Apoi analizăm expresia adusă la forma poloneză astfel:

dacă avem o construcție de genul

`expr1 / expr2`

atunci exponentii variabilelor din `expr2` își schimbă semnul.

Ca să determinăm ce se află în `expr2` parcurgem forma poloneză de la un caracter / spre stânga și aplicăm faptul că într-o expresie numărul de variabile este egal cu 1 + numărul de operatori.

Varianta 2

Citim expresia inițială într-un sir de caractere.

Determinăm prima pereche de paranteze care se închid, o eliminăm din sir, o transformăm rezultând variabilele care o compun și puterile lor sub formă unui sir de caractere, apoi inserăm acest sir în sirul inițial și reluăm procedura.

Obținem în final o expresie fără paranteze formată din variabile urmate de puterile la care apar.

Din aceasta deducem rezultatul cerut.

27.5.2 *Rezolvare detaliată

27.5.3 Cod sursă

Listing 27.5.1: expresie.cpp

```

1 // expresie - varainta 1           Âl' stelian ciurea
2 // 100 de puncte cu forma poloneza
3 #include <iostream>
4 #include <fstream>
5 #include <fstream>
6 #include <cstring>
7 #include <conio.h>
8
9 #define nmax 100001
10
11 using namespace std;
12
13 char expr[nmax], polo[nmax], stiva[nmax];
14 int putere[200], apare[200];
15 short int puteri[nmax];
16 int i, k, vf, j, ctlit, ctsemn, len, test;
17 char lit;
18 char buf[10];
19
20 int main()
21 {
22     ctlit = ctsemn = len = vf = i = j = k = 0;
23     char numein[20] = "expresie.in";
24     char numeout[20] = "expresie.out";
25
26     ifstream f(numein);
27     ofstream g(numeout);
28
29     f>>expr;
30     for (i=0;i<(int)strlen(expr);i++)
31         if (expr[i]>='a'&&expr[i]<='z')
32             apare[(int)expr[i]]=1;
33
34     j = 0;
35     vf = 0;
```

```

36     for (i=0;i<(int)strlen(expr);i++)
37     {
38         if (expr[i]>='a' && expr[i]<='z')
39         {
40             polo[j] = expr[i];
41             j++;
42         }
43         if (expr[i] == '(')
44         {
45             stiva[vf] = expr[i];
46             vf++;
47         }
48         if (expr[i] == ')')
49         {
50             while (stiva[vf-1] != '(') //scoate din stiva pana da de o (
51             {
52                 polo[j] = stiva[vf-1];
53                 vf--;
54                 j++;
55             }
56             vf--; //scoate si ( din stiva
57         }
58         if (expr[i] == '/')
59         {
60             while (vf>0 && stiva[vf-1]!='(')
61             {
62                 polo[j] = stiva[vf-1];
63                 vf--;
64                 j++;
65             }
66             stiva[vf] = expr[i];
67             vf++;
68         }
69         if (expr[i] == '*')
70         {
71             while (vf>0 && stiva[vf-1]!='(')
72             {
73                 polo[j] = stiva[vf-1];
74                 vf--;
75                 j++;
76             }
77             stiva[vf] = expr[i];
78             vf++;
79         }
80     }
81
82     while (vf>0)           //scoate din stiva in poloneza pana goleste stiva
83     {
84         polo[j] = stiva[vf-1];
85         vf--;
86         j++;
87     }
88
89     for (lit = 'a'; lit <= 'z'; lit++)
90         putere[(int)lit] = 1;
91
92     len = strlen(polo);
93     for (i=0; i<len; i++)
94     {
95         if (polo[i] == '/')
96         {
97             //determin grupul B:
98             ctlit=ctsemn=0;
99             for (j=i-1;j>=0;j--)
100             {
101                 if (polo[j]== '/')
102                     ctsemn++;
103                 else
104                     ctlit++;
105                 if (ctlit == 1 + ctsemn)
106                     break;
107             }
108             for (k=j;k<i;k++)
109                 if (polo[k] != '/')
110                     putere[(int)polo[k]] = - putere[(int)polo[k]];
111     }

```

```

112     }
113
114     for (lit = 'a'; lit <= 'z'; lit++)
115         if (apare[(int)lit] == 1)
116             if (putere[(int)lit] == 1)
117             {
118                 //cout << lit;
119                 //g << lit;
120             }
121     cout << endl;
122     for (lit = 'a'; lit <= 'z'; lit++)
123         if (apare[(int)lit] == 1)
124             if (putere[(int)lit] == -1)
125             {
126                 //cout << lit;
127                 //g << lit;
128             }
129     cout << endl << endl;
130
131     for (i=0;i<len;i++)
132         puteri[i]=1;
133
134     for (i=0; i<len; i++)
135         if (polo[i] == '/')
136         {
137             //determin grupul B:
138             ctlim=ctsemn=0;
139             for (j=i-1;j>=0;j--)
140             {
141                 if (polo[j]=='/' || polo[j]=='*')
142                     ctsemn++;
143                 else
144                     ctlim++;
145                 if (ctlim == 1 + ctsemn)
146                     break;
147             }
148             for (k=j;k<i;k++)
149                 if (polo[k] >= 'a' && polo[k] <= 'z')
150                     puteri[k] = - puteri[k];
151
152         }
153
154     for (lit='a';lit<='z';lit++)
155         putere[(int)lit] = 0;
156
157     for (i=0;i<len;i++)
158         if (polo[i]>='a' && polo[i]<='z')
159             putere[(int)polo[i]] += puteri[i];
160
161     for (lit = 'a'; lit <= 'z'; lit++)
162         if (apare[(int)lit] == 1)
163         {
164             //cout << lit << ' ' << putere[(int)lit] << endl;
165             g << lit << ' ' << putere[(int)lit] << endl;
166             //g << lit;
167         }
168
169     cout << endl;
170     f.close();
171     g.close();
172
173     return 0;
174 }
```

Listing 27.5.2: expresie1.cpp

```

1 /*
2  Sursa: Eugen Nodea
3  Rez. forma poloneza -
4 */
5 #include <fstream>
6 #include <iostream>
7 #include <cstring>
8
9 using namespace std;
```

```

10
11 char expr[20001],polo[20001],stiva[20001];
12 int putere[300];
13 bool apare[300];
14 int i, vf, j, semn, poz;
15 char lit;
16 string st[20001];
17
18 ifstream f("expresie.in");
19 ofstream g("expresie.out");
20
21 int main()
22 {
23     f>>expr;
24
25     for (i=0; i<(int)strlen(expr); i++)
26         if (expr[i]>='a' && expr[i]<='z')
27             apare[(int)expr[i]]=1;
28
29     j = 0;
30     vf = 0;
31     for (i=0; i<(int)strlen(expr); i++)
32     {
33         if (expr[i]>='a' && expr[i]<='z')
34         {
35             polo[j] = expr[i];
36             j++;
37         }
38         if (expr[i] == '(')
39         {
40             stiva[vf] = expr[i];
41             vf++;
42         }
43         if (expr[i] == ')')
44         {
45             while (stiva[vf-1] != '(') //scoate din stiva pana de la (
46             {
47                 polo[j] = stiva[vf-1];
48                 vf--;
49                 j++;
50             }
51             vf--; //scoate si ( din stiva
52         }
53         if (expr[i] == '/' || expr[i] == '*')
54         {
55             while (vf>0 && stiva[vf-1]!='(')
56             {
57                 polo[j] = stiva[vf-1];
58                 vf--;
59                 j++;
60             }
61             stiva[vf] = expr[i];
62             vf++;
63         }
64     }
65
66     while (vf>0)          //scoate din stiva in poloneza pana goneste stiva
67     {
68         polo[j] = stiva[vf-1];
69         vf--;
70         j++;
71     }
72
73 //g<<polo << '\n';
74
75 vf = 0;
76 for (i=0; i<(int)strlen(polo); i++)
77     if (polo[i] == '*')
78     {
79         string x="",y="";
80         poz=st[vf].find('/');
81         if (poz!=-1)
82         {
83             x=st[vf].substr(0,st[vf].find('/'));
84             y=st[vf].substr(st[vf].find('/')+1,st[vf].size());
85         }

```

```

86         else
87             x=st[vf].substr(0,st[vf].size());
88
89             vf--;
90             poz=st[vf].find('/');
91             if (poz!=-1)
92             {
93                 x+=st[vf].substr(0,st[vf].find('/'));
94                 y+=st[vf].substr(st[vf].find('/')+1,st[vf].size());
95             }
96             else
97                 x+=st[vf].substr(0,st[vf].size());
98             st[vf]= x;
99             if (y!="")
100                 st[vf]+= '/' + y;
101         }
102         else if (polo[i] == '/')
103         {
104             string x="",y="";
105             poz=st[vf].find('/');
106             if (poz!=-1)
107             {
108                 y=st[vf].substr(0,st[vf].find('/'));
109                 x=st[vf].substr(st[vf].find('/')+1,st[vf].size());
110             }
111             else
112                 y=st[vf].substr(0,st[vf].size());
113
114             vf--;
115             poz=st[vf].find('/');
116             if (poz!=-1)
117             {
118                 x+=st[vf].substr(0,st[vf].find('/'));
119                 y+=st[vf].substr(st[vf].find('/')+1,st[vf].size());
120             }
121             else
122                 x+=st[vf].substr(0,st[vf].size());
123             st[vf]= x + '/' + y;
124         }
125         else
126             st[++vf]=polo[i];
127
128     semn=1;
129     for (j=0; j<st[1].size(); j++)
130         if (st[1][j]== '/') semn=-1;
131         else if (semn==1) putere[(int)st[1][j]]++;
132         else putere[(int)st[1][j]]--;
133
134     for (lit = 'a'; lit <= 'z'; lit++)
135         if (apare[(int)lit])
136             g << lit << ' ' << putere[int(lit)] << '\n';
137
138     return 0;
139 }
```

Listing 27.5.3: expresie2.cpp

```

1 // expresie - varainta 2           Al' stelian ciurea
2 // 100 cu prelucrari succesive de siruri aflate intre paranteze
3 #include <iostream>
4 #include <fstream>
5 #include <string>
6
7 #define nmax 100001
8
9 using namespace std;
10
11 string s,s1;
12 int i, poz;
13 char lit;
14
15 ifstream f("expresie.in");
16 ofstream g("expresie.out");
17 int apare[150];
18
```

```

19 string initial(string s)
20 {
21     string s1;
22     int i, len = s.size();
23     for (i=0;i<len;i++)
24         if (s[i]>='a' && s[i]<='z')
25         {
26             s1 = s1 + s[i] + "+1";
27             apare[s[i]] = 0;
28         }
29     else
30         s1 = s1+s[i];
31     return s1;
32 }
33
34 string transform(string s) // transofrma o expresie fara paranteze
35 // cu / si * intr-una de tip produs
36 {
37     short int ct[150];
38     int i,j,len,semn=1,numar=0,semnnum;
39     string s1;
40     char lit,buf[10];
41
42     for (i='a';i<='z';i++)
43         ct[i]=0;
44
45     len = s.size();
46     for(i=0;i<len;i++)
47     {
48         if (s[i]=='-')
49             semnnum=-1;
50         if (s[i]=='+')
51             semnnum+=1;
52         if (s[i]=='/')
53         {
54             ct[lit] += semn*semnnum*numar;
55             semn = -1;
56             numar=0;
57         }
58         if (s[i]=='*')
59         {
60             ct[lit] += semn*semnnum*numar;
61             semn = +1;
62             numar=0;
63         }
64         if (s[i]<='z' && s[i] >='a')
65         {
66             ct[lit] += semn*semnnum*numar;
67             numar=0;
68             lit = s[i];
69         }
70         if (s[i]>='0'&&s[i]<='9')
71             numar = 10*numar + (s[i]-48);
72     }
73
74     ct[lit] += semn*semnnum*numar;
75     for (lit='a';lit<='z';lit++)
76         if (ct[lit]!=0)
77         {
78             s1 = s1 + lit;
79             ltoa(ct[lit],buf,10);
80             if (ct[lit] >= 0)
81                 s1 = s1 + '+' + buf;
82             else
83                 s1=s1+buf;
84         }
85
86     return s1;
87 }
88
89 int main()
90 {
91
92     for (i=0;i<150;i++)
93         apare[i] = -1000000000;
94     //s = "a*(b/(c/d))";

```

```

95     f >> s;
96     s = initial(s);
97     //cout << s << endl;
98     while ((poz = s.find(')')) >=0)
99     {
100         for(i=poz-1;i>=0;i--)
101             if (s[i]=='(')
102                 break;
103         s1 = s.substr(i+1,poz-1-(i+1)+1);
104         //cout << s1 << endl;
105         s1 = transform(s1);
106         //cout << s1 << endl;
107         s.erase(i,poz-i+1);
108         //cout << s << endl;
109         s.insert(i,s1);
110         //cout << s << endl;
111         //system("PAUSE");
112         //cout << endl;
113     }
114
115     s = transform(s);
116
117     //cout << s << endl;
118
119     int len = s.size();
120     int numar = 0;
121     int semn;
122     for (i=0;i<len;i++)
123     {
124         if (s[i]>='a' && s[i] <='z')
125         {
126             apare[lit] = semn*numar;
127             numar=0;
128             lit = s[i];
129         }
130         if (s[i]=='+')
131             semn = +1;
132         if (s[i]=='-')
133             semn=-1;
134         if (s[i]>='0' && s[i] <='9')
135             numar = 10*numar + (s[i]-48);
136     }
137     apare[lit] = semn*numar;
138
139     for (lit='a';lit<='z';lit++)
140         if (apare[lit]>-1000000000)
141         {
142             cout << lit << ' ' << apare[lit] << endl;
143             g << lit << ' ' << apare[lit] << endl;
144         }
145     return 0;
146 }
```

27.6 telecab

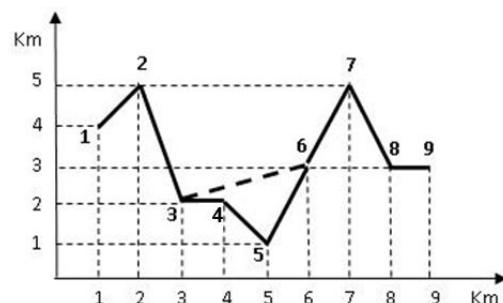
Problema 6 - telecab

100 de puncte

Bobi este un excursionist pasionat. Zona muntoasă pe care o va vizita în această vară are o caracteristică atractivă: există un sistem de transport cu telecabina. Proiecția pe un plan orizontal a traseului ales de Bobi este o linie dreaptă. Anumite puncte situate pe munte, dintre care unele se află pe traseul telecabinei, se numesc cote. Proiecțiile cotelor pe orizontală sunt n puncte coliniare aflate unul față de altul la distanța de un kilometru.

In figură, cu linie plină este reprezentat profilul muntelui, iar cu linie punctată îngroșată traseul telecabinei, acolo unde acesta nu coincide cu profilul muntelui. Telecabina parurge segmentele care unesc cotele: [1,2], [2,3], [3,6], [6,7], [7,8] și [8,9].

Fie h_1, h_2, \dots, h_n înălțimile cotelor. Viteza normală cu care se deplasează telecabina este de $v = 1$ Km / oră. Traseul telecabinei este formată din



segmente de dreaptă și urmează în general profilul muntelui, trecând prin fiecare cotă. Abaterea traseului telecabinei de la profilul muntelui are loc în situația în care un cablu de telecabină poate fi întins direct între o cotă i și prima cotă j , aflată în direcția de deplasare, care se află la o înălțime mai mare decât cota i .

Bobi dispune de suma de S euro. Pentru fiecare segment de drum parcurs între două cote i și j , el trebuie să plătească suma $h_j - h_i$ euro dacă este vorba de o porțiune de urcare în pantă și nu trebuie să plătească nimic dacă este vorba despre o porțiune orizontală.

La coborârea unei pante situată între două cote i și $i+1$ Bobi are două variante: prima variantă este de a coborî cu viteza normală $v = 1 \text{ Km/oră}$ și atunci nu plătește nimic. A doua variantă, pe care băiatul o poate alege prin apăsarea unui buton în telecabină, este de a coborî pantă, indiferent de lungimea ei, în timpul de o oră, deci cu o viteză diferită de cea normală, dar în acest caz Bobi trebuie să plătească suma $h_i - h_{i+1}$ euro.

Cerințe

Cunoscând înălțimile celor n cote prin care va trece telecabina și suma de care dispune Bobi, scrieți un program care determină:

1. lungimea totală a traseului telecabinei măsurat între cota 1 și cota n .

2. timpul minim exprimat în ore de care are nevoie Bobi ca să ajungă la o cotă de pe drum cu numărul de ordine mai mare sau egal cu K dat, știind că pornește de la cota 1 și că există cel puțin o variantă care conduce la acest timp minim și care necesită o sumă mai mică sau egală cu S .

Date de intrare

Fișierul de intrare **telecab.in** conține pe prima linie trei numere naturale $n \ K \ S$ separate prin câte un spațiu.

Pe fiecare dintre următoarele n linii se găsește câte un număr natural. Pe linia $i + 1$ se găsește numărul h_i , exprimat în kilometri, reprezentând înălțimea cotei i ($i = 1, 2, \dots, n$).

Date de ieșire

Fișierul de ieșire **telecab.out** va conține pe prima linie un număr întreg L , reprezentând lungimea totală a traseului telecabinei, între cotele 1 și n , exprimat în kilometri. Pe linia a doua se va scrie numărul natural T_{min} , reprezentând timpul minim de care are nevoie Bobi ca să atingă o cotă având numărul de ordine mai mare sau egal cu K .

Restricții și precizări

- $3 \leq n \leq 100\,000$
- $1 \leq h_1, h_2, \dots, h_n \leq 10$
- $1 \leq K, S \leq 1000$
- distanța dintre două cote succesive de pe traseu se calculează ca fiind partea întreagă a distanței euclidiene în plan dintre cele două cote.
 - între două cote consecutive, profilul muntelui este un segment de dreaptă care unește cotele.
 - pentru toate cazuurile de test se garantează că Bobi are suficienți bani pentru a atinge sau a depăși cota K .
 - pentru mișcarea rectilinie cu viteza constantă, **distanța = viteza × timpul**.
 - pentru rezolvarea primei cerințe se obține 20% din punctajul fiecărui test

Exemple:

telecab.in	telecab.out	Explicații
-------------------	--------------------	-------------------

9 4 5 2 2 1 3 5 3 3	12 9	<p>Exemplul este cel din figură.</p> <p>Lungimea traseului telecabinei este: $1 + 3 + 3 + 2 + 2 + 1 = 12$</p> <p>Timpul minim de deplasare până la cota 8 este: $1 + 1 + 3 + 2 + 2 = 9$</p> <p>Segmentul [1,2] se parcurge în 1 ore și se cheltuie 1 euro.</p> <p>Segmentul [2,3] se parcurge în 1 ore și se cheltuie 3 euro.</p> <p>Segmentul [3,6] se parcurge în 3 ore și se cheltuie 1 euro.</p> <p>(distanța de la cota 3 la cota 6 este: $\sqrt{(6-3)^2 + (3-2)^2} = 3$, iar timpul este $3 / 1 = 3$).</p> <p>Segmentul [6,7] se parcurge în 2 ore și se cheltuie 2 euro.</p> <p>Segmentul [7,8] se parcurge în 2 ore și se cheltuie 0 euro.</p>
5 3 2 1 2 2 3 1	5 3	<p>Lungimea traseului telecabinei este: $1 + 2 + 2 = 5$</p> <p>Timpul minim de deplasare până la cota 4 este: $1 + 2 = 3$</p> <p>Segmentul [1,2] se parcurge în 1 ore și se cheltuie 1 euro.</p> <p>Segmentul [2,4] se parcurge în 2 ore și se cheltuie 1 euro.</p> <p>Se observă că telecabina atinge cotele 2 și 4, trecând pe deasupra cotei 3.</p>

Timp maxim de executare/test: **0.2** secunde

Memorie: total **6 MB**

Dimensiune maximă a sursei: **5 KB**

27.6.1 Indicații de rezolvare

Constantin Gălățan

Soluția 1. 100 puncte (Constantin Gălățan)

Cerința 1).

Traseul telecabinei este unic determinat. Fie $h[1], h[2], \dots, h[n]$ sirul înălțimilor cotelor și $p[1], p[2], \dots, p[n]$ un sir având semnificația: $p[i]$ = poziția primei cote mai înaltă decât cota i .

Exemplu:

pentru $h = 4, 5, 2, 2, 1, 3, 5, 3, 3, p = 2, 0, 6, 6, 6, 7, 8, 0, 0$

Se observă că dacă $p[i] = j$, atunci, $p[i+1] = j, p[i+2] = j, \dots, p[j-1] = j$. Telecabina unește în mod direct cotele i și j .

Pentru o soluție liniară, se menține o *stivă* în care se introduc pe rând toate cotele. Dacă i e cota din vârful stivei și j este cota curentă, atunci înainte de a introduce j în stivă, cât timp $h[j] > h[i]$, se marchează $p[i] = j$ și se scoate i din stivă. Complexitate: $O(n)$.

Cerința 2)

Pentru o soluție de tip *greedy* se acordă 70% din punctajul acordat pentru această cerință.

Soluția oficială, care primește punctajul maxim, folosește *recursie cu memorizare* după cum urmează: definim o funcție recursivă $Time(i, s)$ - timpul minim necesar să se atingă o cotă cu numărul de ordine cel puțin k (este posibil ca telecabina să treacă pe deasupra cotei k), pornind de la cota i cu suma s .

Funcția $Time(i, s)$ se definește recurrent astfel:

- 0 dacă $i \geq k$
- $1 + Time(i+1, s)$ - segment orizontal
- $dist(i, j) + Time(j, s - (h[j] - h[i]))$ - urcare pe segmentul [i j]
- $\min(1 + Time(i+1, s - (h[j] - h[i])), dist(i, i+1) + Time(i+1, s))$ - coborâre pe segmentul [i j]

Întrucât există două opțiuni de coborâre, pentru a se evita recalcularea timpului minim pentru stările cu aceeași sumă s și cotă i , se rețin timpii într-un tablou $tmin[i][s]$ (*memorizare*).

Complexitatea soluției: $O(n + K * S)$.

Soluția 2 - Marius Dumitran

Subpunctul 2.

Rezolvare cu *programare dinamică*:

Vom folosi o matrice $M[i][j]$ unde $M[i][j]$ este cea mai scurta distanta cu care putem ajunge la cota i , folosind j bani. $M[i][j] = INF$ daca nu putem ajunge in punctul i folosind j bani

De asemenea ne vom ajuta si de vectorii *Next*, unde $Next[i]$ este cota urmatoare lui i in traseul telecabinei, si $H[i]$ unde $H[i]$ este inaltimea punctului i .

Initializare : $M[1][0] = 0$.

Rezolvare :

```
for( int i = 1; i <= K; ++i)
for( int j = 0; j <= S; ++j)
{
// urcare
if( h[ i ] < h[ next[ i ] ] )
if( M[ Next[ i ]][ j + h[ j ] - h[ i ] ] > M[ i ][ j ] + dist( i, next[ i ] )
    M[ Next[ i ]][ j + h[ j ] - h[ i ] ] = M[ i ][ j ] + dist( i, next[ i ] );
if( h[ i ] > h[ next[ i ] ] )
{
// coborare
if( M[ i + 1 ][ j ] > M[ i ][ j ] + dist( i , next[ i ] ) // fara bani
    M[ i + 1 ][ j ] = M[ i ][ j ] + dist( i , next[ i ] )
if( M[ i + 1 ][ j + h[ i ] - h[ j ] ] > M[ i ][ j ] + 1 )
    M[ i + 1 ][ j + h[ i ] - h[ j ] ] = M[ i ][ j ] + 1;
}
```

Complexitate $O(K * S)$.

Soluția 3 (prof. Stelian Ciurea)

Cerinta 1:

I - construiesc la citire 10 vectori: d_1, d_2, \dots, d_{10} ; în d_i rețin pozițiile unde apar cote de înălțime i . Din construcție acești vectori vor fi sortați.

II - trebuie să determin cotele care formează traseul telecabinei.

Pentru aceasta parcurg cotele și pentru cota din poziția i având înălțimea $h[i]$, determin prin căutare în vectorii $dh[i] + 1, dh[i] + 2, d_{10}$ cea mai mică valoare mai mare decât i - aceasta va reprezenta poziția primei cote mai mari decât cea din poziția i .

Din enunț, traseul telecabinei va fi de la cota i la cota determinată astfel.

Vectorii fiind sortați, căutarea o voi face binar.

Rețin astfel pozițiile cotelor pe unde va trece traseul telecabinei.

Cunoscând traseul telecabinei determinarea sumei distanțelor este imediată.

Complexitate: $O(n * 10 * \log(n))$

Cerinta 2:

I - deoarece pentru porțiunile în care telecabina urcă sau merge pe orizontală nu am de ales, determin pentru acestea durata și suma necesară.

II - rețin în doi vectori distanțele și diferențele de înălțime pentru porțiunile unde telecabina coboară.

III - sortează descrescător cei doi vectori după valorile din vectorul de distanțe.

Ca să minimizez timpul de coborâre, prefer să-mi cheltuiesc suma rămasă pentru a parurge cât mai repede porțiunile cele mai lungi!

Eventuala sumă ramasa (în cazul în care nu îmi ajung banii pentru ultima distanță astfel determinată), o cheltuiesc pentru a cobori rapid porțiuni cu diferențe de înălțime mici (deoarece înălțimile sunt dintr-un interval foarte mic, probabilitatea să găseasc diferențe de înălțime egale cu 1 pe care să le pot parurge cu suma rămasă este foarte mare).

Complexitatea este cea a sortării: $O(n * \log(n))$

27.6.2 *Rezolvare detaliată

27.6.3 Cod sursă

Listing 27.6.1: telecab.cpp

```

1  /* 100 puncte
2   * Recursie cu memoizare
3   * Complexitate O(n + K*S)
4   * Constantin Galatan
5   */
6 #include <fstream>
7 #include <cmath>
8 #include <cstring>
9 #include <algorithm>
10 #include <iostream>
11
12 using namespace std;
13
14 #define INF 0x3f3f3f3f
15
16 ifstream fin("telecab.in");
17 ofstream fout("telecab.out");
18
19 const int MAXN = 100005;
20
21 int n, h[MAXN], poz[MAXN], st[MAXN];
22 int S, K; // Suma si cota care trebuie atinsa sau depasita
23 int k;
24 int tmin[1001][1001];
25 int cnt;
26 int Dist(int i, int j);
27 int Timp(int i, int s);
28
29 int main()
30 {
31     fin >> n >> K >> S;
32     memset(poz, -1, sizeof(poz));
33     for (int i = 0; i < n; i++)
34     {
35         fin >> h[i];
36         while (k > 0 && h[i] > h[st[k - 1]])
37             poz[st[--k]] = i;
38
39         st[k++] = i;
40     }
41     long long d = 0;
42     for (int i = 0; i < n - 1; )
43     {
44         if( poz[i] > 0 )// exista o cota mai inalta, atunci - legatura directa
45         {
46             d += Dist(poz[i], i);
47             i = poz[i];
48         }
49         else
50         {
51             if ( h[i] > h[i+1] ) // cobor
52                 d += Dist(i, i + 1);
53             else
54                 d++; // orizontala
55             i++;
56         }
57     }
58
59     for( int i = 0; i <= 1000; ++i)
60         for( int j = 0 ; j <= 1000; ++j)
61             tmin[i][j] = INF;
62     fout << d << '\n';
63     fout << Timp(0, S) << '\n';
64
65     fin.close();
66     fout.close();
67
68     return 0;
69 }
70

```

```

71 int Dist(int i, int j) // distanta de la varful i la j
72 {
73     int a = h[i] - h[j], b = j - i;
74     return (int)sqrt(double(a * a + b * b));
75 }
76
77 // Timpul minim necesar ca sa atinga CEL PUTIN cota K, stind ca are
78 // suma initiala S
79
80 int Timp(int i, int s) // timpul minim necesar sa ajung la final incepand
81 // de la poz i cu suma s
82 {
83     if ( s < 0 ) return -1;
84
85     int& ret = tmin[i][s];
86     if ( i >= K - 1 ) return ret = 0;
87     if ( ret != INF ) return ret; // timpul a fost calculat anterior
88
89     if ( poz[i] > 0 ) // am in dreapta o cota mai inalta
90     {
91         int r = Timp(poz[i], s - (h[poz[i]] - h[i]));
92         if ( r == -1 ) return ret = -1;
93         return ret = Dist(i, poz[i]) + r;
94     }
95
96     // cobor
97     if ( h[i+1] < h[i] )
98     {
99         int t1 = Timp(i + 1, s - (h[i] - h[i+1]));
100        t1 = (t1 == -1) ? INF : 1 + t1; // cobor o ora dar platesc
101
102        int t2 = Timp(i + 1, s); // cobor cu viteza v, costa timp, dar
103        // nu platesc
104        t2 = (t2 == -1) ? INF : Dist(i, i + 1) + t2;
105        ret = min(t1, t2);
106        if ( ret == INF )
107            ret = -1;
108        return ret;
109    }
110
111    // inaltimei egale
112    int t = Timp(i + 1, s);
113    if ( t == -1 ) return ret = -1;
114    return ret = 1 + t;
115 }
```

Listing 27.6.2: telecab1.cpp

```

1 // 100 puncte
2 // dinamica O(n + K*S)
3 // Marius Dumitran
4 #include <iostream>
5 #include <stdio.h>
6 #include <time.h>
7 #include <math.h>
8
9 time_t start;
10
11 using namespace std;
12
13 #define maxn 100001
14 #define max2 1024
15
16 int mat[ max2 ][ max2 ];
17
18 int calc_dist( int x1, int y1, int x2, int y2 )
19 {
20     return (int)sqrt((double) (x1 - x2)*(x1 - x2) + (y1 - y2)*(y1 - y2));
21 }
22
23 int main()
24 {
25     //start = clock();
26     freopen("telecab.in","r",stdin);
27     freopen("telecab.out","w",stdout);
```

```

28
29     int n, k, s;
30     int h[ maxn ];
31     int next[ maxn ];
32     int stiv[ maxn ];
33     int last = 0;
34     scanf("%d %d %d", &n, &k, &s);
35     for( int i = 1; i <= n; ++i)
36         scanf("%d", &h[i]);
37
38     for( int i = 1; i <= n; ++i)
39     {
40         next[ i ] = i + 1;
41         while( last >= 1)
42         {
43             if( h[ i ] > h[ stiv[ last ]])
44             {
45                 next[ stiv[ last ] ] = i;
46                 last--;
47             }
48             else break;
49         }
50         stiv[ ++last ] = i;
51     }
52
53     int act = 1, dist = 0;
54     while(act < n )
55     {
56         dist += calc_dist( act, h[act], next[ act ], h[ next[ act ]]);
57         act = next[ act ];
58     }
59 }
60 printf("%d\n", dist);
61
62 //printf("%lf\n", (double)(clock()-start) / CLOCKS_PER_SEC);
63 mat[ 1 ][ 0 ] = 1;
64 act = 1;
65 while( act < k)
66 {
67     int cost = 0, caz = 0,
68         dist = calc_dist( act, h[act], next[ act ], h[next[ act ]]);
69     if( h[ next[ act ]] >= h[ act ])
70     {
71         cost = h[ next[ act ]] - h [ act ];
72     }
73     else
74     {
75         caz = 1;
76         cost = h[act] - h[ act + 1];
77     }
78
79     for( int j = 0; j <= s; ++j)
80     {
81         if( mat[ act ][ j ] == 0) continue;
82         if( caz == 0 )
83         {
84             if( j + cost > s) continue;
85             if( mat[ next[ act ]][ j + cost ] == 0 ||
86                 mat[ next[ act ]][ j + cost] > mat[ act ][ j ] + dist)
87                 mat[ next[ act ]][ j + cost ] = mat[ act ][ j ] + dist;
88
89         }
90         else
91         {
92             if( mat[ next[ act ]][ j ] == 0 ||
93                 mat[ next[ act ]][ j ] > mat[ act ][ j ] + dist)
94                 mat[ next[ act ]][ j ] = mat[ act ][ j ] + dist;
95
96             if( mat[ next[ act ]][ j + cost ] == 0 ||
97                 mat[ next[ act ]][ j + cost] > mat[ act ][ j ] + 1)
98                 mat[ next[ act ]][ j + cost ] = mat[ act ][ j ] + 1;
99         }
100    }
101
102    act = next[ act];
103 }
```

```
104     int sol = 2000000000;
105     for( int i = 0; i <= s; ++i)
106         if( mat[ act ][ i ] < sol && mat[ act ][ i ])
107             sol = mat[ act ][ i ];
108
109     printf("%d\n", sol - 1);
110
111     return 0;
112 }


---


```

Capitolul 28

ONI 2010

28.1 dreptunghiuri

Problema 1 - dreptunghiuri

100 de puncte

Se consideră o matrice cu elemente 0 sau 1, cu L linii (numerotate de la 1 la L) și C coloane (numerotate de la 1 la C).

Definim o zonă dreptunghiulară ca fiind o submatrice ce are pe contur numai valori 1 și cu proprietatea că nu există valori de 1 nesituate pe contur și în același timp la distanța 1 față de un punct de pe contur. Două puncte sunt la distanța 1 dacă și numai dacă sunt vecine pe una dintre cele 8 direcții.

Interiorul unei zone dreptunghiulare constă din elementele din submatrice nesituate pe contur.

O zonă dreptunghiulară poate fi inclusă complet în interiorul alteia. Definim ordinul unei zone dreptunghiulare ca fiind valoarea $d+1$, unde d este numărul de zone în interiorul cărora aceasta este inclusă.

Orice element 1 din matrice se află pe conturul unei singure zone dreptunghiulare.

Fig. 1-4 conțin exemple de zone dreptunghiulare. În fig. 5 este o matrice în care se găsesc trei zone dreptunghiulare, dintre care zonele din interior au ordinul 2 iar cealaltă ordinul 1.

1 1 1 1 1 1 0 0 0 1 1 1 1 1 1	1 1 1	1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 1 0 1 0 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1 1 0 1 0 1 0 1 0 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 0
fig. 1	fig. 2	fig. 3	fig. 4	fig. 5

Cerințe

Să se determine numărul total de zone dreptunghiulare din matrice, ordinul maxim al unei zone și numărul de zone care au acest ordin maxim.

Date de intrare

Fișierul de intrare **dreptunghiuri.in** conține pe prima linie numerele naturale L și C separate printr-un spațiu. Pe fiecare din următoarele L linii din fișier se află câte C numere din mulțimea $\{0,1\}$, separate prin câte un spațiu, reprezentând valorile din matrice.

Date de ieșire

Fișierul de ieșire **dreptunghiuri.out** conține pe prima linie trei numere naturale D , O și NR , separate prin câte un spațiu, unde D este numărul total de zone dreptunghiulare din matrice, O este ordinul maxim al unui astfel de zone, iar NR este numărul de zone de ordin maxim.

Restricții și precizări

- $3 \leq L, C \leq 1\,000$
- Datele de intrare sunt corecte. Va exista cel puțin o zonă dreptunghiulară în matrice.
- Pentru determinarea corectă a numărului de zone se acordă 20% din punctajul pe fiecare test.

Exemplu:

dreptunghiuri.in	dreptunghiuri.out	Explicații
<pre>9 12 0 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 1 0 1 0 1 1 1 1 0 0 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 1 0 1 1 1 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 1 1</pre>	4 3 1	Sunt în total 4 zone dreptunghiulare, ordinul maxim al uneia dintre ele este 3 (cea formată dintr-un singur 1) și este o singură astfel de zonă.

Timp maxim de executare/test: **0.8** secunde pe Windows, **0.6** secunde pe Linux

Memorie: total **16 MB** din care pentru stivă **1 MB**

28.1.1 Indicații de rezolvare

prof. Nicoli Marius, CN "Frații Buzesti" - Craiova

Asociem un dreptunghi cu o pereche de paranteze. Dreptunghiurile direct incluse în el reprezintă astfel alte perechi de paranteze incluse în perechea inițială. Problema se reduce la a determina "adâncimea" maximă a unei perechi de paranteze într-o secvență corect parantezată. Pentru aceasta se poate utiliza o *stivă*.

La implementare nu e necesară explicit transformarea problemei într-un sir de paranteze. Se poate utiliza un algoritm recursiv care tratează un dreptunghi și memorează nivelul său *niv*, autoapelul făcându-se pentru dreptunghiurile incluse în el cu *niv* + 1. Se calculează maximul valorilor *niv* și numărul lor. Complexitatea este $O(L \times C)$.

Pentru 20 de puncte era suficientă o singură parcurgere a matricei, iar la întâlnirea unei valori 1 se marchează cele sunt în același dreptunghi cu ea.

Soluția 2 (Dan Pracsiu)

Se utilizează un *algoritm de tip Fill recursiv* (pentru 70 de puncte pe teste date datorită limitării *segmentului de stivă* la 1 MB) sau *nerecursiv* folosind o *coadă* (pentru 100 puncte).

Inițial valorile egale cu 1 din matrice se fac -1.

Se *bordează* apoi matricea cu valoarea 0 și se apelează un *Fill* din poziția (0, 0) marcând astfel exteriorul dreptunghiurilor cu valoarea 1.

Se parcurge apoi matricea și când este găsită o poziție (x, y) care memorează 0 sau -1, se aplică un $\text{Fill}(x, y)$ care face marcarea cu valoarea de la poziția $(x - 1, y - 1)$, plus 1. Ideea este că astfel conturul unui dreptunghi este marcat cu o valoare pară *p*, exteriorul său este marcat cu *p* - 1, iar interiorul cu *p* + 1. Cea mai mare valoare pară este și ordinul maxim al unui triunghi.

Deoarece fiecare poziție din matrice este parcursă de două ori (la parcurgerea matricei și la un *Fill*), complexitatea este $O(L \times C)$.

28.1.2 *Rezolvare detaliată

28.1.3 Cod sursă

Listing 28.1.1: dreptunghiuri-1-100.cpp

1 #include <stdio.h>

```

2 #define DIM 1003
3
4 int tot, max, nrMax;
5 int N, M, i, j;
6 int A[DIM][DIM];
7 int ii, jj, t;
8
9 void drept(int i1, int j1, int i2, int j2, int niv)
10 {
11     int i, j;
12
13     if (i1>i2 || j1>j2)
14         return;
15
16     for (i=i1;i<=i2;i++)
17         for (j=j1;j<=j2;j++)
18             if (A[i][j] == 1)
19             {
20                 A[i][j] = 2;
21
22                 ii = i+1;
23                 while (A[ii][j] != 0)
24                 {
25                     A[ii][j] = 2;
26                     ii++;
27                 }
28                 ii--;
29                 jj = j+1;
30                 while (A[i][jj] != 0)
31                 {
32                     A[i][jj] = A[ii][jj] = 2;
33                     jj++;
34                 }
35                 jj--;
36
37                 for (t=i;t<=ii;t++)
38                     A[t][jj] = 2;
39
40                 tot++;
41                 if (niv+1>max)
42                 {
43                     max = niv+1;
44                     nrMax = 1;
45                 }
46                 else
47                     if (niv+1 == max)
48                         nrMax++;
49
50                 drept(i+2, j+2, ii-2, jj-2, niv+1);
51             }
52 }
53
54 int main()
55 {
56     FILE *f = fopen("dreptunghiuri.in", "r");
57     fscanf(f, "%d %d", &N, &M);
58     for (i=1;i<=N;i++)
59         for (j=1;j<=M;j++)
60             fscanf(f, "%d", &A[i][j]);
61     fclose(f);
62 //i1,j1,i2,j2 = miezul care poate contine doar 0 si 1
63
64     drept(1, 1, N, M, 0);
65
66     FILE *g = fopen("dreptunghiuri.out", "w");
67     fprintf(g, "%d %d %d", tot, max, nrMax);
68     fclose(g);
69
70     return 0;
71 }
```

Listing 28.1.2: dreptunghiuri-2-100.cpp

```

1 #include<cstdio>
2 #define dim 1005
```

```

3
4 using namespace std ;
5
6 struct coord
7 {
8     int lin, col ;
9 };
10
11 int dx[] = {-1, 0, 1, 0} ;
12 int dy[] = { 0, 1, 0, -1} ;
13 int a[dim][dim], L, C, totalDrept, adancimeMax, nrMax, t, v, top ;
14 coord st[dim * dim] ;
15
16 void Citire()
17 {
18     int i, j, x ;
19     freopen("dreptunghiuri.in", "r", stdin) ;
20     scanf("%d %d", &L, &C) ;
21     for (i=1 ; i<=L ; i++)
22         for (j=1 ; j<=C ; j++)
23         {
24             scanf("%d", &x) ;
25             a[i][j] = -x ;
26         }
27 }
28
29 inline int Interior(int x, int y)
30 {
31     return ( x>=0 && x<=L+1 && y>=0 && y<=C+1 ) ;
32 }
33
34 /*void Fill_1(int x, int y)
35 {
36     if ((a[x][y] == 0) && Interior(x,y))
37     {
38         a[x][y] = v ;
39         Fill_1(x+1, y) ;
40         Fill_1(x, y+1) ;
41         Fill_1(x-1, y) ;
42         Fill_1(x, y-1) ;
43     }
44 }
45 */
46 void Fill_1(int x, int y)
47 {
48     int x1, y1, i ;
49     top = 0 ;
50     st[top].lin = x ;
51     st[top].col = y ;
52     a[x][y] = v ;
53     while (top >= 0)
54     {
55         x = st[top].lin ;
56         y = st[top].col ;
57         top-- ;
58         for (i=0 ; i<4 ; i++)
59         {
60             x1 = x + dx[i] ;
61             y1 = y + dy[i] ;
62             if (Interior(x1,y1))
63             if (a[x1][y1] == 0)
64             {
65                 a[x1][y1] = v ;
66                 top++ ;
67                 st[top].lin = x1 ;
68                 st[top].col = y1 ;
69             }
70         }
71     }
72 }
73
74 void Fill(int x, int y)
75 {
76     int x1, y1, i ;
77     top = 0 ;
78     st[top].lin = x ;

```

```

79     st[top].col = y ;
80     a[x][y] = v ;
81     while (top >= 0)
82     {
83         x = st[top].lin ;
84         y = st[top].col ;
85         top-- ;
86         for (i=0 ; i<4 ; i++)
87         {
88             x1 = x + dx[i] ;
89             y1 = y + dy[i] ;
90             if (a[x1][y1] == t)
91             {
92                 a[x1][y1] = v ;
93                 top++ ;
94                 st[top].lin = x1 ;
95                 st[top].col = y1 ;
96             }
97         }
98     }
99 }
100
101 void Calcul()
102 {
103     int i, j ;
104     totalDrept = adancimeMax = nrMax = 0 ;
105     for (i=1 ; i<=L ; i++)
106         for (j=1 ; j<=C ; j++)
107             if (a[i][j] <= 0)
108             {
109                 v = a[i-1][j-1]+1 ;
110                 if (v % 2 == 0)
111                 {
112                     totalDrept++ ;
113                     if (adancimeMax < v)
114                     {
115                         adancimeMax = v ;
116                         nrMax = 1 ;
117                     }
118                     else if (adancimeMax == v) nrMax++ ;
119                 }
120                 t = a[i][j] ;
121                 Fill(i, j) ;
122             }
123 }
124
125 void Afisare()
126 {
127     freopen("dreptunghiuri.out", "w", stdout) ;
128     printf("%d %d %d\n", totalDrept, adancimeMax/2, nrMax) ;
129 }
130
131 int main()
132 {
133     Citire() ;
134     v = 1 ;
135     Fill_1(0,0) ;
136     Calcul()
137     Afisare() ;
138
139     return 0 ;
140 }
```

Listing 28.1.3: dreptunghiuri-3-70.cpp

```

1 #include<cstdio>
2 #define dim 1005
3
4 using namespace std ;
5
6 short a[dim][dim], L, C, totalDrept, adancimeMax, nrMax, t, v ;
7
8 void Citire()
9 {
10     short i, j, x ;
```

```

11     freopen("dreptunghiuri.in", "r", stdin) ;
12     scanf("%hd %hd", &L, &C) ;
13     for (i=1 ; i<=L ; i++)
14         for (j=1 ; j<=C ; j++)
15         {
16             scanf("%hd", &x) ;
17             a[i][j] = -x ;
18         }
19     }
20
21 inline short Interior(short x, short y)
22 {
23     return ( x>=0 && x<=L+1 && y>=0 && y<=C+1 ) ;
24 }
25
26 void Fill_1(short x, short y)
27 {
28     if ((a[x][y] == 0) && Interior(x,y))
29     {
30         a[x][y] = v ;
31         Fill_1(x+1, y) ;
32         Fill_1(x, y+1) ;
33         Fill_1(x-1, y) ;
34         Fill_1(x, y-1) ;
35     }
36 }
37
38 void Fill(short x, short y)
39 {
40     if (a[x][y] == t)
41     {
42         a[x][y] = v ;
43         Fill(x+1, y) ;
44         Fill(x, y+1) ;
45         Fill(x-1, y) ;
46         Fill(x, y-1) ;
47     }
48 }
49
50 void Calcul()
51 {
52     short i, j ;
53     totalDrept = adancimeMax = nrMax = 0 ;
54     for (i=1 ; i<=L ; i++)
55         for (j=1 ; j<=C ; j++)
56             if (a[i][j] <= 0)
57             {
58                 v = a[i-1][j-1]+1 ;
59                 if (v % 2 == 0)
60                 {
61                     totalDrept++ ;
62                     if (adancimeMax < v)
63                     {
64                         adancimeMax = v ;
65                         nrMax = 1 ;
66                     }
67                     else if (adancimeMax == v) nrMax++ ;
68                 }
69                 t = a[i][j] ;
70                 Fill(i, j) ;
71             }
72     }
73
74 void Afisare()
75 {
76     freopen("dreptunghiuri.out", "w", stdout) ;
77     printf("%hd %hd %hd\n", totalDrept, adancimeMax/2, nrMax) ;
78 }
79
80 int main()
81 {
82     Citire() ;
83     v = 1 ;
84     Fill_1(0,0) ;
85     Calcul() ;
86     Afisare() ;

```

```

87     return 0 ;
88 }
89 }
```

28.2 gaz

Problema 2 - gaz

100 de puncte

O stație de gaz are un rezervor subteran în care poate depozita cel mult L litri de gaz, dar există posibilitatea depozitării unei cantități suplimentare de gaz într-un rezervor închiriat de capacitate nelimitată pentru care se va plăti o taxă de C dolari pentru fiecare litru de gaz depozitat de la o zi la alta. Pentru a-și servi clienții, stația se aprovizează cu gaz cel mult o dată pe zi, dimineața. Prețul unui litru de gaz este de D dolari. Pentru fiecare aprovizionare trebuie plătită o taxă de P dolari în plus față de costul gazului comandat. În aceste condiții, comandarea unei cantități mari de gaz poate crește costul depozitării.

Stația de gaz se închide după N zile. Aceasta livrează clientilor săi G_i litri de gaz, din stocul său, la sfârșitul fiecărei zile i , unde $i = 1, 2, \dots, N$. Problema constă în a alege cantitățile de gaz ce vor fi comandate zilnic, astfel încât la sfârșitul celei de a N -a zi întreaga cantitate de pe stoc să fie consumată și costul total să fie minim. Se consideră că rezervorul este inițial gol.

Cerințe

Scrieți un program care determină costul total minim pentru ca stația să își servească clienții în cele N zile și întreaga cantitate de gaz să fie consumată la sfârșitul celei de a N -a zi.

Date de intrare

Pe prima linie a fișierului de intrare **gaz.in** apar patru numere naturale separate prin câte un spațiu, $L \ P \ D \ C$, cu semnificația din enunț. A doua linie conține numerele naturale $N \ G_1 \ G_2 \dots \ G_N$, separate prin câte un spațiu, unde N reprezintă numărul zilelor după care stația va fi închisă și G_i cantitatea de gaz necesară zilei i , $i = 1, 2, \dots, N$.

Date de ieșire

În fișierul de ieșire **gaz.out** se va scrie pe prima linie costul total minim cerut.

Restricții și precizări

- $1 \leq N \leq 2\,000$
- $1 \leq L, G_i \leq 1\,000$, $i = 1, 2, \dots, N$
- $1 \leq P, D, C \leq 5\,000$
- Pentru 80% din teste vom avea $N \leq 100$.

Exemplu:

gaz.in	gaz.out	Explicații
5 3 1 1 5 3 2 4 5 1	22	<p>O planificare optimă este cea descrisă în continuare.</p> <p>În dimineața primei zile se comandă 5 litri de gaz și se depozitează în rezervorul propriu. La sfârșitul zilei se livrează 3 litri. Costul primei zile este $5+3=8$. Pe timpul nopții vor rămâne 2 litri în rezervorul propriu, fără costuri suplimentare.</p> <p>În a doua zi stația nu se aprovizează, dar livrează 2 litri de gaz. Costul acestei zile este 0.</p> <p>În dimineața celei de a treia zile se comandă 10 litri de gaz, depozitându-se câte 5 litri în fiecare din cele două rezervoare. Seară se livrează 4 litri din rezervorul închiriat. În rezervorul propriu rămân pe timpul nopții 5 litri de gaz, iar în rezervorul închiriat încă un litru pentru care se va plăti un dolar. La costul total se va aduna $10+3+1=14$ dolari.</p> <p>În dimineața zilei a patra stația nu se aprovizează, dar seara livrează 5 litri de gaz, unul din rezervorul închiriat și 4 din rezervorul propriu. În timpul nopții nu vor fi costuri suplimentare de depozitare.</p> <p>În ultima zi se va livra ultimul litru de gaz din rezervorul propriu.</p>

Timp maxim de executare/test: **0.8** secunde pe Windows, **0.6** secunde pe Linux
Memorie: total **32 MB** din care pentru stivă **1 MB**

28.2.1 Indicații de rezolvare

Marius Stroe, student al Univ. Babeș Bolyai, Cluj Napoca

Soluțiile se bazează pe obținerea unor stări în funcție de zile, cantitatea totală de gaz ce trebuie livrată și calcularea lor folosind recurențe.

Soluția 1

Complexitate de timp: $O(N * (\sum G_i)^2)$

Memorie: $O(\sum G_i)$

Punctaj: 60 puncte

O primă soluție are starea formată din indicele zilei și cantitatea de gaz în surplus față de cantitatea de gaz ce trebuie livrată în această zi. Astfel, definim $bst[i, j]$ ca fiind costul minim pentru a ne găsi în ziua i și să avem j litri în surplus. Recurența ce calculează această valoare este:

```
bst[i, j] = min{bst[i-1][j+G[i-1]] + max(0, j-L) * C,
                 min{bst[i-1][k] + max(0, k-G[i-1]-L) * C+P : G[i-1] <= k < j+G[i-1]} }
```

Rezultatul este: $bst[n][0] + \sum G_i * D$.

Recurența are grija de costul suplimentar pentru depozitarea gazului peste noapte. Observăm că nu are rost să comandăm mai mulți litri decât cei care trebuie livrați în mod necesar. Astfel, când vom face o comandă, în cadrul recurenței de mai sus, va fi nevoie doar de costul P al taxei, deoarece vom aduna la rezultatul final $\sum G_i * D$. Pentru obținerea acestui punctaj va fi nevoie să fie reținute doar ultime două linii ale matricii.

Soluția 2

Complexitate de timp: $O(N * \sum G_i)$

Memorie: $O(\sum G_i)$

Punctaj: 80 puncte

Soluția anterioară poate fi îmbunătățită ținând o altă *matrice de minime parțiale*, $r[i, j] = \min(bst[i, j], r[i, j - 1])$. Recurența anterioară se modifică astfel:

```
bst[i, j] = min{bst[i-1][j+G[i-1]] + max(0, j-L) * C,
                 r[i-1][j + G[i-1] - 1] + P};
```

Rezultatul este același ca al soluției anterioare: $bst[n][0] + \sum G_i * D$.

Soluția 3

Complexitate de timp: $O(N^3)$

Memorie: $O(N^2)$

Punctaj: 90 puncte

Observăm că nu are rost să facem o altă comandă atâtă timp cât mai avem litri rămași dintr-o altă comandă. Starea ce se conturează calculează costul pentru a satisface cererile clientilor pentru zilele $i, i + 1, \dots, j$. Calculăm $bst[i, j]$ ca fiind acest *cost*. Recurența fie comandă gaz în ziua i pentru restul zilelor, fie împarte intervalul în două: în $i, i + 1, \dots, k$ și $k + 1, \dots, j$. Astfel, recurența este:

```
bst[i, j] = min{P + costul pentru depozitarea gazului în aceste zile,
                 min{ bst[i][k] + bst[k+1][j] : i <= k < j } }
```

Răspunsul este: $bst[1, n]$.

Soluția 4

Complexitate de timp: $O(N^2)$

Memorie: $O(N)$

Punctaj: 100 puncte

Ultima soluție îmbunătățește soluția anterioară, observând că putem calcula costul pentru a satisface cererile pentru zilele $1, 2, \dots, i$. Fie $bst[i]$ acest cost. Recurența se bazează pe ideea de a comanda gazul într-o zi *anterioară* zilei i . Astfel, avem:

```
bst[i] = min{bst[j] + costul pentru a depozita gazul în zilele j+1...i : 0 <= j < i}
```

Răspunsul este: $bst[n]$.

28.2.2 Rezolvare detaliată

28.2.3 Cod sursă

Listing 28.2.1: gaz-1-N_SG^2.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <algorithm>
4 #include <cassert>
5 #include <memory.h>
6
7 using namespace std;
8
9 const char iname[] = "gaz.in";
10 const char oname[] = "gaz.out";
11
12 const int MAX_N = 105;
13 const int MAX_L = 1005;
14 const int INF = 0x3f3f3f3f;
15
16 int bst[2][MAX_N * MAX_L];
17
18 int main(void) {
19     FILE *fi = fopen(iname, "r");
20     int L, P, D, C, N, G[MAX_N], S[MAX_N];
21
22     assert(fscanf(fi, "%d %d %d %d", &L, &P, &D, &C) == 4);
23     assert(fscanf(fi, "%d", &N) == 1);
24     for (int i = 0; i < N; ++ i)
25         assert(fscanf(fi, "%d", &G[i]) == 1);
26
27     S[N] = 0;
28     for (int i = N - 1; i >= 0; -- i)
29         S[i] = G[i] + S[i + 1];
30
31     int stp = 0;
32     memset(bst[stp], INF, sizeof bst[stp]);
33     for (int j = G[0]; j <= S[0]; ++ j)
34         bst[stp][j] = P;
35
36     for (int i = 0; i < N - 1; ++ i) {
37         memset(bst[stp ^ 1], INF, sizeof bst[stp ^ 1]);
38         for (int j = G[i]; j <= S[i]; ++ j) {
39             int alpha = max(0, j - G[i] - L);
40             for (int k = max(G[i + 1], j - G[i]); k <= S[i + 1]; ++ k)
41                 bst[stp ^ 1][k] = min(bst[stp ^ 1][k],
42                                       bst[stp][j] + alpha*C + (k > j-G[i])*P);
43         }
44         stp ^= 1;
45     }
46
47     fprintf(fopen(oname, "w"), "%d\n", bst[stp][G[N - 1]] + S[0] * D);
48 //    printf("%d\n", bst[stp][G[N - 1]] + S[0] * D);
49
50     fclose(fi);
51     return 0;
52 }
```

Listing 28.2.2: gaz-2-N_SG^2.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <algorithm>
4 #include <cassert>
5 #include <memory.h>
6
7 using namespace std;
8
9 const char iname[] = "gaz.in";
```

```

10 const char oname[] = "gaz.out";
11
12 const int MAX_N = 105;
13 const int MAX_Gi = 1005;
14 const int INF = 0x3f3f3f3f;
15
16 int bst[2][MAX_N * MAX_Gi];
17
18 int main(void)
19 {
20     FILE *fi = fopen(iname, "r");
21     int L, P, D, C, N, G[MAX_N], S[MAX_N];
22
23     assert(fscanf(fi, "%d %d %d %d", &L, &P, &D, &C) == 4);
24     assert(fscanf(fi, "%d", &N) == 1);
25     for (int i = 0; i < N; ++ i)
26         assert(fscanf(fi, "%d", &G[i]) == 1);
27
28     S[N] = 0;
29     for (int i = N - 1; i >= 0; -- i)
30         S[i] = G[i] + S[i + 1];
31
32     int stp = 0;
33     memset(bst[stp], INF, sizeof bst[stp]);
34     for (int j = G[0]; j <= S[0]; ++ j)
35         bst[stp][j] = P;
36
37     for (int i = 1; i < N; ++ i)
38     {
39         stp ^= 1;
40         memset(bst[stp], INF, sizeof bst[stp]);
41         for (int j = G[i]; j <= S[i]; ++ j)
42         {
43             for (int k = G[i - 1]; k <= S[i - 1]; ++ k)
44             {
45                 if (j > k - G[i - 1])
46                     bst[stp][j] = min(bst[stp][j], bst[stp ^ 1][k] +
47                                         max(0, k - G[i - 1] - L) * C + P);
48                 else if (j == k - G[i - 1])
49                     bst[stp][j] = min(bst[stp][j], bst[stp ^ 1][k] +
50                                         max(0, k - G[i - 1] - L) * C);
51             }
52         }
53     }
54
55     fprintf(fopen(oname, "w"), "%d\n", bst[stp][G[N - 1]] + S[0] * D);
56     fclose(fi);
57     return 0;
58 }
```

Listing 28.2.3: gaz-N^2.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <algorithm>
4 #include <cassert>
5 #include <memory.h>
6
7 using namespace std;
8
9 const char iname[] = "gaz.in";
10 const char oname[] = "gaz.out";
11
12 const int MAX_N = 2005;
13 const int INF = 0x3f3f3f3f;
14
15 typedef long long i64;
16
17 i64 bst[MAX_N], G[MAX_N], S[MAX_N];
18
19 int main(void)
20 {
21     FILE *fi = fopen(iname, "r");
22     FILE *fo = fopen(oname, "w");
23     i64 L, P, D, C, N, SG = 0;
```

```

24
25     assert(fscanf(fi, "%lld %lld %lld %lld", &L, &P, &D, &C) == 4);
26     assert(1 <= L && L <= 1000);
27     assert(1 <= P && P <= 5000);
28     assert(1 <= D && D <= 5000);
29     assert(1 <= C && C <= 5000);
30     assert(fscanf(fi, "%lld", &N) == 1);
31     assert(1 <= N && N <= 2000);
32
33     for (int i = 1; i <= N; ++ i)
34     {
35         assert(fscanf(fi, "%lld", &G[i]) == 1);
36         assert(1 <= G[i] && G[i] <= 1000);
37         SG += G[i];
38     }
39
40     memset(bst, INF, sizeof bst);
41
42     bst[0] = 0;
43     for (int i = 1; i <= N; ++ i)
44     {
45         int count = 0, sum = G[i];
46         for (int j = i - 1; j >= 0; -- j)
47         {
48             bst[i] = min(bst[i], bst[j] + count * C + P);
49             sum += G[j];
50             count += max(0LL, sum - G[j] - L);
51         }
52     }
53
54     fprintf(fopen(oname, "w"), "%lld\n", bst[N] + SG * D);
55 // printf("%lld\n", bst[N] + SG * D);
56 fclose(fi), fclose(fo);
57 return 0;
58 }
```

Listing 28.2.4: gaz-N^3.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <algorithm>
4 #include <cassert>
5 #include <memory.h>
6
7 using namespace std;
8
9 const char iname[] = "gaz.in";
10 const char oname[] = "gaz.out";
11
12 const int MAX_N = 2005;
13 const int INF = 0x3f3f3f3f;
14
15 typedef long long i64;
16
17 i64 bst[MAX_N][MAX_N];
18
19 int main(void)
20 {
21     FILE *fi = fopen(iname, "r");
22     FILE *fo = fopen(oname, "w");
23
24     int L, P, D, C, N, SG = 0;
25     i64 G[MAX_N], S[MAX_N] = {0};
26
27     assert(fscanf(fi, "%d %d %d %d", &L, &P, &D, &C) == 4);
28     assert(fscanf(fi, "%d", &N) == 1);
29     for (int i = 0; i < N; ++ i)
30     {
31         assert(fscanf(fi, "%d", &G[i]) == 1);
32         SG += G[i];
33         S[i] = (i > 0 ? S[i - 1] : 0) + G[i];
34     }
35
36     for (int i = 0; i < N; ++ i)
37         bst[i][i] = P;
```

```

38
39     for (int d = 1; d < N; ++ d) for (int i = 0; i < N - d; ++ i)
40     {
41         int j = i + d;
42
43         int sum = S[j] - (i > 0 ? S[i - 1] : 0);
44         bst[i][j] = P;
45         for (int k = i; k <= j; ++ k)
46         {
47             sum -= G[k];
48             bst[i][j] += max(0, sum - L) * C;
49         }
50
51         for (int k = i; k < j; ++ k)
52             bst[i][j] = min(bst[i][j], bst[i][k] + bst[k + 1][j]);
53     }
54
55     fprintf(fopen(oname, "w"), "%lld\n", bst[0][N - 1] + SG * D);
56 // printf("%lld\n", bst[0][N - 1] + SG * D);
57
58     fclose(fi), fclose(fo);
59     return 0;
60 }
```

Listing 28.2.5: gaz-N_SG.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <algorithm>
4 #include <cassert>
5 #include <memory.h>
6
7 using namespace std;
8
9 const char iname[] = "gaz.in";
10 const char oname[] = "gaz.out";
11
12 const int MAX_N = 405;
13 const int MAX_Gi = 1005;
14 const int INF = 0x3f3f3f3f;
15
16 int bst[2][MAX_N * MAX_Gi], r[2][MAX_N * MAX_Gi];
17
18 int main(void)
19 {
20     FILE *fi = fopen(iname, "r");
21     int L, P, D, C, N, G[MAX_N], S[MAX_N];
22
23     assert(fscanf(fi, "%d %d %d %d", &L, &P, &D, &C) == 4);
24     assert(1 <= L && L <= 1000);
25     assert(1 <= P && P <= 5000);
26     assert(1 <= D && D <= 5000);
27     assert(1 <= C && C <= 5000);
28     assert(fscanf(fi, "%d", &N) == 1);
29     assert(1 <= N && N <= 2000);
30
31     for (int i = 0; i < N; ++ i)
32     {
33         assert(fscanf(fi, "%d", &G[i]) == 1);
34         assert(1 <= G[i] && G[i] <= 1000);
35     }
36
37     S[N] = 0;
38     for (int i = N - 1; i >= 0; -- i)
39         S[i] = G[i] + S[i + 1];
40
41     int stp = 0;
42     memset(bst[stp], INF, sizeof bst[stp]);
43     for (int j = G[0]; j <= S[0]; ++ j)
44         bst[stp][j] = P;
45
46     for (int i = 1; i < N; ++ i)
47     {
48         stp ^= 1;
49         memset(bst[stp], INF, sizeof bst[stp]);
50
51         for (int j = G[i]; j <= S[i]; ++ j)
52             bst[stp][j] = P;
53
54         for (int k = i; k < N; ++ k)
55             bst[stp][j] = min(bst[stp][j], bst[stp][k] + bst[k + 1][j]);
56
57         for (int j = G[i]; j <= S[i]; ++ j)
58             bst[stp][j] = min(bst[stp][j], bst[stp][j] + bst[i][j]);
59
60         for (int j = G[i]; j <= S[i]; ++ j)
61             bst[stp][j] = min(bst[stp][j], bst[stp][j] + bst[i][j] * C);
62
63         for (int j = G[i]; j <= S[i]; ++ j)
64             bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
65
66         for (int j = G[i]; j <= S[i]; ++ j)
67             bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
68
69         for (int j = G[i]; j <= S[i]; ++ j)
70             bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
71
72         for (int j = G[i]; j <= S[i]; ++ j)
73             bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
74
75         for (int j = G[i]; j <= S[i]; ++ j)
76             bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
77
78         for (int j = G[i]; j <= S[i]; ++ j)
79             bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
80
81         for (int j = G[i]; j <= S[i]; ++ j)
82             bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
83
84         for (int j = G[i]; j <= S[i]; ++ j)
85             bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
86
87         for (int j = G[i]; j <= S[i]; ++ j)
88             bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
89
90         for (int j = G[i]; j <= S[i]; ++ j)
91             bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
92
93         for (int j = G[i]; j <= S[i]; ++ j)
94             bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
95
96         for (int j = G[i]; j <= S[i]; ++ j)
97             bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
98
99         for (int j = G[i]; j <= S[i]; ++ j)
100            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
101
102         for (int j = G[i]; j <= S[i]; ++ j)
103            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
104
105         for (int j = G[i]; j <= S[i]; ++ j)
106            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
107
108         for (int j = G[i]; j <= S[i]; ++ j)
109            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
110
111         for (int j = G[i]; j <= S[i]; ++ j)
112            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
113
114         for (int j = G[i]; j <= S[i]; ++ j)
115            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
116
117         for (int j = G[i]; j <= S[i]; ++ j)
118            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
119
120         for (int j = G[i]; j <= S[i]; ++ j)
121            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
122
123         for (int j = G[i]; j <= S[i]; ++ j)
124            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
125
126         for (int j = G[i]; j <= S[i]; ++ j)
127            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
128
129         for (int j = G[i]; j <= S[i]; ++ j)
130            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
131
132         for (int j = G[i]; j <= S[i]; ++ j)
133            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
134
135         for (int j = G[i]; j <= S[i]; ++ j)
136            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
137
138         for (int j = G[i]; j <= S[i]; ++ j)
139            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
140
141         for (int j = G[i]; j <= S[i]; ++ j)
142            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
143
144         for (int j = G[i]; j <= S[i]; ++ j)
145            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
146
147         for (int j = G[i]; j <= S[i]; ++ j)
148            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
149
150         for (int j = G[i]; j <= S[i]; ++ j)
151            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
152
153         for (int j = G[i]; j <= S[i]; ++ j)
154            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
155
156         for (int j = G[i]; j <= S[i]; ++ j)
157            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
158
159         for (int j = G[i]; j <= S[i]; ++ j)
160            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
161
162         for (int j = G[i]; j <= S[i]; ++ j)
163            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
164
165         for (int j = G[i]; j <= S[i]; ++ j)
166            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
167
168         for (int j = G[i]; j <= S[i]; ++ j)
169            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
170
171         for (int j = G[i]; j <= S[i]; ++ j)
172            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
173
174         for (int j = G[i]; j <= S[i]; ++ j)
175            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
176
177         for (int j = G[i]; j <= S[i]; ++ j)
178            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
179
180         for (int j = G[i]; j <= S[i]; ++ j)
181            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
182
183         for (int j = G[i]; j <= S[i]; ++ j)
184            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
185
186         for (int j = G[i]; j <= S[i]; ++ j)
187            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
188
189         for (int j = G[i]; j <= S[i]; ++ j)
190            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
191
192         for (int j = G[i]; j <= S[i]; ++ j)
193            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
194
195         for (int j = G[i]; j <= S[i]; ++ j)
196            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
197
198         for (int j = G[i]; j <= S[i]; ++ j)
199            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
200
201         for (int j = G[i]; j <= S[i]; ++ j)
202            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
203
204         for (int j = G[i]; j <= S[i]; ++ j)
205            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
206
207         for (int j = G[i]; j <= S[i]; ++ j)
208            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
209
210         for (int j = G[i]; j <= S[i]; ++ j)
211            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
212
213         for (int j = G[i]; j <= S[i]; ++ j)
214            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
215
216         for (int j = G[i]; j <= S[i]; ++ j)
217            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
218
219         for (int j = G[i]; j <= S[i]; ++ j)
220            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
221
222         for (int j = G[i]; j <= S[i]; ++ j)
223            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
224
225         for (int j = G[i]; j <= S[i]; ++ j)
226            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
227
228         for (int j = G[i]; j <= S[i]; ++ j)
229            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
230
231         for (int j = G[i]; j <= S[i]; ++ j)
232            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
233
234         for (int j = G[i]; j <= S[i]; ++ j)
235            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
236
237         for (int j = G[i]; j <= S[i]; ++ j)
238            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
239
240         for (int j = G[i]; j <= S[i]; ++ j)
241            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
242
243         for (int j = G[i]; j <= S[i]; ++ j)
244            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
245
246         for (int j = G[i]; j <= S[i]; ++ j)
247            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
248
249         for (int j = G[i]; j <= S[i]; ++ j)
250            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
251
252         for (int j = G[i]; j <= S[i]; ++ j)
253            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
254
255         for (int j = G[i]; j <= S[i]; ++ j)
256            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
257
258         for (int j = G[i]; j <= S[i]; ++ j)
259            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
260
261         for (int j = G[i]; j <= S[i]; ++ j)
262            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
263
264         for (int j = G[i]; j <= S[i]; ++ j)
265            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
266
267         for (int j = G[i]; j <= S[i]; ++ j)
268            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
269
270         for (int j = G[i]; j <= S[i]; ++ j)
271            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
272
273         for (int j = G[i]; j <= S[i]; ++ j)
274            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
275
276         for (int j = G[i]; j <= S[i]; ++ j)
277            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
278
279         for (int j = G[i]; j <= S[i]; ++ j)
280            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
281
282         for (int j = G[i]; j <= S[i]; ++ j)
283            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
284
285         for (int j = G[i]; j <= S[i]; ++ j)
286            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
287
288         for (int j = G[i]; j <= S[i]; ++ j)
289            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
290
291         for (int j = G[i]; j <= S[i]; ++ j)
292            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
293
294         for (int j = G[i]; j <= S[i]; ++ j)
295            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
296
297         for (int j = G[i]; j <= S[i]; ++ j)
298            bst[stp][j] = min(bst[stp][j], bst[stp][j] - G[i] * C);
299
299 }
```

```

50     r[stp ^ 1][0] = bst[stp ^ 1][0];
51     for (int j = 1; j <= S[i - 1]; ++ j)
52         r[stp ^ 1][j] = min(bst[stp ^ 1][j], r[stp ^ 1][j - 1]);
53     for (int j = G[i]; j <= S[i]; ++ j) {
54         bst[stp][j] = bst[stp ^ 1][j + G[i - 1]] + max(0, j - L) * C;
55         bst[stp][j] = min(bst[stp][j], r[stp ^ 1][j + G[i - 1] - 1] + P);
56     }
57 }
58
59 fprintf(fopen(oname, "w"), "%d\n", bst[stp][G[N - 1]] + S[0] * D);
60 printf("%d\n", bst[stp][G[N - 1]] + S[0] * D);
61 fclose(fi);
62 return 0;
63 }

```

28.3 xp

Problema 3 - xp

100 de puncte

Se consideră 3 siruri, numite A , B și val , fiecare dintre ele având câte N elemente naturale nenule. Elementele din cadrul sirurilor sunt indexate de la 1 la N . Cunoscându-se $A[1]$, $B[1]$ și o valoare naturală nenulă P , regula după care se calculează elementele sirurilor este următoarea:

Pentru $2 \leq i \leq N$ avem:

$$\begin{aligned} A[i] &= ((A[i-1] + P - 1) \text{ XOR } (B[i-1] + 1)) \bmod P \\ B[i] &= ((A[i-1] + P - 1) \text{ OR } (B[i-1] + 1)) \bmod P \end{aligned}$$

Pentru $1 \leq i \leq N$ avem:

$$val[i] = \max_1, ((i \bmod P) \text{ XOR } (((A[i] + 1) \text{ AND } (B[i] + 1)) \bmod P)) \bmod P$$

Operațiile utilizate în formulele de mai sus au următoare semnificație:

- XOR : sau-exclusiv pe biți
- OR : sau pe biți
- AND : și pe biți
- $F \bmod G$ reprezintă restul împărțirii lui F la G

Definim $Prod[i]$ ca fiind egal cu: (produsul tuturor elementelor sirului val , cu excepția lui $val[i]$) mod Q .

Mai exact, $Prod[i] = (val[1] \cdot val[2] \cdot \dots \cdot val[i - 1] \cdot val[i + 1] \cdot \dots \cdot val[N]) \bmod Q$.

Cerințe

Să se calculeze valoarea $Rez = Prod[1] \text{ XOR } Prod[2] \text{ XOR } \dots \text{ XOR } Prod[N]$ (adică XOR între toate cele N valori $Prod[i]$, $1 \leq i \leq N$).

Date de intrare

Fișierul de intrare **xp.in** conține pe prima (și singura) linie 5 numere naturale separate prin câte un spațiu, reprezentând, în ordine, valorile N , $A[1]$, $B[1]$, P și Q .

Date de ieșire

Fișierul de ieșire **xp.out** va conține valoarea Rez .

Restricții și precizări

- $1 \leq N \leq 4\,000\,000$
- $2 \leq P \leq 1\,000\,000\,000$
- $2 \leq Q \leq 1\,000\,000\,000$
- $0 \leq A[1], B[1] \leq P - 1$
- Pentru 30% dintre teste vom avea $N \leq 10\,000$.
- Pentru alte 20% dintre teste vom avea $10\,001 \leq N \leq 200\,000$.
- Problema nu urmărește găsirea vreunei proprietăți speciale a relațiilor de generare a elementelor sirurilor A , B și val .

Exemplu:

xp.in	xp.out	Explicații
5 4 6 10 15	10	Se obțin următoarele siruri A, B și val: A[1]=4, B[1]=6, val[1]=4 A[2]=0, B[2]=5, val[2]=2 A[3]=5, B[3]=5, val[3]=5 A[4]=8, B[4]=4, val[4]=5 A[5]=0, B[5]=1, val[5]=5 Se obțin următoarele valori pentru sirul Prod (în ordine, de la 1 la 5): 10, 5, 5, 5, 5. Obținem Rez = 10 XOR 5 XOR 5 XOR 5 XOR 5 = 10.
3999999 9003 3333 30000 900330000	594979072	

Timp maxim de executare/test: **7.0** secunde

Memorie: total **1 MB**

28.3.1 Indicații de rezolvare

asist. univ. dr. ing. Mugurel Ionuț Andreica - Universitatea Politehnica din București

Soluția 1

Complexitate de timp: $O(N^2)$

Memorie: $O(1)$ sau $O(N)$

Punctaj: 30 puncte

Fiecare valoare $Prod[i]$ este calculată independent (în timp $O(N)$).

Soluția 2

Complexitate de timp: $O(N)$

Memorie: $O(N)$

Punctaj: 50 puncte

în cadrul acestei soluții se vor genera toate valorile $val[i]$ și se vor memora într-un vector. Apoi vom calcula valorile $PR[i] = (val[i] \cdot \dots \cdot val[N])modQ$, în timp $O(N)$ per total. Avem $PR[N] = val[N]$ și $PR[1 \leq i \leq N-1] = (val[i] \cdot PR[i+1])modQ$.

Apoi vom parcurge valorile $val[i]$ de la 1 la N , menținând pe parcurs produsul primelor $i-1$ elemente. Când ajungem la poziția i , fie PL acest produs. Atunci vom avea $Prod[i] = (PL \cdot PR[i+1])modQ$ (unde considerăm $PR[N+1] = 1$).

Astfel, putem calcula toate valorile $Prod[i]$ în timp $O(N)$ per total.

Soluția 3

Complexitate de timp: $O(N \log(N))$

Memorie: $O(\log(N))$

Punctaj: 100 puncte

Această soluție folosește tehnica de programare denumită *Divide et Impera*. Se va scrie o funcție recursivă `divide_et_impera(i, j, A[i], B[i], outProd)`, în care `outProd` va reprezenta produsul elementelor din afara intervalului de poziții $[i, j]$ (bineînțeles, modulo Q). Vom efectua primul apel `divide_et_impera(1, N, A[1], B[1], 1)`.

În cadrul funcției se vor efectua următoarele procesări.

Dacă $i = j$ atunci avem $Prod[i] = outProd$.

Altfel, fie $mij = (i + j)/2$.

Vom calcula produsul elementelor val de pe pozițiile i, \dots, mij (pornind de la $A[i]$ și $B[i]$, care sunt parametrii ai funcției). Fie acest produs PL și fie $A[mij+1]$ și $B[mij+1]$ valorile ce pot fi calculate imediat din $A[mij]$ și $B[mij]$ ($A[mij]$ și $B[mij]$ sunt obținute la sfârșitul traversării tuturor elementelor de pe pozițiile i, \dots, mij). Apelăm apoi `divide_et_impera(mij+1, j, A[mij+1], B[mij+1], outProd)`.

Apoi, pornind de la $A[mij+1]$ și $B[mij+1]$, vom calcula PR = produsul tuturor elementelor val de pe pozițiile $mij+1, \dots, j$ (modulo Q).

La final apelăm `divide_et_impera(i, mij, A[i], B[i], (outProd * PR) mod Q)`.

Complexitatea de timp a soluției este $O(N \log(N))$, iar memoria folosită este de ordinul $O(\log(N))$ (deoarece dimensiunea stivei de apeluri recursive ajunge la $\log(N)$).

Soluția 4

Complexitate de timp: $O(N)$

Memorie: $O(\sqrt{N})$

Punctaj: 100 puncte

Această soluție se bazează pe soluția 2, însă memorează vectorul PR doar din K în K poziții.

Când ajungem la o poziție i , avem produsul PL al elementelor dinaintea lui i , valoarea $PR[h]$ a următoarei poziții $h > i$ memorate din vectorul PR (h este la distanță de cel mult K de i), precum și un alt vector cu valorile elementelor de pe pozițiile de la i până la h . Procesând valorile cu atenție, putem obține o complexitate liniară ($O(N)$), iar memoria folosită este de ordinul $K + N/K$. Valoarea minimă a memoriei utilizate se obține pentru $K = \sqrt{N}$.

28.3.2 *Rezolvare detaliată

28.3.3 Cod sursă

Listing 28.3.1: xp.cpp

```

1 #include <stdio.h>
2 #include <time.h>
3
4 #define N2_SOL 0
5 #define MINV_SOL 0
6 #define N_MLE_SOL 0
7 #define N_K_SPLIT_SOL 0
8 #define DIVIDE_ET_IMPERA_SOL 1
9 #define DIVIDE_ET_IMPERA_OPT_SOL 0
10
11 #define max(a,b) ((a) >= (b) ? (a) : (b))
12 #define min(a,b) ((a) <= (b) ? (a) : (b))
13 #define A(Aprev, Bprev) (((Aprev + P - 1) ^ (Bprev + 1)) % P)
14 #define B(Aprev, Bprev) (((Aprev + P - 1) | (Bprev + 1)) % P)
15 #define val(Acurr, Bcurr, pos) (max(1,(((pos) % (P)) ^ \
16                                     (((Acurr + 1) & (Bcurr + 1)) % (P))) % (P)))
17 #define aggf(a, b) (((long long)(a) * (long long)(b)) % Q)
18
19 int N, P, Q, A1, B1;
20 int i, j, Ap, Bp, Ac, Bc, Ap2, Bp2, x, rez;
21
22 void readInputData(void)
23 {
24     freopen("xp.in", "r", stdin);
25     scanf("%d %d %d %d", &N, &A1, &B1, &P, &Q);
26 }
27
28 void n2_sol(void)
29 {
30     int crez;
31
32     rez = 0;
33
34     for (i = 1; i <= N; i++)
35     {
36         crez = 1;
37         if (i > 1)
38             crez = val(A1, B1, 1);
39
40         Ap = A1; Bp = B1;
41
42         for (j = 2; j <= N; j++)
43         {
44             Ac = A(Ap, Bp);
45             Bc = B(Ap, Bp);
46
47             if (j != i)
48                 crez = aggf(crez, val(Ac, Bc, j));
49
50             Ap = Ac; Bp = Bc;
51         }
52     }
53 }
```

```

53         rez ^= crez;
54     }
55
56     fprintf(stderr, "N^2 solution: rez=%d\n", rez);
57 }
58
59 int gcd(int a, int b)
60 {
61     if (b == 0) return a;
62     else return gcd(b, a % b);
63 }
64
65 long long extendedEuclid(long long a, long long b, long long *x, long long *y)
66 {
67     long long x1, y1, d;
68
69     if (b == 0)
70     {
71         *x = 1;
72         *y = 0;
73         return a;
74     }
75     else
76     {
77         d = extendedEuclid(b, a % b, &x1, &y1);
78         *x = y1;
79         *y = x1 - (a/b)*y1;
80         return d;
81     }
82 }
83
84 long long modularInverse(long long a)
85 {
86     long long x, y, d;
87
88     d = extendedEuclid((long long) Q, a, &x, &y);
89
90     while (y<0)
91         y = y + Q;
92
93     return (y % Q);
94 }
95
96 void minv_sol(void)
97 {
98     int crez, x, y, z, cnt;
99     int minv, d;
100
101    rez = 0;
102
103    crez = val(A1, B1, 1);
104    Ap = A1; Bp = B1;
105
106    for (i = 2; i <= N; i++)
107    {
108        Ac = A(Ap, Bp);
109        Bc = B(Ap, Bp);
110        crez = aggf(crez, val(Ac, Bc, i));
111        Ap = Ac; Bp = Bc;
112    }
113
114    Ac = A1; Bc = B1;
115    for (i = 1; i <= N; i++)
116    {
117        x = val(Ac, Bc, i);
118
119        d = gcd(x, Q);
120        minv = (int) modularInverse((long long) x);
121
122        if (d == 1)
123            z = ((long long) minv * (long long) (crez / d)) % Q;
124        else
125        {
126            for (z = y = 0; y < Q; y++)
127                if (z == crez)
128                    break;

```

```

129             else
130                 z = (z + d) % Q;
131
132             if (y == Q)
133                 fprintf(stderr, "i=%d: caz nasol\n");
134
135             z = ((long long) minv * (long long) (y)) % Q;
136         }
137
138         rez ^= z;
139
140         Ap = Ac; Bp = Bc;
141         Ac = A(Ap, Bp);
142         Bc = B(Ap, Bp);
143     }
144
145     fprintf(stderr, "minv solution: rez=%d\n", rez);
146 }
147
148 void n_mle_sol(void)
149 {
150     int *SP;
151     int crez, num_zero = 0;
152
153     SP = new int[N + 2];
154     SP[1] = val(A1, B1, 1);
155
156     Ap = A1; Bp = B1;
157     for (i = 2; i <= N; i++)
158     {
159         Ac = A(Ap, Bp);
160         Bc = B(Ap, Bp);
161         SP[i] = val(Ac, Bc, i);
162         Ap = Ac; Bp = Bc;
163     }
164
165     SP[N + 1] = 1;
166     for (i = N - 1; i >= 1; i--)
167         SP[i] = aggf(SP[i], SP[i + 1]);
168
169     rez = SP[2];
170
171     if (rez == 0)
172         num_zero++;
173
174     crez = val(A1, B1, 1);
175     Ap = A1; Bp = B1;
176
177 //fprintf(stderr, "N=%d\n", N);
178
179     for (i = 2; i <= N; i++)
180     {
181         if (N - i <= 100)
182             fprintf(stderr, "i=%d, A=%d, B=%d, val=%d\n", i - 1,
183                     Ap, Bp, val(Ap, Bp, (i - 1)));
184
185         rez ^= aggf(crez, SP[i + 1]);
186
187         if (aggf(crez, SP[i + 1]) == 0)
188             num_zero++;
189
190         Ac = A(Ap, Bp);
191         Bc = B(Ap, Bp);
192         crez = aggf(crez, val(Ac, Bc, i));
193         Ap = Ac; Bp = Bc;
194     }
195
196     fprintf(stderr, "N MLE solution: rez=%d (N=%d, num_zero_sol=%d)\n",
197             rez, N, num_zero);
198 }
199
200 #define K 3163
201
202 void n_k_split_sol(void)
203 {
204     int *AP, *BP, *SP, *valAux;

```

```

205     int AC, BC, ACaux, BCaux, crez, p, num_zero = 0, cnt = 0;
206
207     AP = new int[(N / K) + 2];
208     BP = new int[(N / K) + 2];
209
210     Ap = A1; Bp = B1;
211     for (i = 2; i <= N; i++)
212     {
213         Ac = A(Ap, Bp);
214         Bc = B(Ap, Bp);
215
216         if (i % K == 0)
217         {
218             cnt++;
219             AP[cnt] = Ac;
220             BP[cnt] = Bc;
221         }
222
223         Ap = Ac; Bp = Bc;
224     }
225
226     SP = AP;
227
228     if (cnt >= 1)
229     {
230         Ap = AP[cnt]; Bp = BP[cnt];
231         crez = 1;
232         for (i = cnt * K + 1; i <= N; i++)
233         {
234             Ac = A(Ap, Bp);
235             Bc = B(Ap, Bp);
236             crez = aggf(crez, val(Ac, Bc, i));
237             Ap = Ac; Bp = Bc;
238         }
239
240         SP[cnt] = aggf(val(AP[cnt], BP[cnt], (cnt * K)), crez);
241
242         for (j = cnt - 1; j >= 1; j--)
243         {
244             Ap = AP[j]; Bp = BP[j];
245             crez = 1;
246             for (i = j * K + 1; i < (j + 1) * K; i++)
247             {
248                 Ac = A(Ap, Bp);
249                 Bc = B(Ap, Bp);
250                 crez = aggf(crez, val(Ac, Bc, i));
251                 Ap = Ac; Bp = Bc;
252             }
253
254             SP[j] = aggf(aggf(val(AP[j], BP[j], (j * K)), crez), SP[j + 1]);
255         }
256     }
257
258     delete BP;
259
260     valAux = new int[K + 2];
261     valAux[1] = val(A1, B1, 1);
262     Ap = A1; Bp = B1;
263
264     if (N >= K - 1)
265     {
266         for (i = 2; i < K; i++)
267         {
268             Ac = A(Ap, Bp);
269             Bc = B(Ap, Bp);
270             valAux[i] = val(Ac, Bc, i);
271             Ap = Ac; Bp = Bc;
272         }
273
274         valAux[K] = SP[1];
275         for (i = K - 1; i >= 1; i--)
276             valAux[i] = aggf(valAux[i], valAux[i + 1]);
277     }
278     else
279     {
280         for (i = 2; i <= N; i++)

```

```

281         {
282             Ac = A(Ap, Bp);
283             Bc = B(Ap, Bp);
284             valAux[i] = val(Ac, Bc, i);
285             Ap = Ac; Bp = Bc;
286         }
287
288         valAux[N + 1] = 1;
289         for (i = N - 1; i >= 1; i--)
290             valAux[i] = aggf(valAux[i], valAux[i + 1]);
291     }
292
293     rez = 0;
294     AC = A1; BC = B1;
295     crez = 1; j = 1;
296     for (i = 1; i <= N; i++)
297     {
298         if (i % K == 0)
299         {
300             Ac = A(Ap, Bp);
301             Bc = B(Ap, Bp);
302             Ap = Ac; Bp = Bc;
303
304             if (i + K <= N)
305             {
306                 for (p = 1; p < K; p++)
307                 {
308                     Ac = A(Ap, Bp);
309                     Bc = B(Ap, Bp);
310                     valAux[p] = val(Ac, Bc, (i + p));
311                     Ap = Ac; Bp = Bc;
312                 }
313
314                 j++;
315                 valAux[K] = SP[j];
316                 for (p = K - 1; p >= 1; p--)
317                     valAux[p] = aggf(valAux[p], valAux[p + 1]);
318             }
319             else
320             {
321                 for (p = i + 1; p <= N; p++)
322                 {
323                     Ac = A(Ap, Bp);
324                     Bc = B(Ap, Bp);
325                     valAux[p - i] = val(Ac, Bc, p);
326                     Ap = Ac; Bp = Bc;
327                 }
328
329                 valAux[p - i] = 1;
330                 for (p = p - i - 1; p >= 1; p--)
331                     valAux[p] = aggf(valAux[p], valAux[p + 1]);
332             }
333         }
334
335         rez ^= aggf(crez, valAux[(i % K) + 1]);
336
337         if (aggf(crez, valAux[(i % K) + 1]) == 0)
338             num_zero++;
339
340         crez = aggf(crez, val(AC, BC, i));
341         ACaux = AC; BCaux = BC;
342         AC = A(ACaux, BCaux);
343         BC = B(ACaux, BCaux);
344     }
345
346     fprintf(stderr, "N K-split solution: rez=%d (N=%d, num_zero=%d)\n",
347             rez, N, num_zero);
348 }
349
350 int totalOp;
351
352 void dei(int li, int ls, int Ali, int Bli, int outVal)
353 {
354     int mid, Ap, Bp;
355
356     if (li == ls)

```

```

357         rez ^= outVal;
358     else
359     {
360         mid = (li + ls) >> 1;
361
362         x = val(Ali, Bli, li);
363         Ap = Ali; Bp = Bli;
364         for (i = li + 1; i <= mid; i++)
365         {
366             Ac = A(Ap, Bp);
367             Bc = B(Ap, Bp);
368             x = aggf(x, val(Ac, Bc, i));
369             Ap = Ac; Bp = Bc;
370             totalOp++;
371         }
372
373         dei(mid + 1, ls, A(Ap, Bp), B(Ap, Bp), aggf(outVal, x));
374
375         x = 1;
376         for (i = mid + 1; i <= ls; i++)
377         {
378             Ac = A(Ap, Bp);
379             Bc = B(Ap, Bp);
380             x = aggf(x, val(Ac, Bc, i));
381             Ap = Ac; Bp = Bc;
382             totalOp++;
383         }
384
385         dei(li, mid, Ali, Bli, aggf(outVal, x));
386     }
387 }
388
389 void divide_et_impera_sol(void)
390 {
391     rez = 0;
392     totalOp = 0;
393     dei(1, N, Ali, Bli, 1);
394     fprintf(stderr, "Divide et Impera solution: rez=%d (totalOp=%d)\n",
395             rez, totalOp);
396 }
397
398 #define LMAX 30
399
400 char c[LMAX];
401 int AleftR[LMAX], BleftR[LMAX], aggLeft[LMAX], aggRight[LMAX], lim[LMAX];
402 int lev, mid2, cnt;
403
404 void dei_opt(int li, int ls, int Ali, int Bli, int outVal)
405 {
406     int mid, Ap, Bp, x, y;
407
408     if (li == ls)
409         rez ^= outVal;
410     else
411     if (li > N)
412         return;
413     else
414     {
415         mid = (li + ls) >> 1;
416
417         if (mid > N)
418         {
419             dei_opt(li, mid, Ali, Bli, outVal);
420             return;
421         }
422
423         if (!c[lev])
424         {
425             cnt = 0;
426             mid2 = mid;
427             while (li < mid2)
428             {
429                 mid2 = (li + mid2) >> 1;
430                 cnt++;
431                 lim[cnt] = mid2;
432             }

```

```

433
434         x = y = val(Ali, Bli, li);
435         Ap = Ali; Bp = Bli;
436
437         if (lim[cnt] == li)
438         {
439             AleftR[lev + cnt] = Ap;
440             BleftR[lev + cnt] = Bp;
441             aggLeft[lev + cnt] = x;
442             c[lev + cnt] = 1;
443             cnt--;
444             y = 1;
445         }
446
447         for (i = li + 1; i <= mid; i++)
448         {
449             Ac = A(Ap, Bp);
450             Bc = B(Ap, Bp);
451             x = aggf(x, val(Ac, Bc, i));
452             y = aggf(y, val(Ac, Bc, i));
453             Ap = Ac; Bp = Bc;
454
455             if (cnt > 0 && i == lim[cnt])
456             {
457                 AleftR[lev + cnt] = Ac;
458                 BleftR[lev + cnt] = Bc;
459                 aggLeft[lev + cnt] = x;
460                 c[lev + cnt] = 1;
461                 aggRight[lev + cnt + 1] = y;
462                 y = 1;
463                 cnt--;
464             }
465
466             totalOp++;
467         }
468
469         aggRight[lev + 1] = y;
470
471         y = 1;
472         Ap2 = Ap; Bp2 = Bp;
473
474         cnt = min(N, ls);
475         for (i = mid + 1; i <= cnt; i++)
476         {
477             Ac = A(Ap2, Bp2);
478             Bc = B(Ap2, Bp2);
479             y = aggf(y, val(Ac, Bc, i));
480             Ap2 = Ac; Bp2 = Bc;
481             totalOp++;
482         }
483     }
484     else
485     {
486         x = aggLeft[lev];
487         y = aggRight[lev];
488         Ap = AleftR[lev];
489         Bp = BleftR[lev];
490     }
491
492     lev++;
493     dei_opt(li, mid, Ali, Bli, aggf(outVal, y));
494     lev--;
495
496     c[lev + 1] = 0;
497
498     lev++;
499     dei_opt(mid + 1, ls, A(Ap, Bp), B(Ap, Bp), aggf(outVal, x));
500     lev--;
501 }
502 }
503
504 void divide_et_impera_opt_sol(void)
505 {
506     int p2;
507     rez = lev = 0;
508     for (i = 0; i < LMAX; i++)

```

```

509         c[i] = 0;
510
511     p2 = 1;
512     while (p2 < N)
513         p2 <= 1;
514
515     totalOp = 0;
516     dei_opt(1, p2, A1, B1, 1);
517     fprintf(stderr, "Divide et Impera Opt solution: rez=%d (totalOp=%d)\n",
518             rez, totalOp);
519 }
520
521 void writeOutputData(void)
522 {
523     freopen("xp.out", "w", stdout);
524     printf("%d\n", rez);
525 }
526
527 int main()
528 {
529     int tstart = clock();
530
531     readInputData();
532
533     if (N2_SOL)
534         n2_sol();
535     else
536         if (MINV_SOL)
537             minv_sol();
538     else
539         if (N_MLE_SOL)
540             n_mle_sol();
541     else
542         if (N_K_SPLIT_SOL)
543             n_k_split_sol();
544     else
545         if (DIVIDE_ET_IMPERA_SOL)
546             divide_et_impera_sol();
547     else
548         if (DIVIDE_ET_IMPERA_OPT_SOL)
549             divide_et_impera_opt_sol();
550
551     writeOutputData();
552
553     fprintf(stderr, "Duration=%lf sec\n",
554             (double) (clock() - tstart) / CLOCKS_PER_SEC);
555
556     return 0;
557 }
```

28.4 mesaje

Problema 4 - mesaje

100 de puncte

După multe năzbătii făcute împreună, Alex și Cipri nu mai au voie să se întâlnească. Alex - strategul echipei - a planuit o nouă poznă și a decis să-i transmită prietenului său planul de luptă, constând din anumite cuvinte dintr-un mesaj $m[0]$. Pentru a nu fi descoperiți, i-a trimis ulterior mai multe mesaje $m[1], m[2], \dots$ lui Cipri, acesta trebuind să le deschifeze folosind convenția secretă stabilită la începutul prieteniei lor și să îl adacționeze". Fiecare mesaj $m[i]$ este format din mai multe cuvinte, separate prin câte un spațiu, numerotate cu valori consecutive, începând de la 1.

Pentru a afla planul, Cipri trebuie să găsească cea mai mare valoare $i \geq 0$ astfel încât mesajele $m[i]$ și $m[0]$ să conțină cel puțin un cuvânt identic având același număr de ordine în ambele mesaje. Din $m[0]$ se păstrează toate cuvintele care se găsesc și în mesajul $m[i]$ cu același număr de ordine ca în $m[0]$.

Cuvintele păstrate trebuie ordonate în ordine descrescătoare lexicografică a puterii lor. Puterea cuvântului cu numărul de ordine j în $m[0]$ este egală cu sirul ordonat descrescător al indicilor mesajelor în care apare cu același număr de ordine ca în $m[0]$. Astfel, un cuvânt care a apărut cu numărul de ordine 2 în mesajele $m[0], m[6]$ și $m[8]$ are puterea 8,6,0. Dacă două cuvinte au aceeași putere, vor rămâne în ordinea din mesajul inițial.

Lui Cipri nu i-a mai rămas decât să citească fiecare cuvânt de la dreapta la stânga și a descifrat tot planul de luptă!

Cerințe

Cunoscând mesajele transmise de Alex, ajutați-l pe Cipri să descifreze planul de luptă conform convenției secrete.

Date de intrare

Fișierul de intrare **mesaje.in** conține în ordine mesajele $m[0]$, $m[1]$, $m[2]$, ..., câte unul pe linie.

Date de ieșire

Fișierul de ieșire **mesaje.out** va conține pe prima linie numărul n de cuvinte ale planului de luptă, iar pe cea de a doua linie cele n cuvinte ale planului de luptă.

Restricții și precizări

- Mesajele sunt memorate câte unul pe linie, fiind formate din cuvinte separate prin căte un spațiu.
 - Lungimea unui cuvânt este de maxim 20 de caractere, litere mici ale alfabetului englez.
 - Lungimea unui mesaj este de maxim 30 002 de caractere.
 - Toate mesajele au același număr de cuvinte.
 - Fișierul de intrare conține cel puțin unul și cel mult 128 de mesaje.
 - Orice linie din fișierul de intrare (mesaj) se termină cu marcajul de sfârșit de linie (newline).

Caracterul newline nu va fi considerat ca făcând parte din mesaj.

- Nu există mesaje vide.
- Se acordă 40% din punctajul corespunzător fiecărui test pentru determinarea valorii n și întregul punctaj pentru rezolvarea corectă a ambelor cerințe.

Exemple:

mesaje.in	mesaje.out	Explicații
inosos yy ataeclud ni a yy ataeclud ni yy inosos ni yy inosos bb ataeclud ni acni in e enib	3 dulceata in sosoni	Mesajele $m[0]$ și $m[4]$ nu conțin cuvinte identice cu același număr de ordine. Mesajele $m[0]$ și $m[3]$ conțin trei cuvinte identice cu același număr de ordine: inosos, ataeclud, ni. În ordinea puterii, ele sunt: ataeclud 3,1,0, ni 3,1,0, inosos3,0.
miras ep maeg	3 sarim pe geam	Pentru că a primit un singur mesaj, planul de luptă conține oglinditele cuvintele din textul inițial având toate aceeași putere, citite de la dreapta la stânga.

Timp maxim de executare/test: **0.1** secunde

Memorie: total **2 MB** din care pentru stivă **1 MB**

28.4.1 Indicații de rezolvare

prof. Florentina Ungureanu - Colegiul Național de Informatică Piatra-Neamț
asist. univ. dr. ing. Mugurel Ionuț Andreica - Universitatea Politehnica din București

Se citește primul mesaj (m_0) și se rețin cuvintele sale într-un sir. Se citește câte un nou mesaj (m_i) și se compară fiecare cuvânt al său (c_j) cu cel de pe aceeași poziție în primul mesaj.

Dacă cele două coincid se adună în poziția j a unui sir X valoarea 2^{i-1} dacă $i < 64$, respectiv valoarea 2^{i-64} în poziția j a unui sir Y dacă $i \geq 64$.

Dacă s-a identificat cel puțin o pereche de cuvinte identice, se reține numărul de ordine al mesajului și numărul de cuvinte identice n .

Se ordonează cuvintele de pe pozițiile reținute descrescător după valorile corespunzătoare din sirurile Y și X și se afișează oglinditele lor.

28.4.2 *Rezolvare detaliată

28.4.3 Cod sursă

Listing 28.4.1: mesaje-1.cpp

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <vector>
4 #include <time.h>
5
6 using namespace std;
7
8 #define NMAX 129
9 #define MAXCUV 15001
10 #define LMAX 30010
11
12 char linie[LMAX], l0[LMAX];
13 char *m0[MAXCUV], *mi[MAXCUV];
14 int o[MAXCUV], oaux[MAXCUV];
15
16 unsigned long long vset2[MAXCUV][2];
17
18 int i, j, l, q, better, K, M, N;
19
20 void msort(int li, int ls)
21 {
22     int mid;
23
24     if (li < ls)
25     {
26         mid = (li + ls) >> 1;
27         msort(li, mid);
28         msort(mid + 1, ls);
29
30         i = li;
31         j = mid + 1;
32         l = li - 1;
33
34         while (i <= mid && j <= ls)
35         {
36             better = 0;
37             for (q = 0; q < 2; q++)
38                 if (vset2[o[i]][q] > vset2[o[j]][q])
39                 {
40                     better = 1;
41                     break;
42                 }
43             else
44                 if (vset2[o[i]][q] < vset2[o[j]][q])
45                 {
46                     better = -1;
47                     break;
48                 }
49
50             l++;
51
52             if (better >= 0)
53             {
54                 oaux[l] = o[i];
55                 i++;
56             }
57             else
58             {
59                 oaux[l] = o[j];
60                 j++;
61             }
62         }
63
64         while (i <= mid)
65         {

```

```

66             l++;
67             oaux[1] = o[i];
68             i++;
69         }
70
71         while (j <= ls)
72     {
73             l++;
74             oaux[1] = o[j];
75             j++;
76         }
77
78         for (i = li; i <= ls; i++)
79             o[i] = oaux[i];
80     }
81 }
82
83 void solve(void)
84 {
85     int i, j, L, imax = 0;
86
87     freopen("mesaje.in", "r", stdin);
88
89     for (j = 0; j < MAXCUV; j++)
90         vset2[j][0] = vset2[j][1] = 0;
91
92     for (i = 0; i < imax; i++)
93     {
94         memset(&linie, 0, sizeof(linie));
95         fgets((char*)&linie, LMAX, stdin);
96
97         L = strlen(linie);
98         while (L > 0 && (linie[L] < 'a' || linie[L] > 'z'))
99         {
100             linie[L] = 0;
101             L--;
102         }
103
104         if (L == 0)
105         {
106             M = i;
107             break;
108         }
109
110         N = 1;
111         for (j = 1; j < L; j++)
112             if (linie[j] == ' ')
113                 N++;
114
115         N = 1;
116         mi[0] = &linie[0];
117
118         for (j = 1; j < L; j++)
119             if (linie[j] == ' ')
120             {
121                 linie[j] = 0;
122                 mi[N] = &linie[j + 1];
123                 N++;
124             }
125
126         if (i == 0)
127         {
128             memcpy(10, linie, L + 1);
129             for (j = 0; j < N; j++)
130                 m0[j] = 10 + (mi[j] - linie);
131         }
132         else
133         {
134             for (j = 0; j < N; j++)
135             {
136                 if (!strcmp(mi[j], m0[j]))
137                 {
138                     imax = i;
139
140                     if (i >= 64)
141                         vset2[j][0] |= (1ULL << (i - 64));

```

```

142             else
143                 vset2[j][1] |= (1ULL << i);
144             }
145         }
146     }
147 }
148
149 if (imax == 0)
150 {
151     K = N;
152     for (j = 0; j < N; j++)
153         o[j] = j;
154 }
155 else
156 {
157     K = 0;
158     for (j = 0; j < N; j++)
159     {
160         if ((imax >= 64 && (vset2[j][0] & (1ULL << (imax - 64)))) ||
161             (imax < 64 && (vset2[j][1] & (1ULL << (imax)))))
162         {
163             o[K] = j;
164             K++;
165         }
166     }
167 }
168
169 msort(0, K - 1);
170 }
171
172 void writeOutputData(void)
173 {
174     freopen("mesaje.out", "w", stdout);
175 //fprintf(stderr, "%d\n", K);
176
177     printf("%d\n", K);
178     for (i = 0; i < K; i++)
179     {
180         if (i > 0)
181             printf(" ");
182
183         for (j = strlen(m0[o[i]]) - 1; j >= 0; j--)
184             printf("%c", m0[o[i]][j]);
185     }
186
187     printf("\n");
188 }
189
190 int main()
191 {
192     int tstart = clock();
193     solve();
194     writeOutputData();
195
196 // fprintf(stderr, "Duration=%.3lf sec\n",
197 //           (double) (clock() - tstart) / CLOCKS_PER_SEC);
198
199     return 0;
200 }
```

Listing 28.4.2: mesaje-2.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <sstream>
4 #include <string>
5 #include <vector>
6 #include <algorithm>
7 #include <cassert>
8
9 using namespace std;
10
11 const char iname[] = "mesaje.in";
12 const char oname[] = "mesaje.out";
13
```

```

14 const int MAX_MESAJE = 128;
15
16 vector < vector < string > > mesaje;
17 vector < pair < pair < vector < int >, int >, string > > words_in_common;
18
19 int compare(pair < pair < vector < int >, int >, string > a,
20             pair < pair < vector < int >, int >, string > b)
21 {
22     vector < int > va = a.first.first, vb = b.first.first;
23     for (int i = 0; i < (int) va.size(); ++ i)
24     {
25         if (va[i] != vb[i])
26             return va[i] > vb[i];
27     }
28     return a.first.second <= b.first.second;
29 }
30
31 int main(void)
32 {
33     ifstream in(iname);
34     string str;
35
36     while (getline(in, str))
37     {
38         assert(0 < str.size() && str.size() < 30003);
39         istringstream iss(str);
40         vector < string > temp;
41
42         while (iss >> str)
43         {
44             temp.push_back(str);
45             assert(str.size() < 21);
46         }
47         mesaje.push_back(temp);
48     }
49     in.close();
50
51     for (int i = 0; i < (int) mesaje.size() - 1; ++ i)
52         assert(mesaje[i].size() == mesaje[i + 1].size());
53     assert(mesaje.size() < 129);
54
55     for (int idx = (int) mesaje.size() - 1; idx >= 0 &&
56           words_in_common.size() == 0; idx --)
57     {
58         for (int i = 0; i < (int) mesaje[idx].size(); ++ i)
59         {
60             if (mesaje[0][i] == mesaje[idx][i])
61                 words_in_common.push_back(
62                     make_pair(make_pair(vector < int >(), i), mesaje[0][i]));
63         }
64     }
65
66     for (int i = 0; i < (int) words_in_common.size(); ++ i)
67     {
68         for (int j = 0; j < (int) mesaje.size(); ++ j)
69         {
70             if (words_in_common[i].second ==
71                 mesaje[j][words_in_common[i].first.second])
72                 words_in_common[i].first.first.push_back(j);
73         }
74
75         reverse(words_in_common[i].first.first.begin(),
76                 words_in_common[i].first.first.end());
77     }
78
79     sort(words_in_common.begin(), words_in_common.end(), compare);
80
81     ofstream out(ename);
82     out << words_in_common.size() << "\n";
83
84     for (int i = 0; i < (int) words_in_common.size(); ++ i)
85     {
86         reverse(words_in_common[i].second.begin(),
87                 words_in_common[i].second.end());
88         out << words_in_common[i].second << " ";
89     }

```

```

90     out.close();
91
92     return 0;
93 }
```

Listing 28.4.3: mesaje-3.cpp

```

1 #include <iostream>
2 #include <algorithm>
3 #include <string.h>
4
5 #define nmax 15001
6
7 using namespace std;
8
9 char m[30001],t[21],*p,x;
10
11 struct elem
12 {
13     unsigned long long hi, lo;
14     int poz;
15     char cuv[21];
16 } c[nmax], cc[nmax];
17
18 unsigned long long v,v1,v2;
19 int n,i,j,k,nrc,r,l,s,d,mj;
20
21 ifstream fin("mesaje.in");
22 ofstream fout("mesaje.out");
23
24 int cmp(elem x, elem y)
25 {
26     if(x.hi!=y.hi)
27         return x.hi>y.hi;
28     else
29         if(x.lo!=y.lo)
30             return x.lo>y.lo;
31         else
32             return x.poz<y.poz;
33 }
34
35 int main()
36 {
37     fin.get(m,30001);
38     p=strtok(m, " ");
39     nrc=0;
40     i=0;
41
42     while(p)
43     {
44         c[nrc].hi=cc[nrc].lo=0;
45         c[nrc].poz=cc[nrc].poz=nrc;
46         strcpy(c[nrc].cuv,p);
47         strcpy(cc[nrc].cuv,p);nrc++;
48         p=strtok(NULL, " ");
49     }
50
51     n=nrc;
52     fin.get();
53     while(fin.get(m,30001))
54     {
55         j=k=l=0;
56         v = i<64 ? (1<<i) : (1<<(i-64));
57         p=strtok(m, " ");
58         while(p)
59         {
60             if(!strcmp(p,c[j].cuv))
61             {
62                 i<64 ? c[j].lo+=v : c[j].hi+=v;
63                 k++;
64             }
65             p=strtok(NULL, " ");
66             j++;
67         }
68     }
69 }
```

```

69      if (k)
70      {
71          n=k;
72          r=i;
73          for (j=0; j<nrc; j++)
74              if (c[j].hi>=v || i<64 && c[j].lo>=v)
75                  cc[l++]=c[j];
76      }
77
78      fin.get ();
79      i++;
80  }
81
82  fout<<n<<'\n';
83
84  sort (cc, cc+n, cmp);
85
86  strrev(cc[0].cuv);
87  fout<<cc[0].cuv;
88
89  for (i=1;i<n;i++)
90  {
91      strrev(cc[i].cuv);
92      fout<<' ' <<cc[i].cuv;
93  }
94
95
96  fout<<'\n';
97  fin.close();
98  fout.close();
99 return 0;
100 }
```

Listing 28.4.4: mesaje-4.cpp

```

1 #include<fstream>
2 #include<iostream>
3 #include<cstring>
4
5 #define InFile "mesaje.in"
6 #define OutFile "mesaje.out"
7
8 using namespace std ;
9
10 unsigned int a[4][15001] ;
11 unsigned int poz[15001], n, ultim, k, numar ;
12 char v[30002] ;
13 char text[15001][22] ;
14
15 void RezolvaMesaj(int k)
16 {
17     unsigned int j, cat, rest ;
18     char *p, cuv[22];
19     cat = k / 32 ;
20     rest = k % 32 ;
21     j = 0;
22     while ((p = strchr(v, ' ')) != NULL)
23     {
24         j++;
25         strncpy(cuv, v, p - v) ;
26         cuv[p-v] = 0 ;
27         if (strcmp(cuv, text[j]) == 0)
28         {
29             a[cat][j] |= (1<<rest) ;
30             ultim = k ;
31         }
32         strcpy(v, p+1) ;
33     }
34
35     j++ ;
36     if (strcmp(text[j], v) == 0)
37     {
38         a[cat][j] |= (1<<rest) ;
39         ultim = k ;
40     }

```

```

41 }
42
43 void Descompune(char *v)
44 {
45     unsigned int j ;
46     char *p ;
47     j = 0 ;
48     while ((p = strchr(v, ' ')) != NULL)
49     {
50         j++ ;
51         strncpy(text[j], v, p - v) ;
52         text[j][p-v] = 0 ;
53         strcpy(v, p+1) ;
54     }
55
56     j++ ;
57     strcpy(text[j], v) ;
58     n = j ;
59     for (j=1 ; j<=n ; j++)
60         a[0][j] = 1 ;
61 }
62
63 void Citire()
64 {
65     ifstream fin(InFile) ;
66     fin.getline(v, 30000) ;
67     Descompune(v) ;
68     k = 1 ;
69     while (fin.getline(v, 30000))
70     {
71         RezolvaMesaj(k) ;
72         k++ ;
73     }
74     unsigned int i, rest, cat ;
75     cat = ultim / 32 ;
76     rest = ultim % 32 ;
77     k = 0 ;
78     for (i = 1 ; i<=n ; i++)
79         if (((a[cat][i] >> rest) & 1) == 1)
80             poz[++k] = i ;
81 }
82
83 int ComparaLinii(int k1, int k2)
84 {
85     int i ;
86     for (i=3 ; i>=0 ; i--)
87         if (a[i][k1] > a[i][k2]) return 1 ;
88         else if (a[i][k1] < a[i][k2]) return -1 ;
89     return 0 ;
90 }
91
92 void Sortare()
93 {
94     unsigned int i, j, aux ;
95     for (i=1 ; i<k ; i++)
96         for (j=i+1 ; j<=k ; j++)
97             if (ComparaLinii(poz[i], poz[j]) < 0)
98             {
99                 aux = poz[i] ;
100                poz[i] = poz[j] ;
101                poz[j] = aux ;
102            }
103     ofstream fout(OutFile) ;
104     fout<<k<<"\n" ;
105     for (i=1 ; i<=k ; i++)
106         fout<<strrev(text[poz[i]])<<" " ;
107     fout<<"\n" ;
108     fout.close() ;
109 }
110
111 int main()
112 {
113     Citire() ;
114     Sortare() ;
115
116     return 0 ;

```

117 }

Listing 28.4.5: mesaje-5.cpp

```

1 #include <iostream>
2 #include <algorithm>
3 #include <string.h>
4
5 #define nmax 15001
6
7 using namespace std;
8
9 char m[30001],t[21],*p,*x;
10
11 struct elem
12 {
13     long long hi, lo;
14     int poz;
15     char cuv[21];
16 } c[nmax],cc[nmax],aux;
17
18 long long v,v1,v2;
19 int n,i,j,k,nrc,r,l,s,d,mj;
20
21 ifstream fin("mesaje.in");
22 ofstream fout("mesaje.out");
23
24 int cmp(elem x, elem y)
25 {
26     if(x.hi!=y.hi)
27         return x.hi>y.hi;
28     else
29         if(x.lo!=y.lo)
30             return x.lo>y.lo;
31     else
32         return x.poz>y.poz;
33 }
34
35 int main()
36 {
37     fin.get(m,30001);
38     p=strtok(m, " ");
39     nrc=0;
40     i=0;
41     while(p)
42     {
43         c[nrc].hi=cc[nrc].lo=0;
44         c[nrc].poz=cc[nrc].poz=nrc;
45         strcpy(c[nrc].cuv,p);
46         strcpy(cc[nrc].cuv,p);
47         nrc++;
48         p=strtok(NULL, " ");
49     }
50     n=nrc;
51     fin.get();
52     while(fin.get(m,30001))
53     {
54         j=k=l=0;
55         v=i<64?(1<<i):(1<<(i-64));
56         p=strtok(m, " ");
57         while(p)
58         {
59             if(!strcmp(p,c[j].cuv))
60             {
61                 i<64?c[j].lo+=v:c[j].hi+=v;
62                 k++;
63             }
64             p=strtok(NULL, " ");
65             j++;
66         }
67         if(k)
68         {
69             n=k;
70             r=i;
71         }
72     }
73 }
```

```

72         for(j=0;j<nrc;j++)
73             if(c[j].hi>=v||i<64&&c[j].lo>=v)
74                 cc[l++]=c[j];
75     }
76
77     fin.get();
78     i++;
79 }
80
81     fout<<n<<' \n';
82     i=1;
83     while(i<n)
84         if(cc[i].hi>cc[i-1].hi||(cc[i].hi==cc[i-1].hi&&cc[i].lo>cc[i-1].lo) ||
85             (cc[i].hi==cc[i-1].hi&&
86              cc[i].lo==cc[i-1].lo&&cc[i].poz<cc[i-1].poz))
87             {
88                 aux=cc[i],cc[i]=cc[i-1],cc[i-1]=aux;
89                 if(i>1) i--;
90             }
91         else i++;
92
93     strrev(cc[0].cuv);
94     fout<<cc[0].cuv;
95
96     for (i=1;i<n;i++)
97     {
98         strrev(cc[i].cuv);
99         fout<< ' '<<cc[i].cuv;
100    }
101
102    fout<<' \n';
103    fin.close();
104    fout.close();
105    return 0;
106 }
```

Listing 28.4.6: mesaje-6.cpp

```

1 #include <fstream>
2 #include <string.h>
3
4 #define nmax 15001
5
6 using namespace std;
7
8 char m[30001],t[21],cuv[nmax][21],sel[nmax][21],*p,*x;
9 long long c[nmax],cc[nmax],C[nmax],CC[nmax],v,v1,v2;
10 int n,i,j,k,nrc,r,l,s,d,mj;
11
12 ifstream fin("mesaje.in");
13 ofstream fout("mesaje.out");
14
15 int main()
16 {
17     fin.get(m,30001);
18     p=strtok(m, " ");
19     nrc=0;
20     i=0;
21     while(p)
22     {
23         strcpy(sel[nrc],p);
24         strcpy(cuv[nrc++],p);
25         p=strtok(NULL, " ");
26     }
27
28     n=nrc;
29     fin.get();
30     while(fin.get(m,30001))
31     {
32         j=k=l=0;
33         v=i<64?1<<i:1<<(i-64);
34         p=strtok(m, " ");
35         while(p)
36         {
37             if(!strcmp(p,cuv[j]))
```

```

38         {
39             i<64?c[j]+=v:C[j]+=v;
40             k++;
41         }
42         p=strtok(NULL, " ");
43         j++;
44     }
45
46     if(k)
47     {
48         n=k;
49         r=i;
50         for(j=0; j<nrc; j++)
51             if(C[j]>=v)
52             {
53                 strcpy(sel[1],cuv[j]);
54                 cc[l++]=c[j];
55                 CC[l++]=C[j];
56             }
57     }
58
59     fin.get();
60     i++;
61 }
62
63 fout<<n<<'\n';
64 i=1;
65 while(i<n)
66 {
67     if(CC[i]>CC[i-1]||CC[i]==CC[i-1]&&cc[i]>cc[i-1])
68     {
69         v=cc[i],cc[i]=cc[i-1],cc[i-1]=v;
70         v=CC[i],CC[i]=CC[i-1],CC[i-1]=v;
71         strcpy(t,sel[i]);
72         strcpy(sel[i],sel[i-1]);
73         strcpy(sel[i-1],t);
74         if(i>1) i--;
75     }
76     else i++;
77
78     strrev(sel[0]);
79     fout<<sel[0];
80
81     for (i=1;i<n;i++)
82     {
83         strrev(sel[i]);
84         fout<<' '<<sel[i];
85     }
86
87     fout<<'\n';
88     fin.close();
89     fout.close();
90
91     return 0;
92 }
```

28.5 petrecere

Problema 5 - petrecere

100 de puncte

Se organizează o petrecere la care participă N băieți (numerotați de la 1 la N) și N fete (numerotate de la 1 la N). S-a decis ca petrecerea să dureze mai multe minute. În fiecare minut fetele și băieții formează o configurație de dans, adică N perechi, după una din următoarele reguli:

1. băiatul i dansează cu fata i ;
2. băiatul i dansează cu fata j și atunci obligatoriu băiatul j dansează cu fata i .

De exemplu, pentru $N = 7$, două configurații de dans posibile sunt:

- (1, 1) (2, 2) (3, 7)(4, 5) (5, 4) (6, 6) (7, 3)
- (1, 1) (2, 2) (3, 3)(4, 5) (5, 4) (6, 6) (7, 7)

Prin perechea (i, j) s-a notat faptul că băiatul i dansează cu fata j . Două configurații sunt distincte dacă ele diferă prin cel puțin o pereche.

Cerințe

Știind că în fiecare minut trebuie formate configurații de dans distințe, să se determine câte minute durează petrecerea.

Date de intrare

Fișierul de intrare **petrecere.in** conține pe prima linie un singur număr natural N .

Date de ieșire

Fișierul de ieșire **petrecere.out** va conține o singură linie pe care va fi scris un singur număr natural reprezentând durata în minute a petrecerii.

Restricții și precizări

- $1 \leq N \leq 2\,000$
- Răspunsul este un număr natural de maximum 3 000 de cifre.
- Pentru 20% din teste, vom avea $N \leq 11$.
- Pentru alte 20% din teste, rezultatul poate fi reprezentat pe 64 de biți cu semn.

Exemple:

petrecere.in	petrecere.out	Explicații
2	2	Configurațiile de dans sunt: (1,1) (2,2) (1,2) (2,1)
3	4	Configurațiile de dans sunt: (1,1) (2,2) (3,3) (1,1) (2,3) (3,2) (1,2) (2,1) (3,3) (1,3) (2,2) (3,1)

Timp maxim de executare/test: **0.1** secunde

Memorie: total **2 MB** din care pentru stivă **1 MB**

28.5.1 Indicații de rezolvare

Dan Pracsu

Problema se reduce la a determina numărul de permutări (p_1, p_2, \dots, p_n) pentru care

$p_i = i$ (punct fix) sau

$p_i = j$ și $p_j = i$ (ciclu de lungime 2)

Notăm cu $f[n]$ = numărul permutărilor având proprietățile de mai sus

Construim $f[n+1]$:

Dacă $p_{n+1} = n + 1$ (punct fix), atunci p_1, p_2, \dots, p_n se pot aseza în $f[n]$ moduri

Dacă $p_{n+1} = i$, atunci $p_i = n + 1$ și deci rămân de aranjat $n - 1$ poziții din permutare (în $f[n - 1]$ moduri)

Deoarece p_{n+1} poate fi orice i între 1 și n , obținem recurența:

$$f[n + 1] = f[n] + n * f[n - 1]$$

Date inițiale:

$$f[1] = 1$$

$$f[2] = 2$$

Sunt necesare implementarea operațiilor de adunare a două *numere mari* și înmulțirea unui *număr mare* cu un *număr întreg*.

Un algoritm de tip *backtracking* va lua 20 de puncte, iar recurența corectă, dar implementată *fără numere mari*, va lua 40 puncte.

28.5.2 *Rezolvare detaliată

28.5.3 Cod sursă

Listing 28.5.1: petrecere-1.cpp

```

1 #include<iostream>
2
3 #define dim 3001
4 #define InFile "petrecere.in"
5 #define OutFile "petrecere.out"
6
7 using namespace std ;
8
9 int a[dim], b[dim], c[dim], na, nb, nc, n ;
10
11 void Citire()
12 {
13     ifstream fin(InFile) ;
14     fin>>n ;
15     fin.close() ;
16 }
17
18 void Aduna(int *a, int na, int *b, int nb, int *c, int &nc)
19 {
20     int i, t ;
21     nc = na > nb ? na : nb ;
22     t = 0 ;
23     for (i=0 ; i<nc ; i++)
24     {
25         c[i] = a[i] + b[i] + t ;
26         if (c[i] > 9)
27         {
28             c[i] -= 10 ;
29             t = 1 ;
30         }
31         else t = 0 ;
32     }
33     if (t > 0)
34         c[nc++] = 1 ;
35 }
36
37 void Copie(int *a, int &na, int *b, int nb)
38 {
39     int i ;
40     na = nb ;
41     for (i=0 ; i<na ; i++)
42         a[i] = b[i] ;
43 }
44
45 void Produs(int *a, int &na, int x)
46 {
47     int i, t, p ;
48     t = 0 ;
49     for (i=0 ; i<na ; i++)
50     {
51         p = a[i] * x + t ;
52         a[i] = p % 10 ;
53         t = p / 10 ;
54     }
55
56     while (t > 0)
57     {
58         a[na++] = t %10 ;
59         t /= 10 ;
60     }
61 }
62
63 void Calcul()
64 {
65     int i ;
66     ofstream fout(OutFile) ;
67
68     if (n == 1)
69     {
70         fout<<"1\n" ;

```

```

71         fout.close() ;
72         return ;
73     }
74     if (n == 2)
75     {
76         fout<<"2\n" ;
77         fout.close() ;
78         return ;
79     }
80
81     a[0] = 1 ;
82     b[0] = 2 ;
83     na = nb = 1 ;
84     for (i=3 ; i <= n ; i++)
85     {
86         Produs(a, na, i-1) ;
87         Aduna(a, na, b, nb, c, nc) ;
88         Copie(a, na, b, nb) ;
89         Copie(b, nb, c, nc) ;
90     }
91
92     for (i=nc-1 ; i>=0 ; i--)
93         fout<<c[i] ;
94     fout<<"\n" ;
95     fout.close() ;
96 }
97
98 int main()
99 {
100     Citire() ;
101     Calcul() ;
102
103     return 0 ;
104 }
```

Listing 28.5.2: petrecere-2.cpp

```

1 /*
2  * Implementare cu backtracking
3  *          20 de puncte
4 */
5
6 #include<iostream>
7
8 using namespace std ;
9
10 int st[3001], viz[3001], n, top, nrSol ;
11
12 int Valid(int top, int i)
13 {
14     if (top == i) return 1 ;
15     if ((top > i) && (st[i] != top)) return 0 ;
16     return 1 ;
17 }
18
19 void Back(int top)
20 {
21     int i ;
22     if (top == n+1) nrSol++ ;
23     else
24         for (i=1 ; i<=n ; i++)
25             if (!viz[i] && Valid(top, i))
26             {
27                 viz[i] = 1 ;
28                 st[top] = i ;
29                 Back(top+1) ;
30                 viz[i] = 0 ;
31             }
32 }
33
34 int main()
35 {
36     ifstream fin("petrecere.in") ;
37     fin>>n ;
38     fin.close() ;
```

```

39     Back(1) ;
40
41     ofstream fout("petrecere.out") ;
42     fout<<nrsol<<"\n" ;
43     fout.close() ;
44
45
46     return 0 ;
47 }
```

Listing 28.5.3: petrecere-3.cpp

```

1  /*
2   * Implementare fara numere mari
3   *          40 de puncte
4   */
5  #include<fstream>
6
7  #define dim 3001
8  #define InFile "petrecere.in"
9  #define OutFile "petrecere.out"
10
11 using namespace std ;
12
13 int n ;
14 long long x, y, z ;
15
16 void Citire()
17 {
18     ifstream fin(InFile) ;
19     fin>>n ;
20     fin.close() ;
21 }
22
23 void Calcul()
24 {
25     int i ;
26     ofstream fout(OutFile) ;
27
28     if (n == 1)
29     {
30         fout<<"1\n" ;
31         fout.close() ;
32         return ;
33     }
34     if (n == 2)
35     {
36         fout<<"2\n" ;
37         fout.close() ;
38         return ;
39     }
40
41     x = 1 ;
42     y = 2 ;
43     for (i=3 ; i <= n ; i++)
44     {
45         z = y + (i-1) * x ;
46         x = y ;
47         y = z ;
48     }
49     fout<<z<<"\n" ;
50     fout.close() ;
51 }
52
53 int main()
54 {
55     Citire() ;
56     Calcul() ;
57
58     return 0 ;
59 }
```

Listing 28.5.4: petrecere-4.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <algorithm>
4 #include <cassert>
5 #include <memory.h>
6
7 using namespace std;
8
9 const char iname[] = "petrecere.in";
10 const char oname[] = "petrecere.out";
11
12 #define set(a, d) (memset(a, 0, sizeof a), a[0] = 1, a[1] = d)
13 #define copy(a, b) (memcpy(a, b, sizeof a))
14
15 void mul(int a[], int d)
16 {
17     int i, tr;
18     for (i = 1, tr = 0; i <= a[0] || tr > 0; ++ i, tr /= 10)
19         a[i] = (tr += a[i] * d) % 10;
20     assert(i - 1 <= 3000);
21     a[0] = i - 1;
22 }
23
24 void add(int a[], int b[])
25 {
26     int i, tr;
27     for (i = 1, tr = 0; i <= a[0] || i <= b[0] || tr > 0; ++ i, tr /= 10)
28         a[i] = (tr += a[i] + b[i]) % 10;
29     assert(i - 1 <= 3000);
30     a[0] = i - 1;
31 }
32
33 int main(void)
34 {
35     const int MAX_DIGITS = 3005;
36     int a[MAX_DIGITS], b[MAX_DIGITS], c[MAX_DIGITS], n;
37
38     assert(fscanf(fopen(iname, "r"), "%d", &n) == 1);
39     assert(1 <= n && n <= 2000);
40
41     if (n == 1)
42         set(a, 1);
43     else if (n == 2)
44         set(a, 2);
45     else if (n > 2) {
46         set(b, 2), set(c, 1);
47         for (int i = 3; i <= n; ++ i) {
48             mul(c, i - 1);
49             copy(a, b), add(a, c);
50             copy(c, b), copy(b, a);
51         }
52     }
53
54     FILE *fo = fopen(oname, "w");
55     for (int i = a[0]; i > 0; -- i)
56         fprintf(fo, "%d", a[i]);
57     fclose(fo);
58
59     return 0;
60 }

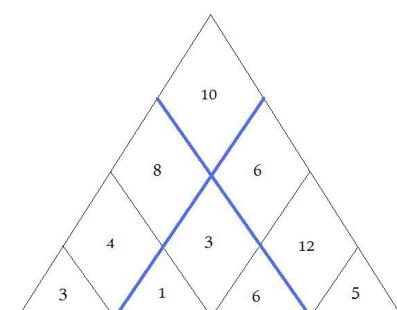
```

28.6 triunghi

Problema 6 - triunghi

Se consideră o placă de dimensiune n , de forma unui triunghi echilateral, ale căruia laturi sunt denumite A , B și C și au lungimea egală cu n . Pe laturile A și B sunt marcate câte $n - 1$ puncte care împart laturile în n porțiuni egale. Din fiecare punct marcat de pe latura A se trasează un segment paralel cu latura B , iar din fiecare punct marcat de pe latura B se trasează un segment paralel cu latura A .

100 de puncte



Celălalt capăt al fiecărui segment trasat se află pe latura C . În felul acesta, placa triunghiulară conține $n \cdot (n + 1)/2$ plăci elementare (care nu sunt traversate de niciun segment).

Astfel, pe o placă triunghiulară de dimensiune $n = 4$, ca în figură, avem 6 plăci elementare în formă de romb și 4 în formă de triunghi (cele cu o latură pe latura C a plăcii triunghiulare).

Se dorește împărțirea triunghiului în plăci elementare cu cost total minim. Dacă $n = 1$ costul împărțirii este 0. Pentru $n \geq 2$ singura operație permisă este tăierea de la un capăt la altul de-a lungul unui segment de lungime maximă, obținându-se un triunghi de dimensiune $n - 1$ și o bandă. Banda va fi împărțită în plăci elementare prin tăieri de-a lungul segmentelor de lungime 1 ce separă plăcile elementare care o compun. Triunghiul obținut va fi împărțit mai departe în plăci elementare folosind în mod repetat operația descrisă mai sus.

Costul total al împărțirii triunghiului de dimensiune n în plăci elementare este egal cu costul tăierii de-a lungul segmentului de lungime maximă, plus costurile împărțirii benzii și triunghiului de dimensiune $n - 1$ obținute, în plăci elementare.

Pe fiecare placă elementară este scris un număr. Costul unei tăieturi (fie că are loc într-un triunghi sau într-o bandă) este egal cu suma valorilor din plăcile elementare care au o latură comună cu segmentul pe care se face tăietura, înmulțită cu lungimea segmentului.

Pentru un triunghi de dimensiune $n \geq 2$ există exact 2 posibilități de a efectua o operație (corespunzătoare celor 2 segmente de lungime maximă, unul paralel cu latura A , iar celălalt paralel cu latura B).

O tăiere pe direcția $NV - SE$ (paralelă cu latura B) în triunghiul din figură are costul $(8 + 10 + 3 + 6 + 6 + 12) \cdot 3 = 135$. Costul împărțirii în plăci elementare a benzii obținute este egal cu $(10 + 6) \cdot 1 + (6 + 12) \cdot 1 + (12 + 5) \cdot 1 = 51$.

Cerințe

Să se determine costul total minim necesar împărțirii plăcii triunghiulare în plăci elementare.

Date de intrare

În fișierul **triunghi.in** pe prima linie se află un număr natural nenul n , reprezentând dimensiunea plăcii. Pe a doua linie apar separate prin câte un spațiu $n \cdot (n + 1)/2$ numere naturale, reprezentând valorile plăcilor elementare în ordinea parcurgerii de sus în jos și apoi de la stânga la dreapta, conform figurii de mai sus.

Date de ieșire

În fișierul **triunghi.out** se va scrie pe prima linie costul total minim cerut.

Restricții și precizări

- $1 \leq n \leq 1000$
- $0 \leq$ numărul de pe o placă elementară $\leq 2\ 000\ 000\ 000$
- Se garantează că rezultatul se va încadra pe 32 biți.
- Pentru 50% din teste vom avea $n \leq 400$.

Exemple:

triunghi.in	triunghi.out	Explicații
4 10 8 6 4 3 12 3 1 6 5	235	<p>Pentru a asigura costul total minim se poate realiza o tăietură pe direcția NE-SV de cost $3 \cdot (10 + 6 + 8 + 3 + 4 + 1) = 96$ la care se adaugă costul tăierii benzii în plăci elementare $3+4+4+8+8+10=37$.</p> <p>Placa triunghiulară rămasă va fi tăiată pe direcția NE-SV. Costul tăierii este $2 \cdot (3+6+6+12)=54$, tăierea benzii în plăci elementare are costul $1+3+3+6=13$</p> <p>Placa triunghiulară rămasă poate fi tăiată pe direcția NV-SE. Costul tăierii este $6+12=18$, tăierea benzii în plăci elementare are costul $12+5=17$, costul total minim este 235.</p>

Timp maxim de executare/test: 0.5 secunde

Memorie: total 20 MB din care pentru stivă 2 MB

28.6.1 Indicații de rezolvare

prof. Marius Nicoli - C. N. "Frații Buzești" - Craiova

Mai întâi se poate roti triunghiul astfel încât vârful de sus sa ajungă elementul 1,1 dintr-o matrice pătratică, iar latura de jos ajunge diagonala secundară (asta pentru ușurință în implementare în continuare).

Acum se calculează $A[i][j] = \text{costul minim al împărțirii unui triunghi care are vârful în poziția } i, j \text{ și celelalte 2 vârfuri pe diagonala secundară, unul pe linia } i \text{ și celălalt pe coloana } j.$

Valorile se calculează pe direcții paralele cu diagonala, începând de la diagonală.

Astfel $A[i][j]$ se poate calcula în funcție de $A[i][j+1]$ și $A[i+1][j]$, calculate anterior.

Aici intervin și sume de secvențe ale elementelor de pe liniile i , $i+1$ și coloanele j , $j+1$.

Pentru a evita calculul repetat al acestora (ce ar ridica complexitatea la n^3) se face o *preprocesare* pentru a calcula aceste sume în $O(1)$. (pe fiecare linie și fiecare coloană se calculează sume parțiale de la începutul liniei/coloanei până la poziția curentă).

Se poate face și o rezolvare recursivă, ce poate porni cu triunghiul inițial și face autoapeluri în cele 2 triunghiuri mai mici. Pentru evitarea calculului repetat al unor valori trebuie folosită *memorizarea*. Este necesară și aici *preprocesarea* în vederea calculării rapide a sumelor.

Complexitate $O(n^2)$

28.6.2 *Rezolvare detaliată

28.6.3 Cod sursă

Listing 28.6.1: triunghi-1.cpp

```

1 //100p n^2 Marius Nicoli
2 #include <stdio.h>
3
4 #define DIM 1002
5
6 int A[DIM][DIM], L[DIM][DIM], C[DIM][DIM], D[DIM][DIM];
7 int N, i, j, ii, a1, a2, a3, a4, b1, b2, b3, b4;
8
9 int min(int a, int b)
10 {
11     return a<b?a:b;
12 }
13
14 int main()
15 {
16     FILE *f = fopen("triunghi.in", "r");
17     FILE *g = fopen("triunghi.out", "w");
18
19     fscanf(f, "%d", &N);
20
21     for (ii=1;ii<=N;ii++)
22         for (j = 1, i = ii; i>=1; j++, i--)
23         {
24             fscanf(f, "%d", &A[i][j]);
25             C[i][j] = C[i-1][j] + A[i][j];
26             L[i][j] = L[i][j-1] + A[i][j];
27         }
28
29     for (i=1;i<=N;i++)
30         D[i][N-i+1] = A[i][N-i+1];
31
32     for (i=1;i<N;i++)
33         D[i][N-i] = 2*A[i][N-i] + A[i][N-i+1] + A[i+1][N-i];
34
35     for (ii=N-2;ii>=1;ii--)
36     {
37         for (i=ii, j=1;i>=1;i--, j++)
38         {

```

```

39  /*
40      a1 = D[i+1][j];
41      a2 = L[i][N-i] - L[i][j-1];
42      a3 = L[i+1][N-(i+1)+1] - L[i+1][j-1];
43      a4 = 2 * (L[i][N-i+1] - L[i][j-1]) - A[i][N-i+1] - A[i][j];
44
45      b1 = D[i][j+1];
46      b2 = C[N-j][j] - C[i-1][j];
47      b3 = C[N-(j+1)+1][j+1] - C[i-1][j+1];
48      b4 = 2 * (C[N-j+1][j] - C[i-1][j]) - A[N-j+1][j] - A[i][j];
49 */
50     D[i][j] = min(
51         D[i+1][j] +
52             (L[i][N-i] - L[i][j-1]) * (N-i+1 - j) +
53             (L[i+1][N-(i+1)+1] - L[i+1][j-1]) * (N-i+1 - j) +
54             2 * (L[i][N-i+1] - L[i][j-1]) - A[i][N-i+1] - A[i][j],
55
56         D[i][j+1] +
57             (C[N-j][j] - C[i-1][j]) * (N-i+1 - j) +
58             (C[N-(j+1)+1][j+1] - C[i-1][j+1]) * (N-i+1 - j) +
59             2 * (C[N-j+1][j] - C[i-1][j]) - A[N-j+1][j] - A[i][j]);
60     }
61 }
62 fclose(f);
63
64 fprintf(g,"%d",D[1][1]);
65 fclose(g);
66 return 0;
67 }

```

Listing 28.6.2: triunghi-2.cpp

```

1 //100p n^2 memorizare Marius Nicoli
2 #include <stdio.h>
3
4 #define DIM 1002
5 #define INF 0x7f7f7f7f
6
7 int A[DIM][DIM], L[DIM][DIM], C[DIM][DIM], D[DIM][DIM];
8 int N, i, j, ii;
9
10 int min(int a, int b)
11 {
12     return a<b?a:b;
13 }
14
15 int rec(int i, int j)
16 {
17     if (D[i][j] != INF)
18         return D[i][j];
19
20     if (j == N-i+1)
21     {
22         D[i][j] = A[i][j];
23         return D[i][j];
24     }
25
26     if (j==N-i)
27     {
28         D[i][j] = 2*A[i][j] + A[i+1][j] + A[i][j+1];
29         return D[i][j];
30     }
31
32     D[i][j] = min(
33         (L[i][N-i] - L[i][j-1]) * (N-i+1 - j) +
34             (L[i+1][N-(i+1)+1] - L[i+1][j-1]) * (N-i+1 - j) +
35             2 * (L[i][N-i+1] - L[i][j-1]) - A[i][N-i+1] - A[i][j] +
36             rec(i+1,j),
37
38             (C[N-j][j] - C[i-1][j]) * (N-i+1 - j) +
39             (C[N-(j+1)+1][j+1] - C[i-1][j+1]) * (N-i+1 - j) +
40             2 * (C[N-j+1][j] - C[i-1][j]) - A[N-j+1][j] - A[i][j] +
41             rec(i,j+1)
42 );

```

```

43
44     return D[i][j];
45 }
46
47 int main() {
48
49     FILE *f = fopen("triunghi.in", "r");
50     FILE *g = fopen("triunghi.out", "w");
51
52     fscanf(f, "%d", &N);
53
54     for (ii=1;ii<=N;ii++)
55         for (j = 1, i = ii; i>=1; j++, i--)
56         {
57             fscanf(f, "%d", &A[i][j]);
58             C[i][j] = C[i-1][j] + A[i][j];
59             L[i][j] = L[i][j-1] + A[i][j];
60             D[i][j] = INF;
61         }
62
63     fclose(f);
64
65     fprintf(g, "%d", rec(1,1));
66     fclose(g);
67     return 0;
68 }
```

Listing 28.6.3: triunghi-3.cpp

```

1 //50p n^3 Marius Nicoli
2 #include <stdio.h>
3
4 #define DIM 1002
5
6 int A[DIM][DIM], L[DIM][DIM], C[DIM][DIM], D[DIM][DIM];
7 int N, i, j, ii, x1, x2, x3, x4, y1, y2, y3, y4, t;
8
9 int min(int a, int b)
10 {
11     return a<b?a:b;
12 }
13
14 int main()
15 {
16
17     FILE *f = fopen("triunghi.in", "r");
18     FILE *g = fopen("triunghi.out", "w");
19
20     fscanf(f, "%d", &N);
21
22     for (ii=1;ii<=N;ii++)
23         for (j = 1, i = ii; i>=1; j++, i--)
24         {
25             fscanf(f, "%d", &A[i][j]);
26         }
27
28     for (i=1;i<=N;i++)
29         D[i][N-i+1] = A[i][N-i+1];
30
31     for (i=1;i<N;i++)
32         D[i][N-i] = 2*A[i][N-i] + A[i][N-i+1] + A[i+1][N-i];
33
34     for (ii=N-2;ii>=1;ii--)
35     {
36         for (i=ii, j=1;i>=1;i--, j++)
37         {
38             x2 = 0;
39             for (t = N-i; t>j-1; t--)
40                 x2 += A[i][t];
41             x3 = 0;
42             for (t = N-(i+1)+1; t>j-1; t--)
43                 x3 += A[i+1][t];
44             x4 = 0;
45             for (t = N-i+1; t>j-1; t--)
46                 x4+=A[i][t];
```

```

47
48     x4*=2;
49     x4-=(A[i][N-i+1] + A[i][j]);
50
51     y2 = 0;
52     for (t = N-j; t>i-1; t--)
53         y2 += A[t][j];
54     y3 = 0;
55     for (t = N-(j+1)+1; t>i-1; t--)
56         y3 += A[t][j+1];
57
58     y4 = 0;
59     for (t = N-j+1; t>i-1; t--)
60         y4+=A[t][j];
61     y4*=2;
62     y4-= (A[N-j+1][j] + A[i][j]);
63
64     D[i][j] = min(D[i+1][j] + x2*(N-i+1 - j) + x3*(N-i+1 - j) + x4,
65                     D[i][j+1] + y2 * (N-i+1 - j) + y3 * (N-i+1 - j) + y4);
66 }
67
68 fclose(f);
69
70 printf(g, "%d", D[1][1]);
71 fclose(g);
72 return 0;
73 }
```

Listing 28.6.4: triunghi-4.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <algorithm>
4 #include <cassert>
5
6 using namespace std;
7
8 const char iname[] = "triunghi.in";
9 const char oname[] = "triunghi.out";
10
11 const int MAX_N = 1005;
12
13 #define FOR(i, a, b) for (int i = (a); i <= (b); ++ i)
14 #define FORR(i, b, a) for (int i = (b); i >= (a); -- i)
15
16 int T[MAX_N][MAX_N], a[2][MAX_N], b[2][MAX_N], bst[2][MAX_N];
17
18 int main(void)
19 {
20     FILE *fi = fopen(iname, "r");
21     int n;
22
23     assert(fscanf(fi, "%d", &n) == 1);
24     assert(1 <= n && n <= 1000);
25
26     FOR (i, 1, n) FOR (j, 1, i)
27         assert(fscanf(fi, "%d", &T[i][j]) == 1);
28
29     fclose(fi);
30
31     int stp = 0;
32     FOR (j, 1, n) a[stp][j] = b[stp][j] = T[n][j], bst[stp][j] = 0;
33     FORR (i, n - 1, 1)
34     {
35         stp ^= 1;
36         FOR (j, 1, i)
37         {
38             a[stp][j] = a[stp ^ 1][j + 1] + T[i][j];
39             b[stp][j] = b[stp ^ 1][j] + T[i][j];
40
41             bst[stp][j] = min(bst[stp ^ 1][j] +
42                               (2 * a[stp][j] - T[i][j] - T[n][j+n-i]) +
43                               (a[stp][j]+a[stp ^ 1][j] - T[n][j+n-i])*(n-i),
44                               bst[stp ^ 1][j + 1] +
45                               (2 * b[stp][j] - T[i][j] - T[n][j]) +

```

```

46                               (b[stp][j] + b[stp ^ 1][j+1] - T[n][j]) * (n-i));
47             }
48         }
49
50         fprintf(fopen(pname, "w"), "%d\n", bst[stp][1]);
51         printf("%d\n", bst[stp][1]);
52
53     return 0;
54 }
```

Listing 28.6.5: triunghi-5.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <algorithm>
4 #include <cassert>
5
6 using namespace std;
7
8 const char iname[] = "triunghi.in";
9 const char oname[] = "triunghi.out";
10
11 const int MAX_N = 1005;
12
13 #define FOR(i, a, b) for (int i = (a); i <= (b); ++ i)
14 #define FORR(i, b, a) for (int i = (b); i >= (a); -- i)
15
16 int T[MAX_N][MAX_N], a[MAX_N][MAX_N], b[MAX_N][MAX_N];
17
18 int main(void)
19 {
20     FILE *fi = fopen(iname, "r");
21     int n;
22
23     assert(fscanf(fi, "%d", &n) == 1);
24     assert(1 <= n && n <= 1000);
25
26     FOR (i, 1, n) FOR (j, 1, i)
27         assert(fscanf(fi, "%d", &T[i][j]) == 1);
28     fclose(fi);
29
30     FOR (j, 1, n) a[n][j] = b[n][j] = T[n][j];
31     FORR (i, n - 1, 1)
32     {
33         FOR (j, 1, i)
34         {
35             a[i][j] = a[i + 1][j + 1] + T[i][j],
36             b[i][j] = b[i + 1][j] + T[i][j];
37         }
38     }
39     int ans = 0;
40     for (int i = 1, j = 1; i < n; )
41     {
42         int cost_diag = (2 * a[i][j] - T[i][j] - T[n][j+n-i]) +
43                         (a[i][j] + a[i + 1][j] - T[n][j+n-i]) * (n - i);
44         int cost_down = (2 * b[i][j] - T[i][j] - T[n][j]) +
45                         (b[i][j] + b[i + 1][j + 1] - T[n][j]) * (n - i);
46         if (cost_down < cost_diag)
47             i++, j++, ans += cost_down;
48         else
49             i++, ans += cost_diag;
50     }
51
52     fprintf(fopen(oname, "w"), "%d\n", ans);
53     printf("%d\n", ans);
54
55     return 0;
56 }
```

Capitolul 29

ONI 2009

29.1 magic

Problema 1 - magic

100 de puncte

Se dă o matrice cu n linii și n coloane. Coloanele și liniile sunt etichetate cu numere de la 1 la $2n$, folosind fiecare număr câte o singură dată (fig. 1 - exemplu pentru $n = 3$). Vom nota sirul etichetelor asociat liniilor matricei o_1, o_2, \dots, o_n , iar sirul etichetelor asociat coloanelor matricei cu v_1, v_2, \dots, v_n (fig. 4).

Trebuie să se completeze fiecare element al matricei cu una dintre cifrele 1 sau 9 (fig. 2). Prin concatenarea cifrelor de pe o linie sau o coloană obținem un număr de n cifre. În total se obțin $2n$ numere. Aceste numere trebuie să fie distințe două câte două și aranjându-le în ordinea etichetelor asociate liniilor și coloanelor trebuie să fie în ordine crescătoare (fig. 3). Vom concatena cele $2n$ numere în ordinea etichetelor și obținem un singur număr de $2n^2$ cifre. Acest număr îl vom denumi *cheie magică*. Pentru exemplul din fig. 3 obținem cheia magică 111 191 199 911 919 991.

	3	5	1
2			
4			
6			

fig. 1

	3	5	1
2	1	9	1
4	9	1	1
6	9	9	1

fig. 2

1	111
2	191
3	199
4	911
5	919
6	991

fig. 3

	v_1	v_2	v_3
o_1			
o_2			
o_3			

fig. 4

Cerințe

Se dă x un număr natural, dimensiunea n a matricei și cele două siruri de etichete o_1, o_2, \dots, o_n respectiv v_1, v_2, \dots, v_n . Să se tipărească numărul de chei magice distințe (dacă $x = 1$) sau cea mai mică cheie magică ce se poate asocia matricei (dacă $x = 2$).

Date de intrare

Fisierul de intrare **magic.in** conține patru linii. Pe linia 1 se află numărul natural x (1 sau 2). Pe linia 2 se află numărul natural n . Pe linia 3 se află n numere naturale distințe separate prin câte un spațiu reprezentând sirul o_1, o_2, \dots, o_n iar pe linia 4 - n numere naturale distințe separate prin câte un spațiu reprezentând sirul v_1, v_2, \dots, v_n .

Date de ieșire

Fisierul de ieșire **magic.out** va conține o singură linie pe care va fi scris un număr natural care reprezintă:

- dacă $x = 1$, numărul cheilor magice distințe;
- dacă $x = 2$, cea mai mică cheie magică.

Restricții și precizări

- $3 \leq n \leq 5$
- Pentru fiecare fișier test există cel puțin o soluție.
- Pentru 50% dintre teste $x = 1$ (aflarea numărului de chei magice), iar pentru 50% $x = 2$ (aflarea celei mai mici chei magice)
- Pentru 20% dintre teste $n = 3$, 30% dintre teste $n = 4$ și pentru 50% dintre teste $n=5$.

Exemple:

magic.in	magic.out	Explicații																		
1 3 2 4 6 3 5 1	2	<p>Avem două soluții</p> <table style="margin-left: auto; margin-right: auto;"> <tr><td>1</td><td>9</td><td>1</td><td>1</td><td>9</td><td>1</td></tr> <tr><td>9</td><td>1</td><td>1</td><td>9</td><td>1</td><td>1</td></tr> <tr><td>9</td><td>9</td><td>1</td><td>9</td><td>9</td><td>9</td></tr> </table> <p>Numerele obținute în ordinea etichetărilor: (111, 191, 199, 911, 919, 991) respectiv (119, 191, 199, 911, 919, 999).</p>	1	9	1	1	9	1	9	1	1	9	1	1	9	9	1	9	9	9
1	9	1	1	9	1															
9	1	1	9	1	1															
9	9	1	9	9	9															
2 3 2 4 6 3 5 1	111191199911919991	Cele două chei magice sunt 111191199911919991 respectiv 119191199911919999, dintre care prima e mai mică.																		

Timp maxim de executare/test: **0.6** secunde pe Windows, **0.1** secunde pe Linux

Memorie: total **2 MB** din care pentru stivă **1 MB**

29.1.1 Indicații de rezolvare

prof. Zoltan Szabo, Gr. Sc. "P. Maior", Reghin

2	5	4
3		
1		
6		

fig. 1

2	5	4
3	1	9
1	1	1
6	9	1

fig. 2

$o_2=1$	1	1	1
$v_1=2$	1	1	9
$o_1=3$	1	9	9
$v_3=4$	9	1	1
$v_2=5$	9	1	9
$o_3=6$	9	9	1

fig. 3

\mathbf{v}_1	\mathbf{v}_2	\mathbf{v}_3
\bullet_1		
\bullet_2		
\bullet_3		

fig. 4

Cum se observă din figura 3, cele două siruri \mathbf{o} și \mathbf{v} formează o permutare a numerelor $1 \dots 2n$, iar conținutul liniilor este echivalent cu numerele naturale $0 \dots 2^{n-1}$.

Intr-adevăr dacă înlocuim 1 cu 0 și 9 cu 1 și toate cifrele de pe o linie le scriem grupate, pentru $n = 3$ vom avea următoarele 8 cazuri:

Linia matricei	schimbăt in 0-1	echivalentul in baza 10
111	000	0
119	001	1
191	010	2
199	011	3
911	100	4
919	101	5
991	110	6
999	111	7

Problema se va rezolva cu tehnica *backtracking*, generând toate combinațiile numerelor $0, 1, \dots, 2^n - 1$, luate câte $2n$, fiecare linie sau coloana a matricei va corespunde unui număr care se va introduce bit cu bit în matrice pe linie sau coloană după cum se cere în enunț.

Dacă un element al matricei completat orizontal intră în contradicție cu un element completat vertical atunci e caz de revenire, Dacă un câmp nu era completat sau era completat cu număr identic, atunci putem continua backtrackingul pentru nivelul următor.

Cu aceasta metodă se memorează prima soluție. Celelalte soluții se vor număra, astfel vom avea răspuns la ambele cerințe ale problemei.

29.1.2 *Rezolvare detaliată

29.1.3 Cod sursă

Listing 29.1.1: magic_50.cpp

```

1 //S. Ganceanu 50
2 #include <stdio.h>
3 #include <string.h>
4
5 const long MAXN = 10;
6
7 long lin_labels[MAXN], col_labels[MAXN];
8 long print_first_sol, N, nsol;
9 long mat[MAXN][MAXN], sol[2 * MAXN], vec[2 * MAXN];
10
11 void read_data()
12 {
13     long i;
14     FILE *f = fopen("magic.in", "rt");
15     long x;
16     fscanf(f, "%ld\n%ld\n", &x, &N);
17     print_first_sol = (x == 2);
18
19     for (i = 0; i < N; ++i)
20     {
21         fscanf(f, "%ld", lin_labels + i);
22         --lin_labels[i];
23     }
24
25     for (i = 0; i < N; ++i)
26     {
27         fscanf(f, "%ld", col_labels + i);
28         --col_labels[i];
29     }
30
31     fclose(f);
32 }
33
34 void solve()
35 {
36     long N2 = N * N;
37     long cfg, i, j;
38     for (cfg = 0; cfg < (1L << N2); ++cfg)
39     {
40         for (i = 0; i < N; ++i)
41             for (j = 0; j < N; ++j)
42             {
43                 long tmp = i * N + j;
44                 mat[i][j] = (cfg & (1L << tmp)) != 0;
45             }
46
47         for (i = 0; i < N; ++i)
48         {
49             long current = 0;
50             for (j = N - 1; j >= 0; --j)
51                 current += mat[i][j] * (1L << (N - 1 - j));
52
53             vec[lin_labels[i]] = current;
54         }
55
56         for (j = 0; j < N; ++j)
57         {
58             long current = 0;
59             for (long i = N - 1; i >= 0; --i)
60                 current += mat[i][j] * (1L << (N - 1 - i));
61
62             vec[col_labels[j]] = current;
63         }
64
65     long ok = 1;

```

```

66     for (i = 1; i < 2 * N; ++i)
67         if (vec[i] <= vec[i - 1])
68             ok = 0;
69
70     if (ok)
71     {
72         ++nsol;
73         if (nsol == 1)
74             memcpy(sol, vec, sizeof(vec));
75         else
76         {
77             long cmp = 0;
78             for (i = 0; i < 2 * N; ++i)
79             {
80                 if (sol[i] != vec[i])
81                 {
82                     cmp = sol[i] - vec[i];
83                     break;
84                 }
85             }
86
87             if (cmp > 0)
88                 memcpy(sol, vec, sizeof(vec));
89         }
90     }
91 }
92 }
93
94 void write_data()
95 {
96     long i, j;
97     FILE *f = fopen("magic.out", "wt");
98     if (print_first_sol)
99     {
100         for (i = 0; i < 2 * N; ++i)
101             for (j = 0; j < N; ++j)
102                 if (sol[i] & (1L << (N - 1 - j)))
103                     fprintf(f, "9");
104                 else
105                     fprintf(f, "1");
106     }
107     else
108         fprintf(f, "%ld", nsol);
109
110     fprintf(f, "\n");
111 }
112
113 int main()
114 {
115     read_data();
116     solve();
117     write_data();
118     return 0;
119 }
```

Listing 29.1.2: magic_S.cpp

```

1 //S. Ganeanu 100
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 const int MAXN = 10;
6
7 int first_sol, N;
8 int num_sol;
9 int startx[2 * MAXN], starty[2 * MAXN];
10 int dx[2 * MAXN], dy[2 * MAXN];
11 int mat[MAXN][MAXN];
12 int trace[MAXN][MAXN];
13 int viz[1 << MAXN];
14
15 void print_first_sol()
16 {
17     FILE *f = fopen("magic.out", "wt");
18     for (int k = 0; k < 2 * N; ++k) {
```

```

19     int sx, sy;
20     for (sx = startx[k], sy = starty[k]; sx < N && sy < N;
21         sx += dx[k], sy += dy[k]) {
22         fprintf(f, "%c", mat[sx][sy] ? '9' : '1');
23     }
24 }
25
26 fprintf(f, "\n");
27 fclose(f);
28 }
29
30 void write_num_sol()
31 {
32     FILE *f = fopen("magic.out", "wt");
33     fprintf(f, "%d\n", num_sol);
34     fclose(f);
35 }
36
37 void back(int lev, int last)
38 {
39     if (lev == 2 * N) {
40         // Found a solution.
41         ++num_sol;
42         if (first_sol) {
43             print_first_sol();
44             exit(0);
45         }
46         return;
47     }
48
49     int sx, sy;
50     int num_free = 0;
51
52     for (sx = startx[lev], sy = starty[lev]; sx < N && sy < N;
53         sx += dx[lev], sy += dy[lev]) {
54         if (trace[sx][sy] == -1) ++num_free;
55     }
56
57     for (int cfg = 0; cfg < (1 << num_free); ++cfg) {
58         int tmp = 0, current = 0, ind = 0;
59         for (sx = startx[lev]+(N-1)*dx[lev], sy=starty[lev]+(N-1)*dy[lev];
60             sx >= 0 && sy >= 0;
61             sx -= dx[lev], sy -= dy[lev], ++ind)
62         {
63             if (trace[sx][sy] == -1) {
64                 mat[sx][sy] = (cfg & (1 << tmp)) != 0;
65                 trace[sx][sy] = lev;
66                 current += (1 << ind) * mat[sx][sy];
67                 ++tmp;
68             } else {
69                 current += (1 << ind) * mat[sx][sy];
70             }
71         }
72
73         if (current > last && viz[current] == 0) {
74             viz[current] = 1;
75             back(lev + 1, current);
76             viz[current] = 0;
77         }
78
79         for (sx = startx[lev], sy = starty[lev]; sx < N && sy < N;
80             sx += dx[lev], sy += dy[lev])
81         {
82             if (trace[sx][sy] == lev)
83                 trace[sx][sy] = -1;
84         }
85     }
86 }
87
88 void read_data()
89 {
90     int i, x, j;
91     FILE *f = fopen("magic.in", "rt");
92
93     if (fscanf(f, "%d\n%d\n", &first_sol, &N)) ;
94

```

```

95     if (first_sol == 2) first_sol = 1;
96     else first_sol = 0;
97
98     for (i = 0; i < N; ++i) {
99         if (fscanf(f, "%d", &x)) ;
100        --x;
101        startx[x] = i;
102        starty[x] = 0;
103        dx[x] = 0;
104        dy[x] = 1;
105    }
106
107    for (i = 0; i < N; ++i) {
108        if (fscanf(f, "%d", &x)) ;
109        --x;
110        startx[x] = 0;
111        starty[x] = i;
112        dx[x] = 1;
113        dy[x] = 0;
114    }
115
116    for (i = 0; i < N; ++i) {
117        for (j = 0; j < N; ++j) {
118            trace[i][j] = -1;
119        }
120    }
121    fclose(f);
122}
123
124 int main()
125{
126    read_data();
127    back(0, -1);
128    write_num_sol();
129    return 0;
130}

```

29.2 reactii

Problema 2 - reactii

100 de puncte

Să considerăm o secvență de n substanțe chimice $s = s_1, s_2, \dots, s_n$. Substanțele sunt numerotate distinct de la 1 la n și fiecare substanță apare în secvența s o singură dată.

Să considerăm o subsecvență $s_{ij} = (s_i s_{i+1} \dots s_j)$ și să notăm cu \min_{ij} și \max_{ij} cel mai mic, respectiv cel mai mare număr din subsecvență.

Subsecvența respectivă constituie un interval dacă ea conține toate numerele naturale cuprinse între \min_{ij} și \max_{ij} .

Cu substanțele din secvența s se vor efectua diferite experimente. În timpul unui experiment pot reacționa două substanțe alăturate s_i și s_{i+1} doar dacă numerele lor de ordine sunt consecutive. În urma reacției se obține o nouă substanță, formată din substanțele care au reacționat, notată (s_i, s_{i+1}) . Mai mult, substanțele obținute pot reacționa dacă ele sunt alăturate, iar prin reunirea subsecvențelor de substanțe ce le compun se obține un interval.

Experimentul este declarat reușit dacă în final, urmând regulile de mai sus, se obține o singură substanță formată din toate cele n substanțe din secvența s , aceasta fiind declarată **stabilă**.

De exemplu, pentru $n = 6$ substanțe și secvența $s = 6, 3, 2, 1, 4, 5$ se poate proceda astfel:

Etapa	Acțiune	Configuracie
1.	Secvența inițială	6 3 2 1 4 5
2.	Reacționează substanță 2 cu 1 și se obține substanță (2,1)	6 3 (2,1) 4 5
3.	Reacționează substanță 4 cu substanță 5 și se obține substanță (4,5)	6 3 (2,1)(4,5)
4.	Reacționează substanță 3 și (2,1) rezultând (3,2,1)	6 (3,2,1)(4,5)
5.	Reacționează substanțele (3,2,1) și (4,5) rezultând substanță (3,2,1,4,5)	6 (3,2,1,4,5)
6.	Reacționează substanță 6 cu (3,2,1,4,5) și rezultă substanță stabilă (6,3,2,1,4,5)	(6,3,2,1,4,5)

Nu din orice secvență de substanțe se poate obține în urma reacțiilor o substanță finală stabilă.

Cerințe

Determinați pentru o secvență dată de substanțe, dacă în urma reacțiilor ce se pot produce conform regulilor din enunț rezultă o substanță stabilă.

Date de intrare

Fișierul de intrare **reactii.in** conține pe prima linie numărul natural n , numărul de substanțe. Pe cea de a doua linie se află un număr natural m , reprezentând numărul de secvențe de n substanțe din fișierul de intrare. Fiecare dintre următoarele m linii conține câte n numere naturale distincte, separate prin câte un spațiu, reprezentând o secvență de n substanțe.

Date de ieșire

Fișierul de ieșire **reactii.out** conține, pentru fiecare secvență de substanțe din fișierul de intrare, câte o linie, pe care este afișată valoarea 1 dacă pentru secvența respectivă se poate obține o substanță stabilă sau valoarea 0 în caz contrar.

Restricții și precizări

- $2 \leq n \leq 15\ 000$
- $1 \leq m \leq 20$
- La un moment dat pot reacționa doar două substanțe.

Exemple:

reactii.in	reactii.out	Explicații
6	1	
4	0	
6 3 2 1 5 4	1	
3 4 1 6 5 2	1	
2 3 1 5 4 6		
6 2 3 1 4 5		

Timp maxim de executare/test: **0.5** secunde pe Windows, **0.1** secunde pe Linux

Memorie: total **2 MB** din care pentru stivă **1 MB**

29.2.1 Indicații de rezolvare

prof. Marinel Șerban, Liceul de Informatică "Grigore Moisil" Iași

Soluția 1 - utilizarea unei *stive*

Se utilizează o stivă S , ale cărei elemente sunt subsecvențe l , $l+1$, ..., $l+k$ ale mulțimii $\{1, 2, \dots, n\}$. De fapt, un element al stivei are doar două componente, elementul minim și respectiv elementul maxim din subsecvență.

În general, pentru a adăuga o substanță s la stiva S , se verifică dacă substanța din vârful stivei S formează o nouă substanță împreună cu s , cu alte cuvinte dacă reuniunea celor două seturi de valori este o substanță (conform regulii). În caz afirmativ, se extrage substanța din vârful stivei S , se formează noua substanță, apoi se repetă pașii anteriori (verificare, extragere, formare nouă substanță). Dacă reuniunea a două seturi de valori (substanțe) nu formează o nouă substanță, atunci ultima nouă substanță obținută se pune în vârful stivei S .

Astfel, setul de substanțe este parcurs o singură dată; se consideră inițial fiecare element ca fiind o substanță separată și se adaugă fiecare dintre aceste substanțe la S , în modul descris mai sus.

Dacă la terminarea parcurgerii setului de valori, stiva S conține un singur element (o singură substanță), adică valorile 1 și n , substanța finală este stabilă, în caz contrar nu.

Pentru valorile 2, 1, 4, 5, 3 și notațiile tradiționale

Push e - pune în vârful stivei elementul e

Pop e - scoate din vârful stivei elementul e

Add e - adaugă elementul e la stivă conform regulilor descrise

procesul arată astfel:

Add 2 : Push 2 Stack : (2,2)

```
Add 1 : Pop 2, [Add (2,1) : Push (2,1)] Stack : (2,1)
Add 4 : Push 4 Stack : (4)(2,1)
Add 5 : Pop 4, [Add (4,5) : Push (4,5)] Stack : (4,5)(2,1)
Add 3 : Pop (4,5), [Add (4,5,3) : Pop (2,1), Add (2,1,4,5,3) : Push (2,1,4,5,3)] Stack : (1,5)
```

Modalitatea de verificare a obținerii unei substanțe stabile poate să fie unică. De exemplu pentru substanțele 4, 5, 3, 1, 2 pot exista două modalități de combinare:

$((4, 5), 3), (1, 2))$
a) $((4, 5), (3, (1, 2)))$

Soluția indicată mai sus validează varianta a).

Observație

Funcție de implementare algoritmul poate utiliza mai multă sau mai puțină memorie.

Dacă verificarea posibilității de combinare se face utilizând *vector characteristic* și în stivă se reține întreaga substanță, se utilizează *memorie multă* și timpul este mai mare.

Dacă fiecare element este reținut pe stivă doar prin două valori (*min* și *max* numerelor de ordine a substanțelor care o compun), memoria utilizată este mult mai mică și, de asemenea, timpul de verificare a posibilității de combinare este foarte mic.

Soluția2 - divide & impera

Se consideră că substanța finală este stabilă și se încearcă (utilizând *divide & impera*) să se detecteze cele două substanțe care o compun. Apoi, recursiv, pentru fiecare substanță detectată se repetă proceful. Dacă în final s-au obținut n substanțe distincte, substanța inițială a fost stabilă, iar dacă, la un anumit pas, nu s-au putut separa, pentru o numită substanță, cele două componente, substanța inițială nu a fost stabilă. Funcție de implementări și de optimizări de pot obține între 40-60% din punctaj.

Soluția3 - brut

Se caută două substanțe care pot reacționa, se reunesc, apoi se reia procesul de la început. Funcție de implementare se poate obține până la 50% din punctaj.

29.2.2 *Rezolvare detaliată

29.2.3 Cod sursă

Listing 29.2.1: reactii_50.cpp

```

1 // Buruiana Filip
2 // simulare pas cu pas a proceului - 50 de puncte
3
4 #include <stdio.h>
5
6 #define NMax 15024
7
8 typedef struct { int lo, hi; } interval;
9
10 int N, T, x;
11 interval Valori[NMax];
12
13 int main()
14 {
15     int i, j;
16
17     freopen("reactii.in", "r", stdin);
18     freopen("reactii.out", "w", stdout);
19
20     scanf("%d %d", &N, &T);
21     for ( ; T; --T)
22     {
23         for (i = 1; i <= N; ++i)
24         {
25             scanf("%d", &x);
26             Valori[i].lo = Valori[i].hi = x;
27         }
28     }
29 }
```

```

28     j = N;
29     int stop = 0;
30     for (; !stop;)
31     {
32         stop = 1;
33         for (i = 1; i < j; ++i)
34         {
35             // se poate uni i si i+1?
36             if (Valori[i].hi + 1 == Valori[i+1].lo)
37             {
38                 Valori[i].hi = Valori[i+1].hi;
39                 for (++i; i < j; ++i)
40                     Valori[i] = Valori[i+1];
41                 --j;
42                 stop = 0;
43                 break;
44             }
45             else if (Valori[i+1].hi + 1 == Valori[i].lo)
46             {
47                 Valori[i].lo = Valori[i+1].lo;
48                 for (++i; i < j; ++i)
49                     Valori[i] = Valori[i+1];
50                 --j;
51                 stop = 0;
52                 break;
53             }
54         }
55     }
56     printf("%d\n", (j == 1));
57 }
58
59 return 0;
60 }
```

Listing 29.2.2: reactii_DI.cpp

```

1 //F. Buruiana
2 #include <stdio.h>
3
4 #define NMax 15024
5 typedef struct { int lo, hi; } interval;
6
7 int N, T, k;
8 interval st[NMax];
9
10 // returneaza 1 daca si numai daca ultimele
11 // doua intervale din stiva pot fi unite, caz in care le si uneste
12 int uneste()
13 {
14     if (st[k-1].hi + 1 == st[k].lo)
15     {
16         st[k-1].hi = st[k].hi;
17         --k;
18         return 1;
19     }
20     else if (st[k].hi + 1 == st[k-1].lo)
21     {
22         st[k-1].lo = st[k].lo;
23         --k;
24         return 1;
25     }
26     return 0;
27 }
28
29 int main()
30 {
31     int i, x;
32
33     freopen("molecule.in", "r", stdin);
34     freopen("molecule.out", "w", stdout);
35
36     scanf("%d %d", &N, &T);
37     for (; T; --T)
38     {
39         k = 0;
```

```

40     for (i = 1; i <= N; ++i)
41     {
42         scanf("%d", &x);
43         ++k; st[k].lo = x; st[k].hi = x;
44         for (; k > 1 && uneste(); );
45     }
46     printf("%d\n", (k == 1));
47 }
48
49 return 0;
50 }
```

29.3 text

Problema 3 - text

100 de puncte

Dintr-o regretabilă eroare, redactorul Vasile a șters toate spațiile din textul la care lucra.

Textul este scris într-o limbă necunoscută, numai cu litere mici ale alfabetului englez. Vasile știe că un cuvânt trebuie să conțină cel puțin o vocală și că nu poate avea lungimea mai mare de 20 de litere. De asemenea, fiind un tipmeticulos, el știe că în text erau (înainte de ștergerea spațiilor) exact N cuvinte.

Vasile trebuie să restaureze textul, inserând spații între cuvinte. Cum există numeroase modalități de restaurare a textului, Vasile a hotărât să aleagă varianta în care literele sunt distribuite în cuvinte într-un mod cât mai armonios. Pentru a măsura armonia, Vasile a calculat suma pătratelor lungimilor cuvintelor. Textul este cu atât mai armonios, cu cât suma obținută este mai mică.

Cerințe

Dat fiind textul fără spații, să se determine câte posibilități de restaurare există (în total, indiferent de armonia lor), precum și cea mai armonioasă modalitate de restaurare.

Date de intrare

Fișierul de intrare **text.in** conține pe prima linie textul fără spații. Pe cea de a doua linie este scris numărul natural N , reprezentând numărul de cuvinte din textul inițial.

Date de ieșire

Fișierul de ieșire **text.out** va conține pe prima linie un număr natural reprezentând numărul total de posibilități de restaurare modulo 1 000 003 (restul împărțirii la 1 000 003). Pe cea de-a doua linie va fi scrisă măsura armoniei textului restaurat (suma minimă a pătratelor lungimilor cuvintelor din text). Pe a treia linie va fi scris cel mai armonios text obținut după restaurare. Între orice două cuvinte consecutive va fi scris un singur spațiu.

Restricții și precizări

- $0 < \text{Lungimea sirului} \leq 200$
- Vocalele alfabetului englez sunt 'a', 'e', 'i', 'o', 'u', 'y'.
- Pentru datele de test există întotdeauna soluție.
- Dacă există mai multe soluții optime de restaurare, va fi scrisă prima variantă în ordine lexicografică (se știe că ' $<$ ' este ' $<$ ').
- sirul (x_1, x_2, \dots, x_n) este mai mic lexicografic decât (y_1, y_2, \dots, y_n) dacă există k ($1 \leq k \leq n$) astfel încât $x_i = y_i$ (pentru orice $1 \leq i < k$) și $x_k < y_k$.
 - Pentru 40% dintre teste lungimea textului este < 70 și $N \leq 7$.
 - Punctajul pe test se va acorda astfel: 50% pentru numărul total de modalități de restaurare modulo 1 000 003; 80% pentru numărul de modalități de restaurare modulo 1 000 003 și suma minimă; 100% pentru rezolvarea corectă a tuturor cerințelor.

Exemple:

text.in	text.out	Explicații
bcaeiuxtz 3	6 34 bca eio uxtz	Possibilitățile de restaurare sunt: bca eio uxtz (armonie: $9+9+16=34$) bca ei ouxtz (armonie: $9+4+25=38$) bca e iouxtz (armonie: $9+1+36=46$) bcae io uxtz (armonie: $16+4+16=36$) bcae i ouxtz (armonie: $16+1+25=42$) bcaei o uxtz (armonie: $25+1+16=42$) Varianta cea mai armonioasă este bca eio uxtz

Timp maxim de executare/test: **0.4** secunde pe Windows, **0.1** secunde pe Linux

Memorie: total **2 MB** din care pentru stivă **1 MB**

29.3.1 Indicații de rezolvare

prof. Emanuela Cerchez, Liceul de Informatică "Grigore Moisil" Iași

Reprezentarea informațiilor:

Vom memora sirul dat în variabila s

Vom nota cu lg lungimea sirului citit.

Vom utiliza un vector poz : $poz[i]$ = poziția celei mai apropiate vocale care urmează după poziția i ; (sau -1 dacă nu mai urmează nici o vocală).

Idee 1.

Vom rezolva problema prin metoda *backtracking*.

Vom genera vectorul sol cu $n - 1$ componente; $sol[i]$ = poziția de început al celui de-al i -lea cuvânt.

Considerăm că numerotăm cuvintele de la 0 (cuvântul 0 începe obligatoriu pe poziția 0).

În vectorul $solmin$ memorăm soluția optimă, iar în variabila nr contorizăm soluțiile.

Când apelăm *funcția recursivă rezolva(k)* sunt fixate deja începuturile cuvintelor 0, 1, ..., $k - 1$. Trebuie să fixăm începutul celui de-al k -lea cuvânt.

În acest scop, vom fixa începutul astfel încât cuvântul precedent să conțină cel puțin o vocală, să mai existe vocale disponibile și pentru cuvântul curent, iar lungimea cuvântului să nu depășească 20 de caractere.

Pe parcursul generării calculăm și costul soluției deja construite.

Când parametrul $k == n$ deducem că soluția este completă.

Verificăm dacă este validă (ultimul cuvânt are cel mult 20 de litere) și în caz afirmativ o contorizăm.

Apoi comparăm costul soluției curente cu costul minim și în cazul în care este mai mic reținem această soluție drept soluție optimă.

O astfel de abordare obține 40 puncte.

Idee 2.

Cerința 1.

Vom aborda problema prin *programare dinamică*.

Subproblemă:

Să se determine numărul de modalități de a construi k cuvinte din sufixul sirului care începe la poziția i ($0 < i < Lg$; $1 \leq K \leq N$).

Pentru a aceasta vom construi un cuvânt de maxim 20 de caractere care începe pe poziția i (în toate modurile posibile).

$nr[i][k] = \sum nr[i + t][k - 1]$, unde $1 \leq t \leq 20$ (cel mult 20 de litere) și $i + t > poz[i]$ (trebuie să conțină o vocală cără).

Evident, nu însumăm valorile -1 (care indică imposibilitatea de construire a unei soluții).

Solutia problemei se va afla în $nr[0][N]$

Observăm că pentru a determina valorile nr nu este necesară toata matricea ci doar două coloane succesive (coloana curentă și coloana precedentă).

Cerința 2.

Vom rezolva și această cerință prin *programare dinamică*.

Subproblemă:

Să se determine cea mai armonioasă împărțire în k cuvinte a *sufixului* sirului s care începe la poziția i .

Vom memora în

$cost[i][k]$ = costul împărțirii (suma diferențelor absolute ale lungimilor cuvintelor consecutive) - sau Infinit dacă o astfel de împărțire nu este posibilă

$urm[i][k]$ = poziția de început a următorului cuvânt (sau lg dacă un astfel de cuvânt nu mai există)

Modul de gândire este asemănător cu cel de la cerința 1, numai că în loc de a însuma numărul de posibilități, vom determina un minim:

$cost[i][k] = \min t^*t + cost[i+t][k-1]$, unde $1 \leq t \leq 20$ (cel mult 20 de litere) și $i + t > poz[i]$

$urm[i][k] = i + t_{\min}$ (t_{\min} fiind valoarea lui t pentru care se obține minimul în relația precedentă).

Din relația de recurență a costului deducem că nu este necesară memorarea întregii matrice $cost$, sunt necesare doar 2 coloane. În schimb, pentru *reconstituirea* soluției este necesară întreaga matrice urm .

29.3.2 *Rezolvare detaliată

29.3.3 Cod sursă

Listing 29.3.1: text_bkt.cpp

```

1 //Emanuela Cerchez 40 puncte
2 #include <iostream.h>
3 #include <string.h>
4
5 #define InFile "text.in"
6 #define OutFile "text.out"
7 #define abs(x) ((x)>0?(x):(-(x)))
8 #define LgMax 202
9 #define R 1000003
10 #define Inf 2000000000
11
12 char s[LgMax], v[]="aeiouy";
13 int n, lg;
14 int poz[LgMax];
15 //poz[i]=pozitia celei mai apropiate vocale >=i
16 int sol[LgMax], solmin[LgMax];
17 long int nr;
18 long int costmin, cost;
19
20
21 void citire ();
22 void rezolva (int);
23 void afisare ();
24 void det_poz();
25
26 int main ()
27 {
28     int i, j, k;
29     ofstream fout (OutFile);
30     citire ();
31     det_poz();
32     costmin=Inf;
33     rezolva(1);
34     fout<<nr<<'<br>';
35     fout<<costmin<<'<br>';
36
37     for (i=0, k=1; k<n; k++)
38         { for (j=i; j<solmin[k]; j++) fout<<s[j];
39             i=solmin[k];
40             fout<<'<br>';
41         }
42     for (j=solmin[n-1]; j<lg; j++) fout<<s[j];
43     fout<<'<br>';
44     fout.close ();
45     return 0;

```

```

46 }
47
48 void citire ()
49 {
50     ifstream fin (InFile);
51     fin.getline(s, LgMax);
52     lg=strlen(s);
53     fin>>n;
54     fin.close ();
55 }
56
57 void det_poz()
58 {
59     int i, last=-1;
60     for (i=lg-1; i>=0; i--)
61         {if (strchr(v,s[i]))last=i;
62          poz[i]=last;}
63 }
64
65 void rezolva (int k)
//pozitia la care incepe cel de-al k-lea cuvant
66 {
67     int t;
68     if (k==n)
69     {
70         if (lg-sol[n-1]<=20)
71         {
72             //am determinat o solutie
73             nr=(nr+1)%R;
74             if (cost+(lg-sol[k-1])*(lg-sol[k-1])<costmin)
75             {
76                 costmin=cost+(lg-sol[k-1])*(lg-sol[k-1]);
77                 for (t=1; t<n; t++) solmin[t]=sol[t];
78             }
79         }
80     }
81     else
82     for (t=1; t<=20 && sol[k-1]+t<lg; t++)
83         if (poz[sol[k-1]]<sol[k-1]+t && //exista o vocala in cuvantul precedent
84             poz[sol[k-1]+t]!=-1)           //mai exista vocala in continuare
85         {
86             sol[k]=sol[k-1]+t;
87             cost+=t*t;
88             rezolva(k+1);
89             cost-=t*t;
90         }
91     }
92 }
```

Listing 29.3.2: text_em.cpp

```

1 //Emanuela Cerchez - 100 puncte
2 #include <iostream>
3 #include <string.h>
4
5 using namespace std;
6
7 #define InFile "text.in"
8 #define OutFile "text.out"
9 #define abs(x) ((x)>0?(x):(-(x)))
10 #define LgMax 210
11 #define Inf 2000000000
12 #define R 1000003
13
14 char s[LgMax], v[]{"aeiouy"};
15 int n, lg, cine;
16 int poz[LgMax];
//poz[i]=pozitia celei mai apropiate vocale >=i
17
18 long int nr[LgMax][2];
//nr[i][k]=nr de posibilitati de a construi k cuvinte incepand cu pozitia i
19
20 long int cost[LgMax][2];
21 unsigned char urm[LgMax][LgMax];
22
23 void citire ();
```

```

26 void rezolvare ();
27 void afisare ();
28 void det_poz();
29 void numarare();
30 void armonie();
31
32 int main ()
33 {
34 int i, j, k;
35 ofstream fout (OutFile);
36 citire ();
37 det_poz();
38 numarare ();
39
40 fout<<nr[0][cine]<<' \n';
41
42 armonie();
43 fout<<cost[0][cine]<<' \n';
44
45 for (i=0, k=n; k>0; k--)
46 { for (j=i; j<urm[i][k]; j++) fout<<s[j];
47   i=urm[i][k];
48   if (k>1) fout<<' ';
49 }
50 fout<<' \n';
51 fout.close ();
52 return 0;
53 }
54
55 void citire ()
56 {
57 ifstream fin (InFile);
58 fin.getline(s, LgMax);
59 lg=strlen(s);
60 fin>>n;
61 fin.close ();
62 }
63
64 void det_poz()
65 {
66 int i, last=-1;
67 for (i=lg-1; i>=0; i--)
68 { if (strchr(v,s[i])) last=i;
69   poz[i]=last; }
70 }
71
72 void numarare ()
73 {
74 int i, k, t, ok;
75 long int sum;
76 cine=0;
77 for (i=0; i<lg; i++)
78 { if (poz[i]!=-1 && lg-i<=20) nr[i][0]=1;
79   else
80     nr[i][0]=-1;
81
82 for (k=2; k<=n; k++)
83 {
84   cine=1-cine;
85   for (i=lg-1; i>=0; i--)
86   {
87     nr[i][cine]=-1;
88     sum=0; ok=0;
89     for (t=1; t<=20 && i+t-1<lg; t++)
90       { if (poz[i]==-1 || i+t-1 <poz[i]) continue;
91         if (nr[i+t][1-cine]==-1)
92           { ok=1; sum=(sum+nr[i+t][1-cine])%R; }
93       }
94     if (ok) nr[i][cine]=sum;
95   }
96 }
97 }
98
99 void armonie ()
100 {
101 int i, k, t, tmin;

```

```

102 long int min;
103
104 cine=0;
105 for (i=0; i<lg; i++)
106 {
107     if (poz[i]!=-1 && lg-i<=20) {cost[i][0]=(lg-i)*(lg-i); urm[i][1]=lg;}
108     else
109         {cost[i][0]=Inf; urm[i][1]=0;}
110 }
111 for (i=0; i<=lg; i++) urm[lg][i]=0;
112 cost[lg][0]=cost[lg][1]=Inf;
113
114 for (k=2; k<=n; k++)
115 {
116     cine=1-cine;
117     for (i=lg-1; i>=0; i--)
118     {
119         cost[i][cine]=Inf; urm[i][k]=0;
120         min=Inf; tmin=-1;
121
122         for (t=1; t<=20 && i+t-1<lg; t++)
123             {if (poz[i]==-1 || i+t-1 < poz[i]) continue;
124              if (cost[i+t][1-cine]==Inf) continue;
125
126              if (cost[i+t][1-cine]+t*t<min)
127                  {
128                      min=cost[i+t][1-cine]+t*t;
129                      tmin=t;
130                  }
131
132             if (tmin!=-1) {cost[i][cine]=min; urm[i][k]=i+tmin;}
133         }
134     }
135 }
136
137 }

```

Listing 29.3.3: text_mia.cpp

```

1 // Mugurel Ionut Andreica
2 // Time Complexity: O(lungime sir * numar cuvinte)
3 // nu depinde de lungime maxima a unui cuvant
4
5 #include <stdio.h>
6 #include <string.h>
7
8 #define LMAX 210
9 #define NMAX 210
10 #define LCUV 20
11 #define SIGMA 256
12 #define PMOD 1000003
13 #define INF 65000
14
15 char sir[LMAX], spaceAfter[LMAX], vocale[SIGMA];
16 long int nrpos[LMAX][2], sprefix[LMAX];
17 long int cmin[LMAX][2];
18 unsigned char prev[LMAX][NMAX];
19 long int i, j, k, N, M, lastVoc, prevLastVoc, sum, li, ls;
20 long int ydq[LMAX], wdq[LMAX], pozdq[LMAX], pozfirst[LMAX];
21 long int y, w, pozf, ykdq, dy, dx, xcross;
22
23 void nr_posib()
24 {
25     nrpos[0][0] = 1;
26     for (i = 1; i <= M; i++)
27         nrpos[i][0] = 0;
28
29     for (j = 1; j <= N; j++)
30     {
31         sprefix[0] = nrpos[0][(j - 1) % 2];
32         for (i = 1; i <= M; i++)
33             sprefix[i] = (sprefix[i - 1] + nrpos[i][(j - 1) % 2]) % PMOD;
34
35     lastVoc = -LCUV - 1;
36 }

```



```

113                         }
114                     }
115                 }
116             ls++;
117             ydq[ls] = y;
118             wdq[ls] = w;
119             pozdq[ls] = k;
120             pozfirst[ls] = pozf;
121         }
122     }
123 }
124
125 if (lastVoc >= i - LCUV + 1)
126 {
127     // clean up the front of the deque
128     while (li <= ls && pozdq[li] < i - LCUV)
129         li++;
130
131     while (li < ls && pozfirst[li + 1] <= i)
132         li++;
133
134     if (li > ls)
135     {
136         cmin[i][j % 2] = INF;
137         prev[i][j] = 0;
138     }
139     else
140     {
141         cmin[i][j % 2] = i*i + ydq[li] + wdq[li] * (i-pozdq[li]);
142         prev[i][j] = pozdq[li];
143     }
144 }
145 else
146     cmin[i][j % 2] = INF;
147 }
148 }
149 printf("%ld\n", cmin[M][N % 2]);
150
151 // reconstituirer
152 i = M; j = N;
153
154 memset(spaceAfter, 0, sizeof(spaceAfter));
155
156 while (i >= 1 && j >= 1)
157 {
158     spaceAfter[prev[i][j]] = 1;
159     i = prev[i][j];
160     j--;
161 }
162
163 for (i = 1; i <= M; i++)
164 {
165     printf("%c", sir[i]);
166     if (spaceAfter[i])
167         printf(" ");
168 }
169
170 printf("\n");
171
172 }
173
174 int main()
175 {
176     memset(vocale, 0, sizeof(vocale));
177     vocale['a'] = 1;
178     vocale['e'] = 1;
179     vocale['i'] = 1;
180     vocale['o'] = 1;
181     vocale['u'] = 1;
182     vocale['y'] = 1;
183
184     freopen("text.in", "r", stdin);
185     scanf("%s %ld", sir + 1, &N);
186     M = strlen(sir + 1);
187
188     freopen("text.out", "w", stdout);

```

```

189     nr_posib();
190     armonie_optima();
191
192     return 0;
193 }
```

29.4 bile

Problema 4 - bile

100 de puncte

N prieteni stau în jurul a N urne cu bile. Prietenii și urnele sunt numerotate cu numere de la 0 la $N - 1$. Fiecare urnă i ($0 \leq i \leq N - 1$) conține S_i bile. Prietenii doresc să extragă bile din urne și să le pună în buzunare. Datorită așezării, din urna i pot extrage bile doar prietenii i și $((i + 1) \bmod N)$. Fiecare prieten i ($0 \leq i \leq N - 1$) are o capacitate a buzunarelor de P_i bile (adică nu poate extrage mai mult de P_i bile în total). Prietenul 0 este liderul lor și își pune întrebări de forma: dacă eu extrag exact x bile din urna 0, atunci care este numărul maxim de bile pe care le pot extrage în total toți prietenii, folosind o strategie adecvată și considerând limitările problemei? O strategie adecvată determină câte bile extrage fiecare prieten din fiecare urnă din care poate extrage bile, considerând că prietenul 0 extrage neapărat x bile din urna 0.

Cerințe

Pentru fiecare valoare posibilă a lui x (de la 0 la $\min\{S_0, P_0\}$), determinați numărul maxim de bile pe care le pot extrage cei N prieteni în total din cele N urne.

Date de intrare

Prima linie a fișierului **bile.in** conține numărul de teste T descrise în continuare. Prima linie a fiecarui test conține numărul N de prieteni. Urmează apoi N linii. Linia i dintre acestea ($0 \leq i \leq N - 1$) conține numerele întregi S_i și P_i , separate printr-un spațiu.

Date de ieșire

În fișierul de ieșire **bile.out** veți afișa răspunsurile pentru fiecare test, în ordinea în care acestea sunt date în fișierul de intrare. Pentru fiecare valoare posibilă a lui x (considerând ordinea crescătoare a valorilor) pentru testul respectiv, veți afișa o linie conținând numărul maxim total de bile pe care îl pot extrage prietenii din urne.

Restricții și precizări

- $1 \leq T \leq 10$
- $2 \leq N \leq 15\ 000$
- $1 \leq S_i \leq 16\ 000$
- $1 \leq P_i \leq 16\ 000$
- 40% dintre teste au $N \leq 500$ și $\max\{S_i, P_i\} \leq 500$ ($0 \leq i \leq N - 1$)
- $A \bmod B$ reprezintă restul împărțirii întregi a lui A la B .
- O bilă poate fi extrasă o singură dată, de un singur prieten.

Exemple:

bile.in	bile.out	Explicații
2	17	Pentru primul test x poate lua valuri între 0 și $\min\{S_0, P_0\} = 5$.
4	18	Pentru $x=0$, cei 4 prieteni pot extrage în total 17 bile.
5 5	19	Pentru $x=1$, cei 4 prieteni pot extrage în total 18 bile.
3 4	19	Pentru $x=2$, cei 4 prieteni pot extrage în total 19 bile.
8 6	19	Pentru $x=3$, cei 4 prieteni pot extrage în total 19 bile.
3 4	18	Pentru $x=4$, cei 4 prieteni pot extrage în total 19 bile.
4	13	Pentru $x=5$, cei 4 prieteni pot extrage în total 18 bile.
2 3	13	Valoarea lui x a fost inclusă de fiecare dată în numărul total de bile ce pot fi extrase de cei 4 prieteni.
6 5	12	
2 4		
3 1		

Timp maxim de executare/test: **1.2** secunde pe Windows, **0.4** secunde pe Linux

Memorie: total **2 MB** din care pentru stivă **1 MB**

29.4.1 Indicații de rezolvare

Mugurel Ionuț Andreica, Universitatea Politehnica București

Pentru o valoare fixată a lui x , următorul algoritm de tip *greedy* determină numărul maxim total de bile extrase în timp liniar ($O(N)$).

Prietenul 0 extrage x bile din urna 0. Restul de bile din urna 0 pot fi extrase doar de către prietenul 1, astfel că acesta va extrage cât mai multe dintre aceste bile. Vom parcurge apoi celelalte urne în ordine, de la 1 la $N - 1$, și pentru fiecare vom proceda după cum urmează: prietenul i va extrage cât mai multe bile posibile din urna i (în limita numărului de bile rămase în urna i și în limita capacitatii buzunarelui prietenului i). Restul de bile rămase în urna i vor fi extrase de prietenul $((i + 1)modN)$ (din nou, în limita capacitatii buzunarelui acestuia).

Vom nota prin $f(x)$ numărul maxim de bile ce pot fi extrase de toți prietenii, dacă prietenul 0 extrage exact x bile din urna 0.

Întrucât $f(x)$ poate fi calculat în timp $O(N)$ folosind algoritmul descris anterior, am obținut deja o soluție de complexitate $O(N \cdot XMAX)$, unde $XMAX$ este numărul de valori pe care le poate lua x (x ia valori de la 0 la $xm = \min\{S0, P0\}$). Această soluție ar trebui să obțină aproximativ 40 puncte.

Pentru a obține punctajul maxim, trebuie observat că graficul funcției $f(x)$ are o formă particulară. Fie $y1 = f(0)$ și $y2 = f(xm)$. Pe intervalul $[0, A]$ valoarea funcției crește cu câte 1 unitate (față de valoarea anterioară); pe intervalul $[A, B]$ funcția are valori constante, iar pe intervalul $[B, xm]$, valoarea funcției scade cu câte 1 unitate (față de valoarea anterioară). Demonstrația acestui fapt nu este complicată și nu folosește elemente care depășesc nivelul de cunoștințe al clasei a X-a.

Așadar, problema se reduce la a determina în mod eficient valorile A și B (de menționat că oricare din intervalele $[0, A]$, $[A, B]$ sau $[B, xm]$ pot avea lungime 0).

O modalitate simplă de a calcula aceste valori este să calculăm acea valoare xs , care are proprietatea că $y1 + xs = y2 + (xm - xs)$ (adică acea valoare xs ce ar corespunde cazului în care lungimea intervalului $[A, B]$ ar fi 0).

Vom calcula $ys = f(xs)$, iar apoi vom calcula diferența $d = y1 + xs - ys$.

Valorile lui A și B sunt $xs - d$ și $xs + d$.

Mai există încă un caz, pentru situația în care xs nu este un număr întreg, care se tratează în mod similar.

În ambele situații, valorile A și B se pot determina folosind un număr constant ($O(1)$) de evaluări ale funcției f .

Odată ce am determinat valorile A și B , putem calcula foarte simplu valoarea $f(x)$ pentru fiecare valoare a lui x .

Așadar, complexitatea soluției optime este $O(N + XMAX)$.

29.4.2 *Rezolvare detaliată

29.4.3 Cod sursă

Listing 29.4.1: bile.cpp

```

1 //Mugurel Andreica 100 puncte
2 #include <stdio.h>
3
4 #define NMAX 15001
5
6 #define min(a, b) ((a) < (b) ? (a) : (b))
7
8 short int P[NMAX], S[NMAX], Pgot[2], Sleft[2];
9 long int i, j, k, N, x, xmax, sumx, y1, y2, x1, x2, d1, d2, yx1, yx2, T;
10
11 long int f(int x)
12 {
13     Pgot[0] = x;
14     Sleft[0] = S[0] - x;

```

```

15     sumx = x;
16
17     for (i = 0; i < N; i++)
18     {
19         if (i > 0)
20         {
21             Sleft[i % 2] = S[i];
22             k = min(Sleft[i % 2], P[i] - Pgout[i % 2]);
23             Sleft[i % 2] = S[i] - k;
24             Pgout[i % 2] += k;
25             sumx += k;
26         }
27
28         if (i < N - 1)
29             k = min(Sleft[i % 2], P[(i + 1) % N]);
30         else
31             k = min(Sleft[i % 2], P[(i + 1) % N] - x);
32
33         Sleft[i % 2] -= k;
34         Pgout[((i + 1) % N) % 2] = k;
35         sumx += k;
36     }
37
38     return sumx;
39 }
40
41 int main()
42 {
43     freopen("bile.in", "r", stdin);
44     freopen("bile.out", "w", stdout);
45
46     scanf("%ld", &T);
47
48     while (T--)
49     {
50         scanf("%ld", &N);
51
52         for (i = 0; i < N; i++)
53             scanf("%d %d", &S[i], &P[i]);
54
55         xmax = min(P[0], S[0]);
56
57         y1 = f(0);
58         y2 = f(xmax);
59         x1 = (y2 - y1 + xmax) / 2;
60         yx1 = f(x1);
61
62         if ((y2 - y1 + xmax) % 2 == 1)
63         {
64             x2 = x1 + 1;
65             yx2 = f(x2);
66         }
67         else
68         {
69             x2 = x1;
70             yx2 = yx1;
71         }
72
73         d1 = y1 + x1 - yx1;
74         d2 = y2 + (xmax - x2) - yx2;
75
76         fprintf(stderr, "0 (y=%ld) => +1 ; \
77             plateau: %d - %d (y=%ld); -1 => xmax=%d (y=%ld); %d\n",
78             y1, x1 - d1, x2 + d2, yx1, xmax, y2, (y2 > y1));
79
80         for (x = 0; x <= x1 - d1; x++)
81             printf("%ld\n", y1 + x);
82
83         for (x = x1 - d1 + 1; x <= x2 + d2; x++)
84             printf("%ld\n", yx1);
85
86         for (x = x2 + d2 + 1; x <= xmax; x++)
87             printf("%ld\n", y2 + (xmax - x));
88     }
89
90     return 0;

```

29.5 check-in

Problema 5 - check-in

100 de puncte

Ministerul organizează o excursie pentru olimpici la Paris. Suntem toți la aeroport, K olimpici având în total P bagaje. Olimpicii la informatică trebuie să rezolve acum următoarea problemă.

Pentru zborul către Paris au fost deschise N ghișee pentru check-in, numerotate de la 1 la N . La fiecare ghișeu lucrează exact un angajat. Angajatul de la ghișeu i are nevoie de A_i secunde pentru a procesa fiecare bagaj al clientului prezentat la ghișeu și B_i secunde pentru a emite toate biletelor de îmbarcare solicitate de client (același timp B_i , indiferent de numărul de bilete solicitate de client).

O persoană poate sta la un singur ghișeu și poate preda 0, 1 sau mai multe bagaje (acestea vor fi trecute pe numele său). Evident, aceeași persoană nu poate sta la mai multe ghișee. De asemenea, o persoană poate să prezinte angajatului de la ghișeu biletele și pașapoartele altor persoane, astfel că poate solicita emiterea mai multor bilete de îmbarcare. O persoană trebuie să solicite de la ghișeu la care se prezintă cel puțin un bilet de îmbarcare.

Înțelegem că nu există coadă la ghișeele pentru Paris. Timpul necesar pentru a face check-in-ul (predarea tuturor celor P bagaje și obținerea biletelor de îmbarcare pentru toți cei K olimpici) depinde de strategia adoptată (alegerea ghișeelor, stabilirea persoanelor care stau la coadă la ghișee și împărțirea bagajelor). Olimpicii la informatică trebuie să găsească o strategie prin care cei K olimpici pot primi cele P bagaje și obține cele K bilete de îmbarcare în cel mai scurt timp.

Cerințe

Scrieți un program care să determine timpul minim necesar pentru check-in.

Date de intrare

Fișierul de intrare **checkin.in** conține pe prima linie numărul natural N reprezentând numărul de ghișee pentru check-in. Urmează N linii pe care sunt descrise ghișeele. Pe linia $i + 1$ sunt scrise numerele naturale A_i B_i (separate prin spațiu) reprezentând timpul necesar angajatului de la ghișeu i pentru a procesa un singur bagaj al clientului de la ghișeu, respectiv timpul necesar pentru emiterea tuturor biletelor de îmbarcare solicitate de clientul de la ghișeu. Pe ultima linie sunt scrise numerele naturale K și P , separate prin spațiu, reprezentând numărul de olimpici și numărul de bagaje pe care le au.

Date de ieșire

Fișierul de ieșire **checkin.out** va conține o singură linie pe care va fi scris timpul minim pentru check-in.

Restricții și precizări

- $1 \leq N \leq 1\,000$
- $1 \leq A_i, B_i \leq 1\,000$
- $1 \leq K \leq 10\,000$
- $0 \leq P \leq 10\,000$

Exemplu:

checkin.in	checkin.out	Explicații
6		O persoană stă la ghișeu 3, predă un bagaj și ia un bilet de îmbarcare.
10 100	70	
20 80		O a doua persoană stă la ghișeu 5, predă 3 bagaje și ia un bilet de îmbarcare.
20 40		
40 50		O a treia persoană stă la ghișeu 6, predă 6 bagaje și ia 2 bilete de îmbarcare.
20 10		
10 10		A patra persoană nu stă la ghișeu.
4 10		

Timp maxim de executare/test: **0.5** secunde

Memorie: total **2 MB** din care pentru stivă **2 MB**

Dimensiune maximă a sursei: **15 KB**

29.5.1 Indicații de rezolvare

prof. Emanuela Cerchez, Liceul de Informatică "Grigore Moisil" Iași

O primă observație: la ghișeu vor sta maxim N persoane (nu are rost ca la un ghișeu să stea două persoane). Deci $K = \min\{K, N\}$.

Determinăm timpul minim prin căutare binară în intervalul $[0, TMIN]$ (unde $TMIN$ este timpul minim care s-ar obține dacă bagajele ar fi date de o singură persoană, la un singur ghișeu: $Tmin = \min\{P * a[i] + b[i] | 1 \leq i \leq N\}$

Pentru a verifica dacă check-in-ul se poate termina în timpul T :

Livrăm în ordinea ghișeelor bagajele care pot fi livrate în timpul T . Dacă nu reușim să livrăm toate bagajele în timpul T folosind primele K ghișee, mărim timpul. În caz contrar, micșorăm timpul.

Pentru fiecare verificare, sortăm ghișeile după $(T - b[i])/a[i]$. (sau determinăm K maxime) Complexitate $O(n \log N \log TMIN)$.

29.5.2 *Rezolvare detaliată

29.5.3 Cod sursă

Listing 29.5.1: checkin.cpp

```

1 //Emanuela Cerchez 100 puncte
2 #include <iostream>
3 #include <assert.h>
4
5
6 using namespace std;
7
8 #define InFile "checkin.in"
9 #define OutFile "checkin.out"
10#define NMax 1001
11
12 int n;
13 long int P, K, a[NMax], b[NMax], TMin, t, nrmax[NMax], vmax[NMax];
14
15 void citire ();
16 void sortare (int, int);
17 void afisare ();
18 int verifica();
19
20 int main ()
21 {
22     long int st, dr;
23     ofstream fout (OutFile);
24     citire ();
25     if (P)
26     {
27         st=0; dr=TMin;
28         while (st<dr)
29         {
30             t=(st+dr)/2;
31             if (!verifica()) st=t+1;
32             else dr=t;
33         }
34         TMin=st;
35     }
36     afisare();
37     return 0;
38 }
39
40 void citire ()
41 {int i;
42     ifstream fin (InFile);
43     fin>>n;

```

```

44     assert(0<n && n<=1000);
45     for (i=0; i<n; i++)
46         {fin>>a[i]>>b[i];
47         assert(0<a[i] && a[i]<=1000);
48         assert(0<b[i] && b[i]<=1000);
49     }
50     fin>>K>>P;
51     assert(0<K && K<=10000);
52     assert(0<=P && P<=10000);
53     if (K>n) K=n;
54     TMin=2000000000;
55     for (i=0; i<n; i++)
56         if (TMin>P*a[i]+b[i]) TMin=a[i]*P+b[i];
57     fin.close ();
58 }
59
60 void afisare()
61 {
62 ofstream fout(OutFile);
63 fout<<TMin<<'\\n';
64 fout.close ();
65 }
66
67 int verifica()
68 {
69 //cout<<"verificare "<<t<<'\n';
70 long sum, i, h, j;
71 for (i=0; i<n; i++)
72 {
73     nrmax[i]=(t-b[i])/a[i];
74     if (nrmax[i]<=0) nrmax[i]=-1;
75 }
76 //for (i=0; i<n; i++) cout<<nrmax[i]<<' ';
77 //cout<<'\\n';
78 for (i=0; i<K; i++) vmax[i]=-1;
79 for (i=0; i<n; i++)
80     for (j=0; j<K; j++)
81         if (nrmax[i]<=vmax[j]) continue;
82         else
83         {
84             for (h=K; h>j; h--) vmax[h]=vmax[h-1];
85             vmax[j]=nrmax[i];
86             break;
87         }
88 //for (i=0; i<K; i++) cout<<max[i]<<' ';
89 //cout<<'\\n';
90
91 sum=0;
92 for (i=0; i<K; i++)
93     if (vmax[i]>0) sum+=vmax[i];
94 //cout<<"sum="<<sum<<'\\n';
95 if (sum>=P) return 1;
96 return 0;
97 }
```

29.6 volei

Problema 6 - volei

100 de puncte

La ora de sport elevii doresc să joace volei. Sunt exact N fete și N băieți și se dorește formarea a două echipe, o echipă formată numai din fete, iar cealaltă numai din băieți. Inițial, elevii au o poziție fixată pe teren. Ei pot fi reprezentați ca $2N$ puncte de coordonate întregi în plan. Pentru a putea începe jocul, profesorul de sport trebuie să traseze un fileu (o dreaptă) între echipa băieților și echipa fetelor, astfel încât cele două echipe să se afle de o parte și de alta a fileului. Cum trasarea unui fileu între cele două echipe poate să nu fie posibilă conform așezării inițiale, trebuie să fie efectuate unele schimbări. Astfel, profesorul poate alege la un moment dat o singură fată și un singur băiat, situați la distanță maxim D și să le interschimbe pozițiile (fata va trece în locul băiatului, iar băiatul în locul fetei).

Cerințe

Să se determine numărul minim de mutări pe care trebuie să le efectueze profesorul de sport pentru a fi posibilă amplasarea unui fileu, precum și mutările efectuate.

Date de intrare

Fișierul de intrare **volei.in** conține pe prima linie N , numărul de fete, egal cu numărul de băieți. A doua linie va conține numărul natural nenul D , cu semnificația din enunț. Fiecare dintre următoarele $2N$ linii va conține un triplet de forma $x \ y \ tip$, unde $x \ y$ sunt coordonatele inițiale ale unui elev (abscisa, ordonata), iar tip este caracterul B dacă elevul corespunzător este băiat sau F dacă este fată.

Date de ieșire

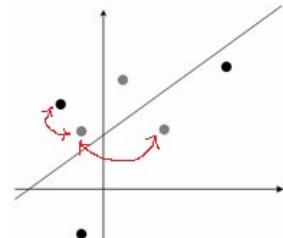
Fișierul de ieșire **volei.out** va conține pe prima linie un număr natural $nrmin$ reprezentând numărul minim de mutări care trebuie să fie efectuate. Pe următoarele $nrmin$ linii se vor afișa mutările efectuate, căte o mutare pe o linie, în ordinea efectuării lor. O mutare este descrisă sub forma $x1 \ y1 \ x2 \ y2$, cu semnificația că elevii situați la pozițiile $(x1, y1)$ și $(x2, y2)$ vor fi interschimbați.

Restricții și precizări

- $2 \leq N \leq 7$
- $1 \leq D \leq 30\ 000$
- Coordonatele sunt numere întregi din intervalul $[-10\ 000, 10\ 000]$
- Nu vor exista 3 poziții coliniare.
- Nu vor exista două persoane în aceeași poziție.
- Distanța dintre punctele $(x1, y1)$ și $(x2, y2)$ este $\sqrt{(x1 - x2)^2 + (y1 - y2)^2}$.
- Dacă sunt mai multe soluții posibile, afișați oricare dintre acestea.
- Se va acorda 40% din punctajul pe test dacă numărul de mutări este corect, respectiv 100% din punctaj dacă atât numărul mutărilor, cât și mutările efectuate sunt corecte.
- Pentru 50% dintre teste D mai mare decât maximul distanțelor între oricare două puncte.

Exemple:

volei.in	volei.out	Explicații
3	2	
4	-2 4 -1 3	
-2 4 B	-1 3 3 3	Se interschimbă băiatul de la poziția $(-2, 4)$ cu fata de la poziția $(-1, 3)$.
-1 3 F		
-1 -2 B		
1 5 F		Apoi se interschimbă băiatul de la poziția $(-1, 3)$
3 3 F		de la poziția $(3, 3)$.
6 6 B		



Timp maxim de executare/test: 0.4 secunde pe Windows, 0.2 secunde pe Linux

Memorie: total 2 MB din care pentru stivă 1 MB

29.6.1 Indicații de rezolvare

Filip Cristian Buruiana, Universitatea Politehnica Bucuresti

Pentru a putea rezolva problema este necesar mai întâi să determinăm dacă pentru o anumită amplasare a fetelor și băieților este posibilă trasarea unei drepte care să separe cele două mulțimi corespunzătoare de puncte.

Propoziție: Fie A și B două mulțimi de puncte și o dreaptă d astfel încât punctele din A să fie într-un semiplan, iar punctele din B în celalalt semiplan determinat de dreapta d (spunem că A și B sunt separate de dreapta d). Atunci există un punct din A și un punct din B astfel încât dreapta care conține cele două puncte să separe mulțimile A și B .

Demonstrația propoziției este intuitivă. Dreapta d de separație poate fi translatată până când conține un punct din A , iar apoi rotită până când conține și un punct din B .

În consecință, dacă avem două mulțimi de puncte, putem verifica în complexitate $O(N^3)$ dacă mulțimile pot fi separate, alegând toate perechile de puncte (P_1P_2), unde P_1 este în prima mulțime, iar P_2 în cea de a doua, și verificând dacă condiția este îndeplinită pentru dreapta P_1P_2 .

Amplasarea inițială a fetelor și băieților poate fi privită ca un număr în baza 2, unde un bit 0 reprezintă o fată iar un bit 1 un băiat.

Trebuie să ajungem în alta configurație cu număr minim de mutări astfel încât configurația finală să fie validă (cele două mulțimi să fie separabile). Prin mutare se înțelege selectarea a doi biți i și j de valori diferite și negarea lor (bitul 0 va deveni 1, iar bitul 1 va deveni 0, ceea ce este echivalent cu interschimbarea unei fete cu un băiat). În plus, pozițiile corespunzătoare biților interschimbați trebuie să fie la distanța euclidiană maxim D .

Pentru a determina numărul minim de mutări, pornim cu o *coadă* care conține inițial doar configurația de start. În afară de coadă, vom reține și un vector de distanțe, unde elementul al p -lea din vector reprezintă distanța de la configurația a p -a din coadă la punctul de start. La fiecare pas adăugăm în coadă toți vecinii configurației curente (configurațiile care se pot obține din configurația curentă prin exact o mutare). Procedeul se repetă până când în coadă este introdusă o configurație validă.

29.6.2 *Rezolvare detaliată

29.6.3 Cod sursă

Listing 29.6.1: volei.cpp

```

1 // Buriana Filip - 100 de puncte
2
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 #define MAX 14
7 #define MaxQueue 3432 // combinari de 7 luate cate 14
8 #define li long int
9
10 typedef struct { int x, y; } point;
11
12 int N, D;
13 unsigned char dist[1<<MAX], valid[1<<MAX];
14 int q[MaxQueue], up[MaxQueue], chx[MaxQueue], chy[MaxQueue];
15 point P[MAX];
16
17 li sqr(int X)
18 { return (li)X * X; }
19
20 int bit(int X, int nr_bit)
21 { return (X & (1<<nr_bit)) != 0; }
22
23 int canChange(int i, int j)
24 {
25     return (sqr(P[i].x-P[j].x)+sqr(P[i].y-P[j].y) <= (li)D * D);
26 }
27
28 int semn(int p1, int p2, int p3)
29 {
30     li A = P[p1].y-P[p2].y, B = P[p2].x-P[p1].x,
31         C = (li)P[p1].x * P[p2].y - (li)P[p2].x * P[p1].y;
32     li E = A * P[p3].x + B * P[p3].y + C;
33
34     if (E < 0)
35         return -1;
36     if (E > 0)
37         return +1;
38     return 0;
39 }
40
41 int separated(int config, int p1, int p2)
42 {
43     int i, semn_b = -2;
```

```

44
45     for (i = 0; i < N; ++i)
46     {
47         if (i == p1 || i == p2)
48             continue;
49         if (semn_b == -2)
50         {
51             if (bit(config, i))
52                 semn_b = semn(p1, p2, i);
53             else
54                 semn_b = -semn(p1, p2, i);
55             continue;
56         }
57         if (bit(config, i) && semn_b != semn(p1, p2, i))
58             return 0;
59         if (!bit(config, i) && semn_b == semn(p1, p2, i))
60             return 0;
61     }
62     return 1;
63 }
64
65 int path[128]; // vectorul solutie
66 void printSolution(int config, int poz)
67 {
68     printf("%d\n", dist[config]);
69
70     int c, n = 0, i1, i2;
71     for (c = poz; c != 0; c = up[c])
72         path[++n] = c;
73     for (c = n; c; --c)
74     {
75         i1 = chx[path[c]];
76         i2 = chy[path[c]];
77         printf("%d %d %d %d\n", P[i1].x, P[i1].y, P[i2].x, P[i2].y);
78     }
79     exit(0);
80 }
81
82 int main()
83 {
84     int i, j, k, nr, qr = 0, ql = 0;
85     int config = 0, newconf = 0;
86     char tip;
87
88     freopen("volei.in", "r", stdin);
89     freopen("volei.out", "w", stdout);
90
91     scanf("%d %d", &N, &D);
92     N *= 2;
93     for (i = 0; i < N; ++i)
94     {
95         scanf("%d %d %c", &P[i].x, &P[i].y, &tip);
96         if (tip == 'B')
97             config += (1<<i);
98     }
99     for (i = 0; i < (1<<N); ++i)
100         dist[i] = N*N;
101     dist[config] = 0;
102     q[ql] = config;
103
104     for (config = 0; config < (1<<N); ++config)
105     {
106         valid[config] = 0;
107         for (j = 0, nr = 0; j < N; ++j)
108             nr += bit(config, j);
109         if (nr != N/2)
110             continue;
111         for (j = 0; j < N && !valid[config]; ++j)
112             if (!bit(config, j))
113                 for (k = 0; k < N && !valid[config]; ++k)
114                     if (bit(config, k))
115                         valid[config] = separated(config, j, k);
116     }
117
118     if (valid[q[0]])
119     {

```

```
120         printf("0\n");
121     return 0;
122 }
123 for (; qr <= ql; ++qr)
124 {
125     config = q[qr];
126
127     for (j = 0; j < N; ++j)
128         if (!bit(config, j))
129         {
130             for (k = 0; k < N; ++k)
131             {
132                 if (bit(config, k) && canChange(j, k))
133                 {
134                     newconf = config-(1<<k)+(1<<j);
135                     if (dist[config] + 1 < dist[newconf])
136                     {
137                         q[++ql] = newconf;
138                         dist[newconf] = dist[config] + 1;
139                         chx[ql] = k; chy[ql] = j;
140                         up[ql] = qr;
141                         if (valid[newconf])
142                             printSolution(newconf, ql);
143                     }
144                 }
145             }
146         }
147     }
148     printf("-1\n");
149
150     return 0;
151 }
```

Capitolul 30

ONI 2008

30.1 banda

Problema 1 - banda

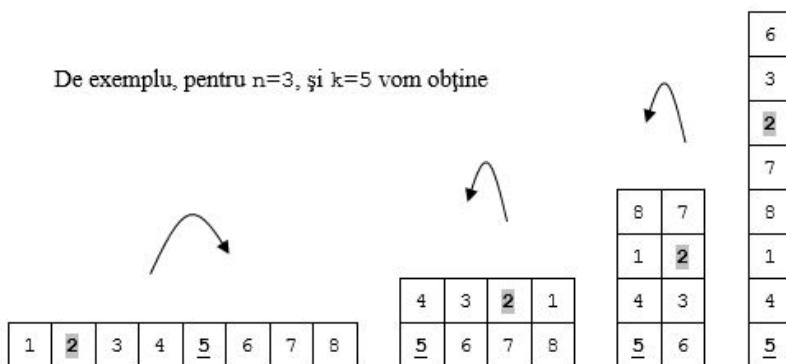
100 de puncte

Pe o masă orizontală se află o bandă super-flexibilă, formată din 2^n pătrățele de latură 1 cm. Grosimea benzii este de 1 mm. Pătrățele sunt numerotate de la stânga la dreapta de la 1 la 2^n . Această bandă se poate plia în două, suprapunând o parte peste cealaltă (stânga peste dreapta sau invers). Astfel, anumite poziții se suprapun și se obține grosime dublă și lungime 2^{n-1} cm. Banda obținută se poate plia iar în două. Acest procedeu se repetă de n ori, până se obține grosimea 2^n mm și lungimea 1 cm (adică toată pătrățele se vor suprapune, formând o singură coloană).

Folosind o pioneză, vom fixa de masă pătrățelul k al benzii și aplicăm procedeul de pliere descris mai sus, obținând astfel o coloană a cărei bază este pătrățelul k . În cadrul acestei coloane vom numera pozițiile pătrățelor de jos în sus de la 1 la 2^n .

Există două tipuri de interogări posibile:

1. Dat fiind un pătrățel (să-l numim pătrățel special), să se determine poziția lui finală (pe coloană).
2. Dată fiind poziția finală a pătrățelului special, să se determine numărul său.



Să observăm că pătrățelul special este 2 și va ajunge în final pe poziția 6.

Cerințe

Date fiind valorile n și k , scrieți un program care răspunde la o interogare de tipul 1 sau de tipul 2.

Date de intrare

Fișierul **banda.in** conține 4 linii. Pe prima linie este scris numărul natural n . Pe a doua linie este scris numărul natural k , reprezentând poziția pătrățelului fixat. Pe cea de a treia linie este scrisă o cifră c care poate fi 1 sau 2, indicând tipul interogării. Dacă cifra c este 1 pe cea de a patra linie este scris numărul natural s reprezentând numărul pătrățelului special. Dacă cifra c este 2 pe cea de a patra linie este scris un număr natural f reprezentând poziția finală a pătrățelului special.

Date de ieșire

Fișierul **banda.out** va conține o singură linie pe care va fi scris un număr natural reprezentând rezultatul interogării.

Restricții și precizări

- $2 \leq n \leq 30$
- $1 \leq k, s, f \leq 2^n$
- Pentru 30% dintre teste $n \leq 7$. Pentru 50% dintre teste $c = 1$.

Exemple:

banda.in	banda.out	Explicații
3 5 1 2	6	În exemplu, banda de lungime $2^3 = 8$ cm este fixată de masă pe poziția 5. În primul exemplu, trebuie să aflăm poziția finală a pătrățelului 2 (aceasta este 6).
3 5 2 4	8	În al doilea exemplu, se știe poziția finală 4, se cere numărul pătrățelului special (8).

Timp maxim de executare/test: **0.1** secunde

Memorie: total **2 MB** din care pentru stivă **1 MB**

30.1.1 Indicații de rezolvare

prof. Zoltan Szabo, Gr. Sc. "Petru Maior", Reghin

Avem două probleme "divide et impera", două variațiuni pe aceeași temă:

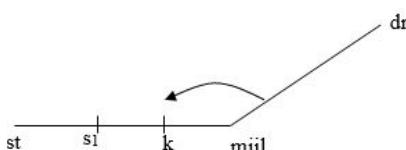
Fie intervalul $[st, dr]$ și mijlocul lui $mijl = (st + dr) \div 2$.

Cerință 1: Unde va ajunge poziția $s1$?

Pornim cu un singur strat de elemente cu lungime 2^n .

La fiecare pliere putem avea două cazuri:

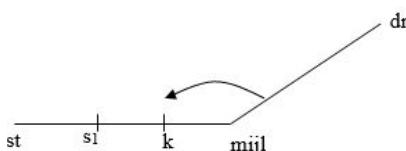
a. $s1$ rămâne pe loc



b. $s1$ își schimbă linia și coloana.

Intervalul studiat se înjumătățește, însă înălțimea lui se dublează (strat)

```
lin:=strat+1-lin;
col:=st+dr-col;
strat:=2*strat;
```



Cerință 2: De unde se ajunge pe $f2$?

Pornim cu o coloană de lungime 1 și înălțime 2^n

Desfacem plierile obținute. Vom avea în vedere o proprietate importantă: în fiecare pas al deplierii, valoarea dr a intervalului $[st, dr]$, dr este divizibil cu $2^{pas} = putere$ și va fi $[dr - putere + 1, dr]$ sau $[st, st + putere - 1]$, în funcție că se desface spre stânga sau spre dreapta.

Dacă linia și coloana pătratului special este sub mijlocul coloanei, atunci aceste valori nu se vor schimba, altfel se vor schimba în simetricele față de extremități:

$lin = 1 + inaltime - lin$

$col = st + dr - col$ (unde st și dr sunt noile dimensiuni ale intervalului)

După n pași se va obține $lin = 1$ și col va fi rezultatul cerut. ($s2 = col$).

30.1.2 *Rezolvare detaliată

30.1.3 Cod sursă

Listing 30.1.1: banda.pas

```

1 program banda;
2 const fin='banda.in';
3         fout='banda.out';
4 var f:text;
5     n:byte;
6     x:char;
7     b:array[1..41] of integer;
8     f1,f2,s1,s2,i,d,cons,k:longint;
9     c:byte;
10
11 function cauta(var n,i:longint):longint;
12 begin
13     i:=0;
14     repeat
15         inc(i);
16     until round(exp(i*ln(2)))>=n;
17     cauta:=round(exp(i*ln(2)));
18 end;
19
20 function initial(poz:longint):longint;
21 var put,pas, d, val,semn:longint;
22 begin
23     if poz=1 then initial:=k
24     else
25         if poz=2 then initial:=cons - k + 1
26         else
27             begin
28                 put:=cauta(poz,pas);
29                 d:=put div 2;
30                 if odd(poz) then semn:=b[pas] else semn:=-b[pas];
31                 initial:=initial(poz-d)+semn* cons div d;
32             end;
33 end;
34
35 procedure vector_de_semne(k:longint);
36 var d:longint;
37     i:byte;
38 begin
39     b[1]:=1;
40     d:=round(exp((n-1)*ln(2)));
41     for i:=2 to n do
42         if k>d then
43             begin
44                 k:=k-d;
45                 b[i]:=-1;
46                 d:=d div 2;
47             end
48         else
49             begin
50                 b[i]:=1;
51                 d:=d div 2;
52             end;
53 end;
54
55 function final(poz:longint):longint;
56 var mijloc,b,j,strat,coloana,linie:longint;
57 begin
58     b:=1;j:=cons;strat:=2;coloana:=poz;linie:=1;
59     repeat
60         mijloc:=(b+j) div 2;
61         if (k<=mijloc) and (mijloc<coloana) then
62             begin
63                 linie:=strat+1-linie;
64                 coloana:=b+j- coloana;
65                 j:=mijloc;

```

```

66      end
67  else
68    if (coloana<=mijloc) and (mijloc<k) then
69      begin
70        linie:=strat+1-linie;
71        coloana:=b+j - coloana;
72        b:=mijloc+1;
73      end
74  else
75    if (k<=mijloc) then
76      j:=mijloc
77    else
78      b:=mijloc+1;
79    strat:=strat*2;
80  until coloana=k;
81  final:=linie;
82 end;
83
84 begin
85   assign(f,fin);
86   reset(f);
87   readln(f,n);
88   readln(f,k);
89   readln(f,c);
90   if c=1 then
91     readln(f,s1)
92   else
93     readln(f,f2);
94   close(f);
95   assign(f,fout);
96   rewrite(f);
97   vector_de_semne(k);
98   cons:=round(exp(n*ln(2)));
99   if c=1 then
100    begin
101      f1:=final(s1);
102      writeln(f,f1);
103    end
104  else
105    begin
106      s2:=initial(f2);
107      writeln(f,s2);
108    end;
109  close(f);
110 end.

```

30.2 pavare

Problema 2 - pavare

100 de puncte

După terminarea facultății, Ionică a ajuns inginer constructor și vrea să se angajeze în orașul său natal. După ce studiază ofertele de muncă, găsește un post de inginer la Primărie. Pentru a ocupa acest post trebuie să susțină o probă teoretică. La această probă, el trebuie să realizeze un proiect pentru pavarea pieței din centrul orașului.

Piața are forma unui dreptunghi și are trasat un caroaj, astfel încât poate fi reprezentată ca un tablou bidimensional cu n linii și p coloane. Fiecare element al matricei corespunde unui pătrat cu latura 1 m. Pavarea se poate realiza folosind dale de două tipuri.

Observați că o dală de tip F este formată din 6 pătrate de latură 1 m, dispuse în forma literei F (deci acoperă o suprafață cu aria de 6 m^2), iar o dală de tipul I este formată din două pătrate cu latura de 1 m (deci va avea aria 2 m^2).

Prin pavarea pieței se înțelege acoperirea fiecărui pătrat de latură 1 m al pieței cu exact o singură dală. Dalele se pot rota și pot fi utilizate pe orice față.

Restriția impusă de primar este ca suprafața din piață pavată cu dale de tipul F să aibă aceeași arie cu cea pavată cu dale de tipul I.

Pentru a vizualiza modalitatea de pavare a pieței, Ionică va numerota dalele cu numere naturale consecutive începând de la 1. Numărul asociat unei dale va fi scris în fiecare pătrat din piață acoperit de dala respectivă.

Tipul F:



Tipul I:



Cerințe

Scrieți un program care să determine o modalitate de pavare a pieței, care să respecte condițiile de mai sus.

Date de intrare

Fișierul de intrare **pavare.in** va conține pe prima linie două numere naturale separate prin spațiu $n \ p$ reprezentând numărul de linii și respectiv numărul de coloane ale matricei.

Date de ieșire

Fișierul de ieșire **pavare.out** va conține n linii, pe fiecare linie fiind scrise p numere naturale separate prin câte un spațiu. Valorile scrise în fișierul de ieșire sunt numerele asociate dalelor care acoperă cele $n \times p$ pătrate ale pieței.

Restricții și precizări

- $3 \leq n, p \leq 150$
- Produsul $n \cdot p$ este multiplu de 24.
- Soluția nu este unică, se poate afișa orice soluție.

Exemplu:

pavare.in	pavare.out	Explicații
6 4	7 7 8 8 1 1 2 2 1 3 3 2 1 1 2 2 1 4 4 2 5 5 6 6	<p>Am pavat o piață cu 6 linii și 4 coloane, având aria $24 m^2$.</p> <p>Pentru pavare s-au utilizat 8 dale (2 dale de tip F care acoperă o suprafață cu aria $2 \cdot 6 = 12 m^2$ și 6 dale de tip I, care acoperă restul pieței, având de asemenea aria $12 m^2$.</p> <p>Fișierul de ieșire corespunde pavării:</p>

Timp maxim de executare/test: **0.1** secunde

Memorie: total **2 MB** din care pentru stivă **1 MB**

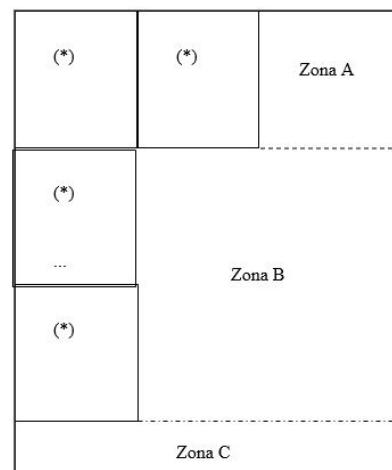
30.2.1 Indicații de rezolvare

prof. Doru Popescu Anastasiu, C. N. "Radu Greceanu" Slatina

Din două dale de tipul F putem acoperi o suprafață dreptunghiulară cu 4 linii și 3 coloane, respectiv 3 linii și 4 coloane (dacă o rotim cu 90° la stânga sau la dreapta):

k	k	h
k	h	h
k	k	h
k	h	h

(*)



Avem două situații:

1. Dacă $m \geq n$, atunci folosind (*) pavăm în jos piața primele 3 coloane, apoi tot în jos coloanele 4, 5, 6 și.m.d., cât timp aria suprafeței acoperite este mai mică decât jumătate din aria suprafeței pieții (adică $m \cdot n / 2$). Acest lucru se poate realiza, pentru că (*) are aria 12 și $m \cdot n$ este multiplu de 24. Apoi zonele rămase (zona A, zona B, zona C) se pavează cu dale de tipul I, orizontale sau verticale. Pot exista și zone vide (din cele trei A, B, C).

Datorită parității laturii verticale din (*) zona A are latura verticală pară și deci poate fi acoperită cu dale de tipul I poziționate vertical. Același lucru se întâmplă cu zona B. Pentru zona C, observăm că latura ei verticală are aceeași paritate cu cea a lui m și cum latura orizontală este n , iar $m \cdot n$ este multiplu de 24, rezultă că zona poate fi pavată ori numai cu dale de tipul I, orizontale, ori numai verticale (după cum este m sau n par).

2. Dacă $n > m$, atunci schimbăm între ele valorile lui m și n , după care ne situăm în primul caz. După construirea soluției, rotim cu 90° la stânga sau la dreapta tabloul și îl afișăm.

30.2.2 *Rezolvare detaliată

30.2.3 Cod sursă

Listing 30.2.1: pavare.pas

```

1 program pavare_dpa;
2 const fi:string='pavare.in';
3     fo:string='pavare.out';
4
5 type figF1=array[1..3,1..4]of integer;
6     figF2=array[1..4,1..3]of integer;
7     figI1=array[1..2,1..1]of integer;
8     figI2=array[1..1,1..2]of integer;
9 var F1:figF1;
10    F2:figF2;
11    I1:figI1;
12    I2:figI2;
13    m,n:integer;
14    a:array[1..170,1..170]of integer;
15
16 procedure cit;
17 var f:Text;
18 begin
19 assign(f,fi);
20 reset(f);
21 readln(f,m,n);
22 close(f);
23 end;
24
25 procedure construire_figF1(var a:figF1; k,h:integer);
26 var i:integer;
27 begin
28 for i:=1 to 4 do
29 begin
30   a[1,i]:=k;
31   a[3,i]:=h;
32   if i mod 2=0 then
33     a[2,i]:=h
34   else
35     a[2,i]:=k;
36   end;
37 end;
38
39 procedure pune_figF1(F1:figF1;i0,j0:integer);
40 var i,j:integer;
41 begin
42 for i:=1 to 3 do
43   for j:=1 to 4 do
44     a[i0+i-1,j0+j-1]:=F1[i,j];
45 end;
46
47 procedure construire_figF2(var a:figF2; k,h:integer);
48 var i:integer;
49 begin
50 for i:=1 to 4 do
51 begin
52   a[i,1]:=h;

```

```

53     a[i,3]:=k;
54     if i mod 2=0 then
55         a[i,2]:=h
56         else
57             a[i,2]:=k;
58     end;
59 end;
60
61 procedure pune_figF2(F2:figF2;i0,j0:integer);
62 var i,j:integer;
63 begin
64   for i:=1 to 4 do
65     for j:=1 to 3 do
66       a[i0+i-1,j0+j-1]:=F2[i,j];
67   end;
68
69 procedure afis2;
70 var i,j:integer;f:text;
71 begin
72   assign(f,fo);
73   rewrite(f);
74   for i:=1 to m do
75     begin
76       for j:=1 to n-1 do
77         write(f,a[i,j],' ');
78         writeln(f,a[i,n]);
79     end;
80   close(f);
81 end;
82
83 procedure afis1;
84 var i,j:integer;f:text;
85 begin
86   assign(f,fo);
87   rewrite(f);
88   for j:=1 to n do
89     begin
90       for i:=1 to m-1 do
91         write(f,a[i,j],' ');
92         writeln(f,a[m,j]);
93     end;
94   close(f);
95 end;
96
97 procedure construire_figI1(var a:figI1;k:integer);
98 begin
99   a[1,1]:=k;a[2,1]:=k;
100 end;
101
102 procedure pune_figI1(I1:figI1;i0,j0:integer);
103 begin
104   a[i0,j0]:=I1[1,1];a[i0+1,j0]:=I1[2,1];
105 end;
106
107 procedure construire_figI2(var a:figI2;k:integer);{cazul m mod 3=0 si n>3}
108 begin
109   a[1,1]:=k;a[1,2]:=k;
110 end;
111
112 procedure pune_figI2(I2:figI2;i0,j0:integer);
113 begin
114   a[i0,j0]:=I2[1,1];a[i0,j0+1]:=I2[1,2];
115 end;
116
117 procedure rez2;{cazul cand m>=n}
118 var
119   r,i,j,k,aria,x,y:integer;
120 begin
121   k:=1;aria:=0;
122   {pavam cu figF2}
123   j:=1;
124   while (j<=n-2)and(aria< m*n div 2) do
125     begin
126       i:=1;
127       while (i<=m-3)and(aria< m*n div 2) do
128         begin

```

```

129      construire_figF2(F2,k,k+1);
130      aria:=aria+12;
131      k:=k+2;
132      pune_figF2(F2,i,j);
133      i:=i+4;
134      end;
135      j:=j+3;
136      end;
137 r:=m;
138 while (r>0) and(a[r,1]=0) do r:=r-1;
139 r:=r+1;
140 {pavam cu I1 zona A}
141 for x:=1 to i-2 do
142   for y:=j to n do
143     if x mod 2=1 then
144       begin
145         construire_figI1(I1,k);
146         k:=k+1;
147         pune_figI1(I1,x,y);
148       end;
149 {pavam cu I2 zona B}
150 for x:=i to r-2 do
151   for y:=j-3 to n do
152     if x mod 2=1 then
153       begin
154         construire_figI1(I1,k);
155         k:=k+1;
156         pune_figI1(I1,x,y);
157       end;
158 {pavam zona C}
159 if m mod 2=0 then
160   for x:=r to m do
161     for y:=1 to n do
162       if x mod 2=1 then
163         begin
164           construire_figI1(I1,k);
165           k:=k+1;
166           pune_figI1(I1,x,y);
167         end
168       else
169         else
170       for x:=r to m do
171         for y:=1 to n do
172           if y mod 2=1 then
173             begin
174               construire_figI2(I2,k);
175               k:=k+1;
176               pune_figI2(I2,x,y);
177             end;
178 end;
179
180 procedure rezolva;
181 var
182   i,j,aux,sw:integer;
183 begin
184   for i:=1 to m do
185     for j:=1 to n do
186       a[i,j]:=0;
187   sw:=1;
188
189   if m>=n then
190     begin
191       rez2;
192       afis2;
193     end
194     else
195     begin
196       aux:=m;
197       m:=n;
198       n:=aux;
199       rez2;
200       afis1;
201     end;
202   end;
203
204 begin{pp}

```

```
205 cit;
206 rezolva;
207 end.
```

30.3 poarta

Problema 3 - poarta

100 de puncte

Harta Galaxiei este reprezentată ca o matrice cu N linii (numerotate de la 1 la N) și M coloane (numerotate de la 1 la M). Orice element al matricei reprezintă o zonă de formă pătrată cu latura de 1 an lumină (denumită quadrant) și poate fi identificat prin coordonatele sale (numărul liniei și respectiv numărul coloanei pe care află).

Nava Enterprise se află într-un quadrant de coordonate cunoscute și trebuie să ajungă la destinație (un alt quadrant, diferit de cel de plecare, de coordonate de asemenea cunoscute).

Nava se poate deplasa de la un quadrant la unul dintre cei învecinăți pe orizontală sau verticală într-o unitate de timp (mai exact, din zona de coordonate (L, C) nava se poate deplasa în una dintre zonele de coordonate $(L - 1, C)$, $(L + 1, C)$, $(L, C - 1)$, $(L, C + 1)$, fără a ieși de pe hartă).

În plus, în unele zone (quadrantii) se găsesc porți stelare. O poartă stelară permite deplasarea navei într-o unitate de timp în oricare altă zonă în care se găsește o altă poartă stelară. Dacă în drumul său nava ajunge într-o zonă cu o poartă stelară, nava se poate deplasa într-o altă zonă cu poartă stelară sau poate să-și continue drumul în una dintre zonele învecinate.

Cerințe

Determinați timpul minim în care nava poate ajunge din zona inițială în cea finală, precum și numărul de trasee de durată minimă.

Date de intrare

Fișierul de intrare **poarta.in** conține pe prima linie numerele naturale $N \ M \ K$, reprezentând în ordine, numărul de linii, numărul de coloane și respectiv numărul de porți stelare de pe hartă. Pe cea de-a doua linie, se află 4 numere naturale $L1 \ C1 \ L2 \ C2$, reprezentând coordonatele zonei de plecare, respectiv coordonatele zonei destinație. Pe următoarele K linii sunt scrise coordonatele zonelor în care se află porți stelare, câte o poartă pe o linie. Numerele de pe aceeași linie sunt separate prin căte un spațiu.

Date de ieșire

Fișierul de ieșire **poarta.out** va conține două linii. Pe prima linie va fi scris numărul natural D , reprezentând timpul minim în care nava ajunge din zona inițială la destinație. Pe cea de-a doua linie va fi scris numărul natural Nr , reprezentând numărul de trasee de durată minimă. Deoarece numărul Nr poate fi foarte mare, trebuie să afișați restul împărțirii lui Nr la 997.

Restricții și precizări

- $1 \leq N, M \leq 100$
- $0 \leq K \leq 1\,000$
- Zona inițială a navei, zona destinație, precum și zonele în care se află porți stelare sunt distințe.
 - Pentru 20% dintre teste $1 \leq N, M \leq 10, 0 \leq K \leq 10$
 - Pentru determinarea corectă a timpului minim se acordă 30% din punctaj. Pentru determinarea corectă a timpului minim și a numărului de trasee de durată minimă se acordă punctajul maxim.

Exemple:

poarta.in	poarta.out	Explicații
6 7 4 2 5 6 2 1 1 5 1 1 6 4 5	5 6	<p>Harta Universului are 6 linii și 7 coloane. Ea conține 4 porți stelare (1 2 3 4). Nava Enterprise se află inițial în zona de coordonate (2,5) și are ca destinație zona de coordonate (6,2).</p> <p>Durata minimă deplasării de la \blacktriangle la \blacksquare este 5, un traseu posibil de durată 5 fiind: $(2,5) \rightarrow (2,6) \rightarrow (1,6) \rightarrow (5,1) \rightarrow (5,2) \rightarrow (6,2)$. Există 6 trasee de lungime minimă 5.</p>

Timp maxim de executare/test: **0.1** secunde

Memorie: total **2 MB** din care pentru stivă **1 MB**

30.3.1 Indicații de rezolvare

prof. Radu Vișinescu, C. N. "I. L. Caragiale" Ploiești

1. Pentru determinarea duratei minime:

Se determină distanța minima $d1$ a unei porți stelare fata de punctul de plecare și analog distanța minima $d2$ fata de punctul de sosire. Distanța(durata) minima folosind porți stelare va fi $d1 + d2 + 1$.

Se determină distanța minima $d0$ între punctul de plecare și cel de sosire fără a folosi porți stelare.

ACESTE DISTANȚE SUNT DE TIP *distanța Manhattan*.

In final se consideră minimul celor două distanțe determinate la punctele anterioare. Durata minima a unui traseu va fi: $d = \min(d0, d1 + d2 + 1)$

2. Pentru determinarea numarului de trasee :

Se consideră submultimea de porți stelare aflate la distanța minima fata de punctul de plecare. Fie (x, y) coordonatele punctului de plecare și (p, q) coordonatele unei astfel de porți stelare. Numarul de trasee distincte între cele două pozitii se poate determina prin *metoda programării dinamice*. Din considerente de simetrie, putem presupune fără a restrange generalitatea că $x < p$ și $y < q$. Folosim o matrice mat cu $n = q - y + 1$ linii și $m = p - x + 1$ coloane. Un drum de distanță minima ce ajunge în poziția (p, q) are ca penultima poziție ori poziția $(p - 1, q)$ ori $(p, q - 1)$. Deci putem calcula recurrent:

$$mat[1, j] = 1 \text{ pentru } j = 1, m$$

$$mat[i, j] = mat[i, j - 1] + mat[i - 1, j] \text{ pentru } 2 \leq i \leq n \text{ și } j = 1, m.$$

Numrul de trasee distincte va fi $mat[n, m]$. Aceasta matrice poate fi calculată la început o singură dată, astfel avem toate valorile dorite.

Adunăm aceste valori pentru toate porțile stelare aflate la distanța minima fata de punctul de plecare și obținem valoarea nr1.

Procedăm analog pentru punctul de sosire și obținem valoarea nr2.

Numarul de trasee distincte de durată minima folosind porți stelare va fi produsul $numar1 = nr1 * nr2$.

Calculăm recurrent, analog, numarul de drumuri directe de distanță (durată) minima între pozițiile initială (x, y) și finală $(x2, y2)$ fără folosirea porților stelare obținând valoarea $numar2$.

In funcție de rezultatul privind prima cerere de distanță minima, valoarea căutată va fi $numar1$, dacă este mai scurt drumul folosind porți stelare, respectiv $numar2$ dacă distanța (durată) minima este cea directă.

Un caz special este cand distanța fără a folosi porți stelare este egală cu cea obținută folosind porți stelare caz în care cele două valori obținute se adună: $numar1 + numar2$.

Complexitatea finală este astfel $O(N * M)$.

30.3.2 *Rezolvare detaliată

30.3.3 Cod sursă

Listing 30.3.1: poarta.pas

```

1 program poarta;
2 const p=997;
3 var f,g:text;
4 n,m,k,x,y,x2,y2:integer;
5 a,b:array[1..5000]of integer;
6 mat:array[1..100,1..100]of integer;
7 d,d2:integer;i,min1,min2:integer;nr,nr2,numar1,numar2:longint;
8
9 {citirea datelor de intrare}
10
11 procedure citire;
12 var i:integer;
13 begin
14 assign(f,'poarta.in');
15 reset(f);
16 readln(f,n,m,k);
17 readln(f,x,y,x2,y2);
18 for i:=1 to k do
19   readln(f,a[i],b[i]);
20 close(f);
21 end;
22
23 function numar(x,y,x2,y2:integer):integer;
24 var n,m:integer;i,j:integer;
25 begin
26 {calculul numarul de trasee dintre pozitiile de
27 coordonate (x,y) si (x2,y2) prin programare dinamica}
28 n:=abs(x-x2)+1;
29 m:=abs(y-y2)+1;
30 for j:=1 to m do mat[1,j]:=1;
31 for i:=2 to n do
32   begin
33     mat[i,1]:=1;
34     for j:=2 to m do
35       mat[i,j]:=((mat[i,j-1]mod p)+(mat[i-1,j]mod p))mod p;
36   end;
37 numar:=mat[n,m];
38 end;
39
40 begin
41 citire;
42 {calculul distantei si traseelor directe}
43 d:=abs(x-x2)+abs(y-y2);
44 nr:=numar(x,y,x2,y2);
45 if k<>0 then
46   begin
47     {calculul distantei minime de la punctul de plecare la
48     portile stelare si numarul de trasee ce respecta distanta respectiva}
49     min1:=32000;numar1:=0;
50     for i:=1 to k do
51       begin
52         d2:=abs(x-a[i])+abs(y-b[i]);
53         if d2<min1 then begin min1:=d2;numar1:=numar(x,y,a[i],b[i]);end
54         else if d2=min1 then numar1:=(numar1+numar(x,y,a[i],b[i]))mod p;
55       end;
56     {calculul distantei minime de la punctul de sosire la
57     portile stelare si numarul de trasee ce respecta distanta respectiva}
58     min2:=32000;numar2:=0;
59     for i:=1 to k do
60       begin
61         d2:=abs(x2-a[i])+abs(y2-b[i]);
62         if d2<min2 then begin min2:=d2;numar2:=numar(x2,y2,a[i],b[i]) end
63         else if d2=min2 then numar2:=(numar2+numar(x2,y2,a[i],b[i]))mod p;
64       end;
65     {distanta minima si numarul de trasee folosind porti stelare}
66     d2:=min1+min2+1;
67     nr2:=((numar1 mod p)*(numar2 mod p))mod p;
68     {rezultatul final}
69     if d2<d then begin d:=d2;nr:=nr2;end
70     else if d2=d then nr:=(nr+nr2)mod p;

```

```

71   end;
72 assign(g,'poarta.out');
73 rewrite(g);
74 writeln(g,d);
75 write(g,nr);
76 close(g);
77 end.

```

30.4 aranjare

Problema 4 - aranjare

100 de puncte

La festivitatea de deschidere a ONI Ploiești, Gigel, mare amator de probleme de logică a observat că primul rând din sală este format din $2 * N + 2$ scaune, numerotate de la stânga la dreapta de la 1 la $2 * N + 2$. Pe acest rând s-au așezat, într-o ordine oarecare, N băieți și N fete, cele două scaune rămase libere fiind alăturate.

Imediat Gigel inventează o problemă. Să considerăm că singura mutare posibilă este ca doi elevi care stau pe scaune alăturate să se ridice și să se așeze (în aceeași ordine) pe cele două locuri libere.

Problema constă în determinarea unei secvențe de mutări după executarea cărora toate fetele să fie grupate în stânga rândului, toți băieții să fie grupați în dreapta, iar cele două locuri libere să separe fetele de băieți.

Cerințe

Cunoscându-se valoarea N , precum și aranjarea inițială a elevilor, să se determine o secvență de mutări care să conducă în final la aranjarea descrisă în enunțul problemei.

Date de intrare

Fisierul de intrare **aranjare.in** conține pe prima linie numărul natural N . Linia a doua a fisierului de intrare conține un sir de $2 * N + 2$ caractere (N caractere *F*, N caractere *B* și două caractere alăturate *S*) care reprezintă aranjarea inițială a elevilor pe rând și cele două locuri libere. Mai exact, *F* indică un loc ocupat de o fată, *B* un loc ocupat de un băiat, iar *SS* reprezintă cele două scaune libere.

Date de ieșire

Fisierul de ieșire **aranjare.out** va conține mutările din secvență determinată, în ordine, către o mutare pe o linie. O mutare este descrisă printr-o valoare naturală K ($1 \leq K < 2 * N + 2$) cu semnificația "se vor muta pe scaunele libere elevii situați pe scaunele K și $K + 1$ ".

Restricții și precizări

- $3 \leq N \leq 5\ 000$
- Pentru datele de test există totdeauna soluție. Soluția nu este unică.

Exemple:

aranjare.in	aranjare.out	Explicații
3	7	Se vor obține pe rând configurațiile: FBFBFBSS
FBSSFBFB	2	FSSBFBBF
	5	FFBBBSSBF
	3	FFSSBBBF
	6	FFBBBSSSF
	1	FFBBBSSSF
	7	SSBBBBFF
	3	FFBBBBFSS
	6	FFSSBFBB
	4	FFFFBBSSB FFFSSBBB

Timp maxim de executare/test: **0.1** secunde

Memorie: total **2 MB** din care pentru stivă **1 MB**

30.4.1 Indicații de rezolvare

prof. Marinel șerban, Liceul de Informatică "Grigore Moisil" Iași

Problema se poate rezolva folosind un algoritm de tip *greedy*.

Cât timp nu s-a ajuns la soluția finală, se alege primul băiat din partea stânga. Evident, în locul lui va trebui așezată o fată, dar pentru aceasta trebuie mai întâi să eliberăm spațiul ocupat de el, deci putem distinge 2 cazuri:

a. Configurația începând cu el este de tipul BSS..., deci nu putem efectua o mutare directă. Pentru aceasta, va trebui să aducem doi copii alăturați pe pozițiile imediat următoare (de exemplu, cei de pe ultimele două poziții) iar apoi putem muta băiatul curent, împreună cu persoana din dreapta, pe pozițiile libere

b. În cazul în care poziția din dreapta băiatului e ocupată, se efectuează direct mutarea.

După ce am eliberat spațiile respective, trebuie să căutăm prima fată (de la poziția curentă în dreapta) și să o mutăm pe poziția eliberată. Mai e un singur caz particular, și anume cel în care fata e așezată pe ultima poziție. În acest caz putem proceda astfel:

```
FFFSBBBBBF
FFFBFBBBSS
FFFBFSSBBB
FFSSFFBBBB
FFFFSSBBBB
```

30.4.2 *Rezolvare detaliată

30.4.3 Cod sursă

Listing 30.4.1: aranjare.cpp

```

1 #include <iostream>
2 using namespace std;
3
4 #define MAXN 10011
5
6 int N, sp;
7 char s[MAXN];
8
9 void read()
10 {
11     scanf("%d", &N);
12     scanf("%s", s);
13 }
14
15 void makeSimpleMove(int pos)
16 {
17     printf("%d\n", pos + 1);
18     s[sp] = s[pos], s[sp + 1] = s[pos + 1];
19     s[pos] = s[pos + 1] = 'S';
20     sp = pos;
21 }
22
23 void makeMove(int pos)
24 {
25     if (s[pos + 1] != 'S')
26         makeSimpleMove(pos);
27     else
28     {
29         makeSimpleMove(2 * N);
30         makeSimpleMove(pos);
31     }
32
33     int i;
34     for (i = pos + 1; i < 2 * N + 2; i++)
35         if (s[i] == 'F')
36             break;
```

```

37     if (i != 2 * N + 1)
38         makeSimpleMove(i);
39     else
40     {
41         makeSimpleMove(2 * N);
42         makeSimpleMove(pos + 2);
43         makeSimpleMove(pos - 1);
44         makeSimpleMove(pos + 1);
45     }
46 }
47
48 void solve()
49 {
50     for (int i = 0; i < 2 * N + 2; i++)
51         if (s[i] == 'S')
52         {
53             sp = i;
54             break;
55         }
56     for (int i = 0; i < N; i++)
57         if (s[i] != 'F')
58             makeMove(i);
59     if (s[N] != 'S')
60         makeSimpleMove(N);
61 }
62
63 int main()
64 {
65     freopen("aranjare.in", "r", stdin);
66     freopen("aranjare.out", "w", stdout);
67
68     read();
69     solve();
70
71     return 0;
72 }
```

30.5 bile

Problema 5 - bile

100 de puncte

Bogdan a primit de ziua sa un joc foarte ingenios. Jocul este constituit dintr-o cutie cu două compartimente. Inițial în primul compartiment se află k bile (numerotate de la 1 la k), iar în al doilea compartiment se află $n - k$ bile (numerotate de la $k + 1$ la n). Cele două compartimente comunică printr-o ușă basculantă specială care are două lăcașuri. Un lăcaș se află în compartimentul 1, iar celălalt în compartimentul 2. Într-un lăcaș poate să încapă o singură bilă.

Vasile poate alege o bilă din compartimentul 1 și o bilă din compartimentul 2, să plaseze bilele alese în cele două lăcașuri ale ușitei și să rotească ușă. Astfel bila din compartimentul 1 va trece în compartimentul 2, iar bila din compartimentul 2 va trece în compartimentul 1.

Aceasta este singura mutare posibilă.

Scopul jocului este de a executa o succesiune de mutări astfel încât în compartimentul 1 să se obțină succesiv toate submulțimile distințe de k elemente ale mulțimii $\{1, 2, \dots, n\}$.

Cerințe

Scriți un program care să afișeze submulțimile de k elemente ale mulțimii $\{1, 2, \dots, n\}$ în ordinea în care acestea pot fi obținute în compartimentul 1 cu ajutorul ușitei basculante.

Date de intrare

Fișierul de intrare **bile.in** va conține pe prima linie numerele naturale n și k , separate printr-un spațiu.

Date de ieșire

Fișierul de ieșire **bile.out** va conține câte o linie pentru fiecare submulțime obținută în compartimentul 1. Pe fiecare linie vor fi scrise în ordine crescătoare k numere naturale din mulțimea $\{1, 2, \dots, n\}$, separate prin câte un spațiu, reprezentând elementele submulțimii. Pe prima linie va fi afișată submulțimea inițială (adică numerele $1 2 \dots k$).

Restricții și precizări

- $1 \leq k < n \leq 20$
- Soluția nu este unică, puteți afișa oricare dintre variantele corecte.

Exemplu:

bile.in	bile.out	Explicații
4 2	1 2	La prima mutare au fost plasate în ușita basculantă bila 2 (din compartimentul 1) și bila 3 (din compartimentul 2).
	1 3	La a doua mutare au fost alese bilele 3 și 4.
	1 4	La a treia mutare au fost alese bilele 1 și 2.
	2 4	La a patra mutare au fost alese bilele 4 și 3.
	2 3	Iar la ultima mutare au fost alese bilele 2 și 4.
	3 4	

Timp maxim de executare/test: **1.4** secunde

Memorie: total **5 MB** din care pentru stivă **1 MB**

30.5.1 Indicații de rezolvare

prof. Emanuela Cerchez, Liceul de Informatică "Grigore Moisil" Iași

Soluție 1.

Problema cere să generăm toate submulțimile de k elemente ale mulțimii $\{1, 2, \dots, n\}$ astfel încât oricare două submulțimi generate consecutiv să difere printr-un singur element.

Numărul de soluții este, evident, $\text{Comb}(n,k)=n!/(k!(n-k)!)$.

Algoritmul de generare este inspirat din relația de recurență a lui *Pascal*:

$\text{Comb}(n,k)=\text{Comb}(n-1,k)+\text{Comb}(n-1,k-1)$.

Mai exact submulțimile de k elemente ale mulțimii $\{1, 2, \dots, n\}$ pot fi particionate în două clase: submulțimile care conțin valoarea n (acestea sunt $\text{Comb}(n-1,k-1)$) și submulțimile care nu conțin valoarea n (acestea sunt $\text{Comb}(n-1,k)$).

Să notăm cu $S_{n,k}$ o soluție corectă (adică o succesiune a combinațiilor de n luate câte k care respectă condițiile din enunț).

$S_{n,1} = \{1\} \{2\} \{3\} \dots \{n\}$, pentru orice n

$S_{n,n} = \{1, 2, 3, \dots, n\}$

Pentru a genera $S_{n,k}$:

1. vom genera combinațiile de $n - 1$ luate câte k în ordinea specificată (adică $S_{n-1,k}$)
2. vom genera combinațiile de $n - 1$ luate câte $k - 1$ în ordinea specificată (adică $S_{n-1,k-1}$), la care reunim elementul n
3. Construim $S_{n,k}$ din $S_{n-1,k}$ urmat de $\underline{S_{n-1,k-1}} \cup \{n\}$ unde cu $\underline{S_{n-1,k-1}}$ am notat $S_{n-1,k-1}$ considerat în ordine inversă.

De exemplu:

$$S_{2,1} = \{1\}\{2\}$$

$$S_{3,1} = \{1\}\{2\}\{3\}$$

$$S_{3,2} = S_{2,2}$$

$$S_{2,1} \cup \{3\} = \{1,2\}\{2,3\}\{1,3\}$$

$$S_{4,1} = \{1\}\{2\}\{3\}\{4\}$$

$$S_{4,2} = S_{3,2}$$

$$S_{3,1} \cup \{4\} = \{1,2\}\{2,3\}\{1,3\}\{3,4\}\{2,4\}\{1,4\}$$

$$S_{4,3} = S_{3,3}$$

$$S_{3,2} \cup \{4\} = \{1,2,3\} \{1,3,4\} \{2,3,4\}\{1,2,4\}$$

$$S_{5,2} = S_{4,2}$$

$$S_{4,1} \cup \{5\} = \{1,2\}\{2,3\}\{1,3\}\{3,4\}\{2,4\}\{1,4\}\{4,5\}\{3,5\}\{2,5\}\{1,5\}$$

$$S_{5,3} = S_{4,3}$$

$$S_{4,2} \cup \{5\} = \{1,2,3\}\{1,3,4\}\{2,3,4\}\{1,2,4\}\{1,4,5\}\{2,4,5\}\{3,4,5\}\{1,3,5\}\{2,3,5\}\{1,2,5\}$$

etc.

Se poate demonstra cu ușurință prin inducție că acest procedeu produce o secvență de combinații care respectă condițiile din enunț.

Observați că procedeul de generare este asemănător celui utilizat pentru *codul Gray*.

Vom descrie un *algoritm de tip succesor* pentru generarea $S_{n,k}$:

Configurația inițială este $s = \{1, 2, \dots, k\}$

Cât timp nu am generat toate combinările:

- afișăm configurația curentă;
- generăm configurația următoare, dacă există.

Generarea succesorului:

Pentru a nu trata în mod special cazul ultimului element,

```

s[k+1]=n+1;
for (j=1; j<=k && s[j]==j; j++)
    if (j%2!=k%2)
        if (j==1) s[1]--;
        else {s[j-1]=j; s[j-2]=j-1;}
    else
        if (s[j+1]!=s[j]+1) {s[j-1]=s[j]; s[j]++;}
        else {s[j+1]=s[j]; s[j]=j}

```

Soluție 2 (Zoltan Szabo)

Se cere generarea tuturor combinărilor a n elemente luate câte k astfel încât două mulțimi successive să difere printr-un singur element.

Să generăm aceste combinări cu *metoda backtracking*, folosind o *stivă st* în care vom pune k elemente în ordine strict crescătoare, generând toate soluțiile în *ordine lexicografică*.

Putem observa, că în toate cazurile în care ultimul element al stivei are succesor ($st[k] < n$), cerința cerută va fi respectată (fiindcă ultimul element crește cu 1, și toate celelalte elemente rămân neschimbate).

În cazurile în care ultimul element al mulțimii nu are succesor ($st[k] = n$), trecerea către următorul element, nu va respecta cerința dorită.

Să observăm, că elementul $st[p]$ nu are successor în ordine lexicografică, dacă $st[p]-p = n-k$. Deci valorile posibile pentru $st[p]$ sunt incluse în intervalul $[st[p-1], n-p-k]$.

Vom modifica algoritmul anterior, astfel încât elementele stivei să poată să aibă la diferite nivele pașii 1 sau -1 astfel:

- pentru pasul 1 valoarea $st[p]=n-p-k$ nu are succesor,
- pentru pasul -1 valoarea $st[p]=st[p-1]$ nu are predecesor .

Combinări în ordine lexicografică	Pasul corespunzător fiecarui element din stivă	Genera rea bilelor	Pasul corespunzător fiecarui element din stivă	Explicație
1 2 3 4	(1,1,1,1)	1 2 3 4	(1,1,1,1)	Pornim cu prima combinare banală și pt. fiecare nivel pas=1
1 2 3 5	(1,1,1,1)	1 2 3 5	(1,1,1,1)	
1 2 3 6	(1,1,1,1)	1 2 3 6	(1,1,1,1)	
1 2 4 5	(1,1,1,1)	1 2 4 6	(1,1,1,-1)	Când ultimul nu mai are succesor, revenim, generăm succesorul, iar pentru elementele schimbăm pasul
1 2 4 6	(1,1,1,1)	1 2 4 5	(1,1,1,-1)	
1 2 5 6	(1,1,1,1)			
1 3 4 5	(1,1,1,1)			
1 3 4 6	(1,1,1,1)	1 2 5 6	(1,1,1,1)	Penultimul și ultimul nivel nu mai au succesor
1 3 5 6	(1,1,1,1)			
1 4 5 6	(1,1,1,1)	1 3 5 6	(1,1,-1,-1)	La nivelul 2 am trecut la succesor, iar nivelele superioare și-au schimbat pasul
2 3 4 5	(1,1,1,1)	1 3 4 6	(1,1,-1,-1)	
2 3 4 6	(1,1,1,1)	1 3 4 5	(1,1,-1,-1)	
2 3 5 6	(1,1,1,1)	1 4 5 6	(1,1,1,1)	
2 4 5 6	(1,1,1,1)	2 4 5 6	(1,-1,-1,-1)	Etc.etc.
3 4 5 6	(1,1,1,1)	2 3 5 6	(1,-1,-1,-1)	
		2 3 4 6	(1,-1,-1,-1)	
		2 3 4 5	(1,-1,-1,-1)	
		3 4 5 6	(1,1,1,1)	

Să observăm că prin acest procedeu, după ce am generat un element, restul elementelor stivei se pot completa instantaneu până la nivelul k . Astfel avem un *algoritm backtracking optimizat*, care generează toate soluțiile în ordine corectă fără să treacă prin valori succesive nevalide.

Soluție 3 (80 puncte): Andrei Grigorean, Universitatea București

Vom construi o procedura recursiva care primind doi parametri N și K va genera toate combinarile de N elemente luate cate K care să resepece proprietatea din enunt.

Vom incerca să generam mai intai toate combinarile care nu contin elementul N , urmate de toate combinarile care îl contin pe N .

Pentru a face acest lucru la inceputul procedurii noastre vom face 2 apeluri recursive cu urmatorii parametri: $N - 1$ și K pentru a genera combinarile care nu îl contin pe N , urmat de un apel cu parametrii $N - 1$ și $K - 1$.

Pentru combinarile rezultate în urma celui de-al doilea apel, vom adauga la sfârșit elementul N .

Astfel am obținut ceea ce ne-am propus: toate combinarile de N elemente luate cate K .

Problema care apare este ca nu avem garantia faptului că ultima combinare din primul apel și prima din al doilea respectă condițiile din enunt. Ne vom folosi de urmatoarea observatie:

Având un sir de combinari de N elemente luate cate K care respectă proprietatea din enunt și o *funcție bijectivă* $f : N \rightarrow N$, înlocuind peste tot un element x cu valoarea $f(x)$, sirul de combinarile rezultă respectă la randul său proprietatea din enunt.

Considerăm că ultima combinare rezultată în urma primului apel este $v_1, v_2, \dots, v_{K-1}, v_K$, iar prima combinare rezultată în urma celui de-al doilea apel este u_1, u_2, \dots, u_{K-1} . Ne vom alege o funcție $f : (N - 1) \rightarrow (N - 1)$ astfel încât $f(u_1) = v_1, \dots, f(u_{K-1}) = v_{K-1}$ (pentru elementele x din intervalul $[1 \dots N - 1]$ care nu se regăsesc printre valorile u_i , $f(x)$ poate lua orice valoare care pastrează bijectivitatea). Nu este deloc dificil să gasim o astfel de funcție bijectivă. Înlocuind fiecare element x din combinarile rezultate în urma celui de-al doilea apel cu $f(x)$, obținem o soluție validă.

30.5.2 *Rezolvare detaliată

30.5.3 Cod sursă

Listing 30.5.1: bile.c

```

1 //EmCerchez
2 #include <stdio.h>
3
4 #define InFile "bile.in"
5 #define OutFile "bile.out"
6
7 #define NMax 50
8
9 int n, k;
10 int s[NMax];
11 long nr;
12
13 long Pascal(void);
14
15 int main()
16 {
17     FILE * fout=fopen(OutFile,"w");
18     FILE * fin=fopen(InFile,"r");
19
20     long int i, j;
21     fscanf(fin,"%d %d",&n,&k);
22     fclose(fin);
23
24     for (i=1; i<=k; i++) s[i]=i;
25     s[k+1]=n+1;
26     nr=Pascal();
27     for (i=0; i<nr; i++)
28     {
29         //afisare
30         for (j=1; j<k; j++) fprintf(fout,"%d ",s[j]);

```

```

31         fprintf(fout,"%d\n",s[k]);
32
33     //succesorul
34     for (j=1; j<=k && s[j]==j; j++);
35
36     if (j%2!=k%2)
37     {
38         if (j==1)
39             s[1]--;
40         else
41             {
42                 s[j-1]=j;
43                 s[j-2]=j-1;
44             }
45         else
46             {
47                 if (s[j+1]!=s[j]+1)
48                 {
49                     s[j-1]=s[j];
50                     s[j]++;
51                 }
52                 else
53                 {
54                     s[j+1]=s[j];
55                     s[j]=j;
56                 }
57             }
58         fclose(fout);
59         return 0;
60     }
61 long Pascal(void)
62 {
63     int i, j;
64     long L1[NMax], L2[NMax];
65
66     L1[0]=L1[1]=L2[0]=1;
67
68     for (i=2; i<=n; i++)
69     {
70         for (j=1; j<i; j++)
71             L2[j]=L1[j-1]+L1[j];
72
73         L2[i]=1;
74         for (j=1; j<=i; j++)
75             L1[j]=L2[j];
76     }
77
78     return L2[k];
79 }
```

Listing 30.5.2: bile.cpp

```

1 //Mugurel Andreica
2 #include <stdio.h>
3
4 #define NMAX 30
5
6 int N, K;
7 int last[NMAX];
8
9 void genComb(int N, int K, int nlast, int reversed)
10 {
11     int i, j;
12
13     if (N < K)
14         return;
15
16     if (K == 1)
17     {
18         if (reversed)
19         {
20             for (i = N; i >= 1; i--)
21             {
22                 printf("%d", i);
23                 for (j = nlast - 1; j >= 0; j--)
```

```

24             printf(" %d", last[j]);
25         }
26     }
27 }
28 }
29 else
30 {
31     for (i = 1; i <= N; i++)
32     {
33         printf("%d", i);
34         for (j = nlast - 1; j >= 0; j--)
35             printf(" %d", last[j]);
36         printf("\n");
37     }
38 }
39
40     return;
41 }
42
43 if (reversed)
44 {
45     last[nlast] = N;
46     genComb(N-1, K-1, nlast + 1, 0);
47     genComb(N-1, K, nlast, 1);
48 }
49 else
50 {
51     genComb(N-1, K, nlast, 0);
52     last[nlast] = N;
53     genComb(N-1, K-1, nlast + 1, 1);
54 }
55 }
56
57 int main()
58 {
59     freopen("bile.in", "r", stdin);
60     freopen("bile.out", "w", stdout);
61
62     scanf("%d %d", &N, &K);
63
64     genComb(N, K, 0, 0);
65
66     return 0;
67 }
```

30.6 subgeom

Problema 6 - subgeom

100 de puncte

Miruna tocmai a învățat la ora de matematică despre progresii geometrice. Un sir de numere naturale nenule se numește *progresie geometrică* dacă este respectată una dintre următoarele două condiții:

- sirul este format dintr-un singur element.
- Dacă sirul este format din N ($N > 1$) elemente v_1, v_2, \dots, v_N , atunci există un număr întreg K mai mare strict decât 1 astfel încât raportul oricărora două elemente consecutive din sir este egal cu K . Altfel spus, oricare ar fi un indice i , $1 \leq i < N$, $v_{i+1}/v_i = K$.

Miruna are o imaginație bogată, aşa că inventează o noțiune nemaîntâlnită până acum - cea de *subsir geometric*: dându-se un sir de numere naturale nenule, un subsir care este progresie geometrică se numește subsir geometric.

Cerințe

Scrieți un program care pentru un sir de N elemente afișează lungimea celui mai lung subsir geometric al său.

Date de intrare

Fișierul de intrare **subgeom.in** va conține pe prima linie numărul natural T reprezentând numărul de seturi de date din fișier.

Fiecare dintre următoarele T linii conține un set de date, sub forma:

$N \ v_1 \ v_2 \dots \ v_N$

Prima valoare este un număr natural N , reprezentând numărul de elemente din sir, urmat de cele N numere naturale nenule ce alcătuiesc sirul, separate prin căte un spațiu.

Date de ieșire

Fișierul de ieșire **subgeom.out** va conține T linii, câte o linie pentru fiecare set de date. Linia i conține un număr natural reprezentând lungimea maximă a unui subșir geometric al sirului descris pe linia $i + 1$ în fișierul de intrare.

Restricții și precizări

- $1 \leq T \leq 10$
- $1 \leq N \leq 5\,000$
- Pentru 40% din teste $1 \leq N \leq 128$
- Elementele sirului sunt numere naturale din intervalul $[1, 105]$
- Un subșir al unui sir $v_1v_2\dots v_N$ este format din elemente ale sirului considerate în ordinea în care acestea apar în sir: $v_{i_1} v_{i_2} \dots v_{i_k}$ ($1 \leq i_1 < i_2 < \dots < i_k \leq N$).

Exemple:

subgeom.in	subgeom.out	Explicații
6	1	Pentru primele trei teste toate subșirurile geometrice au lungimea 1.
3 5 3 7	1	
3 8 4 2	1	Pentru al patrulea test soluția este formată din subșirul 5, 10.
3 4 4 4	2	Pentru al cincilea test soluția este formată din subșirul 1, 3, 9.
3 5 1 10	3	Pentru ultimul test soluția este formată din subșirul 2, 6, 18
4 1 2 3 9	3	
5 6 2 8 6 18		

Timp maxim de executare/test: **1.8** secunde pe Windows, **0.2** secunde pe Linus

Memorie: total **2 MB** din care pentru stivă **1 MB**

30.6.1 Indicații de rezolvare

Andrei Grigorean, Universitatea București

Vom împărți rezolvarea problemei în două:

1. Găsirea unui subșir geometric de lungime mai mare sau egală cu 3 (dacă există).

Notând cu v_0 primul element din subșir și cu R rația, observăm că al doilea element este egal cu $v_0 * R$, iar al treilea cu $v_0 * R^2$. De aici tragem concluzia că valoarea maximă a lui R este $\text{int}(\sqrt{ValMax})$, unde $ValMax$ este valoarea maximă din sir.

Vom încerca toate valorile posibile ale lui R și vom reține maximul dintre subșirurile geometrice găsite.

În continuare vom rezolva *subproblema* găsirii celui mai lung subșir geometric pentru un R fixat în $O(N)$.

Rezolvarea necesită o *sortare* a elementelor din sir anterioară acestui pas.

Având elementele sortate, ne vom construi o *dinamică*

$Best[i] =$ lungimea celui mai lung subșir geometric care are ca ultim element pe $Element[i]$.

La construcția recurenței observăm că elementul de pe poziția i va influența doar acel element egal cu $Element[i] * R$ (dacă există).

Pentru a implementa eficient această recurență ne vom folosi de 2 pointeri $p1$ și $p2$: $p1$ va parcurge pe rând lista sortată a elementelor, iar $p2$ va pointa către cel mai mic element mai mare sau egal cu $Element[p1] * R$ (dacă există).

Observăm că atunci când creștem $p1$ cu o unitate, trebuie să creștem și $p2$.

Trebue acordată o atenție specială cazurilor în care nu avem elemente distincte în sir.

Astfel complexitatea acestui pas este $O(N)$, deoarece vom parcurge cele N elemente de 2 ori, cu cei doi pointeri.

Complexitatea finală a acestui pas este $O(N \log N + N * \text{int}(\sqrt{Valmax}))$.

2. Găsirea unui subşir geometric de lungime egală cu 2.

De fapt ne interesează dacă există 2 indici i și j , $i < j$, astfel încât elementul de pe poziția j să se dividă cu cel de pe poziția i . Implementarea eficientă a acestui pas are la bază metoda numită *Ciurul lui Eratostene*.

Vom folosi un vector de valori booleene *Found*[], de mărime *Valmax*. Vom parcurge pe rând elementele din sir și la fiecare pas vom marca toți multiplii elementului curent. Dacă la un moment dat poziția corespunzătoare elementului din sir la care am ajuns este marcată, înseamnă că am găsit un subşir geometric de lungime 2.

Complexitatea acestui pas este mai dificil de evaluat, deoarece depinde de valorile sirului, însă nu poate depăși $O(Valmax \log Valmax)$.

30.6.2 *Rezolvare detaliată

30.6.3 Cod sursă

Listing 30.6.1: subgeom.cpp

```

1 //Andrei Grigorean
2 #include <iostream>
3 #include <algorithm>
4 #include <vector>
5 #include <bitset>
6
7 using namespace std;
8
9 #define FILEIN "subgeom.in"
10 #define FILEOUT "subgeom.out"
11
12 const int MAXVAL = 100005;
13 const int MAXN = 5005;
14
15 ifstream fin(FILEIN);
16 ofstream fout(FILEOUT);
17
18 int A[MAXN];      // A[i] = greatest length for a "subşir geometric" having
19                  // the last element v[i]
20
21 int Solve(vector< pair<int, int> > &v)
22 {
23     int best = 1;          // length of greatest "subşir geometric" and
24                  // function's return value
25     int N = int(v.size());           // number of elements
26     vector< pair<int, int> > u = v;    // make a copy of the array
27
28     sort(v.begin(), v.end());
29
30     // find result for lengths greater than 2
31     for (int R = 2; R*R <= MAXVAL; ++R)
32     {
33         for (int i = 0; i < N; ++i)
34             A[i] = 1;
35
36         int p1 = 0, p2 = 0;
37         while (p2 < N && v[p2].first < v[p1].first * R)
38             ++p2;
39
40         while (p2 < N)
41         {
42             if (v[p2].first == v[p1].first * R)
43             {
44                 int q1 = p1, q2 = p2;
45                 int sol = 1;
46
47                 while (q2 < N && v[q2].first == v[p2].first)
48                 {
49                     while(v[q1].first == v[p1].first &&
50                           v[q1].second < v[q2].second)
51                         sol = max(sol, A[q1++]+1);
52
53                 }
54             }
55         }
56     }
57
58     return best;
59 }

```

```

52
53             A[q2] = max(A[q2], sol);
54             ++q2;
55         }
56
57         while (v[q1].first == v[p1].first)
58             ++q1;
59
60         p2 = q2;
61         p1 = q1;
62     }
63     else
64         ++p1;
65
66     while (p2 < N && v[p2].first < v[p1].first * R)
67         ++p2;
68     }
69
70     for (int i = 0; i < N; ++i)
71         best = max(best, A[i]);
72 }
73
74 if (best > 1)
75     return best;
76
77 // find out if there is a "subsir geometric" of length 2
78 bitset<MAXVAL> marked;
79 char found = 0;
80
81 for (int i = 0; i < N && !found; ++i)
82 {
83     if (marked[u[i].first]) found = 1;
84     for (int j = 2*u[i].first; j < MAXVAL; j += u[i].first)
85         marked[j] = 1;
86 }
87
88 return found ? 2 : 1;
89 }
90
91 int main()
92 {
93     int T;
94     for (fin >> T; T; --T)
95     {
96         int N;
97         vector< pair<int, int> > v;
98
99         fin >> N;
100        for (int i = 0; i < N; ++i)
101        {
102            int val;
103            fin >> val;
104            v.push_back(make_pair(val, i));
105        }
106
107        fout << Solve(v) << endl;
108    }
109 }
```

Capitolul 31

ONI 2007



Figura 31.1: Sigla ONI 2007

31.1 Apel

Un apel de funcție este un sir de caractere, constituit din numele funcției apelate (o literă mare a alfabetului englez), urmat de lista parametrilor actuali ai funcției, încadrată între paranteze rotunde.

În lista de parametri actuali pot fi 1, 2, ... maximum 10 parametri, separați prin virgulă. Un parametru actual poate fi o constantă (o cifră arabă), o variabilă (o literă mică a alfabetului englez) sau un apel de funcție.

De exemplu: $F(2,a,G(c),G(H(x)))$.

Funcția apelată este F cu 4 parametri. Primul parametru actual este constanta 2, al doilea este variabila a , al treilea este apelul funcției G (funcție cu un singur parametru - variabila c), al patrulea este apelul funcției G (care are ca parametru apelul funcției H).

Numărul de parametri ai unei funcții se numește arititate. O funcție poate fi apelată de ori câte ori, dar de fiecare dată numărul de parametri specificați la apel trebuie să fie egal cu aritatatea funcției.

Fiecare dintre funcțiile care intervin în apel se poate explicita cu ajutorul unei expresii aritmetice sub forma:

$F(a,b,c,...)=$ expresie_aritmetică

Parametrii specificați la explicitarea funcției îi vom denumi parametri formali. Dacă funcția are aritatatea n ($1 \leq n \leq 10$), atunci când explicităm funcția, parametrii formali sunt denumiți utilizând în ordine primele n litere mici ale alfabetului englez. În expresia aritmetică care explicitează funcția apar ca variabile doar parametrii formali ai funcției (denumiți aşa cum am precizat cu primele n litere mici ale alfabetului englez).

Expresia aritmetică ce explicitează o funcție este constituită din unul sau mai mulți termeni separați prin operatorii + (semnificând adunare) sau - (semnificând scădere). Un termen este constituit din unul sau mai mulți factori separați prin operatorul * (semnificând înmulțire). Un factor poate fi o constantă (cifră arabă), o variabilă (un parametru formal al funcției) sau o expresie aritmetică încadrată între paranteze rotunde.

Valoarea obținută în urma unui apel de funcție se obține înlocuind în ordine parametrii formali cu parametrii actuali, apoi efectuând operațiile specificate în expresia aritmetică ce explicitează funcția.

Cerintă

Dacă fiind un apel de funcție, valorile variabilelor care intervin în acest apel, precum și explicitările funcțiilor utilizate în acest apel, să se determine valoarea obținută în urma acestui apel.

Date de intrare

Fișierul de intrare **apel.in** conține pe prima linie sirul de caractere care reprezintă apelul funcției. Pe următoarele linii sunt descrise valorile variabilelor, câte o variabilă pe o linie sub forma:

nume_variabila=valoare

Pe următoarele linii sunt explicitate funcțiile ce intervin în apelul de pe prima linie, sub forma descrisă în enunț.

Date de ieșire

Fișierul de ieșire **apel.out** va conține o singură linie pe care va fi scris un număr întreg reprezentând valoarea obținută în urma apelului din fișierul de intrare.

Restricții și precizări

Orice linie din fișierul de intrare are maximum 250 de caractere.

Valorile variabilelor sunt numere naturale de maximum 3 cifre.

Valoarea obținută în urma apelului funcției este în intervalul [-2 000 000 000, 2 000 000 000].

Exemplu

apel.in	apel.out
F(2,a,G(c),G(H(x)))	
x=3	
a=0	
c=1	
H(a)=2*a-3	
G(a)=2*a*a-5*a+6	
F(a,b,c,d)=a*b*c-2*d*c+4*a*c	

Explicație:

Funcția F are 4 parametri. Primul parametru formal (a) este înlocuit de primul parametru actual (deci are valoarea 2).

Al doilea parametru formal (b) este înlocuit de al doilea parametru actual deci are valoarea variabilei a (adică 0).

Al treilea parametru formal (c) este înlocuit de al treilea parametru actual (apelul G(c)) deci are valoarea $2*1*1-5*1+6=3$.

Al patrulea parametru formal (d) primește valoarea celui de al patrulea parametru actual (apelul G(H(x))) adică $2*H(x)*H(x)-5*H(x)+6=2*(2*x-3)*(2*x-3)-5*(2*x-3)+6=9$.

Deci, valoarea apelului funcției F este: $2*0*3-2*9*3+4*2*3=-30$.

Memorie totală disponibilă: 2 Mb, din care 1 Mb pentru stivă.

Timp maxim de execuție/test: 0.1 secunde pentru Linux și 0.1 secunde pentru Windows

31.1.1 Indicații de rezolvare

Pentru evaluarea unui apel de funcție vom înlocui în ordine parametrii formali cu parametrii actuali, apoi vom evalua expresia aritmetică ce explicitează funcția.

31.1.2 *Rezolvare detaliată

31.1.3 *Cod sursă

31.2 Castel

Înspăimântătorii tăi luptători au răpit-o pe Printesa Ghiocela și au închis-o în castelul tău de pe vârful Muntelui Pleșuv. Deoarece ești un geniu malefic, te-ai hotărât să îi oferi prințesei iluzia unei şanse de evadare.

Castelul tău are forma unui caroaj cu M linii și N coloane. Cele $M \times N$ celule ale castelului sunt numerotate de la 1 la $M \cdot N$ în ordinea parcurgerii caroajului pe linii de sus în jos, iar pe aceeași linie în ordinea coloanelor de la stânga la dreapta.

În fiecare dintre celulele castelului ai pus câte o cheie, mai precis celula i conține cheia cu numărul i .

Evident, pentru a intra într-o cameră, prințesa are nevoie de o anume cheie care permite deschiderea acesteia. Mai mult, dintr-o cameră prințesa se poate deplasa într-un moment numai într-una dintre cele maxim patru camere adiacente pe orizontală și verticală, doar dacă deține cheia necesară deschiderii sale.

Odată ce a intrat într-o cameră și a obținut o cheie, prințesa o păstrează și poate să o utilizeze ori de câte ori dorește.

Cerință

Deși ești convins că prințesa nu va scăpa din castel, ești curios să află câte dintre cele $M \cdot N$ camere îi sunt accesibile. Date fiind dimensiunile castelului, camera în care se află inițial prințesa și cheile necesare deschiderii fiecărei camere, află răspunsul la această întrebare presantă.

Date de intrare

Fișierul de intrare **castel.in** conține pe prima linie trei numere naturale M N K separate prin câte un spațiu reprezentând dimensiunile castelului, respectiv numărul camerei în care se află inițial prințesa. Urmează descrierea castelului. Pe fiecare dintre următoarele M linii se află câte N numere naturale cuprinse între 1 și $M \cdot N$ reprezentând cheile necesare deschiderii fiecărei camere.

Date de ieșire

Fișierul de ieșire **castel.out** va conține o singură linie pe care va fi scris un singur număr natural reprezentând numărul de camere accesibile prințesei.

Restricții și precizări

- $1 \leq M, N \leq 150$
- $1 \leq K \leq M \cdot N$
- Odată ce prințesa a pășit într-o cameră, respectiva cameră va rămâne pentru totdeauna deschisă.

Exemplu

castel.in	castel.out
4 3 1	7
1 1 4	
1 6 2	
6 9 8	
12 10 11	

1	2	3	
1	1		4
4	5	6	
1	6		2
7	8	9	
6	9	8	
10	11	12	
12	10	11	

Figura 31.2: Castel1

Explicații:

Prințesa pornește din camera 1. Aici folosește cheia 1 și intră în camera 4. Se întoarce în camera 1 și descuie camera 2. Folosește cheia luată din camera 4 și descuie camera 3. În acest moment ea detine cheile 1, 2, 3 și 4. Folosește cheia 2 și intră în camera 6, apoi folosește cheia 6 și intră în camera 5, apoi în camera 4, de unde, folosind cheia luată din camera 6 intră în camera 7. La final prințesa are cheile 1, 2, 3, 4, 5, 6, 7 și nu mai poate deschide nici o altă cameră.

Memorie totală disponibilă sub Linux: 2 Mb din care 1 Mb pentru stivă.

Timp maxim de execuție/test: 0.1 secunde pentru Linux și 0.5 secunde pentru Windows

31.2.1 Indicații de rezolvare

In ciuda aparentei simplități a problemei, rezolvarea nu este tocmai trivială. Principala problemă se datorează faptului că, în momentul în care prințesa deschide o cameră și ia cheia aflată acolo, ea poate să se întoarcă și să deschidă camere pe care anterior ar fi "vrut" să le viziteze, dar nu a putut. O parcurgere simplă care nu ține cont de această posibilitate ar primi 10 – 20 de puncte.

O altă posibilitate este un **BF** modificat astfel încât la fiecare pas să se verifice dacă se mai poate deschide vreuna din camerele adiacente cu fiecare dintre nodurile deja procesate. Această soluție are complexitatea $O(N^2 \cdot M^2)$ și obține aproximativ 50 de puncte.

O îmbunătățire constă în menținerea unei liste cu elementele camerele nevizitate care sunt vecine ale celor deschise deja, verificarea de mai sus făcându-se doar pentru aceasta. Soluția de 100 de puncte are complexitatea $O(N \cdot M)$ și se bazează pe următorul algoritm:

$Q = \{K\}$
cât timp $Q \neq \emptyset$
 extrage x din Q
 pentru toate elementele z din $Lst[x]$
 $U[z] = 1$
 $Q = Q \cup \{z\}$
 pentru toți vecinii y ai lui x cu $U[y] = 0$
 dacă $U[A[y]] = 1$
 $u[y] = 1$
 $Q = Q \cup \{y\}$
 altfel
 $Lst[A[y]] = Lst[A[y]] \cup y$

Este posibilă existența unor soluții alternative având aceeași complexitate.

31.2.2 Rezolvare detaliată

Varianta 1:

Listing 31.2.1: castel1.java

```

1 import java.io.*;
2 class castel
3 {
4     static int m,n,p0,nca;      // nca=nr camere accesibile
5     static final int dimq=2000; // 200 coada prea mica pentru t10 ...
6
7     static boolean[] arecheia;
8     static int[][] a;
9     static boolean[][] edeschisa;
10    static boolean[][] amfost;
11
12    static int[] q=new int[dimq]; // coada circulara
13    static final int[] di={-1,+1, 0, 0};
14    static final int[] dj={ 0, 0,-1,+1};

```

```

15
16     static StreamTokenizer st;
17     static PrintWriter out;
18
19     public static void main(String[] args) throws IOException
20     {
21         long t1,t2;
22         t1=System.currentTimeMillis();
23         int i,j,k,iq,sq,d,nr,sql;
24         st=new StreamTokenizer(new BufferedReader(new FileReader("10-castel.in")));
25         out=new PrintWriter(new BufferedWriter(new FileWriter("castel.out")));
26
27         st.nextToken(); m=(int)st.nval;
28         st.nextToken(); n=(int)st.nval;
29         st.nextToken(); p0=(int)st.nval;
30
31         a=new int[m+1][n+1];
32         edeschisa=new boolean[m+1][n+1];
33         amfost=new boolean[m+1][n+1];
34         arecheia=new boolean[m*n+1];
35
36         for(i=1;i<=m;i++)
37             for(j=1;j<=n;j++)
38             {
39                 st.nextToken(); a[i][j]=(int)st.nval;
40             }
41
42         arecheia[p0]=true;
43         nca=1;           // pozitia de start
44         nr=1;
45         while(nr>0)
46         {
47             for(i=1;i<=m; i++) for(j=1; j<=n; j++) amfost[i][j]=false;
48             amfost[ii(p0)][jj(p0)]=true;
49             q[0]=p0;
50             iq=0;
51             sq=1;
52
53             nr=0;
54             while(iq!=sq)
55             {
56                 k=q[iq];
57                 iq=(++iq)%dimq;
58                 for(d=0;d<4;d++)
59                 {
60                     i=ii(k)+di[d]; if(i<1 || i>m) continue;
61                     j=jj(k)+dj[d]; if(j<1 || j>n) continue;
62
63                     if(amfost[i][j]) continue;
64                     amfost[i][j]=true;
65
66                     if(!arecheia[a[i][j]]) continue;
67
68                     if(!edeschisa[i][j])
69                     {
70                         nr++;
71                         nca++;
72                         edeschisa[i][j]=true;
73                         arecheia[kk(i,j)]=true;
74                     }
75                     q[sq]=kk(i,j);
76                     sq=(++sq)%dimq;
77                     //System.out.println(nca+" : "+i+" "+j);
78                     }// for d
79                 }// while
80             }// while nr
81
82             t2=System.currentTimeMillis();
83             System.out.println("nca = "+nca+ "    Time = "+(t2-t1));
84     }// main
85
86     static int ii(int k) { return (k+n-1)/n; }
87
88     static int jj(int k) { return k-(ii(k)-1)*n; }
89
90     static int kk(int i, int j) { return (i-1)*n+j; }

```

```

91     static void afisv(int[] v, int i1, int i2)
92     {
93         int i;
94         System.out.print(" --> ");
95         for(i=i1;i<=i2;i++) System.out.print(v[i]+" ");
96         System.out.println();
97     } // afisv
98
99
100    static void afischei()
101    {
102        int i;
103        for(i=1;i<=m*n;i++) if(arecheia[i]) System.out.print(i+" ");
104        System.out.println();
105    } // afisv
106 } // class

```

Varianta 2:

Listing 31.2.2: castel2.java

```

1 import java.io.*;           // coada cu vecini nerezolvati ...
2 class castel                // t10 ==> Time = 5360 (in Java !)
3 {
4     static int m,n,p0,nca;      // nca=nr camere accesibile
5     static final int dimq=22277; // pentru t10 ...
6     static int sq1max=0, nriter=0;
7
8     static boolean[] arecheia;
9     static int[][] a;
10    static boolean[][] edeschisa;
11
12    static int[] q=new int[dimq]; // coada circulara
13    static int[] q1=new int[dimq]; // coada cu vecini nerezolvati ...
14    static int[] qq;
15
16    static final int[] di={-1,+1, 0, 0};
17    static final int[] dj={ 0, 0,-1,+1};
18
19    static StreamTokenizer st;
20    static PrintWriter out;
21
22    public static void main(String[] args) throws IOException
23    {
24        long t1,t2;
25        t1=System.currentTimeMillis();
26        int i,j,k,iq,sq,d,nr,iql,sq1;
27        boolean ok;
28        st=new StreamTokenizer(new BufferedReader(new FileReader("5-castel.in")));
29        out=new PrintWriter(new BufferedWriter(new FileWriter("castel.out")));
30
31        st.nextToken(); m=(int)st.nval;
32        st.nextToken(); n=(int)st.nval;
33        st.nextToken(); p0=(int)st.nval;
34
35        a=new int[m+1][n+1];
36        edeschisa=new boolean[m+1][n+1];
37        arecheia=new boolean[m*n+1];
38
39        for(i=1;i<=m;i++)
40            for(j=1;j<=n;j++)
41            {
42                st.nextToken(); a[i][j]=(int)st.nval;
43            }
44
45        arecheia[p0]=true;
46        edeschisa[iii(p0)][jj(p0)]=true;
47        nca=1;          // pozitia de start
48        //System.out.println(nca+" : "+ii(p0)+" "+jj(p0));
49        iql=0;
50        sq1=0;
51
52        // pun in coada vecinii lui p0 in q1 = vecinii nerezolvati ai zonei rezolvate ...
53        for(d=0;d<4;d++)
54        {

```

```

55         i=ii(p0)+di[d]; if(i<1 || i>m) continue;
56         j=jj(p0)+dj[d]; if(j<1 || j>n) continue;
57         q1[sq1++]=kk(i,j);
58     }
59
60     nr=1; // nr schimbari (chei adaugate)
61     while(nr>0)
62     {
63         if(sq1>sq1max) sq1max=sq1;
64
65         //System.out.print(" q1 ==> "); afisv(q1,iq1,sq1-1);
66         // q1 <-> q
67         qq=q1; q1=q; q=qq;
68         iq=iq1;
69         sq=sq1;
70         iq1=sq1=0; // coada q1 vida
71
72         //System.out.print(" q ==> "); afisv(q,iq,sq-1);
73         nr=0;
74         while(iq!=sq)
75         {
76             //System.out.println("iq="+iq+" sq="+sq);
77             k=q[iq];
78             iq=(++iq)%dimq;
79
80             i=ii(k);
81             j=jj(k);
82
83             if(edeschisa[i][j])
84             {
85                 //System.out.println("Alta ...");
86                 continue; // intre timp ...
87             }
88
89             if(!arecheia[a[i][j]])
90             {
91                 q1[sq1++]=k; //
92                 //System.out.println("Alta ...");
93                 continue; // alta iteratie in while ... !!!
94             }
95
96             // are cheia
97             nca++;
98             //System.out.println(nca+" : "+i+" "+j);
99             nr++;
100            edeschisa[i][j]=true;
101            arecheia[k]=true;
102
103            // pun in coada vecinii lui ... nerezolvati ... care nu sunt daja in coada ...
104            //!
105            for(d=0;d<4;d++) // daca are vecin nerezolvat ...
106            {
107                i=ii(k)+di[d]; if(i<1 || i>m) continue;
108                j=jj(k)+dj[d]; if(j<1 || j>n) continue;
109
110                if(edeschisa[i][j]) continue;
111
112                q1[sq1]=kk(i,j);
113                sq1=(++sq1)%dimq;
114                //System.out.println("    q1 <-> : "+i+" "+j);
115            } // for d
116
117        } // while coada nevida
118        //System.out.println("nr = "+nr);
119    } // while nr
120
121    t2=System.currentTimeMillis();
122    System.out.println("nca = "+nca+
123        " sq1max = "+sq1max+" nr iter = "+nrIter+" Time = "+(t2-t1));
124 } // main
125 static int ii(int k) { return (k+n-1)/n; }
126 static int jj(int k) { return k-(ii(k)-1)*n; }
127 static int kk(int i, int j) { return (i-1)*n+j; }

```

```

130
131     static void afisv(int[] v, int i1, int i2)
132     {
133         int i;
134         for(i=i1;i<=i2;i++) System.out.print(v[i]+" ");
135         System.out.println();
136     }// afisv
137
138     static void afischei()
139     {
140         int i;
141         for(i=1;i<=m*n;i++) if(arecheia[i])System.out.print(i+" ");
142         System.out.println();
143     }// afisv
144 } // class

```

Varianta 3:

Listing 31.2.3: castel3.java

```

1 import java.io.*;                      // t10 ==> Time = 3891 (in Java !)
2 class castel                         // se mai poate reduce timpul ? (cheie
3     universala + coada heapMIN !!!)
4
5     static int m,n,p0,nca;           // nca=nr camere accesibile
6     static final int dimq=12000;    // 11.177 max pentru t10 ...???
7     static int sq1max=0, nriter=0;
8
9     static boolean[] arecheia;
10    static int[][] a;
11    static boolean[][] edeschisa;
12    static boolean[] eincoada;      // altfel dureaza mult ... !!!
13
14    static int[] q=new int[dimq]; // coada circulara
15    static int[] q1=new int[dimq]; // coada cu vecini nerezolvati ...
16    static int[] qq;
17
18    static final int[] di={-1,+1, 0, 0};
19    static final int[] dj={ 0, 0,-1,+1};
20
21    static StreamTokenizer st;
22    static PrintWriter out;
23
24    public static void main(String[] args) throws IOException
25    {
26        long t1,t2;
27        t1=System.currentTimeMillis();
28        int i,j,k,iq,sq,d,nr,iq1,sq1;
29        boolean ok;
30        st=new StreamTokenizer(new BufferedReader(new FileReader("6-castel.in")));
31        out=new PrintWriter(new BufferedWriter(new FileWriter("castel.out")));
32
33        st.nextToken(); m=(int)st.nval;
34        st.nextToken(); n=(int)st.nval;
35        st.nextToken(); p0=(int)st.nval;
36
37        a=new int[m+1][n+1];
38        edeschisa=new boolean[m+1][n+1];
39        arecheia=new boolean[m*n+1];
40        eincoada=new boolean[m*n+1];
41
42        for(i=1;i<=m;i++)
43            for(j=1;j<=n;j++)
44            {
45                st.nextToken(); a[i][j]=(int)st.nval;
46            }
47
48        arecheia[p0]=true;
49        edeschisa[ii(p0)][jj(p0)]=true;
50        nca=1;          // pozitia de start
51        //System.out.println(nca+" : "+p0);
52        iq1=0;
53        sq1=0;
54

```



```

130         }
131     } // for d
132
133     } // while coada nevida
134     //System.out.println("nr = "+nr);
135 } // while nr
136
137 t2=System.currentTimeMillis();
138
139 System.out.println("nca = "+nca+
140     " sqlmax = "+sqlmax+" nriter = "+nriter+
141     " Time = "+(t2-t1));
142 } // main
143
144 static int ii(int k) { return (k+n-1)/n; }
145
146 static int jj(int k) { return k-(ii(k)-1)*n; }
147
148 static int kk(int i, int j) { return (i-1)*n+j; }
149
150 static void afisv(int[] v, int i1, int i2)
151 {
152     int i;
153     for(i=i1;i<=i2;i++) System.out.print(v[i]+" ");
154     System.out.println();
155 } // afisv
156
157 static void afischei()
158 {
159     int i;
160     for(i=1;i<=m*n;i++) if(arecheia[i]) System.out.print(i+" ");
161     System.out.println();
162 } // afisv
163 } // class

```

Varianta 4:

Listing 31.2.4: castel4.java

```

1 import java.io.*;           // t10 ==> Time = 2281 (in Java !) ...
2 class castel              // se mai poate reduce timpul ? (coada heapMIN !!!)
3 {
4     static int m,n,p0,nca;   // nca=nr camere accesibile
5     static final int dimq=12000; // 11.177 max pentru t10 ...??
6     static int sqlmax=0, nriter=0;
7
8     static int[] cheia;
9     static boolean[] cheieok;    // are deja cheia ...
10    static boolean[] edeschisa;
11    static boolean[] eincoada;   // altfel dureaza mult ... !!!
12
13    static int[] q=new int[dimq]; // coada circulara
14    static int[] ql=new int[dimq]; // coada cu vecini nerezolvati ...
15    static int[] qq;
16
17    static final int[] di={-1,+1, 0, 0};
18    static final int[] dj={ 0, 0,-1,+1};
19
20    static StreamTokenizer st;
21    static PrintWriter out;
22
23    public static void main(String[] args) throws IOException
24    {
25        long t1,t2;
26        t1=System.currentTimeMillis();
27        int i,j,k,iq,sq,d,nr,iql,sql;
28        int vn,ve,vs,vv;           // vecini la nord, est, sud, vest
29        boolean ok;
30        st=new StreamTokenizer(new BufferedReader(new FileReader("10-castel.in")));
31        out=new PrintWriter(new BufferedWriter(new FileWriter("castel.out")));
32
33        st.nextToken(); m=(int)st.nval;
34        st.nextToken(); n=(int)st.nval;
35        st.nextToken(); p0=(int)st.nval;
36

```

```

37     cheia=new int[m*n+1];
38     cheieok=new boolean[m*n+1];           // initial este FALSE
39     edeschisa=new boolean[m*n+1];
40     eincoada=new boolean[m*n+1];
41
42     for(i=1;i<=m;i++)
43         for(j=1;j<=n;j++)
44         {
45             st.nextToken(); cheia[(i-1)*n+j]=(int)st.nval;
46         }
47
48     cheieok[p0]=true;
49     edeschisa[p0]=true;
50     nca=1;          // pozitia de start
51     //System.out.println(nca+" : "+p0);
52     iq1=0;
53     sq1=0;
54
55     // pun in coada vecinii lui p0 in q1 = vecinii nerezolvati ai zonei rezolvate ...
56
57     if(p0<=n) vn=0; else vn=p0-n;
58     if(p0>=(m-1)*n) vs=0; else vs=p0+n;
59     if(p0%n==1) vv=0; else vv=p0-1;
60     if(p0%n==0) ve=0; else ve=p0+1;
61
62     //System.out.println(" Vecinii lui "+p0+" : "+vn+" "+ve+" "+vs+" "+vv);
63
64     if(vn!=0) { q1[sq1]=vn; sq1=(++sq1)%dimq; eincoada[vn]=true; }
65     if(vs!=0) { q1[sq1]=vs; sq1=(++sq1)%dimq; eincoada[vs]=true; }
66     if(vv!=0) { q1[sq1]=vv; sq1=(++sq1)%dimq; eincoada[vv]=true; }
67     if(ve!=0) { q1[sq1]=ve; sq1=(++sq1)%dimq; eincoada[ve]=true; }
68
69     nr=1;      // nr schimbari (chei adaugate)
70     while(nr>0)
71     {
72         nrriter++;
73         if(sq1>sqlmax) sqlmax=sq1;
74         //System.out.print(" q1 ==> "); afisv(q1,iq1,sq1-1);
75         // q1 <--> q
76         qq=q1; q1=q; q=qq;
77         iq=iq1;
78         sq=sq1;
79         iq1=sq1=0;      // coada q1 vida
80
81         //System.out.print("\n\n-----\n\n q ==> "); afisv(q,iq,sq-1);
82         nr=0;
83         while(iq!=sq)
84         {
85             //System.out.println("iq="+iq+" sq="+sq);
86             k=q[iq];
87             iq=(++iq)%dimq;
88             eincoada[k]=false;
89
90             //System.out.println("               q1 --> : "+k);
91
92             if(edeschisa[k])
93             {
94                 System.out.println("E deschisa ...");
95                 continue; // intre timp ...
96             }
97
98             if(!cheieok[cheia[k]])
99             {
100                 //System.out.println(k+"Nu are cheia ...");
101                 q1[sq1]=k;           // il pun din nou in coada ...
102                 sq1=(++sq1)%dimq;
103                 eincoada[k]=true;
104                 //System.out.println("               q1 <-- : "+k);
105
106                 continue;          // alta iteratie in while ... !!!
107             }
108
109             // are cheia
110             //System.out.println("Are cheia ...");
111             nca++;
112             //System.out.println(nca+" : "+k);

```

```

113         nr++;
114         edeschisa[k]=true;
115         cheieok[k]=true;
116
117         // pun in coada vecinii lui ... nerezolvati ... care nu sunt daja in coada ...
118         //!
119
120         if(k<=n) vn=0; else vn=k-n;
121         if(k>=(m-1)*n) vs=0; else vs=k+n;
122         if(k%n==1) vv=0; else vv=k-1;
123         if(k%n==0) ve=0; else ve=k+1;
124         //System.out.println(" Vecinii lui "+k+" : "+vn+" "+ve+" "+vs+" "+vv);
125
126         if(vn!=0)
127             if(!edeschisa[vn] && !eincoada[vn])
128                 { q1[sql]=vn; sql=(++sql)%dimq; eincoada[vn]=true; }
129         if(vs!=0)
130             if(!edeschisa[vs] && !eincoada[vs])
131                 { q1[sql]=vs; sql=(++sql)%dimq; eincoada[vs]=true; }
132         if(vv!=0)
133             if(!edeschisa[vv] && !eincoada[vv])
134                 { q1[sql]=vv; sql=(++sql)%dimq; eincoada[vv]=true; }
135         if(ve!=0)
136             if(!edeschisa[ve] && !eincoada[ve])
137                 { q1[sql]=ve; sql=(++sql)%dimq; eincoada[ve]=true; }
138
139         }// while coada nevida
140         //System.out.println("nr = "+nr);
141     }// while nr
142
143     t2=System.currentTimeMillis();
144     System.out.println("nca = "+nca+
145         " sqlmax = "+sqlmax+" nrriter = "+nrriter+" Time = "+(t2-t1));
146 } // main
147
148 static void afisv(int[] v, int i1, int i2)
149 {
150     int i;
151     for(i=i1;i<=i2;i++) System.out.print(v[i]+" ");
152     System.out.println();
153 } // afisv
154 } // class

```

Varianta 5:

Listing 31.2.5: castel5.java

```

1 import java.io.*;                                // t10 ==> Time = 2375 (in Java !) ...
2 class castel                                    // nu se mai castiga timp ... ba chiar ... !!!
3 {
4     static int m,n,p0,nca;                      // nca=nr camere accesibile
5     static final int dimq=12000;
6
7     static int[] cheia;
8     static boolean[] cheieok;                    // are deja cheia ...
9     static boolean[] edeschisa;
10    static boolean[] eincoada;                  // altfel dureaza mult ... !!!
11
12    static int[] q=new int[dimq];   // coada circulara
13
14    static StreamTokenizer st;
15    static PrintWriter out;
16
17    public static void main(String[] args) throws IOException
18    {
19        long t1,t2;
20        t1=System.currentTimeMillis();
21        int i,j,k,iq,sq,d,nr,sql,nriter=0;
22        int vn,ve,vs,vv;                          // vecini la nord, est, sud, vest
23
24        st=new StreamTokenizer(new BufferedReader(new FileReader("10-castel.in")));
25        out=new PrintWriter(new BufferedWriter(new FileWriter("castel.out")));
26
27        st.nextToken(); m=(int)st.nval;
28        st.nextToken(); n=(int)st.nval;

```

```

29     st.nextToken(); p0=(int)st.nval;
30
31     cheia=new int[m*n+1];
32     cheieok=new boolean[m*n+1];           // initial este FALSE
33     edeschisa=new boolean[m*n+1];
34     eincoada=new boolean[m*n+1];
35
36     for(i=1;i<=m;i++)
37         for(j=1;j<=n;j++)
38         {
39             st.nextToken(); cheia[(i-1)*n+j]=(int)st.nval;
40         }
41
42     cheieok[p0]=true;
43     edeschisa[p0]=true;
44     nca=1;                         // pozitia de start
45     iq=sq=0;
46
47     // pun in coada vecinii lui p0 in q1 = vecinii nerezolvati ai zonei rezolvate ...
48     if(p0<=n) vn=0; else vn=p0-n;
49     if(p0>=(m-1)*n) vs=0; else vs=p0+n;
50     if(p0%n==1) vv=0; else vv=p0-1;
51     if(p0%n==0) ve=0; else ve=p0+1;
52
53     if(vn!=0) { q[sq]=vn; sq=(++sq)%dimq; eincoada[vn]=true; }
54     if(vs!=0) { q[sq]=vs; sq=(++sq)%dimq; eincoada[vs]=true; }
55     if(vv!=0) { q[sq]=vv; sq=(++sq)%dimq; eincoada[vv]=true; }
56     if(ve!=0) { q[sq]=ve; sq=(++sq)%dimq; eincoada[ve]=true; }
57
58     nr=1; // nr schimbari (chei adaugate)
59     while(nr>0)
60     {
61         nrriter++;
62         sq1=sq;
63         nr=0;
64         while(iq!=sq1)
65         {
66             k=q[iq];
67             iq=(++iq)%dimq;
68             eincoada[k]=false;
69
70             if(edeschisa[k]) continue; // intre timp ... !!!
71
72             if(!cheieok[cheia[k]])
73             {
74                 q[sq]=k;           // il pun din nou in coada ...
75                 sq=(++sq)%dimq;
76                 eincoada[k]=true;
77                 continue;        // alta iteratie in while ... !!!
78             }
79
80             // are cheia
81             nca++;
82             nr++;
83             edeschisa[k]=true;
84             cheieok[k]=true;
85
86             // pun in coada vecinii lui ... nerezolvati ... care nu sunt daja in coada ...
87             //!
88             if(k<=n) vn=0; else vn=k-n;
89             if(k>=(m-1)*n) vs=0; else vs=k+n;
90             if(k%n==1) vv=0; else vv=k-1;
91             if(k%n==0) ve=0; else ve=k+1;
92             if(vn!=0) if(!edeschisa[vn] && !eincoada[vn])
93                 { q[sq]=vn; sq=(++sq)%dimq; eincoada[vn]=true; }
94             if(vs!=0) if(!edeschisa[vs] && !eincoada[vs])
95                 { q[sq]=vs; sq=(++sq)%dimq; eincoada[vs]=true; }
96             if(vv!=0) if(!edeschisa[vv] && !eincoada[vv])
97                 { q[sq]=vv; sq=(++sq)%dimq; eincoada[vv]=true; }
98             if(ve!=0) if(!edeschisa[ve] && !eincoada[ve])
99                 { q[sq]=ve; sq=(++sq)%dimq; eincoada[ve]=true; }
100
101            }// while coada nevida
102       }// while nr
103
t2=System.currentTimeMillis();
System.out.println("nca = "+nca+" nrriter = "+nrriter+" Time = "+(t2-t1));

```

```
104 } // main
105 } // class
```

Varianta 6:

Listing 31.2.6: castel6.java

```

1 import java.io.*;                                // t10 ==> Time = 2062 (in Java !) ...
2 class castel                                // cu liste de camere adiacente; coada=index
3 {
4     static int m,n,p0,nca,iq,sq;    // nca=nr camere accesibile
5     static final int dimq=12000;
6
7     static int[][] a;                  // liste de chei adiacente camerelor deschise
8
9     static int[] cheia;
10    static boolean[] cheieok;        // are deja cheia ...
11    static boolean[] edeschisa;
12    static boolean[] eincoada;       // cheia de deschidere
13    static boolean[] einlista;       // camera!
14
15    static int[] q=new int[dimq];   // coada circulara de chei distinste de deschidere
16
17    static StreamTokenizer st;
18    static PrintWriter out;
19
20    public static void main(String[] args) throws IOException
21    {
22        long t1,t2;
23        t1=System.currentTimeMillis();
24        int i,j,k,d,nr,sql,nriter=0,cheiad,camera;
25
26        st=new StreamTokenizer(new BufferedReader(new FileReader("10-castel.in")));
27        out=new PrintWriter(new BufferedWriter(new FileWriter("castel.out")));
28
29        st.nextToken(); m=(int)st.nval;
30        st.nextToken(); n=(int)st.nval;
31        st.nextToken(); p0=(int)st.nval;
32
33        a=new int[m*n+1][1];           // a[i][0]=nr camere cu cheia i de deschidere
34        cheia=new int[m*n+1];
35        cheieok=new boolean[m*n+1]; // initial este FALSE
36        edeschisa=new boolean[m*n+1];
37        eincoada=new boolean[m*n+1];
38        einlista=new boolean[m*n+1];
39
40        for(i=1;i<=m;i++)
41            for(j=1;j<=n;j++)
42            {
43                st.nextToken(); cheiad=(int)st.nval;
44                cheia[(i-1)*n+j]=cheiad;
45                ++a[cheiad][0];
46            }
47
48        for(i=1;i<=m*n;i++) a[i]=new int[a[i][0]+1]; // spatiul maxim alocat ...
49
50        cheieok[p0]=true;
51        edeschisa[p0]=true;
52        nca=1;          // pozitia de start
53        iq=sq=0;
54
55        // pun in coada vecinii lui p0
56        puninlista(vecinnord(p0));
57        puninlista(vecinsud(p0));
58        puninlista(vecinvest(p0));
59        puninlista(vecinest(p0));
60
61        nr=1;             // nr schimbari (chei adaugate)
62        while(nr>0)
63        {
64            nriter++;
65            sq1=sq;
66            nr=0;
67            while(iq!=sq1)
```

```

68      {
69          k=q[iq];                      // scot din coada o cheie de deschidere
70          iq=(++iq)%dimq;
71          eincoada[k]=false;
72
73          if(!cheieok[k])
74          {
75              q[sq]=k;                  // din nou in coada ... la sfarsit ...
76              sq=(++sq)%dimq;
77              eincoada[k]=true;
78              continue;
79          }
80
81          // caut in lista ... camerele adiacente care se deschid cu cheia k
82
83          for(j=a[k][0];j>=1;)
84          {
85              camera=a[k][j];
86              nca++;
87              nr++;
88              edeschisa[camera]=true;
89              cheieok[camera]=true;
90              --a[k][0];
91
92              // pun in coada vecinii ... nerezolvati
93              puninlista(vecinnord(camera));
94              puninlista(vecinsud(camera));
95              puninlista(vecinvest(camera));
96              puninlista(vecinest(camera));
97              j=a[k][0];
98          }// for 1 lista k
99      }// while coada nevida
100 }// while nr
101
102 t2=System.currentTimeMillis();
103 System.out.println("nca = "+nca+"  nrIter = "+nrIter+"  Time = "+(t2-t1));
104 }// main
105
106 static int vecinnord(int k) { return (k<=n) ? (0) : (k-n); }
107 static int vecinsud (int k) { return (k>=(m-1)*n) ? (0) : (k+n); }
108 static int vecinvest (int k) { return (k%n==1) ? (0) : (k-1); }
109 static int vecinest (int k) { return (k%n==0) ? (0) : (k+1); }
110
111 static void puninlista(int camera)
112 {
113     int cheiad;
114     if(camera!=0)
115     {
116         if(edeschisa[camera]) return;
117         if(einlista[camera]) return;
118
119         cheiad=cheia[camera];
120         if(!eincoada[cheiad])
121         {
122             q[sq]=cheiad;                      // cheia de deschidere pentru "camera";
123             sq=(++sq)%dimq;
124             eincoada[cheiad]=true;
125         }
126         einlista[camera]=true;               // inca o camera cu cheia de deschidere "
127         ++a[cheiad][0];                     " cheiad"
128         a[cheiad][a[cheiad][0]]=camera;    // pun "camera" in lista
129     }
130 }// puninlista(...)
131 }// class

```

31.2.3 Cod sursă

Listing 31.2.7: castel.cpp

```

1 #include <stdio.h>
2 #include <vector>

```

```

3
4  using namespace std;
5
6 #define MAX_N 180
7 #define FIN "castel.in"
8 #define FOUT "castel.out"
9
10 const int di[] = {-1,+1, 0, 0};
11 const int dj[] = { 0, 0,-1,+1};
12
13 int N, M, K, A[MAX_N][MAX_N], Q[MAX_N*MAX_N], Res;
14 char U[MAX_N][MAX_N], key[MAX_N*MAX_N];
15 vector<int> key_list[MAX_N*MAX_N];
16
17 int main(void)
18 {
19     int n, i, j, d, ii, jj, ql, qr;
20     vector<int>::iterator it;
21
22     freopen(FIN, "r", stdin);
23     freopen(FOUT, "w", stdout);
24
25     scanf("%d %d %d", &N, &M, &K);
26     for (i = 0; i < N; i++)
27         for (j = 0; j < M; j++)
28         {
29             scanf("%d", A[i]+j);
30             A[i][j]--;
31         }
32
33     Q[ql = qr = 0] = --K;
34     U[K/M][K%M] = 1;
35     for (; ql <= qr; ql++)
36     {
37         // am cheia N
38         key[n = Q[ql]] = 1; Res++;
39         for (it = key_list[n].begin(); it != key_list[n].end(); it++)
40             if (!U[*it/M][*it%M])
41             {
42                 Q[++qr] = *it;
43                 U[*it/M][*it%M] = 1;
44             }
45         i = n/M; j = n%M;
46         for (d = 0; d < 4; d++)
47         {
48             ii = i+di[d], jj = j+dj[d];
49             if (ii < 0 || jj < 0 || ii >= N || jj >= M || U[ii][jj]) continue;
50             if (key[A[ii][jj]])
51             {
52                 Q[++qr] = ii*M+jj;
53                 U[ii][jj] = 1;
54             }
55             else key_list[A[ii][jj]].push_back(ii*M+jj);
56         }
57     }
58
59     printf("%d\n", Res);
60
61     return 0;
62 }

```

31.3 Excusie

Gigel este un mare amator de excursii la munte. În același timp este și un bun informatician. El a observat că făcând un traseu între două obiective turistice obosită mai puțin decât dacă alege un alt traseu între aceleași obiective. Gigel și-a propus să găsească un model care să-i permită determinarea unui traseu pe care, dacă-l alege, va ajunge la destinație cât mai puțin obosit. Astfel, el reprezintă terenul în care se află cele două obiective turistice printr-un tablou bidimensional cu n linii (numerotate de la 1 la n) și m coloane (numerotate de la 1 la m), cu elemente numere naturale strict pozitive, în care fiecare element reprezintă cota unei zone de teren de forma unui pătrat cu latura $1m$. Efortul pe care-l face pentru a trece dintr-o zonă cu cota $c1$ într-o zonă vecină

cu o cotă mai înaltă (c_2) se calculează după cum urmează. Se trasează un triunghi dreptunghic ca în figură:

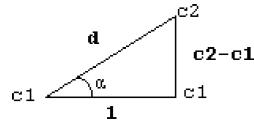


Figura 31.3: Excursie

unde:

- c_1 și c_2 sunt cele două cote, $c_1 < c_2$
- 1 este distanța dintre centrele celor două zone vecine
- $d = \sqrt{(c_2 - c_1)^2 + 1}$
- α este unghiul pantei care trebuie urcată

Apoi calculează efortul astfel:

$$ef = d \cdot \operatorname{tg} \alpha$$

În exemplul următor considerăm patru zone vecine având cotele 1, 2, 6, 10.

10	6
1	2

Pentru a ajunge din zona de cotă 1 în zona de cotă 10 se pot alege două trasee:

1. direct, ceea ce presupune un efort calculat astfel:

$$ef = d * \operatorname{tg} \alpha = \sqrt{89} \cdot 9 \approx 81$$

2. ocolit, prin zonele de cote 2 și 6, ceea ce presupune un efort calculat astfel:

$$ef = ef_1 + ef_2 + ef_3 = \sqrt{2} + \sqrt{17} \cdot 4 + \sqrt{17} \cdot 4 \approx 34$$

Efortul pe care-l face pentru a trece dintr-o zonă având cota c_1 într-o zonă vecină cu aceeași cotă este 1.

Efortul pe care-l face pentru a trece dintr-o zonă având cota c_1 într-o zonă vecină cu o cotă mai joasă (c_2) este jumătate din efortul pe care l-ar face la urcare (adică de la cota c_2 la cota c_1).

Cerință

Scriți un program care să determine efortul minim pentru a ajunge de la un obiectiv turistic la altul, lungimea traseului nedepășind o valoare dată L_{max} .

Date de intrare

Fișierul de intrare **excursie.in** conține:

- pe prima linie două numere naturale n și m separate printr-un spațiu, reprezentând dimensiunile terenului;
- pe linia a doua numărul real L_{max} reprezentând lungimea maximă admisă a drumului;
- următoarele n linii conțin fiecare câte m valori naturale, separate prin spațiu, reprezentând în ordine cotele zonelor de teren;
- ultima linie conține patru valori naturale $li \ ci \ lf \ cf$, separate prin câte un spațiu, unde li , ci reprezintă linia și respectiv coloana punctului de plecare, iar $lf \ cf$ reprezintă linia și respectiv coloana punctului de sosire.

Date de ieșire

Fișierul de ieșire **excursie.out** va conține pe prima linie două numere reale separate printr-un spațiu $ef \ d$, reprezentând efortul minim depus pentru a ajunge de la un obiectiv la altul și respectiv lungimea minimă a unui drum parcurs cu efort minim. Rezultatele vor fi afișate cu câte trei zecimale.

Restricții și precizări

- $2 \leq n, m \leq 50$
- Deplasarea dintr-o zonă în alta se poate face doar în 4 direcții: (N, E, S, V). Mai exact, dacă poziția curentă este pe linia i , coloana j , prin deplasare la N se trece în poziția $(i-1, j)$, la E în $(i, j+1)$, la S în $(i+1, j)$, iar la V în $(i, j-1)$ (dacă aceste poziții există).
- Cotele sunt numere naturale cu valori între 1 și 100.

- Se recomandă utilizarea tipurilor reale pe 64 biți. Rezultatul va fi considerat corect dacă diferența absolută dintre rezultatul afișat și rezultatul corect este < 0.01 .
- Se acordă 60% din punctaj pentru determinarea corectă a efortului minim, respectiv 100% pentru rezolvarea corectă a ambelor cerințe.

Exemplu

excursie.in	excursie.out
2 2	34.399 9.660
11	
10 6	
1 2	
2 1 1 1	

Explicație:

$$\sqrt{2} + \sqrt{17} \cdot 8 \approx 34.399(1.41421356 + 32.98484500 \approx 34.39905856)$$

$$\sqrt{2} + \sqrt{17} \cdot 2 \approx 9.660(1.41421356 + 8.24621125 \approx 9.66042481)$$

Traseul este corect deoarece lungimea drumului 9.660 este mai mică decât valoarea dată $L_{max} = 11$.

Memorie maximă disponibilă: 2 Mb, din care 1 Mb pentru stivă.

Timp maxim de execuție/test: 0.1 secunde pentru Linux și 0.2 secunde pentru Windows

31.3.1 Indicații de rezolvare

Definim tabloul ef , de dimensiuni $n \cdot m$, cu semnificația: $ef[i][j]$ este efortul minim necesar pentru a ajunge la linia i și coloana j , pe un traseu care pornește din poziția inițială.

Tabloul se initializează cu valori foarte mari, mai puțin valoarea corespunzătoare poziției de plecare, care este 0.

Starea curentă este definită de: efortul minim de la poziția de start la cea actuală, poziția în matrice și distanța față de punctul de plecare.

Se expandeză starea curentă.

Se încearcă obținerea unui efort mai mic până la una dintre pozițiile adiacente, pe un traseu care trece prin starea curentă.

La același efort minim obținut, starea vecină se actualizează, dacă se obține o distanță mai mică.

Pentru memorarea stărilor curente și a celor adiacente, se pot utiliza două cozi. În prima coadă se introduce starea inițială (practic doar pozițiile i și j). Stările adiacente, care suferă actualizări, se introduc în coada a doua. Când toate stările din prima coadă s-au expandat, se suprascriu valorile din prima coadă cu cele din coada a doua. Acestea din urmă devin astfel stări curente și se reia procedeul.

Se afișează $ef[lf][cf]$.

O implementare de tip *backtracking* poate obține 50 – 70% din punctaj.

31.3.2 Rezolvare detaliată

Listing 31.3.1: excursie.java

```

1 import java.io.*;
2 class excursie
3 {
4     static StreamTokenizer st;
5     static PrintWriter out;
6
7     static final int[] di={-1, 0, 1, 0};
8     static final int[] dj={ 0, 1, 0, -1};
9     static final double EPS=0.00001;
10    static final int DIM=52;
11    static final double INF=(double)10.0*Integer.MAX_VALUE;
12
13    static double[][] a=new double[DIM][DIM];      // a[i][j] - distantele
14    static int[][] c=new int[DIM][DIM];            // cotele
15    static double[][] ef=new double[DIM][DIM];
16

```

```

17     static int ip, jp, is, js;                                // plecare, sosore
18     static int m, n;
19     static double Lmax;
20     static int[][] c1=new int[20*DIM][2];                  // cozile
21     static int[][] c2=new int[20*DIM][2];                  // cozile
22     static int[][] caux;                                    // pentru interschimbare
23
24 public static void main(String[] args) throws IOException
25 {
26     st=new StringTokenizer(new BufferedReader(new FileReader("19-excursie.in")));
27     out=new PrintWriter(new BufferedWriter(new FileWriter("excursie.out")));
28
29     Read();
30     Init();
31     Solve();
32     Write();
33     out.close();
34 } // main(...)
35
36 static void Read() throws IOException
37 {
38     int i, j;
39     st.nextToken(); n=(int)st.nval;
40     st.nextToken(); m=(int)st.nval;
41     st.nextToken(); Lmax=(int)st.nval;
42
43     for(i=1; i<=n; i++)
44         for(j = 1; j <= m; j++)
45         {
46             st.nextToken(); c[i][j]=(int)st.nval;
47         }
48     st.nextToken(); ip=(int)st.nval;
49     st.nextToken(); jp=(int)st.nval;
50     st.nextToken(); is=(int)st.nval;
51     st.nextToken(); js=(int)st.nval;
52 } // Read(...)
53
54 static void Init()
55 {
56     int i=0, j=0;
57     for(i=1; i<= n; i++)
58         for(j=1; j<= m; j++)
59             ef[i][j]=a[i][j]=INF;
60
61     a[ip][jp]=0.;
62     ef[ip][jp]=0.;
63 } // Init(...)
64
65 static double Dist(int i1, int j1, int i2, int j2)
66 {
67     return Math.sqrt((c[i2][j2]-c[i1][j1])*(c[i2][j2]-c[i1][j1])+1.);
68 } // Dist(...)
69
70 static boolean Ok(int i1, int j1, int i2, int j2)
71 {
72     if(i2<1 || i2>n || j2<1 || j2>m ) return false;
73     if(a[i1][j1]+Dist(i1, j1, i2, j2)>Lmax ) return false;
74     return true;
75 } // Ok(...)
76
77 static double Efort(int i1, int j1, int i2, int j2)
78 {
79     double dif, e;
80     if(c[i1][j1]==c[i2][j2]) return 1.;
81     dif=c[i2][j2]-c[i1][j1];
82     e=dif*Math.sqrt(dif*dif+1);
83     return e > 0 ? e : -(e / 2);
84 } // Efort(...)
85
86 static void Solve()
87 {
88     int i,j,d,s,k=1,l; // k = nr elem din "coada 1", l nr elem din "coada 2"
89     double Ef;
90     c1[k][0]=ip;
91     c1[k][1]=jp;
92 }
```

```

93     for( ;k>0; k=k1,caux=c1,c1=c2,c2=caux) // cat timp coada 1 nu e vida
94     {
95         for(l=0,s=1; s<= k; s++ ) // parcurg prima coada
96             for(i=c1[s][0],j=c1[s][1],d=0; d<4; d++)
97                 if(Ok(i,j,i+di[d],j+dj[d]))
98                 {
99                     Ef=ef[i][j]+Efort(i,j,i+di[d],j+dj[d]);
100                    if(ef[i+di[d]][j+dj[d]]> Ef || // efort mai mic
101                        Math.abs(ef[i+di[d]][j + dj[d]]-Ef)<EPS &&
102                        a[i+di[d]][j+dj[d]]>a[i][j]+Dist(i,j,i+di[d],j+dj[d])) // efort egal
103                    {
104                        ef[i+di[d]][j+dj[d]]=Ef;
105                        a[i+di[d]][j+dj[d]]=a[i][j]+Dist(i,j,i+di[d],j+dj[d]);
106                        c2[+1][0]=i+di[d];
107                        c2[1][1]=j+dj[d];
108                    } // if
109                } // if Ok
110            } // for k
111        } // Solve(...)

112
113    static void Write()
114    {
115        int efin=0, dfin=0;
116        efin=(int)(ef[is][js]*1000.0);
117        dfin=(int)(a[is][js]*1000.0);
118        out.println((double)efin/1000.+" "+(double)dfin/1000.);
119    } // Write
120 } // class

```

31.3.3 Cod sursă

Listing 31.3.2: excursie.cpp

```

1 //C. Galatan
2 #include <stdio.h>
3 #include <string.h>
4 #include <math.h>
5 #include <limits.h>
6
7 #define IN "excursie.in"
8 #define OUT "excursie.out"
9 #define EPS 0.00001
10
11 //##define __DEBUG__
12
13 const int DIM = 52;
14 const double INF = 10.0*INT_MAX;
15 const int di[] = { -1, 0, 1, 0 },
16             dj[] = { 0, 1, 0, -1 };
17
18 double a[DIM][DIM]; // a[i][j] - distantele
19 int c[DIM][DIM]; // cotele
20 double ef[DIM][DIM];
21
22 int ip, jp, is, js; // plecare, sosore
23 int m, n;
24 double Lmax;
25 int c1[20*DIM][2], c2[20*DIM][2]; // cozile
26
27 void Read();
28 int Ok(int, int);
29 void Init();
30 void Solve();
31 void Write();
32
33 int main()
34 {
35     freopen(OUT, "w", stdout);
36     Read();
37     Init();
38     Solve();
39     Write();

```

```

40     return 0;
41 }
42
43 void Read()
44 {
45     freopen(IN, "r", stdin);
46     scanf("%d %d %lf", &n, &m, &Lmax);
47     for ( int i = 1; i <= n; i++ )
48         for ( int j = 1; j <= m; j++ )
49             scanf("%d", &c[i][j]); // cotele
50
51     scanf("%d%d%d%d", &ip, &jp, &is, &js);
52 }
53
54 void Init()
55 {
56     int i = 0, j = 0;
57     for ( i = 1; i <= n; i++ )
58         for ( j = 1; j <= m; j++ )
59             ef[i][j] = a[i][j] = (double)INF;
60
61     a[ip][jp] = 0.;
62     ef[ip][jp] = 0.;
63
64 #ifdef __DEBUG__
65     printf("%ld %lf %d %d \n", INF, a[ip][jp], ip, jp);
66     for ( int i = 1; i <= n; i++ )
67     {
68         for ( int j = 1; j <= m; j++ )
69             printf("%20.f", a[i][j]); // cotele
70         printf("\n");
71     }
72 #endif
73 }
74
75 double Dist(int i1, int j1, int i2, int j2)
76 {
77     return sqrt((c[i2][j2] - c[i1][j1]) * (c[i2][j2] - c[i1][j1]) + 1.);
78 }
79
80 int Ok(int i1, int j1, int i2, int j2)
81 {
82     if ( i2 < 1 || i2 > n || j2 < 1 || j2 > m ) return 0;
83     if ( a[i1][j1] + Dist(i1, j1, i2, j2) > Lmax ) return 0;
84     return 1;
85 }
86
87 double Efort(int i1, int j1, int i2, int j2)
88 {
89     if ( c[i1][j1] == c[i2][j2] ) return 1.;
90     double dif = c[i2][j2] - c[i1][j1];
91     double e = dif * sqrt(dif * dif + 1);
92
93     return e > 0 ? e : -(e / 2);
94 }
95
96 // pozitia vecina
97 #define iv (i + di[d])
98 #define jv (j + dj[d])
99
100 void Solve()
101 {
102     int i,j,d,s,k = 1, l;// k - nr elem din coada 1, l nr elem din coada 2
103     double Ef;
104     c1[k][0] = ip;
105     c1[k][1] = jp;
106
107     for ( ;k>0; k=l, memcpy(c1, c2, sizeof(c2)))// cat timp coada 1 nu e vida
108         for ( l = 0, s = 1 ; s <= k; s++ ) // parcurg prima coada
109             for ( i = c1[s][0], j = c1[s][1], d = 0; d < 4; d++)
110                 if (Ok(i, j, iv, jv))
111                 {
112                     Ef = ef[i][j] + Efort(i, j, iv, jv);
113                     if (ef[iv][jv] > Ef || // efort mai mic
114                         fabs(ef[iv][jv] - Ef) < EPS &&
115                         a[iv][jv] > a[i][j]+Dist(i, j, iv, jv))// efort egal

```

```

116             {
117                 ef[iv][jv] = Ef;
118                 a[iv][jv] = a[i][j] + Dist(i, j, iv, jv);
119                 c2[+1][0] = iv;
120                 c2[1][1] = jv;
121             }
122         }
123     }
124
125 void Write()
126 {
127     long efin=0, dfin=0;
128     efin = ef[is][js] * 1000.0;
129     dfin = a[is][js] * 1000.0;
130     printf("%.3lf %.3lf\n", efin/1000., dfin/1000.);
131
132 #ifdef __DEBUG__
133     for ( int i = 1; i <= n; i++ )
134     {
135         for ( int j = 1; j <= m; j++ )
136             printf("%7.3d", ef[i][j]== INF ? 100 : ef[i][j]);           // cotele
137         printf("\n");
138     }
139 #endif
140 }
```

31.4 Matrice

Se consideră o matrice binară B (cu valori 0 sau 1) cu N linii și M coloane, liniile și coloanele fiind numerotate de la 1 la N , respectiv de la 1 la M .

Matricea B este generată după regula $B_{i,j} = R_i \text{ xor } C_j$, unde R și C sunt vectori binari de lungime N , respectiv M .

Numim dreptunghi de colțuri $(x_1, y_1)(x_2, y_2)$ cu $x_1 \leq x_2$ și $y_1 \leq y_2$, mulțimea elementelor $B_{i,j}$ cu $x_1 \leq i \leq x_2$ și $y_1 \leq j \leq y_2$. Aria unui astfel de dreptunghi este $(x_2 - x_1 + 1) \cdot (y_2 - y_1 + 1)$.

Cerintă

Determinați numărul maxim de elemente egale cu 0 într-un dreptunghi a cărui arie este exact A , precum și numărul de dreptunghiuri pentru care se obține acest număr maxim.

Date de intrare

Fișierul de intrare **matrice.in** conține pe prima linie numerele naturale N, M, A separate prin câte un spațiu. A doua linie va conține N valori 0 sau 1 separate prin câte un spațiu, reprezentând elementele vectorului R , iar a treia linie va conține M valori 0 sau 1 separate prin câte un spațiu, reprezentând elementele vectorului C .

Date de ieșire

Fișierul de ieșire **matrice.out** va conține o singură linie pe care vor fi scrise două numere naturale separate printr-un singur spațiu $Zmax Nsol$, reprezentând în ordine numărul maxim de elemente egale cu 0 într-un dreptunghi a cărui arie este exact A , precum și numărul de dreptunghiuri pentru care se obține acest număr maxim.

Restricții și precizări

- $1 \leq N, M \leq 30000$
- $1 \leq A \leq 5000000$
- Pentru 40% din teste $N, M \leq 300$
- Prin x xor y se înțelege operația sau exclusiv, mai exact:

	0	1
0	0	1
1	1	0

Exemplu

matrice.in	matrice.out
2 4 4 0 1 1 0 0 1	2 5

Explicație

Matricea B:

1	0	0	1
0	1	1	0

Numărul maxim de valori 0 dintr-un dreptunghi de arie 4 este 2.

Cele 5 dreptunghiuri de arie 4 cu număr maxim de 0 sunt:

- (1,1)(1,4)
- (2,1)(2,4)
- (1,1)(2,2)
- (1,2)(2,3)
- (1,3)(2,4)

Memorie totală disponibilă sub Linux: 2 Mb din care 1 Mb pentru stivă.

Timp maxim de execuție/test: 0.5 secunde pentru Linux și 2 secunde pentru Windows

31.4.1 Indicații de rezolvare

1. Se observă că lungimea unui dreptunghi posibil este un divizor d al lui A (iar lătimea A/d).

Divizorii numarului A se pot genera în $O(A)$ sau $O(\sqrt{A})$, și se observă ca numărul lor nu depășește 384 pentru $A \leq 5.000.000$, astfel încât se pot încerca toate perechile $(d, A/d)$ de laturi.

2. Pentru o pereche de laturi (P, Q) vom determina dreptunghiul de marime $P * Q$ cu număr maxim de 0-uri în timp $O(N+M)$. Pentru un dreptunghi $(x, y) (x+P-1, y+Q-1)$, fie ZR numarul de 0-uri din subvectorul $R[x \dots x+P-1]$, iar ZC numarul de 0-uri din subvectorul $C[y \dots y+Q-1]$. Numarul de 0-uri din dreptunghiul respectiv este $f(ZR, ZC) = ZR * ZC + (P - ZR) * (Q - ZC)$.

Presupunem că avem fixată coordonata x .

Puteam determina coordonata optimă y pentru care numărul de 0-uri din dreptunghiul $(x, y) (x+P-1, y+Q-1)$ este maximă. Fie y_1, y_2 două valori candidate, și ZC_1 , respectiv ZC_2 numărul de 0-uri din subvectorul C determinat de fiecare din cele două valori.

$$f(ZR, ZC_1) \geq f(ZR, ZC_2) \iff$$

$$ZR * ZC_1 + (P - ZR) * (Q - ZC_1) \geq ZR * ZC_2 + (P - ZR) * (Q - ZC_2) \iff$$

$$2 * ZR * ZC_1 - P * ZC_1 \geq 2 * ZR * ZC_2 - P * ZC_2 \iff$$

$$ZC_1 * (2 * ZR - P) \geq ZC_2 * (2 * ZR - P)$$

Astfel, dacă $2 * ZR - P = 0$ atunci nu contează valorile ZC_1, ZC_2 , dacă $2 * ZR - P > 0$ atunci pentru un x fixat ne vor interesa coordonatele y care maximizează numărul de 0-uri din $C[y \dots y+Q-1]$, iar dacă $2 * ZR - P < 0$ ne vor interesa coordonatele y care minimizează numărul de 0-uri din $C[y \dots y+Q-1]$.

3. Folosind observațiile de mai sus algoritmul de rezolvare se poate descrie astfel în pseudocod:

pentru fiecare divizor d al lui A execută

$$\text{minZ} = \text{numărul minim de 0-uri din orice secvență de lungime } A/d \text{ din } C$$

$$\text{maxZ} = \text{numărul maxim de 0-uri din orice secvență de lungime } A/d \text{ din } C$$

$$\text{nz} = \text{numărul de 0-uri din } R[1 \dots d];$$

pentru $x = 1, N + 1 - d$ execută

$$\text{actualizează soluția cu } f(\text{nz}, \text{minZ});$$

$$\text{actualizează soluția cu } f(\text{nz}, \text{maxZ});$$

$$\text{dacă } R[x] = 0 \text{ atunci } \text{nz} = \text{nz} - 1;$$

$$\text{dacă } R[x + d] = 0 \text{ atunci } \text{nz} = \text{nz} + 1;$$

Numărarea soluțiilor nu reprezintă o dificultate deosebită, dar pot exista diverse cazuri pe care concurenții nu le iau în calcul.

31.4.2 Rezolvare detaliată

Listing 31.4.1: matrice.java

```

1 import java.io.*;
2 class matrice
3 {

```

```

4   static StreamTokenizer st;
5   static PrintWriter out;
6
7   static final int MAX_N=30005;
8
9   static int N, M;
10  static byte[] R=new byte[MAX_N];
11  static byte[] C=new byte[MAX_N];
12  static int A, P, Q, Zmax=-1, Nsol;
13
14 public static void main(String[] args) throws IOException
15 {
16     st=new StreamTokenizer(new BufferedReader(new FileReader("matrice.in")));
17     out=new PrintWriter(new BufferedWriter(new FileWriter("matrice.out")));
18
19     int i, min_z, max_z, nmin, nmax, nz;
20     int d;
21
22     st.nextToken(); N=(int)st.nval;
23     st.nextToken(); M=(int)st.nval;
24     st.nextToken(); A=(int)st.nval;
25
26     for(i=0; i<N; i++) { st.nextToken(); R[i]=(byte)st.nval; }
27     for(i=0; i<M; i++) { st.nextToken(); C[i]=(byte)st.nval; }
28
29     for(d=1; d<=A && d<=N; d++)
30     {
31         if(A%d!=0 || A/d>M) continue;
32         P=d;
33         Q=A/d;
34
35         min_z=M+1;
36         nmin=0;
37         max_z=-1;
38         nmax=0;
39         for(nz=i=0; i<M; i++)
40         {
41             nz+=1-C[i];
42             if(i>=Q) nz-=1-C[i-Q];
43             if(i<Q-1) continue;
44
45             if(min_z>nz) { min_z= nz; nmin=1; }
46             else
47                 if(min_z==nz) nmin++;
48
49             if(max_z<nz) { max_z=nz; nmax=1; }
50             else
51                 if(max_z==nz) nmax++;
52         } // for i
53
54         for(nz=i=0; i<N; i++)
55         {
56             nz+=1-R[i];
57             if(i>=P) nz-=1-R[i-P];
58             if(i<P-1) continue;
59
60             if(2*nz==P) update(nz,min_z,M-Q+1,P,Q);
61             else
62                 if(2*nz>P) update(nz,max_z,nmax,P,Q);
63                 else
64                     update(nz,min_z,nmin,P,Q);
65         } // for i
66     } // for d
67
68     System.out.println(Zmax+" "+Nsol);
69     out.println(Zmax+" "+Nsol);
70     out.close();
71 } // main...
72
73 static void update(int zr, int zc, int sol, int P, int Q)
74 {
75     int z=zr*zc+(P-zr)*(Q-zc);
76     if(Zmax < z) { Zmax=z; Nsol=sol; }
77     else
78         if(Zmax==z) Nsol+=sol;
79 } // update...

```

```

80     static int min(int a, int b)
81     {
82         return (a<b) ? a : b;
83     }
84
85     static int max(int a, int b)
86     {
87         return (a>b) ? a : b;
88     }
89 } // class

```

31.4.3 Cod sursă

Listing 31.4.2: matrice.cpp

```

1 // sursa oficiala - O(sqrt(A)*(N+M)) - ar trebui sa mearga si pe Borland :
2 #pragma -ml -O2
3 #include <stdio.h>
4
5 #define MAX_N 30005
6 #define FIN "matrice.in"
7 #define FOUT "matrice.out"
8 #define min(a, b) ((a) < (b) ? (a) : (b))
9 #define max(a, b) ((a) > (b) ? (a) : (b))
10
11 int N, M;
12 char R[MAX_N], C[MAX_N];
13 long A, P, Q, Zmax = -1, Nsol;
14
15 inline void update(int zr, int zc, int sol, int P, int Q)
16 {
17     long z = (long)zr*zc + (long)(P-zr)*(Q-zc);
18     if (Zmax < z)
19         Zmax = z, Nsol = sol;
20     else
21         if (Zmax == z)
22             Nsol += sol;
23 }
24
25 int main(void)
26 {
27     int i, min_z, max_z, nmin, nmax, nz;
28     long d;
29
30     freopen(FIN, "r", stdin);
31     freopen(FOUT, "w", stdout);
32
33     scanf("%d %d %ld", &N, &M, &A);
34     for (i = 0; i < N; i++) scanf("%d", R+i);
35     for (i = 0; i < M; i++) scanf("%d", C+i);
36
37     for (d = 1; d <= A && d <= N; d++)
38     {
39         if (A%d || A/d > M) continue;
40         P = d; Q = A/d;
41
42         min_z = M+1; nmin = 0; max_z = -1; nmax = 0;
43         for (nz = i = 0; i < M; i++)
44         {
45             nz += !C[i];
46             if (i >= Q) nz -= !C[i-Q];
47             if (i < Q-1) continue;
48             if (min_z > nz)
49                 min_z = nz, nmin = 1;
50             else
51                 if (min_z == nz)
52                     nmin++;
53                 if (max_z < nz)
54                     max_z = nz, nmax = 1;
55                 else
56                     if (max_z == nz)
57                         nmax++;

```

```

58         }
59
60     for (nz = i = 0; i < N; i++)
61     {
62         nz += !R[i];
63         if (i >= P) nz -= !R[i-P];
64         if (i < P-1) continue;
65         if (2*nz == P)
66             update(nz, min_z, M-Q+1, P, Q);
67         else
68             if (2*nz > P)
69                 update(nz, max_z, nmax, P, Q);
70             else
71                 update(nz, min_z, nmin, P, Q);
72     }
73 }
74
75 printf("%ld %ld\n", Zmax, Nsol);
76
77 return 0;
78 }
```

31.5 Rânduri

Andrei, un Tânăr cu un real talent literar, va recita la următoarea întâlnire a cenaclului literar ultimul său poem. O singură problemă are Andrei: poemul are prea multe rânduri și este conștient că nimeni nu va avea răbdare să-l asculte până la capăt. Pentru că nu mai are timp să-l rescrie, Andrei s-a hotărât să eliminate rânduri din poem. Totuși, nu va elimina rânduri la întâmplare, ci le va alege astfel încât valoarea artistică a poemului să nu se diminueze. După îndelungi frământări, poetul a descoperit criteriul de eliminare: un rând se poate elimina doar dacă imediat înaintea sa se află un rând (să-l notăm *p*), iar imediat după el se află un alt rând (să-l notăm *u*) astfel încât rândurile *p* și *u* au *muzicalitatea* strict mai mare decât 6. Nu întrebăți de ce 6, doar Andrei știe de ce.

Ce înțelege poetul prin *muzicalitatea* a două rânduri ?

Fie rândurile:

alinuta este acasa

și

alin merge cu noi.

Muzicalitatea este egală cu 9, adică exact numărul caracterelor subliniate:

alin ee c.

Așadar, *muzicalitatea* reprezintă dimensiunea celei mai lungi succesiuni formată din caractere ce apar în ordinea din succesiune atât în primul rând, cât și în cel de al doilea, pe poziții nu neapărat consecutive.

Astfel, rândul care este precedat, respectiv urmat de acestea două poate fi eliminat, pentru că $9 > 6$.

Evident, dacă eliminăm rândul existent între *p* și *u*, rândurile *p* și *u* devin consecutive.

Cerință

Determinați numărul maxim de rânduri care pot fi eliminate, respectând criteriul stabilit de poet.

Date de intrare

Fișierul de intrare **randuri.in** conține rândurile poemului, câte unul pe linie.

Date de ieșire

Fișierul de ieșire **randuri.out** va conține o singură linie pe care va fi scris numărul maxim de rânduri care pot fi eliminate, respectând criteriul stabilit de poet.

Restricții și precizări

- Rândurile sunt formate din maxim 100 caractere cu codul ASCII < 127.

- Nu există rânduri goale.
- Fișierul de intrare conține cel mult 100 de rânduri.
- Orice linie din fișierul de intrare se termină cu marcajul de sfârșit de linie (newline). Caracterul newline nu va fi considerat ca făcând parte din rând.

	randuri.in	randuri.out
Exemplu	Te-nalta pana-n nori Tot mai sus, Tot mai departe Ca siragul de cocori	2

Explicație:

Muzicalitatea rândurilor:

Te-nalta pana-n nori
Tot mai departe

este 7 (vezi caracterele subliniate). Prin urmare poate fi eliminat rândul

Tot mai sus,

După eliminarea acestui rând, obținem:

Te-nalta pana-n nori
 Tot mai departe
 Ca siragul de cocori

Muzicalitatea rândurilor:

Te-nalta_pana-n_nori
Ca_siragul_de_cocori

este 7. Prin urmare, se poate elimina și rândul:

Tot mai departe

Memorie totală disponibilă sub Linux: 2 Mb, din care 1 Mb pentru stivă.**Timp maxim de execuție/test:** 0.1 secunde pentru Linux și 0.5 secunde pentru Windows**31.5.1 Indicații de rezolvare**

Soluția propusă se bazează pe *programare dinamică*. Pentru fiecare pereche de rânduri i și j , determinăm $E[i][j]$ - numărul maxim de rânduri cuprinse între rândurile i și j , care se pot elmina. Pentru aceasta, se utilizează optimele obținute pe toate intervalele mai mici: $[i, k]$ și $[k + 1, j]$, reținându-se maximul acestora.

Distingem două cazuri: rândul k nu se elimină, iar atunci

$$E[i][j] = E[i][k] + E[k + 1][j],$$

deci este suma dintre maximul eliminărilor pe intervalul $[i, k]$ și $[k + 1, j]$. Al doilea caz, presupune că și k se poate elmina. Aceasta se întamplă numai dacă se pot elmina toate rândurile cuprinse între i și k , și toate rândurile cuprinse între $k + 1$ și j și în plus, cel mai lung subșir comun rândurilor i și j trebuie să fie mai mare decât 6. În această situație, la $E[i][j]$ de la primul caz, se adaugă valoarea 1, corespunzătoare eliminării rândului k .

O altă soluție bazată tot pe programare dinamică este următoarea:

se calculează o matrice booleană $ok[i][j]$ cu semnificația că $ok[i][j] = true$ dacă există posibilitatea de a elmina toate rândurile din intervalul $[i...j]$, mai puțin rândurile i și j .Matricea se poate calcula în $O(N^3)$ astfel:ok[i][j]=true daca există cel puțin un k astfel încât $ok[i][k]$ și $ok[k][j]$ să fie *true* și cel mai lung subșir comun între i și j să fie mai mare ca 6.Odată calculată această matrice se poate face o altă dinamică cu semnificația ca $nr[i] =$ numărul maxim de rânduri care se pot elmina din primele i rânduri, mai puțin rândurile 1 și i . Relația de recurență pentru această dinamică este $nr[i] = \max(nr[j] + j - i - 1)$ pentru $0 < j < i$ și $ok[j][i] = true$.

31.5.2 Rezolvare detaliată

Listing 31.5.1: randuri.java

```

1 import java.io.*;
2 class randuri
3 {
4     static BufferedReader br;
5     static PrintWriter out;
6
7     static final int MAX_ROWS=102;
8     static final int MAX_COLS=102;
9
10    // E[i][j] = nr max de randuri care se pot elimina in intervalul [i,j]
11    static short[][] E=new short[MAX_ROWS][MAX_ROWS];
12    static byte[][] R=new byte[MAX_ROWS][MAX_COLS];           // randurile
13    static int L;
14    static boolean[][] SC=new boolean[MAX_ROWS][MAX_ROWS];
15
16    public static void main(String[] args) throws IOException
17    {
18        Read();
19        Precompute();
20        EraseRows();
21        Write();
22    } // main(...)

23
24    static void Precompute()
25    {
26        int i,j;
27        for(i=0; i<L; i++)
28            for(j=i; j<L; j++)
29                SC[i][j]=Common(R[i],R[j])>6;
30    } // Precompute(...)

31
32    static int Common(byte a[], byte b[])      // cel mai lung subsir comun
33    {
34        int i,j;
35        // C[i][j] = cel mai lung subsir comun secventelor a[i..m-1] si b[j..n-1]
36        int[][] C=new int[MAX_COLS][MAX_COLS];      // implicit = 0
37        int m=a.length;
38        int n=b.length;
39        for(i=m-1; i>=0; i--)
40            for(j=n-1; j>=0; j--)
41                if(a[i]==b[j])
42                    C[i][j]=C[i+1][j+1]+1;
43                else
44                    C[i][j]=max(C[i+1][j],C[i][j+1]);
45        return C[0][0];
46    } // Common(...)

47
48    static int max(int x,int y)
49    {
50        return (x>y)?x:y;
51    } // max(...)

52
53    static void EraseRows()
54    {
55        int i,j,k,d,max,erased;
56
57        for(i=0; i<L; i++) // initializari; pentru a putea sterge un rand,
58            E[i][i]=0;          // intervalul [i,...j] trebuie sa contina cel putin 3 randuri
59        for(i=0; i<L-1; i++)
60            E[i][i+1]=0;
61
62        for(d=2; d<L; d++)
63            for(i=0; i<L-d ; i++)
64            {
65                j=i+d;
66                max=0;
67                for(k=i+1; k<=j-1; k++)
68                {
69                    erased=E[i][k]+E[k][j];
70                    if(SC[i][j] && erased==d-2) erased++;
71                }
72            }
73    }

```

```

71         if(erased>max) max=erased;
72     } // for k
73     E[i][j]=(short)max;
74   } // for i, d
75 } // EraseRow(...)

76
77 static void Read() throws IOException
78 {
79     br=new BufferedReader(new FileReader("randuri.in"));
80     String s;
81     for(L=0,s=br.readLine(); s!=null; L++,s=br.readLine())
82     {
83         R[L]=s.getBytes();
84         System.out.print(L+" : ");
85     } // for
86 } // Read
87
88 static void afisv(byte[] x)
89 {
90     int i;
91     for(i=0;i<x.length;i++) System.out.print((char)x[i]);
92     System.out.println();
93 } // afisv(...)

94
95 static void Write() throws IOException
96 {
97     out=new PrintWriter(new BufferedWriter(new FileWriter("randuri.out")));
98     System.out.println(E[0][L-1]);
99     out.println(E[0][L-1]);
100    out.close();
101 } // Write(...)

102 } // class

```

31.5.3 Cod sursă

Listing 31.5.2: randuri.cpp

```

1 #include <stdlib.h>
2 #include <string.h>
3 #include <fstream>
4
5 using namespace std;
6
7 #define MAX_ROWS 102
8 #define MAX_COLS 102
9 ifstream fin("randuri.in");
10 ofstream fout("randuri.out");
11 #define max(x,y) ((x)>(y)?(x):(y))
12 unsigned char E[MAX_ROWS][MAX_ROWS]; // E[i][j] - nr max de randuri care
13 // se pot elimina in intervalul [i, j]
14 char R[MAX_ROWS][MAX_COLS]; // randurile
15 int L;
16 int SC[MAX_ROWS][MAX_ROWS];
17
18 int Common(char a[], char b[]) // cel mai lung subsir comun
19 {
20     unsigned char C[MAX_COLS][MAX_COLS]={0}; // C[i][j] - cel mai lung subsir
21 // comun secentelor a[i..m-1] si b[j..n-1]
22     int m = strlen(a), n = strlen(b);
23     for (int i = m - 1; i >= 0; i--)
24         for (int j = n - 1; j >= 0; j--)
25             if( a[i] == b[j] )
26                 C[i][j] = C[i+1][j+1] + 1;
27             else
28                 C[i][j] = max(C[i+1][j], C[i][j+1]);
29     return C[0][0];
30 }
31
32 void Precompute()
33 {
34     for (int i = 0; i < L; i++)
35         for (int j = i; j < L; j++)

```

```

36     SC[i][j] = Common(R[i], R[j]) > 6;
37 }
38
39 void EraseRows()
40 {
41     int i, j, k, d, max, erased;
42
43     for (i = 0; i < L; i++) // initializari; pentru a putea sterge un rand,
44         E[i][i]=0; // intervalul [i,..j] trebuie sa contine cel putin 3 randuri
45     for (i = 0; i < L - 1; i++)
46         E[i][i + 1] = 0;
47
48     for (d = 2; d < L; d++)
49         for (i = 0; i < L - d ; i++)
50         {
51             j = i + d;
52             max = -1;
53             for (k = i + 1; k <= j - 1; k++)
54             {
55                 int erased = E[i][k] + E[k][j];
56                 if ( SC[i][j] && erased == d - 2 ) erased++;
57                 if ( erased > max ) max = erased;
58             }
59             E[i][j] = max;
60         }
61     }
62
63 void Read()
64 {
65     for ( L = 0; fin.getline(R[L], 252); L++ );
66     fin.close();
67 }
68
69 void Write()
70 {
71     fout << int(E[0][L - 1]) << '\n';
72     fout.close();
73 }
74
75 int main()
76 {
77     Read();
78     Precompute();
79     EraseRows();
80     Write();
81     return 0;
82 }
```

31.6 Zidar

Nu se ştie de ce, ai decis subit să începi o carieră în construcții. Zidurile pe care le construiești sunt formate din cărămizi cubice de latură 1, așezate în mai multe straturi.

Pentru a proiecta un astfel de zid, ai trasat un caroiaj format din $M \times N$ pătrate de latură 1, organizate în M linii și N coloane. Liniile caroiajului sunt numerotate de la 1 la M , începând de jos în sus, iar coloanele sunt numerotate de la 1 la N de la stânga la dreapta.

Fiecare căsuță a caroiajului are asociat un anumit cost, care trebuie plătit în cazul în care plasăm o cărămidă în căsuță respectivă. Costul construirii unui zid este egal cu suma costurilor căsuțelor în care sunt plasate cărămizile zidului.

Zidurile tale trebuie să respecte următoarele condiții:

1. fiecare strat de cărămizi este format dintr-o singură secvență de cărămizi, oricare două cărămizi consecutive fiind adiacente (mai exact, cărămizile de pe un strat sunt plasate în căsuțe ale caroiajului situate pe aceeași linie, pe coloane consecutive);

2. cel puțin o cărămidă de pe fiecare strat i trebuie să fie așezată pe o altă cărămidă de pe stratul de dedesubt ($i - 1$); cel mai de jos strat trebuie să fie așezat pe pământ (pământul fiind sub linia 1 a caroiajului);

3. numărul de cărămizi folosit în construcția zidului nu trebuie să depășească un număr natural X .

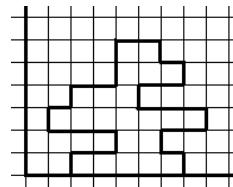


Figura 31.4: Zid valid

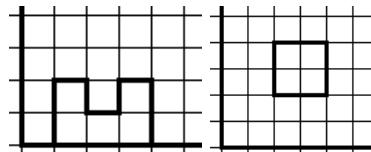


Figura 31.5: Ziduri invalide

Cerință

Fiind un zidar dornic de afirmare și știind că dispui de o sumă de T euro, calculează numărul maxim de cărămizi pe care le poți folosi în construcția unui zid care costă cel mult T euro.

Date de intrare

Fișierul de intrare **zidar.in** conține pe prima linie patru numere naturale $M \ N \ X \ T$ separate prin câte un spațiu cu semnificația din enunț. Pe fiecare dintre următoarele M linii se află câte N numere naturale cuprinse între 1 și 100 reprezentând costul așezării unei cărămizi pentru fiecare dintre căsuțele caroiajului (mai precis, elementul j de pe a $(i+1)$ -a linie a fișierului de intrare reprezintă costul așezării unei cărămizi în căsuța de pe linia i și coloana j a caroiajului).

Date de ieșire

Fișierul de ieșire **zidar.out** va conține o singură linie pe care va fi scris un singur număr natural reprezentând numărul maxim de cărămizi pe care îl poate conține zidul tău, respectând condițiile impuse.

Restricții și precizări

- $1 \leq M \leq 50$
- $1 \leq N \leq 20$
- $1 \leq X \leq 60$
- $1 \leq T \leq 10000$

Pentru 30% din teste $T \leq 60$. Pentru 60% din teste $N \leq 10$.

Exemplu

zidar.in	zidar.out
4 5 20 8	6
2 2 3 2 1	
4 7 1 2 3	
2 1 1 1 1	
1 2 5 7 3	

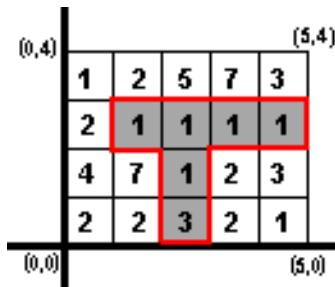


Figura 31.6: Zidar4

Explicație:

Pentru a construi zidul marcat în figură ai nevoie de exact 8 euro. Nu se pot construi ziduri cu mai multe cărămizi folosind această sumă de bani.

Memorie totală disponibilă: 2 Mb, din care 1 Mb pentru stivă.

Timp maxim de execuție/test: 0.1 secunde pentru Linux și 3 secunde pentru Windows

31.6.1 Indicații de rezolvare

Această problemă se rezolvă folosind *metoda programării dinamice*.

O soluție trivială este calcularea matricei 4-dimensionale $M[i][j][k][l] =$ numărul maxim de cărămizi pe care îl poate conține un zid al cărui al i -lea strat este cuprins între coloanele j și k , având prețul total de construcție egal cu l . Din păcate, l poate deveni suficient de mare cât să nu permită o rezolvare eficientă.

Îmbunătățirea adusă este schimbarea punctului de vedere, și anume calcularea matricei $M[i][j][k][l] =$ prețul minim al unui zid al cărui al i -lea strat este cuprins între coloanele j și k , folosind l cărămizi.

Soluția va fi valoarea l maximă (evident, mai mică sau egală cu X) pentru care există un $M[i][j][k][l] \leq T$.

Pentru aceasta, o soluție banală de complexitatea $O(M * N^4 * X)$ obține aproximativ $X\%$ din punctaj.

Menținând pentru fiecare nivel o matrice aditională $Z[i][j] =$ costul minim al unui zid care se termină pe nivelul respectiv, conține cărămida $\#i$ și conține în total j cărămizi, se poate ajunge la o soluție de complexitate $O(M * N^3 * X)$ care obține punctajul maxim.

De notat că este posibilă reducerea memoriei folosite de la $O(M * N^2 * X)$ la $O(N^2 * X)$, folosind doar ultima "linie" a matricei 4-dimensionale de mai sus.

31.6.2 *Rezolvare detaliată

31.6.3 Cod sursă

Listing 31.6.1: zidar.cpp

```

1 // complexitate: O(M*N^3*X)
2 #ifdef __BORLANDC__
3 #pragma option -3 -r -z -O2
4 #endif
5
6 #include <stdio.h>
7 #include <stdlib.h>
8 #include <string.h>
9
10 #define MMAX 50
11 #define NMAX 20
12 #define RMAX 60
13
14 #ifndef __BORLANDC__
15 #define huge
16 #define _fmemset memset
17 #define _fmemcpy memcpy
18 #endif
19
20 short huge m[NMAX][NMAX][RMAX], l[NMAX][NMAX][RMAX],
21         z[NMAX][RMAX], a[MMAX][NMAX];
22 int M, N, X, T, i, j, k, nr, s, v, Ans = 0;
23
24 int main(void)
25 {
26     freopen("zidar.in", "r", stdin);
27     freopen("zidar.out", "w", stdout);
28     scanf("%d %d %d %d", &M, &N, &X, &T);
29     for (i = 0; i < M; i++)
30         for (j = 0; j < N; j++)
31             scanf("%d", &a[i][j]);

```

```

32
33     if (X > M*N) X = M*N;
34
35     if (M > MMAX) fprintf(stderr, "M is out of range\n");
36     if (N > NMAX) fprintf(stderr, "N is out of range\n");
37     if (X > RMAX) fprintf(stderr, "R is out of range\n");
38
39     _fmemset(m, 0xFF, sizeof(m));
40     _fmemset(z, 0xFF, sizeof(z));
41
42     for (i = 0; i < N; i++)
43         for (j = i; j < N && (v = j-i+1) <= X; j++)
44         {
45             m[i][j][v] = j == i ? a[0][i] : m[i][j-1][v-1]+a[0][j];
46             if (m[i][j][v] <= T && v > Ans)
47                 Ans = v;
48
49             //update z
50             for (k = i; k <= j; k++)
51                 if (z[k][v] == -1 || z[k][v] > m[i][j][v])
52                     z[k][v] = m[i][j][v];
53         }
54
55     for (int stp = 1; stp < M; stp++)
56     {
57         _fmemcpy(l, m, sizeof(m));
58         _fmemset(m, 0xFF, sizeof(m));
59
60         for (i = 0; i < N; i++)
61             for (s = 0, j = i; j < N; j++)
62             {
63                 s += a[stp][j], v = j-i+1;
64                 for (nr = stp+v; nr <= N*stp+v && nr <= X; nr++)
65                 {
66                     //calculez m[i][j][nr]
67                     for (k = i; k <= j; k++)
68                         if ((z[k][nr-v] != -1) &&
69                             (m[i][j][nr] == -1 ||
70                             m[i][j][nr] > z[k][nr-v]+s))
71                         {
72                             m[i][j][nr] = z[k][nr-v]+s;
73                             if (m[i][j][nr] <= T && nr > Ans)
74                                 Ans = nr;
75                         }
76                 }
77             }
78
79             //update z
80             _fmemset(z, 0xFF, sizeof(z));
81
82             for (i = 0; i < N; i++)
83                 for (j = i; j < N; j++)
84                 {
85                     v = j-i+1;
86                     for (nr = stp+v; nr <= N*stp+v && nr <= X; nr++)
87                         if (m[i][j][nr] != -1)
88                         {
89                             for (k = i; k <= j; k++)
90                                 if (z[k][nr] == -1 || z[k][nr] > m[i][j][nr])
91                                     z[k][nr] = m[i][j][nr];
92                         }
93                 }
94
95             printf("%d\n", Ans);
96             return 0;
97
98 }
```

Capitolul 32

ONI 2006



Figura 32.1: Sigla ONI 2006

32.1 Bombo

Gigel este un băiat foarte pofticios. El are acasă N stive, fiecare conținând câte M cutii cu bomboane. La începutul fiecărei zile, Gigel ia o singură cutie din vârful uneia dintre stive și mănâncă instantaneu toate bomboanele din ea, după care o aruncă la gunoi (deoarece o cutie cu bomboane fără bomboane în ea îl deprimă). El execută această activitate (de a mâncă toate bomboanele din cutia din vârful unei stive) în fiecare zi, până când se golesc toate stivele.

Gigel ar fi foarte mulțumit dacă numărul de bomboane din fiecare cutie ar rămâne constant, până când ar ajunge el la cutia respectivă pentru a mâncă bomboanele. Realitatea, însă, nu corespunde dorințelor lui Gigel. Fiecare cutie cu bomboane este caracterizată de doi parametri: Z și B . Inițial (la începutul primei zile), cutia conține $Z * B$ bomboane. La sfârșitul fiecărei zile, numărul de bomboane din cutie scade cu B . După Z zile, numărul de bomboane din cutie devine 0. Când numărul de bomboane dintr-o cutie devine 0, se întâmplă ceva spectaculos: cutia dispare, iar toate cutile cu bomboane de deasupra ei, dacă ea nu este, în acel moment, cutia din vârful stivei în care se află, coboară cu o poziție mai jos în stivă; dacă ea se află în vârful stivei, dar este și ultima cutie din stivă, atunci stiva se golește; dacă ea se află în vârful stivei și stiva conține și alte cutii cu bomboane, atunci cutia de sub ea devine noua cutie din vârful stivei (în cazul în care această cutie dispare și ea în aceeași zi, se consideră cutia de dedesubtul acesteia și.a.m.d.).

Cerință

Cunoscând numărul de stive, numărul de cutii de bomboane din fiecare stivă și parametrii Z și B pentru fiecare cutie de bomboane, determinați care este numărul maxim total de bomboane pe care le poate mâncă Gigel.

Datele de intrare

Prima linie a fișierului de intrare **bombo.in** conține numerele întregi N și M . Următoarele N linii conțin câte M perechi de numere, descriind cutile de bomboane din fiecare stivă, de la bază către vârful stivei. O astfel de pereche conține numerele Z și B , având semnificația precizată mai

sus. Oricare două numere de pe aceeași linie din fișierul de intrare sunt separate printr-un singur spațiu.

Datele de ieșire

În fișierul **bombo.out** veți afișa un singur număr, reprezentând numărul maxim de bomboane pe care le poate mâncă Gigel.

Restricții și precizări

- $1 \leq N \leq 4$
- $1 \leq M \leq 10$
- Pentru fiecare cutie de bomboane:
 - $1 \leq Z \leq 50$
 - $1 \leq B \leq 1000000$
- Cel puțin 30% din fișierele de test vor avea $N \leq 2$
- Cel puțin 65% din fișierele de test vor avea $M \leq 6$
- Cel puțin 55% din fișierele de test vor avea pentru fiecare cutie cu bomboane $Z > N * M$

Exemple

bombo.in	bombo.out	bombo.in	bombo.out
2 3		4 6	32
50 1000 1 3 1 100	51100	3 1 2 2 3 3 2 4 2 5 2 6	
2 3000 1 10 1 20		2 1 3 2 2 3 3 4 1 5 3 6	
		1 1 2 2 2 3 2 4 3 5 1 6	
		2 1 2 2 3 3 2 4 2 5 2 6	

Timp maxim de rulare/test: 0.7 secunde pentru Windows și 0.2 secunde pentru Linux

32.1.1 Indicații de rezolvare

Mugurel Ionuț Andreica - Universitatea Politehnica, București

Problema se rezolvă folosind programare dinamică. Se va calcula o matrice

$BMAX[T][C_1][C_2][C_3][C_4]$ = numărul maxim de bomboane pe care le poate mâncă Gigel dacă ajunge după T zile în "starea" în care mai există C_1 cutii în prima stivă, C_2 cutii în a doua stivă, C_3 în a treia și C_4 în a patra (în cazul în care există mai puțin de 4 stive, valorile corespunzătoare pentru numărul de cutii se vor considera ca fiind 0).

Vom calcula $BMAX[T][C_1][C_2][C_3][C_4]$ pe baza valorilor pentru :

- $BMAX[T-1][C_1+1][C_2][C_3][C_4]$, în cazul în care cutia $C_1 + 1$ nu a dispărut până la momentul $T-1$, respectiv $BMAX[T][C_1+1][C_2][C_3][C_4]$, în cazul în care cutia a dispărut
- similar pentru cutiile $C_2 + 1$, $C_3 + 1$ și $C_4 + 1$

Relațiile de calcul sunt ușor de dedus. Se observă că pentru un anumit moment de timp T , sunt necesare numai valorile pentru momentele de timp $T-1$ și T (dar din alte stări). Astfel, memoria folosită poate fi de ordinul $2 * 11^4$ (11, deoarece sunt necesare valorile de la 0 la 10 inclusiv).

32.1.2 Rezolvare detaliată

Listing 32.1.1: bombo.java

```

1 import java.io.*;
2 class Bombo
3 {
4     static StreamTokenizer st;
5     static PrintWriter out;
6
7     static final int NMAX=4;
8     static final int MMAX = 10;

```

```

9   static final int TMAX = NMAX * MMAX;
10  static final int MIC = -1;
11
12  static int[][][] bmax=new int[2][MMAX+1][MMAX+1][MMAX+1];
13  static int[][] z=new int[MMAX+1][MMAX+1];
14  static int[][] b=new int[MMAX+1][MMAX+1];
15
16  static int[] count=new int[NMAX+1];
17  static int[] x=new int[NMAX+1];
18  static int[] y=new int[NMAX+1];
19
20  static int i, j, k, N, M, i1, i2, i3, i4, t, c, a;
21  static int vmax, tt, tmi, tma, bo;
22
23  public static void main(String[] args) throws IOException
24  {
25      st=new StreamTokenizer(new BufferedReader(new FileReader("15-bombo.in")));
26      out=new PrintWriter(new BufferedWriter(new FileWriter("bombo.out")));
27
28      st.nextToken(); N=(int)st.nval;
29      st.nextToken(); M=(int)st.nval;
30
31      for(i=1;i<=N;i++)
32          for(j=1;j<=M; j++)
33          {
34              st.nextToken(); z[i][j]=(int)st.nval;
35              st.nextToken(); b[i][j]=(int)st.nval;
36          }
37
38      for(i=1;i<=N;i++) count[i]=M;
39      for(i=N+1;i<=NMAX;i++) count[i]=0;
40
41      c=0;
42      for(i1=count[1];i1>=0;i1--)
43          for(i2=count[2];i2>=0;i2--)
44              for(i3=count[3];i3>=0;i3--)
45                  for(i4=count[4];i4>=0;i4--)
46                  {
47                      bmax[c][i1][i2][i3][i4]=MIC;
48                  }
49
50      bmax[c][count[1]][count[2]][count[3]][count[4]]=0;
51      vmax=0;
52
53      tmi=TMAX+1;
54      tma=0;
55
56      for(t=1;t<=TMAX;t++)
57      {
58          a=c;
59          c=1-c;
60
61          for(i1=count[1];i1>=0;i1--)
62              for(i2=count[2];i2>=0;i2--)
63                  for(i3=count[3];i3>=0;i3--)
64                      for(i4=count[4];i4>=0;i4--)
65                      {
66                          bmax[c][i1][i2][i3][i4]=MIC;
67                          x[1]=i1; x[2]=i2; x[3]=i3; x[4]=i4;
68
69                          for(i=1;i<=NMAX;i++) y[i]=x[i];
70
71                          for(i=1;i<=N;i++)
72                              if(x[i]<count[i])
73                              {
74                                  y[i]=y[i]+1;
75                                  bo=MIC;
76                                  if(t-1<z[i][y[i]])
77                                  {
78                                      if(bmax[a][y[1]][y[2]][y[3]][y[4]]> bo)
79                                          bo=bmax[a][y[1]][y[2]][y[3]][y[4]]+(z[i][y[i]]-(t-1))*b[i][y[i]];
80                                  }
81
82                                  if(t>=z[i][y[i]])
83                                  {
84                                      if(bmax[c][y[1]][y[2]][y[3]][y[4]]>bo)

```

```

85         bo=bmax[c][y[1]][y[2]][y[3]][y[4]];
86     }
87
88     if(bmax[c][x[1]][x[2]][x[3]][x[4]]<bo)
89         bmax[c][x[1]][x[2]][x[3]][x[4]]=bo;
90
91     y[i]=y[i]-1;
92 }
93 }
94
95 if(bmax[c][0][0][0]>vmax)
96 {
97     vmax=bmax[c][0][0][0];
98     tt=t;
99 }
100
101 if(bmax[c][0][0]>MIC)
102 {
103     if(t<tmi) tmi=t;
104     if(t>tma) tma=t;
105 }
106 } // for t
107
108 //System.out.println(vmax+" -> "+tmi+" - "+tt+" - "+ tma);
109
110 out.println(vmax);
111 out.close();
112 } // main
113 } // class

```

32.1.3 Cod sursă

Listing 32.1.2: bombo.pas

```

1 Program _bombo_;
2
3 const NMAX = 4;
4     MMAX = 10;
5     TMAX = NMAX * MMAX;
6     MIC = -1;
7
8 type longarray = array [0..MMAX, 0..MMAX, 0..MMAX, 0..MMAX] of longint;
9     plongarray = ^longarray;
10
11 var bmax: array[0..1] of plongarray;
12     z, b: array[1..NMAX, 1..MMAX] of longint;
13     count, x, y: array[1..NMAX] of integer;
14     i, j, k, N, M, il, i2, i3, i4, t, c, a: integer;
15     vmax, tt, tmi, tma, bo: longint;
16
17 begin
18     assign(input, 'bombo.in');
19     reset(input);
20     read(N, M);
21
22     for i := 1 to N do
23         for j := 1 to M do
24             read(z[i, j], b[i, j]);
25
26     close(input);
27
28     for i := 1 to N do
29         count[i] := M;
30
31     for i := N + 1 to NMAX do
32         count[i] := 0;
33
34     new(bmax[0]);
35     new(bmax[1]);
36
37     c := 0;
38     for il := count[1] downto 0 do

```

```

39   for i2 := count[2] downto 0 do
40     for i3 := count[3] downto 0 do
41       for i4 := count[4] downto 0 do
42         bmax[c]^*[i1, i2, i3, i4] := MIC;
43
44 bmax[c]^*[count[1], count[2], count[3], count[4]] := 0;
45 vmax := 0;
46
47 tmi := TMAX + 1;
48 tma := 0;
49
50 for t := 1 to TMAX do
51 begin
52   a := c;
53   c := 1 - c;
54
55   for i1 := count[1] downto 0 do
56     for i2 := count[2] downto 0 do
57       for i3 := count[3] downto 0 do
58         for i4 := count[4] downto 0 do
59           begin
60             bmax[c]^*[i1, i2, i3, i4] := MIC;
61
62             x[1] := i1; x[2] := i2; x[3] := i3; x[4] := i4;
63
64             for i := 1 to NMAX do
65               y[i] := x[i];
66
67             for i := 1 to N do
68               if (x[i] < count[i]) then
69                 begin
70                   y[i] := y[i] + 1;
71
72                   bo := MIC;
73
74                   if (t - 1 < z[i, y[i]]) then
75                     begin
76                       if (bmax[a]^*[y[1], y[2], y[3], y[4]] > bo) then
77                         bo := bmax[a]^*[y[1], y[2], y[3], y[4]] +
78                           (z[i, y[i]] - (t - 1)) * b[i, y[i]];
79                     end;
80
81                   if (t >= z[i, y[i]]) then
82                     begin
83                       if (bmax[c]^*[y[1], y[2], y[3], y[4]] > bo) then
84                         bo := bmax[c]^*[y[1], y[2], y[3], y[4]];
85                     end;
86
87                   if (bmax[c]^*[x[1], x[2], x[3], x[4]] < bo) then
88                     bmax[c]^*[x[1], x[2], x[3], x[4]] := bo;
89
90                   y[i] := y[i] - 1;
91                 end;
92               end;
93
94             if (bmax[c]^*[0, 0, 0, 0] > vmax) then
95               begin
96                 vmax := bmax[c]^*[0, 0, 0, 0];
97                 tt := t;
98               end;
99
100            if (bmax[c]^*[0, 0, 0, 0] > MIC) then
101              begin
102                if (t < tmi) then
103                  tmi := t;
104
105                if (t > tma) then
106                  tma := t;
107              end;
108
109            end;
110
111 writeln(vmax, ' -> ', tmi, ' - ', tt, ' - ', tma);
112
113 assign(output, 'bombo.out');
114 rewrite(output);

```

```

115 writeln(vmax);
116 close(output);
117 end.
```

Listing 32.1.3: bombo2.pas

```

1 Program _bombo_;
2
3 const NMAX = 4;
4     MMAX = 10;
5     TMAX = NMAX * MMAX;
6     MIC = -1;
7
8 type timearray = array [0..TMAX] of longint;
9     ptimearray = ^timearray;
10
11 var bmax: array[0..MMAX, 0..MMAX, 0..MMAX, 0..MMAX] of ptimearray;
12     z, b: array[1..NMAX, 1..MMAX] of longint;
13     count, x, y: array[1..NMAX] of integer;
14     i, j, k, N, M, il, i2, i3, i4, tnext: integer;
15     vmax, bo: longint;
16
17 begin
18 assign(input, 'bombo.in');
19 reset(input);
20 read(N, M);
21
22 for i := 1 to N do
23   for j := 1 to M do
24     read(z[i, j], b[i, j]);
25
26 close(input);
27
28 for i := 1 to N do
29   count[i] := M;
30
31 for i := N + 1 to NMAX do
32   count[i] := 0;
33
34 for il := count[1] downto 0 do
35   for i2 := count[2] downto 0 do
36     for i3 := count[3] downto 0 do
37       for i4 := count[4] downto 0 do
38         begin
39           new(bmax[il, i2, i3, i4]);
40
41           for i := 0 to TMAX do
42             bmax[il, i2, i3, i4]^i := MIC;
43
44           if (il = count[1]) and (i2 = count[2]) and
45             (i3 = count[3]) and (i4 = count[4]) then
46             bmax[il, i2, i3, i4]^0 := 0
47           else
48             begin
49               x[1] := il; x[2] := i2; x[3] := i3; x[4] := i4;
50
51               for i := 1 to NMAX do
52                 y[i] := x[i];
53
54               for i := 1 to N do
55                 if (x[i] < count[i]) then
56                   begin
57                     y[i] := y[i] + 1;
58
59                     for j := 0 to TMAX do
60                       begin
61                         if (bmax[y[1], y[2], y[3], y[4]]^j <= MIC) then
62                           continue;
63
64                         if (j < z[i, y[i]]) then
65                           begin
66                             tnext := j + 1;
67                             bo := bmax[y[1], y[2], y[3], y[4]]^j +
68                               (z[i, y[i]] - j) * b[i, y[i]];
69                           end

```

```

70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86 assign(output, 'bombo.out');
87 rewrite(output);
88
89 vmax := MIC;
90
91 for i := 0 to TMAX do
92   if (bmax[0, 0, 0, 0]^i > vmax) then
93     vmax := bmax[0, 0, 0, 0]^i;
94
95 writeln(vmax);
96 close(output);
97 end.

```

32.2 Cub

Stația de cercetări astronomice ONI-2006 are forma unui cub cu latura de lungime N unități. Fiecare dintre cele 6 fețe ale stației este împărțită în $N \times N$ pătrate cu latura unitate. Cosmonauții însărcinați cu întreținerea stației se pot deplasa pe suprafața cubului, folosind șine speciale plasate în unele dintre pătrățelele unitate, pentru a efectua diverse reparații. Unele pătrate unitate, dintre cele prevăzute cu șine, conțin trape de intrare în interiorul stației.

Din cauza radiațiilor cosmice nocive, cosmonauții trebuie să petreacă un timp cât mai scurt în exteriorul stației astfel încât, după terminarea unei reparații, aceștia trebuie să se reîntoarcă în interiorul stației utilizând cea mai apropiată trapă de acces. Cosmonautul se află inițial într-un pătrățel prevăzut cu șine. El se poate deplasa dintr-un pătrățel ce conține șine, în altul, aflat în vecinătatea poziției curente (pe orizontală sau verticală), pătrățel prevăzut de asemenea cu șine. Cosmonautul se poate deplasa și de pe o față a cubului pe o față vecină, traversând latura comună.

Fetele cubului sunt descrise conform figurii de mai jos.

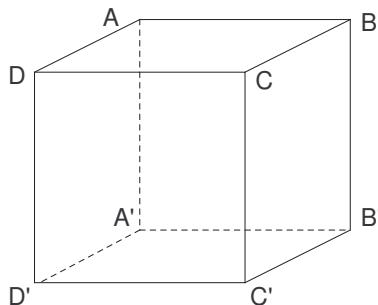


Figura 32.2: Cub1

- Față 1: $ABCD$ - elementul $(1, 1)$ în A și elementul (N, N) în C
- Față 2: $DCC'D'$ - elementul $(1, 1)$ în D și elementul (N, N) în C'
- Față 3: $B'A'D'C'$ - elementul $(1, 1)$ în B' și elementul (N, N) în D'
- Față 4: $BAA'B'$ - elementul $(1, 1)$ în B și elementul (N, N) în A'
- Față 5: $CBB'C'$ - elementul $(1, 1)$ în C și elementul (N, N) în B'
- Față 6: $ADD'A'$ - elementul $(1, 1)$ în A și elementul (N, N) în D'

Cerință

Scrieți un program care să determine distanța minimă de la poziția inițială a cosmonautului până la o trapă de acces, precum și numărul trapelor aflate la distanță minimă.

Datele de intrare

Datele de intrare se citesc din fișierul **cub.in**, care are următoarea structură:

- Pe prima linie se află două numere naturale N și K , separate printr-un spațiu, reprezentând lungimea laturii cubului, respectiv numărul trapelor de acces.
- Pe a doua linie avem numerele naturale F , L , C , separate prin spațiu, desemnând pătratul unitate pe care se află inițial cosmonautul, unde $F \in \{1, 2, 3, 4, 5, 6\}$ reprezintă numărul unei fețe a cubului, iar L și C reprezintă coordonatele poziției inițiale a cosmonautului, relative la colțul $(1, 1)$ al feței cu numărul F (L - numărul liniei, C - numărul coloanei).
- Pe următoarele K linii se află câte 3 numere naturale, separate prin spațiu, reprezentând coordonatele celor K trape de acces (numărul feței, numărul liniei, respectiv numărul coloanei).
- Pe următoarele $6N$ linii se află cele 6 matrice pătratice care descriu fețele cubului, având elemente din mulțimea $\{0, 1\}$ (0 - sănă de acces, 1 - pătrat inaccesibil). Elementele unei fețe sunt date pe linii, de la $(1, 1)$ până la (N, N) .

Datele de ieșire

Rezultatele se scriu în fișierul **cub.out**, care are următoarea structură:

- Pe prima linie se va scrie numărul natural LG , reprezentând lungimea drumului minim parcurs de cosmonaut. Lungimea drumului este dată de numărul pătratelor unitate parcuse, inclusiv poziția inițială și poziția finală.
- Pe a doua linie se va afișa numărul natural T , reprezentând numărul trapelor de acces aflate la distanță minimă față de poziția inițială a cosmonautului.

Restricții și precizări

- $1 \leq N \leq 50$
- $1 \leq F \leq 6$
- $1 \leq L, C \leq N$
- Cosmonautul se află inițial într-o poziție accesibilă (prevăzută cu sănă).
- Există cel puțin o trapă accesibilă din poziția inițială a cosmonautului.

Exemplu

cub.in	cub.out	Explicații
3 2	12	Traseul ilustrat are costul minim 12 și unește elementul (2, 3) de pe față 2 cu elementul (2, 2) de pe față 5.
2 2 3	1	
5 2 2		
3 2 2		
1 1 1		
1 0 0		
1 0 1		
0 0 1		
0 1 0		
0 0 0		
1 1 1		
1 0 1		
1 1 1		
1 1 1		
1 1 1		
1 0 1		
1 0 1		
1 1 1		
1 1 1		
1 1 1		
1 1 1		

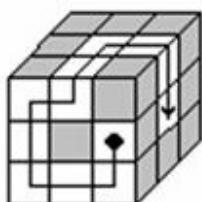


Figura 32.3: Cub2

Timp maxim de rulare/test: 0.5 secunde pentru Windows și 0.2 secunde pentru Linux

32.2.1 Indicații de rezolvare

Alin Burta, C.N. "B.P. Hasdey" Buzău,

Memorez cele 6 fețe ale cubului în tabloul tridimensional *Cube*.

$$Cub[f][i][j] = \begin{cases} 1, & \text{dacă pe fața } f, \text{ poziția } (i,j) \text{ este inaccesibilă} \\ 0, & \text{altfel} \end{cases}$$

Tabloul tridimensional Lg memorează distanța minimă de la poziția de start (poziția inițială (f_i, l_i, c_i)) la oricare altă pozitie (f, i, j) de pe suprafața cubului;

$$Lg[f][i][j] = \begin{cases} \text{lungimea drumului minim ,} & \text{dacă poziția este accesibilă} \\ \infty, & \text{dacă poziția nu este accesibilă} \end{cases}$$

Pentru determinarea lungimii drumurilor minime de la poziția inițială până la celelalte poziții vom utiliza o strategie de tip Lee:

- memorăm poziția initială (fi, li, ci) într-o coadă și $Lg[fi][li][ci] = 1$
 - cât timp mai există elemente necercetate în coadă:
 - extragem primul element al cozii (x, y, z)
 - introducem în coadă toate pozițiile accesibile din acesta, marcând elementele corespunzătoare din Lq cu $Lq[x][y][z] + 1$

Partea mai dificilă a rezolvării este aceea în care determinăm care sunt pozițiile adiacente poziției curente. Dacă ne aflăm pe marginea unei fețe a cubului, una sau două dintre pozițiile vecine se află pe fețe diferite. Functia "urm" determină poziția vecină poziției curente spre Nord, Est, Sud sau Vest, în funcție de fața pe care ne aflăm.

32.2.2 Rezolvare detaliată

Variantă construită după cea oficială (pentru a arăta că nu prea contează limbajul ci algoritm!).

Listing 32.2.1: cub.java

```

1 import java.io.*;
2 class cub
3 {
4     static StreamTokenizer st;
5     static PrintWriter out;
6
7     static final int Nmax=51;
8     static final int infinit=20000;
9
10    static int[][][] Cub=new int[7][Nmax][Nmax]; // fetele cubului
11    static int[][][] Lg=new int[7][Nmax][Nmax]; // lungimea minima a drumului
12
13    static int N;           // latura cubului
14    static int K;           // numarul trapelor de acces
15    static int Fi, Li, Ci; // coordonatele pozitiei initiale
16
17    static int x,y,z;
18
19    static int[] F=new int[7*Nmax*Nmax]; // coordonatele trapelor
20    static int[] L=new int[7*Nmax*Nmax];
21    static int[] C=new int[7*Nmax*Nmax];
22
23    public static void main(String[] args) throws IOException
24    {
25        read_data();
26        init();
27        rez();
28        write_data();
29    } // main...
30
31    static void read_data() throws IOException
32    {
33        int i,j,k;
34
35        st=new StreamTokenizer(new BufferedReader(new FileReader("cub.in")));
36
37        st.nextToken(); N=(int)st.nval;
38        st.nextToken(); K=(int)st.nval;
39
40        st.nextToken(); Fi=(int)st.nval;
41        st.nextToken(); Li=(int)st.nval;
42        st.nextToken(); Ci=(int)st.nval;
43
44        for(i=1;i<=K;i++)
45        {
46            st.nextToken(); F[i]=(int)st.nval;
47            st.nextToken(); L[i]=(int)st.nval;
48            st.nextToken(); C[i]=(int)st.nval;
49        }
50
51        for(k=1;k<=6;k++)
52            for(i=1;i<=N;i++)
53                for(j=1;j<=N;j++)
54                {
55                    st.nextToken(); Cub[k][i][j]=(int)st.nval;
56                }
57    } // read_data...
58
59    static void write_data() throws IOException
60    {
61        out=new PrintWriter(new BufferedWriter(new FileWriter("cub.out")));

```

```

62     int i, nrt, min=infinit,j;
63     nrt=0;
64     for(i=1;i<=K;i++)
65     {
66         if(Lg[ F[i] ][ L[i] ][ C[i] ]!=infinit)
67             if(min>Lg[ F[i] ][ L[i] ][ C[i] ])
68             {
69                 min=Lg[ F[i] ][ L[i] ][ C[i] ];
70                 nrt=1;
71             }
72         else
73             if(min==Lg[ F[i] ][ L[i] ][ C[i] ]) nrt++;
74     }// for i
75
76     if(min==infinit) System.out.println("Eroare !");
77     out.println(min);
78     out.println(nrt);
79     out.close();
80 } // write_data(...)

81
82 static void init()
83 {
84     int i,j,k;
85     for(k=1;k<=6;k++)
86         for(i=1;i<=N;i++)
87             for(j=1;j<=N;j++) Lg[k][i][j]=infinit;
88 } //init(...)

89
90 static void urm(int f,int l,int c,char dir)
91 {
92     if(f==1)
93     {
94         if(dir=='N') if(l==1) {f=4;l=1;c=N-c+1;} else l=l-1; else
95         if(dir=='E') if(c==N) {f=5;c=N-l+1;l=1;} else c=c+1; else
96         if(dir=='S') if(l==N) {f=2;l=1;} else l=l+1; else
97         if(dir=='V') if(c==1) {f=6;c=l;l=1;} else c=c-1;
98     }
99     else
100    if(f==2)
101    {
102        if(dir=='N') if(l==1) {f=1;l=N;} else l=l-1; else
103        if(dir=='E') if(c==N) {f=5;c=1;} else c=c+1; else
104        if(dir=='S') if(l==N) {f=3;c=N-c+1;} else l=l+1; else
105        if(dir=='V') if(c==1) {f=6;c=N;} else c=c-1;
106    }
107    else
108    if(f==3)
109    {
110        if(dir=='N') if(l==1) {f=4;l=N;} else l=l-1; else
111        if(dir=='V') if(c==1) {f=5;c=N-l+1;l=N;} else c=c-1; else
112        if(dir=='S') if(l==N) {f=2;c=N-c+1;l=N;} else l=l+1; else
113        if(dir=='E') if(c==N) {f=6;c=l;l=N;} else c=c+1;
114    }
115    else
116    if(f==4)
117    {
118        if(dir=='N') if(l==1) {f=1;c=N-c+1;l=1;} else l=l-1; else
119        if(dir=='V') if(c==1) {f=5;c=N;} else c=c-1; else
120        if(dir=='S') if(l==N) {f=3;l=1;} else l=l+1; else
121        if(dir=='E') if(c==N) {f=6;c=1;l=N;} else c=c+1;
122    }
123    else
124    if(f==5)
125    {
126        if(dir=='N') if(l==1) {f=1;l=N-c+1;c=N;} else l=l-1; else
127        if(dir=='E') if(c==N) {f=4;c=1;} else c=c+1; else
128        if(dir=='S') if(l==N) {f=3;l=N-c+1;c=1;} else l=l+1; else
129        if(dir=='V') if(c==1) {f=2;c=N;} else c=c-1;
130    }
131    else
132    if(f==6)
133    {
134        if(dir=='N') if(l==1) {f=1;l=c;c=1;} else l=l-1; else
135        if(dir=='E') if(c==N) {f=2;c=1;} else c=c+1; else
136        if(dir=='S') if(l==N) {f=3;l=c;c=N;} else l=l+1; else
137        if(dir=='V') if(c==1) {f=4;c=N;} else c=c-1;

```

```

138      }
139
140      x=f; y=l; z=c;
141 //  urm(...)
142
143 static void rez()
144 {
145     int [] cf=new int [7*Nmax*Nmax];
146     int [] cl=new int [7*Nmax*Nmax];
147     int [] cc=new int [7*Nmax*Nmax];
148     int p,u;
149
150     p=1; u=1;
151     cf[u]=Fi; cl[u]=Li; cc[u]=Ci;
152     Lg[Fi][Li][Ci]=1;
153     while(p<=u)
154     {
155         x=cf[p]; y=cl[p]; z=cc[p]; urm(x,y,z,'N');
156         if((Lg[x][y][z]==infinit) && (Cub[x][y][z]==0))
157         {
158             Lg[x][y][z]=Lg[cf[p]][cl[p]][cc[p]]+1;
159             cf[++u]=x;
160             cl[u]=y; cc[u]=z;
161         }
162
163         x=cf[p]; y=cl[p]; z=cc[p]; urm(x,y,z,'E');
164         if((Lg[x][y][z]==infinit) && (Cub[x][y][z]==0))
165         {
166             Lg[x][y][z]=Lg[cf[p]][cl[p]][cc[p]]+1;
167             cf[++u]=x; cl[u]=y; cc[u]=z;
168         }
169
170         x=cf[p]; y=cl[p]; z=cc[p]; urm(x,y,z,'S');
171         if((Lg[x][y][z]==infinit) && (Cub[x][y][z]==0))
172         {
173             Lg[x][y][z]=Lg[cf[p]][cl[p]][cc[p]]+1;
174             cf[++u]=x; cl[u]=y; cc[u]=z;
175         }
176
177         x=cf[p]; y=cl[p]; z=cc[p]; urm(x,y,z,'V');
178         if((Lg[x][y][z]==infinit) && (Cub[x][y][z]==0))
179         {
180             Lg[x][y][z]=Lg[cf[p]][cl[p]][cc[p]]+1;
181             cf[++u]=x; cl[u]=y; cc[u]=z;
182         }
183
184         p++;
185     } // while
186 } // rez...
187 } // class

```

32.2.3 Cod sursă

Listing 32.2.2: cub.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 #define Fin "cub.in"
6 #define Fout "cub.ok"
7 #define Nmax 51
8 #define infinit 20000
9
10 typedef int matrix[7][Nmax][Nmax];
11 typedef int vector[7*Nmax*Nmax];
12
13 matrix Cub; //retine cele 6 fetze ale cubului
14 matrix Lg; //Lg[f][i][j] = lungimea minima a drumului de la pozitia
15 // initiala la pozitia (f,i,j)
16 int N; //latura cubului
17 int K; //numarul trapelor de acces

```

```

18 int Fi, Li, Ci; //coordonatele pozitiei initiale
19 vector F,L,C; //retin coordonatele trapelor
20
21 //prototipurile functiilor utilize
22 void read_data();
23 void init();
24 void urm(int &, int &, int &, char);
25 void rez();
26 void afis(int);
27 void write_data();
28
29 int main()
30 {
31     read_data();
32     init();
33     rez();
34     write_data();
35     return 0;
36 }
37
38 //definirea functiilor
39 void read_data()
40 {
41     int i,j,k;
42     ifstream f(Fin);
43     f>>N>>K;
44     f>>Fi>>Li>>Ci;
45     for(i=1;i<=K;i++) f>>F[i]>>L[i]>>C[i];
46     for(k=1;k<=6;k++)
47         for(i=1;i<=N;i++)
48             for(j=1;j<=N;j++) f>>Cub[k][i][j];
49     f.close();
50 }
51
52 void write_data()
53 {
54     int i, nrt, min=infinity, j;
55     ofstream g(Fout);
56     nrt=0;
57     for(i=1;i<=K;i++)
58         if(Lg[ F[i] ][ L[i] ][ C[i] ]!=infinity)
59             if(min>Lg[ F[i] ][ L[i] ][ C[i] ]) min=Lg[ F[i] ][ L[i] ][ C[i] ],nrt=1;
60             else if(min==Lg[ F[i] ][ L[i] ][ C[i] ]) nrt++;
61     if(min==infinity) g<<"Eroare !";
62     {
63         g<<min<<'\'n';
64         g<<nrt<<'\'n';
65     }
66     g.close();
67 }
68
69 void init()
70 {
71     int i,j,k;
72     for(k=1;k<=6;k++)
73         for(i=1;i<=N;i++)
74             for(j=1;j<=N;j++) Lg[k][i][j]=infinity;
75 }
76
77 void urm(int &f, int &l, int &c, char dir)
78 {
79     if(f==1){
80         if(dir=='N') if(l==1) f=4,l=1,c=N-c+1; else l=l-1; else
81             if(dir=='E') if(c==N) f=5,c=N-l+1,l=1; else c=c+1; else
82                 if(dir=='S') if(l==N) f=2,l=1; else l=l+1; else
83                     if(dir=='V') if(c==1) f=6,c=l,l=1; else c=c-1;
84     } else
85         if(f==2){
86             if(dir=='N') if(l==1) f=1,l=N; else l=l-1; else
87                 if(dir=='E') if(c==N) f=5,c=1; else c=c+1; else
88                     if(dir=='S') if(l==N) f=3,c=N-c+1; else l=l+1; else
89                         if(dir=='V') if(c==1) f=6,c=N; else c=c-1;
90     } else
91         if(f==3){
92             if(dir=='N') if(l==1) f=4,l=N; else l=l-1; else
93                 if(dir=='V') if(c==1) f=5,c=N-l+1,l=N; else c=c-1; else

```

```

94      if(dir=='S') if(l==N) f=2,c=N-c+1,l=N; else l=l+1; else
95      if(dir=='E') if(c==N) f=6,c=1,l=N; else c=c+1;
96  } else
97  if(f==4){
98      if(dir=='N') if(l==1) f=1,c=N-c+1,l=1; else l=l-1; else
99      if(dir=='V') if(c==1) f=5,c=N; else c=c-1; else
100     if(dir=='S') if(l==N) f=3,l=1; else l=l+1; else
101     if(dir=='E') if(c==N) f=6,c=1; else c=c+1;
102 } else
103 if(f==5){
104     if(dir=='N') if(l==1) f=1,l=N-c+1,c=N; else l=l-1; else
105     if(dir=='E') if(c==N) f=4,c=1; else c=c+1; else
106     if(dir=='S') if(l==N) f=3,l=N-c+1,c=1; else l=l+1; else
107     if(dir=='V') if(c==1) f=2,c=N; else c=c-1;
108 } else
109 if(f==6){
110     if(dir=='N') if(l==1) f=1,l=c,c=1; else l=l-1; else
111     if(dir=='E') if(c==N) f=2,c=1; else c=c+1; else
112     if(dir=='S') if(l==N) f=3,l=c,c=N; else l=l+1; else
113     if(dir=='V') if(c==1) f=4,c=N; else c=c-1;
114 }
115 }
116 void rez()
117 {
118     vector cf,cl,cc;
119     int p,u;
120     int x,y,z;
121     p=1;u=1;
122     cf[u]=Fi; cl[u]=Li; cc[u]=Ci; Lg[Fi][Li][Ci] = 1;
123     while(p<=u)
124     {
125         x=cf[p];y=cl[p];z=cc[p];urm(x,y,z,'N');
126         if(Lg[x][y][z]==infinit && Cub[x][y][z]==0)
127             Lg[x][y][z]=Lg[cf[p]][cl[p]][cc[p]]+1,cf[++u]=x,cl[u]=y,cc[u]=z;
128
129         x=cf[p];y=cl[p];z=cc[p];urm(x,y,z,'E');
130         if(Lg[x][y][z]==infinit && Cub[x][y][z]==0)
131             Lg[x][y][z]=Lg[cf[p]][cl[p]][cc[p]]+1,cf[++u]=x,cl[u]=y,cc[u]=z;
132
133         x=cf[p];y=cl[p];z=cc[p];urm(x,y,z,'S');
134         if(Lg[x][y][z]==infinit && Cub[x][y][z]==0)
135             Lg[x][y][z]=Lg[cf[p]][cl[p]][cc[p]]+1,cf[++u]=x,cl[u]=y,cc[u]=z;
136
137         x=cf[p];y=cl[p];z=cc[p];urm(x,y,z,'V');
138         if(Lg[x][y][z]==infinit && Cub[x][y][z]==0)
139             Lg[x][y][z]=Lg[cf[p]][cl[p]][cc[p]]+1,cf[++u]=x,cl[u]=y,cc[u]=z;
140         p++;
141     }
142 }
```

32.3 Logic

Mircea cel Tânăr trebuie să îmbunătățească permanent performanțele calculatoarelor pe care le are în gestiune. Se întâmplă căteodată, ca unele componente noi pe care le folosește să nu fie compatibile cu vechile calculatoare. Din acest motiv funcționarea calculatoarelor "îmbunătățite" poate să nu fie corectă. Pentru a verifica corectitudinea funcționării acestor calculatoare, Mircea își propune să le testeze cu ajutorul unui program care verifică echivalența unor expresii logice.

Cerință

Scrieți un program care determină dacă două expresii logice sunt echivalente sau nu.

Fiecare expresie este formată din:

- variabile, cele 26 de litere mici ale alfabetului englez, de la 'a' - 'z';
- operatori binari |, &, ^ (SAU, SI respectiv SAU EXCLUSIV);
- operatorul unar ~ (NEGATIE);
- paranteze rotunde.

Expresiile vor fi evaluate respectând regulile de priorități ale operatorilor și parantezelor pentru evaluarea expresiilor logice în care intervin ca operanzi biții 0 și 1. Prioritățile în ordine descrescătoare sunt: parantezele rotunde '(', ')', operatorul unar '~', operatorii binari în ordine descrescătoare '&', '^', '|'.

Două expresii sunt echivalente dacă:

- conțin același set de variabile indiferent de numărul de apariții a variabilei în expresie;
- pentru orice set de date de intrare pentru variabile (valori 0, 1) rezultatul obținut este același.

Datele de intrare

Fișierul de intrare **logic.in** conține pe primul rând un număr natural n , ce reprezintă numărul testelor ce se vor evalua. Fiecare test reprezintă evaluarea a două expresii. Pe următoarele $2 * n$ linii sunt siruri de caractere ce constituie expresiile. Acestea sunt scrise pe câte o linie fiecare.

Datele de ieșire

Fișierul de ieșire **logic.out** va conține n linii, pe fiecare linie k fiind mesajul "egale" sau "diferite" în funcție de rezultatul evaluării expresiilor de pe liniile $2 * k$ și respectiv $2 * k + 1$ din fișierul de intrare.

Restricții și precizări

- $0 < n \leq 50$
- n reprezintă numărul de perechi de expresii ce trebuie evaluate.
- O expresie conține cel mult 100 de operații și maxim 10 variabile.
- O expresie poate avea cel mult 255 caractere. Spațiul nu este separator în interiorul unei expresii.
- Numele unei variabile este un singur caracter, literă mică a alfabetului englez.
- Expresiile date sunt corecte.

Exemplu

logic.in	logic.out
4	diferite
a&(c c)	egale
a	diferite
~(a b c d)	egale
~a&~b&~c&~d	
z&b	
a&b	
a b	
(a ~a)&(a ~a)&(a ~a)&(a b)	

Explicație: Pentru ultimul set de expresii tabelul este:

a	b	$\sim a$	$a \sim a$	$(a \sim a) \& (a \sim a) \& (a \sim a)$	a b	E
0	0	1	1	1	0	0
0	1	1	1	1	1	1
1	0	0	1	1	1	1
1	1	0	1	1	1	1

unde $E = (a| a) \& (a| a) \& (a| a) \& (a| b)$

Timp maxim de rulare/test: 0.4 secunde pentru Windows și 0.2 secunde pentru Linux

32.3.1 Indicații de rezolvare

Maria Niță și Adrian Niță, C.N. "Emanuil Gojdu" Oradea

Se verifică dacă expresiile au aceleași variabile, altfel ele se consideră a fi "diferite".

Se realizează în continuare evaluarea fiecărei expresii prin înlocuirea variabilelor prin valorile 0 și 1, respectându-se prioritățile operatorilor.

Pentru o expresie în care intervin k variabile se realizează 2^k combinații de valori 0 și 1.

Dacă pentru orice set de valori date variabilelor, cele două expresii au valori egale se consideră că acestea sunt "egale", altfel dacă există o combinație de valori 0, 1 pentru variabile astfel încât cele două expresii prin evaluare să dea valori diferite se consideră că cele două expresii sunt "diferite".

32.3.2 *Rezolvare detaliată

32.3.3 Cod sursă

Listing 32.3.1: logic.cpp

```

1 # include <stdio.h>
2 # include <string.h>
3
4 # define _fin "logic.in"
5 # define _fout "logic.out"
6
7 # define maxs 512 // lungimea maxima
8 # define maxv 20 // numarul de variabile
9 # define sigma 26 // literele mici ale alfabetului englez
10
11 char e1[maxs], e2[maxs];
12 int var[maxv], nv, vv[sigma], l1, l2;
13 int ec=1, poz;
14 /*
15     var[i] = xi; => xi+'a' este simbolul variabilei i
16     nv = numarul de variabile,
17     vv = valoarea fiecarei variabile existente
18 */
19
20 void readf(FILE *fin)
21 {
22     int i;
23     int aux[maxs];
24
25 # define rem_endl(a) ( a[strlen(a)-1] = '\0' )
26
27     fgets(e1, maxs, fin), rem_endl(e1);
28     fgets(e2, maxs, fin), rem_endl(e2);
29
30 # undef rem_endl
31
32     for(i=0; i<strlen(e1); i++)
33         if ( e1[i] == ' ' )
34             memmove( e1+i, e1+i+1, sizeof(int) * ( strlen(e1)-i ) ),
35             --i;
36
37     for(i=0; i<strlen(e2); i++)
38         if ( e2[i] == ' ' )
39             memmove( e2+i, e2+i+1, sizeof(int) * ( strlen(e2)-i ) ),
40             --i;
41 }
42
43
44 void writef(FILE *fout)
45 {
46     fprintf(fout, "%s\n", ec==1 ? "egale" : "diferite");
47 }
48
49 # define isalpha(x) ( (x) >= 'a' && (x) <= 'z' )
50
51 void pre_compute()
52 {
53     int i;
54     memset(vv, 0, sizeof(vv));
55     memset(var, 0, sizeof(var));
56     nv = poz = 0;
57     ec = 1;
58
59     l1 = strlen(e1), l2 = strlen(e2);

```

```

60
61     for(i=0; i<l1; i++)
62         if ( isalpha( e1[i] ) && ! vv[ e1[i] - 'a' ] )
63             ++vv[ var[ ++nv ] = e1[i] - 'a' ];
64
65     for(i=0; i<l2; i++)
66         if ( isalpha( e2[i] ) && vv[ e2[i] - 'a' ] & 1 )
67             ++vv[ e2[i] - 'a' ];
68
69     for(i=0; i<sigma; i++) if ( vv[i] & 1 ) ec = 0;
70 // daca o variabila nu se gaseste in ambele expresii
71 // cele doua nu sunt echivalente
72
73     memset(vv, 0, sizeof(vv));
74 }
75
76 inline int getlvalue(char ch) // valoarea logica a variabilei ch | nu
77 // se verifica existenta acestaia
78 {
79     return ( vv[ ch - 'a' ] );
80 }
81
82 inline int variable(const char e[])
83 {
84     if ( e[poz] == '~' )
85     {
86         ++poz;
87         return ( !variable(e) );
88     }
89     else
90         return ( getlvalue( e[ poz++ ] ) );
91 }
92
93 int evaluate(const char e[], int l);
94
95 int log_and(const char e[], int l)
96 { // evaluez o secenta care contine or sau xor
97     int v, neg=0;
98
99     while ( e[poz] == '~' && poz < l )
100    {
101        ++poz, neg = !neg;
102    }
103
104    if ( e[poz] == '(' )
105    {
106        ++poz; // (
107        v = evaluate( e, l );
108        if ( e[poz] != ')' )
109            ec = -1;
110        // nu ar trebui sa se intampla daca expresia este corecta
111        ++poz; // )
112    }
113    else
114        v = variable( e ); // valoarea primei variabile
115
116    if ( neg ) v = !v; // negam NUMAI prima valoare
117
118    while ( e[ poz ] == '&' && poz < l ) // AND are cea mai mare prioritate
119        ++poz, v &= log_and( e, l );
120
121    return v;
122 }
123
124 int evaluate(const char e[], int l)
125 {
126     // XOR, OR
127     int v = log_and( e, l );
128
129     while ( e[ poz ] == '|' || e[ poz ] == '^' && poz < l )
130         if ( e[ poz ] == '|' )
131             ++poz, v |= evaluate( e, l );
132         else
133             ++poz, v ^= evaluate( e, l );
134
135     return v;

```

```

136 }
137
138 void solve()
139 {
140     int max, i, j, vi; // valoarea initiala
141
142     max = ( 1 << nv ) - 1; // 111...1 de nv ori
143
144 # define getbit(a, bit) ( ( (a) >> (bit) ) & 1 )
145
146 // evaluam cele doua expresii cand toate variabilele sunt setate false
147 vi = evaluate(e1, 11), poz = 0;
148 if ( evaluate(e2, 12) != vi ) ec = 0;
149
150 for(i=1; i<=max && ec==1; i++)
151 {
152     for(j=0; j<nv; j++)
153         vv[ var[j+1] ] = getbit(i, j);
154
155     poz = 0, vi = evaluate(e1, 11);
156     poz = 0;
157     if ( vi != evaluate(e2, 12) )
158     {
159         ec = 0, poz = 0;
160         evaluate(e2, 12);
161     }
162 }
163 }
164
165 int main()
166 {
167     int t;
168     FILE *fin = fopen(_fin, "r");
169     FILE *fout= fopen(_fout, "w");
170
171     for(fscanf(fin, "%d\n", &t); t; --t)
172     {
173         readf(fin);
174         pre_compute();
175         solve();
176         writef(fout);
177     }
178
179     fclose(fin);
180     fclose(fout);
181
182     return 0;
183 }
```

32.4 Medie

La Târgoviște, în Cetatea Domnească, a fost descoperit un document în care erau scrise mai multe numere naturale. Mircea cel Tânăr, pasionat de aritmetică, a observat proprietatea că, uneori, un număr din sir poate fi scris ca medie aritmetică a două numere de pe alte două poziții din sir. Întrebarea pe care și-o pune Mircea cel Tânăr este de câte ori se regăsește în sir această proprietate.

Cerință

Scriți un program care determină numărul total de triplete (i, j, k) cu $i \neq j$, $i \neq k$, $j < k$ astfel încât v_i este media aritmetică dintre v_j și v_k .

Datele de intrare

Fișierul de intrare **medie.in** are pe prima linie o valoare n reprezentând numărul de numere din sir, iar pe următoarele n linii câte o valoare v_i pe linie, reprezentând valorile din sir. Valorile din sir nu sunt neapărat distințe.

Datele de ieșire

Fișierul de ieșire **medie.out** va conține o singură linie cu o valoare max , reprezentând numărul total de triplete determinat.

Restricții și precizări

- $0 < n \leq 9000$
- $0 < v_i \leq 7000$

Exemple

medie.in	medie.out	Explicație
5 1 1 1 1 1	30	Fiecare valoare 1 poate fi scrisă ca media a câte două valori din celelalte 4 posibile. Se vor număra tripletele: (1, 2, 3), (1, 2, 4), (1, 2, 5), (1, 3, 4), (1, 3, 5), (1, 4, 5), (2, 1, 3), (2, 1, 4), (2, 1, 5), (2, 3, 4), (2, 3, 5), (2, 4, 5), (3, 1, 2), (3, 1, 4), (3, 1, 5), etc.

medie.in	medie.out	Explicație
3 4 2 1	0	Valoarea 4 nu este media aritmetică a valorilor 2 și 1, Valoarea 2 nu este media aritmetică a valorilor 4 și 1, Valoarea 1 nu este media aritmetică a valorilor 4 și 2.

medie.in	medie.out	Explicație
6 3 1 6 4 5 2	6	$2=(1+3)/2$ $3=(4+2)/2; (1+5)/2$ $4=(3+5)/2; (6+2)/2$ $5=(6+4)/2$ Tripletele sunt: (6, 1, 2), (1, 4, 6), (1, 2, 5), (4, 1, 5), (4, 3, 6), (5, 3, 4).

Timp maxim de rulare/test: 1.5 secunde pentru Windows și 0.6 secunde pentru Linux

32.4.1 Indicații de rezolvare

Maria Niță și Adrian Niță - C.N. "Emanuil Gojdu" Oradea

Rezolvarea problemei se realizează cu complexitate $O(n^2)$.

Pentru fiecare două valori se determină media aritmetică.

Într-un vector v_i s-a reținut numărul de perechi din sir care au media aritmetică egală cu i .

În final ținând cont și de frecvența de apariție a valorii în sir se determină numărul total de valori ce pot fi scrise ca medie aritmetică a două valori din sir.

Dacă n este dimensiunea sirului iar a_i reprezintă valorile din sir, secvența de algoritm este:
pentru $i \leftarrow 1, n-1$ execută

```

frecvență ( $a_i$ )  $\leftarrow$  frecvență ( $a_i$ ) +1;
pentru  $j \leftarrow i + 1, n$  execută
    dacă  $aux * 2 = a_i + a_j$  atunci  $v_{aux} \leftarrow v_{aux} + 1$ 
    sfârșit_dacă
sfârșit_pentru
sfârșit_pentru
frecvență ( $a_n$ )  $\leftarrow$  frecvență ( $a_n$ ) +1;
pentru  $i \leftarrow 1, n$  execută
    dacă  $v(a_i) \neq 0$  atunci total  $\leftarrow$  total +  $v(a_i)$  - frecvență ( $a_i$ ) +1;
    sfârșit_pentru

```

32.4.2 Rezolvare detaliată

Listing 32.4.1: medie.java

```

1 import java.io.*;
2 import java.util.Random;
3 class medie
4 {
5     static StreamTokenizer st;
6     static PrintWriter out;
7

```

```

8  static final int MaxN=9002;
9  static final int MaxX=7002;
10 static int n,sol;
11 static int[] a=new int[MaxN];
12 static int[] rep=new int[MaxX];
13 static int[] v=new int[MaxX];// v[i] = nr perechi cu media aritmetica i
14
15 public static void main(String[] args) throws IOException
16 {
17     readf();
18     solve();
19     writef();
20 } // main
21
22 static void readf() throws IOException
23 {
24     int i;
25     st=new StreamTokenizer(new BufferedReader(new FileReader("9-medie.in")));
26
27     st.nextToken(); n=(int)st.nval;
28
29     for(i=1; i<=n; i++)
30     {
31         st.nextToken(); a[i]=(int)st.nval;
32     }
33 } // readf(...)

34
35 static void writef() throws IOException
36 {
37     out=new PrintWriter(new BufferedWriter(new FileWriter("medie.out")));
38     System.out.println("sol = "+sol);
39     out.println(sol);
40     out.close();
41 } // writef(...)

42
43 static int qs_poz(int st, int dr)
44 {
45     int x = a[st];
46
47     while(st<dr)
48     {
49         while(st<dr && a[dr]>=x) dr--;
50         a[st]=a[dr];
51         while(st<dr && a[st]<=x) st++;
52         a[dr]=a[st];
53     }
54
55     a[st] = x;
56     return st;
57 } // qs_poz(...)

58
59 static void qs_rand(int st, int dr)
60 {
61     int r,m,aux;
62     Random generator = new Random();
63     r=generator.nextInt(dr-st+1)+st;
64
65     if(r!=st) { aux=a[r]; a[r]=a[st]; a[st]=aux; }
66     m=qs_poz(st, dr);
67     if(st<m) qs_rand(st,m-1);
68     if(m<dr) qs_rand(m+1,dr);
69 } // qs_rand

70
71 static void solve()
72 {
73     int i, j, p, step, vs, c, aux;
74
75     qs_rand(1, n);
76
77     for(i=1; i<n; i++)
78     {
79         ++rep[ a[i] ];
80         for(j=i+1; j<=n; j++)
81         {
82             aux=a[i]+a[j];
83             if(aux%2==0) ++v[aux>>1];

```

```

84         }
85     } // for i
86
87     ++rep[a[n]];
88
89     for(i=1; i<=n; i++)
90         if(v[a[i]]>0) sol+=v[a[i]]-rep[a[i]]+1;
91     } // solve(...)
92 } // class

```

32.4.3 Cod sursă

Listing 32.4.2: medi.cpp

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define _fin "medie.in"
6 #define _fout "medie.out"
7
8 # define MaxN 9002
9 # define MaxX 7002
10
11 int a[MaxN], rep[MaxX], n;
12 long sol;
13
14 void readf()
15 {
16     int i;
17     FILE *fin = fopen(_fin, "r");
18
19     for(fscanf(fin, "%d", &n), i=1; i<=n; i++)
20         fscanf(fin, "%d", a+i);
21
22     fclose(fin);
23 }
24
25 void writef()
26 {
27     FILE *fout = fopen(_fout, "w");
28     fprintf(fout, "%ld\n", sol), fclose(fout);
29 }
30
31 int qs_poz(int st, int dr)
32 {
33     int x = a[st];
34
35     while ( st < dr )
36     {
37         while ( st < dr && a[dr] >= x ) dr--; a[st] = a[dr];
38         while ( st < dr && a[st] <= x ) st++; a[dr] = a[st];
39     }
40
41     a[st] = x; return st;
42 }
43
44 void qs_rand(int st, int dr)
45 {
46     int r = rand() % (dr-st+1) + st, m;
47     if ( r != st )
48         a[r] ^= a[st] ^= a[r] ^= a[st];
49
50     m = qs_poz(st, dr);
51     if ( st < m ) qs_rand(st, m-1);
52     if ( m < dr ) qs_rand(m+1, dr);
53 }
54
55 long v[MaxX];
56 //int rep[MaxX];
57 /*
58     v[i] = numarul de perechi din sir care au media aritmetica i

```

```

59  */
60
61 void solve()
62 {
63     int i, j, p, step, vs, c, aux;
64
65     qs_rand(1, n);
66
67 # define div2(x) ( ! ((x) & 1) )
68
69     for(i=1; i<n; i++)
70     {
71         ++rep[ a[i] ];
72
73         for(j=i+1; j<=n; j++)
74             if ( div2( aux=a[i]+a[j] ) )
75                 ++v[aux>>1];
76     }
77
78     ++rep[a[n]];
79
80     for(i=1; i<=n; i++)
81         if ( v[ a[i] ] )
82             sol += long( v[ a[i] ] - rep[ a[i] ] + 1 );
83 }
84
85 int main()
86 {
87     readf();
88     solve();
89     writef();
90
91     return 0;
92 }
```

e

32.5 Prietenii

Mai mulți prieteni iubitori ai muntelui au parte de o adevărată aventură, într-o din zilele vacanței de iarnă. Surprinși de dificultatea traseului ales spre parcurgere și de înrăutățirea rapidă a condițiilor meteo (ninsoare și viscol) își dau seama, odată cu lăsarea întunericului, că ar fi mai bine să se adăpostească pe timpul nopții, urmând să revină la cabană în dimineața zilei următoare. Norocul pare să le surâdă în cele din urmă. Pe partea cealaltă a prăpastiei la marginea căreia se află, se zărește un refugiu. Cercetând împrejurimile, descoperă un podeț din lemn care i-ar putea ajuta să traverseze prăpastia. După o evaluare mai atentă, realizează că traversarea nu va fi deloc ușoară, pentru că există câteva traverse lipsă, iar starea podețului permite traversarea sa de către cel mult 2 persoane, la un moment dat. Vizibilitatea este extrem de redusă și mai există o singură lanternă funcțională ce va trebui folosită judicios pentru traversarea în siguranță a tuturor membrilor grupului. Traversarea se va face alternativ în ambele sensuri, pentru a putea reduce lanterna celor care n-au traversat încă.

Cerință

Cunoscând numărul membrilor grupului, timpul fiecărui de traversare (exprimat în secunde) și știind că, la o traversare în care sunt angajați doi membri, timpul este dat de timpul de traversare al celui mai lent dintre ei, găsiți o strategie adecvată pentru ca toți membrii grupului să traverseze prăpastia în timpul cel mai scurt. Dacă există mai multe soluții cu același timp minim de traversare, se va determina numai una, oricare dintre ele.

Datele de intrare

Datele de intrare se preiau din fișierul **prieteni.in**. Acesta conține pe prima linie valoarea n , adică numărul prietenilor, iar pe următoarele n linii căte un număr pe linie, numărul t_i aflat pe linia $i + 1$ din fișier, reprezentând timpul (exprimat în secunde) în care persoana i din grup poate traversa singură podețul.

Datele de ieșire

Soluția se va scrie în fișierul **prieteni.out**. Pe primele n linii se va descrie strategia urmată pentru atingerea acestui timp minim: fiecare linie i va conține una sau două valori, indicând

persoana sau persoanele ce traversează podețul la pasul i . Fiecare persoană este indicată chiar prin timpul său de traversare, specificat corespunzător în fișierul cu datele de intrare **prieteni.in**.

Pe ultima linie se va scrie valoarea ce reprezintă timpul minim total (exprimat în secunde) necesar pentru ca toți cei n membri ai grupului să traverseze prăpastia.

Restricții și precizări

- $1 \leq n \leq 1000$;
- în orice moment pot traversa podețul cel mult 2 membri ai grupului;
- $1 \leq t_i \leq 1000$;
- pot exista mai multe persoane cu același timp de traversare, dar în specificarea soluției nu are importanță cărei persoane îi aparțin timpul respectiv.

Exemplu

prieteni.in	prieteni.out
3	2 3
2	2
3	2 5
5	10

Explicație: mai întâi traversează persoanele 1 și 2, având timpii 2, respectiv 3 secunde. Timpul de traversare la această trecere este dat de timpul cel mai mare: 3 secunde. Se întoarce apoi persoana 1 cu lanterna, timpul de traversare fiind de 2 secunde. La a treia traversare, trec persoanele 1 și 3, având timpii 2, respectiv 5 secunde. De data aceasta, timpul de traversare la această trecere este dat de timpul cel mai mare: 5 secunde. Timp total: $3 + 2 + 5 = 10$ secunde.

Aceasta este una dintre strategiile posibile. O altă soluție corectă: traversează mai întâi persoana 1 cu persoana 3, se întoarce persoana 1 și traversează apoi persoana 1 cu persoana 2, și în acest caz se obține același timp minim de 10 secunde.

Timp maxim de rulare/test: 0.5 secunde pentru Windows și 0.5 secunde pentru Linux

32.5.1 Indicații de rezolvare

Roxana Tîmplaru - ISJ Dolj

- Dacă avem un grup format din două persoane, timpul de traversare este egal cu timpul celui mai lent
- Dacă avem un grup format din trei persoane, cea mai rapidă persoană va traversa pe rând cu prima persoană și se va întoarce apoi să traverseze cu a doua persoană, timpul total de traversare este egal cu suma timpilor celor trei persoane
- Dacă avem un grup format din patru persoane, presupunem a, b, c, d ca fiind timpii de traversare pentru cele patru persoane, în ordine crescătoare.

În acest caz există două strategii:

- Persoana cea mai rapidă (cu timpul a) va traversa pe rând podețul împreună cu fiecare persoană, revenind apoi cu lanterna pentru a prelua următoarea persoană, va rezulta un timp total de $b + c + d + 2a$.
- Persoanele cu timpii a și b traversează împreună, se întoarce cea mai rapidă persoană (cea cu timpul a), trec apoi împreună cele mai lente (cele cu timpii cei mai mari (c și d)) și se întoarce de pe malul vecin a doua persoană cu timpul cel mai mic (cea cu timpul b), și va trece apoi împreună cu cea mai rapidă persoană, rezultând astfel un timp total de $a + 3b + d$. Dacă $2a + b + c + d > a + 3b + d$, adică $a + c > 2b$ este mai bună a doua strategie, în caz contrar este mai bună prima strategie
- Dacă avem un grup format din cinci persoane, cu timpii ordonați crescător a, b, c, d, e . Dacă decidem să rămână pentru transportat singură persoana cu timpul c, d sau e , timpul cel mai bun s-ar obține evident în cazul în care transportăm împreună d cu e și se obține timpul e și apoi a cu c și se obține timpul c , total $e + c$ (nu au fost luat în calcul timpii pentru traversarea

lui b și timpii pentru reveniri). În cazul în care transportăm împreună c cu d și se obține timpul d și apoi a cu e și se obține timpul e , total $d+e$ (nu au fost luat în calcul timpii pentru traversarea lui b și timpii pentru reveniri), evident mai mare decât anteriorul. Rezultă, dacă se iau în calcul toate cazurile, că cel mai bun timp se obține traversând împreună cele mai lente persoane.

Se poate generaliza deci, studiind la fiecare caz ce strategie trebuie aleasă.

Pentru traversarea persoanei i există două strategii:

- traversează persoana i cu persoana 1 (cea mai rapidă), se întoarce persoana 1
- traversează persoana 1 (cea mai rapidă) cu persoana 2 (cea mai rapidă după persoana 1), se întoarce persoana 1, traversează persoanele i și $i - 1$, se întoarce persoana 2

32.5.2 Rezolvare detaliată

Listing 32.5.1: prieteni.java

```

1 import java.io.*;
2 class prieteni
3 {
4     static StreamTokenizer st;
5     static PrintWriter out;
6     static int n;
7     {
8         st=new StreamTokenizer(new BufferedReader(new FileReader("prieteni.in")));
9         out=new PrintWriter(new BufferedWriter(new FileWriter("prieteni.out")));
10
11     citire();
12     qsort(1,n);
13     calcul();
14
15     out.close();
16 } // main...
17
18 static void citire() throws IOException
19 {
20     int i;
21     st.nextToken(); n=(int)st.nval;
22     for(i=1;i<=n;i++)
23     {
24         st.nextToken(); x[i]=(int)st.nval;
25     }
26 } // citire...
27
28 static void qsort(int l, int r)
29 {
30     int i, j, v, aux;
31
32     i=1; j=r; v=x[(l+r)/2];
33     do
34     {
35         while(x[i]<v) i++;
36         while(v<x[j]) j--;
37         if(i<=j)
38         {
39             aux=x[i]; x[i]=x[j]; x[j]=aux;
40             i++; j--;
41         }
42     } while(i<=j);
43
44     if(l<j) qsort(l, j);
45     if(i<r) qsort(i, r);
46 } // sort...
47
48 static void calcul()
49 {
50     int i,k;
51
52     i=n;
53     k=0;
```

```

54     while(i>=4)
55         if(x[1]+x[i-1]>2*x[2])
56         {
57             out.println(x[1]+" "+x[2]);
58             out.println(x[1]);
59             out.println(x[i-1]+" "+x[i]);
60             out.println(x[2]);
61             k=k+x[1]+2*x[2]+x[i];
62             i=i-2;
63         }
64     else
65     {
66         out.println(x[1]+" "+x[i]);
67         out.println(x[1]);
68         out.println(x[1]+" "+x[i-1]);
69         out.println(x[1]);
70         k=k+2*x[1]+x[i]+x[i-1];
71         i=i-2;
72     }
73
74     if(i==3)
75     {
76         out.println(x[1]+" "+x[3]);
77         out.println(x[1]);
78         out.println(x[1]+" "+x[2]);
79         k=k+x[1]+x[2]+x[3];
80     }
81     else
82     if(i==2)
83     {
84         out.println(x[1]+" "+x[2]);
85         k=k+x[2];
86     }
87     else
88     {
89         out.println(x[1]);
90         k=k+x[1];
91     }
92
93     out.println(k);
94 } //calcul...
95 } // class

```

32.5.3 Cod sursă

Listing 32.5.2: prieteni.cpp

```

1 #pragma option -O2
2 #include <iostream>
3
4 using namespace std;
5
6 ifstream fin("prieteni.in");
7 ofstream fout("prieteni.out");
8
9 int n,pas,timp;
10 int a[1024],b[3][4096];
11
12 int etc(int i,int j)
13 {
14     int i0=0,j0=-1,aux;
15     while(i!=j)
16     {
17         if(a[i]>a[j])
18         {
19             aux=a[i];
20             a[i]=a[j];
21             a[j]=aux;
22             aux=-i0;
23             i0=-j0;
24             j0=aux;
25         }

```

```

26         i+=i0;
27         j+=j0;
28     }
29     return i;
30 }
31
32 void qsort(int li,int ls)
33 {
34     if(li<ls)
35     {
36         int k=etc(li,ls);
37         qsort(li,k-1);
38         qsort(k+1,ls);
39     }
40 }
41
42 void cit()
43 {
44     int i;
45     fin>>n;
46     for(i=1;i<=n;i++)
47         fin>>a[i];
48 }
49
50 void rez()
51 {
52     int i,j,min1,min2,t;
53     j=1;
54     i=1;
55     pas=0;
56     timp=0;
57     min1=a[j];
58     min2=a[j+1];
59     while(i<=n)
60     {
61         pas++;
62         b[0][pas]=min1;
63         t=min1;
64         i++;
65         b[1][pas]=-1;
66         if(i<n)
67         {
68             b[1][pas]=min2;
69             t=min2;
70             b[2][pas]=-1;
71             i++;
72         }
73         timp+=t;
74         if(i==n)
75         {
76             pas++;
77             b[0][pas]=min1;
78             b[1][pas]=-1;
79             timp+=min1;
80             pas++;
81             b[0][pas]=min1;
82             b[1][pas]=a[i];
83             b[2][pas]=-1;
84             timp+=a[i];
85             i++;
86         }
87     else
88     {
89         pas++;
90         b[0][pas]=min1;
91         timp+=min1;
92         b[1][pas]=-1;
93         min2=a[i];
94         i--;
95     }
96 }
97 }
98
99 void scr()
100 {
101     int i,j;

```

```

102     for (i=1; i<=pas; i++)
103     {
104         j=0;
105         while (b[j][i]!=-1)
106         {
107             fout<<b[j][i]<<" ";
108             j++;
109         }
110         fout<<endl;
111     }
112     fout<<timp;
113 }
114
115 int main()
116 {
117     cit();
118     qsort(1,n);
119     rez();
120     scr();
121     fout.close();
122     return 0;
123 }
```

32.6 SG1

O misiune a echipei SG1 constă în activarea unui dispozitiv extraterestru acționat cu ajutorul unei tastaturi ciudate formate din n comutatoare aflate inițial toate în aceeași poziție (să o notăm cu 0). Se știe că trebuie setate (trecute în poziția 1) exact k comutatoare și că nu contează ordinea în care trebuie acționate comutatoarele. În plus, cu ajutorul unor documente antice, au aflat că între oricare două comutatoare succesive setate se pot afla cel puțin d_1 și cel mult d_2 comutatoare nesetate. De exemplu, pentru $n = 7$, $k = 3$, $d_1 = 1$ și $d_2 = 2$, o configurație care corespunde cerinței este: 0100101, în timp ce configurațiile 1010001, 1100100, 1010101 nu corespund datelor problemei.

Dacă o combinație de comutatoare setate nu activează dispozitivul, nu se întâmplă nimic deosebit (ce plăcăsitor episod!), ci comutatoarele se resetează automat, permitând încercarea altelor combinații.

Se cere să se determine numărul maxim de configurații distincte de comutatoare setate pe care trebuie să le încearcă echipa SG1 pentru a activa dispozitivul.

Cerință

Scrieți un program care, pentru valorile n , k , d_1 , d_2 date, determină numărul total de configurații posibile de comutatoare ce respectă condițiile din enunț.

Datele de intrare

În fișierul text **sg1.in** se dau, pe aceeași linie, despărțite prin spații, valorile n , k , d_1 și d_2 .

Datele de ieșire

În fișierul **sg1.out** se scrie numărul de configurații ce corespund cerinței.

Restricții și precizări

- $0 < n < 101$
- $0 < k \leq n$
- $0 \leq d_1 \leq d_2 < n$

Exemple

sg1.in	sg1.out	Explicații
7 3 1 2	8	cele 8 configurații sunt: 1010100, 1010010, 1001010, 1001001, 0101010, 0101001, 0100101, 0010101
5 2 0 0	4	cele 4 configurații sunt: 11000, 01100, 00110, 00011
14 8 1 5	0	

Timp maxim de rulare/test: 1.3 secunde pentru Windows și 0.6 secunde pentru Linux

32.6.1 Indicații de rezolvare

Rodica Pintea, SNEE

Numărul de configurații de n cifre binare cu proprietatea că există exact k cifre de 1 și că între oricare două cifre 1 există cel puțin d_1 și cel mult d_2 cifre de 0 se poate determina printr-o formulă de recurență de forma:

$$S_{n,k} = \sum_{d=d_1}^{d_2} S_{n-d-1,k-1}$$

unde $S_{n,k}$ reprezintă numărul de combinații ce respectă proprietatea dată pentru un sir de n biți din care k sunt biți egali cu 1.

Recurența pornește de la $k = 1$ cu $S_{i,1} = i$ și se calculează pentru toate valorile i de la 2 la k dat.

Deoarece pentru limitele impuse în enunț se poate obține un număr mare (cu circa 30 de cifre) se impune implementarea adunării pe numere mari.

32.6.2 Rezolvare detaliată

Listing 32.6.1: sg1.java

```

1 import java.io.*;
2 class sg1
3 {
4     static StreamTokenizer st;
5     static PrintWriter out;
6
7     static int n,k,d1,d2,i,j,d;
8     static int[][][] s=new int[2][101][1];    // s[k][n][nrcifre]
9
10    public static void main(String[] args) throws IOException
11    {
12        st=new StreamTokenizer(new BufferedReader(new FileReader("sg1.in")));
13        out=new PrintWriter(new BufferedWriter(new FileWriter("sg1.out")));
14
15        st.nextToken(); n=(int)st.nval;
16        st.nextToken(); k=(int)st.nval;
17        st.nextToken(); d1=(int)st.nval;
18        st.nextToken(); d2=(int)st.nval;
19
20        // pentru k=1
21        for(i=1;i<=n;i++) s[1][i]=nr2v(i);
22
23        // pentru k=2,3,....
24        for(i=2;i<=k;i++)
25        {
26            for(j=1;j<=n;j++)
27            {
28                s[i%2][j]=nr2v(0);
29                for(d=d1;d<=d2;d++)
30                if(j-d-1>=1)
31                {
32                    s[i%2][j]=suma(s[i%2][j],s[(i+1)%2][j-d-1]);
33                }
34            }

```

```

35      } // for i
36
37      for(i=s[k%2][n].length-1;i>=0;i--)
38      {
39          System.out.print(s[k%2][n][i]);
40          out.print(s[k%2][n][i]);
41      }
42      out.println();
43      out.close();
44  } // main(...)

45
46  static int[] nr2v(int nr)
47  {
48      if(nr==0)
49      {
50          int[] x=new int[1];
51          x[0]=0;
52          return x;
53      }
54      int[] x;
55      int nc,nrrez=nr;
56      nc=0;
57      while(nr!=0) {nc++; nr=nr/10;}
58      x=new int[nc];
59      nr=nrrez;
60      nc=0;
61      while(nr!=0) {x[nc]=nr%10; nc++; nr=nr/10;}
62      return x;
63  } // nr2v(...)

64
65  static int[] suma(int[] x, int[] y)
66  {
67      int[] z;
68      int i,nx,ny,nz,t,aux;;
69      nx=x.length;
70      ny=y.length;
71
72      if(nx>ny) nz=nx+1; else nz=ny+1;
73      z=new int[nz];
74      t=0;
75      for(i=0;i<=nz-1;i++)
76      {
77          aux=t;
78          if(i<=nx-1) aux+=x[i];
79          if(i<=ny-1) aux+=y[i];
80          z[i]=aux%10;
81          t=aux/10;
82      }
83      if(z[nz-1]!=0) return z;
84      else
85      {
86          int[] zz=new int[nz-1];
87          for(i=0;i<=nz-2;i++) zz[i]=z[i];
88          return zz;
89      }
90  } // suma(...)

91 } // class

```

32.6.3 Cod sursă

Listing 32.6.2: sg1.pas

```

1 const    fi='sg1.in';
2           fo='sg1.out';
3 type nrmare=array[0..20]of byte;
4 var f:text;
5   n,k,d1,d2,i,j,d,c:byte;
6   s:array[1..2,1..100]of nrmare;
7
8 procedure add(var s2:nrmare;s1:nrmare);
9 var c,i,t:byte;
10 begin

```

```

11  while s1[0]<s2[0] do begin
12      inc(s1[0]);s1[s1[0]]:=0
13  end;
14  while s1[0]>s2[0] do begin
15      inc(s2[0]);s2[s2[0]]:=0
16  end;
17  t:=0;i:=0;
18  repeat
19      i:=i+1;
20      c:=s1[i]+s2[i]+t;
21      s2[i]:=c mod 100;
22      t:=c div 100
23  until i=s2[0];
24  if t>0 then begin
25      s2[i+1]:=t;
26      inc(s2[0])
27  end
28 end;
29
30 begin
31  {in}
32  assign(f,fi);reset(f);
33  readln(f,n,k,d1,d2);
34  close(f);
35  {recurenta}
36  {k=1}
37  for i:=1 to n do begin
38      s[1,i][1]:=i;s[1,i][0]:=1;
39  end;
40  if n=100 then begin
41      s[1,n][2]:=1;s[1,n][1]:=0;
42      s[1,n,0]:=2
43  end;
44  {k=2...}
45  for i:=2 to k do begin
46      for j:=1 to n do begin
47          s[2,j][1]:=0;s[2,j][0]:=1;
48          for c:=1 to s[2,j][0] do s[2,j][c]:=0;
49          for d:=d1 to d2 do
50              if j>d+1 then
51                  add(s[2,j],s[1,j-d-1])
52              else break
53          end;
54          for j:=1 to n do s[1,j]:=s[2,j]
55      end;
56  {out}
57  assign(f,fo);rewrite(f);
58  d:=s[1,n][s[1,n][0]];
59  if d>9 then write(f,d div 10);
60  write(f,d mod 10);
61  for i:=s[1,n][0]-1 downto 1 do
62      write(f,s[1,n][i] div 10,s[1,n][i] mod 10);
63  writeln(f);
64  close(f);
65 end.

```

Listing 32.6.3: sg1_1.pas

```

1 const fi='sg1.in';
2     fo='sg1.out';
3 type nrmare:string[50];
4 var f:text; ss:string;
5     n,k,d1,d2,i,j,d:byte;
6     s:array[1..2,1..100]of nrmare;
7
8 procedure add(var s1:nrmare;s2:nrmare);
9 var c,i,t:byte;
10 begin
11     t:=0;
12     while length(s1)<length(s2) do s1:='0'+s1;
13     while length(s2)<length(s1) do s2:='0'+s2;
14     for i:=length(s1) downto 1 do begin
15         c:=ord(s1[i])+ord(s2[i])+t-96;
16         s1[i]:=chr(48+c mod 10);
17         t:=c div 10

```

```
18   end;
19   if t>0 then s1:=chr(48+t)+s1
20 end;
21
22 begin
23   {in}
24   assign(f,fi);reset(f);
25   readln(f,n,k,d1,d2);
26   close(f);
27   {recurenta}
28   {k=1}
29   for i:=1 to n do begin
30     str(i,ss);s[1,i]:=ss
31   end;
32   {k=2...}
33   for i:=2 to k do begin
34     for j:=1 to n do begin
35       s[2,j]:='0';
36       for d:=d1 to d2 do
37         if j>d+1 then
38           add(s[2,j],s[1,j-d-1])
39     end;
40     s[1]:=s[2]
41   end;
42 {out}
43   assign(f,fo);rewrite(f);
44   writeln(f,s[1,n]);
45   close(f);
46 end.
```

Capitolul 33

ONI 2005



Figura 33.1: sigla ONI 2005

33.1 Antena

În Delta Dunării există o zonă sălbatică, ruptă de bucuriile și necazurile civilizației moderne.

În această zonă există doar n case, pozițiile acestora fiind specificate prin coordonatele carteziene de pe hartă.

Postul de radio al ONI 2005 dorește să emită pentru toți locuitorii din zonă și, prin urmare, va trebui să instaleze o antenă de emisie specială pentru aceasta.

O antenă emite unde radio într-o zonă circulară. Centrul zonei coincide cu punctul în care este poziționată antena. Raza zonei este denumită puterea antenei. Cu cât puterea antenei este mai mare, cu atât antena este mai scumpă.

Prin urmare trebuie selectată o poziție optimă de amplasare a antenei, astfel încât fiecare casă să se afle în interiorul sau pe frontieră zonei circulare în care emite antena, iar puterea antenei să fie minimă.

Cerință

Scrieți un program care să determine o poziție optimă de amplasare a antenei, precum și puterea minimă a acesteia.

Datele de intrare

Fișierul de intrare **antena.in** conține pe prima linie un număr natural n , reprezentând numărul de case din zonă. Pe următoarele n linii se află pozițiile caselor. Mai exact, pe linia $i + 1$ se află două numere întregi separate printr-un spațiu $x \ y$, ce reprezintă abscisa și respectiv ordonata casei i . Nu există două case în aceeași locație.

Datele de ieșire

Fișierul de ieșire **antena.out** conține pe prima linie două numere reale separate printr-un spațiu $x \ y$ reprezentând abscisa și ordinata poziției optime de amplasare a antenei.

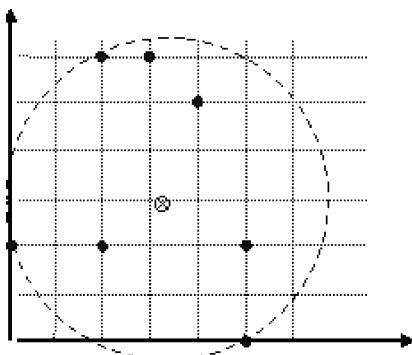
Pe cea de a doua linie se va scrie un număr real reprezentând puterea antenei.

Restricții și precizări

- $2 < N < 15001$
- $-15000 < x, y < 15001$
- Numerele reale din fișierul de ieșire trebuie scrise cu trei zecimale cu rotunjire.
- La evaluare, se verifică dacă diferența dintre soluția afișată și cea corectă (în valoare absolută) este < 0.01 .

Exemplu

antena.in	antena.out	<i>Explicație</i>
7	3.250 2.875	
5 0	3.366	
2 6		Antena va fi plasată în punctul de coordonate (3.250, 2.825) iar puterea antenei este 3.366
4 5		
2 2		
0 2		
3 6		
5 2		



Casale, antena și zona acoperită de aceasta

Figura 33.2: Antena

Timp maxim de execuție/test: 0.3 secunde pentru Windows și 0.1 secunde pentru Linux.

33.1.1 Indicații de rezolvare

prof. Osman Ay, Liceul International de Informatică București

Solutia 1

Se aleg oricare două puncte și se consideră cercul având ca diametru segmentul determinat de cele două puncte.

Verificăm dacă celelalte puncte se află în interiorul acestui cerc.

O astfel de abordare obține 20 de puncte.

Solutia 2

Se aleg oricare 3 puncte, se construiește cercul circumscris triunghiului determinat de cele 3 puncte, se verifică dacă celelalte puncte sunt în interiorul acestui cerc.

O astfel de abordare mai obține 10 puncte.

Solutia 3

O soluție mai bună: se construiește înfașuratoarea convexă pentru cele N puncte și se aleg câte 2 – 3 puncte numai de pe înfașurătoarea convexă.

Solutia oficială

0. Plasăm pe primele poziții puncte de extrem (cel mai din stânga, cel mai din dreapta, cel mai de sus, cel mai de jos).

1. Construim cercul având ca diametru segmentul determinat de primele două puncte.
2. Verificăm dacă următorul punct se află în interiorul acestui cerc.

Continuăm verificările până când gasim un punct care nu se află în interiorul acestui cerc. Când determinam un astfel de punct,

2.1. construim cercul de rază minimă care conține toate punctele precedente și acest nou punct.

Pentru a construi noul cerc, considerăm primul punct și noul punct determinat.

Verificăm dacă toate celelalte puncte sunt interiorul acestui cerc.

Dacă nu,

2.1.1. construim un nou cerc care va contine 3 puncte (primul, noul punct și cel care nu aparține interiorului acestui cerc), etc.

Cu optimizarea initială, solutia obține 100 de puncte.

33.1.2 Rezolvare detaliată

Listing 33.1.1: antena.java

```

1 import java.io.*; // practic, trebuie sa determinam cele trei puncte
2 class Antena // prin care trebuie cercul care le acopera pe toate!!!
3 {
4     static int n;
5     static int[] x,y;
6     static double x0, y0, r0;
7
8     public static void main(String[] args) throws IOException
9     {
10         int k;
11         long t1,t2;
12         t1=System.nanoTime();
13         StreamTokenizer st=new StreamTokenizer(
14             new BufferedReader(new FileReader("antena.in")));
15         PrintWriter out=new PrintWriter(
16             new BufferedWriter(new FileWriter("antena.out")));
17
18         st.nextToken(); n=(int)st.nval;
19         x=new int[n+1];
20         y=new int[n+1];
21
22         for(k=1;k<=n;k++)
23         {
24             st.nextToken(); x[k]=(int)st.nval;
25             st.nextToken(); y[k]=(int)st.nval;
26         }
27
28         if(n>3)
29         {
30             puncteExtreme();
31             cercDeDiametru(x[1],y[1],x[2],y[2]);
32             for(k=3;k<=n;k++)
33                 if(!esteInCerc(k))
34                     cercPrin(x[k],y[k],k-1); // trebuie prin Pk si acopera 1,2,...,k-1
35         }
36         else cercCircumscris(x[1],y[1],x[2],y[2],x[3],y[3]);
37
38         // scriere cu 3 zecimale rotunjite
39         out.print( (double)((int)((x0+0.0005)*1000))/1000+" ");
40         out.println((double)((int)((y0+0.0005)*1000))/1000);
41         out.println((double)((int)((r0+0.0005)*1000))/1000);
42         out.close();
43         t2=System.nanoTime();
44         System.out.println("Timp = "+((double)(t2-t1))/1000000000);
45     }// main...
46
47     // trebuie prin (xx,yy) si acopera punctele 1,2,...,k
48     static void cercPrin(int xx, int yy, int k)
49     {
50         int j;
51         cercDeDiametru(x[1],y[1],xx,yy); // trebuie prin P1 si (xx,yy)
52

```

```

53     for(j=2;j<=k;j++)
54         if(!esteInCerc(j))
55             cercPrin(xx,yy,x[j],y[j],j-1); // ... acopera 1,2,...,j-1
56     }// cercPrin(...)

57
58 // trece prin (xx,yy) si (xxx,yyy) si acopera 1,2,3,...,j
59 static void cercPrin(int xx,int yy,int xxx,int yyy,int j)
60 {
61     int i;
62     cercDeDiametru(xx,yy,xxx,yyy);
63     for(i=1;i<=j;i++)          // acopera 1,2,...,j
64         if(!esteInCerc(i))
65             cercCircumscris(xx,yy,xxx,yyy,x[i],y[i]);
66     }// cercPrin(...)

67
68 static boolean esteInCerc(int k)
69 {
70     if(d(x[k],y[k],x0,y0)<r0+0.0001) return true; else return false;
71 }
72
73 static void puncteExtreme()
74 {
75     int k,aux,min,max,kmin,kmax;
76
77     // caut cel mai din stanga punct (si mai jos) si-l pun pe pozitia 1
78     // (caut incepand cu pozitia 1)
79     kmin=1; min=x[1];
80     for(k=2;k<=n;k++)
81         if((x[k]<min) || (x[k]==min) && (y[k]<y[kmin])) {min=x[k]; kmin=k;}
82     if(kmin!=1) swap(1,kmin);

83
84     // caut cel mai din dreapta (si mai sus) punct si-l pun pe pozitia 2
85     // (caut incepand cu pozitia 2)
86     kmax=2; max=x[2];
87     for(k=3;k<=n;k++)
88         if((x[k]>max) || (x[k]==max) && (y[k]>y[kmax])) {max=x[k]; kmax=k;}
89     if(kmax!=2) swap(2,kmax);

90
91     // caut cel mai de jos (si mai la dreapta) punct si-l pun pe pozitia 3
92     // (caut incepand cu pozitia 3)
93     kmin=3; min=y[3];
94     for(k=4;k<=n;k++)
95         if((y[k]<min) || (y[k]==min) && (x[k]>x[kmin])) {min=y[k]; kmin=k;}
96     if(kmin!=3) swap(3,kmin);

97
98     // caut cel mai de sus (si mai la stanga) punct si-l pun pe pozitia 4
99     // (caut incepand cu pozitia 4)
100    kmax=4; max=y[4];
101    for(k=5;k<=n;k++)
102        if((y[k]>max) || (y[k]==max) && (x[k]<x[kmax])) {max=y[k]; kmax=k;}
103    if(kmax!=4) swap(4,kmax);

104
105    if(d(x[1],y[1],x[2],y[2])<d(x[3],y[3],x[4],y[4])) // puncte mai departate
106    {
107        swap(1,3);
108        swap(2,4);
109    }
110 } // puncteExtreme()

111
112 static void cercCircumscris(int x1,int y1,int x2,int y2,int x3,int y3)
113 { // consider ca punctele nu sunt coliniare !
114     // (x-x0)^2+(y-y0)^2=r^2 ecuatie cercului verificata de punctele P1,P2,P3
115     // 3 ecuatii si 3 necunoscute: x0, y0, r
116
117     double a12, a13, b12, b13, c12, c13; // int ==> eroare !!!
118
119     a12=2*(x1-x2); b12=2*(y1-y2); c12=x1*x1+y1*y1-x2*x2-y2*y2;
120     a13=2*(x1-x3); b13=2*(y1-y3); c13=x1*x1+y1*y1-x3*x3-y3*y3;
121
122     // sistemul devine: a12*x0+b12*y0=c12;
123     //                      a13*x0+b13*y0=c13;
124     if(a12*b13-a13*b12!=0)
125     {
126         x0=(c12*b13-c13*b12)/(a12*b13-a13*b12);
127         y0=(a12*c13-a13*c12)/(a12*b13-a13*b12);
128         r=Math.sqrt((x1-x0)*(x1-x0)+(y1-y0)*(y1-y0));

```

```

129      }
130      else // consider cercul de diametru [(minx,maxx), (miny,maxy)]
131      { // punctele sunt coliniare !
132          x0=(max(x1,x2,x3)+min(x1,x2,x3))/2;
133          y0=(max(y1,y2,y3)+min(y1,y2,y3))/2;
134          r0=d(x0,y0,x1,y1)/2;
135      }
136  } // cercCircumscris(...)

137
138  static void cercDeDiametru(int x1,int y1,int x2,int y2)
139  {
140      x0=((double)x1+x2)/2;
141      y0=((double)y1+y2)/2;
142      r0=d(x1,y1,x2,y2)/2;
143  } // cercDeDiametru(...)

144
145  static int min(int a,int b) { if(a<b) return a; else return b; }
146  static int max(int a,int b) { if(a>b) return a; else return b; }

147
148  static int min(int a,int b,int c) { return min(min(a,b),min(a,c)); }
149  static int max(int a,int b,int c) { return max(min(a,b),max(a,c)); }

150
151  static double d(int x1, int y1, int x2, int y2)
152  {
153      double dx,dy;
154      dx=x2-x1;
155      dy=y2-y1;
156      return Math.sqrt(dx*dx+dy*dy);
157  }

158
159  static double d(double x1, double y1, double x2, double y2)
160  {
161      double dx,dy;
162      dx=x2-x1;
163      dy=y2-y1;
164      return Math.sqrt(dx*dx+dy*dy);
165  }

166
167 //interschimb punctele i si j
168  static void swap(int i, int j)
169  {
170      int aux;
171      aux=x[i]; x[i]=x[j]; x[j]=aux;
172      aux=y[i]; y[i]=y[j]; y[j]=aux;
173  } // swap(...)

174 } // class

```

33.1.3 *Cod sursă

33.2 Avere

Italag a fost toată viața pasionat de speculații bursiere reușind să adune o avere considerabilă. Fiind un tip original și pasionat de matematică a scris un testament inedit. Testamentul conține două numere naturale: S reprezentând averea ce trebuie împărțită moștenitorilor și N reprezentând alegerea sa pentru împărțirea averii.

Italag decide să-și împartă toată averea, iar sumele pe care le acordă moștenitorilor să fie în ordine strict descrescătoare.

De exemplu dacă averea ar fi 7 unități monetare, ar putea fi împărțită astfel:

- 4 (unități primului moștenitor) 3 (unități celui de-al doilea), sau
- 6 (unități primului moștenitor) 1 (unitate celui de-al doilea), sau
- 7 (unități doar primului moștenitor), sau
- 5 (unități primului moștenitor) 2 (unități celui de-al doilea), sau
- 4 (unități primului moștenitor) 2 (unități celui de-al doilea) 1 (unitate celui de-al treilea).

Văzând că îi este foarte greu să verifice dacă nu cumva a omis vreo variantă de împărțire, Italag le-a scris în ordine lexicografică. Pentru exemplul de mai sus: 4 2 1; 4 3; 5 2; 6 1; 7.

A hotărât ca banii să fie distribuiți conform celei de-a N -a posibilități din ordinea lexicografică.

Cerință

Scrieți un program care pentru numerele S, N date să calculeze și să afișeze numărul total de posibilități de împărțire a averii, precum și modul în care se face această împărțire conform cu a N -a posibilitate din ordinea lexicografică.

Datele de intrare

Fișierul de intrare **avere.in** conține o singură linie pe care se află două numere naturale separate printr-un singur spațiu:

- primul număr (S) reprezintă suma totală
- cel de-al doilea (N) reprezintă numărul de ordine al poziției căutate.

Datele de ieșire

Fișierul de ieșire **avere.out** va conține două liniile:

- pe prima linie va fi afișat numărul total de modalități de împărțire a averii;
- pe cea de-a doua linie va fi afișată a N -a posibilitate de împărțire a lui S conform cerinței în ordine lexicografică. Elementele sale vor fi separate prin câte un spațiu.

Restricții și precizări

- $1 < S < 701$
- $0 < N <$ numărul total de posibilități cu suma S
- Se acordă punctaj parțial pentru fiecare test: 5 puncte pentru determinarea corectă a numărului de posibilități de împărțire a lui S și 5 puncte pentru determinarea corectă a posibilității N , din ordinea lexicografică.
- Posibilitățile de împărțire a averii sunt numerotate începând cu 1.
- Fie $x = (x_1, x_2, \dots, x_m)$ și $y = (y_1, y_2, \dots, y_p)$ două șiruri. Spunem că x precedă pe y din punct de vedere lexicografic, dacă există $k \geq 1$, astfel încât $x_i = y_i$, pentru orice $i = 1, \dots, k-1$ și $x_k < y_k$. Exemplu: 4 2 1 precedă secvența 4 3 deoarece (4 = 4, 2 < 3), iar 6 1 precedă 7 deoarece 6 < 7.

Exemple:

avere.in	avere.out
7 2	5 4 3

avere.in	avere.out
12 5	15 6 5 1

avere.in	avere.out
700 912345678912345678	962056220379782044 175 68 63 58 54 45 40 36 34 32 20 18 17 14 11 9 3 2 1

Timp maxim de execuție/test: 1 secundă pentru Windows și 0.1 secunde pentru Linux.
Limita totală de memorie sub Linux este 3Mb din care 1Mb pentru stivă.

33.2.1 Indicații de rezolvare

stud. Emilian Miron, Universitatea București

Problema se rezolvă prin *programare dinamică* după valoarea maximă pe care poate să o ia primul număr în cadrul descompunerii și după suma totală.

Obținem recurența:

$$\begin{aligned} c[v][s] &= c[v-1][s] && //\text{punând pe prima poziție } 1, 2, \dots, v-1 \\ &\quad + c[v-1][s-v] && //\text{punând pe prima poziție } v \\ c[0][0] &= 1, \\ c[0][s] &= 0 && \text{pentru } s > 0 \end{aligned}$$

Numărul total de posibilități va fi egal cu $c[S][S]$.

Reconstituirea soluției se face stabilind primul număr ca fiind cel mai mic i astfel încât $c[i][S] \geq N$ și $c[i-1][S] < N$. Procesul continuă pentru $S = S - i$ și $N = N - c[i-1][S]$ până când $N = 0$.

Observăm că recurența depinde doar de linia anterioară, așa că ea se poate calcula folosind un singur vector. Aceasta ne ajută pentru a respecta limita de memorie. Astfel calculăm toate valorile folosind un singur vector și păstrăm la fiecare pas $c[i][S]$. Reconstituirea se face recalculând valorile la fiecare pas pentru S -ul curent.

Soluția descrisă efectuează $O(S^2 * L)$ operații, unde L este lungimea soluției. Observând $L = \max(O(S^{1/2}))$ timpul total este $O(S^{3/2})$.

O soluție backtracking obține 20 puncte, iar una cu memorie $O(N^2)$ 50 puncte.

33.2.2 Rezolvare detaliată

Listing 33.2.1: averel.java

```

1 import java.io.*;      // c[0][0]=1; c[0][s]=0; pentru s=1,2,...,S
2 class Averel          // c[v][s]=c[v-1][s]; pentru v>=1 si s<v
3 {                      // c[v][s]=c[v-1][s]+c[v-1][s-v]; pentru v>=1 si s>=v
4     static int S;        // test 4 ??? test 5 = gresit N (>...) in fisier intrare
5     static long n;       // toate celelalte: OK rezultat si timp !!!
6     static long c[][];   // dar fara economie de memorie !!!!
7
8     public static void main(String[] args) throws IOException
9     {
10         long t1,t2;
11         t1=System.nanoTime();
12
13         int s,v;
14         StreamTokenizer st=new StreamTokenizer(
15             new BufferedReader(new FileReader("10-averel.in")));
16         PrintWriter out=new PrintWriter(
17             new BufferedWriter(new FileWriter("averel.out")));
18         st.nextToken(); S=(int)st.nval;
19         st.nextToken(); n=(int)st.nval;
20         c=new long[S+1][S+1];
21
22         for(v=0;v<=S;v++) c[v][0]=(long)1;
23
24         for(v=1;v<=S;v++)
25             for(s=0;s<=S;s++)
26                 if(s<v) c[v][s]=c[v-1][s]; else c[v][s]=c[v-1][s]+c[v-1][s-v];
27             //afism();
28
29         out.println(c[S][S]);
30         while((n>0)&&(S>0))
31         {
32             v=0;
33             while(c[v][S]<n) v++;
34             out.print(v+" ");
35             n=n-c[v-1][S];
36             S=S-v;
37         }
38         out.close();
39         t2=System.nanoTime();
40         System.out.println("Timp = "+((double)(t2-t1))/1000000000);
41     }// main...
42
43     static void afism()
44     {
45         int i,j;
46         for(i=0;i<=S;i++)
47         {
48             for(j=0;j<=S;j++) System.out.print(c[i][j]+" ");
49             System.out.println();
50         }
51     }// afism()
52 } // class
53 /*
54 1 0 0 0 0 0 0 0 0 0 0 0
55 1 1 0 0 0 0 0 0 0 0 0 0
56 1 1 1 1 0 0 0 0 0 0 0 0
57 1 1 1 2 1 1 1 0 0 0 0 0
58 1 1 1 2 2 2 2 1 1 1 0 0
59 1 1 1 2 2 3 3 3 3 3 2 2
60 1 1 1 2 2 3 4 4 4 5 5 5
61 1 1 1 2 2 3 4 5 5 6 7 7 8
62 1 1 1 2 2 3 4 5 6 7 8 9 10
63 1 1 1 2 2 3 4 5 6 8 9 10 12
64 1 1 1 2 2 3 4 5 6 8 10 11 13
65 1 1 1 2 2 3 4 5 6 8 10 12 14

```

```

66 1 1 1 2 2 3 4 5 6 8 10 12 15
67 Press any key to continue...
68 */

```

33.2.3 *Cod sursă

33.3 Păianjen

Un păianjen a ţesut o plasă, în care nodurile sunt dispuse sub forma unui caroaj cu m linii (numerotate de la 0 la $m - 1$) și n coloane (numerotate de la 0 la $n - 1$) ca în figură. Inițial, oricare două noduri vecine (pe orizontală sau verticală) erau unite prin segmente de plasă de lungime 1. În timp unele porțiuni ale plasei s-au deteriorat, devenind nesigure. Pe plasă, la un moment dat, se găsesc păianjenul și o muscă, în noduri de coordonate cunoscute.

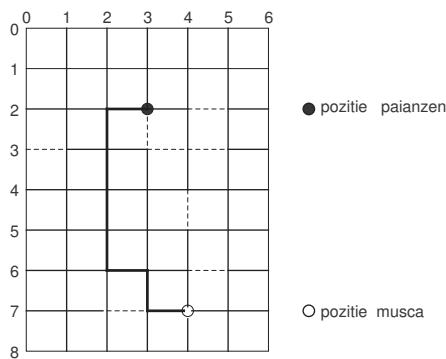


Figura 33.3: Paianjen

Cerință

Să se determine lungimea celui mai scurt traseu pe care trebuie să-l parcurgă păianjenul, folosind doar porțiunile sigure ale plasei, pentru a ajunge la muscă. De asemenea, se cere un astfel de traseu.

Datele de intrare

Fișierul de intrare **paianjen.in** conține:

- pe prima linie două numere naturale m n , separate printr-un spațiu, reprezentând numărul de linii și respectiv numărul de coloane ale plasei;
- pe a doua linie două numere naturale lp cp , separate printr-un spațiu, reprezentând linia și respectiv coloana nodului în care se află inițial păianjenul;
- pe linia a treia două numere naturale lm cm separate printr-un spațiu, reprezentând linia și respectiv coloana pe care se află inițial muscă;
- pe linia a patra, un număr natural k , reprezentând numărul de porțiuni de plasă deteriorate;
- pe fiecare dintre următoarele k linii, câte patru valori naturale $l1$ $c1$ $l2$ $c2$, separate prin câte un spațiu, reprezentând coordonatele capetelor celor k porțiuni de plasă deteriorate (linia și apoi coloana pentru fiecare capăt).

Datele de ieșire

Fișierul de ieșire **paianjen.out** va conține pe prima linie un număr natural min reprezentând lungimea drumului minim parcurs de păianjen, exprimat în număr de segmente de lungime 1. Pe următoarele $min + 1$ linii sunt scrise nodurile prin care trece păianjenul, câte un nod pe o linie. Pentru fiecare nod sunt scrise linia și coloana pe care se află, separate printr-un spațiu.

Restricții și precizări

- $1 \leq m, n \leq 140$
- $1 \leq k \leq 2 * (m * n - m - n + 1)$
- Lungimea drumului minim este cel mult 15000

• Pentru datele de test există întotdeauna soluție. Dacă problema are mai multe soluții, se va afișa una singură.

• Porțiunile nesigure sunt specificate în fișierul de intrare într-o ordine oarecare. Oricare două porțiuni nesigure orizontale se pot intersecta cel mult într-un capăt. De asemenea, oricare două porțiuni nesigure verticale se pot intersecta cel mult într-un capăt.

• Se acordă 30% din punctaj pentru determinarea lungimii drumului minim și 100% pentru rezolvarea ambelor cerințe.

Exemplu

<i>paianjen.in</i>	<i>paianjen.out</i>	<i>Explicație</i>
9 7	8	Problema corespunde figurii de mai sus.
2 3	2 3	Traseul optim este desenat cu linie groasă, iar porțiunile nesigure sunt desenate punctat.
7 4	2 2	
8	3 2	
2 4 2 5	4 2	
2 3 3 3	5 2	
3 0 3 1	6 2	
3 3 3 5	6 3	
4 4 5 4	7 3	
6 4 6 5	7 4	
6 5 7 5		
7 2 7 3		

Timp maxim de execuție/test: 1 secundă pentru Windows și 0.1 secunde pentru Linux.

33.3.1 Indicații de rezolvare

prof. Carmen Popescu, C. N. "Gh. Lazăr" Sibiu

Plasa de păianjen se memorează într-o matrice A cu M linii și N coloane, fiecare element reprezentând un nod al plasei. $A[i, j]$ va codifica pe patru biți direcțiile în care se poate face deplasarea din punctul (i, j) : bitul 0 este 1 dacă păianjenul se poate deplasa în sus, bitul 1 este 1 dacă se poate deplasa la dreapta, bitul 2 - în jos, bitul 3 - la stânga.

Rezolvarea se bazează pe parcurgerea matriciei și reținerea nodurilor parcuse într-o *structură de date de tip coadă* (parcure BF - algoritm Lee).

Drumul minim al păianjenului se reține într-o altă matrice B , unde $B[i, j]$ este 0 dacă nodul (i, j) nu este atins, respectiv o valoare pozitivă reprezentând pasul la care a ajuns păianjenul în drumul lui spre muscă. Deci elementul $B[lm, cm]$ va conține lungimea drumului minim.

Reconstituirea drumului minim se face pornind de la poziția muștei, utilizând, de asemenea un algoritm de tip BF, cu oprirea căutării în jurul nodului curent în momentul detectării unui nod de pas mai mic cu o unitate decât cel al nodului curent.

TESTE					
#	m	n	k	min	Obs
0	10	8	11	16	dimensiune mică, fire puține rupte
1	10	8	46	74	soluție unică, foarte multe fire rupte
2	2	2	0	2	caz particular, nici un fir rupt
3	140	140	20	278	fire puține rupte, dimensiune mare
4	131	131	2000	103	multe fire rupte, dimensiune mare
5	100	130	12771	12999	traseu în spirală soluție unică
6	100	7	23	15	traseu scurt, greu de găsit cu backtracking
7	138	138	9381	9050	multe fire rupte, drum în "serpentine"
8	140	140	38365	555	firele interioare rupte, traseul pe frontieră
9	138	138	273	274	o "barieră" pe mijlocul tablei, cu o "fantă"

O soluție backtracking simplu obține maxim 20 de puncte, iar îmbunătățit maxim 30 de puncte.

33.3.2 Rezolvare detaliată

Varianta 1:

Listing 33.3.1: paianjen1.java

```

1 import java.io.*; // Prima varianta: citiri + initializari + structura !!!
2 class Paianjen1 // 140*140=19.600 spatiu !
3 {
4     static final int sus=1, dreapta=2, jos=4, stanga=8;
5     static int[][] p=new int[140][140]; // plasa
6     static int[][] c=new int[140][140]; // costul ajungerii in (i,j)
7     static int m,n; // dimensiunile plasei
8     static int lp, cp; // pozitie paianjen(lin,col)
9     static int lm, cm; // pozitie musca(lin,col)
10    static int costMin;
11
12    public static void main(String[] args) throws IOException
13    {
14        long t1,t2;
15        t1=System.currentTimeMillis();
16        citescDate(); afism(p);
17        matriceCosturi();
18        afisSolutia();
19        t2=System.currentTimeMillis();
20        System.out.println("TIME = "+(t2-t1)+" millisec ");
21    }// main()
22
23    static void citescDate() throws IOException
24    {
25        int nrSegmenteDeteriorate, k,i,j,l1,c1,l2,c2;
26        int i1,i2,j1,j2;
27
28        StreamTokenizer st=new StreamTokenizer(
29                                new BufferedReader(new FileReader("paianjen.in")));
30        st.nextToken(); m=(int)st.nval;
31        st.nextToken(); n=(int)st.nval;
32        st.nextToken(); lp=(int)st.nval;
33        st.nextToken(); cp=(int)st.nval;
34        st.nextToken(); lm=(int)st.nval;
35        st.nextToken(); cm=(int)st.nval;
36        st.nextToken(); nrSegmenteDeteriorate=(int)st.nval;
37
38        for(i=0;i<m;i++) for(j=0;j<n;j++) p[i][j]=0xF; // 1111
39        for(k=1;k<=nrSegmenteDeteriorate;k++)
40        {
41            st.nextToken(); l1=(int)st.nval;
42            st.nextToken(); c1=(int)st.nval;
43            st.nextToken(); l2=(int)st.nval;
44            st.nextToken(); c2=(int)st.nval;
45
46            i1=min(l1,l2); i2=max(l1,l2);
47            j1=min(c1,c2); j2=max(c1,c2);
48
49            if(j1==j2) // ruptura verticala
50            {
51                p[i1][j1]^=jos; // sau ... p[i1][j1] -=jos; ... !!!
52                for(i=i1+1;i<=i2-1;i++)
53                {
54                    p[i][j1]^=jos;
55                    p[i][j1]^=sus;
56                }
57                p[i2][j1]^=sus; // 0 pe directia sus
58            }
59            else
60            if(i1==i2) // ruptura orizontala
61            {
62                p[i1][j1]^=dreapta; // 0 pe directia dreapta
63                for(j=j1+1;j<=j2-1;j++)
64                {
65                    p[i1][j]^=dreapta;
66                    p[i1][j]^=stanga;
67                }
68                p[i1][j2]^=stanga; // 0 pe directia stanga
69            }
70            else System.out.println("Date de intrare ... eronate !");
71
72            //afism(p);
73        }// for k
74    } //citescDate()

```

```

75
76     static void matriceCosturi()
77     {
78         // ...
79     }//matriceCosturi()
80
81     static int min(int a, int b)
82     {
83         if(a<b) return a; else return b;
84     }// min(...)
85
86     static int max(int a, int b)
87     {
88         if(a>b) return a; else return b;
89     }// max(...)
90
91     static void determinaTraseu()
92     {
93         // ...
94     }//determinaTraseu()
95
96     static void afisSolutia() throws IOException
97     {
98         PrintWriter out=new PrintWriter(
99                         new BufferedWriter(new FileWriter("paianjen.out")));
100        out.println(costMin);
101
102        out.close();
103    }//afisSolutia()
104
105    static void afism(int[][] a)
106    {
107        int i,j;
108        for(i=0;i<m;i++)
109        {
110            for(j=0;j<n;j++)
111                if(p[i][j]>=10) System.out.print(p[i][j]+" ");
112                else System.out.print(" "+p[i][j]+" ");
113            System.out.println();
114        }
115        System.out.println();
116    }// afism(...)
117 } // class

```

Varianta 2:

Listing 33.3.2: paianjen2.java

```

1 import java.io.*; // Varianta 2: a prins musca !
2 class Paianjen2 // 140*140=19.600 spatiu !
3 {
4     static final int sus=1, dreapta=2, jos=4, stanga=8;
5     static int[][] p=new int[140][140]; // plasa
6     static int[][] c=new int[140][140]; // costul ajungerii in (i,j)
7     static int m,n; // dimensiunile plasei
8     static int lp,cp; // pozitie paianjen(lin,col)
9     static int lm,cm; // pozitie musca(lin,col)
10    static int[] qi=new int[100]; // coada pentru pozitia (i,j)
11    static int[] qj=new int[100]; // coada pentru pozitia (i,j)
12    static int ic, sc; // inceput/sfarsit coada
13
14    public static void main(String[] args) throws IOException
15    {
16        long t1,t2;
17        t1=System.currentTimeMillis();
18        citescDate(); afism(p);
19        matriceCosturi(); afism(c);
20        afisSolutia();
21        t2=System.currentTimeMillis();
22        System.out.println("TIME = "+(t2-t1)+" millisec ");
23    }// main()
24
25    static void citescDate() throws IOException
26    {
27        int nrSegmenteDeteriorate, k,i,j,l1,c1,l2,c2;
28        int i1,i2,j1,j2;

```

```

29
30     StreamTokenizer st=new StreamTokenizer(
31             new BufferedReader(new FileReader("paianjen.in")));
32     st.nextToken(); m=(int)st.nval;
33     st.nextToken(); n=(int)st.nval;
34     st.nextToken(); lp=(int)st.nval;
35     st.nextToken(); cp=(int)st.nval;
36     st.nextToken(); lm=(int)st.nval;
37     st.nextToken(); cm=(int)st.nval;
38     st.nextToken(); nrSegmenteDeteriorate=(int)st.nval;
39
40     for(i=0;i<m;i++) for(j=0;j<n;j++) p[i][j]=0xF; // 1111
41     for(k=1;k<=nrSegmenteDeteriorate;k++)
42     {
43         st.nextToken(); l1=(int)st.nval;
44         st.nextToken(); c1=(int)st.nval;
45         st.nextToken(); l2=(int)st.nval;
46         st.nextToken(); c2=(int)st.nval;
47
48         i1=min(l1,l2); i2=max(l1,l2);
49         j1=min(c1,c2); j2=max(c1,c2);
50
51         if(j1==j2) // ruptura verticala
52         {
53             p[i1][j1]^=jos; // sau ... p[i1][j1]^=jos; ...
54             for(i=i1+1;i<=i2-1;i++)
55             {
56                 p[i][j1]^=jos;
57                 p[i][j1]^=sus;
58             }
59             p[i2][j1]^=sus; // 0 pe directia sus
60         }
61         else
62             if(i1==i2) // ruptura orizontala
63             {
64                 p[i1][j1]^=dreapta; // 0 pe directia dreapta
65                 for(j=j1+1;j<=j2-1;j++)
66                 {
67                     p[i1][j]^=dreapta;
68                     p[i1][j]^=stanga;
69                 }
70                 p[i1][j2]^=stanga; // 0 pe directia stanga
71             }
72             else System.out.println("Date de intrare ... eronate !");
73
74             //afism(p);
75     } // for k
76 } //citescDate()

77
78 static void matriceCosturi()
79 {
80     int i,j;
81     ic=sc=0; // coada vida
82     qi[sc]=lp; qj[sc]=cp; sc++; // (lp,cp) --> coada
83     c[lp][cp]=1; // cost 1 pentru pozitie paianjen (pentru marcaj!)
84     while(ic!=sc) // coada nevida
85     {
86         i=qi[ic]; j=qj[ic]; ic++;
87         fill(i,j);
88         if(c[lm][cm]!=0) break; // a ajuns deja la musca !
89     } // while
90 } //matriceCosturi()

91
92 static void fill(int i, int j)
93 {
94     int t=c[i][j]; // timp !
95
96     if((i>0)&&(c[i-1][j]==0)&&ok(i,j,sus))
97         {c[i-1][j]=t+1; qi[sc]=i-1; qj[sc]=j; sc++;} // N
98
99     if((j+1<n)&&(c[i][j+1]==0)&&ok(i,j,dreapta))
100        {c[i][j+1]=t+1; qi[sc]=i; qj[sc]=j+1; sc++;} // E
101
102    if((i+1<m)&&(c[i+1][j]==0)&&ok(i,j,jos))
103        {c[i+1][j]=t+1; qi[sc]=i+1; qj[sc]=j; sc++;} // S
104

```

```

105     if((j>0) &&(c[i][j-1]==0)&&ok(i,j,stanga))
106         {c[i][j-1]=t+1; qi[sc]=i; qj[sc]=j-1; sc++;} // v
107     } // fil(...)
108
109     static boolean ok(int i, int j, int dir)
110     {
111         if((p[i][j]&dir)!=0) return true; else return false;
112     } // ok(...)
113
114     static int min(int a, int b)
115     {
116         if(a<b) return a; else return b;
117     }
118
119     static int max(int a, int b)
120     {
121         if(a>b) return a; else return b;
122     }
123
124     static void afisSolutia() throws IOException
125     {
126         PrintWriter out=new PrintWriter(
127             new BufferedWriter(new FileWriter("paianjen.out")));
128         out.println(c[lm][cm]-1);
129         //...
130         out.close();
131     } //afisSolutia()
132
133     static void afism(int[][] a)
134     {
135         int i,j;
136         for(i=0;i<m;i++)
137         {
138             for(j=0;j<n;j++)
139                 if(a[i][j]>=10) System.out.print(a[i][j]+" ");
140                 else System.out.print(" "+a[i][j]+" ");
141             System.out.println();
142         }
143         System.out.println();
144     } // afism(...)
145 } // class

```

Varianta 3:

Listing 33.3.3: paianjen3.java

```

1 import java.io.*; // Avem si traseul ! - exista mai multe variante !!!
2 class Paianjen3 // Aici este recursivitate pentru traseu dar ...
3 {
4     static final int sus=1, dreapta=2, jos=4, stanga=8;
5     static int[][] p=new int[140][140]; // plasa
6     static int[][] c=new int[140][140]; // costul ajungerii in (i,j)
7     static int[][] d=new int[140][140]; // directii pentru intoarcere de la musca!
8
9     static int m,n; // dimensiunile plasei
10    static int lp, cp; // pozitie paianjen(lin,col)
11    static int lm, cm; // pozitie musca(lin,col)
12    static int[] qi=new int[100]; // coada pentru pozitia (i,j)
13    static int[] qj=new int[100]; // coada pentru pozitia (i,j)
14    static int ic, sc; // inceput/sfarsit coada
15    static PrintWriter out; // scriu si in traseu(...) !
16
17    public static void main(String[] args) throws IOException
18    {
19        long t1,t2;
20        t1=System.currentTimeMillis();
21        citescDate(); afism(p);
22        matriceCosturi(); afism(c); afism(d);
23        afisSolutia();
24        t2=System.currentTimeMillis();
25        System.out.println("\nTIME = "+(t2-t1)+" millisec ");
26    } // main()
27
28    static void citescDate() throws IOException
29    {
30        int nrSegmenteDeteriorate, k,i,j,l1,c1,l2,c2;

```

```

31     int i1,i2,j1,j2;
32
33     StreamTokenizer st=new StreamTokenizer(
34         new BufferedReader(new FileReader("paianjen.in")));
35     st.nextToken(); m=(int)st.nval;
36     st.nextToken(); n=(int)st.nval;
37     st.nextToken(); lp=(int)st.nval;
38     st.nextToken(); cp=(int)st.nval;
39     st.nextToken(); lm=(int)st.nval;
40     st.nextToken(); cm=(int)st.nval;
41     st.nextToken(); nrSegmenteDeteriorate=(int)st.nval;
42
43     for(i=0;i<m; i++) for(j=0; j<n; j++) p[i][j]=0xF; // 1111
44     for(k=1; k<=nrSegmenteDeteriorate; k++)
45     {
46         st.nextToken(); l1=(int)st.nval;
47         st.nextToken(); c1=(int)st.nval;
48         st.nextToken(); l2=(int)st.nval;
49         st.nextToken(); c2=(int)st.nval;
50
51         i1=min(l1,l2); i2=max(l1,l2);
52         j1=min(c1,c2); j2=max(c1,c2);
53
54         if(j1==j2) // ruptura verticala
55         {
56             p[i1][j1]^=jos; // sau ... p[i1][j1]==jos; ... !!!
57             for(i=i1+1; i<=i2-1; i++)
58             {
59                 p[i][j1]^=jos;
60                 p[i][j1]^=sus;
61             }
62             p[i2][j1]^=sus; // 0 pe directia sus
63         }
64         else
65             if(i1==i2) // ruptura orizontala
66             {
67                 p[i1][j1]^=dreapta; // 0 pe directia dreapta
68                 for(j=j1+1; j<=j2-1; j++)
69                 {
70                     p[i1][j]^=dreapta;
71                     p[i1][j]^=stanga;
72                 }
73                 p[i1][j2]^=stanga; // 0 pe directia stanga
74             }
75             else System.out.println("Date de intrare ... eronate !");
76     }// for k
77 } //citescDate()
78
79 static void matriceCosturi()
80 {
81     int i,j;
82     ic=sc=0; // coada vida
83     qi[sc]=lp; qj[sc]=cp; sc++; // (lp,cp) --> coada
84     c[lp][cp]=1; // cost 1 pentru pozitie paianjen (pentru marcaj!)
85     while(ic!=sc) // coada nevida
86     {
87         i=qi[ic]; j=qj[ic]; ic++;
88         fill(i,j);
89         if(c[lm][cm]!=0) break; // a ajuns deja la musca !
90     } // while
91 } //matriceCosturi()
92
93 static void fill(int i, int j)
94 {
95     int t=c[i][j]; // timp !
96
97     if((i>0)&&(c[i-1][j]==0)&&ok(i,j,sus))
98         {c[i-1][j]=t+1; qi[sc]=i-1; qj[sc]=j; sc++; d[i-1][j]=jos;} // N
99
100    if((j+1<n)&&(c[i][j+1]==0)&&ok(i,j,dreapta))
101        {c[i][j+1]=t+1; qi[sc]=i; qj[sc]=j+1; sc++; d[i][j+1]=stanga;} // E
102
103    if((i+1<m)&&(c[i+1][j]==0)&&ok(i,j,jos))
104        {c[i+1][j]=t+1; qi[sc]=i+1; qj[sc]=j; sc++; d[i+1][j]=sus;} // S
105
106    if((j>0) &&(c[i][j-1]==0)&&ok(i,j,stanga))

```

```

107     {c[i][j-1]=t+1; qi[sc]=i; qj[sc]=j-1; sc++; d[i][j-1]=dreapta;} // V
108 } // fil(...)
109
110 static boolean ok(int i, int j, int dir)
111 {
112     if((p[i][j]&dir)!=0) return true; else return false;
113 } // ok(...)
114
115 static int min(int a, int b)
116 {
117     if(a<b) return a; else return b;
118 }
119
120 static int max(int a, int b)
121 {
122     if(a>b) return a; else return b;
123 }
124
125 static void afisSolutia() throws IOException
126 {
127     out=new PrintWriter(new BufferedWriter(
128         new FileWriter("paianjen.out")));
129     out.println(c[lm][cm]-1);
130     traseu(lm,cm);
131     out.close();
132 } //afisSolutia()
133
134 static void traseu(int i, int j)
135 {
136     if(c[i][j]>1)
137         if(d[i][j]==sus) traseu(i-1,j);
138         else if(d[i][j]==jos) traseu(i+1,j);
139         else if(d[i][j]==dreapta) traseu(i,j+1);
140         else if(d[i][j]==stanga) traseu(i,j-1);
141         else System.out.println("Eroare la traseu ... !");
142     System.out.println(i+" "+j);
143     out.println(i+" "+j);
144 } // traseu(...)
145
146 static void afism(int[][] a)
147 {
148     int i,j;
149     for(i=0;i<m;i++)
150     {
151         for(j=0;j<n;j++)
152             if(a[i][j]>=10) System.out.print(a[i][j]+" ");
153             else System.out.print(" "+a[i][j]+" ");
154         System.out.println();
155     }
156     System.out.println();
157 } // afism(...)
158 } // class
159 /*
160 15 15 15 15 15 15 15
161 15 15 15 15 15 15 15
162 15 15 15 11 13 7 15
163 13 7 15 12 5 7 15
164 15 15 15 15 11 15 15
165 15 15 15 15 14 15 15
166 15 15 15 15 13 3 15
167 15 15 13 7 15 14 15
168 15 15 15 15 15 15 15
169
170 6 5 4 3 4 5 6
171 5 4 3 2 3 4 5
172 4 3 2 1 2 5 6
173 5 4 3 4 3 6 7
174 6 5 4 5 4 5 6
175 7 6 5 6 7 6 7
176 8 7 6 7 8 7 8
177 0 8 7 8 9 0 9
178 0 0 8 0 0 0 0
179
180 2 2 2 4 8 8 8
181 2 2 2 4 8 8 8
182 2 2 2 0 8 1 1

```

```

183  1  2  1  8  1  1  1
184  2  2  1  2  1  8  8
185  2  2  1  1  2  1  1
186  2  2  1  1  1  1  1
187  0  2  1  1  1  0  1
188  0  0  1  0  0  0  0
189
190  2  3
191  2  4
192  3  4
193  4  4
194  4  5
195  5  5
196  5  4
197  6  4
198  7  4
199
200 TIME = 151 millisec
201 */

```

Varianta 4:

Listing 33.3.4: paianjen4.java

```

1 import java.io.*; // test 3 eroare date lm=144 ???
2 class Paianjen4 // coada circulara (altfel trebuie dimensiune mare !)
3 { // traseu fara recursivitate (altfel depaseste stiva !)
4     static final int qdim=200; // dimensiune coada circulara
5     static final int sus=1, dreapta=2, jos=4, stanga=8;
6     static int[][] p=new int[140][140]; // plasa
7     static int[][] c=new int[140][140]; // costul ajungerii in (i,j)
8     static int[][] d=new int[140][140]; // directii pentru intoarcere de la musca!
9
10    static int m,n; // dimensiunile plasei
11    static int lp, cp; // pozitie paianjen(lin,col)
12    static int lm, cm; // pozitie musca(lin,col)
13    static int[] qi=new int[qdim]; // coada pentru i din pozitia (i,j)
14    static int[] qj=new int[qdim]; // coada pentru j din pozitia (i,j)
15    static int ic, sc; // inceput/sfarsit coada
16
17    public static void main(String[] args) throws IOException
18    {
19        long t1,t2;
20        t1=System.currentTimeMillis();
21        citescDate();
22        matriceCosturi();
23        afisSolutia();
24        t2=System.currentTimeMillis();
25        System.out.println("TIME = "+(t2-t1)+" millisec ");
26    } // main()
27
28    static void citescDate() throws IOException
29    {
30        int nrSegmenteDeteriorate, k,i,j,l1,c1,l2,c2;
31        int i1,i2,j1,j2;
32
33        StreamTokenizer st=new StreamTokenizer(
34            new BufferedReader(new FileReader("paianjen.in")));
35        st.nextToken(); m=(int)st.nval;
36        st.nextToken(); n=(int)st.nval;
37        st.nextToken(); lp=(int)st.nval;
38        st.nextToken(); cp=(int)st.nval;
39        st.nextToken(); lm=(int)st.nval;
40        st.nextToken(); cm=(int)st.nval;
41        st.nextToken(); nrSegmenteDeteriorate=(int)st.nval;
42
43        for(i=0;i<m;i++) for(j=0;j<n;j++) p[i][j]=0xFF; // 1111=toate firele !
44        for(k=1;k<=nrSegmenteDeteriorate;k++)
45        {
46            st.nextToken(); l1=(int)st.nval;
47            st.nextToken(); c1=(int)st.nval;
48            st.nextToken(); l2=(int)st.nval;
49            st.nextToken(); c2=(int)st.nval;
50
51            i1=min(l1,l2); i2=max(l1,l2);
52            j1=min(c1,c2); j2=max(c1,c2);

```

```

53
54     if(j1==j2)           // ruptura verticala
55     {
56         p[i1][j1]^=jos;      // sau ... p[i1][j1]==jos; ... !!!
57         for(i=i1+1;i<=i2-1;i++)
58         {
59             p[i][j1]^=jos;      // 0 pe directia jos
60             p[i][j1]^=sus;      // 0 pe directia sus
61         }
62         p[i2][j1]^=sus;
63     }
64     else
65     if(i1==i2)           // ruptura orizontala
66     {
67         p[i1][j1]^=dreapta;    // 0 pe directia dreapta
68         for(j=j1+1;j<=j2-1;j++)
69         {
70             p[i1][j]^=dreapta;
71             p[i1][j]^=stanga;
72         }
73         p[i1][j2]^=stanga;    // 0 pe directia stanga
74     }
75     else System.out.println("Date de intrare ... eronate !");
76 } // for k
77 } //citescDate()

78
79 static void matriceCosturi()
80 {
81     int i,j;
82     ic=sc=0;                      // coada vida
83     qi[sc]=lp; qj[sc]=cp; sc=(sc+1)%qdim; // (lp,cp) --> coada !
84     c[lp][cp]=1; // cost 1 pentru pozitie paianjen (pentru marcaj!)
85     while(ic!=sc) // coada nevida
86     {
87         i=qi[ic]; j=qj[ic]; ic=(ic+1)%qdim;
88         fill(i,j);
89         if(c[lm][cm]!=0) break; // a ajuns deja la musca !
90     } // while
91 } //matriceCosturi()

92
93 static void fill(int i, int j)
94 {
95     int t=c[i][j]; // timp !
96
97     if((i-1>=0)&&(c[i-1][j]==0)&&ok(i,j,sus)) // N
98     {
99         c[i-1][j]=t+1;
100        qi[sc]=i-1;
101        qj[sc]=j;
102        sc=(sc+1)%qdim;
103        d[i-1][j]=jos;
104    }
105
106    if((j+1<=n-1)&&(c[i][j+1]==0)&&ok(i,j,dreapta)) // E
107    {
108        c[i][j+1]=t+1;
109        qi[sc]=i;
110        qj[sc]=j+1;
111        sc=(sc+1)%qdim;
112        d[i][j+1]=stanga;
113    }
114
115    if((i+1<=m-1)&&(c[i+1][j]==0)&&ok(i,j,jos)) // S
116    {
117        c[i+1][j]=t+1;
118        qi[sc]=i+1;
119        qj[sc]=j;
120        sc=(sc+1)%qdim;
121        d[i+1][j]=sus;
122    }
123
124    if((j-1>=0)&&(c[i][j-1]==0)&&ok(i,j,stanga)) // V
125    {
126        c[i][j-1]=t+1;
127        qi[sc]=i;
128        qj[sc]=j-1;

```

```

129         sc=(sc+1)%qdim;
130         d[i][j-1]=dreapta;
131     }
132 } // fill(...)
133
134 static boolean ok(int i, int j, int dir)
135 {
136     if((p[i][j]&dir)!=0) return true; else return false;
137 } // ok(...)
138
139 static int min(int a, int b)
140 {
141     if(a<b) return a; else return b;
142 }
143
144 static int max(int a, int b)
145 {
146     if(a>b) return a; else return b;
147 }
148
149 static void afisSolutia() throws IOException
150 {
151     int i,j;
152     PrintWriter out=new PrintWriter(
153         new BufferedWriter(new FileWriter("paianjen.out")));
154     out.println(c[lm][cm]-1);
155
156     i=lm;
157     j=cm;
158     while((i!=lp) || (j!=cp)) // folosesc matricea c care nu mai e necesara !
159     {
160         if(d[i][j]==sus) { c[i-1][j]=jos; i--; }
161         else if(d[i][j]==jos) { c[i+1][j]=sus; i++; }
162         else if(d[i][j]==dreapta) { c[i][j+1]=stanga; j++; }
163         else if(d[i][j]==stanga) { c[i][j-1]=dreapta; j--; }
164         else System.out.println("Eroare la traseu ... !");
165
166         i=lp;
167         j=cp;
168     while((i!=lm) || (j!=cm))
169     {
170         out.println(i+" "+j);
171         if(c[i][j]==sus) i--;
172         else if(c[i][j]==jos) i++;
173         else if(c[i][j]==dreapta) j++;
174         else if(c[i][j]==stanga) j--;
175         else System.out.println("Eroare la traseu ... !");
176     out.println(i+" "+j); // pozitia pentru musca !
177     out.close();
178 } //afisSolutia()
179 } // class

```

33.3.3 *Cod sursă

33.4 Joc

Collapse este un joc foarte popular. Tabla de joc este reprezentată de o zonă dreptunghiulară de pe ecran, zona fiind împărțită în celule, organizate în N linii și M coloane. Fiecare celulă poate conține o piesă roșie (identificată prin litera R), verde (identificată prin litera V) sau albastră (identificată prin litera A).

Grupul unei piese este format din toate piesele care au aceeași culoare cu piesa respectivă și la care se poate ajunge deplasându-ne numai pe piese de aceeași culoare cu piesa respectivă. O deplasare se poate face în 4 direcții (sus, jos, stânga sau dreapta). Un grup trebuie să conțină cel puțin două piese.

Acționând printr-un clic pe o piesă, tot grupul de piese corespunzător piesei respective va dispărea.

După ce grupul piesei asupra căreia am actionat printr-un clic a dispărut, piesele de pe tablă se "prăbușesc". Prăbușirea se realizează executând, în ordine, următoarele două operații:

1. Mai întâi, toate piesele rămase "cad" (sunt deplasate în jos), pentru a umple golarile de pe coloane; ordinea pieselor pe coloane se păstrează.

2. Dacă o coloană se golește complet, ea va fi eliminată, deplasând celelalte coloane spre stânga, cât este posibil; ordinea coloanelor păstrându-se.

De exemplu, să considerăm tabla de joc din figura următoare:

R	R	A	R	V	R	V
R	R	R	R	V	A	V
V	R	R	R	V	A	R
V	V	R	R	R	R	R
V	V	R	R	A	V	V
V	V	R	R	A	A	A
A	A	A	R	V	V	A
A	A	A	R	V	V	R
A	R	R	R	V	R	R

Dacă executăm un clic pe piesa din colțul din stânga sus, obținem:

		A		V	R	V
				V	A	V
V				V	A	
V	V					
V	V			A	V	V
V	V			A	A	A
A	A	A		V	V	A
A	A	A		V	V	R
A				V	R	R

Apoi piesele se prăbușesc. Mai întâi "cad" pentru a umple golarile pe coloane:

				V	R	
V				V	A	V
V				V	A	V
V	V			A	V	V
V	V			A	A	A
A	V	A		V	V	A
A	A	A		V	V	R
A	A	A		V	R	R

Apoi se elimină coloanele goale:

				V	R	
V				V	A	V
V				V	A	V
V	V			A	V	V
V	V			A	A	A
A	V	A	V	V	V	A
A	A	A	V	V	R	
A	A	A	V	R	R	

Jocul se termină când pe tabla de joc nu se mai pot forma grupuri de piese.

Vasile joacă acest joc de mulți ani. Niciodată nu joacă la întâmplare, întotdeauna folosește aceeași strategie. La fiecare pas, Vasile face un clic pe o piesă din cel mai mare grup existent pe tabla de joc. Chiar dacă există mai multe posibilități de a alege piesa, el va face clic pe piesa situată pe cea mai din stânga coloană. și dacă există mai multe posibilități de a alege o piesă pe cea mai din stânga coloană, Vasile o va alege întotdeauna pe cea situată pe linia cea mai joasă.

Cerință

Scrieți un program care să simuleze jocul și care să determine numărul de clicuri pe care le execută Vasile până la terminarea jocului.

Datele de intrare

Fișierul de intrare **joc.in** conține pe prima linie două numere naturale separate printr-un spațiu $N \ M$, care reprezintă numărul de linii și respectiv numărul de coloane de pe tabla de joc. Urmează N linii, fiecare linie conținând M caractere din mulțimea '*R*', '*V*', '*A*'.

Datele de ieșire

Fișierul de ieșire **joc.out** va conține o singură linie pe care va fi scris numărul de clicuri pe care le execută Vasile până la terminarea jocului.

Restricții și precizări

- $1 \leq N, M \leq 50$

Exemple

joc.in	joc.out	joc.in	joc.out
3 4	3	8 7	7
AVVR		RRARVRV	
AAVV		RRRVAV	
AVRR		VRRVAR	
		VVRRAVV	
		VVRRAAA	
		AAARVVA	
		AAARVVR	
		ARRVRR	

Timp maxim de execuție/test: 1 secundă sub Windows și 0.2 secunde sub Linux.

33.4.1 Indicații de rezolvare

Soluția oficială

Se repetă cât timp se mai pot forma grupuri cu cel puțin 2 elemente:

1. Se determină grupurile de pe tablă. Pentru fiecare grup determinat se calculează dimensiunea și se reține dimensiunea maximă, precum și elementul cel mai din stânga-jos care aparține unui grup de dimensiune maximă.

2. Se șterge grupul de dimensiune maximă selectat.
3. Se compactează coloanele.
4. Se elimină coloanele goale.

Pentru determinarea/ștergerea unui grup se utilizează un algoritm de *fill*.

33.4.2 Rezolvare detaliată

Varianta 1:

Listing 33.4.1: joc1.java

```

1 import java.io.*;           // determinarea grupurilor
2 class Joc1
3 {
4     static final int R=1,V=2,A=3,gol=0,nemarcat=0,marcat=1;
5     static int m,n;
6     static int nemax, jmin,    imax;    // stanga_jos grup_maxim
7     static int negc,   jmingc, imaxgc; // nr elemente in grup_curent
8     static int[][] x;
9     static int[][] t;                // traseu
10
11    public static void main(String[] args) throws IOException
12    {
13        long t1,t2;
14        t1=System.currentTimeMillis();
15
16        int i,j;
17        String s;
18
19        BufferedReader br=new BufferedReader(new FileReader("joc.in"));
20        StreamTokenizer st=new StreamTokenizer(br);
21        PrintWriter out=new PrintWriter(
22            new BufferedWriter(new FileWriter("joc.out")));

```

```

23
24     st.nextToken(); m=(int)st.nval;           // m=nr linii !
25     st.nextToken(); n=(int)st.nval;           // n=nr coloane !
26     x=new int[m][n];
27     t=new int[m][n];
28
29     br.readLine();             // citeste CRLF !!!
30     for(i=0;i<m;i++)
31     {
32         s=br.readLine();      // System.out.println(s);
33         for(j=0;j<n;j++)
34             if(s.charAt(j)=='A') x[i][j]=A;
35             else if(s.charAt(j)=='R') x[i][j]=R;
36             else if(s.charAt(j)=='V') x[i][j]=V;
37             else System.out.println("Eroare in fisier de intrare !");
38     }
39     afism(x);
40
41     imax=0;
42     jmin=n+1;
43     nemax=0;
44     for(i=0;i<m;i++)
45         for(j=0;j<n;j++)
46             if(t[i][j]==nemarcat)        // un nou grup
47             {
48                 imaxgc=0;
49                 jmingc=n+1;
50                 negc=0;
51                 System.out.print("traseu: ");
52
53                 fill(i,j);
54
55                 if(negc>nemax) {nemax=negc; jmin=jmingc; imax=imaxgc;}
56                 else if((negc==nemax)&&(jmingc<jmin)) {jmin=jmingc; imax=imaxgc;}
57                 else if((negc==nemax)&&(jmingc==jmin)&&(imaxgc>imax)) imax=imaxgc;
58
59                 System.out.println("negc="+negc+"   jmingc="+jmingc+"   imaxgc="+imaxgc);
60                 afism(x);
61             }
62             System.out.println(nemax+" "+jmin+" "+imax);
63             out.println(" ... ");
64             out.close();
65             t2=System.currentTimeMillis();
66             System.out.println("Timp = "+(t2-t1));
67 } // main()
68
69 static void fill(int i, int j)
70 {
71     System.out.print(i+" "+j+"    ");
72     t[i][j]=marcat;
73     negc++;
74     if(j<jmingc) {jmingc=j; imaxgc=i;}
75     else if((j==jmingc)&&(i>imaxgc)) imaxgc=i;
76
77     if(i+1<=m-1)
78         if((t[i+1][j]==nemarcat)&&(x[i][j]==x[i+1][j])) fill(i+1,j);
79
80     if(i-1>=0)
81         if((t[i-1][j]==nemarcat)&&(x[i][j]==x[i-1][j])) fill(i-1,j);
82
83     if(j+1<=n-1)
84         if((t[i][j+1]==nemarcat)&&(x[i][j]==x[i][j+1])) fill(i,j+1);
85
86     if(j-1>=0)
87         if((t[i][j-1]==nemarcat)&&(x[i][j]==x[i][j-1])) fill(i,j-1);
88 } // fill(...)
89
90 static void afism(int[][] x)
91 {
92     int i,j;
93     for(i=0;i<m;i++)
94     {
95         for(j=0;j<n;j++)
96             if(t[i][j]==marcat) System.out.print("*");
97             else
98             {

```

```

99         if(x[i][j]==A) System.out.print("A"); else
100        if(x[i][j]==R) System.out.print("R"); else
101        if(x[i][j]==V) System.out.print("V"); else System.out.println(" ");
102    }
103    System.out.println();
104  }
105  System.out.println();
106 } // afism()
107 } // class

```

Varianta 2:

Listing 33.4.2: joc2.java

```

1 import java.io.*;           // stergerea grup_maxim
2 class Joc2
3 {
4     static final int R=1,V=2,A=3,gol=0,nemarcat=0,marcat=1;
5     static int m,n;
6     static int nemax, jmin,   imax;      // stanga_jos grup_maxim
7     static int negc,  jmingc, imaxgc;    // nr elemente in grup_curent
8     static int[][] x;
9     static int[][] t;                  // traseu
10
11    public static void main(String[] args) throws IOException
12    {
13        long t1,t2;
14        t1=System.currentTimeMillis();
15
16        int i,j;
17        String s;
18
19        BufferedReader br=new BufferedReader(new FileReader("joc.in"));
20        StreamTokenizer st=new StreamTokenizer(br);
21        PrintWriter out=new PrintWriter(
22            new BufferedWriter(new FileWriter("joc.out")));
23
24        st.nextToken(); m=(int)st.nval; // m=nr linii !
25        st.nextToken(); n=(int)st.nval; // n=nr coloane !
26
27        x=new int[m][n];
28        t=new int[m][n];
29
30        br.readLine(); // citeste CRLF !!!
31        for(i=0;i<m;i++)
32        {
33            s=br.readLine(); // System.out.println(s);
34            for(j=0;j<n;j++)
35                if(s.charAt(j)=='A') x[i][j]=A;
36                else if(s.charAt(j)=='R') x[i][j]=R;
37                else if(s.charAt(j)=='V') x[i][j]=V;
38                else System.out.println("Eroare in fisier de intrare !");
39        }
40        afism(x);
41
42        curatTraseu();
43        determinaGrupMaxim();
44        System.out.println(nemax+" "+jmin+" "+imax);
45
46        curatTraseu();
47        stergGrup(imax, jmin, x[imax][jmin]);
48        afism(x);
49
50        out.println(" ... ");
51        out.close();
52        t2=System.currentTimeMillis();
53        System.out.println("Timp = "+(t2-t1));
54    } // main()
55
56    static void stergGrup(int i, int j, int et)
57    {
58        t[i][j]=marcat;
59        x[i][j]=gol;
60
61        if(i+1<=m-1)
62            if((t[i+1][j]==nemarcat)&&(et==x[i+1][j])) stergGrup(i+1,j,et);

```

```

63
64     if(i-1>=0)
65         if((t[i-1][j]==nemarcat)&&(et==x[i-1][j])) stergGrup(i-1,j,et);
66
67     if(j+1<=n-1)
68         if((t[i][j+1]==nemarcat)&&(et==x[i][j+1])) stergGrup(i,j+1,et);
69
70     if(j-1>=0)
71         if((t[i][j-1]==nemarcat)&&(et==x[i][j-1])) stergGrup(i,j-1,et);
72     } // stergGrup(...)

73
74     static void curatTraseu()
75     {
76         int i,j;
77         for(i=0;i<m;i++) for(j=0;j<n;j++) t[i][j]=nemarcat;
78     } // curatTraseu()

79
80     static void determinaGrupMaxim()
81     {
82         int i,j;
83         imax=0; jmin=n+1; nemax=0;
84         for(i=0;i<m;i++)
85             for(j=0;j<n;j++)
86                 if(t[i][j]==nemarcat) // un nou grup
87                 {
88                     imaxgc=0;
89                     jmingc=n+1;
90                     negc=0;
91
92                     fill(i,j);
93
94                     if(negc>nemax) {nemax=negc; jmin=jmingc; imax=imaxgc;}
95                     else if((negc==nemax)&&(jmingc<jmin)) {jmin=jmingc; imax=imaxgc;}
96                     else if((negc==nemax)&&(jmingc==jmin)&&(imaxgc>imax)) imax=imaxgc;
97                 }
98     } // determinaGrupMaxim()

99
100    static void fill(int i, int j)
101    {
102        t[i][j]=marcat;
103        negc++;
104
105        if(j<jmingc) {jmingc=j; imaxgc=i;}
106        else if((j==jmingc)&&(i>imaxgc)) imaxgc=i;
107
108        if(i+1<=m-1)
109            if((t[i+1][j]==nemarcat)&&(x[i][j]==x[i+1][j])) fill(i+1,j);
110
111        if(i-1>=0)
112            if((t[i-1][j]==nemarcat)&&(x[i][j]==x[i-1][j])) fill(i-1,j);
113
114        if(j+1<=n-1)
115            if((t[i][j+1]==nemarcat)&&(x[i][j]==x[i][j+1])) fill(i,j+1);
116
117        if(j-1>=0)
118            if((t[i][j-1]==nemarcat)&&(x[i][j]==x[i][j-1])) fill(i,j-1);
119    } // fill()

120
121    static void afism(int[][] x)
122    {
123        int i,j;
124        for(i=0;i<m;i++)
125        {
126            for(j=0;j<n;j++)
127                if(t[i][j]==marcat) System.out.print(" ");
128                else
129                {
130                    if(x[i][j]==A) System.out.print("A"); else
131                    if(x[i][j]==R) System.out.print("R"); else
132                    if(x[i][j]==V) System.out.print("V"); else System.out.println("?");
133                }
134            System.out.println();
135        }
136        System.out.println();
137    } // afism()

138 } // class

```

Varianta 3:

Listing 33.4.3: joc3.java

```

1 import java.io.*;                      // si prabusire !
2 class Joc3
3 {
4     static final int R=1,V=2,A=3,gol=0,nemarcat=0,marcat=1;
5     static int m,n;
6     static int nemax, jmin,    imax;      // stanga_jos grup_maxim
7     static int negc,   jmingc, imaxgc;   // nr elemente in grup_curent
8     static int[][] x;                   // traseu
9     static int[][] t;                  // traseu
10
11    public static void main(String[] args) throws IOException
12    {
13        long t1,t2;
14        t1=System.currentTimeMillis();
15
16        int i,j;
17        String s;
18
19        BufferedReader br=new BufferedReader(new FileReader("joc.in"));
20        StreamTokenizer st=new StreamTokenizer(br);
21        PrintWriter out=new PrintWriter(
22            new BufferedWriter(new FileWriter("joc.out")));
23
24        st.nextToken(); m=(int)st.nval; // m=nr linii !
25        st.nextToken(); n=(int)st.nval; // n=nr coloane !
26
27        x=new int[m][n];
28        t=new int[m][n];
29
30        br.readLine();           // citeste CRLF !!!
31        for(i=0;i<m;i++)
32        {
33            s=br.readLine(); // System.out.println(s);
34            for(j=0;j<n;j++)
35                if(s.charAt(j)=='A') x[i][j]=A;
36                else if(s.charAt(j)=='R') x[i][j]=R;
37                else if(s.charAt(j)=='V') x[i][j]=V;
38                else System.out.println("Eroare in fisier de intrare !");
39        }
40        afism(x);
41
42        curatTraseu();
43        determinaGrupMaxim();
44        System.out.println(nemax+" "+jmin+" "+imax);
45
46        curatTraseu();
47        stergGrup(imax,jmin,x[imax][jmin]);
48        afism(x);
49
50        prabusire();
51        afism(x);
52
53        out.println(" ... ");
54        out.close();
55        t2=System.currentTimeMillis();
56        System.out.println("Timp = "+(t2-t1));
57    }// main()
58
59    static void prabusire()
60    {
61        int i1,i2,j;
62        for(j=0;j<n;j++)
63        {
64            i1=m-1;
65            i2=m-1;
66            while(i2>=0)
67            {
68                while((i1>=0)&&(x[i1][j]!=gol)) i1--; // i1 --> pe primul gol
69                if(i1<0) break;
70                i2=i1-1;
71                while((i2>=0)&&(x[i2][j]==gol)) i2--; // i2 --> pe primul plin
72                if(i2<0) break;

```

```

73         x[i1][j]=x[i2][j];
74         x[i2][j]=gol;
75     }
76 }
77 } // prabusire()
78
79 static void stergGrup(int i, int j, int et)
80 {
81     t[i][j]=marcat;
82     x[i][j]=gol;
83
84     if(i+1<=m-1)
85         if((t[i+1][j]==nemarcat)&&(et==x[i+1][j])) stergGrup(i+1,j,et);
86
87     if(i-1>=0)
88         if((t[i-1][j]==nemarcat)&&(et==x[i-1][j])) stergGrup(i-1,j,et);
89
90     if(j+1<=n-1)
91         if((t[i][j+1]==nemarcat)&&(et==x[i][j+1])) stergGrup(i,j+1,et);
92
93     if(j-1>=0)
94         if((t[i][j-1]==nemarcat)&&(et==x[i][j-1])) stergGrup(i,j-1,et);
95 } // stergGrup(...)

96
97 static void curatTraseu()
98 {
99     int i,j;
100    for(i=0;i<m;i++) for(j=0;j<n;j++) t[i][j]=nemarcat;
101 } // curatTraseu()

102
103 static void determinaGrupMaxim()
104 {
105     int i,j;
106     imax=0;
107     jmin=n+1;
108     nemax=0;
109     for(i=0;i<m;i++)
110         for(j=0;j<n;j++)
111             if(t[i][j]==nemarcat) // un nou grup
112             {
113                 imaxgc=0;
114                 jmingc=n+1;
115                 negc=0;
116
117                 fill(i,j);
118
119                 if(negc>nemax) {nemax=negc; jmin=jmingc; imax=imaxgc;}
120                 else if((negc==nemax)&&(jmingc<jmin)) {jmin=jmingc; imax=imaxgc;}
121                 else if((negc==nemax)&&(jmingc==jmin)&&(imaxgc>imax)) imax=imaxgc;
122             }
123 } // determinaGrupMaxim()

124
125 static void fill(int i, int j)
126 {
127     t[i][j]=marcat;
128     negc++;
129
130     if(j<jmingc) {jmingc=j; imaxgc=i;}
131     else if((j==jmingc)&&(i>imaxgc)) imaxgc=i;
132
133     if(i+1<=m-1)
134         if((t[i+1][j]==nemarcat)&&(x[i][j]==x[i+1][j])) fill(i+1,j);
135
136     if(i-1>=0)
137         if((t[i-1][j]==nemarcat)&&(x[i][j]==x[i-1][j])) fill(i-1,j);
138
139     if(j+1<=n-1)
140         if((t[i][j+1]==nemarcat)&&(x[i][j]==x[i][j+1])) fill(i,j+1);
141
142     if(j-1>=0)
143         if((t[i][j-1]==nemarcat)&&(x[i][j]==x[i][j-1])) fill(i,j-1);
144 } // fil(...)

145
146 static void afis(int[][] x)
147 {
148     int i,j;

```

```

149     for(i=0;i<m;i++)
150     {
151         for(j=0;j<n;j++) System.out.print(x[i][j]);
152         System.out.println();
153     }
154     System.out.println();
155 } // afism()
156
157 static void afism(int[][] x)
158 {
159     int i,j;
160     for(i=0;i<m;i++)
161     {
162         for(j=0;j<n;j++)
163             if(x[i][j]==A) System.out.print("A"); else
164             if(x[i][j]==R) System.out.print("R"); else
165             if(x[i][j]==V) System.out.print("V"); else System.out.print(" ");
166         System.out.println();
167     }
168     System.out.println();
169 } // afism()
170 } // class

```

Varianta 4:

Listing 33.4.4: joc4.java

```

1 import java.io.*;           // si eliminarea coloanelor goale !
2 class Joc4
3 {
4     static final int R=1,V=2,A=3,gol=0,nemarcat=0,marcat=1;
5     static int m,n;
6     static int nemax, jmin,           imax;           // stanga_jos grup_maxim
7     static int negc,   jmingc, imaxgc; // nr elemente in grup_curent
8     static int[][] x;
9     static int[][] t; // traseu
10
11    public static void main(String[] args) throws IOException
12    {
13        long t1,t2;
14        t1=System.currentTimeMillis();
15
16        int i,j;
17        String s;
18
19        BufferedReader br=new BufferedReader(new FileReader("joc.in"));
20        StreamTokenizer st=new StreamTokenizer(br);
21        PrintWriter out=new PrintWriter(
22            new BufferedWriter(new FileWriter("joc.out")));
23
24        st.nextToken(); m=(int)st.nval; // m=nr linii !
25        st.nextToken(); n=(int)st.nval; // n=nr coloane !
26
27        x=new int[m][n];
28        t=new int[m][n];
29
30        br.readLine(); // citeste CRLF !!!
31        for(i=0;i<m;i++)
32        {
33            s=br.readLine();
34            for(j=0;j<n;j++)
35                if(s.charAt(j)=='A') x[i][j]=A;
36                else if(s.charAt(j)=='R') x[i][j]=R;
37                else if(s.charAt(j)=='V') x[i][j]=V;
38                else System.out.println("Eroare in fisier de intrare !");
39        }
40        afism(x);
41
42        curatTraseu();
43        determinaGrupMaxim();
44        System.out.println(nemax+" "+jmin+" "+imax);
45
46        curatTraseu();
47        stergGrup(imax,jmin,x[imax][jmin]);
48        afism(x);
49

```

```

50     prabusire();
51     afism(x);
52
53     eliminColoaneGoale();
54     afism(x);
55
56     out.println(" ... ");
57     out.close();
58     t2=System.currentTimeMillis();
59     System.out.println("Timp = "+(t2-t1));
60 // main()
61
62     static void eliminColoaneGoale()
63     {
64         int i,j1,j2;
65         j1=0; j2=0;
66         while(j2<n)
67         {
68             while((j1<n)&&(x[m-1][j1]!=gol)) j1++; // j1 --> pe primul gol
69             if(j1>=n) break;
70             j2=j1+1;
71             while((j2<n)&&(x[m-1][j2]==gol)) j2++; // j2 --> pe primul plin
72             if(j2>=n) break;
73             for(i=0;i<m;i++) {x[i][j1]=x[i][j2]; x[i][j2]=gol;}
74         }
75     } // eliminColoaneGoale()
76
77     static void prabusire()
78     {
79         int i1,i2,j;
80         for(j=0;j<n;j++)
81         {
82             i1=m-1; i2=m-1;
83             while(i2>=0)
84             {
85                 while((i1>=0)&&(x[i1][j]!=gol)) i1--; // i1 --> pe primul gol
86                 if(i1<0) break;
87                 i2=i1-1;
88                 while((i2>=0)&&(x[i2][j]==gol)) i2--; // i2 --> pe primul plin
89                 if(i2<0) break;
90                 x[i1][j]=x[i2][j]; x[i2][j]=gol;
91             }
92         }
93     } //prabusire()
94
95     static void stergGrup(int i, int j, int et)
96     {
97         t[i][j]=marcat;
98         x[i][j]=gol;
99
100        if(i+1<=m-1)
101            if((t[i+1][j]==nemarcat)&&(et==x[i+1][j])) stergGrup(i+1,j,et);
102
103        if(i-1>=0)
104            if((t[i-1][j]==nemarcat)&&(et==x[i-1][j])) stergGrup(i-1,j,et);
105
106        if(j+1<=n-1)
107            if((t[i][j+1]==nemarcat)&&(et==x[i][j+1])) stergGrup(i,j+1,et);
108
109        if(j-1>=0)
110            if((t[i][j-1]==nemarcat)&&(et==x[i][j-1])) stergGrup(i,j-1,et);
111    } // stergGrup(...)
112
113    static void curatTraseu()
114    {
115        int i,j;
116        for(i=0;i<m;i++) for(j=0;j<n;j++) t[i][j]=nemarcat;
117    } // curatTraseu()
118
119    static void determinaGrupMaxim()
120    {
121        int i,j;
122        imax=0; jmin=n+1; nemax=0;
123        for(i=0;i<m;i++)
124            for(j=0;j<n;j++)
125                if(t[i][j]==nemarcat) // un nou grup

```

```

126         {
127             imaxgc=0;
128             jmingc=n+1;
129             negc=0;
130
131             fill(i,j);
132
133             if(negc>nemax) {nemax=negc; jmin=jmingc; imax=imaxgc;}
134             else if((negc==nemax)&&(jmingc<jmin)) {jmin=jmingc; imax=imaxgc;}
135             else if((negc==nemax)&&(jmingc==jmin)&&(imaxgc>imax)) imax=imaxgc;
136         }
137     }// determinaGrupMaxim()
138
139     static void fill(int i, int j)
140     {
141         t[i][j]=marcat;
142         negc++;
143
144         if(j<jmingc) {jmingc=j; imaxgc=i;}
145         else if((j==jmingc)&&(i>imaxgc)) imaxgc=i;
146
147         if(i+1<=m-1)
148             if((t[i+1][j]==nemarcat)&&(x[i][j]==x[i+1][j])) fill(i+1,j);
149
150         if(i-1>=0)
151             if((t[i-1][j]==nemarcat)&&(x[i][j]==x[i-1][j])) fill(i-1,j);
152
153         if(j+1<=n-1)
154             if((t[i][j+1]==nemarcat)&&(x[i][j]==x[i][j+1])) fill(i,j+1);
155
156         if(j-1>=0)
157             if((t[i][j-1]==nemarcat)&&(x[i][j]==x[i][j-1])) fill(i,j-1);
158     }// fil()
159
160     static void afis(int[][] x)
161     {
162         int i,j;
163         for(i=0;i<m;i++)
164         {
165             for(j=0;j<n;j++) System.out.print(x[i][j]);
166             System.out.println();
167         }
168         System.out.println();
169     }// afism()
170
171     static void afism(int[][] x)
172     {
173         int i,j;
174         for(i=0;i<m;i++)
175         {
176             for(j=0;j<n;j++)
177                 if(x[i][j]==A) System.out.print("A"); else
178                 if(x[i][j]==R) System.out.print("R"); else
179                 if(x[i][j]==V) System.out.print("V"); else System.out.print(" ");
180             System.out.println();
181         }
182         System.out.println();
183     }// afism()
184 } // class

```

Varianta 5:

Listing 33.4.5: joc5.java

```

1 import java.io.*; // si repetarea acestor pasi ... cu un for !!!
2 class Joc5        // am modificat un pic determinGrupMax() pentru ca ... !!!
3 {
4     static final int R=1,V=2,A=3,gol=0,nemarcat=0,marcat=1;
5     static int m,n;
6     static int nrClicuri;           // nr clicuri
7     static int nemax, jmin,    imax; // stanga_jos grup_maxim
8     static int negc,   jmingc,  imaxgc; // nr elemente in grup_curent
9     static int[][] x;
10    static int[][] t;              // traseu
11
12    public static void main(String[] args) throws IOException

```

```

13  {
14      long t1,t2;
15      t1=System.currentTimeMillis();
16
17      int i,j;
18      String s;
19
20      BufferedReader br=new BufferedReader(new FileReader("joc.in"));
21      StreamTokenizer st=new StreamTokenizer(br);
22      PrintWriter out=new PrintWriter(
23          new BufferedWriter(new FileWriter("joc.out")));
24      st.nextToken(); m=(int)st.nval; // m=nr linii !
25      st.nextToken(); n=(int)st.nval; // n=nr coloane !
26      x=new int[m][n];
27      t=new int[m][n];
28
29      br.readLine();           // citeste CRLF !!!
30      for(i=0;i<m;i++)
31      {
32          s=br.readLine();
33          for(j=0;j<n;j++)
34              if(s.charAt(j)=='A') x[i][j]=A;
35              else if(s.charAt(j)=='R') x[i][j]=R;
36              else if(s.charAt(j)=='V') x[i][j]=V;
37              else System.out.println("Eroare in fisier de intrare !");
38      }
39      afism(x);
40
41      for(nrClicuri=1;nrClicuri<=7;nrClicuri++)
42      {
43          System.out.println("Clic = "+nrClicuri);
44          curatTraseu();
45          determinaGrupMaxim();
46          System.out.print(nemax+" "+jmin+" "+imax+" "+x[imax][jmin]);
47
48          curatTraseu();
49          stergGrup(imax,jmin,x[imax][jmin]);
50          afism(x);
51
52          prabusire();
53          afism(x);
54
55          eliminColoaneGoale();
56          afism(x);
57      }
58
59      out.println(nrClicuri);
60      out.close();
61      t2=System.currentTimeMillis();
62      System.out.println("Timp = "+(t2-t1));
63 } // main()
64
65 static void eliminColoaneGoale()
66 {
67     int i,j1,j2;
68     j1=0;
69     j2=0;
70     while(j2<n)
71     {
72         while((j1<n)&&(x[m-1][j1]!=gol)) j1++; // j1 --> pe primul gol
73         if(j1>=n) break;
74         j2=j1+1;
75         while((j2<n)&&(x[m-1][j2]==gol)) j2++; // j2 --> pe primul plin
76         if(j2>=n) break;
77         for(i=0;i<m;i++) {x[i][j1]=x[i][j2]; x[i][j2]=gol;}
78     }
79 } // eliminColoaneGoale()
80
81 static void prabusire()
82 {
83     int i1,i2,j;
84     for(j=0;j<n;j++)
85     {
86         i1=m-1;
87         i2=m-1;
88         while(i2>=0)

```

```

89         {
90             while((i1>=0)&&(x[i1][j]!=gol)) i1--; // i1 --> pe primul gol
91             if(i1<0) break;
92             i2=i1-1;
93             while((i2>=0)&&(x[i2][j]==gol)) i2--; // i2 --> pe primul plin
94             if(i2<0) break;
95             x[i1][j]=x[i2][j]; x[i2][j]=gol;
96         }
97     }
98 } //prabusire()
99
100 static void stergGrup(int i, int j, int et)
101 {
102     t[i][j]=marcat;
103     x[i][j]=gol;
104
105     if(i+1<=m-1)
106         if((t[i+1][j]==nemarcat)&&(et==x[i+1][j])) stergGrup(i+1,j,et);
107
108     if(i-1>=0)
109         if((t[i-1][j]==nemarcat)&&(et==x[i-1][j])) stergGrup(i-1,j,et);
110
111     if(j+1<=n-1)
112         if((t[i][j+1]==nemarcat)&&(et==x[i][j+1])) stergGrup(i,j+1,et);
113
114     if(j-1>=0)
115         if((t[i][j-1]==nemarcat)&&(et==x[i][j-1])) stergGrup(i,j-1,et);
116 } // stergGrup(...)
117
118 static void curatTraseu()
119 {
120     int i,j;
121     for(i=0;i<m;i++) for(j=0;j<n;j++) t[i][j]=nemarcat;
122 } // curatTraseu()
123
124 static void determinaGrupMaxim()
125 {
126     int i,j;
127     imax=0;
128     jmin=n+1;
129     nemax=0;
130     for(i=0;i<m;i++)
131         for(j=0;j<n;j++)
132             if(x[i][j]!=gol)
133                 if(t[i][j]==nemarcat) // un nou grup
134                 {
135                     imaxgc=0;
136                     jmingc=n+1;
137                     negc=0;
138
139                     fill(i,j);
140
141                     if(negc>nemax) (nemax=negc; jmin=jmingc; imax=imaxgc;)
142                     else if((negc==nemax)&&(jmingc<jmin)) (jmin=jmingc; imax=imaxgc;)
143                     else if((negc==nemax)&&(jmingc==jmin)&&(imaxgc>imax)) imax=imaxgc;
144                 }
145 } // determinaGrupMaxim()
146
147 static void fill(int i, int j)
148 {
149     t[i][j]=marcat;
150     negc++;
151
152     if(j<jmingc) (jmingc=j; imaxgc=i;)
153     else if((j==jmingc)&&(i>imaxgc)) imaxgc=i;
154
155     if(i+1<=m-1)
156         if((t[i+1][j]==nemarcat)&&(x[i][j]==x[i+1][j])) fill(i+1,j);
157
158     if(i-1>=0)
159         if((t[i-1][j]==nemarcat)&&(x[i][j]==x[i-1][j])) fill(i-1,j);
160
161     if(j+1<=n-1)
162         if((t[i][j+1]==nemarcat)&&(x[i][j]==x[i][j+1])) fill(i,j+1);
163
164     if(j-1>=0)

```

```

165     if((t[i][j-1]==nemarcat)&&(x[i][j]==x[i][j-1])) fill(i,j-1);
166 } // fill()
167
168 static void afis(int[][] x)
169 {
170     int i,j;
171     for(i=0;i<m;i++)
172     {
173         for(j=0;j<n;j++) System.out.print(x[i][j]);
174         System.out.println();
175     }
176     System.out.println();
177 } // afism()
178
179 static void afism(int[][] x)
180 {
181     int i,j;
182     for(i=0;i<m;i++)
183     {
184         for(j=0;j<n;j++)
185             if(x[i][j]==A) System.out.print("A"); else
186             if(x[i][j]==R) System.out.print("R"); else
187             if(x[i][j]==V) System.out.print("V"); else System.out.print(" ");
188         System.out.println();
189     }
190     System.out.println();
191 } // afism()
192 } // class

```

Varianta 6:

Listing 33.4.6: joc6.java

```

1 import java.io.*; // varianta finala, dar ... !!!
2 class Joc6 // se poate optimiza timpul cu numerotare liniilor
3 { // de jos in sus si modificare m si n pe parcurs ... !!!
4     static final int R=1,V=2,A=3,gol=0,nemarcat=0,marcat=1;
5     static int m,n;
6     static int nrClicuri; // nr clicuri
7     static int nemax, jmin, imax; // stanga_jos grup_maxim
8     static int negc, jmingc, imaxgc; // nr elemente in grup_curent
9     static int[][] x;
10    static int[][] t; // traseu
11
12    public static void main(String[] args) throws IOException
13    {
14        long t1,t2;
15        t1=System.currentTimeMillis();
16
17        int i,j;
18        String s;
19
20        BufferedReader br=new BufferedReader(new FileReader("joc.in"));
21        StreamTokenizer st=new StreamTokenizer(br);
22        PrintWriter out=new PrintWriter(
23            new BufferedWriter(new FileWriter("joc.out")));
24
25        st.nextToken(); m=(int)st.nval; // m=nr linii !
26        st.nextToken(); n=(int)st.nval; // n=nr coloane !
27
28        x=new int[m][n];
29        t=new int[m][n];
30
31        br.readLine(); // citeste CRLF !!!
32        for(i=0;i<m;i++)
33        {
34            s=br.readLine();
35            for(j=0;j<n;j++)
36                if(s.charAt(j)=='A') x[i][j]=A;
37                else if(s.charAt(j)=='R') x[i][j]=R;
38                else if(s.charAt(j)=='V') x[i][j]=V;
39                else System.out.println("Eroare in fisier de intrare !");
40        }
41        nrClicuri=0;
42        nemax=2; // initializare ca sa plece while-ul !!!
43        while(nemax>1)

```

```

44      {
45        curatTraseu();
46        determinaGrupMaxim();
47        if(nemax<2) break;
48        nrClicuri++;
49        curatTraseu();
50        stergGrup(imax, jmin, x[imax][jmin]);
51        prabusire();
52        eliminColoaneGoale();
53    }
54
55    out.println(nrClicuri);
56    out.close();
57    t2=System.currentTimeMillis();
58    System.out.println("Timp = "+(t2-t1));
59 } // main()
60
61 static void eliminColoaneGoale()
62 {
63     int i,j1,j2;
64     j1=0;
65     j2=0;
66     while(j2<n)
67     {
68         while((j1<n)&&(x[m-1][j1]!=gol)) j1++; // j1 --> pe primul gol
69         if(j1>=n) break;
70         j2=j1+1;
71         while((j2<n)&&(x[m-1][j2]==gol)) j2++; // j2 --> pe primul plin
72         if(j2>=n) break;
73         for(i=0;i<m;i++) {x[i][j1]=x[i][j2]; x[i][j2]=gol;}
74     }
75 } // eliminColoaneGoale()
76
77 static void prabusire()
78 {
79     int i1,i2,j;
80     for(j=0;j<n;j++)
81     {
82         i1=m-1;
83         i2=m-1;
84         while(i2>=0)
85         {
86             while((i1>=0)&&(x[i1][j]!=gol)) i1--; // i1 --> pe primul gol
87             if(i1<0) break;
88             i2=i1-1;
89             while((i2>=0)&&(x[i2][j]==gol)) i2--; // i2 --> pe primul plin
90             if(i2<0) break;
91             x[i1][j]=x[i2][j]; x[i2][j]=gol;
92         }
93     }
94 } // prabusire()
95
96 static void stergGrup(int i, int j, int et)
97 {
98     t[i][j]=marcat;
99     x[i][j]=gol;
100
101    if(i+1<=m-1)
102        if((t[i+1][j]==nemarcat)&&(et==x[i+1][j])) stergGrup(i+1, j, et);
103
104    if(i-1>=0)
105        if((t[i-1][j]==nemarcat)&&(et==x[i-1][j])) stergGrup(i-1, j, et);
106
107    if(j+1<=n-1)
108        if((t[i][j+1]==nemarcat)&&(et==x[i][j+1])) stergGrup(i, j+1, et);
109
110    if(j-1>=0)
111        if((t[i][j-1]==nemarcat)&&(et==x[i][j-1])) stergGrup(i, j-1, et);
112 } // stergGrup(...)
113
114 static void curatTraseu()
115 {
116     int i,j;
117     for(i=0;i<m;i++) for(j=0;j<n;j++) t[i][j]=nemarcat;
118 } // curatTraseu()
119

```

```

120     static void determinaGrupMaxim()
121     {
122         int i, j;
123         imax=0;
124         jmin=n+1;
125         nemax=0;
126         for(i=0; i<m; i++)
127             for(j=0; j<n; j++)
128                 if(x[i][j] != gol)
129                     if(t[i][j]==nemarcat) // un nou grup
130                     {
131                         imaxgc=0;
132                         jmingc=n+1;
133                         negc=0;
134                         fill(i, j);
135
136                         if(negc>nemax) {nemax=negc; jmin=jmingc; imax=imaxgc;}
137                         else if((negc==nemax) && (jmingc<jmin)) {jmin=jmingc; imax=imaxgc;}
138                         else if((negc==nemax) && (jmingc==jmin) && (imaxgc>imax)) imax=imaxgc;
139                     }
140     } // determinaGrupMaxim()
141
142     static void fill(int i, int j)
143     {
144         t[i][j]=marcat;
145         negc++;
146
147         if(j<jmingc) {jmingc=j; imaxgc=i;}
148         else if((j==jmingc) && (i>imaxgc)) imaxgc=i;
149
150         if(i+1<=m-1)
151             if((t[i+1][j]==nemarcat) && (x[i][j]==x[i+1][j])) fill(i+1, j);
152
153         if(i-1>=0)
154             if((t[i-1][j]==nemarcat) && (x[i][j]==x[i-1][j])) fill(i-1, j);
155
156         if(j+1<=n-1)
157             if((t[i][j+1]==nemarcat) && (x[i][j]==x[i][j+1])) fill(i, j+1);
158
159         if(j-1>=0)
160             if((t[i][j-1]==nemarcat) && (x[i][j]==x[i][j-1])) fill(i, j-1);
161     } // fill...
162 } // class

```

33.4.3 *Cod sursă

33.5 Suma

Tradiția este ca, la ieșirea la pensie, pentru fiecare zi de activitate în slujba sultanului, marea vizir să primească o primă stabilită de marea sfat al țării. Astfel, vizirul Magir a primit pentru doar 5 zile de activitate prima totală de 411 galbeni, deoarece sfatul țării a hotărât pentru ziua întâi o sumă de 53 de galbeni, pentru ziua a doua 200 de galbeni, pentru ziua a treia 12 galbeni, pentru ziua a patra 144 de galbeni, iar pentru ziua a cincea doar 2 galbeni.

Vizirul Jibal, celebru pentru contribuția adusă la rezolvarea conflictului din zonă, primește dreptul ca, la ieșirea la pensie, să modifice sumele stabilite de sfatul țării, dar nu foarte mult. El poate uni cifrele sumelor stabilite și le poate desparti apoi, după dorință, astfel încât, suma primită pe fiecare zi să nu depășească 999 de galbeni și să primească cel puțin un galben pentru fiecare dintre zilele de activitate. Astfel, dacă are doar 5 zile de activitate, plătite cu 23, 417, 205, 5 și respectiv 40 de galbeni, în total 680 de galbeni, el poate opta pentru o nouă distribuție a cifrelor numerelor stabilite de marele sfat astfel: pentru prima zi cere 2 galbeni, pentru a doua 3, pentru a treia 417, pentru a patra 205 și pentru a cincea 540 de galbeni, primind astfel 1167 de galbeni în total.

Cerință

Pentru numărul de zile n și cele n sume stabilite de sfatul țării pentru Jibal, scrieți un program care să determine cea mai mare primă totală care se poate obține prin unirea și despartirea cifrelor

sumelor date.

Datele de intrare

Fișierul de intrare **suma.in** conține:

- pe prima linie un număr natural n reprezentând numărul de zile de activitate
- pe linia următoare, n numere naturale separate prin spații s_1, s_2, \dots, s_n reprezentând sumele atribuite de sfatul țării.

Datele de ieșire

Fișierul de ieșire **suma.out** va conține o singură linie pe care va fi afișat un singur număr natural reprezentând prima totală maximă care se poate obține.

Restricții și precizări

- $1 < n < 501$
- $0 < s_i < 1000$, pentru orice $1 \leq i \leq n$
- În orice distribuție, fiecare sumă trebuie să fie o valoare proprie (să nu înceapă cu 0).
- Orice sumă dintr-o distribuție trebuie să fie nenulă.
- Pentru 20% din teste, $n \leq 10$, pentru 50% din teste $n \leq 50$.

Exemple

suma.in	suma.out	<i>Explicație</i>
3 58 300 4	362	Prima maximă (362) se obține chiar pentru distribuția 58 300 4

suma.in	suma.out	<i>Explicație</i>
5 23 417 205 5 40	1608	Prima maximă (1608) se obține pentru distribuția 2 341 720 5 540

Timp maxim de execuție/test: 1 secundă pentru Windows și 0.1 secunde pentru Linux.

33.5.1 Indicații de rezolvare

Soluția oficială

Soluția I

Soluția propusă utilizează *metoda programării dinamice*. Este implementat un *algoritm de expandare tip Lee* realizat cu o coadă alocată dinamic.

Astfel, fiecare cifră contribuie la expandarea soluțiilor precedente care au șansă de dezvoltare ulterioară. Vectorul *best* memorează la fiecare moment suma cea mai mare formată dintr-un număr dat de termeni.

Condiția $nc - nr \leq (n - p^t - 1) * 3 + 2$ (unde nc este numărul total de cifre care se distribuie, nr este numărul de ordine al cifrei curente, n este numărul total de termeni și p^t este numărul de termeni ai soluției curente) testează ca, prin crearea unui nou termen cu ajutorul cifrei curente, să mai existe șansa construirii cu cifrele rămase a unei soluții cu n termeni.

Condiția $nc - nr >= n - p^t$ testează ca, prin lipirea cifrei curente la ultimul termen al soluției curente, să mai existe șansa construirii cu cifrele rămase a unei soluții cu n termeni.

```

type pnod=^nod;
      nod=record
        s:longint;    {suma}
        t,last:word; {nr. de termeni si ultimul termen}
        next:pnod
      end;

var n,nc,i,k:longint; f:text;
    best:array[1..1000]of longint;
    p,u:pnod;
    c:char;

procedure citire; {determină numărul total de cifre}
var i,x:longint;
begin
  assign(f,'suma.in'); reset(f);

```

```

readln(f,n);
for i:=1 to n do begin
  read(f,x);
  repeat inc(nc);x:=x div 10 until x=0
  end;
  close(f)
end;

{expandarea corespunzatoare cifrei curente}
procedure calc(nr:longint;cif:byte);
var c,q:pnod; gata:boolean;
begin
  c:=u;gata:=false;
  repeat
    if (cif>0) and (nc-nr<=(n-p^.t-1)*3+2) and (best[p^.t]=p^.s) then
      begin
        new(u^.next);u:=u^.next;
        u^.s:=p^.s+cif;
        u^.t:=p^.t+1;
        u^.last:=cif
      end;
    if (p^.last<100)and(nc-nr>=n-p^.t) then
      begin
        new(u^.next);u:=u^.next;
        u^.s:=p^.s+p^.last*9+cif;
        u^.t:=p^.t;
        u^.last:=p^.last*10+cif;
      end;
    if p=c then gata:=true;
    q:=p;p:=p^.next;dispose(q)
  until gata;
end;

{recalcularea valorilor maxime memorate in vectorul best}
procedure optim;
var i:longint;
  q:pnod; gata:boolean;
begin
  for i:=1 to n do best[i]:=0;
  q:=p;gata:=false;
  repeat
    if q^.s>best[q^.t] then best[q^.t]:=q^.s;
    if q=u then gata:=true;
    q:=q^.next
  until gata;
end;

BEGIN
  citire;
  {reluarea citirii cifrelor, ignorand spatiile}
  reset(f); readln(f);
  repeat read(f,c) until c<>' ';
  new(p);
  p^.s:=ord(c)-48;p^.t:=1;
  p^.last:=p^.s;
  best[1]:=p^.s;
  u:=p;
  for i:=2 to nc do begin
    repeat read(f,c) until c<>' ';
    calc(i,ord(c)-48);
  end;
end.

```

```

        optim
    end;
close(f);
assign(f,'suma.out'); rewrite(f); writeln(f,best[n]); close(f)
END.

```

Soluția II

Problema se rezolvă prin metoda *programare dinamică*. Concatenăm numerele și obținem un sir (să îl notăm a) de cifre (de lungime L maxim $N * 3$).

Pentru fiecare poziție p ($p = 1..L$) calculăm prima maximă pe care o putem obține despărțind subșirul de până la p inclusiv în n sume ($n = 1..N$). Obținem relația:

```

smax[p][n] = min(
    smax[p-1][n-1]+a[p], // punând ultima sumă formată doar din cifra a[i]
    smax[p-2][n-1]+a[p-1]*10+a[p], // punând ultima sumă din 2 cifre
    smax[p-3][n-1]+a[p-2]*100+a[p-1]*10+a[p] // punând ultima sumă din 3 cifre
)

```

Trebuie avute în vedere cazurile limită când $p = 1$, $p = 2$ sau $p = 3$ și cazurile în care $a[p]$, $a[p - 1]$ sau $a[p - 2]$ sunt zero, moment în care nu putem forma o sumă de lungimea respectivă, să că excludem termenii din expresia de minim.

Pentru ușurință în implementare stocăm $smax[p][n] = -\text{infinit}$ pentru cazul în care subșirul de până la p nu poate fi împărțit în mod corect în n sume, iar observând că recurența depinde doar de ultimele 3 linii, nu păstrăm decât pe acestea și linia curentă pentru a nu avea probleme cu memoria.

Obținem memorie $O(N)$ și timp de execuție $O(L * N) = O(N^2)$;

33.5.2 Rezolvare detaliată

Varianta 1:

Listing 33.5.1: sumal.java

```

1 import java.io.*; // fara economie de spatiu
2 class Sumal
3 {
4     static int n,nc; // nc=nr cifre din sir
5     static int[] c=new int[1501]; // sirul cifrelor
6     static int[][] s;
7
8     public static void main (String[] args) throws IOException
9     {
10         int i,j,x;
11         StreamTokenizer st=new StreamTokenizer(
12             new BufferedReader(new FileReader("suma.in")));
13         PrintWriter out=new PrintWriter(new BufferedWriter(
14             new FileWriter("suma.out")));
15         st.nextToken(); n=(int)st.nval;
16
17         nc=0;
18         j=0;
19         for(i=1;i<=n;i++)
20         {
21             st.nextToken(); x=(int)st.nval;
22             if(x<10) {c[++j]=x; nc+=1;} else
23                 if(x<100) {c[++j]=x/10; c[++j]=x%10; nc+=2;} else
24                     if(x<1000) {c[++j]=x/100; c[++j]=(x/10)%10; c[++j]=x%10; nc+=3;}
25             else System.out.println("Eroare date !");
26         }
27
28         s=new int[nc+1][n+1];
29         calcul();
30         afism();
31         out.print(s[nc][n]);
32         out.close();
33     }// main...
34
35     static void calcul()

```

```

36  {
37      //  s[i][j]=max(s[i-1][j-1]+c[i],                                // xj are 1: cifra c[i]
38      //          s[i-2][j-1]+c[i-1]*10+c[i], // xj are 2 cifre: c[i-1]c[i]
39      //          s[i-3][j-1]+c[i-2]*100+c[i-1]*10+c[i]); // xj are 3 cifre
40
41      int i,j,smax;
42
43      s[1][1]=c[1];
44      s[2][1]=c[1]*10+c[2];
45      s[3][1]=c[1]*100+c[2]*10+c[3];
46
47      if(c[2]!=0) s[2][2]=c[1]+c[2];
48
49      if((c[2]!=0)&&(c[3]!=0)) s[3][3]=c[1]+c[2]+c[3];
50
51      if(c[3]==0) { if(c[2]!=0) s[3][2]=c[1]+c[2]; }
52      else // c[3]!=0
53      {
54          if(c[2]==0) s[3][2]=c[1]*10+c[3];
55          else // c[2]!=0 && c[3]!=0
56          s[3][2]=max(c[1]+c[2]*10+c[3],c[1]*10+c[2]+c[3]);
57      }
58
59      for(i=4;i<=nc;i++) // i = pozitie cifra in sirul cifrelor
60      for(j=1;j<=n;j++) // j = pozitie numar in sirul final al numerelor
61      {
62          smax=0;
63
64          if(j<=i)
65          {
66              if((c[i]!=0)&&(s[i-1][j-1]!=0))
67                  smax=max(smax,s[i-1][j-1]+c[i]);
68
69              if((c[i-1]!=0)&&(s[i-2][j-1]!=0))
70                  smax=max(smax,s[i-2][j-1]+c[i-1]*10+c[i]);
71
72              if((c[i-2]!=0)&&(s[i-3][j-1]!=0))
73                  smax=max(smax,s[i-3][j-1]+c[i-2]*100+c[i-1]*10+c[i]);
74          }
75
76          s[i][j]=smax;
77      } // for
78  } // calcul()
79
80  static int max(int a, int b)
81  {
82      if(a>b) return a; else return b;
83  }
84
85  static void afism()
86  {
87      int i,j;
88
89      System.out.print("      \t");
90      for(j=1;j<=n;j++) System.out.print(j+"\t");
91      System.out.println();
92
93      for(i=1;i<=nc;i++)
94      {
95          System.out.print(i+" "+c[i]+" :\t");
96          for(j=1;j<=n;j++) System.out.print(s[i][j]+"\t");
97          System.out.println();
98      }
99  } // afism()
100 } // class

```

Varianta 2:

Listing 33.5.2: suma2.java

```

1 import java.io.*; // cu economie de spatiu !!!
2 class Suma2
3 {
4     static int n,nc; // nc=nr cifre din sir
5     static int[] c=new int[1501]; // sirul cifrelor
6     static int[][] s;

```

```

7
8     public static void main (String[] args) throws IOException
9     {
10        long t1,t2;
11        t1=System.currentTimeMillis();
12
13        int i,j,x;
14        StreamTokenizer st=new StreamTokenizer(
15            new BufferedReader(new FileReader("suma.in")));
16        PrintWriter out=new PrintWriter(new BufferedWriter(
17            new FileWriter("suma.out")));
18        st.nextToken(); n=(int)st.nval;
19
20        nc=0;
21        j=0;
22        for(i=1;i<=n;i++)
23        {
24            st.nextToken(); x=(int)st.nval;
25            if(x<10) {c[++j]=x; nc+=1;} else
26            if(x<100) {c[++j]=x/10; c[++j]=x%10; nc+=2;} else
27            if(x<1000) {c[++j]=x/100; c[++j]=(x/10)%10; c[++j]=x%10; nc+=3;}
28            else System.out.println("Eroare date !");
29        }
30
31        s=new int[4][n+1]; // cu economie de spatiu !!!
32        calcul();
33
34        out.print(s[nc%4][n]);
35        out.close();
36        t2=System.currentTimeMillis();
37        System.out.println("Timp = "+(t2-t1));
38    } // main...
39
40    static void calcul()
41    {
42        // s[i][j]=max(s[i-1][j-1]+c[i], // xj are 1: cifra c[i]
43        //                 s[i-2][j-1]+c[i-1]*10+c[i], // xj are 2 cifre: c[i-1]c[i]
44        //                 s[i-3][j-1]+c[i-2]*100+c[i-1]*10+c[i]); // xj are 3 cifre
45
46        int i,j,smax;
47
48        s[1][1]=c[1];
49        s[2][1]=c[1]*10+c[2];
50        s[3][1]=c[1]*100+c[2]*10+c[3];
51
52        if(c[2]!=0) s[2][2]=c[1]+c[2];
53
54        if((c[2]!=0)&&(c[3]!=0)) s[3][3]=c[1]+c[2]+c[3];
55
56        if(c[3]==0) { if(c[2]!=0) s[3][2]=c[1]+c[2]; }
57        else // c[3]!=0
58        {
59            if(c[2]==0) s[3][2]=c[1]*10+c[3];
60            else // c[2]!=0 && c[3]!=0
61            s[3][2]=max(c[1]+c[2]*10+c[3],c[1]*10+c[2]+c[3]);
62        }
63
64        for(i=4;i<=nc;i++) // i = pozitie cifra in sirul cifrelor
65        for(j=1;j<=n;j++) // j = pozitie numar in sirul final al numerelor
66        {
67            smax=0;
68
69            if(j<=i)
70            {
71                if((c[i]!=0)&&(s[(i-1+4)%4][j-1]!=0))
72                    smax=max(smax,s[(i-1+4)%4][j-1]+c[i]);
73
74                if((c[i-1]!=0)&&(s[(i-2+4)%4][j-1]!=0))
75                    smax=max(smax,s[(i-2+4)%4][j-1]+c[i-1]*10+c[i]);
76
77                if((c[i-2]!=0)&&(s[(i-3+4)%4][j-1]!=0))
78                    smax=max(smax,s[(i-3+4)%4][j-1]+c[i-2]*100+c[i-1]*10+c[i]);
79            }
80
81            s[i%4][j]=smax;
82        }

```

```

83 } // calcul()
84
85 static int max(int a, int b)
86 {
87     if(a>b) return a; else return b;
88 }
89 } // class

```

Varianta 3:

Listing 33.5.3: suma3.java

```

1 import java.io.*; // fara economie de spatiu dar cu afisare o solutie !
2 class Suma3
3 {
4     static int n,nc; // nc=nr cifre din sir
5     static int[] c=new int[1501]; // sirul cifrelor
6     static int[][] s;
7     static int[][] p; // predecesori
8     static int[] sol; // o solutie
9
10    public static void main (String[] args) throws IOException
11    {
12        int i,j,x;
13        StreamTokenizer st=new StreamTokenizer(
14            new BufferedReader(new FileReader("suma.in")));
15        PrintWriter out=new PrintWriter(new BufferedWriter(
16            new FileWriter("suma.out")));
17        st.nextToken(); n=(int)st.nval;
18
19        nc=0;
20        j=0;
21        for(i=1;i<=n;i++)
22        {
23            st.nextToken(); x=(int)st.nval;
24            if(x<10) {c[++j]=x; nc+=1;} else
25            if(x<100) {c[++j]=x/10; c[++j]=x%10; nc+=2;} else
26            if(x<1000) {c[++j]=x/100; c[++j]=(x/10)%10; c[++j]=x%10; nc+=3;}
27            else System.out.println("Eroare date !");
28        }
29
30        s=new int[nc+1][n+1];
31        p=new int[nc+1][n+1];
32        calcul();
33        afism(s); System.out.println();
34        afism(p); System.out.println();
35
36        sol=new int[n+1];
37        solutia();
38        afisv(sol);
39
40        out.print(s[nc][n]);
41        out.close();
42    } // main(...)
43
44    static void solutia()
45    {
46        int i,i1,i2,k;
47        i2=nc;
48        for(k=n;k>=1;k--)
49        {
50            i1=p[i2][k];
51            System.out.print(k+" : "+i1+"->"+i2+" ==> ");
52            for(i=i1;i<=i2;i++) sol[k]=sol[k]*10+c[i];
53            System.out.println(sol[k]);
54            i2=i1-1;
55        }
56    } // solutia()
57
58    static void calcul()
59    {
60        // s[i][j]=max(s[i-1][j-1]+c[i], // xj are 1: cifra c[i]
61        //                 s[i-2][j-1]+c[i-1]*10+c[i], // xj are 2 cifre: c[i-1]c[i]
62        //                 s[i-3][j-1]+c[i-2]*100+c[i-1]*10+c[i]); // xj are 3 cifre
63
64        int i,j,smax;

```

```

65
66     s[1][1]=c[1];
67     s[2][1]=c[1]*10+c[2];
68     s[3][1]=c[1]*100+c[2]*10+c[3];
69     p[1][1]=p[2][1]=p[3][1]=1;
70
71     if(c[2]!=0) s[2][2]=c[1]+c[2];
72
73     if((c[2]!=0)&&(c[3]!=0)) s[3][3]=c[1]+c[2]+c[3];
74
75     if(c[3]==0) {   if(c[2]!=0) s[3][2]=c[1]+c[2]; }
76     else // c[3]!=0
77     {
78         if(c[2]==0) { s[3][2]=c[1]*10+0+c[3]; p[3][2]=3; }
79         else // c[2]!=0 && c[3]!=0
80         {
81             s[3][2]=max(c[1]+c[2]*10+c[3],c[1]*10+c[2]+c[3]);
82             if(s[3][2]==c[1]+c[2]*10+c[3]) p[3][2]=2; else p[3][2]=3;
83         }
84     }
85
86     for(i=4;i<=nc;i++) // i = pozitie cifra in sirul cifrelor
87     for(j=1;j<=n;j++) // j = pozitie numar in sirul final al numerelor
88     {
89         smax=0;
90
91         if(j<=i)
92         {
93             if((c[i]!=0)&&(s[i-1][j-1]!=0))
94             {
95                 smax=max(smax,s[i-1][j-1]+c[i]);
96                 if(smax==s[i-1][j-1]+c[i]) p[i][j]=i;
97             }
98
99             if((c[i-1]!=0)&&(s[i-2][j-1]!=0))
100            {
101                smax=max(smax,s[i-2][j-1]+c[i-1]*10+c[i]);
102                if(smax==s[i-2][j-1]+c[i-1]*10+c[i]) p[i][j]=i-1;
103            }
104
105            if((c[i-2]!=0)&&(s[i-3][j-1]!=0))
106            {
107                smax=max(smax,s[i-3][j-1]+c[i-2]*100+c[i-1]*10+c[i]);
108                if(smax==s[i-3][j-1]+c[i-2]*100+c[i-1]*10+c[i]) p[i][j]=i-2;
109            }
110        } // if
111
112        s[i][j]=smax;
113    } // for
114 } // calcul()
115
116 static int max(int a, int b)
117 {
118     if(a>b) return a; else return b;
119 }
120
121 static void afism(int[][] x)
122 {
123     int i,j;
124
125     System.out.print("      \t");
126     for(j=1;j<=n;j++) System.out.print(j+"\t");
127     System.out.println();
128
129     for(i=1;i<=nc;i++)
130     {
131         System.out.print(i+" "+c[i]+" :\t");
132         for(j=1;j<=n;j++) System.out.print(x[i][j]+"\t");
133         System.out.println();
134     }
135 } // afism()
136
137 static void afisv(int[] sol)
138 {
139     int i,sum=0;
140     System.out.println();

```

```

141     for (i=1; i<=n; i++)
142     {
143         System.out.print(sol[i]+" ");
144         sum+=sol[i];
145     }
146     System.out.println(" ==> "+sum);
147 } // afisv()
148 } // class

```

33.5.3 *Cod sursă

33.6 Vizibil

Pentru un sir x_1, x_2, \dots, x_n spunem că elementul x_k este vizibil din stânga dacă pentru orice i , $1 \leq i < k$ avem $x_i < x_k$. Analog, spunem că x_k este vizibil din dreapta dacă pentru orice i , $k < i \leq n$ avem $x_i < x_k$ (primul element se consideră vizibil din stânga, iar ultimul din dreapta).

Cerință

Considerăm permutările mulțimii $\{1, 2, \dots, n\}$. Determinați câte dintre aceste permutări au p elemente vizibile din stânga și q elemente vizibile din dreapta.

Datele de intrare

Fișierul de intrare **vizibil.in** conține pe prima linie numerele naturale $n \ p \ q$, separate prin câte un spatiu.

Datele de ieșire

Fișierul de ieșire **vizibil.out** va conține pe prima linie numărul permutărilor care îndeplinesc condiția din enunț modulo 997.

Restricții și precizări

- $2 < n < 101$
- $0 < p, q \leq n$

Exemplu

vizibil.in	vizibil.out	<i>Explicație</i>
4 2 3	3	permutările cu 2 elemente vizibile din stânga și 3 vizibile din dreapta sunt $(1, 4, 3, 2)$, $(2, 4, 3, 1)$, $(3, 4, 2, 1)$

Timp maxim de execuție/test: 0.5 secunde pentru Windows și 0.1 secunde pentru Linux.

33.6.1 Indicații de rezolvare

Soluția oficială

Vom determina întâi numărul $S_{n,p}$ al permutărilor mulțimii $\{1, 2, \dots, n\}$ care au p elemente vizibile din stânga.

O relație de recurență pentru $S_{n,p}$ se poate găsi dacă stabilim întâi locul celui mai mic număr (obținem $S_{n,p} = S_{n-1,p-1} + (n-1) \cdot S_{n-1,p}$) sau a celui mai mare număr (obținem $S_{n,p} = \sum_{k=p}^n C_{n-1}^{k-1} \cdot S_{k-1,p-1} \cdot (n - k + 1)!$).

În continuare observăm că:

– mulțimea $\{1, 2, \dots, n\}$ poate fi înlocuită cu orice mulțime cu n elemente (distințe), numărul permutărilor cu proprietatea dată rămâne același

– numărul permutărilor unei mulțimi cu n elemente care au p elemente vizibile din dreapta este tot $S_{n,p}$

– numărul permutărilor cu p elemente vizibile din stânga și q vizibile din dreapta se obține însumând numărul de permutări în care cel mai mare număr (adică n) este pe poziția k , iar în fața lui avem o permutare a unei mulțimi cu $k-1$ elemente și $p-1$ vizibile din stânga iar după el avem o permutare a unei mulțimi cu $n-k$ elemente și $q-1$ vizibile din dreapta.

Pentru o implementare economică se pot utiliza doar doi vectori în locul matricei S .

33.6.2 Rezolvare detaliată

Listing 33.6.1: vizibil.java

```

1 import java.io.*; // fara economie de memorie !
2 class Vizibil      // 100*100=10K==> 20K sau 40K ==> destul de mic !
3 {
4     static int n,p,q,npq;
5     static int [][] s;
6
7     public static void main (String[] args) throws IOException
8     {
9         long t1,t2;
10        t1=System.currentTimeMillis();
11
12        int i,j,k;
13        StreamTokenizer st=new StreamTokenizer(
14            new BufferedReader(new FileReader("vizibil.in")));
15        PrintWriter out=new PrintWriter(new BufferedWriter(
16            new FileWriter("vizibil.out")));
17
18        st.nextToken(); n=(int)st.nval;
19        st.nextToken(); p=(int)st.nval;
20        st.nextToken(); q=(int)st.nval;
21
22        s=new int [n+1] [n+1];
23
24        s[0] [0]=1;
25        for(i=1;i<=n;i++)
26            for(j=1;j<=i;j++)
27                s[i] [j]=(s[i-1] [j-1]+(i-1)*s[i-1] [j])%997;
28
29        npq=0;
30        for(k=p;k<=n-q+1;k++)
31            npq=(npq+comb(n-1, k-1)*s[k-1] [p-1]*s[n-k] [q-1])%997;
32
33        out.println(npq);
34        out.close();
35        t2=System.currentTimeMillis();
36        System.out.println("Timp = "+(t2-t1));
37    } // main...
38
39    static int comb(int n,int k)
40    {
41        int i,j,d;
42        if(k>n/2) k=n-k;      // o mica optimizare !
43        int [] x=new int [k+1];
44        int [] y=new int [k+1];
45        for(i=1;i<=k;i++) x[i]=n-k+i;
46        for(j=1;j<=k;j++) y[j]=j;
47        for(j=2;j<=k;j++)
48            for(i=1;i<=k;i++)
49            {
50                d=cmmdc(x[i],y[j]);
51                x[i]=x[i]/d;
52                y[j]=y[j]/d;
53                if(y[j]==1) break;
54            }
55        int p=1;
56        for(i=1;i<=k;i++) p=(p*x[i])%997;
57        return p;
58    } // comb...
59
60    static int cmmdc(int a, int b)
61    {
62        int d,i,c,r;
63        if(a>b) {d=a; i=b;} else {d=b; i=a;}
64        while(i != 0)
65        {
66            c=d/i; // nu se foloseste !
67            r=d%i;
68            d=i;
69            i=r;
70        }

```

```
71     return d;
72 } // cmmdc(...)
73 } // class
```

33.6.3 *Cod sursă

Capitolul 34

ONI 2004



Figura 34.1: Sigla ONI 2004

34.1 Găina

Găina Chucky 767 trebuie să străbată curtea sărind de pe un coteț pe altul, sau zburând peste cotețe, de la primul la ultimul coteț. Cotețele sunt reprezentate prin niște dreptunghiuri de lățime $1m$ și înălțimi date și sunt numerotate în ordine de la stânga cu numerele de 1 la n . Două cotețe cu numere consecutive sunt lipite (adiacente). Inițial, Chucky are un număr de unități de energie. Dacă la un moment dat Chucky ajunge să aibă energie strict negativă sau se află în imposibilitatea de a mai face un pas (întâlneste un coteț mai înalt decât cota la care se află), atunci nu mai poate continua acel drum.

Chucky poate să se deplaseze făcând următoarele tipuri de mișcări:

a) **Pas.** Găina pășește pe orizontală de pe un coteț de înălțime H pe următorul coteț de aceeași înălțime (în desen: de la poziția h la i). În acest caz nu se pierde și nu se câștigă energie.

b) **Aterizare.** Găina aterizează pe un coteț de înălțime H , venind în zbor, tot de la înălțimea H (exemplu în desen: de la g la h). Nici în acest caz nu se pierde și nu se câștigă energie.

c) **Decolare.** Găina zboară $1m$ pe orizontală de pe un coteț (exemplu în desen de la x la a). În acest caz găina pierde o unitate de energie.

d) **Zbor orizontal.** Găina zboară pe orizontală (exemplu în desen, de la a la b , de la b la c , ...). În acest caz pierde câte o unitate de energie pentru fiecare metru parcurs pe orizontală.

e) **Picaj.** Găina coboară pe verticală (exemplu în desen de la a la d , sau de la d la g , ...). În acest caz câștigă câte o unitate de energie pentru fiecare metru coborât.

Mai jos, avem un drum format din 4 cotețe, de înălțimi 4, 1, 2, 2. Exemplificăm diferite tipuri de mișcări din poziția x (ceea ce nu reprezintă un traseu complet):

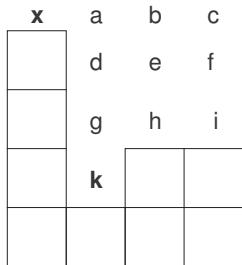


Figura 34.2: Gaina

EXEMPLU: din poziția de start **x** se poate ajunge în poziția **h** pe 3 trasee:

TRASEU	Bilanț energie pas cu pas	Necesar energie inițială minimă
1) x, a, b, e, h	$x \rightarrow a(-1) \rightarrow b(-1) \rightarrow e(+1) \rightarrow h(+1)$	2
2) x, a, d, e, h	$x \rightarrow a(-1) \rightarrow d(+1) \rightarrow e(-1) \rightarrow h(+1)$	1
3) x, a, d, g, h	$x \rightarrow a(-1) \rightarrow d(+1) \rightarrow g(+1) \rightarrow h(0)$	1

Cerință

Să se determine energia minimă (un număr natural) pe care trebuie să o aibă Chucky la începutul călătoriei (când se află pe primul cotet), astfel încât ea să poată ajunge pe cotetul n , având în fiecare moment energia mai mare sau egală cu 0.

Date de intrare

Fișierul de intrare **gaina.in** conține două linii. Pe prima linie se află numărul natural n . Pe linia a doua se află n numere naturale $h_1 h_2 \dots h_n$ (unde h_i reprezintă înălțimea cotetului i), separate de către un spațiu.

Date de ieșire

Fișierul de ieșire **gaina.out** conține o singură linie pe care se află un număr natural K reprezentând energia minimă inițială cu care găina Chucky poate să ajungă la cotetul n , având în fiecare moment energia mai mare sau egală cu 0.

Restricții

$$0 < n < 10001$$

$$0 \leq h_i \leq 30000$$

Pentru datele de test există întotdeauna soluție (primul cotet are înălțimea mai mare sau egală cu înălțimea oricărui alt cotet).

Exemple

gaina.in	gaina.out	gaina.in	gaina.out
3	1	6	2
2 2 0		3 0 0 2 1 1	

Timp maxim de execuție/test: 0.1 sec pentru Linux și 0.1 sec pentru Windows.

34.1.1 Indicații de rezolvare

Soluția oficială

Vom afla pentru fiecare poziție $m \leq n$ costul (energia minimă) pentru a ajunge de pe poziția m pe poziția n . Costul are sens pentru valorile m pentru care $h_m \geq h_p$, orice $p > m$.

Așa că mai întâi aflăm aceste valori ale lui m . Mai exact, $hmax[a]$ este cea mai mică poziție (mai mare decât a) pentru care se realizează maximul înălțimilor pe segmentul $[a + 1, n]$ iar $hmax[n] = n$ prin definiție.

Costul pentru drumul de la pozitia a la pozitia $n = cost_dist(a, hmax[a]) + cost_dist(hmax[a], n)$ unde: $cost_dist(a, hmax[a])$ = costul pentru distanța de la poziția a la poziția $hmax[a]$, fără atențări intermediare (cauză acesta este pentru economie maximă de energie), iar $cost_dist(hmax[a], n)$ se va depune în $cost[hmax[a]]$.

Costul pentru fiecare poziție se va calcula de la dreapta la stânga, pentru că pe baza lui $cost[i]$ se calculează $cost[i - 1]$, iar $cost_dist[a, hmax[a]] = (hmax[a] - a - 1) - (h[a] - h[hmax[a]])$, unde expresia $(hmax[a] - a - 1) =$ consumul pentru zborul orizontal de la poziția a până deasupra poziției $hmax[a]$, iar $(h[a] - h[hmax[a]]) =$ plusul de energie datorat coborârii pe verticală de deasupra poziției $hmax[a]$ (de la nivelul $h[a]$) până la nivelul $h[hmax[a]]$ al pozitiei $hmax[a]$.

Deci $cost[a] = cost_dist(a, hmax[a]) + cost[h[hmax[a]]]$, adică pentru a ajunge de pe poziția a pe poziția n , trecem prin poziția $hmax[a]$, iar $cost_dist[a, hmax[a]]$ este costul pentru a ajunge de pe poziția a pe poziția $hmax[a]$ fără "aterizări" intermedieare.

Observăm că pentru orice i cu $a < i < hmax[a]$, avem $h[i] < h[hmax[a]] \leq h[a]$ aşa că pentru a ajunge de pe poziția a pe poziția $hmax[a]$ facem un pas la dreapta și apoi coborâm până la înălțimea $h[hmax[a]]$ și apoi mergem numai către dreapta, iar $cost_dist[a, hmax[a]] = (hmax[a] - a - 1) - (h[a] - h[hmax[a]])$.

34.1.2 Rezolvare detaliată

Listing 34.1.1: gaina.java

```

1 import java.io.*;
2 class Gaina
3 {
4     static final int nmax=10000;
5     static int n,a,t;
6     static int[] h=new int[nmax+1];
7     static int[] hmax=new int[nmax+1];
8     static int[] cost=new int[nmax+1];
9
10    static void citire() throws IOException
11    {
12        int i;
13        StreamTokenizer st=new StreamTokenizer(
14            new BufferedReader(new FileReader("gaina.in")));
15
16        st.nextToken(); n=(int)st.nval;
17        for(i=1;i<=n;i++) {st.nextToken(); h[i]=(int)st.nval;}
18    }// citire()
19
20    static int actual_cost()
21    {
22        int ac;
23        if((hmax[a]==a+1)&&(h[hmax[a]]==h[a])) ac=cost[hmax[a]];
24        else
25        {
26            t=cost[hmax[a]]+(hmax[a]-a-1)-(h[a]-h[hmax[a]]);
27            if(hmax[a]==a+1) t=t+1;
28            if(t<=0) ac=1; else ac=t;
29        }
30        return ac;
31    }// actual_cost()
32
33    public static void main(String[] args) throws IOException
34    {
35        citire();
36
37        // caut max
38        hmax[n]=n;
39        for(a=n-1;a>=1;a--)
40            if(h[a+1]>=h[hmax[a+1]]) hmax[a]=a+1; else hmax[a]=hmax[a+1];
41
42        // cauta costuri
43        cost[n]=0;
44        for(a=n-1;a>=1;a--)
45            if(h[a]>=h[hmax[a]]) cost[a]=actual_cost();
46
47        PrintWriter out=new PrintWriter(
48            new BufferedWriter(new FileWriter("gaina.out")));
49        out.println(cost[1]);
50        out.close();
51    }// main(...)
52 } // class

```

34.1.3 Cod sursă

Listing 34.1.2: gaina.pas

```

1  {
2  Vom afla pentru fiecare m<=n costul (energia minima)
3  pentru a ajunge de pe pozitia m pe pozitia n. Costul are
4  sens pentru valorile m pentru care h_m >=h_1 , orice 1>m.
5
6  Asa ca mai intai aflam aceste valori ale lui m. Mai exact,
7  hmax[a] este cea mai mica pozitie (mai mare decat a) pentru
8  care se realizeaza maximul inaltimeilor pe segmentul [a+1,n].
9  hmax[n]=n prin definitie.
10
11 Daca h[a]>=h[ hmax[a] ] atunci a este un "m" bun.
12
13 *****
14
15 Pentru actualizarea costurilor avem
16
17   cost[a]= cost[ h[ hmax[a] ] ] + cost_dist[a,hmax[a]];
18
19   adica pentru a ajunge de pe pozitia a pe pozitia n trecem
20   prin hmax[a].
21
22   Iar cost_dist[a,hmax[a]] este costul de a ajunge de pe pozitia a pe
23   pozitia hmax[a] fara "aterizari" intermediare. Observam ca pentru
24   orice i cu a < i < hmax[a] avem h[i] < h[ hmax[a] ] <= h[a] asa ca
25   pentru a ajunge de pe pozitia a pe pozitia hmax[a] facem un pas la
26   dreapta si apoi coboram pana la inaltimea h[ hmax[a] ] si apoi mergem
27   numai catre dreapta.
28
29   Costul este:
30
31   cost_dist[a,hmax[a]] = ( hmax[a]-a-1 )-(h[a] - h[ hmax[a] ] )
32
33 }
34
35 const nmax=10000;
36 var f,g:text;
37   n,a,t:integer;
38   h,hmax,cost: array[1..nmax] of integer;
39
40 procedure citire;
41 var i:integer;
42 begin
43   assign(f,'gaina.in'); reset(f);
44   readln(f,n);
45   for i:=1 to n do read(f,h[i]);
46   close(f);
47 end;
48
49 function actual_cost:integer;
50 begin
51   if (hmax[a]=a+1) and (h[hmax[a]]=h[a])
52     then actual_cost:=cost[hmax[a]]
53   else
54     begin
55       t:=cost[hmax[a]]+(hmax[a]-a-1)-(h[a]-h[hmax[a]]);
56       if hmax[a]=a+1 then t:=t+1;
57       if t<=0 then actual_cost:=1
58         else actual_cost:=t;
59     end;
60 end;
61
62 begin
63   citire;
64   { caut max }
65   hmax[n]:=n;
66   for a:=n-1 downto 1 do
67     if h[a+1]>=h[hmax[a+1]]
68       then hmax[a]:=a+1
69       else hmax[a]:=hmax[a+1];
70   { cauta costuri }
71   cost[n]:=0;
72   for a:=n-1 downto 1 do
73     if h[a]>=h[hmax[a]]
74       then cost[a]:=actual_cost;
75

```

```

76     writeln(cost[1]);
77     assign(g,'gaina.out'); rewrite(g);
78     writeln(g,cost[1]);
79     close(g);
80 end.

```

34.2 Rez

Gigel este electronist amator. El afirmă că a inventat o nouă componentă electronică denumită reziston. În mod ciudat totuși rezistorii au niște proprietăți care nouă ne sună foarte cunoscute:

1. Orice rezistor este caracterizat printr-o mărime fizică numită rezistență. Aceasta poate avea ca valori numai numere naturale.
2. Rezistorii pot fi legați între ei în serie sau în paralel, formând astfel circuite.
3. Fie doi rezistori având rezistențele R_1 , respectiv R_2 . Legarea în serie a rezistorilor se realizează astfel:

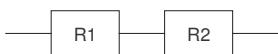


Figura 34.3: Rez1

Rezistența acestui circuit va fi $R_1 + R_2$.

4. Legarea celor doi rezistori în paralel se realizează astfel:

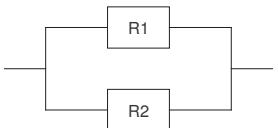


Figura 34.4: rez2

Rezistența acestui circuit va fi $(R_1 * R_2) / (R_1 + R_2)$. Fiindcă rezistențele pot fi numai numere naturale, împărțirea este întreagă (adică rezultatul este câtul împărțirii întregi a lui $(R_1 * R_2)$ la $(R_1 + R_2)$).

5. Prin legarea oricărora rezistori în serie și în paralel se obțin circuite. Circuitele pot fi legate în serie și/sau în paralel după aceleași reguli. Rezistența unui circuit se calculează aplicând regulile de mai sus.

Un circuit va fi codificat printr-un sir de caractere construit după următoarele reguli:

- a. Dacă circuitul C este format dintr-un singur rezistor și acesta are rezistență de valoare x , atunci codificarea circuitului C este Rx . Rezistența circuitului C va fi x .
- b. Dacă circuitul C este obținut prin legarea în serie a două sau mai multe circuite, codificate C_1, C_2, \dots, C_k , atunci codificarea circuitului C se obține concatenând în ordine codificările circuitelor $C_1 C_2 \dots C_k$. Rezistența circuitului C se obține prin însumarea rezistențelor circuitelor C_1, C_2, \dots, C_k .
- c. Dacă circuitul C este obținut prin legarea în paralel a două sau mai multe circuite, atunci codificarea circuitului C se obține încadrând între paranteze rotunde codificările circuitelor din care este format și separând aceste codificări prin virgulă: (C_1, C_2, \dots, C_k) , $k > 1$. Rezistența circuitului C este egală cu câtul împărțirii produsului dintre rezistențele C_1, C_2, \dots, C_k și suma rezistențelor circuitelor C_1, C_2, \dots, C_k .

Cerință

Scrieți un program care să determine rezistența unui circuit.

Date de intrare

Fisierul de intrare **rez.in** conține pe prima linie un sir de caractere care reprezintă codificarea unui circuit conform regulilor de mai sus.

Date de ieșire

Fisierul de ieșire **rez.out** conține o singură linie pe care este scrisă rezistența circuitului specificat în fisierul de intrare.

Restricții și precizări

$0 < \text{lungimea codificării unui circuit} \leq 1000$
 $0 < \text{rezistența oricărui rezistor} < 100$
 $0 < \text{rezistența oricărui circuit} < 2.000.000.000$
 Sirul prin care se codifică un circuit nu conține spații.
 Pentru datele de test nu se vor obține împărțiri la 0.

Exemple

rez.in	rez.out	rez.in	rez.out
R12	12	R42R33R3	78

rez.in	rez.out	rez.in	rez.out
R2(R5,R69,R12)R80	130	(R5,R3(R12,R4),R3)	6

Timp maxim de execuție/test: 0.1 sec Linux și 0.1 sec Windows.

34.2.1 Indicații de rezolvare

Vom citi codificarea circuitului într-un sir de caractere *s*.

Pentru a calcula rezistența circuitului, vom utiliza două *funcții recursive*:

– funcția `rezserie()` calculează rezistența unui circuit serie sau a unui circuit format dintr-un singur rezistor

– funcția `rezparalel()` calculează rezistența unui circuit paralel.

Funcția `rezserie()`:

Însumăm într-o variabilă (*suma*) rezistențele circuitelor legate în serie. Acestea pot fi circuite formate:

– dintr-un singur rezistor (dacă pe poziția curentă în sirul *s* se află litera *R*) ; în acest caz la variabila *suma* adunăm valoarea rezistorului.

– dintr-un circuit paralel (dacă pe poziția curentă în sirul *s* se află litera *paranteză deschisă*) ; în acest caz la variabila *suma* adunăm rezistența circuitului paralel (obținută printr-un apel al funcției `rezparalel()`).

Sfârșitul circuitului serie este determinat de apariția caracterului *virgulă* sau *paranteză închisă* sau de *sfârșitul sirului s*.

Funcția `rezparalel()`:

Calculăm suma și produsul circuitelor legate în paralel (acestea sunt circuite serie a căror rezistență se determină prin apelul funcției `rezserie()`). Evident, sfârșitul circuitului paralel este marcat de întâlnirea caracterului *paranteză închisă*.

Observați că implementarea folosește *recursivitatea indirectă*.

34.2.2 Rezolvare detaliată

Listing 34.2.1: rez.java

```

1 import java.io.*;
2 class Rez
3 {
4   static final int LMax=1001;
5   static int n, p=0;
6   static char[] s=new char[LMax];
7   static long r;
8
9   public static void main(String[] args) throws IOException
10  {
11    citire();
12    r=rezSerie();
13    afisare();
14  } // main...
15
16  static void citire() throws IOException
17  {
18    BufferedReader br=new BufferedReader(
19      new FileReader("rez.in"));
20    s=br.readLine().toCharArray();
21    n=s.length;
22  } // citire()
23
24  static void afisare() throws IOException

```

```

25      {
26          PrintWriter out=new PrintWriter(
27              new BufferedWriter(new FileWriter("rez.out")));
28          out.println(r);
29          out.close();
30      } // afisare()
31
32      static long rezSerie()
33      {
34          long val, suma=0;
35          if((p>=n) || (s[p]==',')) || (s[p]==')')) return 0;
36          while((p<n) && (s[p]!=',') && (s[p]!='')) )
37              if(s[p]=='R')
38              {
39                  p++;
40                  val=s[p]-'0';
41                  p++;
42                  if((s[p]>='0') && (s[p]<='9')) val=val*10+s[p++]-'0';
43                  suma+=val;
44              }
45          else suma+=rezParalel();
46          return suma;
47      } // rezSerie()
48
49      static long rezParalel()
50      {
51          long suma=0, produs=1, val;
52          while(s[p]!='')
53          {
54              p++;
55              val=rezSerie();
56              suma+=val;
57              produs*=val;
58          }
59          p++;
60          return produs/suma;
61      } // rezparalel()
62  } // class

```

34.2.3 Cod sursă

Listing 34.2.2: rez.cpp

```

1 #include <stdio.h>
2
3 #define InFile "rez.in"
4 #define OutFile "rez.out"
5 #define LMax 1001
6
7 int n, p=0;
8 char s[LMax];
9 long r;
10
11 void citire(void);
12 void afisare(void);
13 long rezserie(void);
14 long rezparalel(void);
15
16 int main()
17 {
18     citire();
19     r=rezserie();
20     afisare();
21     return 0;
22 }
23
24 void citire(void)
25 {
26     FILE *fin=fopen(InFile, "r");
27     fscanf(fin, "%s",s);
28     for (n=0; s[n]; n++);
29     fclose(fin);

```

```

30 }
31
32 void afisare(void)
33 {
34 FILE *fout=fopen(OutFile, "w");
35 fprintf(fout, "%ld\n", r);
36 fclose(fout);
37 }
38
39 long rezserie()
40 {
41 long val, suma=0;
42 if (p>=n || s[p]==',' || s[p]==')') return 0;
43 while (p<n && s[p]!=',' && s[p]!=')')
44     if (s[p]=='R')
45         {p++;
46          val=s[p]-'0';
47          p++;
48          if (s[p]>='0' && s[p]<='9')
49              {val=val*10+s[p]-'0';p++;}
50          suma+=val;
51        }
52     else suma+=rezparalel();
53 return suma;
54 }
55
56 long rezparalel()
57 {
58 long suma=0, produs=1, val;
59 while (s[p]!=')')
60     {p++;
61      val=rezserie();
62      suma+=val; produs*=val;
63    }
64 p++;
65 return produs/suma;
66 }

```

34.3 Sortări

Balaourului Arhirel nu îi plac prea mult şirurile care nu sunt ordonate. Din acest motiv, nu poate să suporte permutările de N elemente, aşa că se decide să le sorteze şi pentru asta inventează o metodă proprie de sortare.

El ia iniţial un şir S care reprezintă o permutare de ordin N . Caută în şirul S cel mai mic (*min*) şi cel mai mare element (*max*). Să considerăm că *min* se află în şirul S pe poziţia *pmin*, iar *max* pe poziţia *pmax*. Să notăm cu x minimul dintre *pmin* şi *pmax*, iar cu y maximul dintre *pmin* şi *pmax*. Şirul S a fost astfel partititionat în alte trei şiruri, S_1, S_2, S_3 care pot avea fiecare zero elemente, un element sau mai multe elemente. Şirul S_1 începe la prima poziţie din şir şi se termină la poziţia $x - 1$. Şirul S_2 începe la poziţia $x + 1$ şi se termină la poziţia $y - 1$. Şirul S_3 începe la poziţia $y + 1$ şi se termină la ultima poziţie din şir.

Balaourul Arhirel mută valoarea *min* la capătul din stânga al lui S , iar valoarea *max* la capătul din dreapta al şirului S şi reia sortarea pentru fiecare din şirurile S_1, S_2, S_3 .

De exemplu să considerăm $N = 6$ şi şirul $S = (342165)$. La primul pas, $\min = 1$ şi $\max = 6$. Deci $S_1 = (342)$; $S_2 = ()$; $S_3 = (5)$. Se mută *min* şi *max* la capetele şirului şi se obţine $S = (134256)$ şi se sortează în acelaşi mod S_1, S_2 şi S_3 . S_2 şi S_3 au 0, respectiv 1 element, deci sunt deja sortate. Pentru S_1 , se găseşte $\min = 2$ şi $\max = 4$ şi vom avea şirurile $(3); (); ()$. Se mută *min* şi *max* la capete şi se obţine $S_1 = (234)$. În final, vom avea şirul $S = (123456)$.

Evident, această metodă nu va funcţiona întotdeauna pentru sortarea unei permutări.

Spre exemplu, pentru şirul $S = (341652)$, se găseşte $\min = 1$ şi $\max = 6$, iar $S_1 = (34)$, $S_2 = ()$, $S_3 = (52)$. Se mută *min* şi *max* la capetele lui S : $S = (134526)$ şi se procedează la sortarea pe rând a şirurilor S_1, S_2, S_3 . S_1 este sortat, S_2 nu are elemente, iar S_3 va deveni $S_3 = (25)$. În final, $S = (134256)$.

Cerinţă

Ajuataj-l pe Balaourul Arhirel să afle câte dintre permutările de N elemente pot fi sortate prin metoda sa.

Date de intrare

Fișierul **sortari.in** conține o singură linie pe care se află numărul N .

Date de ieșire

Fișierul **sortari.out** va conține o singură linie pe care se află numărul de permutări de ordin N ce pot fi sortate prin metoda balaurului modulo 19573 (restul împărțirii numărului de permutări la 19573).

Restricții

$0 < N < 201$

Exemplu

sortari.in **sortari.out**

4 18

Explicație: În cazul permutărilor de câte 4 elemente, sirurile (1342); (3124); (3142); (3412); (3421); (4312) nu vor putea fi sortate crescător. Celelalte $24 - 6 = 18$ permutări pot fi sortate crescător.

De exemplu, pentru sirul (1342), după găsirea $\min = 1$, $\max = 4$, se obțin sirurile: $S1 = ()$; $S2 = (3)$; $S3 = (2)$, care, având câte un element sunt sortate. Astfel, după mutarea la capete a lui \min și \max vom avea: (1324).

Timp maxim de execuție/test: 0.2 sec Linux și 1.5 sec Windows.

34.3.1 Indicații de rezolvare

Soluția oficială

Soluția este $O(N^3)$ și se bazează pe *programare dinamică*.

Construim vectorul a cu semnificația $a[i] =$ numărul de siruri de lungime i ce pot fi sortate folosind *metoda balaurului*.

Astfel $a[0] = 1$, $a[1] = 1$, $a[2] = 2$, $a[3] = 6$, $a[4] = 18$.

Să considerăm că am calculat deja $a[0], \dots, a[i-1]$. Pentru a calcula $a[i]$ fixăm poziția minimului și a maximului (2 for-uri) și înmulțim $a[\text{lungimea}(S1)]$ cu $a[\text{lungimea}(S2)]$ și cu $a[\text{lungimea}(S3)]$ și adunăm produsul la $a[i]$.

Facem această înmulțire pentru că toate numerele din $S1 <$ toate numerele din $S2 <$ toate numerele din $S3$. Astfel noi știm ce numere trebuie să conțină sirurile $S1$, $S2$, $S3$. Trebuie doar ca aceste siruri să fie sortabile cu metoda lui Arhirel și de fapt ne interesează doar numărul lor și nu sirurile efective.

Evident nu trebuie uitat că după fiecare operație să păstrăm doar restul împărțirii la acel număr.

34.3.2 Rezolvare detaliată

Listing 34.3.1: sortari.java

```

1 import java.io.*;
2 class Sortari
3 {
4     static final int nr=19573;
5
6     public static void main(String[] args) throws IOException
7     {
8         int n;
9         int[] a=new int[201];
10        int i,j,k;
11        int l,s;
12
13        a[0]=1;
14        a[1]=1;
15        a[2]=2;
16        a[3]=6;
17
18        StreamTokenizer st=new StreamTokenizer(
19            new BufferedReader(new FileReader("9-sortari.in")));
20
21        st.nextToken(); n=(int)st.nval;
22
23        for(i=4;i<=n;i++)
24        {

```

```

25      s=0;
26      l=1;
27      for(j=1;j<=i-1;j++)
28          for(k=j+1;k<=i;k++)
29          {
30              l=((a[j-1]*a[k-1-j])%nr)*a[i-k];
31              l=l%nr;
32              s=s+l;
33              s=s%nr;
34          }
35      a[i]=(2*s)%nr;
36  } // for i
37
38  PrintWriter out=new PrintWriter(
39      new BufferedWriter(new FileWriter("sortari.out")));
40
41  out.println(a[n]);
42  out.close();
43 } // main
44 } // class

```

34.3.3 Cod sursă

Listing 34.3.2: sortari.pas

```

1 program sortari;
2 const nr=19573;
3 var n:longint;
4 a:array[0..200] of longint;
5 i,j,k:longint;f:text;
6 l,s:longint;
7
8 procedure initializari;
9 begin
10 a[0]:=1;
11 a[1]:=1;
12 a[2]:=2;
13 a[3]:=6;
14 end;
15
16 begin
17     initializari;
18     assign(f, 'sortari.in');
19     reset(f);
20     read(f,n);
21     for i:=4 to n do
22         begin
23             s:=0;
24             l:=1;
25             for j:=1 to i-1 do
26                 for k:=j+1 to i do
27                     begin
28                         l:=(a[j-1]*a[k-1-j]) mod nr*a[i-k];
29                         l:=l mod nr;
30                         s:=s+l;
31                         s:=s mod nr;
32                     end;
33             a[i]:=(2*s) mod nr;
34         end;
35     close(f);
36     assign(f, 'sortari.out');
37     rewrite(f);
38     writeln(f,a[n]);
39     close(f);
40 end.

```

34.4 Cuvinte

Balaourul Arhirel se decide să învețe biologie, aşa că doreşte să cumpere manualul de clasa a X-a. Din păcate, acesta nu se mai găseşte pe piaţă, dar Balaourul reuşeşte să găsească o copie la un prieten. După ce începe să citească, Balaourul observă că există greşeli în copia prietenului, iar într-un impuls de energie, se hotărăşte să corecteze manualul. El are la dispoziţie un dicţionar de M cuvinte dintre care trebuie să extragă variante pentru cuvântul greşit. Asupra cuvântului greşit balaourul poate să facă următoarele schimbări în aşa fel încât acesta să ajungă la o variantă din dicţionar:

- poate şterge o literă;
- poate insera o literă;
- poate schimba o literă în altă literă.

Totuşi, Balaourul Arhirel este lenuş, aşa că nu doreşte să opereze mai mult de K schimbări în cuvântul greşit pentru a-l aduce la o formă corectă (existentă în dicţionar).

Cerinţă

Ajutaţi-l pe Balaourul Arhirel să afle care dintre cuvintele din dicţionar ar putea fi variante ale cuvântului greşit.

Date de intrare

Fişierul de intrare **cuvinte.in** conţine pe prima linie cele două numere M şi K , separate printr-un spaţiu, reprezentând numărul de cuvinte din dicţionar şi numărul maxim de modificări ce pot fi efectuate asupra cuvântului ce trebuie corectat. Pe a doua linie se găsesc separate printr-un spaţiu lungimea cuvântului greşit, L_{cuvant} , şi cuvântul greşit. Pe următoarele M linii se găsesc cuvintele din dicţionar, câte un cuvânt pe o linie în forma următoare: pe linia i lungimea L_{i-2} a cuvântului $i - 2$, separată printr-un singur spaţiu de cuvântul $i - 2$.

Date de ieşire

Fişierul de ieşire **cuvinte.out** va conţine M linii. Pe linia i se află valoarea 1 pentru cazul în care cuvântul i din dicţionar este o variantă pentru cuvântul greşit dat, respectiv valoarea 0 în caz contrar.

Restricţii

$$0 < M < 21$$

$$0 < K < 31$$

$$0 < \text{lungimea oricărui cuvânt (inclusiv cel greşit)} < 10001$$

Cuvintele sunt formate doar din literele alfabetului latin, iar literele mici diferă de cele mari (de exemplu, Z nu este acelaşi lucru cu z).

Exemplu

cuvinte.in	cuvinte.out
6 2	0
6 radiux	1
5 ladin	0
6 Radius	1
6 ridica	0
5 radio	1
6 adipos	
5 cadiu	

Timp maxim de execuţie/test: 0.3 sec pentru Linux şi 2 sec pentru Windows.

34.4.1 Indicaţii de rezolvare

Soluţia oficială

Cu toţii cunoaştem dinamica clasice a distanţei de editare (dacă nu, vezi Cormen). Problema e că rezolvarea clasica este $O(N^2)$ (de fapt $N * M$, dar N şi M sunt foarte apropiate ca valori). Noi trebuie să rezolvăm însă în $O(K * N)$.

Pentru aceasta observăm că din toată acea matrice nu ne trebuie decât K elemente în jurul diagonalei principale.

Orice element aflat în afara acestei benzi poate fi considerat infinit, pentru că ajungerea în colt necesită cel puțin K incrementari, deci rezultatul final va fi mai mare decât K .

34.4.2 Rezolvare detaliată

Listing 34.4.1: cuvinte.java

```

1 import java.io.*;
2 class Cuvinte
3 {
4     static final int NMAX=10000, KMAX=30, oo=667;
5     static int[][] A=new int[2][NMAX+KMAX];
6     static int M, K, L1, L2;
7     static char[] s1=new char[NMAX+KMAX+1];
8     static char[] s2=new char[NMAX+KMAX+1];
9
10    public static void main(String[] args) throws IOException
11    {
12        int t,i,j,v=0,n=1,pl,total=0;
13        char[] s;
14
15        StreamTokenizer st=new StreamTokenizer(
16            new BufferedReader(new FileReader("cuvinte.in")));
17        PrintWriter out=new PrintWriter(
18            new BufferedWriter(new FileWriter("cuvinte.out")));
19
20        st.nextToken(); M=(int)st.nval;
21        st.nextToken(); K=(int)st.nval;
22
23        st.nextToken(); L1=(int)st.nval;
24
25        st.nextToken(); s=st.sval.toCharArray();
26        for(i=0;i<s.length;i++) s1[i+1]=s[i];
27
28        for(t=0;t<M;t++)
29        {
30            st.nextToken(); L2=(int)st.nval;
31
32            st.nextToken(); s=st.sval.toCharArray();
33            for(i=0;i<s.length;i++) s2[i+1]=s[i];
34
35            for(i=0;i<=K;i++) A[v][i] = i;
36
37            A[v][K+1] = oo;
38
39            for(i=1;i<=L2;i++)
40            {
41                pl=Math.min(i+K+2, L1+1);
42                if(i-K-2>=0) A[n][i-K-2]=oo; else A[n][0] = i;
43
44                for(j=Math.max(1,i-K-1);j<pl;j++)
45                    if(s1[j]==s2[i]) A[n][j]=A[v][j-1];
46                    else
47                    {
48                        A[n][j]=Math.min(A[v][j],A[v][j-1]);
49                        A[n][j]=Math.min(A[n][j],A[n][j-1])+1;
50                    }
51
52                A[n][pl]=oo;
53                v^=1;
54                n^=1;
55            }// for i
56
57            out.println((A[v][L1]<=K)?1:0);
58        }// for t
59
60        out.close();
61    }// main
62 } // class

```

34.4.3 Cod sursă

Listing 34.4.2: cuvinte.c

```

1 #include <stdio.h>
2 #include <string.h>
3

```

```

4 #define NMAX 10000
5 #define KMAX 30
6
7 #define min(a,b) (((a) < (b)) ? a : b)
8 #define max(a,b) (((a) > (b)) ? a : b)
9 #define INF 667
10
11 int A[2][NMAX+KMAX];
12 int M, K, L1, L2;
13 char s1[NMAX+KMAX+1], s2[NMAX+KMAX+1];
14
15 int main()
16 {
17     int t, i, j, v = 0, n = 1, pl, total = 0;
18
19     freopen("cuvinte.in", "r", stdin);
20     scanf("%d %d", &M, &K);
21     scanf("%d %s", &L1, s1+1);
22
23     freopen("cuvinte.out", "w", stdout);
24     for (t = 0; t < M; t++)
25     {
26         scanf("%d %s", &L2, s2+1);
27         for (i = 0; i <= K; i++)
28             A[v][i] = i;
29         A[v][K+1] = INF;
30
31         for (i = 1; i <= L2; i++)
32         {
33             pl = min(i+K+2, L1+1);
34
35             if (i-K-2 >= 0)
36                 A[n][i-K-2] = INF;
37             else
38                 A[n][0] = i;
39
40             for (j = max(1, i-K-1); j < pl; j++)
41                 if (s1[j] == s2[i])
42                     A[n][j] = A[v][j-1];
43                 else
44                 {
45                     A[n][j] = min(A[v][j], A[v][j-1]);
46                     A[n][j] = min(A[n][j], A[n][j-1]) + 1;
47                 }
48             A[n][pl] = INF;
49
50             v ^= 1; n ^= 1;
51         }
52
53         printf("%d\n", (A[v][L1]<=K));
54     }
55
56     return 0;
57 }

```

34.5 Puncte

Considerăm că toate punctele de coordonate întregi din plan sunt colorate în negru, cu excepția a n puncte care sunt colorate în roșu. Două puncte roșii aflate pe aceeași linie orizontală sau pe aceeași linie verticală (adică puncte care au aceeași ordonată sau aceeași abscisă) pot fi unite printr-un segment. Colorăm în roșu toate punctele de coordonate întregi de pe acest segment. Repetăm operația cât timp se obțin puncte roșii noi.

Cerință

Cunoscând coordonatele celor n puncte care erau inițial roșii, aflați numărul maxim de puncte roșii care vor exista în final.

Date de intrare

Pe prima linie a fișierului de intrare **puncte.in** este scris numărul n . Pe următoarele n linii sunt date coordonatele punctelor, separate printr-un singur spațiu.

Date de ieșire

Fisierul de ieșire **puncte.out** va conține o singură linie pe care se află numărul maxim de puncte roșii existente în final.

Restricții

$0 \leq n \leq 100.000$

coordonatele sunt numere întregi din intervalul $[0, 1000]$

Exemplu

puncte.in puncte.out

4 12

0 2

3 1

1 4

4 4

Explicație

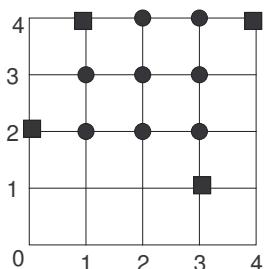


Figura 34.5: Poarta

Timp maxim de execuție/test: 0.1 sec Linux și 0.1 sec Windows.

34.5.1 Indicații de rezolvare

Soluția oficială

Este evident că un punct roșu situat între alte două puncte roșii (pe aceeași orizontală sau verticală) nu contribuie cu nimic la apariția de noi puncte roșii. De aceea e suficient să memorăm pentru fiecare din cele M linii orizontale coordonatele celui mai din stanga și a celui mai din dreapta punct roșu. Analog pentru cele M linii verticale (am notat cu M dimensiunea maximă a pătratului în care se găsesc toate punctele care ne interesează, în enunt $M = 1001$).

Folosind vectorii care dau limita inferioară și superioară a segmentelor facem o *baleiere* repetată a zonei de $M \times M$ puncte mai întâi pe orizontală, apoi pe verticală, modificând limitele respective atât timp cât este posibil. Se poate arăta că numarul de baleieri nu depășește $4M$.

Rezultă o complexitate $O(M * M)$.

34.5.2 Rezolvare detaliată

Listing 34.5.1: puncte.java

```

1 import java.io.*;
2 class Puncte
3 {
4     static final int M=1001;
5
6     public static void main(String[] args) throws IOException
7     {
8         int n,i,j,k,x0,y0,x1,y1;
9
10        int[] xs=new int[M];
11        int[] ys=new int[M];
12        int[] xd=new int[M];
13        int[] yd=new int[M];
14
15        int np=0;
16        StreamTokenizer st=new StreamTokenizer(

```

```

17         new BufferedReader(new FileReader("9-puncte.in")));
18     PrintWriter out=new PrintWriter(
19         new BufferedWriter(new FileWriter("puncte.out")));
20
21     for(i=0;i<M;i++) {xs[i]=ys[i]=M+1; xd[i]=yd[i]=-1;}
22
23     st.nextToken(); n=(int)st.nval;
24
25     for(i=0;i<n;i++)
26     {
27         st.nextToken(); j=(int)st.nval;
28         st.nextToken(); k=(int)st.nval;
29
30         if(j<xs[k]) xs[k]=j;
31         if(j>xd[k]) xd[k]=j;
32         if(k<ys[j]) ys[j]=k;
33         if(k>yd[j]) yd[j]=k;
34     }
35
36     i=0; while(xd[i]<0) i++;
37     y0=i;
38     i=M-1; while(xd[i]<0) i--;
39     y1=i;
40     i=0; while(yd[i]<0) i++;
41     x0=i;
42     i=M-1; while(yd[i]<0) i--;
43     x1=i;
44
45     /* x0=0; y0=0; x1=M-1; y1=M-1; */
46     do
47     {
48         k=0;
49         for(i=x0;i<=x1;i++)
50             for(j=ys[i];j<=yd[i];j++)
51             {
52                 if(i<xs[j]) {xs[j]=i; k=1;}
53                 if(i>xd[j]) {xd[j]=i; k=1;}
54             }
55
56         for(j=y0;j<=y1;j++)
57             for(i=xs[j];i<=xd[j];i++)
58             {
59                 if(j<ys[i]) {ys[i]=j; k=1;}
60                 if(j>yd[i]) {yd[i]=j; k=1;}
61             }
62     } while(k>0);
63
64     for(j=y0;j<=y1;j++)
65         if(xd[j]>=0) np+=(xd[j]-xs[j]+1);
66
67     out.println(np);
68     out.close();
69 } // main
70 } // class

```

34.5.3 Cod sursă

Listing 34.5.2: puncte.c

```

1 #include <stdio.h>
2 #define M 1001
3
4 int main()
5 {
6     long int n,i,j,k,xs[M],ys[M],xd[M],yd[M],x0,y0,x1,y1;
7     long np=0;
8
9     FILE *fi,*fo;
10
11    for(i=0;i<M;i++)
12    {
13        xs[i]=ys[i]=M+1;

```

```

14         xd[i]=yd[i]=-1;
15     }
16
17     fi=fopen("puncte.in", "rt");
18     fscanf(fi, "%d", &n);
19     for(i=0; i<n; i++)
20     {
21         fscanf(fi, "%d %d", &j, &k);
22         if(j<xs[k]) xs[k]=j;
23         if(j>xd[k]) xd[k]=j;
24         if(k<ys[j]) ys[j]=k;
25         if(k>yd[j]) yd[j]=k;
26     }
27
28     i=0;
29     while(xd[i]<0) i++;
30     y0=i;
31     i=M-1;
32     while(xd[i]<0) i--;
33     y1=i;
34     i=0;
35     while(yd[i]<0) i++;
36     x0=i;
37     i=M-1;
38     while(yd[i]<0) i--;
39     x1=i;
40
41 /* x0=0; y0=0; x1=M-1; y1=M-1; */
42 fclose(fi);
43
44 do
45 {
46     k=0;
47     for(i=x0; i<=x1; i++)
48         for(j=ys[i]; j<=yd[i]; j++)
49         {
50             if(i<xs[j])
51             {
52                 xs[j]=i;
53                 k=1;
54             }
55
56             if(i>xd[j])
57             {
58                 xd[j]=i;
59                 k=1;
60             }
61         }
62
63     for(j=y0; j<=y1; j++)
64         for(i=xs[j]; i<=xd[j]; i++)
65         {
66             if(j<ys[i])
67             {
68                 ys[i]=j;
69                 k=1;
70             }
71
72             if(j>yd[i])
73             {
74                 yd[i]=j;
75                 k=1;
76             }
77         }
78     while(k);
79
80     for(j=y0; j<=y1; j++)
81         if(xd[j]>=0)
82             np+=(xd[j]-xs[j]+1);
83
84     fo=fopen("puncte.out", "wt");
85     fprintf(fo, "%ld\n", np);
86
87     fclose(fo);
88     return 0;
89 }
```

34.6 Materom

Liceul Național Anonim (LNA) este invitat să participe la olimpiada de matematică-română cu o echipă formată din m elevi. La această olimpiadă elevii lucrează în echipă și trebuie să rezolve două subiecte: unul de română și altul de matematică.

Au fost testați și punctați la cele două materii n elevi, numerotați de la 1 la n . Așa cum era de așteptat, în general, elevii buni la matematică s-au dovedit cam slăbuți la română și viceversa.

Pentru a maximiza şansele de câștig ale echipei LNA, directorul a decis să trimită m elevi dintre cei n elevi testați, astfel încât diferența în modul dintre suma punctajelor de la limba română ale elevilor din echipă și suma punctajelor la matematică ale elevilor din echipă să fie minimă. Dacă există mai multe echipe de elevi care îndeplinesc condiția precedentă, va fi selectată dintre acestea o echipă pentru care suma tuturor notelor să fie maximă.

Cerință

Scrieți un program care să determine în conformitate cu decizia directorului, diferența în modul dintre suma punctajelor de la limba română ale elevilor din echipa LNA și suma punctajelor la matematică ale elevilor din echipă, precum și suma tuturor punctajelor elevilor din echipa LNA.

Date de intrare

În fișierul de intrare **materom.in** se află pe prima linie numerele naturale n și m separate printr-un spațiu, având semnificația din enunț.

Pe fiecare dintre următoarele n linii se află două numere naturale separate printr-un spațiu. Mai exact linia i din fișier ($i = 2, \dots, n + 1$) conține m_i r_i , unde m_i este punctajul obținut la matematică, iar r_i este punctajul obținut la limba română de elevul $i - 1$.

Date de ieșire

Fișierul de ieșire **materom.out** conține două linii. Pe prima linie se va afișa diferența (în modul) dintre suma punctajelor de la limba română ale elevilor din echipă și suma punctajelor la matematică ale elevilor din echipă. Pe cea de a doua linie se va afișa suma punctajelor elevilor selectați în echipa LNA.

Restricții

$$1 \leq m < 20$$

$$1 \leq n \leq 500$$

$$m \leq n$$

$$0 \leq m_i, r_i \leq 20$$

Exemplu

materom.in **materom.out**

4 2	2
2 3	10
1 2	
6 2	
4 1	

Explicație: Dintre cei 4 elevi trebuie să selectăm 2. Avem 6 posibilități, dintre care 3 au diferența (în modul) dintre suma notelor la matematică și suma notelor la română 2. Acestea sunt:

- (1, 2) pentru care suma punctajelor este 8
- (1, 4) pentru care suma punctajelor este 10
- (2, 4) pentru care suma punctajelor este 8.

Alegem combinația 1 4 deoarece are suma maximă.

Timp maxim de execuție/test: 0.1 sec Linux și 0.2 sec Windows.

34.6.1 Indicații de rezolvare

Soluția oficială

Pentru început rezolvăm problema pentru un elev, apoi pentru 2, 3, ..., m și pentru fiecare diferență de punctaj, între matematică și română, calculez suma punctajelor maximă ce se poate realiza prin *programare dinamică*, după care pentru diferență minimă de punctaj determin suma maximă.

Pentru a realiza ceea ce e descris mai sus se folosesc două matrice (cu m linii și 2^m coloane – maxima_grupa^* note) unde linia reprezintă a câtă persoană din grup este, iar indicele coloanei este calculat după formula $m * \text{nota_maximă_admisă} + \text{diferența dintre suma notelor la matematică și suma notelor la română}$:

- Matricea S este folosită pentru a memora sumele maxime pentru o anumită diferență. $S[i,j]$ memorează pentru al i -lea membru al grupei care a generat diferența j suma notelor celor i concurenți.

- Matricea L este folosită pentru a memora elevii din grupă. $L[i,j]$ memorează pentru cel de-al i -lea membru ce a generat diferența j numărul de ordine al concurrentului de pe poziția i .

34.6.2 Rezolvare detaliată

Listing 34.6.1: materom.java

```

1 import java.io.*;
2 class Materom
3 {
4     static final int nmax=200, nota=21, grup=19;
5     static int[] mat=new int[nmax+1];
6     static int[] rom=new int[nmax+1];
7     static int[][] l=new int[grup+1][2*nota*grup+1+1];
8     static int[][] s=new int[grup+1][2*nota*grup+1+1];
9
10    static int[] sol=new int[21]; // daca vreau sa o afisez !!!
11    static int n,m,suma,diferenta,sm,sr;
12
13    public static void main(String[] args) throws IOException
14    {
15        citire();
16        rezolva();
17        scrie();
18    }// main
19
20    static void citire() throws IOException
21    {
22        int i;
23        StreamTokenizer st=new StreamTokenizer(
24            new BufferedReader(new FileReader("9-materom.in")));
25
26        st.nextToken(); n=(int)st.nval;
27        st.nextToken(); m=(int)st.nval;
28
29        for(i=0;i<=n-1;i++)
30        {
31            st.nextToken(); mat[i]=(int)st.nval;
32            st.nextToken(); rom[i]=(int)st.nval;
33        }
34    }// citire()
35
36    static void rezolva()
37    {
38        int i,j,k,p,p2,v;
39
40        for(i=0;i<=m;i++)
41            for(j=0;j<=2*nota*m+1;j++) l[i][j]=-1;
42
43        // s=l;// rezolva pentru un elev      !!!!!!!!
44        for(i=0;i<=m;i++)
45            for(j=0;j<=2*nota*m+1;j++) s[i][j]=l[i][j];
46
47        for(i=0;i<=n-1;i++)
48            if(mat[i]+rom[i]>s[0][m*nota+mat[i]-rom[i]])
49            {
50                l[0][m*nota+mat[i]-rom[i]]=i;
51                s[0][m*nota+mat[i]-rom[i]]=mat[i]+rom[i];
52            }
53
54        // restul
55        for(j=0;j<=m-2;j++)
56            for(k=0;k<=2*nota*m-1;k++)
57                if(l[j][k]>=0)
58                    for(i=0;i<=n-1;i++)
59                        if(s[j+1][k+mat[i]-rom[i]]<s[j][k]+mat[i]+rom[i])
60                        {
61                            p2=k;p=j; // cautam daca l-am mai folosit
62                            while((p>=0)&&(l[p][p2]!=i))

```

```

63             {
64                 p2=p2-(mat[l[p][p2]]-rom[l[p][p2]]);
65                 p=p-1;
66             }
67
68         // daca nu il folosim
69         if(p<0)
70         {
71             l[j+1][k+mat[i]-rom[i]]=i;
72             s[j+1][k+mat[i]-rom[i]]=s[j][k]+mat[i]+rom[i];
73         }
74     }
75
76     // determinare diferența minima
77     v=nota*m+1;
78     for(i=0;i<=nota*m;i++)
79     {
80         if((s[m-1][nota*m+i]>=0) || (s[m-1][nota*m-i]>=0))
81         {
82             if(s[m-1][nota*m+i]>s[m-1][nota*m-i]) v=nota*m+i;
83             else v=nota*m-i;
84             break;
85         }
86
87     // determina solutia
88     sm=0;
89     sr=0;
90     for(j=m-1;j>=0;j--)
91     {
92         sol[j]=l[j][v]+1;
93
94         sm=sm+mat[l[j][v]];
95         sr=sr+rom[l[j][v]];
96         v=v-(mat[l[j][v]]-rom[l[j][v]]));
97     }
98 } // rezolva
99
100 static void scrie() throws IOException
101 {
102     int i;
103     PrintWriter out=new PrintWriter(
104         new BufferedWriter(new FileWriter("materom.out")));
105
106     //calculeaza suma
107     suma=sr+sm;
108
109     //calculeaza diferența
110     diferența=Math.abs(sr-sm);
111
112     out.println(diferența);
113     out.println(suma);
114     out.close();
115 } // scrie()
116 } // class

```

34.6.3 Cod sursă

Listing 34.6.2: materom.pas

```

1 program cna;
2 const nmax=200;
3     nota=21;
4     grup=19;
5 var mat,rom:array[0..nmax] of integer;
6     l,s:array[0..grup,0..2*nota*grup+1] of integer;
7     sol:set of byte;{solutia}
8 n,m,suma,diferența,sm,sr:integer;
9
10 procedure citire;
11 var f:text;i:byte;
12 begin
13     assign(f,'materom.in');reset(f);

```

```

14  readln(f,n,m);
15  for i:=0 to n-1 do readln(f,mat[i],rom[i]);
16  close(f);
17 end;
18
19 procedure rezolva;
20 var i,j,k,p,p2,v:integer;
21 begin
22 for i:=0 to m do
23   for j:=0 to 2*nota*m+1 do l[i,j]:=-1;
24 s:=1;{rez. pt. un elev}
25 for i:=0 to n-1 do
26   if mat[i]+rom[i]>s[0,m*nota+mat[i]-rom[i]] then
27     begin
28       l[0,m*nota+mat[i]-rom[i]]:=i;
29       s[0,m*nota+mat[i]-rom[i]]:=mat[i]+rom[i];
30     end;
31 {restul}
32 for j:=0 to m-2 do
33   for k:=0 to 2*nota*m-1 do
34     if l[j,k]>=0 then
35       for i:=0 to n-1 do
36         if s[j+1,k+mat[i]-rom[i]]<s[j,k]+mat[i]+rom[i] then
37           begin
38             p2:=k;p:=j;{cautam daca l-am mai folosit}
39             while (p>=0) and (l[p,p2]<>i) do
40               begin
41                 p2:=p2-(mat[l[p,p2]]-rom[l[p,p2]]);
42                 p:=p-1;
43               end;
44             {daca nu il folosim}
45             if p<0 then
46               begin
47                 l[j+1,k+mat[i]-rom[i]]:=i;
48                 s[j+1,k+mat[i]-rom[i]]:=s[j,k]+mat[i]+rom[i];
49               end;
50             end;
51 {determinare diferenta minima}
52 v:=nota*m+1;
53 for i:=0 to nota*m do
54   if (s[m-1,nota*m+i]>=0) or (s[m-1,nota*m-i]>=0) then
55     begin
56       if s[m-1,nota*m+i]>s[m-1,nota*m-i] then v:=nota*m+i
57       else v:=nota*m-i;
58       break;
59     end;
60 {det solutie}
61 sol:=[];sm:=0;sr:=0;
62 for j:=m-1 downto 0 do
63   begin
64     sol:=sol+[l[j,v]+1];
65     sm:=sm+mat[l[j,v]];
66     sr:=sr+rom[l[j,v]];
67     v:=v-(mat[l[j,v]]-rom[l[j,v]]);
68   end;
69 end;
70
71 procedure scrie;
72 var i:byte;f:text;
73 begin
74 assign(f,'materom.out');
75 rewrite(f);
76 {calculeaza diferenta}
77 {calculeaza suma}
78 suma:=sr+sm;
79 diferența:=abs(sr-sm);
80 writeln(f, diferența);
81 writeln(f, suma);
82 close(f);
83 end;
84
85 BEGIN
86 citire;
87 rezolva;

```

90 scrie;
91 END.

Capitolul 35

ONI 2003



Figura 35.1: Sigla ONI 2003

35.1 Asediu

Se aproximează o zonă de război sub forma unui cerc pe circumferința căruia se stabilesc N puncte reprezentând comandamentele trupelor aliate cu proprietatea că nu există trei corzi cu capetele în aceste N puncte care să fie concurente într-un punct situat în interiorul cercului.

Între oricare două puncte (comandamente) există un drum sigur de acces direct. Aceste drumi împreună cu circumferința cercului delimită un număr de regiuni distincte.

Există informații că regiunile astfel delimitate reprezintă de fapt terenuri minate de combatanții inamici. Fiecare astfel de regiune va fi cercetată amănușit de căte un soldat aliat echipat corespunzător cu detectoare de mine.

Cerință

Indicați numărul de soldați de care este nevoie pentru cercetarea tuturor regiunilor formate.

Datele de intrare

Fișierul de intrare **asediu.in** conține:

- Pe prima linie N numărul de comandamente.

Datele de ieșire

Ieșirea se va face în fișierul **asediu.out** în formatul următor:

- Pe prima linie se află numărul T de soldați necesari pentru cercetarea tuturor regiunilor formate.

Restricții și precizări

- $2 \leq N \leq 2.000.000$

Exemplu

asediu.in	asediu.out
5	16

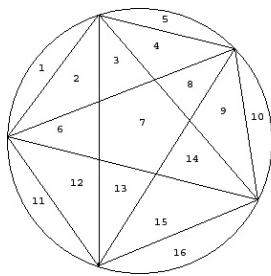


Figura 35.2: Asediu

Timp maxim de executare: 1 secundă/test

35.1.1 Indicații de rezolvare

Roxana Tîmplaru

Problema este de *combinatorică*.

Dacă se consideră poligonul format din cele n puncte, atunci fiecare latură formează cu circumferința cercului n regiuni plus o regiune interioară poligonului.

Fiecare diagonală trasată pe rând, presupune obținerea unui număr de regiuni egal cu unu plus numărul de intersecții cu diagonalele duse deja. Se ține cont de faptul că nu există trei corzi care să se intersecteze în interiorul cercului.

Deci, numărul total de regiuni este $n + 1 + \frac{n(n-3)}{2} + C_n^4 = 1 + C_n^2 + C_n^4$.

Mihai Stroe, GInfo nr. 13/6 - octombrie 2003

Acestă problemă se rezolvă matematic. Considerând poligonul format din cele n puncte fără a trasa diagonalele, fiecare latură formează cu circumferința cercului n regiuni, plus o regiune interioară poligonului.

Trasând pe rând fiecare diagonală, se adaugă un număr de regiuni egal cu unu plus numărul de intersecții cu diagonalele duse deja. Astfel, fiecare intersecție adaugă o regiune la numărul total de regiuni.

Considerând fiecare grup de 4 puncte, se observă că acesta determină exact o intersecție. În total sunt C_n^4 grupuri de câte 4 puncte.

Numărând și diagonalele, se ajunge la:

$$\text{numărul total de regiuni} = n + 1 + \frac{n(n-3)}{2} + C_n^4 = 1 + C_n^2 + C_n^4.$$

În implementare se ține cont de faptul căse ajunge la numere mari.

Dacă se folosește formula finală, se poate scăpa de implementarea împărțirilor pe numere mari, simplificând fractiile rezultate înainte de a efectua produsele.

De exemplu: $(18 \cdot 17 \cdot 16 \cdot 15) / (2 \cdot 3 \cdot 4 \cdot 5)$ devine $3 \cdot 17 \cdot 4 \cdot 15$ etc.

Deoarece formula finală este un polinom de gradul 4 în funcție de N calcularea rezultatului necesită un număr constant de operații, deci ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(1)$.

35.1.2 Rezolvare detaliată

Listing 35.1.1: asediul.java

```

1 import java.io.*; // rezultat=1+comb(n, 2)+comb(n, 4)
2 class Asediu
3 {
4     public static void main (String[] args) throws IOException
5     {
6         int n,i;
7         StreamTokenizer st=new StreamTokenizer(
8             new BufferedReader(new FileReader("asediu.in")));
9         PrintWriter out=new PrintWriter(new BufferedWriter(
10            new FileWriter("asediu.out")));
11         st.nextToken(); n=(int)st.nval;
12         int[] sol=nr2v(1);
13     }

```

```

14     sol=suma(sol,comb(n,2));
15     sol=suma(sol,comb(n,4));
16     for(i=sol.length-1;i>=0;i--) out.print(sol[i]);
17     out.close();
18 } // main(...)

19
20 static int[] suma(int[] x, int[] y)
{
21     int nx=x.length,ny=y.length,i,t;
22     int nz;
23     if(nx>ny) nz=nx+1; else nz=ny+1;
24     int[] z=new int[nz];
25
26     t=0;
27     for(i=0;i<nz;i++)
28     {
29         z[i]=t;
30         if(i<nx) z[i]+=x[i];
31         if(i<ny) z[i]+=y[i];
32
33         t=z[i]/10;
34         z[i]=z[i]%10;
35     }
36
37     if(z[nz-1]!=0) return z;
38     else
39     {
40         int[] zz=new int[nz-1];
41         for(i=0;i<nz-1;i++) zz[i]=z[i];
42         return zz;
43     }
44 }
45 } //suma

46
47 static int[] comb(int n,int k)
48 {
49     if(k>n/2) k=n-k; //optim !
50     int[] rez=nr2v(1);
51     int[] x=new int[k+1];
52     int[] y=new int[k+1];
53     int i,j,d;
54     for(i=1;i<=k;i++) x[i]=n-k+i;
55     for(j=1;j<=k;j++) y[j]=j;
56     for(j=2;j<=k;j++)
57     {
58         for (i=1;i<=k;i++)
59         {
60             d=cmmdc(y[j],x[i]);
61             y[j]=y[j]/d;
62             x[i]=x[i]/d;
63             if(y[j]==1) break;
64         }
65     }
66     for(i=1;i<=k;i++) rez=inm(rez,nr2v(x[i]));
67     return rez;
68 } //comb

69
70 static int cmmdc(int a,int b)
71 {
72     int d,i,r;
73     if(a>b) { d=a; i=b; } else { d=b; i=a; }
74     while(i!=0){ r=d%i; d=i; i=r; }
75     return d;
76 } //cmmdc

77
78 static int[] nr2v(int nr)
79 {
80     int nrr=nr,nc=0,i;
81     while(nr!=0) { nc++; nr=nr/10; }
82     int[] nrw=new int[nc];
83     nc=0;
84     nr=nrr;
85     while(nr!=0) { nrw[nc]=nr%10; nr=nr/10; nc++; }
86     return nrw;
87 } //nr2v(...)

88
89 static int[] inm(int[]x, int[]y)

```

```

90  {
91      int nx=x.length, ny=y.length, i,j,t;
92      int[][] a=new int[ny][nx+ny];
93      int[] z=new int[nx+ny];
94      for(j=0;j<ny;j++)
95      {
96          t=0;
97          for(i=0;i<nx;i++)
98          {
99              a[j][i+j]=y[j]*x[i]+t;
100             t=a[j][i+j]/10;
101             a[j][i+j]=a[j][i+j]%10;
102         }
103         a[j][j+nx]=t;
104     }
105     t=0;
106     for(j=0;j<nx+ny;j++)
107     {
108         z[j]=0;
109         for(i=0;i<ny;i++) z[j]=z[j]+a[i][j];
110         z[j]=z[j]+t;
111         t=z[j]/10;    z[j]=z[j]%10;
112     }
113     if(z[nx+ny-1]!=0) return z;
114     else
115     {
116         int[] zz=new int[nx+ny-1];
117         for(i=0;i<nx+ny-1;i++) zz[i]=z[i];
118         return zz;
119     }
120 } //inm(...)
121 } // class

```

35.1.3 Cod sursă

Listing 35.1.2: asediul.pas

```

1 type vector=array[0..100] of longint;
2 var v,v1,v2,v3:vector;
3     unu:array[0..100] of 0..1;
4     g:text;
5     n:longint;
6
7 procedure CreazaVector;
8 var f:text;
9 begin
10    assign(f,'asediu.in');
11    reset(f);
12    readln(f,n);
13    while n<>0 do
14    begin
15        inc(v[0]);
16        v[v[0]]:=n mod 10;
17        n:=n div 10;
18    end;
19 end;
20
21 procedure InmultireVectori(v1,v2:vector;var v:vector);
22 var i,j,nr,tr:integer;
23 begin
24    v[0]:=v1[0]+v2[0]-1;
25    for i:=1 to v[0]+1 do v[i]:=0;
26    for i:=1 to v1[0] do
27        for j:=1 to v2[0] do
28            v[i+j-1]:=v[i+j-1]+v1[i]*v2[j];
29    tr:=0;
30    for i:=1 to v[0] do
31    begin
32        nr:=v[i]+tr;
33        v[i]:=nr mod 10;
34        tr:=nr div 10;
35    end;

```

```

36   if tr<>0 then begin
37     inc(v[0]);
38     v[v[0]]:=tr;
39   end;
40 end;
41
42 procedure InmultireScalar(var v:vector;k:integer);
43 var i,tr,cifra:integer;
44 begin
45   tr:=0;
46   for i:=1 to v[0] do
47   begin
48     cifra:=(k*v[i]+tr) mod 10;
49     tr:=(k*v[i]+tr) div 10;
50     v[i]:=cifra;
51   end;
52   while tr<>0 do
53   begin
54     v[0]:=v[0]+1;
55     v[v[0]]:=tr;
56     tr:=tr div 10;
57   end;
58 end;
59
60 procedure ImpartireScalar(var v:vector;k:integer);
61 var i,tr:integer;
62 begin
63   tr:=0;
64   for i:=v[0] downto 1 do
65   begin
66     v[i]:=10*tr+v[i];
67     tr:=v[i] mod k;
68     v[i]:=v[i] div k;
69   end;
70   i:=v[0];
71   while v[i]=0 do
72   begin
73     dec(v[0]);
74     dec(i);
75   end;
76 end;
77
78 procedure afis;
79 var i:integer;
80 begin
81   for i:=v[0] downto 1 do
82     write(g,v[i]);
83 end;
84
85 procedure scade(var v1:vector);
86 var i,tr:integer;
87 begin
88   tr:=0;
89   unu[1]:=1;
90   for i:=1 to v1[0] do
91   begin
92     v1[i]:=v1[i]-(unu[i]+tr);
93     if v1[i]<0 then tr:=1 else tr:=0;
94     if tr=1 then v1[i]:=v1[i]+10;
95   end;
96   i:=v1[0];
97   while v1[i]=0 do
98   begin
99     dec(v1[0]);
100    dec(i);
101  end;
102 end;
103
104 procedure Combinari(k:integer;v:vector;var v1:vector);
105 var i:integer;
106 begin
107   {n*(n-1)*(n-2)*(n-3)}
108   v1:=v;
109   for i:=1 to k-1 do
110   begin
111     scade(v1);

```

```

112      InmultireVectori(v,v1,v);
113      end;
114  v1:=v;
115 end;
116
117 procedure SumaVectori;
118 var i,tr:integer;
119 begin
120  if v2[0]>v1[0] then
121      v[0]:=v2[0]
122  else
123      v[0]:=v1[0];
124  tr:=0;
125  for i:=1 to v[0] do
126  begin
127    v[i]:=v1[i]+v2[i]+unu[i]+tr;
128    tr:=v[i] div 10;
129    v[i]:=v[i] mod 10;
130  end;
131  if tr>>0 then
132    begin
133      inc(v[0]);
134      v[v[0]]:=tr;
135    end;
136 end;
137
138 begin
139  CreazaVector;
140  assign(g,'asediul.out');
141  rewrite(g);
142  if (n<>2) and (n<>3) then
143  begin
144    v3:=v;
145    combinari(4,v,v1);
146    ImpartireScalar(v1,24);
147    combinari(2,v3,v2);
148    ImpartireScalar(v2,2);
149    SumaVectori;
150    afis;
151  end
152  else
153    if n=2 then writeln(g,'2')
154    else if n=3 then writeln(g,'4');
155  close(g);
156 end.

```

35.2 Muzeu

Sunteți un participant la Olimpiada Națională de Informatică. În programul olimpiadei intră și câteva activități de divertisment. Una dintre ele este vizitarea unui muzeu. Acesta are o structură de matrice dreptunghiulară cu M linii și N coloane; din orice cameră se poate ajunge în camerele vecine pe direcțiile nord, est, sud și vest (dacă aceste camere există). Pentru poziția (i, j) deplasarea spre nord presupune trecerea în poziția $(i - 1, j)$, spre est în $(i, j + 1)$, spre sud în $(i + 1, j)$ și spre vest în $(i, j - 1)$.

Acest muzeu are câteva reguli speciale. Fiecare cameră este marcată cu un număr între 0 și 10 inclusiv. Mai multe camere pot fi marcate cu același număr. Camerele marcate cu numărul 0 pot fi vizitate gratuit. Într-o cameră marcată cu numărul i ($i > 0$) se poate intra gratuit, dar nu se poate ieși din ea decât dacă arătați supraveghetorului un bilet cu numărul i . Din fericire, orice cameră cu numărul i ($i > 0$) oferă spre vânzare un bilet cu numărul i ; o dată cumpărat acest bilet, el este valabil în toate camerele marcate cu numărul respectiv. Biletele pot avea prețuri diferite, dar un bilet cu numărul i va avea același preț în toate camerele în care este oferit spre vânzare.

Dumneavoastră intrați în muzeu prin colțul de Nord-Vest (poziția $(1, 1)$ a matricei) și doriți să ajungeți la ieșirea situată în colțul de Sud-Est (poziția (M, N) a matricei). O dată ajuns acolo primiți un bilet gratuit care vă permite să vizitați tot muzeul.

Pozиїile $(1, 1)$ și (M, N) sunt marcate cu numărul 0.

Cerință

Cunoscându-se structura muzeului, determinați o strategie de parcurs a camerelor, astfel

încât să ajungeti în camera (M, N) plătind cât mai puțin. Dacă există mai multe variante, alegeti una în care este parcurs un număr minim de camere (pentru a cătiga timp și pentru a avea mai mult timp pentru vizitarea integrală a muzeului).

Date de intrare

Prima linie a fișierului de intrare **muzeu.in** conține două numere întregi M și N , separate printr-un spațiu, numărul de linii, respectiv de coloane, al matricei care reprezintă muzeul. Următoarele M linii conțin structura muzeului; fiecare conține N numere întregi între 0 și 10 inclusiv, separate prin spații. Linia $M + 2$ conține 10 numere întregi între 0 și 10000 inclusiv, reprezentând costurile biletelor 1, 2, 3, ... 10 în această ordine.

Date de ieșire

În fișierul **muzeu.out** veți afișa:

pe prima linie suma minimă necesară pentru a ajunge din $(1, 1)$ în (M, N) ;

pe a doua linie numărul minim de mutări L efectuate dintr-o cameră într-o cameră vecină, pentru a ajunge din $(1, 1)$ în (M, N) ;

pe a treia linie L caractere din multimea N, E, S, V reprezentând deplasări spre Nord, Est, Sud sau Vest.

Restricții și precizări

$2 \leq N \leq 50$

Exemplu:

muzeu.in	muzeu.out
5 6	12
0 0 0 0 0 2	9
0 1 1 1 4 3	EEEEESSSSS
0 1 0 0 0 0	
0 1 5 1 0 0	
0 0 0 1 0 0	
1000 5 7 100 12 1000 1000 1000 1000	

Timp maxim de executare: 1 secundă/test.

35.2.1 Indicații de rezolvare

Mihai Stroe, GInfo nr. 13/6 - octombrie 2003

Se observă că numărul variantelor de cumpărare a biletelor (cumpăr / nu cumpăr biletul 1, biletul 2, ..., biletul 10) este $2^{10} = 1024$.

Se generează fiecare din aceste variante. Pentru o astfel de variantă, se calculează costul și se transformă matricea inițială într-o matrice cu 0 și 1, în care 0 reprezintă o cameră pentru care se plătește biletul (sau nu e nevoie de bilet) și 1 reprezintă o cameră pentru care nu se cumpără bilet (deci în care nu se intră).

Pentru o astfel de matrice, problema determinării celui mai scurt drum de la $(1, 1)$ la (M, N) se rezolvă cu *algoritmul lui Lee*.

Se rezolvă această problemă pentru toate cele 1024 variante. Eventual, dacă la un moment dat există o soluție cu un cost mai bun decât al variantei curente, aceasta nu mai este abordată.

Evident, problema determinării celui mai scurt drum se poate rezolva și pe matricea inițială, ținând cont la fiecare pas de biletele cumpărate; cum *algoritmul lui Lee* este folosit de obicei pentru o matrice cu 0 și 1, am folosit inițial convenția respectivă pentru a ușura înțelegerea.

Se alege soluția de cost minim și, în caz de egalitate, cea cu număr minim de camere.

Analiza complexității

Operația de citire a datelor are ordinul de complexitate $O(M \cdot N)$.

Fie C numărul de numere de marcat diferențe din matrice. Pentru fiecare dintre cele 2^C variante, găsirea drumului minim cu *algoritmul lui Lee* are complexitatea $O(M \cdot N)$.

Operația de scriere a soluției are ordinul de complexitate $O(M \cdot N)$ pentru cazul cel mai defavorabil.

Ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(M \cdot N \cdot 2^C)$.

Algoritmul funcționează datorită restricției impuse pentru C (cel mult 10 numere de marcat). Considerând 2^C o constantă, ordinul de complexitate devine $O(M \cdot N)$.

35.2.2 Rezolvare detaliată

Listing 35.2.1: muzeu.java

```

1 import java.io.*;
2 class Muzeu
3 {
4     static final int qdim=200;           // dimensiune coada circulara
5     static final int sus=1, dreapta=2, jos=3, stanga=4;
6
7     static int m,n;                   // dimensiuni muzeu
8     static int ncmin;                 // nr camere minim
9     static int cc,cmin=50*50*10000/2; // cost curent/minim
10    static boolean amAjuns;
11
12    static int[][] x=new int[51][51];   // muzeu
13    static int[][] c=new int[51][51];   // costul (1,1) --> (i,j)
14    static int[][] d=new int[51][51];   // directii pentru "intoarcere"
15    static int[][] dsol=new int[51][51];
16
17    static int[] cb=new int[11];        // costuri bilete
18    static boolean[] amBilet=new boolean[11];
19
20    static int[] qi=new int[qdim];    // coada pentru i din pozitia (i,j)
21    static int[] qj=new int[qdim];    // coada pentru j din pozitia (i,j)
22    static int ic, sc;               // ic=inceput coada
23
24    public static void main(String[] args) throws IOException
25    {
26        int i,j;
27        StreamTokenizer st=new StreamTokenizer(
28            new BufferedReader(new FileReader("10-muzeu.in")));
29
30        st.nextToken(); m=(int)st.nval;
31        st.nextToken(); n=(int)st.nval;
32
33        for(i=1;i<=m;i++)
34            for(j=1;j<=n;j++) { st.nextToken(); x[i][j]=(int)st.nval; }
35
36        for(i=1;i<=10;i++) { st.nextToken(); cb[i]=(int)st.nval; }
37
38        amBilet[0]=true; // 0 ==> gratuit
39        for(i=0;i<=1023;i++)
40        {
41            bilet(i);
42            cc=0; for(j=1;j<=10;j++) if(amBilet[j]) cc+=cb[j];
43            if(cc>cmin) continue;
44
45            amAjuns=false;
46            matriceCosturi();
47
48            if(!amAjuns) continue;
49            if(cc>cmin) continue;
50            if(cc<cmin) { cmin=cc; ncmin=c[m][n]; copieDirectii(); }
51            else // costuri egale
52                if(c[m][n]<ncmin) { ncmin=c[m][n]; copieDirectii(); }
53        }
54
55        d=dsol; // schimbare "adresa" ... ("pointer") ...
56        afisSolutia();
57    } // main()
58
59    static void bilet(int i)
60    {
61        int j;
62        for(j=1;j<=10;j++)
63        {
64            if(i%2==1) amBilet[j]=true; else amBilet[j]=false;
65            i/=2;
66        }
67    } // bilet(...)
68
69    static void matriceCosturi()
70    {
71        int i,j;
72        for(i=1;i<=m;i++)
73            for(j=1;j<=n;j++) c[i][j]=0;           // curat "numai" traseul !
74        ic=sc=0; // coada vida

```

```

75     qi[sc]=1;   qj[sc]=1;   sc=(sc+1)%qdim; // (1,1) --> coada circulara !
76     c[1][1]=1; // cost 1 pentru pozitia (1,1) (pentru marcaj!)
77     while(ic!=sc) // coada nevida
78     {
79         i=qi[ic]; j=qj[ic]; ic=(ic+1)%qdim;
80         vecini(i,j);
81         if(amAjuns) break;
82     }
83 } //matriceCosturi()

84
85 static void copieDirectii()
86 {
87     int i,j;
88     for(i=1;i<=m;i++) for(j=1;j<=n;j++) dsol[i][j]=d[i][j];
89 } // copieDirectii()

90
91 static void vecini(int i, int j)
92 {
93     int t=c[i][j]; // "tmp" = nr camere parcurse
94
95     if((i-1>=1)&&(c[i-1][j]==0)&&amBilet[x[i-1][j]]) // N
96     {
97         c[i-1][j]=t+1; qi[sc]=i-1; qj[sc]=j; sc=(sc+1)%qdim;
98         d[i-1][j]=jos;
99         if((i-1==m)&&(j==n)) {amAjuns=true; return;}
100    }
101
102    if((j+1<=n)&&(c[i][j+1]==0)&&amBilet[x[i][j+1]]) // E
103    {
104        c[i][j+1]=t+1; qi[sc]=i; qj[sc]=j+1; sc=(sc+1)%qdim;
105        d[i][j+1]=stanga;
106        if((i==m)&&(j+1==n)) {amAjuns=true; return;}
107    }
108
109    if((i+1<=m)&&(c[i+1][j]==0)&&amBilet[x[i+1][j]]) // S
110    {
111        c[i+1][j]=t+1; qi[sc]=i+1; qj[sc]=j; sc=(sc+1)%qdim;
112        d[i+1][j]=sus;
113        if((i+1==m)&&(j==n)) {amAjuns=true; return;}
114    }
115
116    if((j-1>=1)&&(c[i][j-1]==0)&&amBilet[x[i][j-1]]) // V
117    {
118        c[i][j-1]=t+1; qi[sc]=i; qj[sc]=j-1; sc=(sc+1)%qdim;
119        d[i][j-1]=dreapta;
120        if((i==m)&&(j-1==n)) {amAjuns=true; return;}
121    }
122 } // vecini(...)

123
124 static void afisSolutia() throws IOException
125 {
126     int i,j;
127     PrintWriter out = new PrintWriter(
128         new BufferedWriter(new FileWriter("muzeu.out")));
129
130     out.println(cmin);
131     out.println(ncmin-1);
132
133     i=m; j=n; // (m,n) --> (1,1)
134     while((i!=1)|| (j!=1)) // folosesc "c" care nu mai e necesara !
135     {
136         if(d[i][j]==sus) { c[i-1][j]=jos; i--; }
137         else if(d[i][j]==jos) { c[i+1][j]=sus; i++; }
138         else if(d[i][j]==dreapta) { c[i][j+1]=stanga; j++; }
139         else if(d[i][j]==stanga) { c[i][j-1]=dreapta; j--; }
140         else System.out.println("Eroare la traseu ... (m,n)-->(1,1)!");
141
142         i=1; j=1; // (1,1) --> (m,n)
143         while((i!=m)|| (j!=n))
144         {
145             if(c[i][j]==sus) {out.print("N"); i--;}
146             else if(c[i][j]==jos) {out.print("S"); i++;}
147             else if(c[i][j]==dreapta) {out.print("E"); j++;}
148             else if(c[i][j]==stanga) {out.print("V"); j--;}
149             else System.out.println("Eroare la traseu ... (1,1)--(m,n)!");
150         }
151     }
152     out.close();

```

```

151     } //afisSolutia()
152 } // class

```

35.2.3 Cod sursă

Listing 35.2.2: muzeu.pas

```

1 {$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S+,T-,V+,X+,Y+}
2 {$M 16384,0,655360}
3 const put:array[1..10]of longint=(1,2,4,8,16,32,64,128,256,512);
4 { NESV }
5 const dl:array[1..4]of integer=(-1,0,1,0);
6 const dc:array[1..4]of integer=(0,1,0,-1);
7 const dir:string='NESV';
8 var suma,start,finish,nropt,optim,lopt,i,j,k,l,m,n,nr:longint;
9     fi,fo:text;
10    a,b,c,sel:array[0..51,0..51]of integer;
11    cost,ok:array[1..100]of longint;
12    cx,cy:array[1..2500]of integer;
13
14 procedure readdata;
15 begin
16     assign(fi,'muzeu.in');
17     assign(fo,'muzeu.out');
18     reset(fi);
19     readln(fi,m,n);
20     for i:=0 to m+1 do
21         for j:=0 to n+1 do
22             a[i,j]:=1;
23     for i:=1 to m do
24         begin
25             for j:=1 to n do
26                 read(fi,a[i,j]);
27             readln(fi);
28         end;
29     for i:=1 to 10 do
30         read(fi,cost[i]);
31     close(fi);
32     b:=a;
33 end;
34
35 procedure try;
36 begin
37     for i:=1 to 10 do
38         ok[i]:=nr and put[i];
39     for i:=1 to 10 do
40         if ok[i]>0 then
41             ok[i]:=1;
42     suma:=0;
43     for i:=1 to 10 do
44         suma:=suma+ok[i]*cost[i];
45     if suma>optim then exit;
46
47     a:=b;
48     for i:=1 to m do
49         for j:=1 to n do
50             if a[i,j]<>0 then
51                 if ok[a[i,j]]=1 then
52                     a[i,j]:=0
53                 else
54                     a[i,j]:=1;
55
56     for i:=1 to m do
57         for j:=1 to n do
58             sel[i,j]:=0;
59     start:=1;
60     finish:=1;
61
62     cx[1]:=1;
63     cy[1]:=1;
64     sel[1,1]:=1;
65

```

```

66  while start<=finish do
67    begin
68      i:=cx[start];
69      j:=cy[start];
70      for k:=1 to 4 do
71        if a[i+dl[k],j+dc[k]]=0 then
72          if sel[i+dl[k],j+dc[k]]=0 then
73            begin
74              sel[i+dl[k],j+dc[k]]:=sel[i,j]+1;
75              inc(finish);
76              cx[finish]:=i+dl[k];
77              cy[finish]:=j+dc[k];
78            end;
79            inc(start);
80          end;
81        if sel[m,n]=0 then exit;
82        if suma=optim then
83          if sel[m,n]<l0pt then
84            begin
85              optim:=suma;
86              l0pt:=sel[m,n];
87              c:=sel;
88              nropt:=nr;
89            end;
90        if suma<optim then
91          begin
92              optim:=suma;
93              l0pt:=sel[m,n];
94              c:=sel;
95              nropt:=nr;
96            end;
97        end;
98
99  procedure rec(i,j:integer);
100 var k:integer;
101 begin
102   if (i=1)and(j=1) then
103     exit;
104   for k:=1 to 4 do
105     if a[i-dl[k],j-dc[k]]=a[i,j]-1 then
106       begin
107         rec(i-dl[k],j-dc[k]);
108         write(fo,dir[k]);
109         exit;
110       end;
111   end;
112
113 procedure afis;
114 begin
115   a:=c;
116   rewrite(fo);
117   writeln(fo,optim);
118   writeln(fo,l0pt-1);
119   rec(m,n);
120   writeln(fo);
121   close(fo);
122 end;
123
124 procedure solve;
125 begin
126   optim:=maxlongint;
127   l0pt:=maxlongint;
128   for nr:=0 to 1023 do
129     try;
130     afis;
131     { writeln(nropt); }
132   end;
133
134 begin
135   readdata;
136   solve;
137 end.

```

35.3 Munte

Într-o zonă montană se dorește deschiderea unui lanț de telecabine. Stațiile de telecabine pot fi înființate pe oricare din cele N vârfuri ale zonei montane. Vârfurile sunt date în ordine de la stânga la dreapta și numerotate de la 1 la N , fiecare vârf i fiind precizat prin coordonata $X[i]$ pe axa OX și prin înălțimea $H[i]$.

Se vor înființa exact K stații de telecabine. Stația de telecabine i ($2 \leq i \leq K$) va fi conectată cu stațiile $i - 1$ și $i + 1$; stația 1 va fi conectată doar cu stația 2, iar stația K , doar cu stația $K - 1$. Stația 1 va fi obligatoriu amplasată în vârful 1, iar stația K în vârful N .

Se dorește ca lanțul de telecabine să asigure legătura între vârful 1 și vârful N . Mai mult, se dorește ca lungimea totală a cablurilor folosite pentru conectare să fie minimă. Lungimea cablului folosit pentru a conecta două stații este egală cu distanța dintre ele. În plus, un cablu care unește două stații consecutive nu poate avea lungimea mai mare decât o lungime fixată L .

O restricție suplimentară este introdusă de formele de relief. Astfel, vârfurile i și j ($i < j$) nu pot fi conectate direct dacă există un vârf v ($i < v < j$) astfel încât segmentul de dreapta care ar uni vârfurile i și j nu ar trece pe deasupra vârfului v . În cazul în care cele trei vârfuri sunt coliniare, se consideră toate trei ca fiind stații, chiar dacă distanța dintre vârfurile i și j este mai mică decât L .

Cerință

Dându-se amplasarea celor N vârfuri ale lanțului muntos, stabiliți o modalitate de dispunere a celor K stații de telecabine astfel încât lungimea totală a cablurilor folosite pentru conectare să fie minimă, cu restricțiile de mai sus.

Se garantează că, pe toate testele date la evaluare, conectarea va fi posibilă.

Date de intrare

Prima linie a fișierului de intrare **munte.in** conține trei numere întregi N , K și L , separate prin spații, cu semnificațiile de mai sus. Următoarele N linii conțin coordonatele vârfurilor; linia $i + 1$ conține coordonatele vârfului i , $X[i]$ și $H[i]$, separate printr-un spațiu.

Date de ieșire

În fișierul **munte.out** veți afișa:

- pe prima linie lungimea totală minimă a cablurilor, rotunjită la cel mai apropiat număr întreg (pentru orice întreg Q , $Q.5$ se rotunjeste la $Q + 1$);
- pe a doua linie K numere distincte între 1 și N , ordonate crescător, numerele vârfurilor în care se vor înființa stații de telecabine. Dacă există mai multe variante, afișați una oarecare.

Restricții și precizări

$$2 \leq N \leq 100$$

$$2 \leq K \leq 30 \text{ și } K \leq N$$

$$0 \leq L, X[i], H[i] \leq 100.000 \text{ și } X[i] < X[i + 1]$$

Exemplu

munte.in munte.out

7 5 11	22
0 16	1 3 5 6 7
4 3	
6 8	
7 4	
12 16	
13 16	
14 16	

Explicații

- trasarea unui cablu direct între vârfurile 1 și 5 ar fi contravenit restricției referitoare la lungimea maximă a unui cablu; în plus, s-ar fi obținut o soluție cu 2 stații de telecabine în loc de 3 (deci soluția ar fi invalidă și pentru valori mari ale lui L);

- pentru a ilustra restricția introdusă de formele de relief, precizăm că vârfurile 1 și 4 nu au putut fi conectate direct datorită înălțimii vârfului 3. De asemenea, vârfurile 5 și 7 nu au putut fi conectate direct datorită înălțimii vârfului 6.

Timp maxim de executare: 1 secundă/test.

35.3.1 Indicații de rezolvare

Problema se rezolvă prin *metoda programării dinamice*.

Practic, problema se poate împărți în două subprobleme. Primul pas constă în determinarea perechilor de vârfuri care pot fi unite printr-un cablu. Acest pas se rezolvă folosind cunoștințe elementare de geometrie plană (ecuația dreptei, $y = a * x + b$). Se va obține o matrice OK , unde $OK_{i,j}$ are valoarea 1 dacă vârfurile i și j pot fi unite și 0 în caz contrar. Folosind această matrice, se vor conecta vârfurile 1 și N cu un lanț de K stații, astfel încât oricare două stații consecutive să aibă OK -ul corespunzător egal cu 1.

Această subproblemă se rezolvă prin *metoda programării dinamice* după cum urmează: se construiește o matrice A cu K linii și N coloane, unde $A_{i,j}$ reprezintă lungimea totală minimă a unui lanț cu i stații care conectează vârfurile 1 și j .

Inițial $A_{1,1} = 0$, $A_{1,i} = +\infty$ și $A_{i,1} = +\infty$ pentru $i > 1$.

Pentru i cuprins între 2 și N , componentele matricei se calculează astfel:

$$A_{i,j} = \min\{A_{i-1,v} + dist(v,j)\}, \text{ unde } v < j \text{ și } OK_{v,j} = 1$$

Concomitent cu calculul elementelor $A_{i,j}$ se construiește o matrice T , unde $T_{i,j}$ reprezintă v -ul care minimizează expresia de mai sus. Matricea T este folosită pentru reconstituirea soluției.

Lungimea totală minimă este regasită în $A_{K,N}$.

Subproblemele puteau fi tratate și simultan, ceea ce complica implementarea.

Analiza complexității

Operația de citire a datelor are ordinul de complexitate $O(N)$.

Calculul matricei OK are ordinul de complexitate $O(N^3)$.

Algoritmul bazat pe *metoda programării dinamice* are ordinul de complexitate $O(N^2 \cdot K)$.

Reconstituirea soluției și afișarea au ordinul de complexitate $O(N)$.

Deoarece $K \leq N$, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(N^3)$.

35.3.2 Rezolvare detaliată

Prima variantă:

```
import java.io.*;           // cu mesaje pt depanare dar ... fara traseu
class Munte
{
    static final int oo=Integer.MAX_VALUE;
    static int n,m,L;        // m=nr statii (in loc de K din enunt)

    static int[] x,h;
    static double[][] a;
    static int[][] t;
    static boolean[][] ok;

    public static void main(String[] args) throws IOException
    {
        long t1,t2;
        t1=System.currentTimeMillis();

        int i;
        BufferedReader br=new BufferedReader(new FileReader("munte.in"));
        StreamTokenizer st=new StreamTokenizer(br);

        st.nextToken(); n=(int)st.nval;
        st.nextToken(); m=(int)st.nval;
        st.nextToken(); L=(int)st.nval;

        x=new int[n+1];
        h=new int[n+1];
        ok=new boolean[n+1][n+1];      // implicit este false
        a=new double[m+1][n+1];
        t=new int[m+1][n+1];
    }
}
```

```

for(i=1;i<=n;i++)
{
    st.nextToken(); x[i]=(int)st.nval;
    st.nextToken(); h[i]=(int)st.nval;
}

matriceaOK();
afism(ok);

matriceaA();

afisSolutia();

t2=System.currentTimeMillis();
System.out.println("Timp = "+(t2-t1));
}// main()

static void matriceaA()
{
    int i,j,k,kmin;
    double d,dkj,min;
    a[1][1]=0;
    for(i=2;i<=m;i++) a[i][1]=oo;
    for(j=2;j<=n;j++) a[1][j]=oo;
    afism(a);

    for(i=2;i<=m;i++)
    {
        for(j=2;j<=n;j++)
        {
            min=oo;
            kmin=-1;
            for(k=1;k<j;k++)
            {
                System.out.println(i+" "+j+" "+k+" "+ok[k][j]);
                if(ok[k][j])
                {
                    dkj=dist(k,j);
                    d=a[i-1][k]+dkj;
                    System.out.println(i+" "+j+" "+k+" dkj="+dkj+" d="+d+" min="+min);
                    if(d<min) { min=d;kmin=k; }
                }
            }
            // for k
            a[i][j]=min;
        }
        // for j
        System.out.println("Linia: "+i);
        afism(a);
    }
    // for i
}

static double dist(int i, int j)
{
    double d;
    d=(double)(x[i]-x[j])*(x[i]-x[j])+(double)(h[i]-h[j])*(h[i]-h[j]);
    return Math.sqrt(d);
}

static void matriceaOK()
{
}

```

```

int i,j,ij,x1,y1,x2,y2,sp1,sp2;
for(i=1;i<=n-1;i++)
{
    x1=x[i]; y1=h[i];
    for(j=i+1;j<=n;j++)
    {
        x2=x[j]; y2=h[j];
        ok[i][j]=ok[j][i]=true;
        for(ij=i+1;ij<=j-1;ij++) // i .. ij .. j
        {
            sp1=(0-y1)*(x2-x1)-(y2-y1)*(x[ij]-x1);
            sp2=(h[ij]-y1)*(x2-x1)-(y2-y1)*(x[ij]-x1);
            if(sp1*sp2<=0)
            {
                ok[i][j]=ok[j][i]=false;
                System.out.println(i+" "+j+" ("+ij+") \t"+sp1+"\t"+sp2);
                break;
            }
            if(!ok[i][j]) break;
        }//for ij
        if(ok[i][j]) if(dist(i,j)>L+0.0000001) ok[i][j]=ok[j][i]=false;
    }//for j
}// for i
}// matriceaOK()

static void afisSolutia() throws IOException
{
    int i,j;
    PrintWriter out = new PrintWriter(
        new BufferedWriter(new FileWriter("munte.out")));
    out.println(a[m][n]);
    out.close();
}//afisSolutia()

static void afism(int[][] a)
{
    int i,j;
    for(i=0;i<a.length;i++)
    {
        for(j=0;j<a[i].length;j++) System.out.print(a[i][j]+" ");
        System.out.println();
    }
    System.out.println();
}// afism(...)

static void afism(boolean[][] a)
{
    int i,j;
    for(i=1;i<a.length;i++)
    {
        for(j=1;j<a[i].length;j++) System.out.print(a[i][j]+" ");
        System.out.println();
    }
    System.out.println();
}// afism(...)

static void afism(double[][] a)
{
    int i,j;
    for(i=1;i<a.length;i++)

```

```

{
    for(j=1;j<a[i].length;j++) System.out.print(a[i][j]+" ");
    System.out.println();
}
System.out.println();
}// afis(...)

}// class

```

A doua variantă:

```

import java.io.*; // FINAL: fara mesaje si cu traseu ... dar
class Munte2 // test 8 : P1(99,59) P2(171,96) P3(239,81) P4(300,78)
{           // solutia: 1 4 5 ... este gresita (1 4 nu trece de P2 si P3!)

    static final int oo=Integer.MAX_VALUE;
    static int n,m,L; // m=nr statii (in loc de K din enunt)

    static int[] x,h;
    static double[][] a;
    static int[][] t;
    static boolean[][] ok;
    static int[] statia;

    public static void main(String[] args) throws IOException
    {
        long t1,t2;
        t1=System.currentTimeMillis();

        int i,j;
        BufferedReader br=new BufferedReader(new FileReader("munte.in"));
        StreamTokenizer st=new StreamTokenizer(br);

        st.nextToken(); n=(int)st.nval;
        st.nextToken(); m=(int)st.nval;
        st.nextToken(); L=(int)st.nval;

        x=new int[n+1];
        h=new int[n+1];
        ok=new boolean[n+1][n+1];
        a=new double[m+1][n+1];
        t=new int[m+1][n+1];
        statia=new int[m+1];
        for(i=1;i<=n;i++)
        {
            st.nextToken(); x[i]=(int)st.nval;
            st.nextToken(); h[i]=(int)st.nval;
        }

        matriceaOK();
        matriceaA();
        j=n;
        for(i=m;i>=1;i--) { statia[i]=j; j=t[i][j]; }
        afisSolutia();

        t2=System.currentTimeMillis();
        System.out.println("Timp = "+(t2-t1));
    }// main()

    static void matriceaA()
    {
        int i,j,k,kmin;
    }
}

```

```

double d,dkj,min;
a[1][1]=0;
for(i=2;i<=m;i++) a[i][1]=oo;
for(j=2;j<=n;j++) a[1][j]=oo;

for(i=2;i<=m;i++)
{
    for(j=2;j<=n;j++)
    {
        min=oo;
        kmin=0;
        for(k=1;k<j;k++)
        {
            if(ok[k][j])
            {
                dkj=dist(k,j);
                d=a[i-1][k]+dkj;
                if(d<min) { min=d;kmin=k; }
            }
        }// for k
        a[i][j]=min;
        t[i][j]=kmin;
    }// for j
}// for i
}// matriceaA()

static double dist(int i, int j)
{
    double d;
    d=(double)(x[i]-x[j])*(x[i]-x[j])+(double)(h[i]-h[j])*(h[i]-h[j]);
    return Math.sqrt(d);
}// dist(...)

static void matriceaOK()
{
    int i,j,ij,x1,y1,x2,y2;
    long spl,sp2;           // 100.000*100.000=10.000.000.000 depaseste int !!!
    for(i=1;i<=n-1;i++)
    {
        x1=x[i]; y1=h[i];
        for(j=i+1;j<=n;j++)
        {
            x2=x[j]; y2=h[j];
            ok[i][j]=ok[j][i]=true;
            for(ij=i+1;ij<=j-1;ij++)          // i .. ij .. j
            {
                spl=(0-y1)*(x2-x1)-(y2-y1)*(x[ij]-x1);
                sp2=(h[ij]-y1)*(x2-x1)-(y2-y1)*(x[ij]-x1);
                if(spl*sp2<=0)
                {
                    ok[i][j]=ok[j][i]=false;
                    break;
                }
                if(!ok[i][j]) break;
            }//for ij
            if(ok[i][j]) if(dist(i,j)>L) ok[i][j]=ok[j][i]=false;
        }//for j
    }// for i
}// matriceaOK()

```

```

    static void afisSolutia() throws IOException
    {
        int i;
        PrintWriter out = new PrintWriter(
            new BufferedWriter(new FileWriter("munte.out")));
        out.println((int)(a[m][n]+0.5));
        for(i=1;i<=m;i++) out.print(statia[i]+" ");
        out.println();
        out.close();
    } //afisSolutia()
} // class

```

35.3.3 Cod sursă

Listing 35.3.1: munte.cpp

```

1 #include <iostream>
2 #include <stdio.h>
3 #include <math.h>
4
5 using namespace std;
6
7 float *a[2], **b;
8 int n,K,l,L,**c,*d;
9
10 void citire();
11 void rezolv();
12 void shtdwn();
13 float pune(int a1,int a2);
14 void drum(int a1,int a2);
15 void scrier();
16 int mic(int a);
17 int mare(int a);
18
19 int main()
20 {
21     citire();
22     rezolv();
23     shtdwn();
24     return 0;
25 }
26
27 void citire()
28 {
29     int i;
30     FILE *f;
31     f = fopen("munte.in","r");
32     fscanf(f,"%d %d %d\n",&n,&K,&l);
33     a[0] = new float [n+1];
34     a[1] = new float [n+1];
35     for(i=1;i<=n;i++)
36         fscanf(f,"%f %f\n",&a[0][i],&a[1][i]);
37     fclose(f);
38 }
39
40 void rezolv()
41 {
42     int i,j,k,aux;
43
44     b = new float *[n+1];
45     for(i=0;i<=n;i++)
46         b[i] = new float [n+1];
47     c = new int *[n+1];
48     for(i=0;i<=n;i++)
49         c[i] = new int [n+1];
50     d = new int[n+1];
51     for(i=0;i<=n;i++)
52         d[i] = 0;
53
54     for(i=0;i<=n;i++)
55         for(j=0;j<=n;j++)

```

```

56         c[i][j]=0;
57
58     L = l*l;
59
60     for(i=1;i<=n;i++)
61         for(j=i+1;j<=n;j++)
62         {
63             b[i][j] = pune(i,j);
64             if(b[i][j])
65                 c[i][j]=1;
66         }
67     delete []a[1];
68     delete []a[0];
69
70     for(k=1;k<=n;k++)
71         for(i=1;i<=n;i++)
72             for(j=i+1;j<=n;j++)
73             if(c[i][k] && c[k][j])
74                 if(!b[i][j] || (b[i][j]>b[i][k]+b[k][j]))
75                 {
76                     c[i][j] = c[i][k]+ c[k][j];
77                     b[i][j] = b[i][k]+ b[k][j];
78                 }
79
80     d[1]=1;
81     drum(1,n);
82     if(c[1][n] == K-1)
83         scrier();
84     else
85     {
86         float cost,C;
87         int poz,m,M;
88         aux = K - 1 - c[1][n];
89         while(aux)
90         {
91             cost= 99999999999999999999999999999999.f;
92             for(i=2;i<n;i++)
93             if(!d[i])
94             {
95                 m = mic(i);
96                 M = mare(i);
97                 C = b[m][i] + b[i][M];
98                 if(C<cost)
99                 {
100                     cost = C;
101                     poz = i;
102                 }
103             }
104             b[1][n]+=cost;
105             d[poz]=1;
106             aux--;
107         }
108         scrier();
109     }
110 }
111
112 void shtdwn()
113 {
114     int i;
115     for(i=1;i<=n;i++)
116         delete []b[i];
117     delete []b;
118     for(i=1;i<=n;i++)
119         delete []c[i];
120     delete []c;
121     delete []d;
122 }
123
124 float pune(int a1,int a2)
125 {
126     int i;
127     float d,difx,dify;
128     difx = a[0][a1]-a[0][a2];
129     dify = a[1][a1]-a[1][a2];
130     d = difx*difx + dify*dify;
131     if(d>L)

```

```

132     return 0;
133     for(i=a1+1; i<a2; i++)
134         if(a[1][i] >= ( (a[0][i]-a[0][a1])*dify + a[1][a1]*difx )/difx )
135             return 0;
136     return (float)sqrt(d);
137 }
138
139 void drum(int a1,int a2)
140 {
141     int k;
142     if(a1>=a2)
143         return;
144     if(c[a1][a2]==1)
145         d[a2]=1;
146     else
147         for(k=a1+1;k<=n;k++)
148             if(c[a1][k] && c[k][a2])
149                 if(b[a1][a2] == b[a1][k] + b[k][a2])
150                 {
151                     drum(a1,k);
152                     drum(k,a2);
153                     return;
154                 }
155 }
156
157 void scrier()
158 {
159     FILE *f;
160     f = fopen("munte.out", "w");
161     fprintf(f, "%d\n", (int)b[1][n]);
162     for(int i=1;i<=n;i++)
163         if(d[i])
164             fprintf(f, "%d ",i);
165     fclose(f);
166 }
167
168 int mic(int a)
169 {
170     for(int i=a; i>0; i--)
171         if(d[i])
172             return i;
173     return 0;
174 }
175
176 int mare(int a)
177 {
178     for(int i=a; i<=n; i++)
179         if(d[i])
180             return i;
181     return 0;
182 }
```

35.4 Partiție

Se definește o partiție a unui număr natural n ca fiind o scriere a lui n sub forma:

$$n = n_1 + n_2 + \dots + n_k, \quad (k \geq 1)$$

unde n_1, n_2, \dots, n_k sunt numere naturale care verifică următoarea relație:

$$n_1 \geq n_2 \geq \dots \geq n_i \geq \dots \geq n_k \geq 1$$

Cerință

Fiind dat un număr natural n , să se determine câte partiții ale lui se pot scrie, conform cerințelor de mai sus, știind că oricare număr n_i dintr-o partiție trebuie să fie un număr impar.

Datele de intrare

Fișierul **partitie.in** conține pe prima linie numărul n

Datele de ieșire

Fișerul **partitie.out** va conține pe prima linie numărul de partiții ale lui n conform cerințelor problemei.

Restricții și precizări

- $1 \leq N \leq 160$

Exemplu

partitie.in	partitie.out
7	5

Explicații:

Cele cinci partiții sunt:

- $1+1+1+1+1+1+1$
- $1+1+1+1+3$
- $1+1+5$
- $1+3+3$
- 7

Timp maxim de executare: 3 secunde/test

35.4.1 Indicații de rezolvare

Stelian Ciurea

Problema se poate rezolva în mai multe moduri:

Soluția comisiei se bazează pe următoarele formule și *teoreme de combinatorică*:

- numărul de partiții ale unui număr n în k părți (nu neapărat distințe) este

$$P(n, k) = P(n - k, 1) + P(n - k, 2) + \dots + P(n - k, k)$$

cu $P(n, 1) = P(n, n) = 1$ și $P(n, k) = 0$ dacă $n < k$.

- numărul de partiții ale unui număr n în k părți distințe este

$$P(n, k) = P(n - k(k - 1)/2, k)$$

– numărul de partiții ale unui număr n în k părți impare care se pot și repeta este egal cu numărul de partiții ale unui număr n în k părți distințe.

Problema se poate rezolva și prin backtracking; fără prea mari optimizări se poate obține rezultatul în mai puțin de 3 secunde pentru $n < 120$. Pentru valori mai mari ale lui n , se poate lăsa programul să ruleze și se rețin rezultatele într-un vector cu valori inițiale.

Rezultatul pentru $n = 160$ are 8 cifre, deci nu este necesară implementarea operațiilor cu *numere mari*!

Mihai Stroe, GInfo nr. 13/6 - octombrie 2003

Problema se poate rezolva în mai multe moduri.

O idee bună de rezolvare a problemei constă în folosirea *metodei programării dinamice*. Se poate construi o matrice A , unde $A_{i,k}$ reprezintă numărul de partiții ale numărului i cu numere impare, din care ultimul este cel mult egal cu k . Un element din A se calculează observând că o partiție a lui i cu numere impare, cel mult egale cu k , este formată dintr-un un număr M , mai mic sau egal cu k , și alte numere, mai mici sau egale cu M . De aici rezultă relația:

$$A_{i,k} = \sum_{\substack{M=1, \dots, k; M \leq i; \\ M = impar}} A_{i-M,M}$$

Inițial $A_{0,0}$ este 1. La implementare, pentru a economisi spațiu, se poate alege o variantă în care $A_{i,k}$ să reprezinte numărul de partiții ale lui i cu numere impare, din care ultimul este cel mult egal cu al k -lea număr impar, adică $2 \cdot k - 1$.

După calcularea elementelor matricei, soluția pentru numărul i se găsește în $A_{i,i}$. Aceste valori pot fi salvate într-un vector de constante, care este transformat într-un nou program, în același mod ca la problema "Circular" de la clasa a IX-a. Această metodă conduce la o rezolvare instantanee a testelor date în concurs.

O altă metodă, bazată pe vectorul de constante, ar fi însemnat generarea soluțiilor folosind *metoda backtracking*. Un backtracking lăsat să se execute în timpul concursului ar fi putut genera soluțiile pentru toate valorile lui N , după care se putea folosi metoda vectorului de constante, în

tim ce un backtracking folosit ca soluție a problemei ar fi obținut punctajul maxim doar pentru testele mici și medii (pentru valori ale lui N mai mici decât 130).

Limitele problemei și timpul de execuție de 3 secunde permit rezolvării prin backtracking și bătinerea unui punctaj mulțumitor.

Analiza complexității

Pentru o rezolvare care se bazează pe metoda vectorului de constante, ordinul de complexitate al soluției finale ar fi fost $O(1)$; soluția constă în citirea valorii lui N și afișarea rezultatului memorat.

Soluția descrisă anterior, bazată pe metoda programării dinamice, are ordinul de complexitate $O(n^3)$. Invităm cititorul să caute metode mai eficiente.

Ordinul de complexitate al unei soluții care se bazează pe metoda backtracking este exponential.

35.4.2 Rezolvare detaliată

Listing 35.4.1: PartitieNrGenerare.java

```

1 class PartitieNrGenerare // n = suma de numere
2 {
3     static int dim=0,nsol=0;
4     static int n=6;
5     static int[] x=new int[n+1];
6
7     public static void main(String[] args)
8     {
9         long t1,t2;
10        t1=System.currentTimeMillis();
11        f(n,n,1);
12        t2=System.currentTimeMillis();
13        System.out.println("nsol = "+nsol+" timp = "+(t2-t1)+"\n");
14    } // main...
15
16    static void f(int val, int maxp, int poz)
17    {
18        if(maxp==1)
19        {
20            nsol++;
21            dim=poz-1;
22            afis2(val,maxp);
23            return;
24        }
25        if(val==0)
26        {
27            nsol++;
28            dim=poz-1;
29            afis1();
30            return;
31        }
32
33        int i;
34        int maxok=min(maxp,val);
35        for(i=maxok;i>=1;i--)
36        {
37            x[poz]=i;
38            f(val-i,min(val-i,i),poz+1);
39        }
40    } // f...
41
42    static void afis1()
43    {
44        int i;
45        System.out.print("\n"+nsol+" : ");
46        for(i=1;i<=dim;i++) System.out.print(x[i]+" ");
47    } // afis1()
48
49    static void afis2(int val,int maxip)
50    {
51        int i;
52        System.out.print("\n"+nsol+" : ");
53        for(i=1;i<=dim;i++) System.out.print(x[i]+" ");
54    }

```

```

54     for(i=1;i<=val;i++) System.out.print("1 ");
55 }afis2(...)
56
57 static int min(int a,int b) { return (a<b)?a:b; }
58 }// class

```

Listing 35.4.2: PartitieNrImpare1.java

```

1 class PartitieNrImpare1 // n = suma de numere impare
2 { // nsol = 38328320 timp = 8482
3     static int dim=0,nsol=0;
4     static int[] x=new int[161];
5
6     public static void main(String[] args)
7     {
8         long t1,t2;
9         int n=160;
10        int maxi=((n-1)/2)*2+1;
11        t1=System.currentTimeMillis();
12        f(n,maxi,1);
13        t2=System.currentTimeMillis();
14        System.out.println("nsol = "+nsol+" timp = "+(t2-t1)+"\n");
15    }// main...
16
17    static void f(int val, int maxip, int poz)
18    {
19        if(maxip==1)
20        {
21            nsol++;
22            // dim=poz-1;
23            // afis2(val,maxip);
24            return;
25        }
26        if(val==0)
27        {
28            nsol++;
29            // dim=poz-1;
30            // afis1();
31            return;
32        }
33
34        int maxi=((val-1)/2)*2+1;
35        int maxiok=min(maxip,maxi);
36        int i;
37        for(i=maxiok;i>=1;i=i-2)
38        {
39            x[poz]=i;
40            f(val-i,min(maxiok,i),poz+1);
41        }
42    }// f...
43
44    static void afis1()
45    {
46        int i;
47        System.out.print("\n"+nsol+" : ");
48        for(i=1;i<=dim;i++) System.out.print(x[i]+" ");
49    }// afis1()
50
51    static void afis2(int val,int maxip)
52    {
53        int i;
54        System.out.print("\n"+nsol+" : ");
55        for(i=1;i<=dim;i++) System.out.print(x[i]+" ");
56        for(i=1;i<=val;i++) System.out.print("1 ");
57    }// afis2...
58
59    static int max(int a,int b) { return (a>b)?a:b; }
60
61    static int min(int a,int b) { return (a<b)?a:b; }
62 }// class

```

Listing 35.4.3: PartitieNrImpare2.java

```

1 import java.io.*;

```

```

2  class PartitieNrImpare2    // n = suma de numere impare ;
3  {                           // nsol = 38328320 timp = 4787
4      static int dim=0,nsol=0;
5      static int[] x=new int[161];
6
7      public static void main(String[] args) throws IOException
8      {
9          long t1,t2;
10         t1=System.currentTimeMillis();
11
12         int n,i;
13         StreamTokenizer st=new StreamTokenizer(
14             new BufferedReader(new FileReader("partitie.in")));
15         PrintWriter out=new PrintWriter(new BufferedWriter(
16             new FileWriter("partitie.out")));
17
18         st.nextToken(); n=(int)st.nval;
19
20         int maxi=((n-1)/2)*2+1;
21         f(n,maxi,1);
22
23         out.print(nsol);
24         out.close();
25         t2=System.currentTimeMillis();
26         System.out.println("nsol = "+nsol+" timp= "+(t2-t1));
27     }// main...
28
29     static void f(int val, int maxip, int poz)
30     {
31         if(maxip==1)
32         {
33             nsol++;
34             dim=poz-1;
35             afis2(val);
36             return;
37         }
38         if(val==0)
39         {
40             nsol++;
41             dim=poz-1;
42             //afis1();
43             return;
44         }
45
46         int maxi=((val-1)/2)*2+1;
47         int maxiok=min(maxip,maxi);
48         int i;
49         for(i=maxiok;i>=3;i=i-2)
50         {
51             x[poz]=i;
52             f(val-i,i,poz+1);
53         }
54         nsol++;
55         dim=poz-1;
56         //afis2(val);
57     }// f...
58
59     static void afis1()
60     {
61         int i;
62         System.out.print("\n"+nsol+" : ");
63         for(i=1;i<=dim; i++) System.out.print(x[i]+" ");
64     }// afis1()
65
66     static void afis2(int val)
67     {
68         int i;
69         System.out.print("\n"+nsol+" : ");
70         for(i=1;i<=dim; i++) System.out.print(x[i]+" ");
71         for(i=1;i<=val; i++) System.out.print("1 ");
72     }// afis2...
73
74     static int max(int a,int b) { return (a>b)?a:b; }
75
76     static int min(int a,int b) { return (a<b)?a:b; }
77 } // class

```

35.4.3 Cod sursă

Listing 35.4.4: partitie.pas

```

1 { $A+, B-, D+, E+, F-, G-, I+, L+, N-, O-, P-, Q-, R-, S+, T-, V+, X+ }
2 { $M 16384,0,655360}
3 type solutie=array[0..200] of integer;
4 var fi,fo:text;
5   n,np:word;
6   sol:solutie;
7   i,j,k:integer;
8   p:array[1..200,1..200] of longint;
9   contor:longint;
10
11 function part(n,k:integer):longint;
12 var suma:longint;
13 begin
14   if n<k then part:=0
15   else
16     if (n=k) or (k=1) then part:=1
17     else
18       if p[n,k]<>0 then part:=p[n,k]
19       else
20         begin
21           suma:=0;
22           for i := 1 to k do
23             suma:=suma+part(n-k,i);
24           part:=suma;
25           p[n,k]:=suma;
26         end
27       end;
28
29 begin
30   assign(fi,'partitie.in');
31   reset(fi);
32   readln(fi,n);
33   close(fi);
34   contor:=0;
35   for i:=1 to n do
36     contor:=contor+part(n-i*(i-1) div 2,i);
37   assign(fo,'partitie.out');
38   rewrite(fo);
39   writeln(fo,contor);
40   close(fo);
41 end.
```

35.5 Rubine

Ali-Baba și cei N hoți ai săi au descoperit harta unei comori pe o insulă izolată. Se asimilează harta cu un caroaj de $L * C$ regiuni (L linii și C coloane). În fiecare regiune se află un sipet fermecat, în care se găsesc rubine (0 sau mai multe), numărul de rubine din fiecare sipet fiind păstrat în elementul corespunzător al caroajului.

Se stabilește următorul plan de acțiune:

- deplasarea hoțului se poate face dintr-o regiune într-o altă regiune vecină (o regiune are maxim 8 regiuni vecine);
- fiecare hoț pleacă dintr-o anumită regiune, de coordonate date, spre regiunea în care se află Ali-Baba (regiunea din colțul dreapta jos, de coordonate L și C), pe drumul cel mai scurt (lungimea unui drum fiind egală cu numărul regiunilor parcurse);
- la trecerea printr-o regiune hoțul pună într-un sac rubinele existente în sipetul existent acolo;
- dacă există mai multe drumuri de lungime minimă, un hoț îl va alege pe acela pe care parcurgându-l, reușește să strângă cât mai multe rubine;
- primul hoț care se deplasează este cel cu numărul de ordine 1, urmează apoi cel cu numărul de ordine 2, 3, și a.m.d., iar în timpul deplasării unui hoț, ceilalți stau pe loc până ce acesta ajunge la Ali-Baba;
- sipetul fiind fermecat, în locul rubinelor luate de hoț, apar altele identice, apariția petrecându-se înaintea plecării fiecărui hoț din punctul lui de pornire spre Ali-Baba.

Fiecare hoț a strâns în sacul său un număr de rubine, cunoscut doar de el și ajunge cu sacul în regiunea lui Ali-Baba. La împărțirea rubinelor apare și Sindbad Marinarul care vrea o parte din saci. Ali-Baba cunoscând anumite relații care există între perechi de hoți din regiune va trebui să aleagă acele relații care să implice repartizarea unui număr minim de saci lui Sindbad.

Se știe că alegera unei perechi de hoți (i, j) implică repartizarea sacului hoțului j din "cauza" lui i în visteria lui Ali-Baba, însă din acest moment nu se mai poate folosi nici o pereche de forma (j, k) , hoțul j fiind eliminat.

Cerință: Ajutați-l pe Ali-Baba să aleagă ordinea perechilor astfel încât sacii care rămân să fie într-un număr cât mai mic posibil, știind că aceștia-i revin lui Sindbad.

Date de intrare:

Fișierul de intrare **rubine.in**, conține:

Pe prima linie dimensiunile caroajului: valorile lui L și C separate printr-un spațiu;

Pe următoarele L linii se află câte C valori separate prin spațiu, reprezentând numărul de rubine din sippetul aflat în regiunea respectivă;

Pe următoarea linie urmează N , numărul de hoți;

Pe următoarele N linii se află coordonatele regiunilor (valori separate printr-un spațiu) în care se află inițial hoții, pe prima linie dintre acestea pentru hoțul unu, pe linia a doua pentru hoțul doi, etc;

Pe următoarele linii, până la sfârșitul fișierului se află perechile care reprezintă relațiile din problemă.

Date de ieșire:

Ieșirea se va face în fișierul **rubine.out** în formatul următor:

Pe prima linie se află numărul T de perechi folosite de Ali-Baba, respectându-se cerințele problemei.

Pe următoarele T linii se află câte patru valori separate două câte două printr-un spațiu: $i\ nr1\ j\ nr2$, unde i reprezintă prima valoare din pereche (hoțul cu numărul de ordine i), $nr1$ reprezintă numărul de rubine adunate de hoțul i , j reprezintă al doilea hoț din pereche (hoțul cu numărul de ordine j), $nr2$ reprezentând numărul de rubine strânse de hoțul j .

Pe următoarea linie se află numerele de ordine ale hoților ai căror saci au rămas lui Sindbad, valori separate două câte două printr-un spațiu.

Pe ultima linie se află suma rubinelor din sacii rămași lui Sindbad.

Restricții:

$1 \leq L, C, N \leq 50$

$0 \leq A[i, j] \leq 10$

Observații:

Întotdeauna există soluții, dacă există mai multe, se va alege una.

În fiecare regiune, indiferent dacă este punctul de pornire al unui hoț sau regiunea în care se află Ali-Baba există câte un sippet cu un anumit număr de rubine.

Ali-Baba nu cunoaște numărul de rubine din fiecare sac.

Nu există doi hoți care se află inițial în același regiune.

Exemplu:

rubine.in rubine.out

4 5	4
1 0 0 0 0	2 9 5 6
0 1 0 0 0	1 12 2 9
0 0 1 4 0	1 12 4 11
0 0 0 1 5	3 10 1 12
5	3
1 1	10
1 5	
4 1	
2 2	
4 4	
1 2	
1 4	
2 5	
3 1	
4 5	

Timp maxim de executare: 1 secundă/test

35.5.1 Indicații de rezolvare

Roxana Timplaru

Deplasarea hoților până la Ali-Baba se rezolvă cu *algoritmul Lee*. Se folosește apoi un *Greedy* pentru selectarea perechii (i, j) , j fiind un hoț pentru care nu există perechi (j, k) .

Mihai Stroe, GInfo nr. 13/6 - octombrie 2003

Prima parte a problemei constă în determinarea, pentru fiecare hoț, a numărului maxim de rubine pe care le poate strângă, respectând cerințele enunțului. Acest număr se poate determina, pentru fiecare hoț, plecând din poziția de start a acestuia și folosind un algoritm de complexitate $O(L \cdot C)$.

Puteam optimiza această metodă dacă se pornește din poziția destinație (comună) a tuturor hoților; astfel acest algoritm se va executa o singură dată, obținându-se după executare rezultatul cerut pentru fiecare poziție de start posibilă.

În prima fază a algoritmului (amintit mai sus) se obține pentru fiecare poziție (i, j) numărul minim de mutări $M(i, j)$ până la ajungerea la destinație, pornind din poziția respectivă. Pentru aceasta putem folosi *algoritmul lui Lee*, de complexitate $O(L \cdot C)$.

O examinare mai atentă a numerelor rezultate conduce la un algoritm mult mai simplu, de aceeași complexitate. Astfel, $M[L, C] = 0$, $M[L - 1, C] = M[L - 1, C - 1] = M[L, C - 1] = 1$, ..., $M[L - d_1, C - d_2] = \max(d_1, d_2)$.

Folosind această informație vom calcula, pentru fiecare poziție (i, j) din matrice, numărul maxim de rubine $R(i, j)$ care poate fi strâns pornind spre destinație din poziția respectivă. Astfel, pentru o poziție (i, j) , $R(i, j)$ este maximul dintre R -urile vecinilor care au M -ul mai mic cu 1 decât $M(i, j)$, la care se adaugă cantitatea de rubine din poziția (i, j) .

Pozиїile pot fi parcuse în ordinea numerelor $M(i, j)$; astfel, pentru fiecare poziție, R -urile vecinilor care ne interesează vor fi calculate anterior, deci vor fi disponibile la calculul $R(i, j)$.

Deoarece pentru fiecare poziție din matrice se vor examina cel mult trei vecini, calcularea matricei R are complexitatea $O(L \cdot C)$. Se folosește apoi un *algoritm greedy* pentru selectarea perechii (i, j) , j fiind un hoț pentru care nu există perechi (j, k) .

Ordinul de complexitate al unei implementări pentru acest algoritm este $O(N \cdot N)$; o implementare mai pretențioasă poate avea ordinul de complexitate $O(P)$, unde P este numărul de perechi (i, j) prezente în fișierul de intrare.

Limitele din enunț permit și variante mai puțin optime, cum ar fi cea în care *algoritmul lui Lee* este executat pentru fiecare hoț.

Analiza complexității

Ordinul de complexitate al operației de citire a datelor de intrare este $O(L \cdot C + N + P)$. Deoarece P este limitat superior de $N \cdot N$, ordinul de complexitate al acestei operații devine $O(L \cdot C + N \cdot N)$.

Calcularea cantității de rubine strânse de fiecare hoț are ordinul de complexitate $O(L \cdot C)$.

Alegerea perechilor pentru repartizarea sacilor are ordinul de complexitate $O(P)$ sau $O(N \cdot N)$, în funcție de implementarea aleasă.

Afișarea soluției are ordinul de complexitate $O(N)$. În final, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(L \cdot C + N \cdot N) + O(N \cdot N) + O(N) = O(L \cdot C + N \cdot N)$.

35.5.2 Rezolvare detaliată

Listing 35.5.1: rubine.java

```

1 import java.io.*; // Final ... fara mesaje dar ... daca e circuit ...!!!
2 class Rubine
3 {
4     static final int qdim=200;
5
6     static int m,n;           // dimensiuni insula
7     static int nh;            // nr hoti
8     static int nr;            // nr relati
9     static int nrf;           // nr relati folosite
10
11    static int[][] x=new int[51][51]; // insula
12    static int[][] c=new int[51][51]; // cost maxim (nr rubine...)

```

```

13     static int[][] t=new int[51][51];    // traseu curent (m,n) --> (i,j)
14
15     static int[] ih=new int[51];        // coordonatele hotilor
16     static int[] jh=new int[51];
17     static int[] nrh=new int[51];       // nrh[k]=nr rubine stranse de hotul k
18     static int[] ge=new int[51];        // grad exterior
19     static boolean[] e=new boolean[51]; // eliminat
20
21     static int[] ir=new int[25*51];    // relatii intre hoti
22     static int[] jr=new int[25*51];
23     static int[] rf=new int[25*51];    // relatii folosite
24
25     static int[] qi=new int[qdim];     // coada pentru i din pozitia (i,j)
26     static int[] qj=new int[qdim];     // coada pentru j din pozitia (i,j)
27     static int ic, sc;               // ic=inceput coada, ic=inceput coada
28
29 public static void main(String[] args) throws IOException
30 {
31     int i,j,k,k0;
32     boolean gata;
33
34     citire();
35
36     for(k=1;k<=nh;k++)
37     {
38         matriceCosturi(ih[k],jh[k]);
39         nrh[k]=c[m][n];
40     }
41
42     for(k=1;k<=nh;k++) e[k]=false;
43     k0=nr+1;
44     nrf=0;
45     gata=false;
46     while(!gata)
47     {
48         gata=true;
49         for(j=1;j<=nh;j++)
50             if(!e[j]&&ge[j]==0)
51             {
52                 // caut prima relatie i --> j cu ge[j]==0
53                 for(k=1;k<=nr;k++)
54                     if(jr[k]==j)
55                     {
56                         nrf++;
57                         i=ir[k]; rf[nrf]=k; e[j]=true; k0=k;
58                         gata=false;
59                         break;
60                     }
61                 // micsorez gradele exterioare pentru toti i cu i --> j
62                 for(k=k0;k<=nr;k++) if(jr[k]==j) ge[ir[k]]--;
63             } // if
64     } // while(!gata)
65     afisSolutia();
66 } // main()
67
68 static void citire() throws IOException
69 {
70     int i,j,k;
71     StreamTokenizer st=new StreamTokenizer(
72         new BufferedReader(new FileReader("rubine.in")));
73
74     st.nextToken(); m=(int)st.nval;
75     st.nextToken(); n=(int)st.nval;
76     for(i=1;i<=m;i++)
77         for(j=1;j<=n;j++) { st.nextToken(); x[i][j]=(int)st.nval; }
78
79     st.nextToken(); nh=(int)st.nval;
80     for(k=1;k<=nh;k++)
81     {
82         st.nextToken(); ih[k]=(int)st.nval;
83         st.nextToken(); jh[k]=(int)st.nval;
84     }
85
86     nr=0;
87     while(st.nextToken()!=StreamTokenizer.TT_EOF)
88     {

```

```

89         nr++;
90         ir[nr]=(int)st.nval;
91         st.nextToken(); jr[nr]=(int)st.nval;
92         ge[ir[nr]]++;
93     }
94 } // citire()
95
96 static void matriceCosturi(int i0, int j0)
97 {
98     int i,j;
99     for(i=1;i<=m;i++)
100        for(j=1;j<=n;j++) {t[i][j]=0; c[i][j]=0;}
101
102    c[i0][j0]=x[i0][j0];
103    ic=sc=0;
104    qi[sc]=i0; qj[sc]=j0; sc=(sc+1)%qdim;
105    t[i0][j0]=1;
106    while(ic!=sc)
107    {
108        i=qi[ic]; j=qj[ic]; ic=(ic+1)%qdim;
109        vecini(i,j);
110    }
111 } //matriceCosturi()
112
113 static void vecini(int i, int j)
114 {
115     int tt=t[i][j];
116
117     if((i-1>=1)&&(j+1<=n)) // NE
118     {
119         if(t[i-1][j+1]==0)
120         {
121             c[i-1][j+1]=c[i][j]+x[i-1][j+1];
122             t[i-1][j+1]=tt+1; qi[sc]=i-1; qj[sc]=j+1; sc=(sc+1)%qdim;
123         }
124         else
125             if(t[i-1][j+1]==t[i][j]+1)
126                 c[i-1][j+1]=max(c[i-1][j+1],c[i][j]+x[i-1][j+1]);
127     }
128
129     if(i+1<=m) // S
130     {
131         if(t[i+1][j]==0)
132         {
133             c[i+1][j]=c[i][j]+x[i+1][j];
134             t[i+1][j]=tt+1; qi[sc]=i+1; qj[sc]=j; sc=(sc+1)%qdim;
135         }
136         else
137             if(t[i+1][j]==t[i][j]+1) c[i+1][j]=max(c[i+1][j],c[i][j]+x[i+1][j]);
138     }
139
140     if((i+1<=m)&&(j+1<=n)) // SE
141     {
142         if(t[i+1][j+1]==0)
143         {
144             c[i+1][j+1]=c[i][j]+x[i+1][j+1];
145             t[i+1][j+1]=tt+1; qi[sc]=i+1; qj[sc]=j+1; sc=(sc+1)%qdim;
146         }
147         else
148             if(t[i+1][j+1]==t[i][j]+1)
149                 c[i+1][j+1]=max(c[i+1][j+1],c[i][j]+x[i+1][j+1]);
150     }
151
152     if((i+1<=m)&&(j-1>=1)) // SV
153     {
154         if(t[i+1][j-1]==0)
155         {
156             c[i+1][j-1]=c[i][j]+x[i+1][j-1];
157             t[i+1][j-1]=tt+1; qi[sc]=i+1; qj[sc]=j-1; sc=(sc+1)%qdim;
158         }
159         else
160             if(t[i+1][j-1]==t[i][j]+1)
161                 c[i+1][j-1]=max(c[i+1][j-1],c[i][j]+x[i+1][j-1]);
162     }
163
164     if(j+1<=n) // E

```

```

165      {
166          if(t[i][j+1]==0)
167          {
168              c[i][j+1]=c[i][j]+x[i][j+1];
169              t[i][j+1]=tt+1; qj[sc]=i; qj[sc]=j+1; sc=(sc+1)%qdim;
170          }
171          else
172              if(t[i][j+1]==t[i][j]+1) c[i][j+1]=max(c[i][j+1],c[i][j]+x[i][j+1]);
173          }
174      }// vecini...
175
176  static void afisSolutia() throws IOException
177  {
178      int k,s=0;
179      PrintWriter out = new PrintWriter(
180          new BufferedWriter(new FileWriter("rubine.out")));
181
182      out.println(nrf);
183      for(k=1;k<=nrf;k++)
184          out.println(ir[rf[k]]+" "+nrh[ir[rf[k]]]+" "+jr[rf[k]]+" "+nrh[jr[rf[k]]]);
185      for(k=1;k<=nh;k++) if(!e[k]) {out.print(k+" "); s+=nrh[k];}
186      out.println();
187      out.println(s);
188      out.close();
189  } //afisSolutia()
190
191  static int max(int a, int b)
192  {
193      if(a>b) return a; else return b;
194  } // max...
195 } // class

```

35.5.3 Cod sursă

Listing 35.5.2: rubine.pas

```

1 {$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R+,S+,T-,V+,X+}
2 {$M 16384,0,655360}
3
4 const dl:array[1..8] of -1..1=(-1,-1,0,1,1,1,0,-1);
5     dc:array[1..8] of -1..1=(0,1,1,1,0,-1,-1,-1);
6
7 type vector1=array[1..50] of integer;
8     matrice1=array[1..50,1..50] of integer;
9     celula=record
10        nr,l,c:0..50;
11        suma:integer;
12    end;
13    matrice2=array[1..50,1..50] of celula;
14    matrice3=array[1..50,1..50] of 0..1;
15
16 var m,n,nrh:1..50;
17     a:matrice1;
18     b:matrice2;
19     x,y,rubine:vector1;
20     c:matrice3;
21
22 procedure date;
23 var f:text;
24     i,j:integer;
25 begin
26     assign(f,'rubine.in');
27     reset(f);
28     readln(f,m,n);
29     for i:=1 to m do
30     begin
31         for j:=1 to n do
32             read(f,a[i,j]);
33             readln(f);
34         end;
35     readln(f,nrh);
36     for i:=1 to nrh do

```

```

37   readln(f,x[i],y[i]);
38   while not eof(f) do
39   begin
40     readln(f,i,j);
41     c[i,j]:=1;
42   end;
43   close(f);
44 end;
45
46 procedure calcul(l,c:integer);
47 var marcaj:boolean;
48   i,j,lnou,cnou:0..51;
49   k:1..2500;
50   t:1..8;
51 begin
52   b[l,c].nr:=1;b[l,c].suma:=a[l,c];
53   k:=1;
54   marcaj:=true;
55   while (marcaj) and (b[m,n].nr=0) do
56   begin
57     marcaj:=false;
58     for i:=1 to m do
59       for j:=1 to n do
60         if b[i,j].nr=k then
61           for t:=1 to 8 do
62             begin
63               lnou:=i+dl[t];
64               cnou:=j+dc[t];
65               if (lnou>0) and (lnou<=m) and (cnou>0) and (cnou<=n) then
66                 if (b[lnou,cnou].nr=0) or (b[lnou,cnou].nr=k+1) and
67                   (b[lnou,cnou].suma< b[i,j].suma+a[lnou,cnou]) then
68                   begin
69                     b[lnou,cnou].nr:=k+1;
70                     b[lnou,cnou].suma:=b[i,j].suma+a[lnou,cnou];
71                     b[lnou,cnou].l:=i;
72                     b[lnou,cnou].c:=j;
73                     marcaj:=true;
74                   end
75                 end;
76               k:=k+1;
77             end;
78   end;
79
80 procedure drum(l,c:integer);
81 begin
82   if (b[l,c].l<>0) and (b[l,c].c<>0) then begin
83     a[l,c]:=0;
84     drum(b[l,c].l,b[l,c].c);
85   end;
86 end;
87
88 procedure ordine;
89 var ramasi,r:set of byte;
90   g:text;
91   nr,i,j,l,t:integer;
92   ok,gata:boolean;
93   s:longint;
94 begin
95   nr:=0;
96   r:=[];
97   s:=0;
98   assign(g,'rubine.out');
99   rewrite(g);
100  ramasi:=[1..nrh];
101  repeat
102    i:=0;
103    repeat
104      i:=i+1;
105      ok:=false;
106      if i in ramasi then
107        begin
108          ok:=true;
109          for j:=1 to nrh do
110            if c[i,j]=1 then ok:=false;
111        end;
112      until ok;

```

```

113  if ok then
114    begin
115      gata:=true;
116      for j:=1 to nrh do
117        if c[j,i]=1 then
118          begin
119            inc(nr);
120            x[nr]:=j;
121            y[nr]:=i;
122          {      writeln(g,j,' ',rubine[j],' ',i,' ',rubine[i],' ');
123            ramasi:=ramasi-[i];
124            gata:=false;
125            for l:=1 to nrh do
126              c[l,i]:=0;
127              j:=nrh;
128            end;
129            if gata then
130              begin
131                r:=r+[i];
132                ramasi:=ramasi-[i];
133              end;
134            end;
135        until ramasi=[];
136        writeln(g,nr);
137        for i:=1 to nr do
138          writeln(g,x[i],' ',rubine[x[i]],' ',y[i],' ',rubine[y[i]],' ');
139        s:=0;
140        for i:=1 to nrh do
141          if i in r then
142            begin
143              s:=s+rubine[i];
144              write(g,i,' ');
145            end;
146        writeln(g);
147        writeln(g,s);
148        close(g);
149      end;
150
151 procedure deplasari;
152 var i,l,j:integer;
153 begin
154   for i:=1 to nrh do
155     begin
156       for l:=1 to m do
157         for j:=1 to n do
158           begin
159             b[l,j].nr:=0;
160             b[l,j].suma:=0;
161             b[l,j].l:=0;
162             b[l,j].c:=0;
163           end;
164           calcul(x[i],y[i]);
165           rubine[i]:=b[m,n].suma;
166           { a[x[i],y[i]]:=0;
167             drum(b[m,n].l,b[m,n].c);
168             a[m,n]:=0; }
169           end;
170         end;
171       begin
172         date;
173         deplasari;
174         ordine;
175       end.

```

35.6 Scufița

Majoritatea participanților la ONI2003 au auzit, în copilărie, povestea Scufiței Roșii. Pentru cei care o știu, urmează partea a două; pentru cei care nu o știu, nu vă faceți griji, cunoasterea ei nu este necesară pentru a rezolva această problemă. Povestea nu spune ce s-a întâmplat pe drumul pe care Scufița Roșie s-a întors de la bunicuță. Veți afla amănunte în continuare.

Pe drum, ea s-a întâlnit cu Lupul (fratele lupului care a părăsit povestea în prima parte). Acesta dorea să o mănânce, dar a decis să-i acorde o sansă de scăpare, provocând-o la un concurs de cules ciupercuțe.

Scufița Roșie se află în poziția $(1, 1)$ a unei matrice cu N linii și N coloane, în fiecare poziție a matricei fiind amplasate ciupercuțe. Lupul se află în poziția $(1, 1)$ a unei alte matrice similare. Pe parcursul unui minut, atât Scufița, cât și Lupul se deplasează într-o din pozițiile vecine (pe linie sau pe coloană) și culeg ciupercuțele din poziția respectivă. Dacă Scufița Roșie ajunge într-o poziție în care nu sunt ciupercuțe, va pierde jocul. Dacă la sfârșitul unui minut ea are mai puține ciupercuțe decât Lupul, ea va pierde jocul de asemenea. Jocul începe după ce amândoi participanții au culeș ciupercuțele din pozițiile lor inițiale (nu contează cine are mai multe la începutul jocului, ci doar după un număr întreg strict pozitiv de minute de la început). Dacă Scufița Roșie pierde jocul, Lupul o va mâncă.

Înainte de începerea jocului, Scufița Roșie l-a sunat pe Vâنător, care i-a promis că va veni într-un sfert de ora (15 minute) pentru a o salva. Deci Scufița Roșie va fi liberă să plece dacă nu va pierde jocul după 15 minute.

Din acest moment, scopul ei este nu numai să rămână în viață, ci și să culeagă cât mai multe ciupercuțe, pentru a le duce acasă (după ce va veni, vânătorul nu o va mai lăsa să culeagă).

Lupul, cunoscut pentru lăcomia sa proverbială, va alege la fiecare minut mutarea în câmpul vecin cu cele mai multe ciupercuțe (matricea sa este dată astfel încât să nu existe mai multe posibilități de alegere la un moment dat).

Povestea spune că Scufița Roșie a plecat acasă cu coșulețul plin de ciupercuțe, folosind indicațiile date de un program scris de un concurent la ONI 2003 (nu vom da detalii suplimentare despre alte aspecte, cum ar fi călătoria în timp, pentru a nu complica inutil enunțul problemei). Să fi fost acest program scris de către dumneavoastră? Vom vedea...

Cerință

Scrieți un program care să o ajute pe Scufița Roșie să rămână în joc și să culeagă cât mai multe ciupercuțe la sfârșitul celor 15 minute!

Date de intrare

Fișierul **scufita.in** are următoarea structură:

N - dimensiunea celor două matrice

$a_{11}a_{12}\dots a_{1n}$ - matricea din care culege Scufița Roșie

$a_{21}a_{22}\dots a_{2n}$

...

$a_{n1}a_{n2}\dots a_{nn}$

$b_{11}b_{12}\dots b_{1n}$ - matricea din care culege Lupul

$b_{21}b_{22}\dots b_{2n}$

...

$b_{n1}b_{n2}\dots b_{nn}$

Date de ieșire

Fișierul **scufita.out** are următoarea structură:

NR - numărul total de ciupercuțe culese

$d_1d_2\dots d_{15}$ - direcțiile pe care s-a deplasat Scufița Roșie, separate prin câte un spațiu (direcțiile pot fi N, E, S, V indicând deplasări spre Nord, Est, Sud, Vest; poziția $(1, 1)$ este situată în colțul de Nord-Vest al matricei)

Restricții

- $4 \leq N \leq 10$;
- valorile din ambele matrice sunt numere naturale mai mici decât 256;
- nici Scufița Roșie și nici Lupul nu vor părăsi matricele corespunzătoare;
- după ce unul din jucători culege ciupercuțele dintr-o poziție, în poziția respectivă rămân 0 ciupercuțe;
- pe teste date, Scufița Roșie va avea întotdeauna posibilitatea de a rezista 15 minute.

Exemplu

```

scufita.in    scufita.out
4            137
2 2 3 4      SSSEEEENVVNEENVV
5 6 7 8
9 10 11 12
13 14 15 16
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

```

Explicație:

Scufița Roșie a efectuat aceleasi mutări cu cele efectuate de Lup și a avut tot timpul o ciuperță în plus. În final ea a cules toate ciupercuțele din matrice.

Timp maxim de executare: 1 secundă/test

35.6.1 Indicații de rezolvare

Mihai Stroe, GInfo nr. 13/6 - octombrie 2003

Problema se rezolvă folosind *metoda backtracking în plan*.

Se observă că mutările *Lupului* sunt independente de mutările *Scufiței*, astfel ele pot fi determinate imediat după citirea datelor.

În acest punct al rezolvării, *Scufița* trebuie să mute la fiecare moment, astfel încât să rămână în joc, iar în final să strângă cât mai multe ciupercuțe. Restricțiile sunt date de enunțul problemei și de numărul de ciupercuțe culese de *Lup*, care este cunoscut la fiecare moment.

Rezolvarea prin *metoda backtracking* încearcă la fiecare pas, pe rând, fiecare mutare din cele cel mult patru posibile (teoretic; de fapt, cel mult trei mutări sunt posibile în fiecare moment, deoarece Scufița nu se va întoarce în poziția din care tocmai a venit).

Se urmărește respectarea restricțiilor impuse. În momentul găsirii unei soluții, aceasta este comparată cu soluția optimă găsită până atunci și dacă este mai bună este reținută.

Încă nu a fost găsită o rezolvare polinomială pentru această problemă (și este improbabil ca o astfel de rezolvare să existe), dar limitele mici și faptul că numărul de mutări disponibile începe să scadă destul de repede, în multe cazuri permit un timp de execuție foarte bun.

Analiza complexității

Fie M numărul de mutări pe care le are de efectuat Scufița.

Operațiile de citire și scriere a datelor au ordinul de complexitate $O(N^2)$, respectiv $O(M)$, deci nu vor fi luate în calcul. Ordinul de complexitate este dat de rezolvarea prin *metoda backtracking*.

Având în vedere că la fiecare pas *Scufița* poate alege dintre cel mult 3 mutări (deoarece nu se poate întoarce în poziția din care a venit), ordinul de complexitate ar fi $O(3^M)$. De fapt, numărul maxim de stări examineate este mult mai mic; de exemplu, primele două mutări oferă două variante de alegere în loc de trei. Alte restricții apar datorită limitării la o matrice de 10×10 elemente.

Cum numărul M este cunoscut și mic, s-ar putea considera că ordinul de complexitate este limitat superior, deci constant (constanta introdusă fiind totuși destul de mare).

35.6.2 Rezolvare detaliată

Listing 35.6.1: scufita.java

```

1 import java.io.*;
2 class Scufita
3 {
4     static int n, maxc=0;
5     static int[][] a,b;
6     static int[] x,y;
7     static char[] tc=new char[16];
8     static char[] tmax=new char[16];
9
10    public static void main(String[] args) throws IOException
11    {
12        int i,j;
13        StreamTokenizer st=new StreamTokenizer(
14            new BufferedReader(new FileReader("scufita.in")));

```

```

15     PrintWriter out=new PrintWriter(
16         new BufferedWriter(new FileWriter("scufita.out")));
17
18     st.nextToken(); n=(int)st.nval;
19
20     a=new int[n+2][n+2];
21     b=new int[n+2][n+2];
22     x=new int[17];
23     y=new int[17];
24
25     for(i=1;i<=n;i++)
26         for(j=1;j<=n;j++) { st.nextToken(); a[i][j]=(int)st.nval; }
27     for(i=1;i<=n;i++)
28         for(j=1;j<=n;j++) { st.nextToken(); b[i][j]=(int)st.nval; }
29
30     culegeLupul(1,1,1);
31     f(1,1,1);
32
33     out.println(maxc);
34     for(i=1;i<=15;i++) out.print(tmax[i]);
35     out.println();
36     out.close();
37 } // main()
38
39 static void f(int i, int j, int k) throws IOException
40 {
41     int aij=a[i][j];
42
43     x[k]=x[k-1]+a[i][j];
44     a[i][j]=0;
45
46     if(k==16)
47     {
48         if(x[16]>maxc)
49         {
50             maxc=x[16];
51             for(int ii=1;ii<=15;ii++) tmax[ii]=tc[ii];
52         }
53         a[i][j]=aij;
54         return;
55     }
56
57     if((a[i-1][j]>0)&&(a[i-1][j]+x[k]>=y[k+1])) { tc[k] ='N'; f(i-1,j,k+1); }
58     if((a[i+1][j]>0)&&(a[i+1][j]+x[k]>=y[k+1])) { tc[k] ='S'; f(i+1,j,k+1); }
59     if((a[i][j-1]>0)&&(a[i][j-1]+x[k]>=y[k+1])) { tc[k] ='V'; f(i,j-1,k+1); }
60     if((a[i][j+1]>0)&&(a[i][j+1]+x[k]>=y[k+1])) { tc[k] ='E'; f(i,j+1,k+1); }
61
62     a[i][j]=aij;
63 } // f(...)
64
65 static void culegeLupul(int i, int j, int k)
66 {
67     if(k>16) return;
68     y[k]=y[k-1]+b[i][j];
69     b[i][j]=0;
70     if((b[i-1][j]>b[i+1][j])&&(b[i-1][j]>b[i][j-1])&&(b[i-1][j]>b[i][j+1]))
71         culegeLupul(i-1,j,k+1);
72     else if((b[i+1][j]>b[i][j-1])&&(b[i+1][j]>b[i][j+1]))
73         culegeLupul(i+1,j,k+1);
74     else if(b[i][j-1]>b[i][j+1])
75         culegeLupul(i,j-1,k+1);
76     else
77         culegeLupul(i,j+1,k+1);
78 } // culegeLupul(...)
79 } // class

```

35.6.3 Cod sursă

Listing 35.6.2: scufita.pas

```

1 {$A+,B-,D+,E-,G-,I+,L+,N-,O-,P-,Q-,R-,S-,T-,V+,X+,Y+}
2 {$M 16384,0,655360}
3 const dx:array[1..4]of longint=(-1,0,1,0);

```

```

4  const dy:array[1..4]of longint=(0,1,0,-1);
5  const mut:string='NESV';
6
7  var a,b,x:array[0..11,0..11]of longint;
8      optim,ii,jj,xl,yl,max,mp,i,j,k,l,m,n:longint;
9      fi,fo:text;
10     st,sol,lup,sumalup:array[0..100]of longint;
11
12 procedure readdata;
13 begin
14     assign(fi,'scufita.in');
15     assign(fo,'scufita.out');
16     reset(fi);
17     readln(fi,n);
18     for i:=1 to n do
19         begin
20             for j:=1 to n do
21                 read(fi,a[i,j]);
22             readln(fi);
23         end;
24     for i:=1 to n do
25         begin
26             for j:=1 to n do
27                 read(fi,b[i,j]);
28             readln(fi);
29         end;
30     close(fi);
31 end;
32
33 procedure mutaLupul;
34 begin
35     xl:=1;
36     yl:=1;
37     sumalup[0]:=b[1,1];
38     for i:=1 to 15 do
39         begin
40             b[xl,yl]:=0;
41             max:=b[xl+dx[1],yl+dy[1]];
42             mp:=1;
43             for j:=2 to 4 do
44                 if b[xl+dx[j],yl+dy[j]]>max then
45                     begin
46                         max:=b[xl+dx[j],yl+dy[j]];
47                         mp:=j;
48                     end;
49             for j:=mp+1 to 4 do
50                 if b[xl+dx[j],yl+dy[j]]=max then
51                     begin
52                         writeln('Test eronat: lupul nu are drum unic');
53                         exit;
54                     end;
55             sumalup[i]:=sumalup[i-1]+max;
56             xl:=xl+dx[mp];
57             yl:=yl+dy[mp];
58         end;
59     end;
60
61 procedure react(cost:longint);
62 begin
63     if cost>optim then
64         begin
65             optim:=cost;
66             sol:=st;
67         end;
68     end;
69
70
71 procedure back(k,i,j,suma:longint);
72 var d:integer;
73 begin
74     inc(m);
75     if suma<sumalup[k] then
76         exit;
77
78     if k=15 then

```

```

80      begin
81          react(suma);
82          exit;
83      end;
84
85      if suma<sumalup[k] then
86          exit;
87
88      for d:=1 to 4 do
89          begin
90              ii:=i+dx[d];
91              jj:=j+dy[d];
92              if x[ii,jj]=0 then
93                  begin
94                      x[ii,jj]:=k+2;
95                      st[k]:=d;
96                      back(k+1,ii,jj,suma+a[ii,jj]);
97                      x[i+dx[d],j+dy[d]]:=0;
98                  end;
99
100
101
102      end;
103
104 end;
105
106 procedure solve;
107 begin
108     mutalupul;
109     optim:=-1;
110
111     for i:=0 to n+1 do
112         for j:=0 to n+1 do
113             if a[i,j]=0 then
114                 x[i,j]:=1;
115     x[1,1]:=1;
116     back(0,1,1,a[1,1]);
117
118     if optim=-1 then
119         writeln('Nu exista solutie')
120     else
121         begin
122             rewrite(fo);
123             writeln(fo,optim);
124             for i:=0 to 14 do
125                 write(fo,mut[sol[i]]);
126             writeln(fo);
127             close(fo);
128         end;
129     end;
130
131 begin
132     readdata;
133     solve;
134 end.

```

Capitolul 36

ONI 2002



Figura 36.1: Sigla ONI 2002

36.1 Hotel

prof. Doru Popescu Anastasiu, Slatina

Într-un hotel există n angajați în Departamentul Administrativ. Patronul hotelului hotărâște să schimbe costumele personalului din acest departament astfel încât angajații care lucrează la etaje diferite să fie îmbrăcați în haine colorate diferite, iar cei care lucrează la același etaj să fie îmbrăcați în haine colorate la fel. Angajații au fiecare un cod unic dat printr-un număr natural format din maxim 4 cifre.

Cerință

Să se determine o modalitate de alegere a culorilor costumelor care să respecte condițiile de mai sus, precum și numărul de modalități.

Date de intrare

Fisierul de intrare HOTEL.IN are urmatoarea structură:

pe prima linie se află două numere naturale, n și k separate printr-un spațiu (k este numarul de culori).

pe urmatoarele n linii se află câte două numere naturale separate printr-un spațiu, primul fiind codul, iar al doilea etajul asociat angajatului.

Date de ieșire

Prima linie a fișierului de ieșire HOTEL.OUT va conține numărul de modalități.

Știind că o culoare este codificată printr-un număr natural nenul mai mic sau egal cu k , în fișier se va scrie pe câte o linie (începând cu a doua) codul unei persoane și culoarea costumului, valori separate prin câte un spațiu. Ordinea de scriere în fișierul de ieșire va fi aceeași cu cea din fișierul de intrare.

Restrictii

$1 \leq n \leq 1000$

Numarul de etaje din hotel ≤ 200

$1 \leq k \leq 200$.

Observații:

Dacă există mai multe soluții se va afișa una singura.

Dacă nu există soluții, în fișier se va scrie o singura linie care va conține numarul 0.

Numărul de modalități și modalitatea de alegere a culorii costumelor se punctează separat (70% din punctaj pentru numărul de modalități și 30% pentru alegerea corectă a culorilor costumelor)

Exemple

HOTEL.IN	HOTEL.OUT	HOTEL.IN	HOTEL.OUT
4 5	60	5 2	0
123 2	123 1	12 1	
35 1	35 2	13 0	
430 2	430 1	14 1	
13 0	13 3	10 2	
		11 0	

Timp de execuțare/test: 1 secundă.

36.1.1 Indicații de rezolvare

Soluția oficială a comisiei

Pentru determinarea numărului de modalități de alegere a culorilor costumelor care respectă proprietățile din enunț se calculează numărul p de etaje la care lucrează angajații din hotel, după care se calculează numărul: aranjamente de k luate câte p , dacă $p \leq k$, altfel avem 0 modalități.

Numărul căutat fiind mare trebuie folosite *operații cu numere mari*.

O modalitate de determinare a culorii angajaților din hotel constă în ordonarea angajaților după etajul la care lucrează și construirea culorii necesare care respectă condițiile problemei.

Soluție prezentată în GInfo 12/6 - octombrie 2002

Pentru început, vom determina numărul de etaje la care există angajați și vom atribui fiecărui etaj câte o culoare. Pe măsură ce citim datele referitoare la angajați, vom verifica dacă etajul la care lucrează angajatul are asociată o culoare și, dacă este cazul, îi vom atribui o culoare și vom crește numărul etajelor distincte.

Dacă ajungem în situația în care numărul culorilor disponibile este mai mic decât numărul etajelor distincte, atunci problema nu are soluție.

Atribuind câte o culoare fiecărui etaj, practic, am rezolvat cea de-a doua parte a problemei.

Pentru prima parte vom considera că avem la dispoziție n culori și există k etaje distincte la care lucrează angajați. Va trebui să determinăm numărul de posibilități de a atribui celor k etaje câte o culoare astfel încât fiecare etaj să aibă propria sa culoare.

Practic, având la dispoziție o mulțime formată din n elemente va trebui să determinăm câte posibilități de alegere a k dintre aceste elemente există. Este foarte ușor de observat că, de fapt, propoziția anterioară reprezintă definiția noțiunii matematice de *aranjamente*. Așadar, soluția primei părți a problemei este dată de formula:

$$A_n^k = \frac{n!}{(n-k)!} = (n-k+1) \cdot (n-k+2) \cdot \dots \cdot (n-1) \cdot n.$$

Datorită faptului că se obțin numere foarte mari, nu se pot folosi tipurile de date puse la dispoziție de limbajele de programare pentru operații aritmetice. Se observă că, folosind formula anterioară, avem nevoie doar de înmulțiri dintre un număr mare și un număr cel mult egal cu 200. Ca urmare, va trebui să implementăm operația de înmulțire a unui număr mare cu un scalar. Pentru a mări viteza de execuție a programului și a utiliza mai eficient memoria, se poate folosi baza 10000 pentru efectuarea calculelor.

Analiza complexității

Citirea datelor de intrare se realizează în timp liniar, deci ordinul de complexitate al acestei operații este $O(n)$.

Verificarea faptului că unui etaj îi corespunde o culoare și eventuala atribuire a unei culori se realizează în timp constant. Deoarece această operație se realizează pentru etajul corespunzător fiecarui angajat, stabilirea corespondenței dintre etaje și culori se realizează într-un timp cu ordinul de complexitate $O(n) \cdot O(1) = O(n)$.

Calcularea numărului de modalități de alegere a culorilor implică *folosirea numerelor mari*. Se observă că numărul de cifre (în baza 10000) al rezultatului este mai mic decât 100, așadar putem considera că o înmulțire se realizează în timp liniar. Numărul înmulțirilor efectuate este k , deci ordinul de complexitate al operației de determinare a rezultatului este $O(k)$.

Afișarea numărului de modalități este realizată în timp constant, iar operația de afișare a culorilor corespunzătoare fiecarui angajat are ordinul de complexitate $O(n)$.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(n) + O(n) + O(k) + O(1) + O(n) = O(k + n)$.

36.1.2 Rezolvare detaliată

Listing 36.1.1: hotel.java

```

1 import java.io.*;
2 class Hotel
3 {
4     static int[] cod=new int[1000]; // cod angajat
5     static int[] e=new int[1000]; // etaj angajat
6     static int[] c=new int[201]; // culoare etaj
7     static int na; // nr angajati
8     static int nc; // nr culori
9     static int ne; // nr etaje
10    static int[] p=new int[101]; // nr posib. in baza 10000, p[0]=nr cifre
11    static int[] np; // nr posibilitati in baza 10
12    static boolean ok;
13
14    public static void main(String[] args) throws IOException
15    {
16        int i;
17        StreamTokenizer st=new StreamTokenizer(
18            new BufferedReader(new FileReader("hotel2.in")));
19        st.nextToken(); na=(int)st.nval;
20        st.nextToken(); nc=(int)st.nval;
21
22        for(i=1;i<=na;i++)
23        {
24            st.nextToken(); cod[i]=(int)st.nval;
25            st.nextToken(); e[i]=(int)st.nval;
26            if(c[e[i]]==0) c[e[i]]+=ne; // este etaj "nou" aparut!
27        }
28
29        ok=true;
30        if(nc>=ne) np=Aranjamente(nc,ne); else ok=false;
31        PrintWriter out=new PrintWriter(
32            new BufferedWriter(new FileWriter("hotel.out")));
33        if(!ok) out.println(0);
34        else
35        {
36            np=formatBaza10(p);
37            for(i=np.length-1;i>=0;i--) out.print(np[i]);
38            out.println();
39            for(i=1;i<=na;i++) out.println(cod[i]+" "+c[e[i]]);
40        }
41        out.close();
42    } // main...
43
44    static int[] formatBaza10(int[] x) // x_(10000) --> y_(10)
45    {
46        int[] y;
47        int ncy,nr,i,k;
48
49        nr=x[x[0]]; ncy=0; // nr cifre din ultimul grup (cel semnificativ)
50        while(nr!=0) {ncy++; nr/=10;}
51
52        ncy+=4*(x[0]-1); // nr cifre din y (in baza 10)
53        y=new int[ncy];
54        for(k=1;k<=x[0];k++)
55        {
56            nr=x[k];
57            i=4*(k-1);
58            while(nr!=0) {y[i]=nr%10; nr/=10; i++;}
59        }
60        return y;
61    } // formatBaza10...
62
63    static int[] Aranjamente(int n, int k)//A(n,k)=(n-k+1)*(n-k+2)*...*n
64    {
65        int i;
66        p[0]=p[1]=1;
```

```

67     for(i=n-k+1;i<=n;i++) produs(p,i);
68     np=formatBaza10(p);
69     return np;
70 } // Aranjamente(...)
71
72 static void produs(int[] x, int k)
73 {
74     int i,cifra,t=0;
75     for(i=1;i<=x[0];i++)
76     {
77         cifra=(k*x[i]+t)%10000;
78         t=(k*x[i]+t)/10000;
79         x[i]=cifra;
80     }
81     while(t!=0) { x[++x[0]]=t%10000; t/=10000; } // mareste nr cifrelor ...
82 } // produs(...)
83 } // class

```

36.1.3 Cod sursă

Listing 36.1.2: hotel.pas

```

1 program hotel;
2 {D.P.A.}
3 type vectT=array[1..1000]of integer;
4   vec=array[1..700]of integer;
5
6 var c,e,x,o:vectT;
7   n,k:integer;
8   f:text;
9
10 procedure sumal(x:vec;dx:integer;y:vec;dy:integer;var z:vec;var dz:integer);
11 var i,mi,dt,tr:integer;t:vec;
12 begin
13 if dx>dy then
14 begin
15   for i:=1 to dx-dy do
16     t[i]:=0;
17   dt:=dx-dy;
18   for i:=1 to dy do
19     begin
20       inc(dt);
21       t[dt]:=y[i];
22     end;
23   for i:=1 to dt do
24     y[i]:=t[i];
25   dy:=dx;
26 end
27   else
28 if dy>dx then
29 begin
30   for i:=1 to dy-dx do
31     t[i]:=0;
32   dt:=dy-dx;
33   for i:=1 to dx do
34     begin
35       inc(dt);
36       t[dt]:=x[i];
37     end;
38   for i:=1 to dt do
39     x[i]:=t[i];
40   dx:=dy;
41 end;
42 dt:=0;
43 mi:=0;
44 for i:=dx downto 1 do
45 begin
46   inc(dt);
47   tr:=x[i]+y[i]+mi;
48   t[dt]:=tr mod 10;
49   mi:=tr div 10;
50 end;

```

```

51  if mi>0 then
52    begin
53      inc(dt);
54      t[dt]:=mi;
55    end;
56    for i:=1 to dt do
57      z[i]:=t[dt+1-i];
58    dz:=dt;
59  end;{suma}
60
61 procedure produs1(x:vec;dx:integer;y:vec;dy:integer;var z:vec;var dz:integer);
62 var i,j,k,dt,du:integer;t,u:vec;
63 begin
64   dz:=1;z[1]:=0;
65   for i:=1 to dy do
66     begin
67       t[1]:=0;dt:=1;
68       for j:=1 to y[dy-i+1] do
69         begin
70           sumal(x,dx,t,dt,u,du);
71           dt:=du;
72           for k:=1 to du do t[k]:=u[k];
73         end;
74       for j:=1 to i-1 do
75         begin
76           inc(dt);
77           t[dt]:=0;
78         end;
79       sumal(z,dz,t,dt,u,du);
80       dz:=du;
81       for j:=1 to du do z[j]:=u[j];
82     end;
83   end;{produs}
84
85 procedure readdata;
86 var f:text;
87   i:integer;
88 begin
89   assign(f,'T1');reset(f);
90   readln(f,n,k);
91   for i:=1 to n do
92     begin
93       readln(f,c[i],e[i]);
94       o[i]:=i;
95     end;
96   close(f);
97 end;
98
99 procedure sort(lo,hi:integer);
100 var k:integer;
101
102 procedure pos(lo,hi:integer;var k:integer);
103 var i,j:integer;aux:longint;
104 begin
105   i:=lo;j:=hi;
106   while (i<j) do
107     begin
108       while ((i<j) and (e[o[i]]<=e[o[j]])) do inc(i);
109       while ((i<j) and (e[o[i]]>=e[o[j]])) do dec(j);
110       aux:=o[i];o[i]:=o[j];o[j]:=aux;
111     end;
112   k:=i;
113 end;
114
115 begin
116   if lo<hi then
117     begin
118       pos(lo,hi,k);
119       sort(lo,k-1);
120       sort(k+1,hi);
121     end;
122   end;
123
124 procedure transf(p:integer;var v:vec;var dv:integer);
125 var i,aux:integer;
126 begin

```

```

127 dv:=0;
128 while p<>0 do
129 begin
130   inc(dv);
131   v[dv]:=p mod 10;
132   p:=p div 10;
133 end;
134 for i:=1 to dv div 2 do
135 begin
136   aux:=v[i];
137   v[i]:=v[dv-i+1];
138   v[dv-i+1]:=v[i];
139 end;
140
141 end;
142
143 procedure numar;
144 var i,j,dr,dx:integer;
145   nr,dy:integer;
146   rez,x,y:vec;
147 begin
148   j:=1;
149   for i:=2 to n do
150     if e[o[i]]<>e[o[i-1]] then inc(j);
151   nr:=1;
152   for i:=1 to 500 do rez[i]:=0;
153   dr:=1;rez[1]:=1;
154   for i:=k downto (k-j+1) do
155 begin
156   transf(i,x,dx);
157   produs1(rez,dr,x,dx,y,dy);
158   rez:=y;
159   dr:=dy;
160   end;
161   for i:=1 to dr do write(f,rez[i]);
162   writeln(f);
163 end;
164
165
166 procedure rezolvare;
167 var i,j,cl:integer;
168 begin
169   assign(f,'hotel.out');
170   rewrite(f);
171   j:=1;
172   for i:=2 to n do
173     if e[o[i]]<>e[o[i-1]] then inc(j);
174     if k<j then writeln(f,'0')
175   else
176   begin
177     x[o[1]]:=1;cl:=1;
178     for i:=2 to n do
179       if e[o[i]]=e[o[i-1]] then x[o[i]]:=x[o[i-1]]
180     else
181     begin
182       inc(cl);
183       x[o[i]]:=cl;
184     end;
185   numar;
186   for i:=1 to n do writeln(f,c[i],' ',x[i]);
187
188 end;
189 close(f);
190 end;
191
192 begin
193   readdata;
194   sort(l,n);
195   rezolvare;
196 end.

```

36.2 Lac

prof. Dan Grigoriu, Bucureşti

O zonă mlăştinoasă are formă dreptunghiulară, având nl linii și nc coloane. Ea este formată din celule cu latura de o unitate. O parte din acestea reprezintă uscat, iar altele reprezintă apă, uscatul fiind codificat cu 0, iar apa cu 1. Se dorește să se obțină un drum de pe malul de nord spre cel de sud, trecând doar pe uscat. Celulele cu apă pot fi transformate în uscat, parașutând într-un loc cu apă câte un ponton (o plută) de dimensiunea unei celule. Deoarece parașutarea este periculoasă, se dorește să avem un număr minim de parașutări. Deplasarea se poate face cu către o celulă pe linie, pe coloană, sau pe diagonală.

Cerință

Scrieți un program care determină numărul minim de pontoane și coordonatele acestora.

Datele de intrare

Fișierul **lac.in** are următoarea structură:

- pe prima linie se află două numere naturale nl și nc separate de un spațiu, reprezentând numărul de linii, respectiv de coloane ale zonei;
- pe următoarele nl linii se află câte nc valori binare separate de către un spațiu, reprezentând configurația zonei (0 pentru uscat și 1 pentru apă)

Datele de ieșire

Fișierul **lac.out** va avea următoarea structură:

- pe prima linie un număr natural min , care reprezintă numărul cerut de pontoane
- pe următoarele min linii vom avea căte două numere naturale separate de către un spațiu, reprezentând coordonatele celor min pontoane (linie și coloană).

Restricții și precizări

- $1 \leq nl \leq 100$
- $1 \leq nc \leq 100$
- Soluția cu număr minim de pontoane nu este unică.
- Dacă există mai multe soluții se va afișa una singură.
- Numerotarea liniilor și coloanelor începe cu valoarea 1.

Exemplu

lac.in	lac.out
8 9	2
0 1 1 1 1 1 1 1 1	4 5
0 1 1 1 1 1 1 1 1	7 8
1 0 1 1 1 0 1 1 1	
1 1 0 0 1 1 0 1 1	
1 1 1 1 1 1 1 0 1	
1 1 1 1 1 1 1 1 0	
1 1 1 1 1 1 1 1 1	
1 1 1 1 1 1 0 1 1	

Timp maxim de executare: 1 secundă/test

36.2.1 Indicații de rezolvare

Soluția comisiei

Detalii privind rezolvarea problemei:

– se lucrează cu 3 matrice:

- * una ce reține harta zonei (cu valori 0 pentru uscat și 1 pentru apă);
 - * a două în care se calculează costurile minime ale deplasărilor prin fiecare punct al zonei ;
 - * a treia ce reține direcțiile din care s-a ajuns pe pozițiile respective pentru acel cost minim.
- inițial, matricea de costuri are costurile maxime (cazul în care zona e plină de apă și atunci linia 1 are costul 1, linia 2 are costul 2, etc), iar matricea de direcții conține doar direcția spre jos;
- pozițiile punctelor de uscat se rețin într-o coadă; pentru fiecare din ele se verifică dacă ele au costuri mai mari decât ale vecinilor din zonă, caz în care costul acestui punct de uscat devine egal cu acelui vecin, iar direcția din care s-a ajuns de la vecin la punctul curent se reține în

matricea de direcții; totodată, punctul care a suferit modificări de cost e trecut în *coadă*, pentru un nou calcul și aşa mai departe, până la epuizarea cozii.

– apoi se cercetează minimul de pe ultima linie a matricei de costuri, și cu ajutorul matricei de direcții se reface drumul până la prima linie; pentru fiecare punct prin care se trece, se adaugă în prima matrice valoarea 2; astfel, fiecare punct de uscat prin care s-a trecut devine 2, iar fiecare punct cu 1 (apă) devine 3, care e interpretat ca fiind ponton.

– În final, se parcurge această matrice și se numără valorile 3 (pontoanele) și se trec în fisier, următoarele coordonatele lor din matrice.

– Eventual se poate afisa (selectiv) matricea pentru a se evidenția drumul ales și pozițiile pontoanelor, dar asta nu s-a cerut.

GInfo 12/6 octombrie 2002

Pentru rezolvarea acestei probleme vom folosi o variantă puțin modificată a *algoritmului lui Lee*.

Pentru fiecare poziție a matricei care reprezintă o zonă mlăștinoasă vom determina numărul minim de pontoane care trebuie să fie amplasate pentru a se ajunge în poziția respectivă. Pentru aceasta vom considera că pentru a ajunge în imediata vecinătate a primei linii a matricei numărul de pontoane necesare este 0.

Pentru o anumită poziție, numărul de pontoane necesare ajungerii în punctul respectiv este dat de cel mai mic număr corespunzător uneia dintre pozițiile învecinate la care se adaugă, eventual, un ponton dacă poziția nu corespunde unei porțiuni de uscat.

După o astfel de parcursare a matricei, vom avea pentru fiecare poziție a matricei o anumită valoare, dar nu suntem siguri că aceasta este cea minimă. Vom parcurge succesiv matricea încercând să îmbunătățim valorile obținute.

O valoare va fi modificată dacă se poate ajunge în poziția curentă dintr-o poziție învecinată și se obține un număr mai mic de pontoane pentru poziția curentă.

În momentul în care nu va mai exista nici o parcursare care să aducă îmbunătățiri, am obținut rezultatul final.

Practic cele două tipuri de parcursuri pot fi "asimilate" în una singură dacă, inițial, se marchează toate pozițiile matricei cu o valoare suficient de mare.

De fiecare dată când are loc o îmbunătățire, vom păstra direcția din care s-a ajuns în poziția curentă pentru a putea reconstituui drumul.

În final, numărul minim de pontoane va fi dat de cea mai mică valoare de pe ultima linie a matricei. Pentru reconstituirea drumului, se va porni în sens invers, de pe poziția de pe ultima linie în poziția din care s-a ajuns în ea și aşa mai departe, până se ajunge pe prima linie. Dacă se ajunge într-o poziție care nu corespunde unei zone de uscat, atunci va trebui amplasat un ponton în acea poziție.

Analiza complexității

Citirea datelor de intrare implică o traversare a matricei, ordinul de complexitate al acestei operații fiind $O(mn)$.

O parcursare a matricei în vederea realizării unei îmbunătățiri are același ordin de complexitate.

Datorită faptului că, după prima parcursare, suntem siguri că se poate ajunge pe ultima linie folosind m pontoane (completarea integrală a coloanei), nu se vor realiza în nici o situație mai mult de m parcursuri de îmbunătățire. Din aceste motive, ordinul de complexitate al operației de determinare a numărului de pontoane necesare pentru ajungerea în fiecare poziție este $O(m)O(mn) = O(m^2n)$.

Determinarea valorii minime de pe ultima linie se realizează în timp liniar, ordinul de complexitate fiind $O(n)$.

Reconstituirea drumului ar putea, teoretic, să necesite traversarea întregii matrice. Astfel, în cel mai defavorabil caz, ordinul de complexitate al acestei operații este $O(mn)$.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(m \cdot n) + O(m^2 \cdot n) + O(n) + O(m \cdot n) = O(m^2 \cdot n)$.

36.2.2 Rezolvare detaliată

Listing 36.2.1: lac.java

```

1 import java.io.*; // matrice bordata ==> "vecini" pentru oricare (i,j)
2 class Lac
3 {

```

```

4   static final String fisi="lac.in";
5   static int[][] lac=new int[102][102];      // lacul
6   static int[][] cost=new int[102][102];     // costul ajungerii in (i,j)
7   static int[][] di=new int[102][102];       // directia sosirii in (i,j)
8   static int[][] dj=new int[102][102];       // pozitia din care s-a ajuns
9   static int[][] pontoane=new int[2][101];
10  static int dii,djj;                      // directia sosirii in (i,j) pentru cost minim
11  static int m,n;                         // dimensiunile lacului
12  static int pozMin, costMin;             // pozitia minimului pe ultima linie
13
14 public static void main(String[] args) throws IOException
15 {
16     long t1,t2;
17     t1=System.currentTimeMillis();
18     citescDate();
19     matriceCosturi();
20     pozMin();
21     determinaPontoane();
22     afisSolutia();
23     t2=System.currentTimeMillis();
24     System.out.println("TIME = "+(t2-t1)+" millisec ");
25 } // main()
26
27 static void citescDate() throws IOException
28 {
29     int i,j;
30     StreamTokenizer st=new StreamTokenizer(
31         new BufferedReader(new FileReader(fisi)));
32     st.nextToken(); m=(int)st.nval;
33     st.nextToken(); n=(int)st.nval;
34
35     for(i=1;i<=m;i++)
36         for(j=1;j<=n;j++)
37         {
38             st.nextToken(); lac[i][j]=(int)st.nval;
39         }
40 } // citescDate()
41
42 static void matriceCosturi()
43 {
44     // i pe linia i
45     // daca nu apare minimizare de cost, consider ca "vine" de sus
46     // adica di=-1 si dj=0
47     int i,j;
48     boolean micsoratCost;
49     int niter=0,minij;
50     for(i=0;i<=m+1;i++)
51         for(j=0;j<=n+1;j++)
52         {
53             cost[i][j]=i;
54             di[i][j]=-1;
55             dj[i][j]=0;
56         }
57
58     do // parcurgeri pana cand nu mai apar costuri micsorate
59     {
60         ++niter;
61         micsoratCost=false;
62         for(i=1;i<=m;i++)
63             for(j=1;j<=n;j++)
64             {
65                 minij=minCostVecini(i,j);
66                 if(cost[i][j]>minij+lac[i][j])
67                 {
68                     cost[i][j]=minij+lac[i][j];
69                     di[i][j]=dii;
70                     dj[i][j]=djj;
71                     micsoratCost=true;
72                 }
73             }
74         } while(micsoratCost);
75 } // matriceCosturi()
76
77 static int minCostVecini(int ii, int jj)
78 {
79     int i,j,min;

```

```

80     min=cost[iii][jjj];           // vechiul cost
81     for(i=-1;i<=1;i++)          // i=0 pentru verticala
82         for(j=-1;j<=1;j++)        // j=0 pentru orizontala
83             if(cost[pii+i][jj+j]<min)
84             {
85                 min=cost[pii+i][jj+j];
86                 pii=i;
87                 jjj = j;
88             }
89     return min;
90 }// minCostVecini(...)

91
92     static void pozMin()
93     {
94         int i;
95         costMin=cost[m][1];
96         pozMin=1;
97         for(i=2;i<=n;i++)
98             if(cost[m][i]<costMin) {costMin=cost[m][i]; pozMin=i;}
99 }// pozMin()

100
101    static void determinaPontoane()
102    {
103        int ii=m, jj=pozMin, kponton=costMin;
104
105        while(kponton>0)
106        {
107            if(lac[ii][jj]>0)
108            {
109                puntoane[0][kponton]=ii;
110                puntoane[1][kponton]=jj;
111                kponton--;
112            }
113
114            // trec in pozitia vecina din care s-a ajuns in (i,j)
115            ii+=di[ii][jj];   // trecere in noua pozitie (s-a schimbat i)
116            jj+= dj[ii][jj];
117        }// while
118    }//determinaPontoane()

119
120    static void afisSolutia() throws IOException
121    {
122        PrintWriter out=new PrintWriter(
123                    new BufferedWriter(new FileWriter("lac.out")));
124        out.println(costMin);
125        for(int k=1;k<=costMin;k++)
126            out.println(pontoane[0][k]+" "+pontoane[1][k]);
127        out.close();
128    }//afisSolutia()
129 }// class

```

36.2.3 Cod sursă

Listing 36.2.2: lac.pas

```

1 uses dos;
2 const dl:array[1..8] of -1..1=(-1,-1,-1,0,0,1,1,1);
3     dc:array[1..8] of -1..1=(-1,0,1,-1,1,-1,0,1);
4 type pelem^elem;
5     elem=record l,c:byte; next:pelem end;
6 var lac:array[0..100,0..100] of 0..1;
7     cost:array[0..100,0..100] of byte;
8     directie:array[0..100,0..100] of 1..8;
9     pc,uc,p0,u0:pelem;
10    min,nl,nc,i,j:byte;
11    h,m,s,hs:word;
12
13 procedure push(linie,coloana:byte; var p,u:pelem);
14 var q:pelem;
15 begin
16     new(q);
17     q^.l:=linie; q^.c:=coloana;

```

```

18     q^.next:=nil;
19     if p=nil
20         then p:=q
21         else u^.next:=q;
22     u:=q;
23 end;
24
25 procedure pop(var linie,coloana:byte; var p,u:pelem);
26 var q:pelem;
27 begin
28     q:=p;
29     linie:=q^.l; coloana:=q^.c;
30     if p=u then u:=nil;
31     p:=p^.next;
32     dispose(q);
33 end;
34
35 procedure init;
36 var i,j,il,jl:byte;
37   f:text;
38 begin
39     assign(f,'lac.in'); reset(f);
40     readln(f,nl,nc);
41     fillchar(lac,sizeof(lac),0);
42     fillchar(directie,sizeof(directie),7);
43 { for i:=0 to nc+1 do cost[0,i]:=9; } { Bordare sa n-o ia in sus }
44     for i:=1 to nl do
45     begin
46         cost[i,0]:=i; cost[i,nc+1]:=i;
47         for j:=1 to nc do
48         begin
49             read(f,lac[i,j]);
50             cost[i,j]:=i;
51             if lac[i,j]=0
52                 then
53                 push(i,j,p0,u0);
54             end;
55             readln(f);
56         end;
57         for j:=0 to nc+1 do
58             cost[nl+1,j]:=nl+1;
59         close(f);
60     end;
61
62 function OK (l0,c0:byte):boolean;
63 var l1,c1,i:byte;
64 begin
65     OK:=true;
66     for i:=1 to 8 do
67     begin
68         l1:=l0+dl[i]; c1:=c0+dc[i];
69         if cost[l1,c1]<cost[l0,c0]
70             then
71             begin
72                 cost[l0,c0]:=cost[l1,c1]; directie[l0,c0]:=9-i;
73                 OK:=false;
74             end;
75     end;
76 end;
77
78 procedure reconfig;
79 var l0,c0,l1,c1,nou,i,j:byte;
80 begin
81     while pc<>nil do
82     begin
83         pop(l0,c0,pc,uc);
84         for i:=1 to 8 do
85         begin
86             l1:=l0+dl[i]; c1:=c0+dc[i];
87             nou:=cost[l0,c0]+lac[l1,c1];
88             if nou<cost[l1,c1]
89                 then
90                     if l1*c1*(nl+1-l1)*(nc+1-c1)<>0
91                         then
92                             begin
93                                 push(l1,c1,pc,uc);

```

```

94                               cost[l1,c1]:=nou; directie[l1,c1]:=i;
95                           end;
96           end;
97       end;
98   end;
99
100  procedure config;
101  var l0,c0,i1,j1:byte;
102 begin
103     while p0<>nil do
104         begin
105             pop(l0,c0,p0,u0);
106             if not OK(l0,c0)
107                 then
108                     begin
109                         push(l0,c0,pc,uc);
110                         reconfig;
111                     end;
112         end;
113     end;
114
115  procedure solutie;
116  var i,l0,c0,i1,j1,x1,y1:byte;
117 begin
118     config;
119     min:=cost[nl,1];
120     c0:=1;
121     for i:=2 to nc do
122         begin
123             if cost[nl,i]<min
124                 then
125                     begin
126                         min:=cost[nl,i]; c0:=i;
127                     end;
128         end;
129     l0:=nl;
130     repeat
131         x1:=l0; y1:=c0;
132         lac[l0,c0]:=lac[l0,c0]+2;
133         i:=directie[l0,c0];
134         l0:=l0-dl[i];
135         c0:=c0-dc[i];
136     until l0=0;
137 end;
138
139  procedure afisare;
140  var i,j:integer;
141  g:text;
142 begin
143     assign(g,'lac.out'); rewrite(g);
144     writeln(g,min);
145     for i:=1 to nl do
146         for j:=1 to nc do
147             if lac[i,j]=3
148                 then writeln(g,i,' ',j);
149     close(g);
150 end;
151
152 BEGIN
153     init;
154     solutie;
155     afisare;
156 END.
```

36.3 Logic

Marius Andrei, Bucureşti

Într-o zi a venit la mine un coleg din clasa a X-a și mi-a propus un joc de logică. Mi-a arătat următoarea figură:

1	2	4	3	5
6	8	7	9	10
16			15	14

Figura 36.2: Logic1

Pe ea am numerotat segmentele, pentru a fi mai clară noțiunea de segment. Eu am la dispoziție un creion care se află pe hârtie în zona exterioară și trebuie să trasez curbe pe foaie, fără să ridic creionul, astfel încât să trec prin toate segmentele (fără extremități) exact o dată. La sfârșit trebuie să mă afli tot în exterior. Linile (curbele) mele se pot intersecta.

Am început și am încercat de mai multe ori, dar n-am reușit.

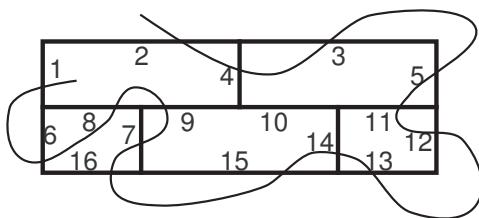


Figura 36.3: Logic2

Acum, când am mai crescut mi-am demonstrat că nu se poate, dar am văzut că pentru alte figuri se poate. De exemplu aceasta:

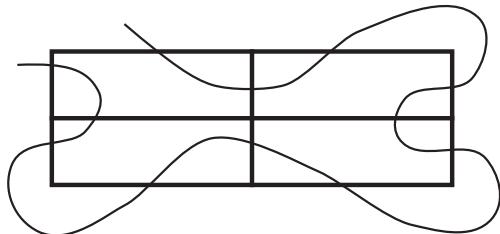


Figura 36.4: Logic3

Eu m-am prins de ce uneori merge și alteori nu, dar vreau să văd dacă voi putea să vă prindeți. De aceea o să vă dau câteva figuri și pentru fiecare trebuie să răspundeți cu *DA* sau *NU*, dacă se poate sau nu trasa o curbă de tipul descris mai sus.

Cerință

Scrieți un program care răspunde cu *DA* sau *NU* pentru figurile propuse de mine.

Date de intrare

Fișierul de intrare LOGIC.IN are următoarea structură:

pe prima linie se află numărul *T* de figuri

pe următoarele *T* grupuri de linii se află datele celor *T* figuri, după cum urmează:

pe prima linie a grupului se află valoarea lui *N* reprezentând numărul de linii și de coloane din matrice.

pe următoarele *N* linii se află câte *N* numere separate printr-un spațiu (numere întregi între 0 și 255 inclusiv), reprezentând elementele matricei.

Această matrice codifică figura astfel:

Definim o zonă ca fiind o parte continuă a matricei care conține același număr și este aria maximă cu această proprietate. Altfel spus, două căsuțe alăturate (care diferă la o singură coordonată printr-o unitate) care conțin același număr, se lipesc. Astfel, în interiorul zonei nu vor exista segmente. La aceste zone se mai adaugă și exteriorul matricei ca fiind o nouă zonă. În matrice pot fi zone cu același număr, dar care nu au legătură și se consideră două zone diferite.

Segmentele sunt segmente de dreaptă care separă două zone și sunt maxime ca lungime (adică nu putem considera două segmente unul în prelungirea celuilalt, în loc de unul singur). De exemplu, figura definită prin:

3
1 1 2
1 2 2
1 1 2

arată ca în desenul de mai jos (am numerotat și segmentele):

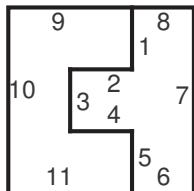


Figura 36.5: Logic4

Se vede din figură că între zona 1 și 2 există 5 segmente. Între 1 și exterior 3 segmente, iar între 2 și exterior tot 3 segmente. De remarcat că segmentul 10 este compus din 3 segmente mici, dar noi îl considerăm unul singur, conform definiției.

Date de ieșire

În fișierul LOGIC.OUT se vor scrie T linii, corespunzătoare fiecărui test. Pe fiecare linie va fi scris *DA* sau *NU* (litere mari) în funcție de testul respectiv, dacă se poate trasa o linie de tipul cerut sau nu.

Restricții

$$0 < T < 11$$

$$0 < N < 101$$

Exemplu

LOGIC.IN LOGIC.OUT

2 DA

2 NU

1 2

3 4

4

1 1 2 2

1 1 2 2

3 4 4 1

3 4 4 1

Explicație: Cele două figuri sunt primele 2 exemple.

Timp de execuțare/test: 1 secundă

36.3.1 Indicații de rezolvare

Indicația oficială

Pentru ca problema să aibă soluție trebuie ca numărul de segmente care delimită fiecare zonă să fie par. Problema se reduce deci la a calcula, pentru fiecare zonă, numărul de segmente care o delimită.

Vom considera că nu există două zone marcate prin același element. Primul pas al rezolvării constă în transformarea matricei astfel încât să se respecte această condiție și se rezolvă aplicând *algoritmul FILL* fiecărei zone și marcând pătratele componente cu culori unice pentru fiecare zonă.

Să analizăm linia de demarcație dintre două linii i și $i + 1$ din matrice. Pe această linie, un segment este determinat de un sir maximal de pozitii în care valorile de deasupra, respectiv de dedesubtul liniei sunt egale între ele (ambele condiții simultan). Acest segment crește cu o unitate numerele de segmente asociate zonelor date de elementele din matrice.

Parcurgand linia de demarcație de la stanga la dreapta, se observă că apariția unui segment orizontal este dată de o schimbare a cel puțin uneia din valorile de deasupra și de dedesubt. Dacă ambele rămân egale cu cele din coloana precedentă, ne aflăm în cadrul unui același segment, deci nu se va incrementa numărul de segmente.

Se face o analiză similară pe verticală. Complexitatea rezolvării este $O(N * N)$.

Soluție prezentată în GInfo 12/6 - octombrie 2002

Pentru început vom observa că, pentru ca trasarea să fie posibilă, fiecare zonă trebuie să fie delimitată de un număr par de segmente. Aceasta se datorează faptului că linia trebuie să intre și să iasă din fiecare zonă de un anumit număr de ori.

Așadar, pentru fiecare zonă va trebui să numărăm segmentele (așa cum sunt definite ele în enunț) care o delimită și să verificăm dacă numărul obținut este par sau impar. În momentul în care găsim o zonă delimitată de un număr par de segmente, vom ști că trasarea nu este posibilă. Dacă toate zonele sunt delimitate de un număr par de segmente, atunci trasarea este posibilă.

Numărarea segmentelor care delimită o zonă se realizează pe baza unui *algoritm de umplere*. În momentul în care ajungem la un punct de la marginea zonei, se pune problema creșterii numărului de segmente. Există mai multe situații în care numărul de segmente va crește:

- se ajunge în partea de sus a zonei, iar punctul aflat imediat deasupra și cel din dreapta acestuia fac parte din zone diferite;
- se ajunge în partea de jos a zonei, iar punctul aflat imediat dedesubt și cel din dreapta acestuia fac parte din zone diferite;
- se ajunge în partea din stânga a zonei, iar punctul aflat imediat la stânga și cel de deasupra sa fac parte din zone diferite;
- se ajunge în partea din dreapta a zonei, iar punctul aflat imediat la dreapta și cel de deasupra sa fac parte din zone diferite;
- se ajunge într-unul din colțurile zonei.

Dacă ultima condiție este îndeplinită simultan cu una dintre primele patru, numărul segmentelor va crește doar cu 1. Creșterile datorate primelor patru condiții sunt cumulative.

Analiza complexității

Vom studia acum complexitatea algoritmului pentru un desen. Citirea datelor de intrare implică o traversare a matricei, deci această operație se realizează într-un timp de ordinul $O(n^2)$.

La fiecare pas al algoritmului de umplere se numără segmentele care trebuie adăugate pentru poziția curentă. Se pot adăuga cel mult patru segmente, așadar ordinul de complexitate al operației este $O(1)$. Algoritmul de umplere implică parcurgerea integrală a matricei, motiv pentru care vor fi vizitate toate cele n^2 poziții. Așadar, algoritmul de verificare a posibilității de trasare a liniei are ordinul de complexitate $O(n^2)O(1) = O(n^2)$.

Datele de ieșire constau din scrierea unui singur mesaj, deci această operație se efectuează în timp liniar.

În concluzie, algoritmul de rezolvare a acestei probleme, pentru un desen, are ordinul de complexitate $O(n^2) + O(n^2) + O(1) = O(n^2)$. Pentru a stabili ordinul de complexitate al algoritmului de rezolvare a întregii probleme, vom considera că un set de date cuprinde t desene. Așadar, ordinul de complexitate este $O(t) \cdot O(n^2) = O(t \cdot n^2)$.

36.3.2 Rezolvare detaliată

Varianta 1:

Listing 36.3.1: logic1.java

```

1 import java.io.*;      // teste 9 si 10 ?
2 class Logic1          // depăsește stiva ... !!! recursivitatea ... !!!
3 {                      // la col=45 !!! (cu toate liniile)
4     static int[][] a=new int[102][102];
5     static int nt;           // nr teste
6     static int n;            // dimensiune matrice
7     static boolean ok;
8     static int idz;          // identificator zona
9     static int ns;           // nr segmente
10    static StreamTokenizer st;
11
12    public static void main(String[] args) throws IOException
13    {
14        st=new StreamTokenizer(new BufferedReader(new FileReader("logic.in")));
15        PrintWriter out=new PrintWriter(
16            new BufferedWriter(new FileWriter("logic.out")));
17
18        st.nextToken(); nt=(int)st.nval;
19        for(int k=1;k<=nt;k++)
20        {
21            preiaTestul();

```

```

22     rezolvare();
23     if(ok) out.println("DA"); else out.println("NU");
24   }
25   out.close();
26 } // main(...)

27
28 static void preiaTestul() throws IOException
29 {
30   int i,j;
31   st.nextToken(); n=(int)st.nval;
32   for(i=1;i<=n;i++)           // matrice bordata cu zero
33     a[0][i]=a[i][0]=a[n+1][i]=a[i][n+1]=0;
34
35   a[0][0]=a[n+1][0]=a[0][n+1]=a[n+1][n+1]=-1;    // in "colturi"
36
37   for(i=1;i<=n;i++)
38     for(j=1;j<=n;j++) { st.nextToken(); a[i][j]=(int)st.nval; }
39 } // citesteMatrice()

40
41 static void rezolvare()
42 {
43   int i,j;
44   ok=true;
45   for(i=1;i<=n;i++)
46     for(j=1;j<=n;j++)           // nemarcat
47       if(a[i][j]>0)           // nemarcat
48     {
49       ns=0;
50       idz=a[i][j];           // identificatorul zonei
51       zona(i,j);
52       if(ns%2==1) { ok=false; return; }
53     }
54 } // rezolvare()

55
56 static void zona(int i, int j)
57 {
58   a[i][j]=-a[i][j];           // marcare zona: +=nemarcat -=marcat
59
60   if(Math.abs(a[i-1][j])!=idz) // la N este alta zona
61     if((a[i-1][j-1]!=a[i-1][j]) || (Math.abs(a[i][j-1])!=idz))
62       ns++;
63
64   if(Math.abs(a[i+1][j])!=idz) // la S este alta zona
65     if((a[i+1][j-1]!=a[i+1][j]) || (Math.abs(a[i][j-1])!=idz))
66       ns++;
67
68   if(Math.abs(a[i][j-1])!=idz) // la V este alta zona
69     if((a[i-1][j-1]!=a[i][j-1]) || (Math.abs(a[i-1][j])!=idz))
70       ns++;
71
72   if(Math.abs(a[i][j+1])!=idz) // la E este alta zona
73     if((a[i-1][j+1]!=a[i][j+1]) || (Math.abs(a[i-1][j])!=idz))
74       ns++;
75
76   if(a[i-1][j]==idz) zona(i-1,j);
77   if(a[i+1][j]==idz) zona(i+1,j);
78   if(a[i][j-1]==idz) zona(i,j-1);
79   if(a[i][j+1]==idz) zona(i,j+1);
80 } // zona(...)
81 } // class

```

Varianta 2:

Listing 36.3.2: logic2.java

```

1 import java.io.*; // testele 9 si 10 ?
2 class Logic2      // pun i, j globale dar ... depaseste stiva ... !!!
3 {                  // la col=48 !!! (cu toate liniile)
4   static int[][] a=new int[102][102];
5   static int nt;        // nr teste
6   static int n;         // dimensiune matrice
7   static boolean ok;
8   static int idz;       // identificator zona
9   static int ns;        // nr segmente
10  static StreamTokenizer st;
11  static int i,j;

```

```

12
13     public static void main(String[] args) throws IOException
14     {
15         st=new StreamTokenizer(new BufferedReader(new FileReader("logic.in")));
16         PrintWriter out=new PrintWriter(
17             new BufferedWriter(new FileWriter("logic.out")));
18
19         st.nextToken(); nt=(int)st.nval;
20
21         for(int k=1;k<=nt;k++)
22         {
23             preiaTestul();
24             rezolvare();
25             if(ok) out.println("DA"); else out.println("NU");
26         }
27         out.close();
28     }// main...
29
30     static void preiaTestul() throws IOException
31     {
32         st.nextToken(); n=(int)st.nval;
33         for(i=1;i<=n;i++) // matrice bordata cu zero
34             a[0][i]=a[i][0]=a[n+1][i]=a[i][n+1]=0;
35
36         a[0][0]=a[n+1][0]=a[0][n+1]=a[n+1][n+1]=-1; // in "colturi"
37
38         for(i=1;i<=n;i++)
39             for(j=1;j<=n;j++) {st.nextToken(); a[i][j]=(int)st.nval;}
40     }// citesteMatrice()
41
42     static void rezolvare()
43     {
44         ok=true;
45         for(i=1;i<=n;i++)
46             for(j=1;j<=n;j++)
47                 if(a[i][j]>0) // nemarcat
48                 {
49                     ns=0;
50                     idz=a[i][j]; // identificatorul zonei
51                     zona();
52                     if(ns%2==1) { ok=false; return; }
53                 }
54     }// rezolvare()
55
56     static void zona()
57     {
58         a[i][j]=-a[i][j]; // marcare zona: +=nemarcat -=marcat
59
60         if(Math.abs(a[i-1][j])!=idz) // la N este alta zona
61             if((a[i-1][j-1]!=a[i-1][j]) || (Math.abs(a[i][j-1])!=idz))
62                 ns++;
63
64         if(Math.abs(a[i+1][j])!=idz) // la S este alta zona
65             if((a[i+1][j-1]!=a[i+1][j]) || (Math.abs(a[i][j-1])!=idz))
66                 ns++;
67
68         if(Math.abs(a[i][j-1])!=idz) // la V este alta zona
69             if((a[i-1][j-1]!=a[i][j-1]) || (Math.abs(a[i-1][j])!=idz))
70                 ns++;
71
72         if(Math.abs(a[i][j+1])!=idz) // la E este alta zona
73             if((a[i-1][j+1]!=a[i][j+1]) || (Math.abs(a[i-1][j])!=idz))
74                 ns++;
75
76         if(a[i-1][j]==idz) {i--; zona(); i++;} //zona(i-1,j);
77         if(a[i+1][j]==idz) {i++; zona(); i--;} //zona(i+1,j);
78         if(a[i][j-1]==idz) {j--; zona(); j++;} //zona(i,j-1);
79         if(a[i][j+1]==idz) {j++; zona(); j--;} //zona(i,j+1);
80     }// zona...
81 } // class

```

Varianta 3:

Listing 36.3.3: logic3.java

```

1 import java.io.*; // iterativ ... cu stiva !

```

```

2  class Logic3
3  {
4      static int[][] a=new int[102][102];
5      static int[] is=new int[100*100]; // stiva pentru i
6      static int[] js=new int[100*100]; // stiva pentru j
7      static int vs; // varf stiva
8      static int nt; // nr teste
9      static int n; // dimensiune matrice
10     static boolean ok;
11     static int idz; // identificator zona
12     static int ns; // nr segmente
13     static StreamTokenizer st;
14
15    public static void main(String[] args) throws IOException
16    {
17        st=new StreamTokenizer(new BufferedReader(new FileReader("logic.in")));
18        PrintWriter out=new PrintWriter(
19            new BufferedWriter(new FileWriter("logic.out")));
20        st.nextToken(); nt=(int)st.nval;
21        for(int k=1;k<=nt;k++)
22        {
23            preiaTestul();
24            rezolvare();
25            if(ok) out.println("DA"); else out.println("NU");
26        }
27        out.close();
28    } // main...
29
30    static void preiaTestul() throws IOException
31    {
32        int i,j;
33        st.nextToken(); n=(int)st.nval;
34        for(i=1;i<=n;i++) a[0][i]=a[i][0]=a[n+1][i]=a[i][n+1]=0; // bordare
35        a[0][0]=a[n+1][0]=a[0][n+1]=a[n+1][n+1]=-1; // initializare in "colturi"
36        for(i=1;i<=n;i++)
37            for(j=1;j<=n;j++) {st.nextToken(); a[i][j]=(int)st.nval;}
38    } // citesteMatrice()
39
40    static void rezolvare()
41    {
42        int i,j;
43        ok=true;
44        for(i=1;i<=n;i++)
45            for(j=1;j<=n;j++)
46                if(a[i][j]>0) // nemarcat
47                {
48                    ns=0;
49                    idz=a[i][j]; // identificatorul zonei
50                    vs=1; is[vs]=i; js[vs]=j;
51                    zona();
52                    if(ns%2==1) { ok=false; return; }
53                }
54    } // rezolvare()
55
56    static void zona()
57    {
58        int i,j;
59        a[is[1]][js[1]]*=-1; // marcare zona: +=nemarcat -=marcat
60        while(vs>0)
61        {
62            i=is[vs]; j=js[vs]; vs--;
63
64            if(Math.abs(a[i-1][j])!=idz) // la N este alta zona
65                if((a[i-1][j-1]!=a[i-1][j]) || (Math.abs(a[i][j-1])!=idz)) ns++;
66
67            if(Math.abs(a[i+1][j])!=idz) // la S este alta zona
68                if((a[i+1][j-1]!=a[i+1][j]) || (Math.abs(a[i][j-1])!=idz)) ns++;
69
70            if(Math.abs(a[i][j-1])!=idz) // la V este alta zona
71                if((a[i-1][j-1]!=a[i][j-1]) || (Math.abs(a[i-1][j])!=idz)) ns++;
72
73            if(Math.abs(a[i][j+1])!=idz) // la E este alta zona
74                if((a[i-1][j+1]!=a[i][j+1]) || (Math.abs(a[i-1][j])!=idz)) ns++;
75
76            if(a[i-1][j]==idz){a[i-1][j]=-idz; vs++; is[vs]=i-1; js[vs]=j;}
77            if(a[i+1][j]==idz){a[i+1][j]=-idz; vs++; is[vs]=i+1; js[vs]=j;}

```

```

78     if(a[i][j-1]==idz){a[i][j-1]=-idz; vs++; is[vs]=i; js[vs]=j-1;}
79     if(a[i][j+1]==idz){a[i][j+1]=-idz; vs++; is[vs]=i; js[vs]=j+1;}
80   } // while(vs>0)
81 } // zona(...)
82 } // class

```

Varianta 4:

Listing 36.3.4: logic4.java

```

1 import java.io.*;                                // iterativ ... cu coada !
2 class Logic4
3 {
4     static final int qdim=101;                    // dim coada circulara
5
6     static int[][] a=new int[102][102];
7
8     static int[] qi=new int[qdim];    // coada pentru i din pozitia (i,j)
9     static int[] qj=new int[qdim];    // coada pentru j din pozitia (i,j)
10    static int ic, sc;                  // ic=inceput coada
11
12   static int nt;                        // nr teste
13   static int n;                         // dimensiune matrice
14   static boolean ok;                   // identificator zona
15   static int idz;                     // identificator zona
16   static int ns;                      // nr segmente
17   static StreamTokenizer st;
18
19   public static void main(String[] args) throws IOException
20   {
21       st=new StreamTokenizer(new BufferedReader(new FileReader("logic.in")));
22       PrintWriter out=new PrintWriter(
23           new BufferedWriter(new FileWriter("logic.out")));
24
25       st.nextToken(); nt=(int)st.nval;
26
27       for(int k=1;k<=nt;k++)
28       {
29           preiaTestul();
30           rezolvare();
31           if(ok) out.println("DA"); else out.println("NU");
32       }
33       out.close();
34   } // main...
35
36   static void preiaTestul() throws IOException
37   {
38       int i,j;
39       st.nextToken(); n=(int)st.nval;
40       for(i=1;i<=n;i++)                  // matrice bordata cu zero
41           a[0][i]=a[i][0]=a[n+1][i]=a[i][n+1]=0;
42
43       a[0][0]=a[n+1][0]=a[0][n+1]=a[n+1][n+1]=-1; // in "colturi"
44
45       for(i=1;i<=n;i++)
46           for(j=1;j<=n;j++) {st.nextToken(); a[i][j]=(int)st.nval;}
47   } // citesteMatrice()
48
49   static void rezolvare()
50   {
51       int i,j;
52       ok=true;
53       for(i=1;i<=n;i++)
54           for(j=1;j<=n;j++)
55               if(a[i][j]>0)          // nemarcat
56               {
57                   ns=0;
58                   idz=a[i][j];      // identificatorul zonei
59
60                   ic=sc=0;           // coada vida
61                   qi[sc]=i; qj[sc]=j; sc=(sc+1)%qdim; // (i,j) --> coada circulara !
62
63                   zona();
64                   if(ns%2==1) { ok=false; return; }
65               }
66   } // rezolvare()

```

```

67
68     static void zona()
69     {
70         int i,j;
71         i=qj[ic]; j=qj[ic];
72         a[i][j]*=-1;           // marcare zona: +=nemarcat -=marcat
73         while(ic!=sc)        // coada nevida
74         {
75             i=qj[ic]; j=qj[ic]; ic=(ic+1)%qdim;
76             if(Math.abs(a[i-1][j])!=idz)    // la N este alta zona
77                 if((a[i-1][j-1]!=a[i-1][j]) || (Math.abs(a[i][j-1])!=idz))
78                     ns++;
79
80             if(Math.abs(a[i+1][j])!=idz)    // la S este alta zona
81                 if((a[i+1][j-1]!=a[i+1][j]) || (Math.abs(a[i][j-1])!=idz))
82                     ns++;
83
84             if(Math.abs(a[i][j-1])!=idz)    // la V este alta zona
85                 if((a[i-1][j-1]!=a[i][j-1]) || (Math.abs(a[i-1][j])!=idz))
86                     ns++;
87
88             if(Math.abs(a[i][j+1])!=idz)    // la E este alta zona
89                 if((a[i-1][j+1]!=a[i][j+1]) || (Math.abs(a[i-1][j])!=idz))
90                     ns++;
91
92             if(a[i-1][j]==idz){a[i-1][j]=-idz; qj[sc]=j; sc=(sc+1)%qdim;}
93             if(a[i+1][j]==idz){a[i+1][j]=-idz; qj[sc]=i+1; sc=(sc+1)%qdim;}
94             if(a[i][j-1]==idz){a[i][j-1]=-idz; qj[sc]=i; sc=(sc+1)%qdim;}
95             if(a[i][j+1]==idz){a[i][j+1]=-idz; qj[sc]=j+1; sc=(sc+1)%qdim;}
96         } // while(ic!=sc)
97     } // zona(...)
98 } // class

```

36.3.3 Cod sursă

Listing 36.3.5: logic.pas

```

1  {$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S+,T-,V+,X+}
2  {$M 65012,0,655360}
3  var orig,n,x,y,i,j,t,i1,ns:integer;
4      f:text;
5      a:array[0..101,0..101] of integer; {limita valori}
6      g:text;
7      bun:boolean;
8
9  procedure plimba;
10 begin
11   a[x,y]:=-a[x,y];
12   {in jur}
13   if abs(a[x-1,y])>>orig then
14       if ((a[x-1,y-1]>>a[x-1,y]) or (abs(a[x,y-1])>>orig)) then ns:=ns+1;
15   if abs(a[x+1,y])>>orig then
16       if ((a[x+1,y-1]>>a[x+1,y]) or (abs(a[x,y-1])>>orig)) then ns:=ns+1;
17   if abs(a[x,y-1])>>orig then
18       if ((a[x-1,y-1]>>a[x,y-1]) or (abs(a[x-1,y])>>orig)) then ns:=ns+1;
19   if abs(a[x,y+1])>>orig then
20       if ((a[x-1,y+1]>>a[x,y+1]) or (abs(a[x-1,y])>>orig)) then ns:=ns+1;
21
22   {altele}
23   if x>>1 then if a[x-1,y]=orig then begin
24       dec(x);
25       plimba;
26       inc(x);
27       end;
28   if x<>n then if a[x+1,y]=orig then begin
29       inc(x);
30       plimba;
31       dec(x);
32       end;
33   if y>>1 then if a[x,y-1]=orig then begin
34       dec(y);
35       plimba;

```

```

36      inc(y);
37      end;
38  if y>>n then if a[x,y+1]=orig then begin
39      inc(y);
40      plimba;
41      dec(y);
42      end;
43  end;
44
45 begin
46 assign(f,'logic.in');reset(f);
47 {assign(f,'t1.in');reset(f);}
48 assign(g,'logic.out');rewrite(g);
49 read(f,t);
50 for i1:=1 to t do begin
51     bun:=true;
52     read(f,n);
53     for i:=0 to n+1 do for j:=0 to n+1 do a[i,j]:=0;
54     a[0,0]:=-1;
55     a[n+1,0]:=-1;
56     a[0,n+1]:=-1;
57     a[n+1,n+1]:=-1;
58     for i:=1 to n do for j:=1 to n do read(f,a[i,j]);
59     for i:=1 to n do for j:=1 to n do if a[i,j]>-1 then
60     begin
61         ns:=0;
62         orig:=a[i,j];
63         x:=i;y:=j;
64         plimba;
65         if ns mod 2=1 then bun:=false;
66         { writeln(ns); }
67     end;
68
69     if bun then writeln(g,'DA');
70     if not bun then writeln(g,'NU');
71
72 end;
73 close(f);
74 close(g);
75 end.

```

36.4 Foto

prof. Roxana Tamplaru, Craiova

Gigel, specialist în editare grafică pe calculator, se confruntă cu o problemă. El trebuie să aranjeze patru fotografii, disponibile în format electronic, într-o pagină de prezentare astfel încât suprafața paginii să fie complet "acoperită" de cele patru fotografii și fără ca acestea să se suprapună în vreun fel. Gigel poate modifica dimensiunile inițiale ale fotografilor însă fără a deforma imaginile. Pentru aceasta el trebuie să păstreze neschimbări raportul dintre lungimea și înălțimea inițiale ale fotografilor, chiar dacă este nevoie să mărească sau să micșoreze fotografile, pentru a putea acoperi integral suprafața paginii și fără suprapunerea lor. Nu contează ordinea așezării fotografilor, putând fi translațate oriunde în cadrul paginii, însă operațiile de rotație nu sunt permise.

Cerință

Determinați pentru fiecare fotografie dimensiunile finale, cunoșcându-se dimensiunile paginii, precum și dimensiunile inițiale ale fotografilor.

Date de intrare

Fișierul de intrare: FOTO.IN are următoarea structură:

pe linia 1: numerele naturale nenule l și h separate prin spațiu reprezentând lungimea, respectiv înălțimea paginii;

pe liniile 2...5: perechi de numere naturale nenule x și y separate prin spațiu, reprezentând lungimea și înălțimea fiecărei fotografii (pe linia $i + 1$ fotografia i , $1 \leq i \leq 4$)

Date de ieșire

Fișierul de ieșire: FOTO.OUT are următoarea structură:

pe liniile 1..4: numere naturale nenule a și b separate prin spațiu, reprezentând dimensiunile finale: lungime, înălțime pentru fiecare fotografie (pe linia i pentru fotografia i , $1 \leq i \leq 4$)

Restricții: $l, h \leq 2000$ $x, y \leq 2000$ **Observații**

Dacă există mai multe soluții, se va scrie una singură.

Pentru datele de intrare alese, întotdeauna există soluție.

Exemplu:

FOTO.IN	FOTO.OUT
140 140	20 10
24 12	40 130
4 13	100 140
10 14	20 10
4 2	

Timp maxim de executare/test: 1 secundă

36.4.1 Indicații de rezolvare

Soluția oficială

Se testează toate cazurile posibile. Există două categorii de cazuri:

1) o fotografie acoperă integral în condițiile problemei o latură a paginii (orizontal sau vertical), fără a ieși din pagină pe cealaltă direcție - problema se continuă în același mod pentru porțiunea rămasă liberă în pagină, luând în considerare celelalte trei fotografii.

2) două fotografii acoperă integral în condițiile problemei o latură a paginii (orizontal sau vertical), fără a ieși din pagină pe cealaltă direcție, iar restul paginii este un dreptunghi care va fi completat de celelalte două.

Nu există soluție pentru orice set de date de intrare, dar au fost alese pentru evaluare, conform cu precizările făcute pe foaia de concurs, numai seturi de date pentru care există soluție.

GInfo 12/6 octombrie 2002

Se observă foarte ușor că există doar nouă posibilități de lipire a patru fotografii pe o pagină, indiferent de dimensiunile acestora. Toate celelalte posibilități sunt echivalente cu una dintre cele nouă, ele obținându-se prin translațări sau oglindiri. Cele nouă aranjamente posibile pe care le vom lua în considerare sunt prezentate în figura alăturată.

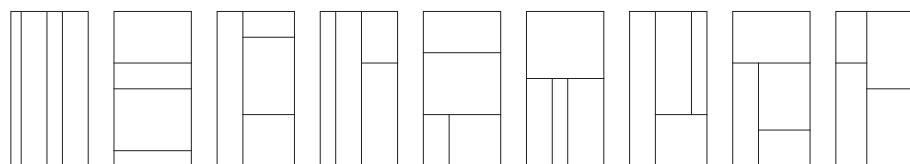


Figura 36.6: Foto

Va trebui să alegem pozițiile celor patru fotografii pentru fiecare dintre cele nouă configurații. Teoretic, pentru fiecare dintre cele nouă configurații, există $4! = 24$ posibilități. Practic, se observă că există și în acest caz configurații echivalente. De exemplu, pentru prima și a doua configurație, nu contează ordinea în care sunt așezate fotografii, pentru a treia configurație nu contează ordinea în care sunt dispuse cele trei poze de pe a doua coloană etc. Practic, pentru fiecare configurație vom avea 1, 2, 4, 6 sau 12 posibilități.

Cu excepția ultimei configurații, în toate celelalte există o fotografie care ocupă o întreagă latură a paginii. Cunoșcând raportul dintre lungimea și lățimea sa se determină laturile spațiului liber rămas pe foaie.

În continuare, una dintre laturile spațiului este ocupat integral de o fotografie.

Procedura continuă până determinăm dimensiunile tuturor fotografiilor.

În final se verifică dacă foaia este acoperită integral. Pentru ultima configurație vom alege, pe rând, dimensiunile posibile ale fotografiei din colțul din stânga sus și dimensiunile celorlalte fotografii vor fi determinate în același mod ca și în situațiile anterioare.

Analiza complexității

Citirea datelor se realizează în timp constant, deoarece numărul fotografilor este întotdeauna 4.

Studierea primelor opt configurații se realizează de asemenea în timp constant deoarece numărul posibilităților de amplasare a fotografilor respectând configurațiile considerate este constant.

Pentru cea de-a nouă configurație putem avea cel mult $\min(X, Y)$ dimensiuni posibile ale primei fotografii, unde X și Y sunt dimensiunile paginii. Dacă notăm cu n acest minim, ordinul de complexitate al acestei operații va fi $O(n)$.

Datele de ieșire constau în exact opt numere, deci scrierea rezultatelor se realizează în timp *constant*.

În concluzie, algoritmul de rezolvare a acestei probleme are ordinul de complexitate $O(1) + 8O(1) + O(n) + O(1) = O(n)$.

36.4.2 Rezolvare detaliată

Varianta 1:

Listing 36.4.1: foto1.java

```

1 import java.io.*; // nu merg: 5, 6, 8 si 9 (==> 60 pct varianta asta !)
2 class Foto
3 {
4     static int xp,yp; // dimensiuni pagina
5     static int[] xf=new int[5]; // dimensiuni figură
6     static int[] yf=new int[5];
7     static int[] fs=new int[5]; // factor scalare
8     static boolean[] eFixata=new boolean[5]; // figura fixata in pagina
9
10    static boolean ok;
11
12    public static void main(String[] args) throws IOException
13    {
14        int i;
15        StreamTokenizer st=new StreamTokenizer(
16            new BufferedReader(new FileReader("foto0.in")));
17        st.nextToken(); xp=(int)st.nval;
18        st.nextToken(); yp=(int)st.nval;
19
20        for(i=1;i<=4;i++)
21        {
22            st.nextToken(); xf[i]=(int)st.nval;
23            st.nextToken(); yf[i]=(int)st.nval;
24            fs[i]=cmmdc(xf[i],yf[i]);
25            xf[i]/=fs[i]; // dimensiuni micsorate ...
26            yf[i]/=fs[i];
27        }
28
29        ok=false;
30        pagina(4,xp,yp);
31
32        PrintWriter out=new PrintWriter(
33            new BufferedWriter(new FileWriter("foto.out")));
34
35        if(ok)
36            for(i=1;i<=4;i++) out.println(xf[i]*fs[i]+" "+yf[i]*fs[i]);
37            else out.println("???");
38        out.close();
39    } // main...
40
41    static void pagina(int nf, int xpr, int ypr)
42    {
43        // xpr=x pagina ramasa, nf=nr figură de plasat
44        int i,d;
45        int xprr=xpr;
46
47        if(ok) return;
48
49        if(nf==1)
50        {
51            i=1;
52            while(eFixata[i]) i++;
53            d=cmmdc(xpr,ypr);

```

```

54         if((xpr/d==xf[i])&&(ypr/d==yf[i]))
55     {
56         eFixata[i]=true;
57         ok=true;
58         fs[i]=xpr/xf[i];
59     }
60     return;
61 }
62
63 // cel putin 2 figurile ramase de plasat
64 for(i=1;i<=4;i++)
65 {
66     if(eFixata[i]) continue;
67
68     fs[i]=xpr/xf[i];           // in avans ... dar ...
69     if( (xpr%xf[i]==0)&&      // poate acoperi x
70         (fs[i]*yf[i]<ypr))   // ramane y
71     {
72         eFixata[i]=true;
73         pagina(nf-1,xpr, ypr-fs[i]*yf[i]);
74         if(ok) return; else eFixata[i]=false;
75     }
76
77     fs[i]=ypr/yf[i];          // in avans ... dar ...
78     if( (ypr%yf[i]==0)&&      // poate acoperi y
79         (fs[i]*xf[i]<xpr))   // ramane x
80     {
81         eFixata[i]=true;
82         pagina(nf-1,xpr-fs[i]*xf[i], ypr);
83         if(ok) return; else eFixata[i]=false;
84     }
85 } //for i
86 } // pagina(...)

87 static int cmmdc(int a, int b)
88 {
89     int r;
90     while(a%b!=0) { r=a%b; a=b; b=r; }
91     return b;
92 }
93 } // class

```

Varianta 2:

Listing 36.4.2: foto2.java

```

1 import java.io.*;    // nu merg testele: 5, 9 (==> 80 pct varianta asta !)
2 class Foto
3 {
4     static int xp,yp;           // dimensiuni pagina
5     static int[] xf=new int[5]; // dimensiuni figurile
6     static int[] yf=new int[5];
7     static int[] fs=new int[5]; // factorul scalar
8     static boolean[] eFixata=new boolean[5]; // figura fixata in pagina
9
10    static boolean ok;
11
12    public static void main(String[] args) throws IOException
13    {
14        int i;
15        StreamTokenizer st=new StreamTokenizer(
16            new BufferedReader(new FileReader("foto5.in")));
17        st.nextToken(); xp=(int)st.nval;
18        st.nextToken(); yp=(int)st.nval;
19
20        for(i=1;i<=4;i++)
21        {
22            st.nextToken(); xf[i]=(int)st.nval;
23            st.nextToken(); yf[i]=(int)st.nval;
24            fs[i]=cmmdc(xf[i],yf[i]);
25            xf[i]/=fs[i];           // dimensiuni micsorate ...
26            yf[i]/=fs[i];
27        }
28
29        ok=false;
30        pagina(4,xp,yp);

```

```

31
32     PrintWriter out=new PrintWriter(
33         new BufferedWriter(new FileWriter("foto.out")));
34
35     if(!ok) test1(3,2,1); if(!ok) test1(2,1,3);if(!ok) test1(1,2,3);
36     if(ok)
37         for(i=1;i<=4;i++) out.println(xf[i]*fs[i]+" "+yf[i]*fs[i]);
38         else out.println("???");
39     out.close();
40 } // main(...)

41
42 static void test1(int i3, int i2, int i1) // figura i3 pe verticala cu 4
43 {                                         // figura i2 pe verticala cu i1
44     int x,y,x1,x2,x3,x4,y1,y2,y3,y4,a1,a2,a3,a4;
45     x=xp; y=yp;
46     x1=xf[i1]; x2=xf[i2]; x3=xf[i3]; x4=xf[4];
47     y1=yf[i1]; y2=yf[i2]; y3=yf[i3]; y4=yf[4];
48
49     if((x4*y)% (x3*y4+x4*y3)==0) a3=(x4*y) / (x3*y4+x4*y3); else return;
50     if((x3*a3)%x4==0) a4=(x3*a3)/x4; else return;
51     if(!(a3*x3<x)) return;
52     if((x-a3*x3)%x1==0) a1=(x-a3*x3)/x1; else return;
53     if((x-a3*x3)%x2==0) a2=(x-a3*x3)/x2; else return;
54     if(a1*y1+a2*y2!=y) return;
55     ok=true;
56     fs[i1]=a1; fs[i2]=a2; fs[i3]=a3; fs[4]=a4;
57 } // test1(...)

58
59 static void pagina(int nf, int xpr, int ypr)
60 {
61     // xpr=x pagina ramasa, nf=nr figuri de plasat
62     int i,d;
63     int xprr=xpr;
64
65     if(ok) return;
66     if(nf==1)
67     {
68         i=1;
69         while(eFixata[i]) i++;
70         d=cmmdc(xpr,ypr);
71         if((xpr/d==xf[i])&&(ypr/d==yf[i]))
72         {
73             eFixata[i]=true;
74             ok=true;
75             fs[i]=xprr/xf[i];
76         }
77         return;
78     }
79
80     // cel putin 2 figuri ramase de plasat
81     for(i=1;i<=4;i++)
82     {
83         if(eFixata[i]) continue;
84         fs[i]=xpr/xf[i];           // in avans ... dar ...
85         if( (xpr%xf[i]==0)&&          // poate acoperi x
86             (fs[i]*yf[i]<ypr))        // ramane y
87         {
88             eFixata[i]=true;
89             pagina(nf-1,xpr, ypr-fs[i]*yf[i]);
90             if(ok) return; else eFixata[i]=false;
91         }
92
93         fs[i]=ypr/yf[i];           // in avans ... dar ...
94         if( (ypr%yf[i]==0)&&          // poate acoperi y
95             (fs[i]*xf[i]<xpr))        // ramane x
96         {
97             eFixata[i]=true;
98             pagina(nf-1,xpr-fs[i]*xf[i], ypr);
99             if(ok) return; else eFixata[i]=false;
100        }
101    } //for i
102 } // pagina(...)

103
104 static int cmmdc(int a, int b)
105 {
106     int r;

```

```

107     while(a%b!=0) { r=a%b; a=b; b=r; }
108     return b;
109 } // cmmdc(...)
110 } // class

```

Varianta 3:

Listing 36.4.3: foto3.java

```

1 import java.io.*;      // merg toate testele ==> 100 pct varianta asta !
2 class Foto
3 {
4     static int xp,yp;           // dimensiuni pagina
5     static int[] xf=new int[5]; // dimensiuni figuri
6     static int[] yf=new int[5];
7     static int[] fs=new int[5]; // factor scalare
8     static boolean[] eFixata=new boolean[5]; // figura fixata in pagina
9
10    static boolean ok;
11
12    public static void main(String[] args) throws IOException
13    {
14        int i;
15        StreamTokenizer st=new StreamTokenizer(
16            new BufferedReader(new FileReader("foto.in")));
17        st.nextToken(); xp=(int)st.nval;
18        st.nextToken(); yp=(int)st.nval;
19        for(i=1;i<=4;i++)
20        {
21            st.nextToken(); xf[i]=(int)st.nval;
22            st.nextToken(); yf[i]=(int)st.nval;
23            fs[i]=cmmdc(xf[i],yf[i]);
24            xf[i]/=fs[i];           // dimensiuni micsorate ...
25            yf[i]/=fs[i];
26        }
27        ok=false;
28        pagina(4,xp,yp);
29        if(!ok) test1(3,2,1); if(!ok) test1(2,1,3); if(!ok) test1(1,2,3);
30        if(!ok) test2(3,2,1); if(!ok) test2(2,1,3); if(!ok) test2(1,2,3);
31
32        PrintWriter out=new PrintWriter(
33            new BufferedWriter(new FileWriter("foto.out")));
34        if(ok) for(i=1;i<=4;i++) out.println(xf[i]*fs[i]+" "+yf[i]*fs[i]);
35        else out.println("???");
36        out.close();
37    } // main...
38
39    static void test1(int i3, int i2, int i1) // figura i3 pe verticala cu 4
40    {                                     // figura i2 pe verticala cu i1
41        int x,y,x1,x2,x3,x4,y1,y2,y3,y4,a1,a2,a3,a4;
42        x=xp; y=yp;
43        x1=xf[i1]; x2=xf[i2]; x3=xf[i3]; x4=xf[4];
44        y1=yf[i1]; y2=yf[i2]; y3=yf[i3]; y4=yf[4];
45        if((x4*y)% (x3*y4+x4*y3)==0) a3=(x4*y)/(x3*y4+x4*y3); else return;
46        if((x3*a3)%x4==0) a4=(x3*a3)/x4; else return;
47        if(!(a3*x3<x)) return;
48        if((x-a3*x3)%x1==0) a1=(x-a3*x3)/x1; else return;
49        if((x-a3*x3)%x2==0) a2=(x-a3*x3)/x2; else return;
50        if(a1*y1+a2*y2!=y) return;
51        ok=true; fs[i1]=a1; fs[i2]=a2; fs[i3]=a3; fs[4]=a4;
52    } // test1...
53
54    static void test2(int i3, int i2, int i1) // figura i3 pe orizontala cu 4
55    {                                     // figura i2 pe orizontala cu i1
56        int x,y,x1,x2,x3,x4,y1,y2,y3,y4,a1,a2,a3,a4;
57        x=xp; y=yp;
58        x1=xf[i1]; x2=xf[i2]; x3=xf[i3]; x4=xf[4];
59        y1=yf[i1]; y2=yf[i2]; y3=yf[i3]; y4=yf[4];
60
61        // simetric cu test1: x<-->y
62        if((y4*x)% (y3*x4+y4*x3)==0) a3=(y4*x)/(y3*x4+y4*x3); else return;
63        if((y3*a3)%y4==0) a4=(y3*a3)/y4; else return;
64        if(!(a3*y3<y)) return;
65        if((y-a3*y3)%y1==0) a1=(y-a3*y3)/y1; else return;
66        if((y-a3*y3)%y2==0) a2=(y-a3*y3)/y2; else return;
67        if(a1*x1+a2*x2!=x) return;

```

```

68     ok=true; fs[i1]=a1; fs[i2]=a2; fs[i3]=a3; fs[4]=a4;
69 } // test2(...)
70
71 static void pagina(int nf, int xpr, int ypr)
72 {
73     // xpr=x pagina ramasa, nf=nr figuri de plasat
74     int i,d;
75     int xprr=xpr;
76     if(ok) return;
77     if(nf==1)
78     {
79         i=1;
80         while(eFixata[i]) i++;
81         d=cmmdc(xpr,ypr);
82         if((xpr/d==xf[i])&&(ypr/d==yf[i]))
83         {
84             eFixata[i]=true;
85             ok=true;
86             fs[i]=xprr/xf[i];
87         }
88         return;
89     }
90     // cel putin 2 figuri ramase de plasat
91     for(i=1;i<=4;i++)
92     {
93         if(eFixata[i]) continue;
94         fs[i]=xpr/xf[i];           // in avans ... dar ...
95         if( (xpr%xf[i]==0)&&      // poate acoperi x
96             (fs[i]*yf[i]<ypr))    // ramane y
97         {
98             eFixata[i]=true;
99             pagina(nf-1,xpr, ypr-fs[i]*yf[i]);
100            if(ok) return; else eFixata[i]=false;
101        }
102        fs[i]=ypr/yf[i];          // in avans ... dar ...
103        if( (ypr%yf[i]==0)&&      // poate acoperi y
104            (fs[i]*xf[i]<xpr))    // ramane x
105        {
106            eFixata[i]=true;
107            pagina(nf-1,xpr-fs[i]*xf[i], ypr);
108            if(ok) return; else eFixata[i]=false;
109        }
110    } //for i
111 } // pagina(...)

112 static int cmmdc(int a, int b)
113 {
114     int r;
115     while(a%b!=0) { r=a%b; a=b; b=r; }
116     return b;
117 } // cmmdc(...)
118 } // class

```

36.4.3 Cod sursă

Listing 36.4.4: foto.pas

```

1 const er=0.000001;
2 type foto=record
3     l,h,lfin,hfin:integer;
4     end;
5 var l,h:integer;
6     a,b:array[1..4] of foto;
7     g:text;
8     x:array[1..4] of 1..2;
9     ok:boolean;
10
11 procedure date;
12 var f:text;
13     i:integer;
14 begin

```

```

15  assign(f,'foto.in');
16  reset(f);
17  readln(f,l,h);
18  for i:=1 to 4 do
19    readln(f,a[i].l,a[i].h);
20  close(f);
21  assign(g,'foto.out');
22  rewrite(g);
23 end;
24
25 procedure afisare;
26 var i:integer;
27 begin
28 if not ok then
29   for i:=1 to 4 do
30     writeln(g,b[i].lfin,' ',b[i].hfin);
31 end;
32
33 function cont(kk:integer):boolean;
34 var t1,q:real;
35   i,k:integer;
36   ok:boolean;
37 begin
38   cont:=true;
39   for i:=1 to kk-1 do
40     if abs(x[i])=abs(x[kk]) then begin
41       cont:=false;
42       exit;
43     end;
44   cont:=false;
45   k:=x[kk];
46   t1:=l/a[k].l;q:=t1*a[k].h;
47   ok:=true;
48   if (abs(trunc(q)-q)>er) or (l=0) then ok:=false
49   else
50     if t1*a[k].h<=h then
51       begin
52         ok:=true;
53         b[k].hfin:=trunc(q);
54         b[k].lfin:=l;
55         h:=h-b[k].hfin;
56       end
57     else ok:=false;
58   if not ok then
59   begin
60     t1:=h/a[k].h;q:=t1*a[k].l;
61     if (abs(trunc(q)-q)>er) or (h=0) then ok:=false
62     else
63       if t1*a[k].l<=l then
64         begin
65           ok:=true;
66           b[k].lfin:=trunc(q);
67           b[k].hfin:=h;
68           l:=l-b[k].lfin;
69           x[kk]:=-x[kk];
70         end
71       else
72         ok:=false;
73   end;
74   cont:=ok;
75   if kk=4 then if (l=0) or (h=0) then cont:=true
76   else if (l<>0) and (h<>0) then begin
77     cont:=false;
78     if ok then
79       if (x[kk]>0) then h:=h+b[k].hfin
80       else l:=l+b[k].lfin;
81     end;
82 end;
83
84 procedure back(k:integer);
85 var i:integer;
86 begin
87   if k=5 then begin
88     afisare;
89     ok:=true;
90   end

```

```

91           else
92             for i:=1 to 4 do
93               begin
94                 x[k]:=i;
95                 if cont(k) then
96                   begin
97                     back(k+1);
98                     if x[k]>0 then h:=h+b[x[k]].hfin
99                     else l:=l+b[abs(x[k])].lfin;
100                end;
101               end;
102             end;
103
104 procedure schimba;
105 var r,i:integer;
106 begin
107   for i:=1 to 4 do
108     begin
109       r:=a[i].l;
110       a[i].l:=a[i].h;
111       a[i].h:=r;
112       r:=a[i].lfin;
113       a[i].lfin:=a[i].hfin;
114       a[i].hfin:=r;
115     end;
116   end;
117
118 procedure pune_oriz(i,l,h:integer; var hf:integer;var ok:boolean);
119 var t1,q:real;
120 begin
121   t1:=l/a[i].l;q:=t1*a[i].h;
122   if (abs(trunc(q)-q)>er) or (l=0) then ok:=false
123   else
124     if t1*a[i].h<=h then
125       begin
126         ok:=true;
127         hf:=trunc(q);
128       end
129     else
130       ok:=false;
131   end;
132
133 procedure altfel(t,lf,hf:integer);
134 var q,i,j,k,ll:integer;
135   ok:boolean;
136   l1,h1,rap1:real;
137   utiliz:array[1..4] of boolean;
138 begin
139   for i:=1 to 4 do utiliz[i]:=false;
140   for i:=1 to 2 do
141   begin
142     utiliz[i]:=true;
143     for j:=1 to 4 do
144       if not utiliz[j] then
145         begin
146           utiliz[j]:=true;
147           rap1:=lf/(a[i].l+a[j].l*a[i].h/a[j].h);
148           l1:=a[i].l*rap1;
149           h1:=a[i].h*rap1;
150           if (l1-trunc(l1)<=er) and (h1-trunc(h1)<=er) then
151             begin
152               a[i].lfin:=trunc(l1);a[i].hfin:=trunc(h1);
153               pune_oriz(j,lf-a[i].lfin,a[i].hfin,a[j].hfin,ok);
154               if ok and (a[j].hfin=a[i].hfin)then
155                 begin
156                   a[j].lfin:=lf-a[i].lfin;
157                   hf:=hf-a[i].hfin;
158                   for k:=1 to 4 do
159                     if not utiliz[k] then
160                       begin
161                         utiliz[k]:=true;
162                         for ll:=1 to 4 do
163                           if not utiliz[ll] then
164                             begin
165                               utiliz[ll]:=true;
166                               rap1:=lf/(a[k].l+a[ll].l*a[k].h/a[ll].h);

```

```

167           l1:=a[k].l*rap1;
168           h1:=a[k].h*rap1;
169           if (l1-trunc(l1)<=er) and (h1-trunc(h1)<=er) then
170           begin
171               a[k].lfin:=trunc(l1);
172               a[k].hfin:=trunc(h1);
173               punе_ориз(l1,lf-a[k].lfin,
174                           a[k].hfin,a[ll].hfin,ok);
175               if ok and (a[k].hfin=a[ll].hfin) and
176                   (((a[i].hfin+a[k].hfin=h) and (t=1))
177                   or ((a[i].hfin+a[k].hfin=l) and (t=2))) then
178               begin
179                   a[ll].lfin:=lf-a[k].lfin;
180                   a[ll].hfin:=a[k].hfin;
181                   hf:=hf-a[k].hfin;
182                   {ok:=true;}
183                   if t=2 then schimba;
184                   for q:=1 to 4 do
185                       writeln(g,a[q].lfin, ' ', a[q].hfin);
186                   ok:=true;
187                   close(g);
188                   halt;
189               end;
190           end;
191
192           utiliz[ll]:=false;
193       end;
194
195           utiliz[k]:=false;
196       end;
197   end;
198
199           utiliz[j]:=false;
200   end;
201
202           utiliz[i]:=false;
203   end;
204   { ok:=false; }
205
206 end;
207
208 begin
209     date;
210     ok:=false;
211     back(1);
212     if not ok then
213     begin
214         altfel(1,l,h);
215         if not ok then
216             begin
217                 schimba;
218                 altfel(2,h,l);
219             end;
220         end;
221
222         close(g);
223     end.

```

36.5 Balanță

Mihai Stroe, București

Gigel are o "balanță" mai ciudată pe care vrea să o echilibreze. De fapt, aparatul este diferit de orice balanță pe care ati vazut-o până acum.

Balanța lui Gigel dispune de două brațe de greutate neglijabilă și lungime 15 fiecare. Din loc în loc, la aceste brațe sunt atașate cârlige, pe care Gigel poate atârna greutăți distincte din colecția sa de G greutăți (numere naturale între 1 și 25). Gigel poate atârna oricără greutăți de orice cărlig, dar trebuie să folosească toate greutățile de care dispune.

Folosindu-se de experiența participării la Olimpiada Națională de Informatică, Gigel a reușit să echilibreze balanța relativ repede, dar acum dorește să știe în căte moduri poate fi ea echilibrată.

Cerință

Cunoscând amplasamentul cârligelor și setul de greutăți pe care Gigel îl are la dispoziție, scrieți un program care calculează în câte moduri se poate echilibra balanța.

Se presupune că este posibil să se echilibreze balanța (va fi posibil pe toate testele date la evaluare).

Datele de intrare

Fișierul de intrare **balanta.in** are următoarea structură:

- pe prima linie, numărul C de cârlige și numărul G de greutăți, valori separate prin spațiu;
- pe următoarea linie, C numere întregi, distințe, separate prin spațiu, cu valori cuprinse între -15 și 15 inclusiv, reprezentând amplasamentele cârligelor față de centrul balanței; valoarea absolută a numerelor reprezintă distanța față de centrul balanței, iar semnul precizează brațul balanței la care este atașat cârligul, $"+"$ pentru brațul stâng și $"+"$ pentru brațul drept;
- pe următoarea linie, G numere naturale distințe, cuprinse între 1 și 25 inclusiv, reprezentând valorile greutăților pe care Gigel le va folosi pentru a echilibra balanța.

Datele de ieșire

Fișierul de ieșire **balanta.out** conține o singură linie, pe care se află un număr natural M , numărul de variante de plasare a greutăților care duc la echilibrarea balanței.

Restricții și precizări

- $2 \leq C \leq 20$, $2 \leq G \leq 20$;
- greutățile folosite au valori naturale între 1 și 25 ;
- numărul M cerut este între 1 și $100.000.000$;
- celelalte restricții (lungimea brațelor balanței etc.) au fost prezentate anterior.
- balanța se echilibrează dacă suma produselor dintre greutăți și coordonatele unde ele sunt plasate este 0 (suma momentelor greutăților față de centrul balanței este 0).

Exemplu

balanta.in	balanta.out
2 4	2
-2 3	
3 4 5 8	

Timp maxim de executare: 1 secundă/test

36.5.1 Indicații de rezolvare

Soluția comisiei

Problema se rezolva prin *metoda programării dinamice*.

Se calculează în câte moduri se poate scrie fiecare sumă j , folosind primele i greutăți. Inițial $i = 0$ și suma 0 se poate obține într-un singur mod, restul sumelor în 0 moduri.

Urmeaza G pași. La fiecare astfel de pas i se calculează în câte moduri putem obține fiecare sumă introducând o nouă greutate - a i -a - în toate configurațiile precedente. Practic, dacă suma S s-a obținut cu primele $i - 1$ greutăți în M moduri, punând greutatea i pe cârligul k se va obține suma $S + (greutate[i] * coordonata[k])$ în M moduri (la care, evident, se pot adăuga alte moduri de obținere plasând greutatea i pe un alt cârlig și folosind suma respectivă).

În acest mod s-ar construi o matrice cu G linii și $2 * (sumamaxima) + 1$ coloane, cu elemente numere întregi pe 32 de biți; de fapt se memorează doar ultimele două linii (fiecare linie se obține din precedenta). Suma maximă este $15 * 25 * 20 = 7500$, deci o linie se incadrează în mai puțin de $64K$.

Rezultatul final se obține pe ultima linie, în coloana asociată sumei 0 .

GInfo 12/6 octombrie 2002

Se observă că suma momentelor greutăților este cuprinsă între -6000 și 6000 (dacă avem 20 de greutăți cu valoarea 20 și acestea sunt amplasate pe cel mai îndepărtat cârlig față de centrul balanței, atunci modulul sumei momentelor forțelor este $152020 = 6000$). Ca urmare, putem păstra un sir a ale căruia valori a_i vor conține numărul posibilităților ca suma momentelor greutăților să fie i .

Indicii sirului vor varia între -6000 și 6000 . Pentru a îmbunătăți viteza de execuție a programului, indicii vor varia între $-300g$ și $300g$, unde g este numărul greutăților. Pot fi realizate îmbunătățiri suplimentare dacă se determină distanțele maxime față de mijlocul balanței ale celor

mai îndepărtate cârlige de pe cele două talere și suma totală a greutăților. Dacă distanțele sunt d_1 și d_2 , iar suma este s , atunci indicii vor varia între $-d_1s$ și d_2s .

Initial, pe balanță nu este agățată nici o greutate, aşadar suma momentelor greutăților este 0. Ca urmare, initial valorile a_i vor fi 0 pentru orice indice nenul și $a_0 = 1$ (există o posibilitate ca initial suma momentelor greutăților să fie 0 și nu există nici o posibilitate ca ea să fie diferită de 0).

În continuare, vom încerca să amplasăm greutățile pe cârlige. Fiecare greutate poate fi amplasată pe oricare dintre cârlige. Să presupunem că la un moment dat există a_i posibilități de a obține suma i . Dacă vom amplasa o greutate de valoare g pe un cârlig aflat la distanța d față de centrul balanței, suma momentelor greutăților va crește sau va scădea cu gd (în funcție de brațul pe care se află cârligul). Ca urmare, după amplasarea noii greutăți există a_i posibilități de a obține suma $i + gd$. Considerăm că sirul b va conține valori care reprezintă numărul posibilităților de a obține sume ale momentelor forțelor după amplasarea greutății curente. Înainte de a testa posibilitățile de plasare a greutății, sirul b va conține doar zerouri. Pentru fiecare pereche (i, d) , valoarea b_{i+gd} va crește cu a_i . După considerarea tuturor perechilor, vom putea trece la o nouă greutate.

Valorile din sirul b vor fi salvate în sirul a , iar sirul b va fi reinitializat cu 0. Rezultatul final va fi dat de valoarea a_0 obținută după considerarea tuturor greutăților disponibile.

Analiza complexității

Pentru studiul complexității vom nota numărul greutăților cu g , iar cel al cârligelor cu c .

Citirea datelor de intrare corespunzătoare cârligelor și greutăților se realizează în timp liniar, deci ordinul de complexitate al acestor operații este $O(g)$, respectiv $O(c)$.

Singura mărime care nu este considerată constantă și de care depinde numărul de posibilități de a obține sumele este numărul greutăților. Așadar, vom considera că ordinul de complexitate al traversării sirului a este $O(g)$.

Numărul de traversări este dat tot de numărul greutăților disponibile, deci vom avea $O(g)$ traversări.

În timpul traversării vom considera toate cârligele pentru fiecare element al sirului. Ca urmare, ordinul de complexitate al operațiilor efectuate asupra unui element într-o parcurgere este $O(c)$. Rezultă că ordinul de complexitate al unei parcurgeri este $O(g)O(c) = O(gc)$, în timp ce ordinul de complexitate al întregii operații care duce la obținerea rezultatului este $O(g)O(gc) = O(g^2c)$.

Afișarea numărului de posibilități de a echilibra balanța se realizează în timp constant.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(c) + O(g) + O(g^2c) + O(1) = O(g^2c)$.

36.5.2 Rezolvare detaliată

Listing 36.5.1: balanta.java

```

1 import java.io.*;
2 class Balanta
3 {
4     static long[] a=new long[15001]; // a[i] i = -7500, 7500
5     static long[] b=new long[15001]; // sir auxiliar (a+7500)!!!
6     static int[] carlig=new int[20]; // coordonatele cârligelor
7     static int[] greutate=new int[20]; // valorile greutăților
8     static int nrCarlige, nrGreutati;
9
10    public static void main(String[] args) throws IOException
11    {
12        long t1,t2;
13        t1=System.currentTimeMillis();
14        citesteDatele();
15        determinaSolutia();
16        scrieSolutia();
17        t2=System.currentTimeMillis();
18        System.out.println("TIMP = "+(t2-t1)+" milisecunde");
19    } // main()
20
21    static void citesteDatele() throws IOException
22    {
23        StreamTokenizer st=new StreamTokenizer(
24            new BufferedReader(new FileReader("balanta9.in")));
25        st.nextToken(); nrCarlige=(int)st.nval;

```

```

26     st.nextToken(); nrGreutati=(int)st.nval;
27
28     for(int i=0;i<nrCarlige;i++)
29     {
30         st.nextToken(); carlige[i]=(int)st.nval;
31     }
32     for(int i=0;i<nrGreutati;i++)
33     {
34         st.nextToken(); greutate[i]=(int)st.nval;
35     }
36 } // citesteDate()
37
38 static void scrieSolutia() throws IOException
39 {
40     PrintWriter out=new PrintWriter(
41             new BufferedWriter(new FileWriter("balanta9.out")));
42     out.print(a[7500+0]);
43     out.close();
44 } // scrieSolutia()
45
46 static void determinaSolutia()
47 {
48     int i,j,k;
49     a[7500+0]=1; // initial balanta este echilibrata
50     for(i=0;i<nrGreutati;i++)
51     {
52         for(j=-7500;j<=7500;j++)
53             if(a[7500+j]!=0)
54                 for(k=0;k<nrCarlige;k++)
55                     b[7500+j+carlige[k]*greutate[i]]+=a[7500+j];
56         for (j=-7500;j<=7500;j++)
57         {
58             a[7500+j]=b[7500+j];
59             b[7500+j]=0;
60         }
61     }
62 } // determinaSolutia()
63 } // class

```

36.5.3 Cod sursă

Listing 36.5.2: balanta.pas

```

1 {$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R+,S+,T-,V+,X+,Y+}
2 {$M 16384,0,655360}
3 type ar=array[-8000..8000]of longint;
4 var i,j,k,l,m,n,c,g:longint;
5   fi,fo:text;
6   carlige,greutati:array[1..100]of longint;
7   a,b,aux:^ar;
8
9 procedure readdata;
10 begin
11 { writeln('Starting...');}
12 assign(fi,'balanta.in');
13 assign(fo,'balanta.out');
14 reset(fi);
15 readln(fi,c,g);
16 for i:=1 to c do
17   read(fi,carlige[i]);
18 readln(fi);
19 for i:=1 to g do
20   read(fi,greutati[i]);
21 readln(fi);
22 close(fi);
23 end;
24
25 procedure solve;
26 begin
27   new(a);
28   fillchar(a^,sizeof(a^),0);
29   new(b);

```

```

30     fillchar(b^, sizeof(b^), 0);
31
32     a^[0]:=1;
33     for k:=1 to g do
34         begin
35             for i:=-8000 to 8000 do
36                 if a^i<>0 then
37                     begin
38                         for j:=1 to c do
39                             begin
40                                 if b^i+carlige[j]*greutati[k]+a^i>=100000000 then
41                                     begin
42                                         writeln('Invalid test case');
43                                         halt;
44                                     end;
45                                     inc(b^i+carlige[j]*greutati[k], a^i);
46                                 end;
47                             end;
48                         aux:=b; b:=a; a:=aux;
49                         fillchar(b^, sizeof(b^), 0);
50                     end;
51     { writeln('OK'); }
52     rewrite(fo);
53     writeln(fo, a^0);
54     close(fo);
55 end;
56
57 begin
58     readdata;
59     solve;
60 end.

```

36.6 Aliniere

prof. Mot Nistor, Brăila

În armată, o companie este alcătuită din n soldați. La inspecția de dimineață soldații stau aliniati în linie dreaptă în fața căpitanului. Acesta nu e mulțumit de ceea ce vede; e drept că soldații sunt așezați în ordinea numerelor de cod 1, 2, ..., n din registru, dar nu în ordinea înălțimii. Căpitanul cere câtorva soldați să iasă din rând, astfel ca cei rămași, fără a-și schimba locurile, doar apropiindu-se unul de altul (pentru a nu rămâne spații mari între ei) să formeze un șir în care fiecare soldat vede privind de-a lungul șirului, cel puțin una din extremități (stânga sau dreapta). Un soldat vede o extremitate dacă între el și capătul respectiv nu există un alt soldat cu înălțimea mai mare sau egală ca a lui.

Cerință

Scrieți un program care determină, cunoscând înălțimea fiecărui soldat, numărul minim de soldați care trebuie să părăsească formația astfel ca șirul rămas să îndeplinească condiția din enunț.

Datele de intrare

Pe prima linie a fișierului de intrare **aliniere.in** este scris numărul n al soldaților din șir, iar pe linia următoare un șir de n numere reale, cu maximum 5 zecimale fiecare și separate prin spații. Al k -lea număr de pe această linie reprezintă înălțimea soldatului cu codul k ($1 \leq k \leq n$).

Datele de ieșire

Fișierul **aliniere.out** va conține pe prima linie numărul soldaților care trebuie să părăsească formația, iar pe linia următoare codurile acestora în ordine crescătoare, separate două câte două printr-un spațiu. Dacă există mai multe soluții posibile, se va scrie una singură.

Restricții și precizări

- $2 \leq n \leq 1000$
- înălțimile sunt numere reale în intervalul $[0.5; 2.5]$.

Exemplu

aliniere.in	aliniere.out
8	4
1.86 1.86 1.30621 2 1.4 1 1.97 2.2	1 3 7 8

Explicație

Rămân soldații cu codurile 2, 4, 5, 6 având înălțimile 1.86, 2, 1.4 și 1.
 Soldatul cu codul 2 vede extremitatea stângă.
 Soldatul cu codul 4 vede ambele extremități.
 Soldații cu codurile 5 și 6 văd extremitatea dreaptă.

Timp maxim de executare: 1 secundă/test

36.6.1 Indicații de rezolvare

Soluția comisiei

Problema se rezolvă prin metoda *programării dinamice*.

Se calculează, pentru fiecare element, lungimea celui mai lung subșir strict crescător care se termină cu el și lungimea celui mai lung subșir strict descrescător care începe cu el.

Solutia constă în păstrarea a două astfel de subșiruri de soldați (unul crescător și unul descrescător) pentru DOI soldați de aceeași înălțime (eventual identici) și eliminarea celorlalți. Soldații din primul subșir privesc spre stanga, ceilalți spre dreapta. Primul subșir se termină înainte de a începe al doilea. Se au în vedere cazurile particulare.

Deoarece s-a considerat că o parte din concurenți vor rezolva problema pentru un singur soldat central (toți ceilalți soldați păstrati având înălțimea mai mică) și nu vor observa cazul în care se pot păstra doi soldați de aceeași înălțime, majoritatea testelor se încadrează în acest caz.

GInfo 12/6 octombrie 2002

Pentru fiecare soldat vom determina cel mai lung subșir strict crescător (din punct de vedere al înălțimii) de soldați care se termină cu el, respectiv cel mai lung subșir strict descrescător de soldați care urmează după el.

După această operație, vom determina soldatul pentru care suma lungimilor celor două șiruri este maximă. Chiar dacă s-ar părea că în acest mod am găsit soluția problemei, mai există o posibilitate de a mări numărul soldaților care rămân în șir. Să considerăm soldatul cel mai înalt în șirul rămas (cel căruia îi corespunde suma maximă). Acesta poate privi fie spre stânga, fie spre dreapta șirului. Din aceste motive, la stânga sau la dreapta sa poate să se afle un soldat de aceeași înălțime; unul dintre cei doi va privi spre dreapta, iar celălalt spre stânga. Totuși, nu putem alege orice soldat cu aceeași înălțime, ci doar unul pentru care lungimea șirului strict crescător (dacă se află spre stânga) sau a celui strict descrescător (dacă se află spre dreapta) este aceeași cu lungimea corespunzătoare șirului strict crescător, respectiv strict descrescător, corespunzătoare celui mai înalt soldat dintre cei rămași în șir.

După identificarea celor doi soldați de înălțimi egale (sau demonstrarea faptului că nu există o pereche de acest gen care să respecte condițiile date) se marchează toți soldații din cele două subșiruri. Ceilalți soldați vor trebui să părăsească formăția.

Analiza complexității

Citirea datelor de intrare se realizează în timp liniar, deci ordinul de complexitate al acestei operații este $O(n)$.

Chiar dacă există algoritmi eficienți (care rulează în timp liniar-logaritmic) de determinare a celui mai lung subșir ordonat, timpul de execuție admis ne permite folosirea unui algoritm simplu, cu ordinul de complexitate $O(n^2)$. Aceasta va fi aplicat de două ori, după care se va căuta valoarea maximă a sumei lungimilor șirurilor corespunzătoare unui soldat; aşadar identificarea soldatului care poate privi în ambele direcții este o operație cu ordinul de complexitate $O(n^2) + O(n^2) + O(n) = O(n^2)$.

Urmează eventuala identificare a unui alt soldat de aceeași înălțime care respectă condițiile referitoare la lungimile subșirurilor. Pentru aceasta se parcurge șirul soldaților de la soldatul identificat anterior spre extremități; operația necesită un timp liniar.

Determinarea celor două subșiruri se realizează în timp liniar dacă, în momentul construirii celor două subșiruri, se păstrează predecesorul, respectiv succesorul fiecărui soldat. În timpul parcurgerii subșirurilor sunt marcați soldații care rămân în formăție.

Pentru scrierea datelor de ieșire se parcurge șirul marcajelor și sunt identificați soldații care părăsesc formăția. Ordinul de complexitate al acestei operații este $O(n)$.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(n) + O(n^2) + O(n) + O(n) + O(n) = O(n^2)$.

36.6.2 Rezolvare detaliată

Listing 36.6.1: aliniere.java

```

1 import java.io.*;
2 class Aliniere
3 {
4     static final String fisi="aliniere.in";
5     static float[] inalt;      // inaltimile soldatilor
6     static int ns, nsr;        // nr soldati, nr soldati ramasi
7     static int[] predCresc;   // predecesor in subsirul crescator
8     static int[] lgCresc;     // lungimea sirului crescator care se termina cu i
9     static int[] succDesc;    // succesor in subsirul descrescator
10    static int[] lgDesc;      // lungimea sirului descrescator care urmeaza dupa i
11    static boolean[] ramas;   // ramas in sir
12
13    public static void main(String[] args) throws IOException
14    {
15        long t1,t2;
16        t1=System.currentTimeMillis();
17        citescDate();
18        subsirCresc();
19        subsirDesc();
20        alegeSoldati();
21        scrieRezultate();
22        t2=System.currentTimeMillis();
23        System.out.println("TIME = "+(t2-t1)+" millisec ");
24    } // main()
25
26    static void citescDate() throws IOException
27    {
28        StreamTokenizer st=new StreamTokenizer(
29                            new BufferedReader(new FileReader(fisi)));
30        st.nextToken(); ns=(int)st.nval;
31
32        predCresc=new int[ns];
33        lgCresc=new int[ns];
34        succDesc=new int[ns];
35        lgDesc=new int[ns];
36        ramas=new boolean[ns];
37        inalt=new float[ns];
38
39        for(int i=0;i<ns;i++) {st.nextToken(); inalt[i]=(float)st.nval;}
40    } //citescDate()
41
42    static void subsirCresc()
43    {
44        int i,j;
45        lgCresc[0]=1;
46        predCresc[0]=-1;
47        for(i=1;i<ns;i++)
48        {
49            lgCresc[i]=1;           // subsirul formar doar din i
50            predCresc[i]=-1;       // nu are predecesor
51            for (int j=0;j<i;j++)
52                if(inalt[j]<inalt[i])
53                    if(lgCresc[i]<lgCresc[j]+1) // sir mai lung
54                    {
55                        lgCresc[i]=lgCresc[j]+1;
56                        predCresc[i] = j;
57                    }
58        }
59    } //subsrirCresc()
60
61    static void subsirDesc()
62    {
63        int i,j;
64        lgDesc[ns-1]=0;          // nu exista nici un soldat mai mic dupa ns-1
65        succDesc[ns-1]=-1;        // ns-1 nu are succesor
66        for(i=ns-2;i>=0;i--)
67        {
68            lgDesc[i]=0;           // nu exista nici un soldat mai mic dupa i
69            succDesc[i]=-1;         // i nu are succesor
70            for(j=ns-1;j>i;j--)

```

```

71         if(inalt[j]<inalt[i])           // soldat mai mic
72             if(lgDesc[i]<lgDesc[j]+1)    // sir mai lung
73             {
74                 lgDesc[i]=lgDesc[j]+1;   // actualizarea lg subsir
75                 succDesc[i]=j;        // actualizare successor
76             }
77     }
78 } // subsirDesc()

79
80 static void alegeSoldati()
81 {
82     int i;
83     // este posibil ca in mijloc sa fie doi soldati cu inaltime egale
84     int im=-1;           // indicele soldatului din mijloc
85     int ic, id;         // indicii care delimitaza in interior cele doua subsiruri
86     for(i=0;i<ns;i++)
87     if(lgCresc[i]+lgDesc[i]>nsl)
88     {
89         nsr=lgCresc[i]+lgDesc[i];
90         im=i;
91     }
92
93     // in "mijlocul" sirului se pot afla doi soldati cu aceeasi inaltime
94     ic=im;
95     id=im;
96
97     // caut in stanga un subsir cu aceeasi lungime --> soldat cu aceeasi inaltime
98     for(i=im-1;i>=0;i--)
99     if(lgCresc[ic]==lgCresc[i]) ic=i;
100
101    // caut in dreapta un subsir cu aceeasi lungime --> soldat cu aceeasi inaltime
102    for(i=im+1;i<ns;i++)
103    if(lgDesc[id]==lgDesc[i]) id=i;
104    if(ic!=id)           // in "mijloc" sunt doi soldati cu aceeasi inaltime
105    nsr++;
106    while(id!=-1)       // indice descrescator
107    {
108        ramas[id]=true;
109        id=succDesc[id];
110    }
111    while(ic!=-1)       // indice crescator
112    {
113        ramas[ic] = true;
114        ic=predCresc[ic];
115    }
116 } // alegeSoldati()
117
118 static void scrieRezultate() throws IOException
119 {
120     PrintWriter out=new PrintWriter(
121                 new BufferedWriter(new FileWriter("aliniere.out")));
122     out.println(ns- nsr);
123     for(int i=0;i<ns;i++) if(!ramas[i]) out.print((i+1)+" ");
124     out.close();
125 } // scrieRezultate()
126 } // class

```

36.6.3 Cod sursă

Listing 36.6.2: aliniere.pas

```

1 {$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S-,T-,V+,X+,Y+}
2 {$M 16384,0,655360}
3 var optim,x1,x2,rez,caz,i,j,k,l,m,n:longint;
4   fi,fo:text;
5   a:array[1..2048]of longint;
6   r:real;
7   left,right,t1,t2,sel:array[1..2048]of integer;
8   t12:longint absolute $0:$46c;
9
10 procedure readdata;
11 begin

```

```

12 assign(fi,'aliniere.in');
13 assign(fo,'aliniere.out');
14 reset(fi);
15 readln(fi,n);
16 for i:=1 to n do
17 begin
18   read(fi,r);
19   a[i]:=round(r*100000);
20 end;
21 close(fi);
22 end;
23
24 procedure subsir1;
25 begin
26 for i:=1 to n do
27 begin
28   left[i]:=1;
29   t1[i]:=0;
30   for j:=1 to i-1 do
31     if (a[j]<a[i])and(left[j]+1>left[i]) then
32     begin
33       left[i]:=left[j]+1;
34       t1[i]:=j;
35     end;
36   end;
37 end;
38
39 procedure subsir2;
40 begin
41 for i:=n downto 1 do
42 begin
43   right[i]:=1;
44   t2[i]:=0;
45   for j:=i+1 to n do
46     if (a[j]<a[i])and(right[j]+1>right[i]) then
47     begin
48       right[i]:=right[j]+1;
49       t2[i]:=j;
50     end;
51   end;
52 end;
53
54 procedure check;
55 begin
56   optim:=0;x1:=0;x2:=0;caz:=0;
57   for i:=1 to n do
58     for j:=i to n do
59       if a[i]=a[j] then
60         begin
61           rez:=left[i]+right[j];
62           if i=j then
63             rez:=rez-1;
64           if rez>optim then
65             begin
66               x1:=i;
67               x2:=j;
68               optim:=rez;
69               if i=j then caz:=1;
70             end;
71         end;
72   end;
73
74 procedure afis;
75 begin
76   rewrite(fo);
77   writeln(fo,n-optim);
78
79   repeat
80     sel[x1]:=1;
81     x1:=t1[x1];
82   until x1=0;
83
84   repeat
85     sel[x2]:=1;
86     x2:=t2[x2];
87   until x2=0;

```

```
88
89   for i:=1 to n do
90     if sel[i]=0 then
91       write(fo,i,' ');
92     writeln(fo);
93     close(fo);
94   end;
95
96 procedure solve;
97 begin
98   subsir1;
99   subsir2;
100
101  check;
102  afis;
103 end;
104
105 begin
106   writeln(tl2);
107   readdata;
108   solve;
109   writeln(tl2);
110 end.
```

Capitolul 37

ONI 2001



Figura 37.1: Sigla ONI 2001

37.1 Alpinistul

Bogdan Stroe, Bucureşti

Un alpinist ar vrea să cucerească un vârf cât mai înalt din Munții Carpați. Din păcate îi este frică să urce de pe o stâncă pe alta alăturată, dacă diferența de altitudine depășește 100 de metri. În schimb el coboară fără frică de pe o stâncă pe alta alăturată, indiferent de diferența de altitudine.

Alpinistul are harta muntelui pe care vrea să-l escaladeze. Harta este codificată sub forma unui carioaj în care în pătrățele sunt notate altitudinile punctelor (înălțimile stâncilor) de pe hartă, date în metri față de nivelul mării. Alpinistul poate porni din orice pătrățel de pe hartă cu altitudine 0 (aflat la nivelul mării) și poate efectua un pas doar într-o porțiune de teren corespunzătoare unui pătrățel de pe hartă, învecinat pe verticală sau pe orizontală cu pătrățelul în care se află, cu condiția să nu îi fie frică.

Cerință

Alpinistul apelează la ajutorul vostru pentru a afla traseul de lungime minimă pe care trebuie să-l urmeze pentru a escalada un vârf cât mai înalt.

Date de intrare

Fișier de intrare: ALPINIST.IN

Linia 1: M N două numere naturale nenule, separate printr-un spațiu, reprezentând dimensiunile caroiajului corespunzător hărții;

Liniile 2..M + 1: v1 v2 Ě vN pe aceste M linii sunt scrise câte N numere naturale, separate prin câte un spațiu, reprezentând valorile asociate caroiajului care codifică harta (linie după linie).

Date de ieșire

Fișier de ieșire: ALPINIST.OUT

Linia 1: I număr natural ce reprezintă înălțimea maximă la care poate ajunge alpinistul;

Linia 2: XP YP XD YD patru numere naturale nenule, separate prin câte un spațiu, reprezentând coordonatele pătrățelului din care pleacă alpinistul și coordonatele pătrățelului având

înălțimea maximă în care poate ajunge; prin coordonatele pătrățelului se înțeleg numărul liniei și cel al coloanei pe care se află pătrățelul în caroaj;

Linia 3: L – număr natural reprezentând lungimea drumului; lungimea unui drum se definește ca fiind egal cu numărul pătrățelelor străbătute de alpinist;

Linia 4: $D_1 \ D_2 \dots \ D_L$ – L caractere, separate prin căte un spațiu, reprezentând direcția în care se mișcă alpinistul la fiecare pas i , $i = 1, 2, \dots, L$; caracterele pot fi: 'N'- corespunzător unei mișcări în sus, 'S'- corespunzător unei mișcări în jos, 'W'- corespunzător unei mișcări la stânga, 'E'- corespunzător unei mișcări la dreapta.

Restricții și precizări

$2 \leq M, N \leq 100$

$0 \leq v_i \leq 1000 \ i = 1, 2, \dots, N$ (pe fiecare din cele M linii)

dacă sunt mai multe drumuri de lungime minimă, în fișier se va scrie unul singur.

Exemplu

ALPINIST.IN	ALPINIST.OUT (un conținut posibil)
3 5	250
0 101 2 14 100	1 1 2 4
50 149 149 250 200	8
0 100 10 400 10	S E E N E E S W

Timp maxim de executare/test: 1 secundă

37.1.1 *Indicații de rezolvare

37.1.2 Rezolvare detaliată

Varianta 1:

Listing 37.1.1: alpinist1.java

```

1 import java.io.*; // cu mesaje pentru depanare
2 class Alpinist // coada circulara + traseu fara recursivitate
3 {
4     static final int oo=Integer.MAX_VALUE;
5     static final int qdim=200;
6     static final int sus=1, dreapta=2, jos=3, stanga=4;
7     static int[][] a=new int[100][100]; // muntele (harta)
8     static int[][] c=new int[100][100]; // cost deplasare in (i,j)
9     static byte[][] d=new byte[100][100]; // directii pentru intoarcere
10
11    static int m,n,istart,jstart,ifinal,jfinal,lgmin;
12    static int[] qi=new int[qdim]; // coada pentru i din pozitia (i,j)
13    static int[] qj=new int[qdim]; // coada pentru j din pozitia (i,j)
14    static int ic, sc;
15
16    public static void main(String[] args) throws IOException
17    {
18        long t1,t2;
19        t1=System.currentTimeMillis();
20        int i,j;
21        citescDate();
22        for(i=0;i<m;i++)
23            for(j=0;j<n;j++)
24                if(a[i][j]==0) matriceCosturi(i,j);
25        afisSolutia();
26        t2=System.currentTimeMillis();
27        System.out.println("TIME = "+(t2-t1)+" millisec ");
28    } // main()
29
30    static void citescDate() throws IOException
31    {
32        int i,j;
33        StreamTokenizer st=new StreamTokenizer(
34            new BufferedReader(new FileReader("alpinist.in")));
35        st.nextToken(); m=(int)st.nval;
36        st.nextToken(); n=(int)st.nval;
37
38        for(i=0;i<m;i++)

```

```

39         for(j=0;j<n;j++)
40         {
41             c[i][j]=oo;
42             st.nextToken(); a[i][j]=(int)st.nval;
43         }
44     } //citescDate()
45
46     static void matriceCosturi(int i0, int j0)
47     {
48         System.out.println("Plec din: "+i0+" "+j0);
49         int i,j;
50         ic=sc=0; // coada vida
51         qi[sc]=i0; qj[sc]=j0; sc=(sc+1)%qdim; // (i0,j0) --> coada circulara !
52         c[i0][j0]=1; // cost 1 pentru pozitie start
53         while(ic!=sc) // coada nevida
54         {
55             i=qi[ic]; j=qj[ic]; ic=(ic+1)%qdim;
56             fill(i,j);
57         } // while
58
59         afism(c); afism(d);
60     } //matriceCosturi()
61
62     static void fill(int i, int j)
63     {
64         int t=c[i][j]; // timp = lungime traseu !
65
66         if((i-1>=0)&&(c[i-1][j]>t+1)&&(a[i-1][j]<=a[i][j]+100)&&(a[i-1][j]!=0)) // N
67         {
68             c[i-1][j]=t+1;
69             qi[sc]=i-1;
70             qj[sc]=j;
71             sc=(sc+1)%qdim;
72             d[i-1][j]=jos;
73         }
74
75         if((j+1<=n-1)&&(c[i][j+1]>t+1)&&(a[i][j+1]<=a[i][j]+100)&&(a[i][j+1]!=0)) // E
76         {
77             c[i][j+1]=t+1;
78             qi[sc]=i;
79             qj[sc]=j+1;
80             sc=(sc+1)%qdim;
81             d[i][j+1]=stanga;
82         }
83
84         if((i+1<=m-1)&&(c[i+1][j]>t+1)&&(a[i+1][j]<=a[i][j]+100)&&(a[i+1][j]!=0)) // S
85         {
86             c[i+1][j]=t+1;
87             qi[sc]=i+1;
88             qj[sc]=j;
89             sc=(sc+1)%qdim;
90             d[i+1][j]=sus;
91         }
92
93         if((j-1>=0)&&(c[i][j-1]>t+1)&&(a[i][j-1]<=a[i][j]+100)&&(a[i][j-1]!=0)) // V
94         {
95             c[i][j-1]=t+1;
96             qi[sc]=i;
97             qj[sc]=j-1;
98             sc=(sc+1)%qdim;
99             d[i][j-1]=dreapta;
100        }
101    } // fill(...)
102
103    static void afisSolutia() throws IOException
104    {
105        int i,j,max;
106        PrintWriter out = new PrintWriter(
107            new BufferedWriter(new FileWriter("alpinist.out")));
108
109        max=0;
110        lgmin=oo;
111        for(i=0;i<m;i++)
112            for(j=0;j<n;j++)
113                if(c[i][j]<oo) // varf atins
114                    if(a[i][j]>max)

```

```

115         {
116             max=a[i][j];
117             lgmin=c[i][j];
118             ifinal=i;
119             jfinal=j;
120         }
121     else
122         if((a[i][j]==max)&&(c[i][j]<lgmin))
123     {
124         lgmin=c[i][j];
125         ifinal=i;
126         jfinal=j;
127     }
128
129     i=ifinal;
130     j=jfinal;
131     while(a[i][j]!=0)
132     {
133         if(d[i][j]==sus) { c[i-1][j]=jos; i--; }
134         else if(d[i][j]==jos) { c[i+1][j]=sus; i++; }
135         else if(d[i][j]==dreapta) { c[i][j+1]=stanga; j++; }
136         else if(d[i][j]==stanga) { c[i][j-1]=dreapta; j--; }
137         else System.out.println("Eroare la traseu ... !");
138
139     istart=i;
140     jstart=j;
141
142     out.println(a[ifinal][jfinal]);
143     // corectie: 0..i..m-1; 0..j..n-1
144     out.println((istart+1)+" "+(jstart+1)+" "+(ifinal+1)+" "+(jfinal+1));
145     out.println(lgmin-1));
146
147     i=istart;
148     j=jstart;
149     while((i!=ifinal)|| (j!=jfinal))
150     {
151         out.print(i+" "+j+" ");
152         if(c[i][j]==sus) { i--; out.println("N"); }
153         else if(c[i][j]==jos) { i++; out.println("S"); }
154         else if(c[i][j]==dreapta) { j++; out.println("E"); }
155         else if(c[i][j]==stanga) { j--; out.println("W"); }
156         else System.out.println("Eroare la traseu ... !");
157     }
158     out.close();
159 }//afisSolutia()
160
161 static void afism(int[][] a)
162 {
163     int i,j;
164     for(i=0;i<m;i++)
165     {
166         for(j=0;j<n;j++)
167             if(a[i][j]<oo) System.out.print(a[i][j]+" ");
168             System.out.println();
169     }
170 }// afism...
171
172 static void afism(byte[][] d)
173 {
174     int i,j;
175     for(i=0;i<m;i++)
176     {
177         for(j=0;j<n;j++)
178             if(a[i][j]==0) System.out.print(" ");
179             if(d[i][j]==sus) System.out.print("N ");
180             if(d[i][j]==dreapta) System.out.print("E ");
181             if(d[i][j]==jos) System.out.print("S ");
182             System.out.println();
183     }
184     System.out.println();
185 }// afism...
186 }// class

```

Varianta 2:

Listing 37.1.2: alpinist2.java

```

1 import java.io.*; // coada circulara + traseu fara recursivitate
2 class Alpinist
3 {
4     static final int oo=Integer.MAX_VALUE;
5     static final int qdim=200;
6     static final int sus=1, dreapta=2, jos=3, stanga=4;
7     static int[][] a=new int[100][100]; // muntele (harta)
8     static int[][] c=new int[100][100]; // costul ajungerii in (i,j)
9     static byte[][] d=new byte[100][100]; // directii pentru intoarcere
10    static int m,n,istart,jstart,jfinal,lgmin;
11    static int[] qi=new int[qdim]; // coada pentru i din pozitia (i,j)
12    static int[] qj=new int[qdim]; // coada pentru j din pozitia (i,j)
13    static int ic, sc;
14
15    public static void main(String[] args) throws IOException
16    {
17        int i,j;
18        citescDate();
19        for(i=0;i<m;i++)
20            for(j=0;j<n;j++)
21                if(a[i][j]==0) matriceCosturi(i,j);
22        afisSolutia();
23    } // main()
24
25    static void citescDate() throws IOException
26    {
27        int i,j;
28        StreamTokenizer st=new StreamTokenizer(
29            new BufferedReader(new FileReader("alpinist.in")));
30        st.nextToken(); m=(int)st.nval;
31        st.nextToken(); n=(int)st.nval;
32
33        for(i=0;i<m;i++)
34            for(j=0;j<n;j++)
35            {
36                c[i][j]=oo;
37                st.nextToken(); a[i][j]=(int)st.nval;
38            }
39    } // citescDate()
40
41    static void matriceCosturi(int i0, int j0)
42    {
43        int i,j;
44        ic=sc=0; // coada vida
45        qi[sc]=i0; qj[sc]=j0; sc=(sc+1)%qdim; // (i0,j0) --> coada circulara !
46        c[i0][j0]=1; // cost 1 pentru pozitie start
47        while(ic!=sc) // coada nevida
48        {
49            i=qi[ic]; j=qj[ic]; ic=(ic+1)%qdim;
50            fill(i,j);
51        } // while
52    } // matriceCosturi()
53
54    static void fill(int i, int j)
55    {
56        int t=c[i][j]; // timp = lungime traseu !
57
58        if((i-1>=0)&&(c[i-1][j]>t+1)&&(a[i-1][j]<=a[i][j]+100)&&(a[i-1][j]!=0)) // N
59        {
60            c[i-1][j]=t+1;
61            qi[sc]=i-1;
62            qj[sc]=j;
63            sc=(sc+1)%qdim;
64            d[i-1][j]=jos;
65        }
66
67        if((j+1<=n-1)&&(c[i][j+1]>t+1)&&(a[i][j+1]<=a[i][j]+100)&&(a[i][j+1]!=0)) // E
68        {
69            c[i][j+1]=t+1;
70            qi[sc]=i;
71            qj[sc]=j+1;
72            sc=(sc+1)%qdim;
73            d[i][j+1]=stanga;
74        }
75    }

```

```

76      if ((i+1<=m-1) && (c[i+1][j]>t+1) && (a[i+1][j]<=a[i][j]+100) && (a[i+1][j]!=0)) // s
77      {
78          c[i+1][j]=t+1;
79          qi[sc]=i+1;
80          qj[sc]=j;
81          sc=(sc+1)%qdim;
82          d[i+1][j]=sus;
83      }
84
85      if ((j-1>=0) && (c[i][j-1]>t+1) && (a[i][j-1]<=a[i][j]+100) && (a[i][j-1]!=0)) // v
86      {
87          c[i][j-1]=t+1;
88          qi[sc]=i;
89          qj[sc]=j-1;
90          sc=(sc+1)%qdim;
91          d[i][j-1]=dreapta;
92      }
93  } // fill(...)

94
95  static void afisSolutia() throws IOException
96  {
97      int i,j,max;
98      PrintWriter out = new PrintWriter(
99          new BufferedWriter(new FileWriter("alpinist.out")));
100
101     max=0;
102     lgmin=oo;
103     for(i=0;i<m;i++)
104         for(j=0;j<n;j++)
105             if(c[i][j]<oo) // varf atins
106                 if(a[i][j]>max)
107                 {
108                     max=a[i][j];
109                     lgmin=c[i][j];
110                     ifinal=i;
111                     jfinal=j;
112                 }
113                 else
114                     if((a[i][j]==max) && (c[i][j]<lgmin))
115                     {
116                         lgmin=c[i][j];
117                         ifinal=i;
118                         jfinal=j;
119                     }
120
121     i=ifinal;
122     j=jfinal;
123     while(a[i][j]!=0)
124         if(d[i][j]==sus) { c[i-1][j]=jos; i--; }
125         else if(d[i][j]==jos) { c[i+1][j]=sus; i++; }
126         else if(d[i][j]==dreapta) { c[i][j+1]=stanga; j++; }
127         else if(d[i][j]==stanga) { c[i][j-1]=dreapta; j--; }
128         else System.out.println("Eroare la traseu ... !");
129     istart=i;
130     jstart=j;
131
132     out.println(a[ifinal][jfinal]);
133     // corectie de afisare: 0..i..m-1; 0..j..n-1; 1..lgmin..
134     out.println((istart+1)+" "+(jstart+1)+" "+(ifinal+1)+" "+(jfinal+1));
135     out.println(lgmin-1);
136
137     i=istart;
138     j=jstart;
139     while((i!=ifinal)|| (j!=jfinal))
140     {
141         if(c[i][j]==sus) { i--; out.print("N"); }
142         else if(c[i][j]==jos) { i++; out.print("S"); }
143         else if(c[i][j]==dreapta) { j++; out.print("E"); }
144         else if(c[i][j]==stanga) { j--; out.print("W"); }
145         else System.out.println("Eroare la traseu ... !");
146     }
147     out.println();
148     out.close();
149 } // afisSolutia()
150 } // class

```

37.1.3 *Cod sursă

37.2 Asfaltare

prof. Roxana Tîmplaru, Craiova

Un primar proaspăt ales vrea să dovedească electoratului său că votându-l, cetățenii au făcut o alegeră bună. În acest scop el a decis săreasfalteze străzile dintre N edificii importante din oraș, numerotate de la 1 la N . Între oricare două dintre aceste edificii există o singură stradă cu două sensuri de circulație. Edificiul numerotat cu 1 este primăria.

Primarul cere consilierilor să stabilească toate traseele pe care reasfaltarea străzilor o poate urma printre cele N edificii, știind că are H străzi "preferate" pe care trebuie să le asfalteze în mod obligatoriu. Se stie că oricare două străzi preferate nu au capete comune. Traseele care se vor reasfalta trebuie să pornească de la primărie, să ajungă o singură dată la fiecare din celelalte $N - 1$ edificii și să se întoarcă tot la primărie.

Cerință

Determinați numărul traseelor distințe, respectând cerințele de mai sus.

Date de intrare

Fișier de intrare: ASFALT.IN

Linia 1: $N \ H$ două numere naturale, separate printr-un spațiu, reprezentând numărul edificiilor (N), respectiv numărul străzilor preferate ale primarului (H).

Date de ieșire

Fișier de ieșire: ASFALT.OUT

Linia 1: x număr întreg pozitiv, reprezentând numărul traseelor distințe, posibile;

Restricții

$3 \leq N \leq 1000$

$0 \leq H \leq N/2$

Dacă un traseu este diferit de un altul doar prin direcția în care se parcurge drumul, pornind de la primărie și revenind aici, acesta se consideră identic cu primul. De exemplu, traseul 1-2-3-4-1 este identic cu traseul 1-4-3-2-1.

Exemplu

ASFALT.IN ASFALT.OUT

4 1 2

Timp maxim de executare/test: 5 secunde

37.2.1 *Indicații de rezolvare

37.2.2 Rezolvare detaliată

Listing 37.2.1: asfaltare.java

```

1 import java.io.*; // rezultat=(n-h-1)!*2^{h-1} cu numere mari
2 class Asfaltare
3 {
4     public static void main (String[] args) throws IOException
5     {
6         int n,h,i;
7         StreamTokenizer st=new StreamTokenizer(
8             new BufferedReader(new FileReader("asfalt10.in")));
9         PrintWriter out=new PrintWriter(new BufferedWriter(
10            new FileWriter("asfalt.out")));
11         st.nextToken(); n=(int)st.nval;
12         st.nextToken(); h=(int)st.nval;
13
14         int[] sol=prod(fact(n-h-1),putere(2,h-1));
15         for(i=sol.length-1;i>=0;i--) out.print(sol[i]);
16         out.println();
17         out.close();
18     } // main(...)
```

```

19
20     static int[] putere(int a, int n)
21     {
22         int k;
23         int[] p=nr2v(1);
24         for(k=1;k<=n;k++) p=prod(p,nr2v(a));
25         return p;
26     } // putere(...)

27
28     static int[] fact(int n)
29     {
30         int k;
31         int[] p=nr2v(1);
32         for(k=1;k<=n;k++) p=prod(p,nr2v(k));
33         return p;
34     } // fact(...)

35
36     static int[] nr2v(int nr)
37     {
38         int[] x;
39         if(nr==0) { x=new int[1]; x[0]=0; return x; }
40         int nc, nrrez=nr;
41         nc=0;
42         while(nr!=0) { nc++; nr=nr/10; }

43         x=new int[nc];
44         nr=nrrez;
45         nc=0;
46         while(nr!=0) { x[nc]=nr%10; nc++; nr=nr/10; }
47         return x;
48     } // nr2v(...)

49
50     static int[] prod(int[] x,int[] y)
51     {
52         int i,j,t,s;
53         int nx=x.length;
54         int ny=y.length;
55         int nz=nx+ny;
56         int[][] a=new int[ny][nx+ny];
57         int[] z=new int[nx+ny];
58         for(j=0;j<ny;j++)
59         {
60             t=0;
61             for(i=0;i<nx;i++)
62             {
63                 s=t+y[j]*x[i];
64                 z[j+i]+=s;
65                 t=z[j+i]/10;
66                 z[j+i]%=10;
67             }
68         }
69         z[j+i]+=t;
70     }
71     if(z[nz-1]!=0) return z;
72     else
73     {
74         int[] zz=new int[nz-1];
75         for(j=0;j<zz.length;j++) zz[j]=z[j];
76         return zz;
77     }
78 } // prod(...)

79 } // class

```

37.2.3 *Cod sursă

37.3 Oracolul decide

prof. Doru Popescu Anastasiu, Slatina

La un concurs participă N concurenți. Fiecare concurent primește o foaie de hârtie pe care va scrie un cuvânt având cel mult 100 de caractere (litere mici ale alfabetului englez). Cuvintele vor

fi distințe.

Pentru departajare, concurenții apelează la un oracol. Acesta produce și el un cuvânt. Va câștiga concurrentul care a scris cuvântul "cel mai apropiat" de al oracolului.

Gradul de "apropiere" dintre două cuvinte este lungimea subcuvântului comun de lungime maximă. Prin subcuvânt al unui cuvânt dat se înțelege un cuvânt care se poate obține din cuvântul dat, eliminând 0 sau mai multe litere și păstrând ordinea literelor rămase.

Cerință

Se cunosc cuvântul c_0 produs de oracol și cuvintele c_i , $i = 1, \dots, N$ scrise de concurenți. Pentru a ajuta comisia să desemneze câștigătorul, se cere ca pentru fiecare i să identifică pozițiile literelor ce trebuie șterse din c_0 și din c_i astfel încât prin ștergere să se obțină unul dintre subcuvintele comune de lungime maximă.

Date de intrare

Fișier de intrare: ORACOL.IN

Linia 1: N număr natural nenul, reprezentând numărul concurenților;

Linia 2: c_0 cuvântul produs de oracol;

Linile 3.. $N+2$: *cuvânt* pe aceste N linii se află cuvintele scrise de cei N concurenți, un cuvânt pe o linie;

Date de ieșire

Fișier de ieșire: ORACOL.OUT

Linile $1..2*N$: *pozițiile literelor ce trebuie șterse* pe fiecare linie i ($i = 1, 3, \dots, 2 * N - 1$) se vor scrie numere naturale neneule, separate prin căte un spațiu, reprezentând pozițiile de pe care se vor șterge litere din cuvântul produs de oracol; pe fiecare linie j ($j = 2, 4, \dots, 2 * N$) se vor scrie numere naturale neneule, separate prin căte un spațiu, reprezentând pozițiile de pe care se vor șterge litere din cuvântul concurrentului cu numărul $j/2$.

Restricții

$2 \leq N \leq 100$

Dacă există mai multe soluții, în fișier se va scrie una singură.

Dacă dintr-un cuvânt nu se va tăia nici o literă, linia respectivă din fișierul de intrare va rămâne vidă.

Exemplu

ORACOL.IN ORACOL.OUT poate conține soluția:

3	3
abc	3 4
abxd	
aabxyc	1 4 5
acb	3
	2

Timp maxim de executare/test: 1 secundă

37.3.1 *Indicații de rezolvare

37.3.2 Rezolvare detaliată

Varianta 1:

Listing 37.3.1: oracol1.java

```

1 import java.io.*; // subsir comun maximal - problema clasica ...
2 class Oracol      // varianta cu mesaje pentru depanare ...
3 {
4     static final char sus='|', stanga='-', diag='*';
5     static int[][] a;
6     static char[][] d;
7
8     static String x,y;
9     static boolean[] xx=new boolean[101];
10    static boolean[] yy=new boolean[101];
11    static char[] z;
12    static int m,n,nc;
13
14    public static void main(String[] args) throws IOException

```

```

15  {
16      int i,j,k;
17      BufferedReader br=new BufferedReader(new FileReader("oracol.in"));
18      PrintWriter out=new PrintWriter(
19          new BufferedWriter(new FileWriter("oracol.out")));
20
21      nc=Integer.parseInt(br.readLine());
22
23      x=br.readLine().replace(" ",""); // elimina spatiile ... de la sfarsit !
24      m=x.length();
25
26      for(k=1;k<=nc;k++)
27      {
28          y=br.readLine().replaceAll(" ",""); // elimina spatiile ... daca sunt!
29          n=y.length();
30
31          matrad();
32          afism(a);
33          afism(d);
34
35          System.out.print("O solutie oarecare: ");
36          z=new char[a[m][n]+1];
37          for(i=1;i<=m;i++) xx[i]=false;
38          for(j=1;j<=n;j++) yy[j]=false;
39          osol(m,n);
40          System.out.println("\n");
41
42          for(i=1;i<=m;i++) if(!xx[i]) out.print(i+" ");
43          out.println();
44          for(j=1;j<=n;j++) if(!yy[j]) out.print(j+" ");
45          out.println();
46      }
47      out.close();
48      System.out.println("\n");
49 } // main(...)

50
51 static void matrad()
52 {
53     int i,j;
54     a=new int[m+1][n+1];
55     d=new char[m+1][n+1];
56     for(i=1;i<=m;i++)
57         for(j=1;j<=n;j++)
58             if(x.charAt(i-1)==y.charAt(j-1))
59             {
60                 a[i][j]=1+a[i-1][j-1];
61                 d[i][j]=diag;
62             }
63             else
64             {
65                 a[i][j]=max(a[i-1][j],a[i][j-1]);
66                 if(a[i-1][j]>a[i][j-1]) d[i][j]=sus;
67                 else d[i][j]=stanga;
68             }
69 } // matrad()

70
71 static void osol(int lin, int col)
72 {
73     if((lin==0) || (col==0)) return;
74
75     if(d[lin][col]==diag) osol(lin-1,col-1);
76     else if(d[lin][col]==sus) osol(lin-1,col);
77     else osol(lin,col-1);
78
79     if(d[lin][col]==diag)
80     {
81         System.out.print(x.charAt(lin-1));
82         xx[lin]=yy[col]=true;
83     }
84 } // osol(...)

85
86 static int max(int a, int b)
87 {
88     if(a>b) return a; else return b;
89 } // max(...)
90

```

```

91     static void afism(int[][] a)
92     {
93         int i,j;
94
95         System.out.print("      ");
96         for(j=0;j<n;j++) System.out.print(y.charAt(j)+" ");
97         System.out.println();
98
99         System.out.print("  ");
100        for(j=0;j<=n;j++) System.out.print(a[0][j]+" ");
101        System.out.println();
102
103        for(i=1;i<=m;i++)
104        {
105            System.out.print(x.charAt(i-1)+" ");
106            for(j=0;j<=n;j++) System.out.print(a[i][j]+" ");
107            System.out.println();
108        }
109        System.out.println("\n");
110    }// afism(int[][]...)
111
112    static void afism(char[][] d) // difera tipul parametrului
113    {
114        int i,j;
115        System.out.print("      ");
116        for(j=0;j<n;j++) System.out.print(y.charAt(j)+" ");
117        System.out.println();
118
119        System.out.print("  ");
120        for(j=0;j<=n;j++) System.out.print(d[0][j]+" ");
121        System.out.println();
122
123        for(i=1;i<=m;i++)
124        {
125            System.out.print(x.charAt(i-1)+" ");
126            for(j=0;j<=n;j++) System.out.print(d[i][j]+" ");
127            System.out.println();
128        }
129        System.out.println("\n");
130    }// afism(char[][]...)
131}// class

```

Varianta 2:

Listing 37.3.2: oracol2.java

```

1 import java.io.*; // problema reala ...
2 class Oracol
3 {
4     static final char sus='|', stanga='-', diag='*';
5     static int[][] a;
6     static char[][] d;
7     static String x,y;
8     static boolean[] xx=new boolean[101];
9     static boolean[] yy=new boolean[101];
10    static int m,n,nc;
11
12    public static void main(String[] args) throws IOException
13    {
14        int i,j,k;
15        BufferedReader br=new BufferedReader(new FileReader("oracol.in"));
16        PrintWriter out=new PrintWriter(
17            new BufferedWriter( new FileWriter("oracol.out")));
18
19        nc=Integer.parseInt(br.readLine());
20
21        x=br.readLine().replace(" ",""); // elimina spatiile ... de la sfarsit !
22        m=x.length();
23
24        for(k=1;k<=nc;k++)
25        {
26            y=br.readLine().replaceAll(" ",""); // elimina spatiile ... daca sunt !
27            n=y.length();
28
29            matrad();
30            for(i=1;i<=m;i++) xx[i]=false;

```

```

31         for(j=1; j<=n; j++) yy[j]=false;
32         osol(m,n);
33
34         for(i=1; i<=m; i++) if(!xx[i]) out.print(i+" ");
35         out.println();
36         for(j=1; j<=n; j++) if(!yy[j]) out.print(j+" ");
37         out.println();
38     }
39     out.close();
40 } // main(...)

41
42 static void matrad()
43 {
44     int i,j;
45     a=new int[m+1][n+1];
46     d=new char[m+1][n+1];
47     for(i=1; i<=m; i++)
48     for(j=1; j<=n; j++)
49     if(x.charAt(i-1)==y.charAt(j-1))
50     {
51         a[i][j]=1+a[i-1][j-1];
52         d[i][j]=diag;
53     }
54     else
55     {
56         a[i][j]=max(a[i-1][j],a[i][j-1]);
57         if(a[i-1][j]>a[i][j-1]) d[i][j]=sus; else d[i][j]=stanga;
58     }
59 } // matrad()
60
61 static void osol(int lin, int col)
62 {
63     if((lin==0) || (col==0)) return;
64     if(d[lin][col]==diag) osol(lin-1,col-1); else
65     if(d[lin][col]==sus) osol(lin-1,col); else osol(lin,col-1);
66     if(d[lin][col]==diag) xx[lin]=yy[col]=true;
67 } // osol(...)

68
69 static int max(int a, int b)
70 {
71     if(a>b) return a; else return b;
72 } // max(...)

73 } // class

```

37.3.3 *Cod sursă

37.4 Alipiri

prof. Gelu Manolache, Piatra Neamț

Spunem că numărul natural nenul n_1 se poate alipi pe K biți cu numărul natural nenul n_2 dacă reprezentările lor binare au fiecare cel puțin K biți (primul bit fiind 1), iar ultimii K biți ai numărului n_1 coincid cu primii K biți ai numărului n_2 . După alipire rezultă un alt număr, format din concatenarea lui n_1 cu n_2 , dar în care secvența comună de K biți apare o singură dată și în ea fiecare bit este înlocuit cu complementarul său (0 devine 1 și invers).

Exemplu: Numărul $78=10011102$ se poate alipi pe 3 biți cu $25=110012$ și rezultă numărul $1001001012=293$.

Numerelor nr_1, nr_2, \dots, nr_M formează o listă de alipiri pe K biți, de lungimea $M \geq 1$, cu finalul P , dacă nr_1 se alipește pe K biți cu nr_2 , rezultatul se alipește pe K biți cu nr_3, \dots , rezultatul se alipește pe K biți cu nr_M , iar rezultatul final este numărul P .

Notă: Dacă $M = 1$, atunci trebuie să avem $nr_1 = P$.

Cerință

Pentru un sir dat de numere naturale nenule și valorile naturale nenule K și P , se cere:

- a) să se determine lungimea maximă a unei liste de alipiri pe K biți, formată cu elemente din sirul dat, nu neapărat în ordinea din acest sir;

b) să se verifice dacă se poate obține, cu numere din sirul dat, o listă de alipiri pe K biți cu finalul P ; în caz afirmativ să se precizeze o listă de lungime minimă de alipiri pe K biți cu finalul P .

Date de intrare

Fișier de intrare: ALIPIRI.IN

Linia 1: $N \ K \ P$ trei numere naturale nenule, separate prin câte un spațiu, reprezentând numărul N al numerelor date, lungimea K a secvențelor comune din alipiri, respectiv valoarea P a numărului care trebuie obținut prin alipiri;

Linia 2: $nr_1 \ nr_2 \dots \ nr_N$ N numere naturale nenule, separate prin câte un spațiu, reprezentând sirul dat.

Date de ieșire

Fișier de ieșire: ALIPIRI.OUT

Linia 1: M număr natural nenul, reprezentând rezultatul de la cerința a);

Linia 2: *mesaj* dacă cerința b) poate fi satisfăcută, pe această linie se va scrie 'DA', în caz contrar se va scrie 'NU';

Linia 3: $nr_1 \ nr_2 \dots$ dacă pe linia 2 ați scris 'DA', pe această linie se va scrie o listă de alipiri (pe K biți) de lungime minimă cu finalul P ; numerele din listă se vor separa prin câte un spațiu.

Restricții

$1 \leq K \leq 8, 1 \leq N \leq 9, 1 \leq P \leq 2.000.000.000$.

dacă cerința b) este satisfăcută de mai multe liste (de lungime minimă), în fișier se va scrie una singură.

pentru ambele cerințe fiecare număr din sirul de intrare poate apărea cel mult o dată în listele de alipiri considerate.

Exemplu

ALIPIRI.IN ALIPIRI.OUT

4 3 291	4
14 59 36 31	DA
	59 31 14

Notă: Se acordă punctaje parțiale: a) 40%; b) 60%

Timp maxim de execuție/test: 10 secunde

37.4.1 *Indicații de rezolvare

37.4.2 Rezolvare detaliată

Varianta 1:

Listing 37.4.1: alipiril.java

```

1 import java.io.*;      // varianta cu mesaje ... pentru depanare!
2 class Alipiri          // ??? t8(6?4)  t10(9?5) ???
3 {
4     static int n,k;
5     static int nmax=0,nmin=10;
6
7     static int[] nr=new int[10];      // numerele ... p=nr[0]
8     static int[] ncb=new int[10];    // ncb[i]=nr cifre binare din nr[i]
9     static int[][] a=new int[10][33]; // a[i]=cifrele binare din nr[i]
10
11    static boolean[] eFolosit=new boolean[10];
12
13    static int[] x=new int[10];      // solutia curenta backtracking
14    static int[] xmin=new int[10];   // solutia finala backtracking
15
16    static int[] y=new int[321];    // bitii din lista
17    static int py=0;                // prima pozitie libera in y
18    static int idmesaj=0;
19
20    public static void main (String[] args) throws IOException
21    {
22        int i;
23        long t1,t2;

```

```

24     t1=System.currentTimeMillis();
25     StreamTokenizer st=new StreamTokenizer(
26         new BufferedReader(new FileReader("alipiri.in")));
27     PrintWriter out=new PrintWriter(new BufferedWriter(
28         new FileWriter("alipiri.out")));
29
30     st.nextToken(); n=(int)st.nval;
31     st.nextToken(); k=(int)st.nval;
32     System.out.println("k="+k+"\n");
33
34     for(i=0;i<=n;i++) { st.nextToken(); nr[i]=(int)st.nval; }
35
36     for(i=0;i<=n;i++) cifreBinare(i);
37     System.out.println();
38
39     f(1);                                // backtracking
40
41     out.println(nmax);
42
43     if(idmesaj==0) out.println("NU");
44     else
45     {
46         out.println("DA");
47         for(i=1;i<=nmin;i++) out.print(nr[xmin[i]]+" ");
48         out.println();
49     }
50     out.close();
51     t2=System.currentTimeMillis();
52     System.out.println("\nTime: "+(t2-t1));
53 } // main(...)

54
55     static void cifreBinare(int i)
56     {
57         int nri=nr[i];
58         while(nri!=0) { a[i][ncb[i]]=nri%2; ncb[i]++; nri/=2; }
59         System.out.print(i+": "+nr[i]+" = ");
60         afisnr(i);
61 } // cifreBinare(...)

62
63     static void f(int i)
64     {
65         byte j,i1;
66         for(j=1;j<=n;j++)
67         {
68             if(eFolosit[j]) continue;
69             if(i>1) if(!seAlipeste(j)) continue;
70
71             x[i]=j;
72             eFolosit[j]=true;
73             alipeste(j);
74
75             if(i>nmax) nmax=i;
76             System.out.print(" --> "); afisx(i); afisy(i);
77
78             if(i<n) f(i+1);
79
80             eFolosit[j]=false;
81             // scot nr[j] din lista si refac cele k pozitii din y=list !!
82             if(i==1) py=0; // este numai primul numar
83             else
84             {
85                 py=py-ncb[j];
86                 for(i1=1;i1<=k;i1++)
87                 {
88                     y[py]=1-y[py];
89                     py++;
90                 }
91             }
92             System.out.print("<-- "); afisx(i); afisy(i);
93         } // for j
94     } // f(...)

95     static void afisnr(int i)
96     {
97         int j;
98         for(j=ncb[i]-1;j>=0;j--) System.out.print(a[i][j]+"");


```

```

100     System.out.println();
101 } // afisnr(...)
102
103 static void afisy(int i)
104 {
105     int j,s=0;
106
107     for(j=0;j<py;j++) System.out.print(y[j]+");
108
109     if(py<32) for(j=0;j<py;j++) s=s*2+y[j];
110     System.out.print(" = "+s+" ");
111
112     if(s==nr[0])
113     {
114         idmesaj=1;
115         if(i<nmin)
116         {
117             nmin=i;
118             for(j=1;j<=i;j++) xmin[j]=x[j];
119             System.out.print("...*****... ");
120         }
121     }
122     System.out.println();
123 } // afisy(...)
124
125 static void afisx(int i)
126 {
127     int j;
128     for(j=1;j<=i;j++) System.out.print(x[j]+");
129     System.out.print(" : ");
130 } // afisx(...)
131
132 static boolean seAlipeste(int j)
133 {
134     int i;
135     if(py==0) return true;
136     if(py<k) return false; // primul pus este prea mic !
137     for(i=0;i<k;i++) if(y[py-k+i]!=a[j][ncb[j]-1-i]) return false;
138     return true;
139 } // seAlipeste(...)
140
141 static void alipeste(int i)
142 {
143     int j;
144     if(py>0)
145     {
146         for(j=0;j<k;j++) y[py-k+j]=1-y[py-k+j];
147         for(j=ncb[i]-k-1;j>=0;j--) y[py++]=a[i][j];
148     }
149     else // plasez primul numar
150     {
151         for(j=ncb[i]-1;j>=0;j--) y[py++]=a[i][j];
152     }
153 } // alipeste(...)
154 } // class

```

Varianta 2:

Listing 37.4.2: alipiri2.java

```

1 import java.io.*; // Solutie finala; ??? t8(6?4) t10(9?5) ???
2 class Alipiri
3 {
4     static int n,k,nmax=0,nmin=10;
5     static int[] nr=new int[10]; // numerele ... p=nr[0] ...
6     static int[] ncb=new int[10]; // ncb[i]=nr cifre binare din nr[i]
7     static int[][] a=new int[10][33]; // a[i]=cifrele binare din nr[i]
8     static boolean[] eFolosit=new boolean[10];
9     static int[] x=new int[10]; // solutia curenta backtracking
10    static int[] xmin=new int[10]; // solutia finala backtracking
11    static int[] y=new int[321]; // bitii din lista
12    static int py=0; // prima pozitie libera in y
13    static int idmesaj=0;
14
15    public static void main (String[] args) throws IOException
16    {

```

```

17     int i;
18     StreamTokenizer st=new StreamTokenizer(
19         new BufferedReader(new FileReader("alipiri.in")));
20     PrintWriter out=new PrintWriter(new BufferedWriter(
21         new FileWriter("alipiri.out")));
22     st.nextToken(); n=(int)st.nval;
23     st.nextToken(); k=(int)st.nval;
24     for(i=0;i<=n;i++) { st.nextToken(); nr[i]=(int)st.nval; }
25     for(i=0;i<=n;i++) cifreBinare(i);
26     f(1);
27     out.println(nmax);
28     if(idmesaj==0) out.println("NU");
29     else
30     {
31         out.println("DA");
32         for(i=1;i<=mmin;i++) out.print(nr[xmin[i]]+" ");
33         out.println();
34     }
35     out.close();
36 } // main(...)

37
38 static void cifreBinare(int i)
39 {
40     int nri=nr[i];
41     while(nri!=0) { a[i][ncb[i]]=nri%2; ncb[i]++; nri/=2; }
42 } // cifreBinare(...)

43
44 static void f(int i)
45 {
46     byte j,i1;
47     for(j=1;j<=n;j++)
48     {
49         if(eFolosit[j]) continue;
50         if(i>1) if(!seAlipeste(j)) continue;
51         x[i]=j;
52         eFolosit[j]=true;
53         alipeste(j);
54         if(i>nmax) nmax=i;
55
56         if(i<n) f(i+1);
57
58         eFolosit[j]=false;
59         // scot nr[j] din lista si refac cele k pozitii din y=list
60         if(i==1) py=0; // este numai primul numar
61         else
62         {
63             py=py-ncb[j];
64             for(i1=1;i1<=k;i1++) { y[py]=1-y[py]; py++; }
65         }
66     } // for j
67 } // f(...)

68
69 static boolean seAlipeste(int j)
70 {
71     int i;
72     if(py==0) return true;
73     if(py<k) return false; // primul nr pus ... este prea mic !
74     for(i=0;i<k;i++) if(y[py-k+i]!=a[j][ncb[j]-1-i]) return false;
75     return true;
76 } // seAlipeste(...)

77
78 static void alipeste(int i)
79 {
80     int j;
81     if(py>0)
82     {
83         for(j=0;j<k;j++) y[py-k+j]=1-y[py-k+j];
84         for(j=ncb[i]-k-1;j>=0;j--) y[py++]=a[i][j];
85     }
86     else
87         for(j=ncb[i]-1;j>=0;j--) y[py++]=a[i][j]; // plasez primul numar
88 } // alipeste(...)

89 } // class

```

37.4.3 *Cod sursă

37.5 Drum

prof. Doru Popescu Anastasiu, Slatina

În regatul XLand orașele erau înconjurate de ziduri în formă de poligoane convexe. Împăratul a dispus construirea unui drum de legătură directă între capitală și un alt oraș dat. Fiecare extremitate a drumului poate fi orice punct situat pe zidul orașului, respectiv capitalei. Lungimea drumului este distanța dintre extremitățile sale.

Cerință

Determinați cel mai scurt drum dintre capitală și orașul dat.

Date de intrare

Fișier de intrare: DRUM.IN

Linia 1: K_1 număr natural nenul, reprezentând numărul de colțuri ale zidurilor capitalei;

Linia 2: $x_1 \ y_1 \ x_2 \ y_2 \dots x_{K_1} \ y_{K_1}$ K_1 perechi de numere întregi, separate prin câte un spațiu, reprezentând coordonatele vârfurilor zidurilor capitalei;

Linia 3: K_2 număr natural nenul, reprezentând numărul de colțuri ale zidurilor orașului dat;

Linia 4: $x_1 \ y_1 \ x_2 \ y_2 \dots x_{K_2} \ y_{K_2}$ K_2 perechi de numere întregi, separate prin câte un spațiu, reprezentând coordonatele vârfurilor zidurilor acestui oraș.

Date de ieșire

Fișier de ieșire: DRUM.OUT

Linia 1: $x_1 \ y_1 \ x_2 \ y_2$ patru numere reale trunchiate la 4 zecimale, separate prin câte un spațiu, reprezentând extremitățile drumului de legătură respectiv.

Restricții

$2 \leq K_1, K_2 \leq 20$

Coordonatele vârfurilor zidurilor ce încadrează orașul, respectiv capitala sunt numere întregi aparținând intervalului $[-100, 100]$ și sunt date fie în ordinea deplasării acelor de ceasornic, fie în sens invers deplasării acelor de ceasornic.

Capitala și orașul nu au nici un punct comun (nu au puncte interioare comune și nu au puncte comune pe zidurile lor).

Exemplu

DRUM.IN	DRUM.OUT
4	5.0000 3.5000 8.0000 3.5000
3 4 3 2 5 2 5 4	
4	
8 3 8 6 11 6 11 3	

Timp maxim de executare/test: 1 secundă

37.5.1 *Indicații de rezolvare

37.5.2 Rezolvare detaliată

Varianta 1:

Listing 37.5.1: drum1.java

```

1 import java.io.*;
2 class Drum
3 {
4     static int m,n;
5     static int[] xx1=new int[20];    static int[] yy1=new int[20];
6     static int[] xx2=new int[20];    static int[] yy2=new int[20];
7     static int[] x1=new int[3];      static int[] y1=new int[3];
8     static int[] x2=new int[3];      static int[] y2=new int[3];
9     static double x0, y0;           // proiecție punct pe segment sau ...
10    static double x1s, y1s, x2s, y2s;

```

```

11
12  public static void main(String[] args) throws IOException
13  {
14      int i,j,i0,j0;
15      double d,dmin;
16      StreamTokenizer st=new StreamTokenizer(
17          new BufferedReader(new FileReader("drum.in")));
18      PrintWriter out=new PrintWriter(
19          new BufferedWriter(new FileWriter("drum.out")));
20
21      st.nextToken(); m=(int)st.nval;
22      for(i=0;i<m;i++)
23      {
24          st.nextToken(); xx1[i]=(int)st.nval;
25          st.nextToken(); yy1[i]=(int)st.nval;
26      }
27
28      st.nextToken(); n=(int)st.nval;
29      for(j=0;j<n;j++)
30      {
31          st.nextToken(); xx2[j]=(int)st.nval;
32          st.nextToken(); yy2[j]=(int)st.nval;
33      }
34
35      dmin=300; i0=-1; j0=-1;
36      for(i=0;i<m;i++)
37          for(j=0;j<n;j++)
38          {
39              d=dist(xx1[i],yy1[i],xx2[j],yy2[j]);
40              if(d<dmin)
41              {
42                  dmin=d; i0=i; j0=j;
43                  x1s=xx1[i]; y1s=yy1[i];
44                  x2s=xx2[j]; y2s=yy2[j];
45              }
46          }
47      System.out.println(i0+" "+j0+" "+dmin);
48
49      x1[0]=xx1[(i0-1+m)%m]; y1[0]=yy1[(i0-1+m)%m];
50      x1[1]=xx1[i0]; y1[1]=yy1[i0];
51      x1[2]=xx1[(i0+1)%m]; y1[2]=yy1[(i0+1)%m];
52
53      x2[0]=xx2[(j0-1+n)%n]; y2[0]=yy2[(j0-1+n)%n];
54      x2[1]=xx2[j0]; y2[1]=yy2[j0];
55      x2[2]=xx2[(j0+1)%n]; y2[2]=yy2[(j0+1)%n];
56
57      // dist de la P(x1[i],y1[i]) la segm [A(x2[0],y2[0]);B(x2[1],y2[1])]
58      //                                         [A(x2[1],y2[1]);B(x2[2],y2[2])]
59      for(i=0;i<=2;i++)
60          for(j=0;j<=1;j++)
61          {
62              d=dist(x1[i],y1[i],x2[j],y2[j],x2[j+1],y2[j+1]);
63
64              System.out.println(i+" "+j+" : "+x1[i]+" "+y1[i]+" --> "+
65                                 "x2[j]+" "+y2[j]+" "+x2[j+1]+" "+y2[j+1] +
66                                 "\t==> "+d+" : "+x0+" "+y0);
67
68              if(d<dmin)
69              {
70                  dmin=d;
71                  x1s=x1[i]; y1s=y1[i];
72                  x2s=x0; y2s=y0; // proiectia sau ...
73              }
74          }
75      System.out.println("dmin = "+dmin+" : "+
76                         "x1s+" "+y1s+" "+x2s+" "+y2s+"\n");
77
78      // dist de la P(x2[j],y2[j]) la segm [A(x1[0],y1[0]);B(x1[1],y1[1])]
79      //                                         [A(x1[1],y1[1]);B(x1[2],y1[2])]
80      for(j=0;j<=2;j++)
81          for(i=0;i<=1;i++)
82          {
83              d=dist(x2[j],y2[j],x1[i],y1[i],x1[i+1],y1[i+1]);
84
85              System.out.println(i+" "+j+" : "+x2[j]+" "+y2[j]+" --> "+
86                                 "x1[i]+" "+y1[i]+" "+x1[i+1]+" "+y1[i+1]+

```

```

87                         "\t==> "+d+" : "+x0+" "+y0);
88
89         if(d<dmin)
90     {
91             dmin=d;
92             x2s=x2[j]; y2s=y2[j];
93             x1s=x0;      y1s=y0;    // proiectia sau ...
94         }
95     }
96
97     System.out.println("dmin = "+dmin+" : "+x1s+" "+y1s+" "+x2s+" "+y2s);
98
99     out.println(((int)(x1s*10000))/10000.0+" "+((int)(y1s*10000))/10000.0+" "+
100           ((int)(x2s*10000))/10000.0+" "+((int)(y2s*10000))/10000.0);
101     out.close();
102 } // main...
103
104 static double dist(double xp1, double yp1, double xp2, double yp2)
105 {
106     return Math.sqrt((xp2-xp1)*(xp2-xp1)+(yp2-yp1)*(yp2-yp1));
107 }
108
109 //distanta de la (a0,b0) la segm [(a1,b1);(a2,b2)]
110 static double dist(double a0,double b0,double a1,double b1,double a2,double b2)
111 {
112     double m;
113
114     if(Math.abs(a1-a2)<0.00001) // a1==a2 ... reale ...
115     if((b0-b1)*(b0-b2)<=0)    // b0 intre b1 si b2
116     {x0=a1; y0=b0; return Math.abs(a1-a0);}
117     else                      // proiectia nu cade pe segment
118     if(dist(a0,b0,a1,b1)<dist(a0,b0,a2,b2))
119     {
120         x0=a1; y0=b1; return dist(a0,b0,a1,b1);
121     }
122     else
123     {
124         x0=a2; y0=b2; return dist(a0,b0,a2,b2);
125     }
126
127     if(Math.abs(b1-b2)<0.00001) // b1==b2 ... reale ...
128     if((a0-a1)*(a0-a2)<=0)    // a0 intre a1 si a2
129     {x0=a0; y0=b1; return Math.abs(b1-b0);}
130     else                      // proiectia nu cade pe segment
131     if(dist(a0,b0,a1,b1)<dist(a0,b0,a2,b2))
132     {
133         x0=a1; y0=b1; return dist(a0,b0,a1,b1);
134     }
135     else
136     {
137         x0=a2; y0=b2; return dist(a0,b0,a2,b2);
138     }
139
140     m=(b2-b1)/(a2-a1);
141     x0=(b0-b1+m*a1+a0/m)/(m+1/m);
142     y0=b1+m*(x0-a1);
143     if(((x0-a1)*(x0-a2)<=0)&& // x0 intre a1 si a2
144       ((y0-b1)*(y0-b2)<=0)) // y0 intre b1 si b2
145     return dist(x0,y0,a0,b0);
146     else
147     if(dist(a0,b0,a1,b1)<dist(a0,b0,a2,b2))
148     {
149         x0=a1; y0=b1; return dist(a0,b0,a1,b1);
150     }
151     else
152     {
153         x0=a2; y0=b2; return dist(a0,b0,a2,b2);
154     }
155 } // dist...
156 } // class

```

Varianta 2:

Listing 37.5.2: drum2.java

```

1 import java.io.*; // in fis rezultat sunt rotunjite nu "trunchiate"

```

```

2  class Drum
3  {
4      static int m,n;
5      static int[] xx1=new int[20]; static int[] yy1=new int[20];
6      static int[] xx2=new int[20]; static int[] yy2=new int[20];
7      static int[] x1=new int[3]; static int[] y1=new int[3];
8      static int[] x2=new int[3]; static int[] y2=new int[3];
9      static double x0, y0; // proiectie punct pe segment sau ...
10     static double x1s, y1s, x2s, y2s;
11
12    public static void main(String[] args) throws IOException
13    {
14        int i,j,i0,j0;
15        double d,dmin;
16        StreamTokenizer st=new StreamTokenizer(
17            new BufferedReader(new FileReader("drum.in")));
18        PrintWriter out=new PrintWriter(
19            new BufferedWriter(new FileWriter("drum.out")));
20
21        st.nextToken(); m=(int)st.nval;
22        for(i=0;i<m;i++)
23        {
24            st.nextToken(); xx1[i]=(int)st.nval;
25            st.nextToken(); yy1[i]=(int)st.nval;
26        }
27
28        st.nextToken(); n=(int)st.nval;
29        for(j=0;j<n;j++)
30        {
31            st.nextToken(); xx2[j]=(int)st.nval;
32            st.nextToken(); yy2[j]=(int)st.nval;
33        }
34
35        dmin=300; i0=-1; j0=-1;
36        for(i=0;i<m;i++)
37            for(j=0;j<n;j++)
38            {
39                d=dist(xx1[i],yy1[i],xx2[j],yy2[j]);
40                if(d<dmin)
41                {
42                    dmin=d; i0=i; j0=j;
43                    x1s=xx1[i]; y1s=yy1[i];
44                    x2s=xx2[j]; y2s=yy2[j];
45                }
46            }
47
48        x1[0]=xx1[(i0-1+m)%m]; y1[0]=yy1[(i0-1+m)%m];
49        x1[1]=xx1[i0]; y1[1]=yy1[i0];
50        x1[2]=xx1[(i0+1)%m]; y1[2]=yy1[(i0+1)%m];
51
52        x2[0]=xx2[(j0-1+n)%n]; y2[0]=yy2[(j0-1+n)%n];
53        x2[1]=xx2[j0]; y2[1]=yy2[j0];
54        x2[2]=xx2[(j0+1)%n]; y2[2]=yy2[(j0+1)%n];
55
56        // dist de la P(x1[i],y1[i]) la segm [A(x2[j],y2[j]);B(x2[j+1],y2[j+1])]
57        for(i=0;i<2;i++)
58            for(j=0;j<=1;j++)
59            {
60                d=dist(x1[i],y1[i],x2[j],y2[j],x2[j+1],y2[j+1]);
61
62                if(d<dmin)
63                {
64                    dmin=d;
65                    x1s=x1[i]; y1s=y1[i];
66                    x2s=x0; y2s=y0; // proiectia sau ...
67                }
68            }
69
70        // dist de la P(x2[j],y2[j]) la segm [A(x1[i],y1[i]);B(x1[i+1],y1[i+1])]
71        for(j=0;j<=2;j++)
72            for(i=0;i<=1;i++)
73            {
74                d=dist(x2[j],y2[j],x1[i],y1[i],x1[i+1],y1[i+1]);
75                if(d<dmin)
76                {
77                    dmin=d;

```

```

78         x2s=x2[j];  y2s=y2[j];
79         x1s=x0;      y1s=y0;    // proiectia sau ...
80     }
81 }
82
83 // rotunjire la 4 zecimali
84 if(x1s<0) x1s=((int)((x1s-0.00005)*10000))/10000.0;
85 else      x1s=((int)((x1s+0.00005)*10000))/10000.0;
86 if(y1s<0) y1s=((int)((y1s-0.00005)*10000))/10000.0;
87 else      y1s=((int)((y1s+0.00005)*10000))/10000.0;
88 if(x2s<0) x2s=((int)((x2s-0.00005)*10000))/10000.0;
89 else      x2s=((int)((x2s+0.00005)*10000))/10000.0;
90 if(y2s<0) y2s=((int)((y2s-0.00005)*10000))/10000.0;
91 else      y2s=((int)((y2s+0.00005)*10000))/10000.0;
92
93 out.println(x1s+" "+y1s+" "+x2s+" "+y2s);
94 out.close();
95 } // main(...)

96
97 static double dist(double xp1, double yp1, double xp2, double yp2)
98 {
99     return Math.sqrt((xp2-xp1)*(xp2-xp1)+(yp2-yp1)*(yp2-yp1));
100 } // dist(...)

101
102 //distanta de la (a0,b0) la segm [(a1,b1);(a2,b2)]
103 static double dist(double a0,double b0,double a1,double b1,double a2,double b2)
104 {
105     double m;
106
107     // segment vertical
108     if(Math.abs(a1-a2)<0.00001) // a1==a2 ... reale ...
109     {
110         if((b0-b1)*(b0-b2)<=0) // b0 intre b1 si b2
111             {x0=a1; y0=b0; return Math.abs(a1-a0);}
112         else // proiectia nu cade pe segment
113             if(dist(a0,b0,a1,b1)<dist(a0,b0,a2,b2))
114                 {
115                     x0=a1; y0=b1; return dist(a0,b0,a1,b1);
116                 }
117             else
118                 {
119                     x0=a2; y0=b2; return dist(a0,b0,a2,b2);
120                 }
121
122     // segment orizontal
123     if(Math.abs(b1-b2)<0.00001) // b1==b2 ... reale ...
124     {
125         if((a0-a1)*(a0-a2)<=0) // a0 intre a1 si a2
126             {x0=a0; y0=b1; return Math.abs(b1-b0);}
127         else // proiectia nu cade pe segment
128             if(dist(a0,b0,a1,b1)<dist(a0,b0,a2,b2))
129                 {
130                     x0=a1; y0=b1; return dist(a0,b0,a1,b1);
131                 }
132             else
133                 {
134                     x0=a2; y0=b2; return dist(a0,b0,a2,b2);
135                 }
136
137     // segment oblic
138     m=(b2-b1)/(a2-a1);
139     x0=(b0-b1+m*a1+a0/m)/(m+1/m);
140     y0=b1+m*(x0-a1);
141     if(((x0-a1)*(x0-a2)<=0)&& // x0 intre a1 si a2
142        ((y0-b1)*(y0-b2)<=0)) // y0 intre b1 si b2
143         return dist(x0,y0,a0,b0);
144     else
145         if(dist(a0,b0,a1,b1)<dist(a0,b0,a2,b2))
146             {
147                 x0=a1; y0=b1; return dist(a0,b0,a1,b1);
148             }
149         else
150             {
151                 x0=a2; y0=b2; return dist(a0,b0,a2,b2);
152             }
153 } // dist(...)

154 } // class

```

37.5.3 *Cod sursă

37.6 Pavări

prof. Doru Popescu Anastasiu, Slatina

Se dă un dreptunghi cu lungimea egală cu $2N$ centimetri și lățimea egală cu 3 centimetri.

Cerință

Să se determine numărul M al pavărilor distințe cu dale dreptunghiulare care au lungimea egală cu un centimetru și lățimea egală cu 2 centimetri.

Datele de intrare

Fișier de intrare: **pavari.in**

Linia 1: N - număr natural nenul, reprezentând jumătatea lungimii dreptunghiului.

Datele de ieșire

Fișier de ieșire: **pavari.out**

Linia 1: M - număr natural nenul, reprezentând numărul modalităților de a pava dreptunghiul.

Restricții și precizări

- $1 \leq N \leq 100$

Exemplu

pavari.in	pavari.out
2	11

Timp maxim de executare: 1 secundă/test

37.6.1 *Indicații de rezolvare

37.6.2 Rezolvare detaliată

Listing 37.6.1: pavari.java

```

1 import java.io.*; // x[n]=x[n-1]+2*y[n-1]; x[1]=3;
2 class Pavari // y[n]=y[n-1]+x[n]; y[1]=4;
3 {
4     public static void main(String[] args) throws IOException
5     {
6         int k,kk,n;
7         StreamTokenizer st=new StreamTokenizer(
8             new BufferedReader(new FileReader("pavari9.in")));
9         st.nextToken(); n=(int)st.nval;
10        int[] xv,xn,yv,yn;
11        xv=new int[1];
12        yn=new int[1];
13
14        xv[0]=3;
15        yv[0]=4;
16        xn=xv;
17        for(k=2;k<=n;k++)
18        {
19            xn=suma(xv,suma(yv,yv));
20            yn=suma(yv,xn);
21            xv=xn;
22            yv=yn;
23        }
24
25        PrintWriter out=new PrintWriter(
26             new BufferedWriter(new FileWriter("pavari.out")));
27        for(kk=xn.length-1;kk>=0;kk--) out.print(xn[kk]);
28        out.close();
29    } // main...

```

```
30
31     static int[] suma(int[] x, int[] y)
32     {
33         int nx=x.length,ny=y.length,i,t;
34         int nz;
35         if(nx>ny)  nz=nx+1; else nz=ny+1;
36         int[] z=new int[nz];
37
38         t=0;
39         for(i=0;i<nz;i++)
40         {
41             z[i]=t;
42             if(i<nx) z[i]+=x[i];
43             if(i<ny) z[i]+=y[i];
44             t=z[i]/10;
45             z[i]=z[i]%10;
46         }
47
48         if(z[nz-1]!=0) return z;
49         else
50         {
51             int[] zz=new int[nz-1];
52             for(i=0;i<nz-1;i++) zz[i]=z[i];
53             return zz;
54         }
55     } //suma
56 }
```

37.6.3 *Cod sursă

Capitolul 38

ONI 2000



Figura 38.1: Sigla ONI 2000

38.1 Castelul

prof. Dana Lica, Liceul "Ion Luca Caragiale", Ploiești

Unui muncitor i se încredințează o grea misiune. El are de lăcuit toate coridoarele unui castel. Pentru aceasta el primește o hartă în care sunt reprezentate coridoarele și cele N puncte de intersecție ale acestora (numerotate de la 1 la N). Unele puncte de intersecție sunt considerate de muncitor ca fiind "puncte speciale" deoarece ele sunt singurele puncte care îi permit ieșirea și intrarea în castel, prin intermediul unor uși.

Deoarece operația de lăciuire trebuie să fie efectuată în cel mai scurt timp, muncitorului i s-a impus ca orice corridor să nu fie traversat decât o singură dată, adică doar atunci când este lăcuit.

În aceste condiții, muncitorul trebuie să lăciuască toate coridoarele, folosindu-se eventual de "punctele speciale" ale castelului, plecând din orice intersecție de coridoare dorește și terminând operația în același punct.

Cerință: Determinați un traseu ciclic prin care operația este satisfăcută.

Date de intrare: Fișierul de intrare CASTEL.IN, conține:

Pe prima linie numărul N .

Pe a doua linie numărul de coridoare M

Pe următoarele M linii perechi de două numere reprezentând perechi de puncte între care există coridor ce trebuie lăcuit.

Pe următoarea linie numărul P reprezentând numărul de "puncte speciale".

Pe ultima linie sirul "punctelor speciale" separate prin câte un spațiu.

Datele de ieșire:

Ieșirea se va face în fișierul CASTEL.OUT în formatul următor:

Pe prima linie se va scrie mesajul DA sau NU, după cum există sau nu un traseu ciclic care respectă cerințele.

În caz afirmativ fișierul va conține pe a doua linie un sir de numere reprezentând punctele în ordinea de apariție pe traseu, despărțite după caz de secvență de caractere ' - ' (spațiu minus spațiu) sau ' * ' (spațiu asterisc spațiu).

O secvență de tipul

$i - j$

semnifică faptul că muncitorul a traversat și lăcuit corridorul dintre punctele i și j .

O secvență de tipul

$i * j$

semnifică faptul că muncitorul a părăsit castelul prin "punctul special" i și a revenit în castel prin "punctul special" j .

Restricții:

$1 \leq N \leq 5000$

$0 \leq M \leq 10000$

Exemplu:

CASTEL.IN	CASTEL.OUT poate conține:
6	DA
6	1 - 2 - 5 * 6 - 4 - 3 - 2 * 4 - 1
1 2	
2 3	
3 4	
4 1	
2 5	
4 6	
5	
1 2 5 4 6	

Timp maxim de execuție: 2 secunde / test.

Notă

Jumătate din teste folosite pentru evaluare vor avea dimensiunea $N \leq 100$

În cazul în care nici un test cu soluție nu va fi trecut, punctajele pentru testele pentru care răspunsul este NU nu vor fi acordate.

38.1.1 *Indicații de rezolvare

38.1.2 Rezolvare detaliată

Varianta 1:

Listing 38.1.1: castel1.java

```

1 import java.io.*;
2 class Castel
3 {
4     static int n,m;
5     static int ncs;      // nr coridoare speciale adaugate
6     static int pozfcs;  // pozitia finala in coridoare speciale
7
8     static int nods;    // nod start ciclul curent
9     static int poznods; // pozitia lui nods in ciclul final
10    static int nnc;     // nr noduri din ciclul curent
11    static int nnf;     // nr noduri din ciclul final
12    static int pozp;    // pozitia de permutare in vectorii muchiilor
13
14    static int[] c1=new int[12501]; // c1[k]= un capat al corridorului k
15    static int[] c2=new int[12501]; // c2[k]= alt capat al corridorului k
16    static int[] nc=new int[5001];  // nc[i]= nr coridoare incidente in i
17    static int[] cc=new int[12502]; // ciclul curent
18    static int[] cf=new int[12502]; // ciclul final
19
20    static boolean[] eSpecial=new boolean[5001];
21    static boolean[] eLacuit=new boolean[12501];
22
23    static int[] cs1=new int[2501]; // cs1[k]= un capat al corridorului special
24    static int[] cs2=new int[2501]; // cs2[k]= alt capat al corridorului special
25    static boolean[] csf=new boolean[2501]; // corridor special folosit deja ...
26
27    public static void main(String[] args) throws IOException
28    {
29        long t1,t2;
30        t1=System.currentTimeMillis();
31
32        int i,j,k;
33        int nps;           // nr puncte speciale
34        boolean ok;
```

```

35
36     StreamTokenizer st= new StreamTokenizer(
37         new BufferedReader(new FileReader("castel0.in")));
38     PrintWriter out= new PrintWriter(
39         new BufferedWriter(new FileWriter("castel.out")));
40
41     st.nextToken(); n=(int)st.nval;
42     st.nextToken(); m=(int)st.nval;
43
44     for(k=1;k<=m;k++)
45     {
46         st.nextToken(); i=(int)st.nval;
47         st.nextToken(); j=(int)st.nval;
48         c1[k]=i; c2[k]=j;
49         nc[i]++; nc[j]++;
50     }
51
52     st.nextToken(); nps=(int)st.nval;
53     for(k=1;k<=nps;k++)
54     {
55         st.nextToken(); i=(int)st.nval;
56         eSpecial[i]=true;
57     }
58
59     ok=true;
60     for(i=1;i<=n;i++)
61         if((nc[i]%2==1)&&!eSpecial[i]) {ok=false; break;}
62
63     if(!ok) out.println("NU");
64     else
65     {
66         out.println("DA");
67         rezolvare();
68         pozfcs=ncs;
69         out.print(cf[1]);
70         for(i=2;i<nnf;i++)
71             if(eCorridorSpecialNefolosit(cf[i-1],cf[i]))
72                 out.print(" * "+cf[i]);
73             else out.print(" - "+cf[i]);
74     }
75     out.close();
76
77     t2=System.currentTimeMillis();
78     System.out.println("Time: "+(t2-t1));
79 } //main
80
81 static boolean eCorridorSpecialNefolosit(int i, int j)
82 {
83     int k,aux;
84     boolean auxb;
85     for(k=1;k<=pozfcs;k++)
86     {
87         if(((cs1[k]==i)&&cs2[k]==j)|| (cs1[k]==j&&cs2[k]==i))&&!csf[k])
88         {
89             csf[k]=true;
90             if(k<pozfcs)
91             {
92                 aux=cs1[k]; cs1[k]=cs1[pozfcs]; cs1[pozfcs]=aux;
93                 aux=cs2[k]; cs2[k]=cs2[pozfcs]; cs2[pozfcs]=aux;
94                 auxb=csf[k]; csf[k]=csf[pozfcs]; csf[pozfcs]=auxb;
95                 pozfcs--;
96             }
97             return true;
98         } //if
99     } //for k
100    return false;
101 } //eCorridorSpecialNefolosit...
102
103 static void rezolvare() throws IOException
104 {
105     int aux;
106     ncs=0;
107     adaugCoridoare();
108     pozp=m+1;
109     nnc=0;
110     nods=1;

```

```

111     ciclu();
112
113     nnf=nnc;
114     for(int i=1;i<=nnc;i++) cf[i]=cc[i];      // primul ciclu
115
116     while(nnf<m+1)
117     {
118         nnc=0;
119         nods=nodstart();
120         ciclu();
121         inserare();
122     }
123 } // rezolvare()
124
125 static int succesor(int i)
126 {
127     int j=0,k,aux;
128     for(k=1;k<=m;k++)
129     {
130         if((!eLacuit[k]) && ((c1[k]==i) || (c2[k]==i)))
131         {
132             pozp--;
133             aux=c1[k]; c1[k]=c1[pozp]; c1[pozp]=aux;
134             aux=c2[k]; c2[k]=c2[pozp]; c2[pozp]=aux;
135
136             k=pozp;
137             if(c1[k]==i) j=c2[k]; else j=c1[k];
138             eLacuit[k]=true;
139             break;
140         } // if
141     } // for
142     return j;
143 } // succesor()
144
145 static int nodstart()
146 {
147     for(int i=1;i<=nnf;i++)
148         if(nc[cf[i]]>0) {poznods=i; return cf[i];}
149     return 0;
150 } // nodstart()
151
152 static void ciclu()
153 {
154     int i,j;
155     i=nods;
156     j=succesor(i);
157
158     cc[++nnc]=i;
159
160     while((i!=nods) || (nc[nods]>0))
161     {
162         cc[++nnc]=j;
163
164         nc[i]--; nc[j]--;
165         i=j; j=succesor(i);
166     }
167 } // ciclu()
168
169 static void inserare()
170 {
171     int i;
172     for(i=1;i<=nnf-poznods;i++)
173         cf[nnf+nnc-i]=cf[nnf-i+1];// deplasare ...
174     nnf=nnf+nnc-1;
175     for(i=1;i<=nnc;i++)
176         cf[poznods+i-1]=cc[i];    // inserare ciclul curent
177 } // inserare()
178
179 static void adaugCoridoare()// intre punctele speciale de grad impar
180 {
181     // nr puncte speciale k cu nc[k]=impar ... este ... par !!!
182     // daca aici adaug [i,j] unde i si j sunt speciale si de grad impar dar
183     // coridorul [i,j] exista deja, atunci ... este inca OK pentru problema !!!
184     int i,j;
185     i=1;
186     while(i<n)

```

```

187    {
188        while((i<n)&&((nc[i]&l)==0)) i++; // caut primul punct cu grad impar
189        if(i<n)
190        {
191            j=i+1;
192            while((j<n)&&((nc[j]&l)==0)) j++; // caut un punct j cu grad impar
193            // ar trebui sa nu existe corridorul [i,j] dar ... merge si asa ...
194            // cu atentie la afisarea traseului final "-"/*" pentru coridoarele
195            // care sunt si normale si speciale ...!!!
196            m++;
197            c1[m]=i; c2[m]=j; // am adaugat corridorul special [i,j] chiar daca ...
198            ncs++;
199            cs1[ncs]=i; cs2[ncs]=j; // am adaugat corridorul special [i,j]
200            nc[i]++; nc[j]++;
201            // maresc gradele
202            i=j+1;
203        } //if(i<n)
204    } // while(i<n)
205 } // adaugCoridoare()
206 } //class

```

Varianta 2:

Listing 38.1.2: castel2.java

```

1 import java.io.*; // cu sevenita de optimizare ...
2 class Castel // t9=2.1sec --> 1.1sec t10=3.1sec --> 1.5sec
3 {
4     static int pozfcl; // pozitie finala corridor lacuit (pentru optimizare!)
5     static int n,m;
6     static int[] c1=new int[12501]; // c1[k]= un capat al corridorului k
7     static int[] c2=new int[12501]; // c2[k]= alt capat al corridorului k
8     static int[] cu=new int[12501]; // corridorul urmator din ciclu
9     static boolean[] cs=new boolean[12501]; // cs[k]= este corridor special ?
10    static boolean[] cl=new boolean[12501]; // cl[k]= este corridor lacuit ?
11
12    static int[] nc=new int[5001]; // nc[i]= nr coridoare incidente in i
13    static boolean[] ps=new boolean[5001]; // ps[i]= i este punct special ?
14
15    static int nodscf; // nod start ciclu curent
16    static int nodscf; // nod start ciclu final
17    static int corscc; // corridor start din ciclu curent
18    static int corccc; // corridor curent din ciclu curent
19    static int corfcc; // corridor final din ciclu curent
20    static int coracc; // corridor anterior din ciclu curent
21    static int corscf; // corridor start din ciclu final
22    static int corins; // corridor ... inserare dupa el
23    static int nclacuite; // nr coridoare lacuite
24    static PrintWriter out;
25
26    public static void main(String[] args) throws IOException
27    {
28        long t1,t2;
29        t1=System.currentTimeMillis();
30
31        int i,j,k;
32        int nps; // nr puncte speciale
33        boolean ok;
34
35        StreamTokenizer st= new StreamTokenizer(
36            new BufferedReader(new FileReader("castel.in")));
37        out= new PrintWriter(new BufferedWriter(new FileWriter("castel.out")));
38
39        st.nextToken(); n=(int)st.nval;
40        st.nextToken(); m=(int)st.nval;
41
42        for(k=1;k<=m;k++)
43        {
44            st.nextToken(); i=(int)st.nval;
45            st.nextToken(); j=(int)st.nval;
46            c1[k]=i; c2[k]=j;
47            nc[i]++; nc[j]++;
48        }
49
50        st.nextToken(); nps=(int)st.nval;
51        for(k=1;k<=nps;k++)

```

```

52      {
53         st.nextToken(); i=(int)st.nval;
54         ps[i]=true;
55     }
56
57     ok=true;
58     for(i=1;i<=n;i++) if((nc[i]%2==1)&&!ps[i]) {ok=false; break;}
59
60     if(!ok) out.println("NU");
61     else
62     {
63        out.println("DA");
64        rezolvare();
65        afisCiclu(nodscf,corscf);
66    }
67    out.close();
68
69    t2=System.currentTimeMillis();
70    System.out.println("Time: "+(t2-t1));
71 } //main
72
73 static void rezolvare() throws IOException
74 {
75     int aux;
76     adaugCoridoare(); pozfcl=m+1;
77
78     // primul ciclu
79     nclacuite=0;
80     corscc=1;           // corridor start ciclul curent
81     corscf=corscc;
82     nodscc=c1[corscc]; // nod start ciclul curent
83     nodscf=nodscc;
84
85     ciclu();           // caut primul ciclu:
86
87     // alte cicluri
88     while(nclacuite<m)
89     {
90         corins=coridorInserare();
91         if(nc[c1[corins]]>0) nodscf=c1[corins]; else nodscf=c2[corins];
92
93         corscc=coridorStart();
94         ciclu();           // caut alt ciclu
95
96         // inserez noul ciclu in ciclul final anterior
97         aux=cu[corins];
98         cu[corins]=corscc;
99         cu[corfcc]=aux;
100    } // while
101 } // rezolvare()
102
103 static int corridorStart() // un corridor nelacuit cu un nod=nodscf
104 {
105     int k=1;
106     while(c1[k]==nodscf&&c2[k]==nodscf) k++; // corridor ...
107     return k;
108 }
109
110 static int corridorInserare() // nod (din ciclul "final") cu corridor nelacuit
111 {
112     int k;
113     k=corscf;
114     while((nc[c1[k]]==0)&&(nc[c2[k]]==0)) k=cu[k]; // corridor ...
115     return k;
116 }
117
118 static void ciclu()
119 {
120     int i,j;
121     i=nodscf;
122
123     if(c1[corscc]==i) j=c2[corscc]; else j=c1[corscc];
124     nclacuite++;
125     c1[corscc]=true;
126     coracc=corscc; // anterior=start
127     nc[i]--; nc[j]--;

```

```

128     i=j;
129     j=succesor(i);
130     while((i!=nodscc) || (nc[nodscc]>0))
131     {
132         nclacuite++;
133         cl[corfcc]=true;
134         cu[coracc]=corfcc;
135         coracc=corfcc;
136         nc[i]--;
137         nc[j]--;
138         i=j; j=succesor(i);
139     }
140     cu[corfcc]=corscc;
141 } // ciclul()
142
143 static int sucessor(int i) // determina si ... corfcc
144 {
145     int j,k,aux;
146     boolean auxb;
147     j=0;
148     for(k=1;k<=m;k++)
149     {
150         if(!cl[k])
151             if(c1[k]==i) {j=c2[k]; corfcc=k; break;} else
152                 if(c2[k]==i) {j=c1[k]; corfcc=k; break;}
153     } //-----*
154     // sevenita de optimizare: mut la sfarsit corridorul gasit !!!
155     if((j!=0)&&(corfcc>1)&&(corfcc<pozfcl-1)&&!cl[pozfcl-1])
156     {
157         pozfcl--;
158         aux=c1[pozfcl]; c1[pozfcl]=c1[corfcc]; c1[corfcc]=aux;
159         aux=c2[pozfcl]; c2[pozfcl]=c2[corfcc]; c2[corfcc]=aux;
160         auxb=cs[pozfcl]; cs[pozfcl]=cs[corfcc]; cs[corfcc]=auxb;
161         auxb=cl[pozfcl]; cl[pozfcl]=cl[corfcc]; cl[corfcc]=auxb;
162         corfcc=pozfcl;
163     } //-----*
164     return j;
165 }
166
167 static void afisCiclu(int i,int k0) throws IOException
168 { // ciclul pleaca din i pe corridorul k0
169     int j,k;
170     if(c1[k0]==i) j=c2[k0]; else j=c1[k0];
171     if(cs[k0]) out.print(i+" * "+j); else out.print(i+" - "+j);
172     k=cu[k0];
173     i=j;
174     while(k!=k0)
175     {
176         if(c1[k]==i) j=c2[k]; else j=c1[k];
177         if(cs[k]) out.print(" * "+j); else out.print(" - "+j);
178         i=j;
179         k=cu[k];
180     } // while
181     out.println("\n");
182 } // afisCiclul()
183
184 static void adaugCoridoare() // intre punctele speciale de grad impar
185 {
186     // nr puncte speciale k cu nc[k]=impar ... este ... par !!!
187     int i,j;
188     i=1;
189     while(i<n)
190     {
191         while((i<n)&&((nc[i]&1)==0)) i++; // caut primul punct cu grad impar
192         if(i<n)
193         {
194             j=i+1;
195             while((j<n)&&((nc[j]&1)==0)) j++; // caut alt punct cu grad impar
196             m++;
197             c1[m]=i; c2[m]=j; // am adaugat corridorul [i,j]
198             nc[i]++; nc[j]++; // maresc gradele
199             cs[m]=true;
200             i=j+1;
201         } //if(i<n)
202     } // while(i<n)
203 } // adaugCoridoare()

```

204 } //class

38.1.3 Cod sursă

Listing 38.1.3: castel.pas

```

1 program euler;
2
3 const ni='castel.in';
4     no='castel.out';
5     nmax=5000;
6     mmax=10000;
7     normal='-';
8     special='*';
9
10 type lista=^camp;
11     camp=record
12         inf:integer;
13         tip:byte;
14         leg:lista;
15     end;
16     vector_int=array [1..2*mmax+nmax] of integer;
17     pvector_int=^vector_int;
18     vector_byte=array [1..2*mmax+nmax] of byte;
19     pvector_byte=^vector_byte;
20     sir_int=array [1..mmax+1] of integer;
21     psir_int=^sir_int;
22     sir_byte=array [1..mmax+1] of byte;
23     psir_byte=^sir_byte;
24
25 var f:text;
26     a,b:pvector_int;
27     s,t:pvector_byte;
28     li,ls,g:array [1..nmax] of integer;
29     spc:array [1..nmax] of byte;
30     tmp:psir_int;
31     fel:psir_byte;
32     n,m,p,i,j,k,kk,start,nod,vecin,med,aux,lng,tip:integer;
33     prim,prev,baza,last,crt,x:lista;
34     ok:boolean;
35
36 procedure memorie;
37 begin
38     new(a);
39     new(b);
40     new(t);
41     new(s);
42     new(tmp);
43     new(fel);
44 end;
45
46 procedure load;
47 begin
48     assign(f,ni);reset(f);
49     readln(f,n);
50     readln(f,m);
51     for k:=1 to m do begin
52         readln(f,i,j);
53         a^[k]:=i;
54         b^[k]:=j;
55         t^[k]:=0;
56         s^[k]:=0;
57         a^[m+k]:=j;
58         b^[m+k]:=i;
59         t^[m+k]:=0;
60         s^[m+k]:=0;
61         g[i]:=g[i]+1;
62         g[j]:=g[j]+1;
63     end;
64     m:=2*m;
65     for i:=1 to n do
66         spc[i]:=0;

```

```

67      readln(f,p);
68      for i:=1 to p do begin
69          read(f,j);
70          spc[j]:=1;
71          end;
72      close(f);
73  end;
74
75 procedure nu;
76 begin
77    ok:=true;
78    for i:=1 to n do
79        if (g[i] and 1)=1 then
80            if spc[i]=0 then ok:=false;
81    if not ok then begin
82        assign(f,no); rewrite(f);
83        writeln(f,'NU');
84        close(f);
85        halt;
86        end;
87    end;
88
89 procedure adauga_muchii;
90 begin
91    k:=1;
92    repeat
93        while (k<=n) and ((g[k] and 1)=0) do k:=k+1;
94        if k<=n then begin
95            kk:=k+1;
96            while (g[kk] and 1)=0 do kk:=kk+1;
97            m:=m+1;
98            a^[m]:=k;
99            b^[m]:=kk;
100           t^[m]:=1;
101           s^[m]:=0;
102           m:=m+1;
103           a^[m]:=kk;
104           b^[m]:=k;
105           t^[m]:=1;
106           s^[m]:=0;
107           g[k]:=g[k]+1;
108           g[kk]:=g[kk]+1;
109           k:=kk+1;
110           end;
111       until k>n;
112   end;
113
114 procedure qsort(var c:pvector_int;li,ls:integer);
115 var i,j:integer;
116 begin
117     i:=li;
118     j:=ls;
119     med:=c^[(li+ls) shr 1];
120     repeat
121         while c^[i]<med do i:=i+1;
122         while c^[j]>med do j:=j-1;
123         if i<=j then begin
124             aux:=a^[i];
125             a^[i]:=a^[j];
126             a^[j]:=aux;
127             aux:=b^[i];
128             b^[i]:=b^[j];
129             b^[j]:=aux;
130             aux:=t^[i];
131             t^[i]:=t^[j];
132             t^[j]:=aux;
133             i:=i+1;
134             j:=j-1;
135             end;
136         until i>j;
137         if li<j then qsort(c,li,j);
138         if i<ls then qsort(c,i,ls);
139     end;
140
141 procedure sort;
142 begin

```

```

143      qsort(a,1,m);
144      k:=1;
145      repeat
146          kk:=k;
147          while (kk<=m) and (a^[kk]=a^[k]) do kk:=kk+1;
148          li[a^[k]]:=k;
149          ls[a^[k]]:=kk-1;
150          qsort(b,k,kk-1);
151          k:=kk;
152      until k>m;
153  end;
154
155 function cauta(var ce:integer;li,ls:integer):integer;
156 begin
157     med:=(li+ls) shr 1;
158     if ce<b^[med] then cauta:=cauta(ce,li,med-1)
159     else
160         if ce=b^[med] then cauta:=med
161         else cauta:=cauta(ce,med+1,ls);
162 end;
163
164 procedure ciclu(nod:integer);
165 begin
166     tmp^[1]:=nod;
167     lng:=1;
168     repeat
169         nod:=tmp^[lng];
170         start:=li[nod];
171         while s^[start]=1 do start:=start+1;
172         vecin:=b^[start];
173         tip:=t^[start];
174         li[nod]:=start+1;
175         s^[start]:=1;
176         p:=cauta(nod,li[vecin],ls[vecin]);
177         if t^[p]<>tip then begin
178             if p>1 then
179                 if b^[p-1]=nod then
180                     if t^[p-1]=tip then p:=p-1;
181             if p<m then
182                 if b^[p+1]=nod then
183                     if t^[p+1]=tip then p:=p+1;
184             end;
185             s^[p]:=1;
186             lng:=lng+1;
187             tmp^[lng]:=vecin;
188             fel^[lng]:=tip;
189             g[nod]:=g[nod]-1;
190             g[vecin]:=g[vecin]-1;
191         until tmp^[lng]=tmp^[1];
192 end;
193
194 procedure init;
195 begin
196     ciclu(1);
197     new(prim);
198     prim^.inf:=1;
199     prim^.leg:=nil;
200     prev:=prim;
201     for i:=2 to lng do begin
202         new(x);
203         x^.inf:=tmp^[i];
204         x^.tip:=fel^[i];
205         x^.leg:=nil;
206         prev^.leg:=x;
207         prev:=x;
208         end;
209     end;
210
211 procedure work;
212 begin
213     crt:=prim;
214     repeat
215         while (crt<>nil) and (g[crt^.inf]=0) do crt:=crt^.leg;
216         if crt<>nil then begin
217             ciclu(crt^.inf);
218             new(baza);

```

```

219         baza^.inf:=1;
220         baza^.leg:=nil;
221         prev:=baza;
222         for i:=2 to lng do begin
223             new(x);
224             x^.inf:=tmp^[i];
225             x^.tip:=fel^[i];
226             x^.leg:=nil;
227             prev^.leg:=x;
228             prev:=x;
229             end;
230         last:=prev;
231         last^.leg:=crt^.leg;
232         crt^.leg:=baza^.leg;
233         baza^.leg:=nil;
234         dispose(baza);
235         end;
236     until crt=nil;
237 end;
238
239 procedure scrie;
240 begin
241     assign(f,no); rewrite(f);
242     writeln(f,'DA');
243     write(f,prim^.inf);
244     prim:=prim^.leg;
245     while prim<>nil do begin
246         if prim^.tip=0 then write(f,' ',normal,' ')
247         else write(f,' ',special,' ');
248         write(f,prim^.inf);
249         prim:=prim^.leg;
250         end;
251     close(f);
252 end;
253
254 begin {programul principal}
255     memorie;
256     load;
257     nu;
258     adauga_muchii;
259     sort;
260     init;
261     work;
262     scrie;
263 end.

```

38.2 Drumuri scurte

prof. Stelian Ciurea, Universitatea "Lucian Blaga", Sibiu

Se consideră un graf turneu cu n vârfuri. Numim *graf turneu* un graf orientat la care între oricare două vârfuri se află un arc și numai unul.

Cerință:

Să se găsească o dispunere a arcelor grafului astfel încât între oricare două noduri să existe un drum de lungime 1 sau 2.

Date de intrare:

Fișierul de intrare GRAF.IN, conține pe prima linie valoarea lui n .

Datele de ieșire:

Ieșirea se va face în fișierul GRAF.OUT care va conține:

- matricea de adiacență a grafului care îndeplinește cerințele problemei (pe linia i a fișierului se va afla linia i a matricei) în cazul în care există soluție;

- textul "fără soluție" (cu litere mici!), dacă nu există nici o dispunere a arcelor.

Restricții:

$2 \leq n \leq 250$

Exemplu:

GRAF.IN	GRAF.OUT poate conține:
6	0 1 1 1 0 0
	0 0 1 1 1 0
	0 0 0 1 0 1
	0 0 0 0 1 1
	1 0 1 0 0 1
	1 1 0 0 0 0

Explicații: matricea de adiacență din GRAF.OUT corespunde grafului din imaginea de mai jos:

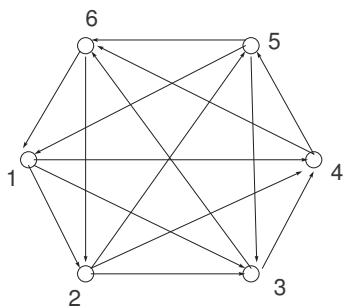


Figura 38.2: Drum scurt

Timp maxim de execuție pe test: 3 secunde

38.2.1 *Indicații de rezolvare

38.2.2 Rezolvare detaliată

Varianta 1:

Listing 38.2.1: graf1.java

```

1 import java.io.*;
2 class DrumuriScurte
3 {
4     static int n;
5     static byte[][] a=new byte[251][251];
6
7     public static void main(String[] args) throws IOException
8     {
9         long t1,t2;
10        t1=System.currentTimeMillis();
11
12        int i,j;
13        StreamTokenizer st=new StreamTokenizer(
14            new BufferedReader(new FileReader("graf.in")));
15        PrintWriter out=new PrintWriter(
16            new BufferedWriter(new FileWriter("graf.out")));
17
18        st.nextToken(); n=(int)st.nval;
19
20        if((n==2) || (n==4)) out.println("fara solutie");
21        else
22        {
23            solutie();
24            for(i=1;i<=n; i++)
25            {
26                for(j=1; j<=n; j++) out.print(a[i][j]+" ");
27                out.println();
28            }
29        }
30        out.close();
31        t2=System.currentTimeMillis();

```

```

32     System.out.println("Time: "+(t2-t1));
33 } // main(...)

34
35 static void solutie()
36 {
37     int i,k;
38     if(n%2==1) // n=impar >= 3
39     {
40         a[1][2]=a[2][3]=a[3][1]=1; // pentru 3
41         k=5;
42     }
43     else // n=par >= 6
44     {
45         // pentru 6
46         a[1][2]=a[1][3]=a[1][4]=a[2][3]=a[2][4]=a[2][5]=1;
47         a[3][4]=a[3][6]=a[4][5]=a[4][6]=1;
48         a[5][1]=a[5][3]=a[5][6]=a[6][1]=a[6][2]=1;
49         k=8;
50     }
51
52     while(k<=n)
53     {
54         for(i=1;i<=k-2;i++) a[i][k-1]=a[k][i]=1;
55         a[k-1][k]=1;
56         k=k+2;
57     }
58 } // solutie()
59 } // class

```

Varianta 2:

Listing 38.2.2: graf2.java

```

1 import java.io.*; // 0.40 sec --> 0.14 sec... cu scriere string in fisier
2 class DrumuriScurte
3 {
4     static int n;
5     static byte[][] a=new byte[251][251];
6     static char[] c;
7     static String s="123";
8
9     public static void main(String[] args) throws IOException
10    {
11        long t1,t2;
12        t1=System.currentTimeMillis();
13
14        int i,j;
15        StreamTokenizer st=new StreamTokenizer(
16            new BufferedReader(new FileReader("graf.in")));
17        PrintWriter out=new PrintWriter(
18            new BufferedWriter(new FileWriter("graf.out")));
19
20        st.nextToken(); n=(int)st.nval;
21
22        c=new char[2*n-1];
23        for(j=0;j<2*n-1;j++) c[j]=32;
24
25        if((n==2) || (n==4)) out.println("fara solutie");
26        else
27        {
28            solutie();
29            for(i=1;i<=n;i++)
30            {
31                for(j=1;j<=n;j++) c[2*(j-1)]=(char)(a[i][j]+'0');
32                s=new String(c);
33                out.println(s);
34            }
35        }
36        out.close();
37        t2=System.currentTimeMillis();
38        System.out.println("Time: "+(t2-t1));
39    } // main(...)

40
41 static void solutie()
42 {
43     int i,k;

```

```

44      if (n%2==1) // n=impar >= 3
45      {
46          a[1][2]=a[2][3]=a[3][1]=1;      // pentru 3
47          k=5;
48      }
49      else // n=par >= 6
50      {
51          // pentru 6
52          a[1][2]=a[1][3]=a[1][4]=a[2][3]=a[2][4]=a[2][5]=1;
53          a[3][4]=a[3][6]=a[4][5]=a[4][6]=1;
54          a[5][1]=a[5][3]=a[5][6]=a[6][1]=a[6][2]=1;
55          k=8;
56      }
57
58      while (k<=n)
59      {
60          for (i=1;i<=k-2;i++) a[i][k-1]=a[k][i]=1;
61          a[k-1][k]=1;
62          k=k+2;
63      }
64  } // solutie()
65 } // class

```

38.2.3 Cod sursă

Listing 38.2.3: graf.pas

```

1 const np=6;
2 var s:array[1..np*(np-1) div 2] of byte;
3     a:array[1..250,1..250] of byte;
4     ok:boolean;
5     i,j,n:byte; ct:longint;
6     fi,fo:text;
7
8 procedure final;
9 var i,j,k:byte;
10 begin
11     k:=1;inc(ct);
12     for i := 1 to np do
13         for j := i+1 to np do
14             begin a[i,j]:=abs(s[k]-1); a[j,i]:=s[k];
15             inc(k);
16         end;
17     for i := 1 to np do
18         for j := 1 to np do
19             if i>j then
20                 begin
21                     ok := false;
22                     if a[i,j]=1 then begin ok := true;continue; end;
23                     for k := 1 to np do
24                         if a[i,k]=1 then
25                             if a[k,j]=1 then begin ok := true; break; end;
26                         if not ok then exit;
27                 end;
28             for i := 1 to n do
29                 begin
30                     for j := 1 to n do write(fo,a[i,j]:2);
31                     writeln(fo)
32                 end; close(fo); halt(0);
33             end;
34
35 procedure back(k:byte);
36 var i:byte;
37 begin
38     if k=np*(np-1) div 2 +1 then final
39     else for i := 0 to 1 do
40         begin s[k]:=i;
41             back(k+1)
42         end;
43     end;
44
45 procedure rezolva(k:integer);

```

```

46 var i,j:byte;
47 begin
48 if k=3 then begin a[1,2]:=1;a[2,3]:=1;a[3,1]:=1; exit end;
49 if k=6 then begin back(1); exit end;
50 for j := 1 to k-2 do
51   a[k,j]:=1;
52 for i := 1 to k-2 do a[i,k-1]:=1;
53 a[k-1,k]:=1;
54 rezolva(k-2);
55 end;
56
57 begin
58 assign(fi,'graf.in');
59 reset(fi);
60 assign(fo,'graf.out');
61 rewrite(fo);
62 readln(fi,n); close(fi);
63 if n in [1,2,4] then begin write(fo,'NU'); close(fo); exit end;
64 rezolva(n);
65 for i := 1 to n do
66 begin
67   for j := 1 to n do write(fo,a[i,j]:2);
68   writeln(fo)
69 end; close(fo);
70 inc(ct);
71 end.

```

38.3 Lac - Scafandru

prof. Dan Grigoriu, Colegiul Național "Tudor Vianu", București

Un lac are forma cubică, cu latura M (număr natural). El este împărțit în M^3 poziții elementare. O poziție(o căsuță) este identificată prin coordonatele n, l, c , ca într-un spațiu tridimensional, unde:

n reprezintă nivelul (1 pentru cel din imediata apropiere a suprafeței apei, M pentru nivelul de pe fundul lacului),

l reprezintă linia (linia 1 este linia cea mai apropiată de privitor)

c reprezintă coloana (coloana 1 este cea din stânga privitorului).

Exemplu pentru $m = 4$.

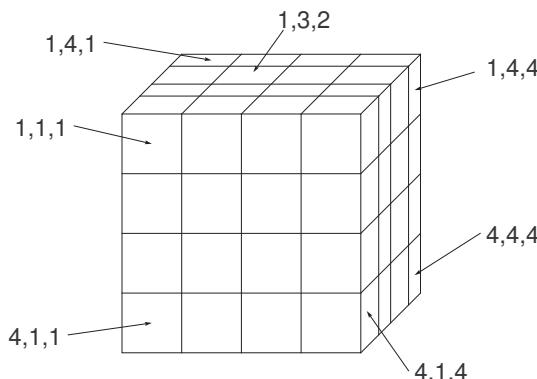


Figura 38.3: Scafandru

Exemple de deplasări elementare:

- din poziția $(4, 2, 2)$ se poate ajunge la una din pozițiile: $(3, 1, 1)$, $(3, 1, 2)$, $(3, 1, 3)$, $(3, 2, 1)$, $(3, 2, 2)$, $(3, 2, 3)$, $(3, 3, 1)$, $(3, 3, 2)$, $(3, 3, 3)$;

- din poziția $(2, 1, 1)$ se poate ajunge în una din pozițiile: $(1, 1, 1)$, $(1, 1, 2)$, $(1, 2, 1)$, $(1, 2, 2)$.

Există o căsuță specială, la suprafața apei, considerată pe nivelul 0 și anume cea în care trebuie să ajungă în final un scafandru ce pleacă de pe fundul lacului.

În lac există poziții elementare considerate ca fiind stații de lucru ce au de transmis la suprafață anumite mesaje. Aceste stații pot fi oricâte și pot fi dispuse oriunde în cub (chiar și în poziția de plecare a scafandrului).

Scafandrul se poate deplasa din poziția sa numai într-o poziție de pe nivelul imediat superior (de la nivelul i la nivelul $i - 1$, pentru $1 \leq i \leq m$) și care diferă ca linie și coloană cu cel mult 1 față de poziția precedentă. Deci dintr-o poziție se poate ajunge (vezi figura) în maxim 9 poziții pe nivelul imediat superior.

Scafandrul se află pe fundul lacului la coordonatele m, l_1, c_1 . El trebuie să ajungă la suprafața lacului (adică deasupra primului nivel al cubului), într-o poziție de coordonate $0, l_2, c_2$, adică imediat deasupra poziției $1, l_2, c_2$ din cub.

Cerință: Trebuie determinat ce drum trebuie să aleagă scafandrul pentru a duce la destinație cât mai multe mesaje, respectând restricțiile din enunț.

Date de intrare:

Fișierul de intrare LAC.IN, conține:

pe prima linie cinci numere $ml_1c_1l_2c_2$ despărțite prin căte un spațiu, reprezentând m dimensiunea cubului, l_1, c_1 linia și coloana poziției de plecare, iar l_2, c_2 linia și coloana poziției de sosire;

pe a doua linie numărul y de stații din lac;

pe următoarele y linii seturi de căte patru numere nlp . Primele trei numere nlc reprezintă coordonatele unei stații iar ultimul număr p reprezintă numărul de mesaje ce trebuesc transmise de stația respectivă;

Datele de ieșire:

Ieșirea se va face în fișierul LAC.OUT în formatul următor:

pe prima linie se va scrie numărul maxim de mesaje aduse la suprafață.

pe următoarele $m + 1$ linii se vor scrie seturi de patru numere nlp , primele trei numere reprezentând coordonatele stațiilor vizitate (în ordinea parcurgerii), iar ultimul număr reprezentând numărul de mesaje culese din stația respectivă. Prima linie din cele $m + 1$ reprezintă punctul de plecare, iar ultima punctul de sosire (nivelul 0).

Restricții:

$2 \leq M \leq 20$

$0 \leq P \leq 100$

Exemplu:

LAC.IN	LAC.OUT poate conține:
3 2 2 2	22
6	3 2 2 5
1 3 1 10	2 3 2 7
2 1 3 9	1 3 1 10
2 2 1 6	0 2 2 0
2 3 2 7	
2 3 3 8	
3 2 2 5	

Timp maxim de execuție: 1 secundă/ per test.

Notă:

Nu este permisă ieșirea din lac decât prin punctul de sosire.

În punctul de sosire nu se află nici o stație.

Dacă pentru o anumită poziție toate mutările posibile sunt în căsuțe fără stații, atunci se va scrie în fișierul de ieșire o linie cu coordonatele căsuței alese și valoarea 0 pentru numărul de mesaje (ca pentru o stație cu 0 mesaje).

38.3.1 Indicații de rezolvare

Rezolvarea problemei se bazează pe *tehnica programării dinamice*.

Fie A matricea cu lacul. S-a ales un tablou cubic pentru a putea exista un drum de la orice poziție de pe fundul lacului la orice punct final posibil.

Folosim o matrice auxiliară C , tot cu $m * m * m$ elemente, unde $C[i, j, k]$ reprezintă numărul maxim de mesaje culese pe un drum de la (i, j, k) la ieșire, adică la $(0, l_2, c_2)$.

Se "vede" ușor următoarea relație de recurență, pe care se bazează programul:

$$\begin{aligned} C[i, j, k] = A[i, j, k] + \max(C[i-1, j+1, k+1], & C[i-1, j+1, k], C[i-1, j+1, k-1] \\ & C[i-1, j, k+1], C[i-1, j, k], C[i-1, j, k-1]) \\ & C[i-1, j-1, k+1], C[i-1, j-1, k], C[i-1, j-1, k-1] \end{aligned}$$

După ce este facută o parcurgere pentru completarea matricii C vom avea în $C[m, l_1, c_1]$ numărul maxim de mesaje culese.

Ideea procedurii *Drum*:

Se testează cum s-a ajuns în $C[i, j, k]$ la valoarea curentă, adică de la ce casuță s-a ajuns la cea curentă și se repetă recursiv apelul pentru testarea căsuței următoare din drumul scafandrului.

Pe masură ce se descoperă căsuța următoare, ea este trecută (prin linia sa) în fișier.

38.3.2 Rezolvare detaliată

Listing 38.3.1: lac.java

```

1 import java.io.*;
2 class Scafandrul
3 {
4     static int m,ns;           // dim lac, nr statii
5     static int[][][] a;        // lacul ... cu nr mesaje
6     static int[][][] c;        // suma max mesaje pana aici
7     static int[][][] d;        // din ce directie a ajuns
8     static int l1,c1,l2,c2;    // pozitii initiale si finale (linie,coloana)
9     static PrintWriter out;
10    static StreamTokenizer st;
11
12    public static void main(String[] args) throws IOException
13    {
14        citire();
15        rezolvare();
16        afisare();
17    } // main...
18
19    static void citire() throws IOException
20    {
21        int niv,lin,col,s;
22        st=new StreamTokenizer(
23            new BufferedReader(new FileReader("lac.in")));
24        st.nextToken(); m=(int)st.nval;
25        st.nextToken(); l1=(int)st.nval;
26        st.nextToken(); c1=(int)st.nval;
27        st.nextToken(); l2=(int)st.nval;
28        st.nextToken(); c2=(int)st.nval;
29
30        a=new int[m+1][m+1][m+1];
31        c=new int[m+1][m+1][m+1];
32        d=new int[m+1][m+1][m+1];
33
34        st.nextToken(); ns=(int)st.nval;
35
36        for(s=1;s<=ns; s++) // statia
37        {
38            st.nextToken(); niv=(int)st.nval;
39            st.nextToken(); lin=(int)st.nval;
40            st.nextToken(); col=(int)st.nval;
41            st.nextToken(); a[niv][lin][col]=(int)st.nval;
42        }
43    } // citire()
44
45    static void rezolvare()
46    {
47        int i,j,niv,lin,col;
48        for(niv=m-1;niv>=0; niv--)
49        for(lin=1;lin<=m; lin++)
50        for(col=1;col<=m; col++)
51        {
52            c[niv][lin][col]=-1;
53            for(i=-1;i<=1; i++)
54            {
55                if((lin+i<1) || (lin+i>m) ||
56                    (lin+i>l1-(m-niv-1)) || (lin+i>l1+(m-niv-1))) continue;
57                for(j=-1;j<=1; j++)
58                {
59                    if((col+j<1) || (col+j>m) ||
60                        (col+j<c1-(m-niv-1)) || (col+j>c1+(m-niv-1))) continue;
61                    if(c[niv][lin][col]< c[niv+1][lin+i][col+j]+a[niv+1][lin+i][col+j])
62                    {
63                        c[niv][lin][col]=c[niv+1][lin+i][col+j]+a[niv+1][lin+i][col+j];

```

```

64         d[niv][lin][col]=(i+1)*3+(j+2);
65     }
66     } // for j/col
67   } // for i/lin
68 } // col
69 } // rezolvare
70
71 static void afisare() throws IOException
72 {
73   out=new PrintWriter(new BufferedWriter(new FileWriter("lac.out")));
74   out.println(c[0][12][c2]);
75   drum(0,12,c2);
76   out.close();
77 } // afisare()
78
79 static void drum(int niv, int lin, int col) throws IOException
80 {
81   int i,j;
82   i=(-1+d[niv][lin][col])/3; // 0,1,2
83   j=(-1+d[niv][lin][col])%3; // 0,1,2
84   i--;
85   j--;
86   if(niv<m) drum(niv+1,lin+i,col+j);
87   out.println(niv+" "+lin+" "+col+" "+a[niv][lin][col]);
88 } // drum(...)
89 } // class

```

38.3.3 *Cod sursă

38.4 Pariul broscuțelor

prof. Stelian Ciurea, Universitatea "Lucian Blaga", Sibiu

Câteva broscuțe din orașul C..., mergând la plimbare prin parcul din centrul orașului și descorezind un mozaic asemănător unei table de săh de dimensiune nn , au inventat un joc cu următoarele reguli:

fiecare broscuță pornește din centrul pătrățelului aflat în colțul din stânga-sus al tablei, trece prin fiecare pătrățel al mozaicului, și se întoarce în pătrățelul de unde a pornit,

poate sări din centrul unui pătrățel oarecare în centrul oricărui din pătrățelele vecine aflate în direcțiile: N, S, E, V, NE, NV, SE, SV,

dacă se unesc centrele pătrățelelor în ordinea în care au fost parcurse, se obține o linie închisă care nu trebuie să se autointersecteze.

Una din broscuțe - mai nechibzuită - face pariu pe greutatea ei în musculițe, că poate găsi o ordine în care să acopere cu sărituri mozaicul astfel încât, lungimea drumului parcurs să nu poată fi depășită. Lungimea este în sens geometric: drumul dintre centrele a două pătrate cu o latură comună este 1, iar între centrele a două pătrate ce au doar un vîrf comun, distanța este $\sqrt{2}$.

Cerință:

Din păcate, broscuța nu prea știe cum să procedeze, aşa că vă roagă să îi indicați ordinea în care trebuie să parcurgă pătrățelele tablei astfel încât să nu piardă pariu.

Date de intrare:

Fisierul de intrare FROG.IN, conține pe prima linie valoarea lui n .

Date de ieșire:

În fisierul FROG.OUT se vor scrie, pe linia i ($i = 1 \dots n^2 + 1$), două perechi de valori separate printr-un spațiu, reprezentând coordonatele pătrățelului unde trebuie să sară broscuța la pasul i (se consideră că în colțul din stânga sus avem coordonatele 1,1 iar localizarea unui pătrățel se face ca în matrice).

Restricții:

$2 \leq n \leq 250$

Exemplu:

FROG.IN	FROG.OUT poate conține:
4	1 1
	2 2
	2 1
	3 1
	3 2
	4 1
	4 2
	3 3
	4 3
	4 4
	3 4
	2 3
	2 4
	1 4
	1 3
	1 2
	1 1

Timp maxim de execuție pe test: 3 secunde

38.4.1 *Indicații de rezolvare

38.4.2 Rezolvare detaliată

Varianta 1:

Listing 38.4.1: frog1.java

```

1 import java.io.*;      // cu stop() ... pentru depanare ...
2 class PariulBroscutelor
3 {
4     public static void main(String[] args) throws IOException
5     {
6         int i,j,n;
7         StreamTokenizer st=new StreamTokenizer(
8             new BufferedReader(new FileReader("frog.in")));
9         PrintWriter out=new PrintWriter(
10            new BufferedWriter(new FileWriter("frog.out")));
11         st.nextToken(); n=(int)st.nval;
12         if(n==2)
13         {
14             out.println(1+" "+1);
15             out.println(2+" "+1);
16             out.println(2+" "+2);
17             out.println(1+" "+2);
18             out.println(1+" "+1);
19         }
20         else
21         {
22             i=j=1;    System.out.println(i+" "+j);
23             i++;    System.out.println(i+" "+j); stop();
24
25             while(j+2<=n-1)
26             {
27                 j++;    // E
28                 System.out.println(i+" "+j);
29                 while(j>1) {i++; j--; System.out.println(i+" "+j);} // SV
30                 i++;    // S
31                 System.out.println(i+" "+j); stop();
32                 while(i>2) {i--; j++; System.out.println(i+" "+j);} // NE
33                 stop();
34             }
35             if(j==n-2) {j++; System.out.println(i+" "+j);} // E
36             else {i++; System.out.println(i+" "+j);} // S
37             stop();
38

```

```

39         while(i+2<=n-1)
40     {
41         while(i<n) {i++; j--; System.out.println(i+" "+j);}      // SV
42         j++;          // E
43         System.out.println(i+" "+j); stop();
44         while(j<n-1) {i--; j++; System.out.println(i+" "+j);}    // NE
45         stop();
46         i++;          // S
47         System.out.println(i+" "+j); stop();
48     }
49
50     i++; j--;      // SV
51     System.out.println(i+" "+j);
52     j++;          // E
53     System.out.println(i+" "+j);
54     j++;          // E
55     System.out.println(i+" "+j);
56     stop();
57
58     for(i=n-1;i>=1;i--) System.out.println(i+" "+n);
59     stop();
60
61     for(j=n-1;j>=1;j--) System.out.println(1+" "+j);
62     stop();
63   }
64   out.close();
65 } // main...
66
67 static void stop() throws IOException
68 {
69   BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
70   String s=br.readLine();
71 } // stop()
72 } // class

```

Varianta 2:

Listing 38.4.2: frog2.java

```

1 import java.io.*; // varianta finala ...
2 class PariulBroscutelor
3 {
4   public static void main(String[] args) throws IOException
5   {
6     int i,j,n;
7     StreamTokenizer st=new StreamTokenizer(
8       new BufferedReader(new FileReader("frog.in")));
9     PrintWriter out=new PrintWriter(
10       new BufferedWriter(new FileWriter("frog.out")));
11     st.nextToken(); n=(int)st.nval;
12     if(n==2)
13     {
14       out.println(1+" "+1);
15       out.println(2+" "+1);
16       out.println(2+" "+2);
17       out.println(1+" "+2);
18       out.println(1+" "+1);
19     }
20   else
21   {
22     i=j=1;  out.println(i+" "+j);    // start
23     i++;   out.println(i+" "+j);    // S
24
25     while(j+2<=n-1)           // zona NV fara linia 1
26     {
27       j++; out.println(i+" "+j);    // E
28       while(j>1) {i++; j--; out.println(i+" "+j);}    // SV
29
30       i++; out.println(i+" "+j);    // S
31       while(i>2) {i--; j++; out.println(i+" "+j);}    // NE
32     }
33     if(j==n-2) {j++; out.println(i+" "+j);}    // E
34     else       {i++; out.println(i+" "+j);}    // S
35
36     while(i+2<=n-1)           // zona SE fara coloana n
37   {

```

```

38     while(i<n) { i++; j--; out.println(i+" "+j); } // SV
39     j++; out.println(i+" "+j); // E
40
41     while(j<n-1) { i--; j++; out.println(i+" "+j); } // NE
42     i++; out.println(i+" "+j); // S
43   }
44
45     i++; j--; out.println(i+" "+j); // SV
46     j++; out.println(i+" "+j); // E
47     j++; out.println(i+" "+j); // E
48
49     for(i=n-1;i>=1;i--) out.println(i+" "+n); // N
50     for(j=n-1;j>=1;j--) out.println(1+" "+j); // V
51   }
52   out.close();
53 } // main...
54 } // class

```

38.4.3 Cod sursă

Listing 38.4.3: frog.pas

```

1 var {a:array[1..150,1..150] of byte;}
2   ok:boolean;
3   d,k,i,j,n:word; lung:real;
4   fi,fo:text; sens,pas:longint;
5 procedure scrie;
6 begin writeln(fo,i,' ',j);
7 end;
8
9 begin
10  assign(fi,'frog.in');
11  reset(fi);
12  assign(fo,'frog.out');
13  rewrite(fo);
14  readln(fi,n);
15  if n=2 then
16    begin writeln(fo,1,' ',1);
17      writeln(fo,2,' ',1);
18      writeln(fo,2,' ',2);
19      writeln(fo,1,' ',2);
20      writeln(fo,1,' ',1);
21      close(fo);
22      exit;
23    end;
24  lung:=4*(n-1)+(n-2)*(n-2)*sqrt(2);
25 { a[1,1]:=1; } i:=1; j:=1; scrie;
26 { a[2,1]:=2; } i:=2; j:=1; scrie;
27 { a[2,2]:=3; } i:=2; j:=2; scrie;
28 sens:=1;
29 pas:=4;
30 for d := 1 to n-3 do {pana deasupra DS}
31 begin
32   for k := 1 to d do
33     begin
34       i:=i+sens;
35       j:=j-sens;
36 { a[i,j]:=pas; } scrie;
37       inc(pas);
38     end;
39   if sens=1 then
40     begin
41       inc(i);
42 { a[i,j]:=pas; } scrie;
43       inc(pas);
44     end
45   else
46     begin
47       inc(j);
48 { a[i,j]:=pas; } scrie;
49       inc(pas);
50     end;

```

```

51     sens := -sens;
52   end;
53   for k := 1 to n-2 do {Diag Secundara}
54     begin i:=i+sens;
55       j:=j-sens;
56     {   a[i,j]:=pas;} scrie;
57     inc(pas);
58   end; sens := -sens;
59   if sens=1 then
60     begin
61       inc(i);
62     {   a[i,j]:=pas;} scrie;
63     inc(pas);
64   end
65   else
66     begin
67       inc(j);
68     {   a[i,j]:=pas;} scrie;
69     inc(pas);
70   end;
71   for d := n-3 downto 1 do {de sub DS pana la coltul dreapa jos}
72 begin
73   for k := 1 to d do
74     begin
75       i:=i+sens;
76       j:=j-sens;
77     {   a[i,j]:=pas;} scrie;
78     inc(pas);
79   end;
80   if sens=1 then
81     begin
82       inc(j);
83     {   a[i,j]:=pas;} scrie;
84     inc(pas);
85   end
86   else
87     begin
88       inc(i);
89     {   a[i,j]:=pas;} scrie;
90     inc(pas);
91   end;
92   sens := -sens;
93 end;
94 i:=n; j:=n; {a[n,n] :=pas;} scrie; inc(pas);
95 for i := n-1 downto 1 do
96   begin
97     {   a[i,j]:=pas;} scrie;
98     inc(pas);
99   end;
100  for j := n-1 downto 2 do
101    begin
102    {   a[i,j]:=pas;} scrie;
103    inc(pas);
104  end;
105  writeln(fo,'1 1');
106  { for i := 1 to n do
107    begin
108      for j := 1 to n do write(a[i,j]:8);
109      writeln
110    end;} close(fo);
111 end.

```

38.5 Obiective turistice

Cristian Cadar, student, Universitatea Politehnica, Bucureşti

În judeţul Constanţa se găsesc cel mult 2500 de obiective turistice dispuse pe o suprafaţă asemănătoare cu un caroiaj, format din pătrătele având latura de 1 kilometru. Obiectivele sunt amplasate în unele puncte de intersecţie ale caroiajului.

Două obiective se consideră vecine dacă sunt situate pe una din direcţiile Nord, Sud, Est, Vest la o distanţă de 1 kilometru una faţă de alta.

Între anumite perechi de obiective vecine au fost construite drumuri directe, pentru a se asigura deplasarea între oricare două obiective.

Despre fiecare obiectiv avem următoarele informații:

numărul asociat acestuia, care este unic și are o valoare cuprinsă între 1 și numărul total al obiectivelor;

obiectivele vecine cu care acesta este conectat prin drumuri directe (cel mult patru obiective).

Pentru fiecare două obiective care comunică se cunoaște costul asfaltării drumului direct dintre ele.

Cerințe:

1. Se dorește realizarea unei hărți a județului în care să apară reprezentate toate obiectivele. Harta se va reprezenta sub forma unei matrice având dimensiunea maximă 50×50 .

Fiecare element al matricei codifică printr-o succesiune de patru cifre binare existența drumurilor directe care încadrează un pătrățel din caroiaj. Dacă pe latura din nord a acestui pătrățel există un drum direct între două obiective, atunci bitul cel mai semnificativ va fi 1. În caz contrar acesta va fi 0. Ceilalți trei biți corespund, în ordine, direcțiilor Sud, Est, Vest.

De exemplu, dacă pentru un pătrățel există drumuri pe laturile Nord, Sud și Vest, cei patru biți vor fi: 1 1 0 1

Harta trebuie să aibă suprafață minimă dar să cuprindă toate obiectivele.

2. Se dorește asfaltarea unora dintre drumurile existente astfel încât să se poată ajunge, mergând numai pe drumuri asfaltate de la oricare obiectiv la oricare altul. Costul total al asfaltării trebuie să fie minim. Să se precizeze care dintre drumuri vor fi asfaltate.

Date de intrare:

Fișierul de intrare OB.IN are un număr necunoscut de linii, fiecare fiind de forma:

NO nvN cdsN nvS cdsS nvE cdsE nvV cdsV

având semnificația:

NO = număr obiectiv

nvN = număr obiectiv vecin nord

cdsN = cost drum spre nord

nvS = număr obiectiv vecin sud

cdsS = cost drum spre sud

nvE = număr obiectiv vecin est

cdsE = cost drum spre est

nvV = număr obiectiv vecin vest

cdsV = cost drum spre vest

Dacă un anumit obiectiv nu are vecin într-o anumită direcție, numărul vecinului din direcția respectivă va fi înlocuit cu 0, iar costul drumului direct dintre cele două cu 0.

Date de ieșire:

Datele de ieșire se scriu în fișierul OB1.OUT și OB2.OUT. Fișierul OB1.OUT conține o hartă a obiectivelor folosind numere între 0 și 15. Fiecare număr reprezintă codificarea în baza zece a celor patru cifre binare. Fișierul OB2.OUT conține pe prima linie costul optim cerut și pe următoarele linii obiectivele între care se asfaltează drumul.

Restricții:

numărul de obiective este cel mult 2500

harta se încadrează într-o matrice de 50 de linii și 50 de coloane

lungimile drumurilor directe sunt cel mult 100

Exemplu:

OB.IN	OB1.OUT	OB2.OUT
1 0 0 0 0 2 3 0 0	1 2	11
3 6 3 0 0 0 0 2 1	14 15	1 2
6 0 0 3 3 0 0 5 2		2 3
2 5 2 0 0 3 1 1 3		6 5
4 0 0 0 0 5 3 0 0		2 5
5 0 0 2 2 6 2 4 3		5 4

Fișierului OB1.OUT îi corespunde următoarea hartă:

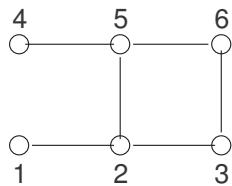


Figura 38.4: Obiective

Notă:

Drumurile directe între obiective sunt bidirectionale.

Se garantează că datele de intrare nu conțin date incompatibile.

Se garantează că nici una din dimensiunile hărții nu va depăși 50.

Punctajele pentru cele două cerințe sunt independente.

Timp maxim de execuție: 2 secunde/per test.

38.5.1 *Indicații de rezolvare

38.5.2 Rezolvare detaliată

Listing 38.5.1: ob.java

```

1 import java.io.*;
2 class ObiectiveTuristice
3 {
4     static int n,m,cost;           // noduri, muchii, cost asfaltare
5     static int xmin,xmax,ymin,ymax;
6     static boolean[] v=new boolean[2501]; // vizitat
7
8     static int[] on=new int[2501];
9     static int[] os=new int[2501];
10    static int[] oe=new int[2501];
11    static int[] ov=new int[2501];
12    static int[] et=new int[2501];
13
14    static int[] m1=new int[4901];
15    static int[] m2=new int[4901];
16    static int[] cm=new int[4901];
17
18    static int[][] h=new int[100][100];
19
20    static PrintWriter out1;
21    static PrintWriter out2;
22    public static void main(String[] args) throws IOException
23    {
24        long t1,t2;
25        t1=System.currentTimeMillis();
26
27        int i,j,x,y,cod;
28
29        StreamTokenizer st= new StreamTokenizer(
30            new BufferedReader(new FileReader("ob.in")));
31        out1= new PrintWriter(new BufferedWriter(new FileWriter("obl.out")));
32        out2= new PrintWriter(new BufferedWriter(new FileWriter("ob2.out")));
33
34        n=0;
35        m=0;
36        while(st.nextToken()!=StreamTokenizer.TT_EOF)
37        {
38            n++;
39            i=(int)st.nval;
40            st.nextToken(); on[i]=(int)st.nval;
41            st.nextToken(); j=(int)st.nval;
42            if(on[i]!=0)          // muchie spre nord: sud=i-->on[i]=nord
43            {
  
```

```

44         m++; m1[m]=i; m2[m]=on[i]; cm[m]=j;
45     }
46
47     st.nextToken(); os[i]=(int)st.nval;
48     st.nextToken(); j=(int)st.nval;
49
50     st.nextToken(); oe[i]=(int)st.nval;
51     st.nextToken(); j=(int)st.nval;
52     if(oe[i]!=0)      // muchie spre est: vest=i<-->oe[i]=est
53     {
54         m++; m1[m]=i; m2[m]=oe[i]; cm[m]=j;
55     }
56
57     st.nextToken(); ov[i]=(int)st.nval;
58     st.nextToken(); j=(int)st.nval;
59 } // while
60
61 asfaltare();
62 out2.println(cost);
63 for(i=1;i<=m;i++) if(cm[i]!=0) out2.println(m1[i]+" "+m2[i]);
64 out2.close();
65
66 xmin=xmax=ymin=ymax=50;
67 f(1,50,50);
68 out1.println((ymax-ymin)+" "+(xmax-xmin));
69
70 for(y=ymax;y>ymin;y--)
71 {
72     for(x=xmin+1;x<=xmax;x++)
73     {
74         cod=0;
75         if(h[x][y]!=0) // daca exista obiectiv dreapta_sus ... patratel
76         {
77             if(ov[h[x][y]]!=0) cod+=8; // latura sus... patratel
78             if(os[h[x][y]]!=0) cod+=2; // latura dreapta... patratel
79         }
80         if(h[x-1][y-1]!=0) // daca exista obiectiv stanga_jos ... patratel
81         {
82             if(on[h[x-1][y-1]]!=0) cod+=1; // latura stanga... patratel
83             if(oe[h[x-1][y-1]]!=0) cod+=4; // latura jos... patratel
84         }
85
86         out1.print(cod+" ");
87     } // for x
88     out1.println();
89 } // for y
90
91 out1.close();
92
93 t2=System.currentTimeMillis();
94 System.out.println("Time: "+(t2-t1));
95 } //main
96
97 static void f(int k, int x, int y) // obiectiv, coordonatele lui
98 {
99     h[x][y]=k;
100    v[k]=true;
101    if(on[k]!=0) // exista obiectiv la nord
102        if(!v[on[k]]) // care nu este inca vizitat
103            f(on[k],x,y+1);
104    if(os[k]!=0) // exista obiectiv la sud
105        if(!v[os[k]]) // care nu este inca vizitat
106            f(os[k],x,y-1);
107    if(oe[k]!=0) // exista obiectiv la est
108        if(!v[oe[k]]) // care nu este inca vizitat
109            f(oe[k],x+1,y);
110    if(ov[k]!=0) // exista obiectiv la vest
111        if(!v[ov[k]]) // care nu este inca vizitat
112            f(ov[k],x-1,y);
113
114    if(x<xmin) xmin=x;
115    if(x>xmax) xmax=x;
116    if(y<ymin) ymin=y;
117    if(y>ymax) ymax=y;
118 } //f(...)
119

```

```

120     static void asfaltare()
121     {
122         int i,nm,k,etg1,etg2;
123         for(k=1;k<=n;k++) et[k]=k;
124
125         qsort(1,m);
126
127         cost=0;
128         nm=0; // nr muchii selectate
129         for(k=1;k<=m;k++) // parcurg toate muchiile
130         {
131             if(nm==n-1) { cm[k]=0; continue; }
132             if(et[m1[k]]!=et[m2[k]]) {
133                 nm++;
134                 cost+=cm[k];
135                 etg1=et[m1[k]]; etg2=et[m2[k]];
136                 for(i=1;i<=n;i++) if(et[i]==etg2) et[i]=etg1;
137             }
138             else cm[k]=0; // pun pe 0 costul muchiei nefolosite
139         }// for k
140     }//asfaltare()
141
142     static void qsort(int li, int ls)
143     {
144         int i,j,em,aux;
145         em=cm[(li+ls)/2];
146         i=li;
147         j=ls;
148         while(i<j)
149         {
150             while(cm[i]<em) i++;
151             while(cm[j]>em) j--;
152             if(i<=j)
153             {
154                 aux=cm[i]; cm[i]=cm[j]; cm[j]=aux;
155                 aux=m1[i]; m1[i]=m1[j]; m1[j]=aux;
156                 aux=m2[i]; m2[i]=m2[j]; m2[j]=aux;
157                 i++;
158                 j--;
159             } // if
160         } // while
161         if(li<j) qsort(li,j);
162         if(i<ls) qsort(i,ls);
163     }// qsort(...)
164 } //class

```

38.5.3 Cod sursă

Listing 38.5.2: ob.pas

```

1 const
2     NMAX=5000;
3 type
4     vector=array[0..NMAX] of boolean;
5     iverator=array[0..NMAX] of integer;
6     matrice=array[0..100,0..100] of integer;
7     obiectiv=record
8         o1,o2,o3,o4,c1,c2,c3,c4:integer;
9         end;
10    muchie=array[1..3] of integer;
11    muchii=array[1..10000] of muchie;
12 var
13     a,c:matrice;
14     n,nr,select,mm,x,y,xmin,xmax,ymin,ymax:integer;
15     f:text;
16     o:array[1..NMAX] of ^obiectiv;
17     v:^vector;
18     m:^muchii;
19     indici,disj:^iverator;
20
21 procedure citire;

```

```

22 var
23   i:integer;
24 begin
25   assign(f,'ob.in');
26   reset(f);
27   for i:=1 to NMAX do
28     new(o[i]);
29   n:=0;mm:=0;
30   new(m);
31   while not seekeof(f) do
32     begin
33       inc(n);
34       read(f,nr);
35       readln(f,o[nr]^ .o1,o[nr]^ .c1,o[nr]^ .o2,o[nr]^ .c2,
36               o[nr]^ .o3,o[nr]^ .c3,o[nr]^ .o4,o[nr]^ .c4);
37       if o[nr]^ .o1<>0
38         then
39           begin
40             inc(mm);
41             m^ [mm,1]:=nr;
42             m^ [mm,2]:=o[nr]^ .o1;
43             m^ [mm,3]:=o[nr]^ .c1;
44           end;
45       if o[nr]^ .o3<>0
46         then
47           begin
48             inc(mm);
49             m^ [mm,1]:=nr;
50             m^ [mm,2]:=o[nr]^ .o3;
51             m^ [mm,3]:=o[nr]^ .c3;
52           end;
53     end;
54   close(f);
55 end;
56
57 procedure rec(k:integer);
58 begin
59   if k=0
60     then exit;
61   v^ [k]:=true;
62   if not v^ [o[k]^ .o1] { nord }
63     then
64       begin
65         x:=x-1;
66         rec(o[k]^ .o1);
67         x:=x+1;
68       end;
69   if not v^ [o[k]^ .o2] { sud }
70     then
71       begin
72         x:=x+1;
73         rec(o[k]^ .o2);
74         x:=x-1;
75       end;
76   if not v^ [o[k]^ .o3] { est }
77     then
78       begin
79         y:=y+1;
80         rec(o[k]^ .o3);
81         y:=y-1;
82       end;
83   if not v^ [o[k]^ .o4] { vest }
84     then
85       begin
86         y:=y-1;
87         rec(o[k]^ .o4);
88         y:=y+1;
89       end;
90   if x<xmin
91     then xmin:=x;
92   if x>xmax
93     then xmax:=x;
94   if y<ymin
95     then ymin:=y;
96   if y>ymax
97     then ymax:=y;

```

```

98      a[x,y]:=k;
99  end;
100
101 procedure scrie(i,j:integer);
102 var
103   rez:integer;
104 begin
105   rez:=0;
106   if (a[i-1,j-1]<>0) and (o[a[i-1,j-1]]^.o3<>0)
107     then inc(rez,8); { nord }
108   if (a[i,j]<>0) and (o[a[i,j]]^.o4<>0)
109     then inc(rez,4); { sud }
110   if (a[i,j]<>0) and (o[a[i,j]]^.o1<>0)
111     then inc(rez,2); { est }
112   if (a[i-1,j-1]<>0) and (o[a[i-1,j-1]]^.o2<>0)
113     then inc(rez,1); { vest }
114   write(f,rez,' ');
115 end;
116
117 procedure prelucrare1;
118 var
119   i,j:integer;
120 begin
121   { Constructie matrice }
122   new(v);
123   fillchar(v^,sizeof(v^),false);
124   x:=50; y:=50;
125   xmin:=50; ymin:=50; xmax:=50; ymax:=50;
126   rec(1);
127   {--}
128   assign(f,'obj.out');
129   rewrite(f);
130   writeln(f,xmax-xmin,' ',ymax-ymin);
131   for i:=xmin+1 to xmax do
132     begin
133       for j:=ymin+1 to ymax do
134         scrie(i,j);
135       writeln(f);
136     end;
137   close(f);
138 end;
139
140 {procedure qsort2(li,ls:integer);
141 var i,j:integer;
142 begin
143   i:=li;
144   j:=ls;
145   med:=c^(li+ls) shr 1;
146   repeat
147     while c^[i]<med do i:=i+1;
148     while c^[j]>med do j:=j-1;
149     if i<=j then begin
150       aux:=a^[i];
151       a^[i]:=a^[j];
152       a^[j]:=aux;
153       aux:=b^[i];
154       b^[i]:=b^[j];
155       b^[j]:=aux;
156       aux:=t^[i];
157       t^[i]:=t^[j];
158       t^[j]:=aux;
159       i:=i+1;
160       j:=j-1;
161       end;
162     until i>j;
163     if li<j then qsort(c,li,j);
164     if i<ls then qsort(c,i,ls);
165   end; }
166
167 procedure qsort(li,ls:integer);
168 var
169   i,j,elem:integer;
170   aux:muchie;
171 begin
172   elem:=m^[(li+ls) div 2][3];
173   i:=li;

```

```

174      j:=ls;
175      while i<j do
176          begin
177              while m^[i][3]<elem do
178                  inc(i);
179              while m^[j][3]>elem do
180                  dec(j);
181              if i<=j
182                  then
183                      begin
184                          aux:=m^[i];
185                          m^[i]:=m^[j];
186                          m^[j]:=aux;
187                          inc(i);
188                          dec(j);
189                      end;
190                  end;
191              if li<j
192                  then qsort(li,j);
193              if i<ls
194                  then qsort(i,ls);
195      end;
196
197 procedure init_disj;
198 var
199     i:integer;
200 begin
201     new(disj);
202     for i:=1 to n do
203         disj^[i]:=-1;
204 end;
205
206 function mult(k:integer):integer;
207 begin
208     while disj^[k]>0 do
209         k:=disj^[k];
210     mult:=k;
211 end;
212
213 procedure reun(a,b:integer);
214 begin
215     while disj^[a]>0 do
216         a:=disj^[a];
217
218     while disj^[b]>0 do
219         b:=disj^[b];
220
221     if abs(disj^[a])<abs(disj^[b])
222         then
223             begin
224                 disj^[b]:=disj^[a]+disj^[b];
225                 disj^[a]:=b;
226             end
227         else
228             begin
229                 disj^[a]:=disj^[a]+disj^[b];
230                 disj^[b]:=a;
231             end
232     end;
233
234 procedure verif_sort;
235 var
236     i:integer;
237 begin
238     for i:=1 to mm-1 do
239         if m^[i][3]>m^[i+1][3]
240             then
241                 begin
242                     writeln('GRESIT !!!');
243                     halt;
244                 end;
245     end;
246
247 procedure prelucrare2;
248 var
249     k:integer;

```

```

250      cost:longint;
251  begin
252    new(indici);
253    qsort(1,mm);
254    verif_sort;
255    init_disj;
256    select:=0;
257    k:=0;
258    assign(f,'ob2.out');
259    rewrite(f);
260  {    writeln(f,mm-1);}
261  cost:=0;
262  while select<n-1 do
263  begin
264    inc(k);
265    if mult(m^ [k] [1])<>mult(m^ [k] [2])
266    then
267    begin
268      inc(select);
269      writeln(f,m^ [k] [1],',',m^ [k] [2]);
270      inc(cost,m^ [k] [3]);
271      reun(m^ [k] [1],m^ [k] [2]);
272      indici^ [select]:=k;
273    end;
274    end;
275    writeln(f,cost);
276    for k:=1 to n-1 do
277      writeln(f,m^ [indici^ [k]] [1],',',m^ [indici^ [k]] [2]);
278    close(f);
279  end;
280
281 begin
282   citire;
283   prelucrare1;
284   prelucrare2;
285 end.

```

38.6 Taxi

Cristian Cadar, student, Universitatea Politehnica, Bucureşti

Un cător dorește să străbată o distanță de n kilometri cu taxiul, astfel încât prețul pe care îl plătește să fie minim. Compania de taximetre pe care această persoană o folosește are un tarif unitar fix de C lei pentru un kilometru, însă are de asemenea niște oferte speciale de tipul (k, L) cu semnificația că pentru o porțiune de drum de k kilometri călătorul poate să plătească L lei (în locul unui tarif unitar pe numărul de kilometri). Din păcate, aceste oferte sunt foarte ciudate, și dacă nu este atent, călătorul nostru se poate păcăli ușor. De exemplu compania de taximetre poate să ceară pentru 2,43 km un tarif de 1370,89 lei și pentru 3,8 km un tarif de 1509,43 lei. Cum călătorul nostru are dificultăți în manevrarea unor astfel de numere, el vă roagă să-l ajutați să afle modul optim de organizare a excursiei sale, cunoșcând atât numărul de km pe care acesta trebuie să-i parcurgă cât și tarifele practicate de compania de taximetre.

Date de intrare:

Pe prima linie a fișierului TAXI.IN se găsește numărul n ($n \leq 100$) de kilometri pe care călătorul trebuie să îi parcurgă. Pe cea de-a doua linie se află tariful unitar L practicat de compania de taximetre. Următoarele linii (cel mult 100), până la sfârșitul fișierului conțin perechi de forma kC reprezentând ofertele speciale ale companiei. Toate numerele care apar în fișierul de intrare sunt numere reale, strict pozitive, mai mici decât 1000 și cu fix 2 zecimale.

Date de ieșire:

Prima linie a fișierului TAXI.OUT va conține suma minimă pe care o poate achita călătorul, scrisă cu trei zecimale exacte. Următoarea linie va conține o succesiune de numere, reprezentând ordinea în care se face alegerea ofertelor speciale. Aceste numere pot fi atât naturale cât și numere reale negative. Numerele naturale indică câtă ofertă din fișierul de intrare a fost aleasă la un moment dat, iar numerele reale negative indică numărul de km parcurși (în modul), folosind prețul unitar.

Restricții:

$0 < n \leq 100$

$0 < L \leq 1000$

$0 < k \leq 1000$

$0 < C \leq 1000$

Numerele n, L, k, C sunt reale și au maximum câte două zecimale.

Exemplu:

TAXI.IN	TAXI.OUT poate conține:
52.65 // nr. km	28.380
0.80 // tarif unitar	-0.69 1 3 3 -2.41
8.75 5.50 // 8.75 km cu prețul de 5.5	
60.35 20.47 // 60.35 km cu prețul de 20.47	
20.40 10.20 // 20.4 km cu prețul de 10.2	

Timp maxim de execuție pentru un test: 2 secunde.

38.6.1 *Indicații de rezolvare

38.6.2 Rezolvare detaliată

Listing 38.6.1: taxi.java

```

1 import java.io.*; // "fix 2 zecimale" ==> conversie la intregi !
2 class Taxi
3 {
4     static int n; // nr zeci de metri de parcurs
5     static double t; // tarif unitar pentru 10m
6     static int m; // nr oferte
7     static int[] d=new int[102]; // oferta: dist in zeci de metri
8     static double[] c=new double[102]; // oferta: cost pentru 10m
9     static double[] smin=new double[10001]; // suma minima
10    static byte[] o=new byte[10001]; // ou[i]=oferta utilizata la km i
11
12    public static void main(String []args) throws IOException
13    {
14        int i,x;
15        byte j;
16
17        StreamTokenizer st=new StreamTokenizer(
18            new BufferedReader(new FileReader("taxi.in")));
19        PrintWriter out = new PrintWriter (
20            new BufferedWriter( new FileWriter("taxi.out")));
21
22        st.nextToken(); n=(int)(st.nval*100+0.001);
23        st.nextToken(); t=st.nval/100+0.0000000001;
24
25        m=0;
26        while(st.nextToken()!=StreamTokenizer.TT_EOF)
27        {
28            m++;
29            d[m]=(int)(st.nval*100+0.001); // distanta
30            st.nextToken(); c[m]=st.nval; // cost "segment intreg"
31        }
32
33        m++; // oferta normala o consider ca "ultima oferta speciala"
34        d[m]=1;
35        c[m]=t;
36
37        for(i=1;i<=n;i++) // zeci de metri parcursi
38            for(j=1;j<=m; j++) // oferte ...
39            {
40                if( (i>=d[j])&&
41                    ((smin[i-d[j]]>-0.001) || (i==d[j]))&&
42                    ((smin[i]<0.0010) || (smin[i]>=smin[i-d[j]]+c[j]-0.001)) )
43                {
44                    smin[i]=smin[i-d[j]]+c[j];
45                    o[i]=j;
46                }
47            }
48

```

```

49     out.println(((int)(smin[n]*1000+0.5))/1000.0);
50
51     while (n>0)
52     {
53         x=0;
54         while ((n>0) && (o[n]==m)) { x++; n--; }
55         if (x>0) out.print("-"+((int)((x/100.0)*1000+0.5))/1000.0+" ");
56         else { out.print(o[n]+" "); n=n-d[o[n]]; }
57     }
58     out.close();
59 } //main
60 } //class

```

38.6.3 Cod sursă

Listing 38.6.2: taxi.pas

```

1 {$b-,r-,q-,s-}
2 {$M 65000,0,655360}
3 type
4     vector=array[0..10000] of byte;
5 var
6     a:array[0..10000] of real;
7     t:^vector;
8     pret:array[1..101] of real;
9     km:array[1..101] of longint;
10    x,i,j,k,n,nr:longint;
11    tarif,aux:real;
12    f:text;
13
14 procedure citire;
15 begin
16     assign(f,'taxi.in');
17     reset(f);
18     readln(f,aux); n:=round(aux*100);
19     readln(f,tarif);
20     tarif:=tarif/100;
21     nr:=0;
22     while not seeeof(f) do
23     begin
24         inc(nr);
25         read(f,aux); km[nr]:=round(aux*100);
26         readln(f,pret[nr]);
27     end;
28     inc(nr); km[nr]:=1; pret[nr]:=tarif;
29     close(f);
30 end;
31
32 procedure dinamica;
33 begin
34     fillchar(a,sizeof(a),0);
35     new(t);
36     for i:=1 to n do
37         for j:=1 to nr do
38             if (i>=km[j]) and ( (a[i-km[j]]<>0) or
39             (i-km[j]=0) ) and ( (a[i]=0) or (a[i]>=a[i-km[j]]+pret[j]) )
40             then
41             begin
42                 a[i]:=a[i-km[j]]+pret[j];
43                 t^[i]:=j;
44             end;
45 end;
46
47 procedure afisare;
48 begin
49     assign(f,'taxi.out');
50     rewrite(f);
51     writeln(f,a[n]:0:3);
52     while n>0 do
53     begin
54         x:=0;
55         while (n>0) and (t^[n]=nr) do

```

```
56          begin
57              inc(x);
58              n:=n-km[t^n];
59          end;
60      if x<>0
61          then write(f,'-',x/100:0:2,' ')
62      else
63          begin
64              write(f,t^n,' ');
65              n:=n-km[t^n];
66          end;
67      end;
68  close(f);
69 end;
70
71 begin
72     citire;
73     dinamica;
74     afisare;
75 end.
```

Glosar

```
#define
    mp make_pair, 628
    pb push_back, 628
    INF 0x3f3f3f3f, 579
^=, 870
__builtin_parity, 42
şirul Fibonacci, 572
şirul lui Fibonacci, 133
şirul sumelor parțiale, 406
şmenul lui Mars, 422

a<b?a:b, 700
algoritm backtracking
    optimizat, 745
algoritm de baleiere, 439, 615
algoritm de fill, 587, 634
algoritm de tip Fill, 638, 659
algoritm de tip succesor, 744
algoritm de umplere, 122, 145
algoritm de umplere (fill), 30
algoritm FILL, 195
algoritm greedy, 519
algoritm Lee, 80, 107, 195
algoritmul FILL, 205
algoritmul fill, 145
algoritmul Lee, 51, 168, 601, 905
algoritmul lui LEE, 576
algoritmul lui Lee, 885, 905
algoritmul nth_element, 180
algoritmului lui Lee, 923
aranjamente, 282
assert, 387, 398, 400, 407
atoi, 201, 202, 626, 628
atol, 626, 628
atoll, 626
auto, 400
auto &, 298, 319, 336, 391, 414

back(), 336, 391
backtracking, 493, 503, 619, 691, 703, 712, 768
backtracking pe biti, 555
baleiere, 871
begin(), 82, 86, 90, 319, 336, 368, 383, 387, 391,
    398, 400, 407, 685
bits/stdc++.h, 315
bitset, 616
bool, 82, 86, 90, 316, 338, 577, 628
bordare matrice, 659
brute force, 554
c_str(), 628

căutare binară, 51, 140, 381, 388, 406, 464, 510,
    540, 653
caracterul sfârșit de linie, 566
cerr, 421
char, 84, 90, 337
char *, 626
cin.sync_with_stdio(false);, 465
ciurul lui Erastotene, 567
ciurul lui Eratostene, 394, 400, 462, 496, 554,
    749
Ciurul lui Eratosthenes, 318
clear(), 336, 400
clock(), 657, 683
CLOCKS_PER_SEC, 657, 683
cmmdc, 168, 317
coadă, 30, 195, 293, 422, 587, 659, 726, 754, 768,
    923
coadă (queue), 634
codul Gray, 744
combinatorică, 880
const, 82, 86, 398, 577, 628
const char, 670
const int, 670
    INF = 0x3f3f3f3f;, 670
Cormen, 868
count, 368
CountSort, 538

delete, 898
deque, 82, 86, 90
distanța Manhattan, 738
divide & impera, 709
Divide et Impera, 671
divide et impera, 151, 388, 562, 730

emplace, 296
empty(), 82, 86, 113, 296, 298, 336, 338, 368,
    383, 391, 577
end(), 319, 336, 368, 383, 387, 391, 398, 400,
    407, 628, 685
erase, 336, 407
exponențiere rapidă, 388
expresie regulată, 519

fgets, 683
Fill, 662
fill, 364, 636
find, 628
find_first_of, 628
first, 82, 86, 296, 298, 368, 383, 391, 398, 628,
    636, 685
forma canonica, 33
```

- forma poloneză, 644
 formulele lui Euclid, 381
 front(), 82, 86, 113, 296, 298, 383
 funcție bijectivă, 745
 funcție recursivă, 712
 funcții recursive, 863
- get, 364, 686
 get(), 82, 84, 364, 628, 686
 getline, 84, 184, 201, 202, 628, 688
 greater, 90
 Greedy, 94, 905
 greedy, 720, 741
- heap, 587
- inline, 82, 86, 90, 577
 int64_t, 319, 387, 407
 invers modular, 494, 536, 567
 invmod, 501
 istringstream, 685
- lambda, 82, 86, 90, 391
 Lee, 82, 84, 112, 113, 603
 Legendre, 496
 liste înlănțuite, 16
 liste circulare, 16
 liste dublu înlănțuite, 16
 lower_bound, 383, 391, 407
- make_pair, 84, 296, 298, 383, 398, 628, 636
 make_triple, 383
 map, 316, 567, 628
 max(), 298, 387
 max_element, 319
 mediana, 184, 612
 mediana unui vector, 179
 memcpy, 574, 683, 695
 memorizare, 652, 697
 memset, 590, 605, 670, 683, 695
 sizeof, 670
 MergeSort, 179
 metoda backtracking, 744
 metoda backtracking în plan, 912
 metoda programării dinamice, 782
 metoda programării dinamice, 188, 738, 848, 891, 899, 946
 minim local, 95
 mypow, 501
- new, 626, 898
 next_permutation, 337
 normalizare coordonate, 381
 normalizarea sirului, 406
 NULL, 626, 686, 688
 numărul de partiții, 899
 numărul lui Catalan, 619
 numere mari, 158, 188, 606, 691, 811, 899
 numeric_limits, 298
 numeric_limits<int64_t>::max(), 387
- operator, 90, 628
- ordine lexicografică, 744
- pair, 82, 84, 90, 296, 298, 368, 383, 391, 398, 400, 628, 636, 685
 parcurgere în lățime, 293
 parsare, 180
 partiție, 139
 partițiile numărului, 410
 partition, 391
 Pascal, 743
 permutare
 ciclu de lungime 2, 691
 punct fix, 691
- pop(), 82, 296, 298, 383, 577
 pop_back(), 336, 338, 368, 391
 pow, 316
 preprocesare, 697
 principiul cutiei, 325, 366
 principiul includerii și excluderii, 318
 priority_queue, 90
 problema rucsacului, 503
 programării dinamice, 950
 programare dinamică, 653, 712, 777, 785, 820, 866
 progresia geometrică, 133
 progresie geometrică, 747
 proprietatea anti-triunghi, 133
 pseudocod, 773
 punct laticeal, 414
 puncte laticeale, 381
 push, 82, 113, 296, 383, 577
 push_back, 82, 86, 90, 336, 338, 383, 387, 391, 398, 400, 628, 685
 push_front, 82, 86, 90
- qsort, 202
 queue, 82, 86, 113, 296, 298, 383, 577, 603, 636
 empty(), 603, 636
 front(), 603, 636
 push, 603, 636
 QuickSort, 179, 538
 quicksort, 179
- rand(), 462, 463
 rbegin(), 338
 recursie cu memorizare, 652
 recursivitate, 493
 recursivitate indirectă, 180, 863
 regex, 519
 regula sumă-produs, 410
 relație de recurență, 519
 reverse, 336, 391, 400, 685
- second, 82, 86, 296, 298, 368, 383, 391, 398, 628, 636, 685
 secvență, 139
 secvență de sumă maximă, 179
 segment de stivă, 206
 segmentul de stivă, 659
 set, 567
 short, 82, 84, 577

short int, 636
size(), 82, 86, 90, 296, 298, 338, 368, 383, 387,
391, 398, 400, 628, 685
sizeof, 202, 670
sort, 82, 86, 90, 337, 364, 383, 387, 391, 398,
407, 616, 685, 686
 cmp, 686
sortare, 388, 439, 448, 454, 748
rand(time(NULL)), 463
stack, 99, 102
stderr, 683
Stirling de speță a II-a, 410
stivă, 333, 388, 606, 652, 659, 708, 744
stiva, 94, 179
STL, 179, 567
stp ^ 1, 701
strategie Lee, 792
strcat, 201, 202, 364
strchr, 201, 202, 626, 688
strcmp, 201, 202, 683, 686, 688
strcpy, 201, 202, 626, 686, 688
strdup, 626
string, 86, 90, 296, 298, 315, 336, 338, 387, 462,
628, 685
 begin(), 315
 end(), 315
 push_back, 315
 reverse, 315
 size(), 315, 336
string::npos, 628
strlen, 84, 184, 201, 337, 626, 683
strncpy, 626, 688
strrev, 686, 688
strtok, 201, 202, 626, 686
struct, 90, 113, 201, 202, 338, 391, 577, 616, 628,
686
structură, 199
subșir, 139
substr, 628
sume partiale, 381, 384
swap, 184
swap rapid, 311

tablou Young, 17
tehnica programării dinamice, 993
template, 391
teorema Legendre, 494
teoreme de combinatorică, 899
tie, 298
time_t, 657
triunghiul lui Pascal, 188, 536, 567, 619
two pointers technique, 16
typedef, 636

unique, 383, 407
unordered_map, 409
unsigned long long, 683
using, 86

vector, 82, 86, 90, 296, 298, 316, 319, 387, 391,
398, 400, 407, 414, 628, 685
 new vector, 414
 resize, 414
vector caracteristic, 709
vector de frecvență, 317, 333
vector de frecvențe, 454
vector<vector<int>, 338

W1.8s vs L0.2s, 612, 748

Bibliografie

- [1] Aho, A., Hopcroft, J., Ullman, J.D.; Data strutures and algorithms, Addison Wesley, 1983
- [2] Andreica M.I.; Elemente de algoritmică - probleme și soluții, Cibernetica MC, 2011
- [3] Andonie R., Gârbacea I.; Algoritmi fundamentali, o perspectivă C++, Ed. Libris, 1995
- [4] Atanasiu, A.; Concursuri de informatică. Editura Petrion, 1995
- [5] Bell D., Perr M.; Java for Students, Second Edition, Prentice Hall, 1999
- [6] Calude C.; Teoria algoritmilor, Ed. Universității București, 1987
- [7] Cerchez, E.; Informatică - Culegere de probleme pentru liceu, Ed. Polirom, Iași, 2002
- [8] Cerchez, E., Șerban, M.; Informatică - manual pentru clasa a X-a., Ed. Polirom, Iași, 2000
- [9] Cori, R.; Lévy, J.J.; Algorithmes et Programmation, Polycopié, version 1.6; <http://w3.edu.polytechnique.fr/informatique/>
- [10] Cormen, T.H., Leiserson C.E., Rivest, R.L.; Introducere în Algoritmi, Ed Agora, 2000
- [11] Cormen, T.H., Leiserson C.E., Rivest, R.L.; Pseudo-Code Language, 1994
- [12] Cristea, V.; Giumale, C.; Kalisz, E.; Paunoiu, Al.; Limbajul C standard, Ed. Teora, București, 1992
- [13] Erickson J.; Combinatorial Algorithms; <http://www.uiuc.edu/~jeffe/>
- [14] Flanagan, D.; Java in a Nutshell, O'Reilly, 1997.
- [15] Giumale C., Negreanu L., Călinoiu S.; Proiectarea și analiza algoritmilor. Algoritmi de sortare, Ed. All, 1997
- [16] Halim S., Halim F., Competitive programming, 2013
- [17] Knuth, D.E.; Arta programării calculatoarelor, vol. 1: Algoritmi fundamentali, Ed. Teora, 1999.
- [18] Knuth, D.E.; Arta programării calculatoarelor, vol. 2: Algoritmi seminumerici, Ed. Teora, 2000.
- [19] Knuth, D.E.; Arta programării calculatoarelor, vol. 3: Sortare și căutare, Ed. Teora, 2001.
- [20] Knuth, D.E.; The art of computer programming, vol. 4A: Combinatorial algorithms, Part 1, Addison Wesley, 2011.
- [21] Lambert, K. A., Osborne, M.; Java. A Framework for Programming and Problem Solving, PWS Publishing, 1999
- [22] Laaksonen A.; Guide to competitive programming, Springer, 2017
- [23] Livovschi, L.; Georgescu H.; Analiza și sinteza algoritmilor. Ed. Enciclopedică, București, 1986.
- [24] Niemeyer, P., Peck J.; Exploring Java, O'Reilly, 1997.
- [25] Odăgescu, I., Smeureanu, I., Ștefănescu, I.; Programarea avansată a calculatoarelor personale, Ed. Militară, București 1993

- [26] Odăgescu, I.; Metode și tehnici de programare, Ed. Computer Lobris Agora, Cluj, 1998
- [27] Popescu Anastasiu, D.; Puncte de articulație și punți în grafuri, Gazeta de Informatică nr. 5/1993
- [28] Răbâea, A.; https://math.univ-ovidius.ro/Doc/Admitere/CentruPregatire/2007/Info/Lista_probleme_2000-2007.pdf
- [29] Răbâea, A.; https://math.univ-ovidius.ro/Doc/Admitere/CentruPregatire/2007/Info/Rezolvari_C09.pdf
- [30] Răbâea, A.; https://math.univ-ovidius.ro/Doc/Admitere/CentruPregatire/2007/Info/Rezolvari_C10.pdf
- [31] Răbâea, A.; https://math.univ-ovidius.ro/Doc/Admitere/CentruPregatire/2007/Info/Rezolvari_C11.pdf
- [32] Răbâea, A.; https://math.univ-ovidius.ro/Doc/Admitere/CentruPregatire/2007/Info/Rezolvari_Baraj.pdf
- [33] Skiena S.S., Revilla M.A.; Programming challenges - The Programming Contest Training Manual, Springer, 2003
- [34] Tomescu, I.; Probleme de combinatorică și teoria grafurilor, Editura Didactică și Pedagogică, București, 1981
- [35] Tomescu, I.; Leu, A.; Matematică aplicată în tehnica de calcul, Editura Didactică și Pedagogică, București, 1982
- [36] Văduva, C.M.; Programarea in JAVA. Microinformatica, 1999
- [37] Vișinescu, R.; Vișinescu, V.; Programare dinamică - teorie și aplicații; GInfo nr. 15/4 2005
- [38] Vladă, M.; Conceptul de algoritm - abordare modernă, GInfo, 13/2,3 2003
- [39] Vladă, M.; Grafuri neorientate și aplicații. Gazeta de Informatică, 1993
- [40] Weis, M.A.; Data structures and Algorithm Analysis, Ed. The Benjamin/Cummings Publishing Company. Inc., Redwoods City, California, 1995.
- [41] Winston, P.H., Narasimhan, S.; On to JAVA, Addison-Wesley, 1996
- [42] Wirth N.; Algorithms + Data Structures = Programs, Prentice Hall, Inc 1976
- [43] *** - Gazeta de Informatică, Editura Libris, 1991-2005
- [44] *** - https://github.com/DinuCr/CS/blob/master/Info/stuff%20stuff/Rezolvari_C09.pdf
- [45] *** - <https://dokumen.tips/documents/rezolvaric09.html>
- [46] *** - <https://www.scribd.com/doc/266218102/Rezolvari-C09>
- [47] *** - <https://www.scribd.com/document/396362669/Rezolvari-C10>
- [48] *** - <https://www.scribd.com/document/344769195/Rezolvari-C11>
- [49] *** - <https://www.scribd.com/document/364077679/Rezolvari-C11-pdf>