

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 11-a

... draft (ciornă) ...

Dr. Adrian Răbâea

24 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 așa! 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!)³.

Am început câteva cărți (pentru clasele de liceu) cu mai mulți ani în urmă, pentru perioada 2000-2007 ([29] - [33]), 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" ([47] - [54])! 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 ... descrescătoare! :-)

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!

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

²IOI2015 a permis utilizarea limbajelor de programare C++, Java, Pascal, Python și Rubi (...)

³vezi cele 5 secunde pentru **Timp maxim** de executare/test din problema "avârcolaci" - ONI2014 clasa a 11-a

(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! De folos sunt cu siguranță [1] - [28], [34] - [46], ... și multe alte cărți!.

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

Și un ultim gând: criticele sunt utile dacă au valoare! Dacă sunt numai aşa ... cum critică lumea la un meci de fotbal ... sau pe bancă în parc despre rezolvarea problemelor economice ale țării ... atunci ... !!!

Adrese utile:

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

<http://www.cplusplus.com/>
<http://www.infolcup.com/>
<https://www.infoarena.ro/>
<https://www.infoarena.ro/ciorna>
<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 conexidad	2
1.1.1 Indicații de rezolvare	3
1.1.2 Cod sursă	3
1.1.3 *Rezolvare detaliată	9
1.2 rufe	9
1.2.1 Indicații de rezolvare	10
1.2.2 Cod sursă	12
1.2.3 *Rezolvare detaliată	18
1.3 tairos	18
1.3.1 Indicații de rezolvare	20
1.3.2 Cod sursă	21
1.3.3 *Rezolvare detaliată	28
2 OJI 2018	29
2.1 galeti	29
2.1.1 Indicații de rezolvare	30
2.1.2 Cod sursă	31
2.1.3 *Rezolvare detaliată	35
2.2 ramen	35
2.2.1 Indicații de rezolvare	37
2.2.2 Cod sursă	37
2.2.3 *Rezolvare detaliată	43
2.3 aquapark	43
2.3.1 Indicații de rezolvare	44
2.3.2 Cod sursă	45
2.3.3 *Rezolvare detaliată	51
3 OJI 2017	52
3.1 armonica	52
3.1.1 Indicații de rezolvare	53
3.1.2 Cod sursă	53
3.1.3 *Rezolvare detaliată	60
3.2 ninjago	60
3.2.1 Indicații de rezolvare	62
3.2.2 Cod sursă	63
3.2.3 *Rezolvare detaliată	84
3.3 permutare	84
3.3.1 Indicații de rezolvare	86
3.3.2 Cod sursă	86

3.3.3 *Rezolvare detaliată	94
4 OJI 2016	95
4.1 elicoptere	95
4.1.1 Indicații de rezolvare	96
4.1.2 Cod sursă	97
4.1.3 *Rezolvare detaliată	103
4.2 summax	103
4.2.1 Indicații de rezolvare	105
4.2.2 Cod sursă	106
4.2.3 *Rezolvare detaliată	108
5 OJI 2015	109
5.1 2sah	109
5.1.1 Indicații de rezolvare	110
5.1.2 Cod sursă	111
5.1.3 *Rezolvare detaliată	115
5.2 Dragoni	115
5.2.1 Indicații de rezolvare	117
5.2.2 Cod sursă	117
5.2.3 *Rezolvare detaliată	129
6 OJI 2014	130
6.1 cartite	130
6.1.1 Indicații de rezolvare	132
6.1.2 Cod sursă	132
6.1.3 *Rezolvare detaliată	137
6.2 fractii2	137
6.2.1 Indicații de rezolvare	138
6.2.2 Cod sursă	139
6.2.3 *Rezolvare detaliată	141
7 OJI 2013	142
7.1 biperm	142
7.1.1 Indicații de rezolvare	143
7.1.2 Cod sursă	143
7.1.3 *Rezolvare detaliată	147
7.2 subsecvente	148
7.2.1 Indicații de rezolvare	148
7.2.2 Cod sursă	150
7.2.3 *Rezolvare detaliată	164
8 OJI 2012	165
8.1 blis	165
8.1.1 Indicații de rezolvare	166
8.1.2 Cod sursă	167
8.1.3 *Rezolvare detaliată	171
8.2 parc	171
8.2.1 Indicații de rezolvare	172
8.2.2 Cod sursă	172
8.2.3 *Rezolvare detaliată	176
9 OJI 2011	177
9.1 suma	177
9.1.1 Indicații de rezolvare	179
9.1.2 Cod sursă	179
9.1.3 *Rezolvare detaliată	182
9.2 ubuntzei	182
9.2.1 Indicații de rezolvare	183
9.2.2 Cod sursă	184
9.2.3 *Rezolvare detaliată	191

10 OJI 2010	192
10.1 immortal	192
10.1.1 Indicații de rezolvare	193
10.1.2 Cod sursă	193
10.1.3 *Rezolvare detaliată	195
10.2 joc	195
10.2.1 Indicații de rezolvare	196
10.2.2 Cod sursă	197
10.2.3 *Rezolvare detaliată	203
11 OJI 2009	204
11.1 cerc	204
11.1.1 Indicații de rezolvare	205
11.1.2 *Cod sursă	206
11.1.3 *Rezolvare detaliată	206
11.2 Project management	206
11.2.1 Indicații de rezolvare	207
11.2.2 *Cod sursă	208
11.2.3 *Rezolvare detaliată	208
12 OJI 2008	209
12.1 iepuri	209
12.1.1 Indicații de rezolvare	210
12.1.2 Cod sursă	210
12.1.3 *Rezolvare detaliată	212
12.2 numar	212
12.2.1 Indicații de rezolvare	213
12.2.2 Cod sursă	214
12.2.3 *Rezolvare detaliată	218
13 OJI 2007	219
13.1 Numere	219
13.1.1 Indicații de rezolvare	219
13.1.2 Cod sursă	220
13.1.3 Rezolvare detaliată	221
13.2 Cezar	224
13.2.1 Indicații de rezolvare	225
13.2.2 Cod sursă	225
13.2.3 Rezolvare detaliată	227
14 OJI 2006	232
14.1 Graf	232
14.1.1 Indicații de rezolvare	232
14.1.2 Cod sursă	233
14.1.3 Rezolvare detaliată	236
14.2 Cifru	244
14.2.1 Indicații de rezolvare	244
14.2.2 Cod sursă	245
14.2.3 Rezolvare detaliată	246
15 OJI 2005	249
15.1 Lanț	249
15.1.1 Indicații de rezolvare	250
15.1.2 Cod sursă	250
15.1.3 Rezolvare detaliată	252
15.2 Scara	256
15.2.1 Indicații de rezolvare	257
15.2.2 Cod sursă	257
15.2.3 Rezolvare detaliată	260

16 OJI 2004	265
16.1 Moșia	265
16.1.1 *Indicații de rezolvare	266
16.1.2 Cod sursă	266
16.1.3 *Rezolvare detaliată	267
16.2 Lanterna	267
16.2.1 *Indicații de rezolvare	268
16.2.2 Cod sursă	268
16.2.3 *Rezolvare detaliată	271
17 OJI 2003	272
17.1 Compus	272
17.1.1 Indicații de rezolvare	273
17.1.2 Cod sursă	274
17.1.3 *Rezolvare detaliată	275
17.2 Zmeu	275
17.2.1 Indicații de rezolvare	276
17.2.2 Cod sursă	276
17.2.3 *Rezolvare detaliată	278
18 OJI 2002	279
18.1 Urgența	279
18.1.1 *Indicații de rezolvare	280
18.1.2 *Codul sursă	280
18.1.3 *Rezolvare detaliată	280
18.2 Nunta	280
18.2.1 *Indicații de rezolvare	281
18.2.2 *Codul sursă	281
18.2.3 *Rezolvare detaliată	281
II ONI - Olimpiada națională de informatică	282
19 ONI 2019	283
19.1 lexicografic	283
19.1.1 Indicații de rezolvare	284
19.1.2 Cod sursă	284
19.1.3 *Rezolvare detaliată	289
19.2 oracol	289
19.2.1 Indicații de rezolvare	290
19.2.2 Cod sursă	290
19.2.3 *Rezolvare detaliată	292
19.3 treegcd	292
19.3.1 Indicații de rezolvare	293
19.3.2 *Cod sursă	294
19.3.3 *Rezolvare detaliată	294
19.4 compact	294
19.4.1 Indicații de rezolvare	295
19.4.2 *Cod sursă	296
19.4.3 *Rezolvare detaliată	296
19.5 hipersimetrie	296
19.5.1 Indicații de rezolvare	297
19.5.2 *Cod sursă	298
19.5.3 *Rezolvare detaliată	298
19.6 linegraph	298
19.6.1 Indicații de rezolvare	299
19.6.2 *Cod sursă	300
19.6.3 *Rezolvare detaliată	300

20 ONI 2018	301
20.1 aranjare	301
20.1.1 Indicații de rezolvare	302
20.1.2 Cod sursă	303
20.1.3 *Rezolvare detaliată	306
20.2 poligon	306
20.2.1 Indicații de rezolvare	308
20.2.2 Cod sursă	309
20.2.3 *Rezolvare detaliată	313
20.3 tricolor	313
20.3.1 Indicații de rezolvare	314
20.3.2 Cod sursă	315
20.3.3 *Rezolvare detaliată	320
20.4 antivirus	320
20.4.1 Indicații de rezolvare	321
20.4.2 Cod sursă	322
20.4.3 *Rezolvare detaliată	328
20.5 dungeon	328
20.5.1 Indicații de rezolvare	329
20.5.2 Cod sursă	330
20.5.3 *Rezolvare detaliată	338
20.6 zuma	338
20.6.1 Indicații de rezolvare	339
20.6.2 Cod sursă	339
20.6.3 *Rezolvare detaliată	342
21 ONI 2017	343
21.1 incurcatura	343
21.1.1 Indicații de rezolvare	344
21.1.2 Cod sursă	344
21.1.3 *Rezolvare detaliată	348
21.2 startrek	348
21.2.1 Indicații de rezolvare	349
21.2.2 Cod sursă	350
21.2.3 *Rezolvare detaliată	370
21.3 tris	370
21.3.1 Indicații de rezolvare	372
21.3.2 Cod sursă	372
21.3.3 *Rezolvare detaliată	378
21.4 bvarcolaci	378
21.4.1 Indicații de rezolvare	379
21.4.2 Cod sursă	380
21.4.3 *Rezolvare detaliată	385
21.5 minarea	385
21.5.1 Indicații de rezolvare	387
21.5.2 Cod sursă	387
21.5.3 *Rezolvare detaliată	391
21.6 order	391
21.6.1 Indicații de rezolvare	392
21.6.2 Cod sursă	393
21.6.3 *Rezolvare detaliată	394
22 ONI 2016	395
22.1 euro	395
22.1.1 Indicații de rezolvare	396
22.1.2 Cod sursă	397
22.1.3 *Rezolvare detaliată	408
22.2 sushi	409
22.2.1 Indicații de rezolvare	410
22.2.2 Cod sursă	411
22.2.3 *Rezolvare detaliată	430

22.3 xor	430
22.3.1 Indicații de rezolvare	431
22.3.2 Cod sursă	433
22.3.3 *Rezolvare detaliată	438
22.4 arbore	438
22.4.1 Indicații de rezolvare	439
22.4.2 Cod sursă	440
22.4.3 *Rezolvare detaliată	444
22.5 calafat	444
22.5.1 Indicații de rezolvare	445
22.5.2 Cod sursă	446
22.5.3 *Rezolvare detaliată	449
22.6 transform	449
22.6.1 Indicații de rezolvare	451
22.6.2 Cod sursă	451
22.6.3 *Rezolvare detaliată	465
23 ONI 2015	466
23.1 metrou	466
23.1.1 Indicații de rezolvare	467
23.1.2 Cod sursă	468
23.1.3 *Rezolvare detaliată	471
23.2 spiridusi	471
23.2.1 Indicații de rezolvare	473
23.2.2 Cod sursă	473
23.2.3 *Rezolvare detaliată	475
23.3 text	475
23.3.1 Indicații de rezolvare	476
23.3.2 Cod sursă	476
23.3.3 *Rezolvare detaliată	478
23.4 arbvalmax	478
23.4.1 Indicații de rezolvare	479
23.4.2 Cod sursă	479
23.4.3 *Rezolvare detaliată	484
23.5 ksecv	484
23.5.1 Indicații de rezolvare	485
23.5.2 Cod sursă	485
23.5.3 *Rezolvare detaliată	488
23.6 trenuri	488
23.6.1 Indicații de rezolvare	490
23.6.2 Cod sursă	490
23.6.3 *Rezolvare detaliată	501
24 ONI 2014	502
24.1 avârcolaci	502
24.1.1 Indicații de rezolvare	503
24.1.2 Cod sursă	503
24.1.3 Rezolvare detaliată	523
24.2 karb	523
24.2.1 Indicații de rezolvare	524
24.2.2 Cod sursă	524
24.2.3 *Rezolvare detaliată	532
24.3 volum	532
24.3.1 Indicații de rezolvare	533
24.3.2 Cod sursă	534
24.3.3 *Rezolvare detaliată	538
24.4 clepsidra	538
24.4.1 Indicații de rezolvare	539
24.4.2 Cod sursă	540
24.4.3 *Rezolvare detaliată	544
24.5 permutare	544

24.5.1	Indicații de rezolvare	545
24.5.2	Cod sursă	546
24.5.3	*Rezolvare detaliată	551
24.6	xcmmdc	551
24.6.1	Indicații de rezolvare	552
24.6.2	*Cod sursă	553
24.6.3	*Rezolvare detaliată	560
25	ONI 2013	561
25.1	amici	561
25.1.1	Indicații de rezolvare	562
25.1.2	Cod sursă	562
25.1.3	*Rezolvare detaliată	567
25.2	bemo	567
25.2.1	Indicații de rezolvare	568
25.2.2	Cod sursă	569
25.2.3	*Rezolvare detaliată	576
25.3	confuzie	576
25.3.1	Indicații de rezolvare	578
25.3.2	Cod sursă	579
25.3.3	*Rezolvare detaliată	600
25.4	ausoara	600
25.4.1	Indicații de rezolvare	601
25.4.2	Cod sursă	601
25.4.3	*Rezolvare detaliată	608
25.5	drumuri	608
25.5.1	Indicații de rezolvare	609
25.5.2	Cod sursă	610
25.5.3	*Rezolvare detaliată	618
25.6	xnumere	618
25.6.1	Indicații de rezolvare	619
25.6.2	Cod sursă	620
25.6.3	*Rezolvare detaliată	627
26	ONI 2012	628
26.1	search	628
26.1.1	Indicații de rezolvare	629
26.1.2	Cod sursă	629
26.1.3	*Rezolvare detaliată	632
26.2	urat	632
26.2.1	Indicații de rezolvare	633
26.2.2	Cod sursă	634
26.2.3	*Rezolvare detaliată	635
26.3	zlego	635
26.3.1	Indicații de rezolvare	636
26.3.2	Cod sursă	637
26.3.3	*Rezolvare detaliată	643
26.4	drumuri	643
26.4.1	Indicații de rezolvare	645
26.4.2	*Cod sursă	646
26.4.3	*Rezolvare detaliată	669
26.5	minerale	669
26.5.1	Indicații de rezolvare	670
26.5.2	Cod sursă	670
26.5.3	*Rezolvare detaliată	674
26.6	tarabe	674
26.6.1	Indicații de rezolvare	675
26.6.2	Cod sursă	675
26.6.3	*Rezolvare detaliată	679

27 ONI 2011	680
27.1 fotbal	680
27.1.1 Indicații de rezolvare	681
27.1.2 Cod sursă	681
27.1.3 *Rezolvare detaliată	683
27.2 ikebana	683
27.2.1 Indicații de rezolvare	684
27.2.2 Cod sursă	686
27.2.3 *Rezolvare detaliată	688
27.3 posta	688
27.3.1 Indicații de rezolvare	689
27.3.2 Cod sursă	690
27.3.3 *Rezolvare detaliată	693
27.4 pamant	693
27.4.1 Indicații de rezolvare	694
27.4.2 Cod sursă	694
27.4.3 *Rezolvare detaliată	698
27.5 radare	698
27.5.1 Indicații de rezolvare	700
27.5.2 Cod sursă	700
27.5.3 *Rezolvare detaliată	705
27.6 xmoto	705
27.6.1 Indicații de rezolvare	706
27.6.2 Cod sursă	708
27.6.3 *Rezolvare detaliată	709
28 ONI 2010	710
28.1 conex	710
28.1.1 Indicații de rezolvare	711
28.1.2 Cod sursă	712
28.1.3 *Rezolvare detaliată	719
28.2 kmax	719
28.2.1 Indicații de rezolvare	720
28.2.2 Cod sursă	721
28.2.3 *Rezolvare detaliată	723
28.3 submatrix	723
28.3.1 Indicații de rezolvare	724
28.3.2 Cod sursă	725
28.3.3 *Rezolvare detaliată	731
28.4 minuni	731
28.4.1 Indicații de rezolvare	732
28.4.2 Cod sursă	732
28.4.3 *Rezolvare detaliată	740
28.5 diff	740
28.5.1 Indicații de rezolvare	741
28.5.2 Cod sursă	741
28.5.3 *Rezolvare detaliată	746
28.6 stalpi	746
28.6.1 Indicații de rezolvare	747
28.6.2 Cod sursă	748
28.6.3 *Rezolvare detaliată	753
29 ONI 2009	754
29.1 reinvent	754
29.1.1 Indicații de rezolvare	755
29.1.2 Cod sursă	755
29.1.3 *Rezolvare detaliată	758
29.2 revolutie	758
29.2.1 Indicații de rezolvare	759
29.2.2 Cod sursă	759
29.2.3 *Rezolvare detaliată	762

29.3	sirag	762
29.3.1	Indicații de rezolvare	763
29.3.2	Cod sursă	763
29.3.3	*Rezolvare detaliată	768
29.4	matrice	768
29.4.1	Indicații de rezolvare	769
29.4.2	Cod sursă	770
29.4.3	*Rezolvare detaliată	773
29.5	numere	773
29.5.1	Indicații de rezolvare	774
29.5.2	Cod sursă	774
29.5.3	*Rezolvare detaliată	777
29.6	pikachu	777
29.6.1	Indicații de rezolvare	778
29.6.2	Cod sursă	778
29.6.3	*Rezolvare detaliată	784
30	ONI 2008	785
30.1	albinuta	785
30.1.1	Indicații de rezolvare	786
30.1.2	Cod sursă	786
30.1.3	*Rezolvare detaliată	787
30.2	current	788
30.2.1	Indicații de rezolvare	789
30.2.2	Cod sursă	790
30.2.3	*Rezolvare detaliată	792
30.3	pviz	792
30.3.1	Indicații de rezolvare	793
30.3.2	Cod sursă	794
30.3.3	*Rezolvare detaliată	795
30.4	atac	795
30.4.1	Indicații de rezolvare	796
30.4.2	Cod sursă	796
30.4.3	*Rezolvare detaliată	798
30.5	drum	798
30.5.1	Indicații de rezolvare	800
30.5.2	Cod sursă	800
30.5.3	*Rezolvare detaliată	801
30.6	virus	801
30.6.1	Indicații de rezolvare	802
30.6.2	Cod sursă	803
30.6.3	*Rezolvare detaliată	803
31	ONI 2007 clasa a XI-a	804
31.1	Descompunere	804
31.1.1	Indicații de rezolvare	805
31.1.2	Cod sursă	805
31.1.3	*Rezolvare detaliată	806
31.2	Felinare	806
31.2.1	Indicații de rezolvare	807
31.2.2	Cod sursă	807
31.2.3	*Rezolvare detaliată	809
31.3	Joc	809
31.3.1	Indicații de rezolvare	810
31.3.2	Cod sursă	810
31.3.3	*Rezolvare detaliată	811
31.4	Logaritmi	811
31.4.1	Indicații de rezolvare	812
31.4.2	*Cod sursă	812
31.4.3	*Rezolvare detaliată	813
31.5	Maxq	813

31.5.1	Indicații de rezolvare	814
31.5.2	Cod sursă	814
31.5.3	*Rezolvare detaliată	816
31.6	Tric	816
31.6.1	Indicații de rezolvare	817
31.6.2	Cod sursă	817
31.6.3	*Rezolvare detaliată	820
32	ONI 2006 clasa a XI-a	821
32.1	borg	821
32.1.1	Indicații de rezolvare	822
32.1.2	Cod sursă	822
32.1.3	*Rezolvare detaliată	824
32.2	diamant	824
32.2.1	Indicații de rezolvare	825
32.2.2	Cod sursă	825
32.2.3	*Rezolvare detaliată	826
32.3	matrice	826
32.3.1	Indicații de rezolvare	827
32.3.2	Cod sursă	827
32.3.3	*Rezolvare detaliată	829
32.4	Petrom	829
32.4.1	Indicații de rezolvare	829
32.4.2	Cod sursă	830
32.4.3	*Rezolvare detaliată	831
32.5	ratina	831
32.5.1	Indicații de rezolvare	832
32.5.2	Cod sursă	833
32.5.3	*Rezolvare detaliată	834
32.6	vitale	834
32.6.1	Indicații de rezolvare	835
32.6.2	Cod sursă	835
32.6.3	*Rezolvare detaliată	838
33	ONI 2005 clasa a XI-a	839
33.1	Mașina	839
33.1.1	Indicații de rezolvare	840
33.1.2	*Cod sursă	840
33.1.3	*Rezolvare detaliată	840
33.2	Matrice	840
33.2.1	Indicații de rezolvare	841
33.2.2	*Cod sursă	842
33.2.3	*Rezolvare detaliată	842
33.3	Ziduri	842
33.3.1	Indicații de rezolvare	843
33.3.2	*Cod sursă	843
33.3.3	*Rezolvare detaliată	843
33.4	Lsort	844
33.4.1	Indicații de rezolvare	844
33.4.2	*Cod sursă	845
33.4.3	*Rezolvare detaliată	845
33.5	Pătrat	845
33.5.1	Indicații de rezolvare	845
33.5.2	*Cod sursă	846
33.5.3	*Rezolvare detaliată	846
33.6	Cşir	846
33.6.1	Indicații de rezolvare	847
33.6.2	*Cod sursă	847
33.6.3	*Rezolvare detaliată	847

34 ONI 2004	848
34.1 BASE3	848
34.1.1 Indicații de rezolvare	848
34.1.2 Cod sursă	849
34.1.3 *Rezolvare detaliată	858
34.2 Coach	858
34.2.1 Indicații de rezolvare	859
34.2.2 Cod sursă	860
34.2.3 *Rezolvare detaliată	862
34.3 Color	862
34.3.1 Indicații de rezolvare	863
34.3.2 Cod sursă	863
34.3.3 *Rezolvare detaliată	865
34.4 Magic	865
34.4.1 Indicații de rezolvare	865
34.4.2 Cod sursă	866
34.4.3 *Rezolvare detaliată	870
34.5 Pătrate	870
34.5.1 Indicații de rezolvare	871
34.5.2 Cod sursă	871
34.5.3 *Rezolvare detaliată	872
34.6 Turnuri	872
34.6.1 Indicații de rezolvare	873
34.6.2 Cod sursă	873
34.6.3 *Rezolvare detaliată	874
35 ONI 2003	875
35.1 Asmin	875
35.1.1 Indicații de rezolvare	876
35.1.2 Cod sursă	876
35.1.3 *Rezolvare detaliată	878
35.2 Căutare	879
35.2.1 Indicații de rezolvare	880
35.2.2 Cod sursă	880
35.2.3 *Rezolvare detaliată	881
35.3 A007	881
35.3.1 Indicații de rezolvare	882
35.3.2 Cod sursă	883
35.3.3 *Rezolvare detaliată	884
35.4 Inter	884
35.4.1 Indicații de rezolvare	885
35.4.2 Cod sursă	885
35.4.3 *Rezolvare detaliată	888
35.5 Nr	888
35.5.1 Indicații de rezolvare	889
35.5.2 Cod sursă	889
35.5.3 *Rezolvare detaliată	891
35.6 Proc	891
35.6.1 Indicații de rezolvare	892
35.6.2 Cod sursă	893
35.6.3 *Rezolvare detaliată	897
36 ONI 2002	898
36.1 Arboare	898
36.1.1 Indicații de rezolvare	899
36.1.2 Cod sursă	899
36.1.3 *Rezolvare detaliată	907
36.2 Decod	907
36.2.1 *Indicații de rezolvare	908
36.2.2 Cod sursă	908
36.2.3 *Rezolvare detaliată	913

36.3	Seti	913
36.3.1	Indicații de rezolvare	914
36.3.2	*Rezolvare detaliată	914
36.3.3	Cod sursă	914
36.4	Suma divizorilor	917
36.4.1	Indicații de rezolvare	918
36.4.2	Cod sursă	918
36.4.3	*Rezolvare detaliată	919
36.5	Sistem	919
36.5.1	Indicații de rezolvare	920
36.5.2	*Rezolvare detaliată	920
36.5.3	Cod sursă	920
36.6	Comitat	923
36.6.1	Indicații de rezolvare	925
36.6.2	Cod sursă	925
36.6.3	*Rezolvare detaliată	931
37	ONI 2001	932
37.1	Comparări	932
37.1.1	*Indicații de rezolvare	935
37.1.2	Cod sursă	935
37.1.3	*Rezolvare detaliată	937
37.2	Relee	937
37.2.1	*Indicații de rezolvare	938
37.2.2	Cod sursă	938
37.2.3	*Rezolvare detaliată	940
37.3	Telecomanda	940
37.3.1	*Indicații de rezolvare	941
37.3.2	Cod sursă	941
37.3.3	*Rezolvare detaliată	944
37.4	Entries	945
37.4.1	*Indicații de rezolvare	945
37.4.2	Cod sursă	945
37.4.3	*Rezolvare detaliată	948
37.5	Robot	948
37.5.1	*Indicații de rezolvare	949
37.5.2	Cod sursă	949
37.5.3	*Rezolvare detaliată	958
37.6	Text mare	958
37.6.1	*Indicații de rezolvare	959
37.6.2	Cod sursă	959
37.6.3	*Rezolvare detaliată	964
38	ONI 2000	965
38.1	Arbore	965
38.1.1	Indicații de rezolvare	966
38.1.2	Cod sursă	966
38.1.3	*Rezolvare detaliată	970
38.2	Moara	970
38.2.1	*Indicații de rezolvare	971
38.2.2	Cod sursă	971
38.2.3	*Rezolvare detaliată	980
38.3	Puncte	980
38.3.1	Indicații de rezolvare	981
38.3.2	Cod sursă	981
38.3.3	*Rezolvare detaliată	981
38.4	SICN	982
38.4.1	Indicații de rezolvare	982
38.4.2	Cod sursă	982
38.4.3	*Rezolvare detaliată	986
38.5	Spioni	987

38.5.1	Indicații de rezolvare	987
38.5.2	Cod sursă	988
38.5.3	*Rezolvare detaliată	989
38.6	Tezaur	990
38.6.1	Indicații de rezolvare	991
38.6.2	Cod sursă	991
38.6.3	*Rezolvare detaliată	998
Index		999
Bibliografie		1002
Lista autorilor		1005

Lista figurilor

1.1 rufe	11
1.2 rufe	11
1.3 rufe	11
1.4 rufe	12
1.5 tairoș	19
2.1 aquapark	43
4.1 summax	103
4.2 summax	103
5.1 2sah	109
6.1 cartite	130
7.1 biperm	142
7.2 biperm	143
8.1 parc	171
9.1 suma	177
9.2 suma	177
11.1 cerc	205
13.1 Cezar	225
17.1 Compus1	272
17.2 Compus2	273
18.1 Nunta	281
19.1 hipersimetrie	297
31.1 Sigla ONI 2007	804
32.1 Sigla ONI 2006	821
32.2 Vitale	835
33.1 Masina	839
33.2 Ziduri1	842
33.3 Ziduri2	843
34.1 Patrate1	870
34.2 Patrate2	870
34.3 Turnuri1	872
34.4 Turnuri2	873
35.1 Asmin	876
35.2 Cautare	879
36.1 Sistem	920

36.2 Comitat	924
37.1 Relee	938
37.2 Telecomanda	940
38.1 Arboare	966
38.2 Tezaur	990

Lista tabelelor

Lista programelor

1.1.1	conexidad_bogdan100.cpp	3
1.1.2	conexidad_lukacs100.cpp	4
1.1.3	conexidad_mihai100.cpp	6
1.1.4	conexidad_radu100.cpp	8
1.2.1	rufe_bc.cpp	12
1.2.2	rufe_LS_LogN.cpp	13
1.2.3	rufe_LS_NplusM.cpp	14
1.2.4	rufe_PA_log.cpp	15
1.2.5	rufe_SzZ_brut.cpp	16
1.2.6	rufe_SzZ_log.cpp	17
1.3.1	tairos_bogdan.cpp	21
1.3.2	tairos_Lukacs.cpp	22
1.3.3	tairos_mihai.cpp	24
1.3.4	tairos_radu.cpp	25
1.3.5	tairos_radu_brut.cpp	26
1.3.6	tairos_zoli.cpp	27
2.1.1	galeti_rec_PA.cpp	31
2.1.2	galeti_SzZ_ite.cpp	32
2.1.3	galeti_SzZ_rec.cpp	33
2.1.4	galeti100_PA.cpp	33
2.1.5	mihai-100.cpp	34
2.2.1	mihai100.cpp	38
2.2.2	mihaiSmallN.cpp	39
2.2.3	mihaiSmallT.cpp	40
2.2.4	nostl_PA.cpp	41
2.2.5	ramen_set_PA.cpp	42
2.3.1	Adrian-100-set.cpp	45
2.3.2	Adrian-100-stack.cpp	47
2.3.3	aquapark75_SzZ.cpp	50
3.1.1	armonica_eugen.cpp	53
3.1.2	armonica_eugen0.cpp	54
3.1.3	armonica_eugen1.cpp	55
3.1.4	armonica_eugen2.cpp	55
3.1.5	armonica_VG.cpp	56
3.1.6	armonica_Zoli.cpp	57
3.1.7	armonicaPA.cpp	58
3.1.8	armonicaPAb Brut.cpp	59
3.2.1	eric_ninjago.cpp	63
3.2.2	fastlikeaninja_PA.cpp	65
3.2.3	ninjago_eugen.cpp	66
3.2.4	ninjago_primPA.cpp	68
3.2.5	ninjagoBubbleS_Codruta.cpp	70
3.2.6	ninjagoCRadix2.cpp	72
3.2.7	ninjagoCRadixS_Codruta.cpp	74
3.2.8	ninjagoPA.cpp	76
3.2.9	ninjagoPMD_Codruta.cpp	77
3.2.10	ninjagoPrim_mat_Codruta.cpp	79
3.2.11	ninjagoQuickS_Codruta.cpp	80
3.2.12	ninjagoSTL_Codruta.cpp	82

3.3.1	back_Eric_.cpp	86
3.3.2	permutare_Eric.cpp	87
3.3.3	permutare_VG_.cpp	89
3.3.4	permutarebrut_Zoli_.cpp	90
3.3.5	permutarePA.cpp	92
3.3.6	permutareZoli.cpp	93
4.1.1	elicoptere100_DAP.cpp	97
4.1.2	elicoptere100_PA.cpp	100
4.1.3	elicoptere100_PA_naiv.cpp	101
4.2.1	summax_80.cpp	106
4.2.2	summax_100_PA.cpp	106
5.1.1	2sah_brut.cpp	111
5.1.2	2sah_clasic.cpp	112
5.1.3	2sah_liniar.cpp	112
5.1.4	2sah_oficiala.cpp	113
5.1.5	2sah_smart.cpp	114
5.2.1	dragoni.cpp	117
5.2.2	Dragoni_GD.cpp	119
5.2.3	dragoni_noGraph_int.cpp	121
5.2.4	dragoni_WithGraphMLE.cpp	122
5.2.5	dragoniBeFo.cpp	124
5.2.6	dragomiHeap.cpp	126
5.2.7	dragoniRF.cpp	128
6.1.1	AdrianPanaeteFleury2.cpp	132
6.1.2	AdrianPanaeteRecursiv1.cpp	134
6.1.3	DoruAnastasiuPopescucartite.cpp	135
6.2.1	fractii2.cpp	139
6.2.2	fractii2_back.cpp	139
6.2.3	fractii2_var2.cpp	140
7.1.1	biperm.cpp	143
7.1.2	biperm_exp.cpp	145
7.1.3	biperm1.cpp	146
7.2.1	subsecvente_ans_build.cpp	150
7.2.2	subsecvente_arb.cpp	151
7.2.3	subsecvente_arb_static_arrays.cpp	152
7.2.4	subsecvente_arb_stl.cpp	153
7.2.5	subsecvente_binary_search.cpp	154
7.2.6	subsecvente_binary_search_naive.cpp	155
7.2.7	subsecvente_dp.cpp	156
7.2.8	subsecvente_linear_search.cpp	159
7.2.9	subsecvente_log.cpp	160
7.2.10	subsecvente_naive.cpp	161
7.2.11	subsecvente_naive_optimized.cpp	162
7.2.12	subsecvente_radix.cpp	163
8.1.1	blis70p.cpp	167
8.1.2	blis100p.cpp	168
8.1.3	PA_n2blis.cpp	169
8.1.4	PA_pblis.cpp	170
8.2.1	PA_parc2.cpp	172
8.2.2	parc_Zoli_double.cpp	174
9.1.1	suma1_100.cpp	179
9.1.2	suma2_100.cpp	180
9.1.3	suma3_st.cpp	181
9.2.1	bellmanford.cpp	184
9.2.2	dijkstra.cpp	186
9.2.3	dp.cpp	188
9.2.4	royfloyd.cpp	189
10.1.1	immortalc_c.c	193
10.1.2	immortal_cpp.cpp	194
10.2.1	joc_back1.cpp	197

10.2.2	joc_dinamical.cpp	198
10.2.3	joc_dinamica2.cpp	198
10.2.4	joc_greedy.cpp	200
10.2.5	joc_memoizare.cpp	201
12.1.1	IEPURI.cpp	210
12.1.2	IEPURI60.cpp	211
12.2.1	numar30.cpp	214
12.2.2	numar40.cpp	215
12.2.3	numar50.cpp	216
12.2.4	numar100.cpp	217
13.1.1	numerev1.cpp	220
13.1.2	numere1.java	221
13.1.3	numere2.java	222
13.2.1	cezar.cpp	225
13.2.2	cezar1a.java	227
13.2.3	cezar1b.java	228
13.2.4	cezar2.java	230
14.1.1	GRAF40.cpp	233
14.1.2	graf100.cpp	234
14.1.3	graf1.java	236
14.1.4	graf2.java	239
14.1.5	graf3.java	241
14.2.1	stelian.cpp	246
14.2.2	cifru1.java	246
14.2.3	cifru2.java	247
15.1.1	LANT.cpp	250
15.1.2	Lant1.java	252
15.1.3	Lant2.java	254
15.2.1	scara.cpp	257
15.2.2	scaraDP.cpp	259
15.2.3	Scara1a.java	260
15.2.4	Scara1b.java	261
15.2.5	Scara2a.java	262
15.2.6	Scara2b.java	263
16.1.1	POLIGON.pas	266
16.2.1	LANT2.pas	269
17.1.1	compus.pas	274
17.1.2	COMPUS1.pas	275
17.2.1	zmeu.pas	276
19.1.1	lexicografic_100p_1.cpp	284
19.1.2	lexicografic_100p_2.cpp	286
19.1.3	lexicografic_O(n^2).cpp	288
19.2.1	oracol_kruskal_100p.cpp	290
19.2.2	oracol_prim_100p.cpp	291
20.1.1	bogd_ciob_aranjare100.cpp	303
20.1.2	mapa_radix.cpp	304
20.1.3	tamio_100.cpp	305
20.2.1	n2_costel_100.cpp	309
20.2.2	n5_bogdan_30.cpp	311
20.3.1	alex_80.cpp	315
20.3.2	alex_100.cpp	316
20.3.3	mapa_2_n.cpp	318
20.3.4	n2_tamio_100.cpp	319
20.4.1	alex_50.cpp	322
20.4.2	alex_100.cpp	323
20.4.3	costel_n2_100.cpp	324
20.4.4	costel_n3_50.cpp	326
20.4.5	tamio_n2_100.cpp	327
20.5.1	adrian_100.cpp	330
20.5.2	eudanip_100.cpp	332

20.5.3	eudanip_back_20.cpp	334
20.5.4	mapa_lant_20.cpp	335
20.6.1	bc_zuma_n3_100.cpp	339
20.6.2	bc_zuma_n3sigma_70.cpp	340
20.6.3	eric_back.cpp	341
21.1.1	incurcatura_70p.cpp	344
21.1.2	incurcatura_100p_1.cpp	345
21.1.3	incurcatura_100p_2.cpp	347
21.2.1	startrek_30p.cpp	350
21.2.2	startrek_70p.cpp	355
21.2.3	startrek_80p_1.cpp	357
21.2.4	startrek_80p_2.cpp	359
21.2.5	startrek_90p.cpp	363
21.2.6	startrek_100p_1.cpp	365
21.2.7	startrek_100p_2.cpp	368
21.3.1	tris_94p.cpp	372
21.3.2	tris_100p.cpp	374
21.4.1	bvarcolaci_80p.cpp	380
21.4.2	bvarcolaci_100p_1.cpp	382
21.4.3	bvarcolaci_100p_2.cpp	383
21.5.1	minarea.cpp	387
21.5.2	minarea_100p_1.cpp	389
21.5.3	minarea_100p_2.cpp	390
21.6.1	order_100p_1.cpp	393
21.6.2	order_100p_2.cpp	393
22.1.1	euro_alex_100.cpp	397
22.1.2	euro_ciucu_100.cpp	398
22.1.3	euro_denis_20.cpp	399
22.1.4	euro_denis_50.cpp	401
22.1.5	euro_denis_100.cpp	403
22.1.6	euro_radu_50.cpp	405
22.1.7	euro_razvan_40.cpp	405
22.1.8	euro_razvan_100.cpp	407
22.1.9	euro_stefan_100.cpp	408
22.2.1	daniBrute.cpp	411
22.2.2	daniN.cpp	413
22.2.3	daniNlogN.cpp	415
22.2.4	Stefan_Popa_NlogN.cpp	417
22.2.5	sushi_bulaneala.cpp	419
22.2.6	sushi-HH-100.cpp	421
22.2.7	sushi-HH-100-hash.cpp	423
22.2.8	sushi-HH-brute.cpp	426
22.2.9	sushi-Nlog-vladgavrila.cpp	428
22.3.1	denis_brut.cpp	433
22.3.2	denis_brut_precalculare.cpp	433
22.3.3	oficiala.cpp	434
22.3.4	optim.cpp	434
22.3.5	radu_oficial.cpp	436
22.3.6	radu_task3.cpp	436
22.3.7	radu_task12.cpp	437
22.4.1	arboreAH.cpp	440
22.4.2	arboreDM1.cpp	441
22.4.3	arboreRS.cpp	442
22.5.1	calafatDEM.cpp	446
22.5.2	calafatDM_aib.cpp	447
22.5.3	calafatDM_aint.cpp	448
22.6.1	AH_75p.cpp	451
22.6.2	transformAH1.cpp	454
22.6.3	transformAH2.cpp	458
22.6.4	transformZS.cpp	461

23.1.1	.cpp	468
23.2.1	spiridusi.cpp	473
23.3.1	.cpp	476
23.4.1	arborevalmax.cpp	479
23.4.2	arbvalmax-cartita100.cpp	481
23.4.3	arbvalmax-vladii.cpp	482
23.5.1	ksecv.cpp	485
23.5.2	ksecv-mihai.cpp	487
23.6.1	trenuri100.cpp	490
23.6.2	trenuri-andrei-ciocan.cpp	492
23.6.3	trenuri-cartita100.cpp	494
23.6.4	trenuri-HH-100.cpp	497
23.6.5	trenuri-vladii.cpp	499
24.1.1	avarcolaci-10-andrei.c	503
24.1.2	avarcolaci-20-andrei.c	504
24.1.3	avarcolaci-40-andrei.c	505
24.1.4	avarcolaci-20-mihai-random.cpp	506
24.1.5	avarcolaci-30-adrian-8bit.cpp	507
24.1.6	avarcolaci-40-adrian-16bit.cpp	508
24.1.7	avarcolaci-50-mihai-parsare.cpp	509
24.1.8	avarcolaci-70-mihai.cpp	510
24.1.9	avarcolaci-80-adrian-8bit.cpp	511
24.1.10	avarcolaci-80-andrei.cpp	512
24.1.11	avarcolaci-80-andrei-sort.cpp	513
24.1.12	avarcolaci-80-bogdan.cpp	514
24.1.13	avarcolaci-80-mihai-sortare.cpp	515
24.1.14	avarcolaci-90-andrei.cpp	516
24.1.15	avarcolaci-100-adrian-8bit.cpp	517
24.1.16	avarcolaci-100-andrei.cpp	519
24.1.17	avarcolaci-100-bogdan.cpp	520
24.1.18	avarcolaci-100-bogdan-parsare.cpp	521
24.2.1	karb_brut-mihai.cpp	525
24.2.2	karb_cartita.cpp	526
24.2.3	karb_npatrat-mihai.cpp	527
24.2.4	karb-100p-mihai.cpp	529
24.2.5	karb-ciocan.cpp	530
24.3.1	volum1.cpp	534
24.3.2	volum2.cpp	535
24.3.3	volum3.cpp	536
24.3.4	volum4.cpp	537
24.4.1	clepsidra.cpp	540
24.4.2	clepsidra_bulaneala.cpp	540
24.4.3	clepsidra_gavrilă_log.cpp	541
24.4.4	clepsidra-100cartita.cpp	542
24.4.5	clepsidra-brutcartita.cpp	543
24.5.1	brut_40.cpp	546
24.5.2	permutare-100cartita.cpp	547
24.5.3	permutare-ciocan-100.cpp	548
24.5.4	permutarePA.cpp	549
24.5.5	permutarePAn2.cpp	550
24.6.1	xcmmdc.cpp	553
24.6.2	xcmmdc_brut_brut.cpp	555
24.6.3	xcmmdc-cartita1.cpp	556
24.6.4	xcmmdc-cartita2.cpp	557
24.6.5	xcmmdc-cartita3.cpp	558
24.6.6	xcmmdc-cartita4.cpp	559
25.1.1	amici.cpp	562
25.1.2	amici_bogdan.cpp	563
25.1.3	amici_brute.cpp	564
25.1.4	amici_simulare.cpp	566

25.2.1	bemo_adi.cpp	569
25.2.2	bemo_mugurel_log.cpp	569
25.2.3	bemo_mugurel_log_parsed.cpp	571
25.2.4	bemo_n^2_logn.cpp	573
25.2.5	bemo-bogdan.cpp	574
25.2.6	bemo-N3-cristi.cpp	575
25.3.1	confuzie-40-adrian.cpp	579
25.3.2	confuzie-40-cristi.cpp	580
25.3.3	confuzie-50-andrei.cpp	581
25.3.4	confuzie-50-mugurel.cpp	583
25.3.5	confuzie-70-andrei.cpp	585
25.3.6	confuzie-70-dragos.cpp	587
25.3.7	confuzie-100-adi.cpp	590
25.3.8	confuzie-100-alex.cpp	593
25.3.9	confuzie-100-andrei.cpp	596
25.3.10	confuzie-100-bogdan.cpp	598
25.4.1	ausoara-90-bogdan.cpp	601
25.4.2	ausoara-100-bogdan.c	602
25.4.3	ausoara_60_marius.cpp	603
25.4.4	ausoara_dragos.cpp	604
25.4.5	ausoara_PA.cpp	605
25.4.6	ausoara-1-adrianP.cpp	606
25.4.7	ausoara-100-mugurel.cpp	606
25.4.8	ausoara-100-stl.cpp	607
25.5.1	drumuri-50-mugurel.cpp	610
25.5.2	drumuri-100-adi.cpp	611
25.5.3	drumuri-100-alex.cpp	613
25.5.4	drumuri-100-andrei.cpp	616
25.6.1	xnumere-85-bogdan.c	620
25.6.2	xnumere-100-bogdan.c	621
25.6.3	xnumere-10-dragos.cpp	622
25.6.4	xnumere-30-dragos.cpp	623
25.6.5	xnumere-60-adrian.cpp	624
25.6.6	xnumere-100-mugurel.cpp	625
26.1.1	search-cazacu-100.cpp	629
26.1.2	vlad_search_100.cpp	630
26.2.1	uratAdrian100.cpp	634
26.2.2	uratBKAdrian20.cpp	634
26.3.1	zlego.cpp	637
26.3.2	zlego-20.cpp	638
26.3.3	zlego-50.cpp	638
26.3.4	zlego-50-adrian.cpp	639
26.3.5	zlego-50-mugurel.cpp	640
26.3.6	zlego-100-adrian.cpp	641
26.3.7	zlego-100-bogdan.cpp	642
26.3.8	zlego-100-mugurel.cpp	642
26.4.1	drumuri.cpp	646
26.4.2	drumuri-40-cosmin.cpp	649
26.4.3	drumuri-40-mugurel.cpp	651
26.4.4	drumuri-40-mugurel2.cpp	655
26.4.5	drumuri-100-bogdan.cpp	658
26.4.6	drumuri-100-tibi.cpp	662
26.5.1	minerale.cpp	671
26.5.2	minerale-20-vlad.cpp	671
26.5.3	minerale-70-mugurel.cpp	672
26.6.1	tarabe.cpp	675
26.6.2	tarabe-20.cpp	676
26.6.3	tarabe-60.cpp	677
26.6.4	tarabe-100-mugurel.cpp	678
27.1.1	fotbal_100.cpp	681

27.1.2	fotbal100.cpp	682
27.2.1	ikebana-100.c	686
27.3.1	posta_100.cpp	690
27.3.2	posta100.cpp	691
27.4.1	pamant0.cpp	694
27.4.2	pamant1.cpp	695
27.4.3	pamant2.cpp	697
27.5.1	radare-60.c	700
27.5.2	radare_100.cpp	702
27.5.3	radare60.cpp	703
27.6.1	xmoto.cpp	708
28.1.1	conex4.cpp	712
28.1.2	conex_10p.cpp	712
28.1.3	conex_50p_3.cpp	713
28.1.4	conex_50p.cpp	714
28.1.5	conex_50p2.cpp	715
28.1.6	conex_100p.cpp	716
28.1.7	conex_100p2.cpp	717
28.1.8	conex3.cpp	718
28.2.1	kmax_20.cpp	721
28.2.2	kmax_100.cpp	721
28.2.3	kmax_100b.cpp	723
28.3.1	submatrix_20.cpp	725
28.3.2	submatrix_40_adrian.cpp	725
28.3.3	submatrix_70.cpp	726
28.3.4	submatrix_100.cpp	728
28.3.5	submatrix_100_adrian.cpp	729
28.4.1	minuni1.cpp	732
28.4.2	minuni2.cpp	734
28.4.3	minuni3.cpp	735
28.4.4	minuni4.cpp	737
28.4.5	minuni5.cpp	739
28.5.1	diff1.cpp	741
28.5.2	diff2.cpp	742
28.5.3	diff3.cpp	743
28.5.4	diff4.cpp	745
28.5.5	diff5.cpp	745
28.6.1	stalpi1.cpp	748
28.6.2	stalpi2.cpp	749
28.6.3	stalpi3.cpp	750
28.6.4	stalpi4.cpp	750
28.6.5	stalpi5.cpp	751
29.1.1	reinvent.cpp	755
29.1.2	reinvent2.cpp	756
29.1.3	reinvent3.cpp	757
29.2.1	revolutie.cpp	759
29.2.2	revolutie_eval.cpp	760
29.2.3	revolutie_gener.cpp	761
29.3.1	brute_sirag.cpp	763
29.3.2	brute_sirag2.cpp	764
29.3.3	brute_sirag3.cpp	765
29.3.4	sirag.cpp	766
29.3.5	sirag_10.cpp	767
29.4.1	brute15.cpp	770
29.4.2	brute35.cpp	771
29.4.3	brute.cpp	772
29.5.1	back20p.cpp	774
29.5.2	numere.cpp	775
29.5.3	numere100p.cpp	776
29.6.1	pikachu.cpp	778

29.6.2	pikachu2.cpp	780
29.6.3	pikachu3.cpp	782
29.6.4	pikachu4.cpp	783
30.1.1	albinuta.cpp	786
30.2.1	curent.cpp	790
30.3.1	pviz.cpp	794
30.4.1	atac.cpp	796
30.5.1	drum.cpp	800
30.6.1	virus.cpp	803
31.1.1	desc.c	805
31.2.1	felinare.cpp	808
31.3.1	joc.cpp	810
31.4.1	log.cpp	812
31.5.1	maxq.cpp	814
31.6.1	tric.cpp	817
32.1.1	borg.cpp	822
32.1.2	borg2.cpp	823
32.2.1	diamant.cpp	825
32.3.1	matrice.cpp	827
32.4.1	petrom.c	830
32.5.1	ratina.cpp	833
32.6.1	vitale.pas	836
34.1.1	base3bkt1.pas	849
34.1.2	base3bkt2.pas	850
34.1.3	base3ok.pas	852
34.1.4	base3ok2.pas	854
34.2.1	coach.c	860
34.2.2	coachsl.c	861
34.3.1	color_ok.pas	863
34.4.1	magic.cpp	866
34.4.2	magicsl.cpp	868
34.5.1	patrate.c	871
34.5.2	psmall.c	872
34.6.1	turnuri.cpp	873
35.1.1	asmin.cpp	876
35.2.1	cautare.pas	880
35.3.1	a007.pas	883
35.4.1	inter.pas	885
35.5.1	Nr.cpp	889
35.6.1	proc.pas	893
35.6.2	PROCBKT1.pas	894
35.6.3	PROCGRE1.pas	895
35.6.4	procok.pas	896
36.1.1	ARB_MIA.pas	899
36.1.2	arbor.pas	903
36.2.1	decod.pas	908
36.3.1	seti.c	914
36.4.1	sumdiv.c	918
36.5.1	Sist_ok.pas	920
36.6.1	COMI.pas	925
36.6.2	COMIBCK.pas	928
36.6.3	COMITATB.pas	930
37.1.1	Comp_MIA.pas	935
37.1.2	COMPAR.pas	936
37.2.1	Relee_MIA.pas	938
37.3.1	Tele_MIA.pas	941
37.4.1	Entries_MIA.pas	945
37.5.1	Rob2_MIA.pas	949
37.5.2	Rob3_MIA.pas	952
37.5.3	Robot_MIA.pas	955

37.6.1	Textmare_MIA.pas	959
38.1.1	Arbore_comisie.cpp	966
38.1.2	Arbore_MIA.pas	968
38.2.1	Moara_MIA.pas	971
38.2.2	Moara_MIA_2.pas	974
38.2.3	Moara-ofic.pas	978
38.3.1	puncte_dpa.pas	981
38.4.1	Sicn_MIA.pas	982
38.4.2	Sicn-ofic.pas	985
38.5.1	Spioni_comisie.pas	988
38.6.1	Tezaur_comisie.pas	991
38.6.2	Tezaur_MIA.pas	994

Partea I

OJI - Olimpiada județeană de informatică

Capitolul 1

OJI 2019

1.1 conexidad

Problema 1 - conexidad

90 de puncte

Fie un graf neorientat cu N noduri și M muchii, care NU este conex.

Cerințe

Să i se adauge grafului un număr minim de muchii, astfel încât acesta să devină conex.

Fie $extra_i$ numărul de muchii nou-adăugate care sunt incidente cu nodul i , iar max_extra cea mai mare dintre valorile $extra_1, extra_2, \dots, extra_N$. Multimea de muchii adăugate trebuie să respecte condiția ca valoarea max_extra să fie minimă.

Date de intrare

Pe prima linie a fișierului de intrare **conexidad.in** se află două numere naturale N și M , iar pe fiecare dintre următoarele M linii se află câte o pereche de numere a, b , semnificând faptul că există muchia $[a, b]$. Numerele aflate pe aceeași linie a fișierului sunt separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire **conexidad.out** va conține pe prima linie valoarea max_extra . Pe a doua linie va conține valoarea K reprezentând numărul de muchii nou-adăugate în graf. Fiecare dintre următoarele K linii va conține câte o pereche de numere c, d , separate prin câte un spațiu, semnificând faptul că se adaugă grafului muchia $[c, d]$.

Restricții și precizări

- $1 \leq N \leq 100$
- $0 \leq M \leq N * (N - 1)/2$
- Nodurile grafului sunt numerotate de la 1 la N inclusiv.
- Muchiile prezente în fișierul de intrare sunt distințe.
- Pentru orice muchie $[a, b]$ aflată în fișierul de intrare, avem $a \neq b$.
- Graful din fișierul de intrare nu este conex.
- În cazul în care soluția afișată pentru un anumit test conectează graful cu număr minim de muchii, dar nu minimizează valoarea lui max_extra , se vor acorda 50% din punctajul pentru testul respectiv.
- Dacă există mai multe soluții optime, se va admite oricare dintre acestea.

Exemple

conexidad.in	conexidad.out	Explicații
4 2 1 2 4 2	1 1 3 1	Graful este format din două componente conexe, cu noduri din multimea $\{1, 2, 4\}$ respectiv nodul izolat 3. După adăugarea muchiei $(3, 1)$ vom avea valorile $extra_1 = 1$, $extra_2 = 0$, $extra_3 = 1$, $extra_4 = 0$, deci $max_extra = 1$. Se poate demonstra că nu există soluție cu $max_extra < 1$.
5 1 3 4	2 3 1 3 2 3 4 5	Graful este format din patru componente conexe, cu noduri din multimea $\{3, 4\}$, respectiv nodurile izolate 1, 2 și 5. După adăugarea muchiilor $(1, 3)$, $(2, 3)$ și $(4, 5)$, vom avea valorile $extra_1 = 1$, $extra_2 = 1$, $extra_3 = 2$, $extra_4 = 1$, $extra_5 = 1$, deci $max_extra = 2$. Se poate demonstra că nu există soluție cu $max_extra < 2$.

Timp maxim de executare/test: 1.0 secunde

Memorie: total 64 MB din care pentru stivă 32 MB

Dimensiune maximă a sursei: 20 KB

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

1.1.1 Indicații de rezolvare

Mihai Calancea

- Numărul minim de muchii necesare pentru a conecta graful este $C - 1$, unde C este numărul de *componente conexe*. Componentele conexe se pot determina printr-o *parcursere în lățime* sau *în adâncime*.

- Observăm că este posibil întotdeauna să conectăm componentele într-un *lanț*, selectând câte un nod din fiecare componentă iar apoi legând aceste noduri secvențial.

- Această construcție produce o valoare max_extra cel mult egală cu 2. Valoarea lui max_extra nu poate fi niciodată 0, deci în continuare rămână să analizăm cazurile în care este posibil să obținem $max_extra = 1$.

- Observăm că dacă toate *componentele conexe* au mărime cel puțin 2, este posibil să le legăm în *lanț* cu valoarea $max_extra = 1$, folosind pentru fiecare componentă două noduri diferite pentru a duce muchia către componenta precedentă, respectiv către componenta următoare. Rămâne să analizăm cazul componentelor de mărime 1 (numite și *noduri izolate*).

- La modul general, o soluție care urmărește $max_extra = 1$ nu va duce muchie între două noduri izolate, fiindcă conectarea ulterioară a acestora cu restul grafului va adăuga sigur o a doua muchie cel puțin una dintre noduri. Excepție face graful alcătuit din doar două noduri, ambele izolate.

- În consecință, dorim să conectăm nodurile izolate cu componente mari (de mărime cel puțin 2) prin noduri ale acestor componente care nu au fost încă folosite pentru a duce muchii.

- Având B componente mari, exact $2 * (B - 1)$ noduri ale acestora vor fi folosite pentru a conecta componente mari între ele. Restul nodurilor sunt considerate libere și pot fi legate cu noduri izolate. Dacă există suficiente noduri libere pentru a acoperi toate nodurile izolate, $max_extra = 1$, altfel $max_extra = 2$.

- Soluția poate fi implementată în timp $O(N + M)$, dar limitele datelor de intrare sunt suficient de mici pentru a oferi punctaj maxim unor soluții de complexitate $O(N^3)$ sau $O(N^2)$.

1.1.2 Cod sursă

Listing 1.1.1: conexidad_bogdan100.cpp

```

1 #include <stdio.h>
2 #include <algorithm>
3 #include <numeric>
4 #include <memory>
5
6 using namespace std;
7
8 using file_pointer = unique_ptr<FILE, decltype(&fclose)>;
9
10 enum { OUTSIDE_COMPONENT, IN_COMPONENT };

```

```

11
12 int p[100], extra[100];
13 int N, M;
14
15 int root(int node)
16 {
17     while (node != p[node]) node = p[node];
18     return node;
19 }
20
21 void join(int x, int y)
22 {
23     x = root(x); y = root(y);
24     if (x == y) return;
25     p[x] = y;
26 }
27
28 void solve(file_pointer& f)
29 {
30     int node[2] = {-1, -1};
31     for (int lap: {OUTSIDE_COMPONENT, IN_COMPONENT})
32         for (int i = 0; i < N; ++i)
33             if ((root(i) == root(0)) == lap
34                 && (node[lap] == -1 || extra[node[lap]] > extra[i]))
35                 node[lap] = i;
36
37     if (node[0] == -1 || node[1] == -1)
38     {
39         fprintf(f.get(), "%d\n%d\n", *max_element(extra, extra + N),
40                 accumulate(extra, extra + N, 0) / 2);
41     }
42     else
43     {
44         join(node[0], node[1]);
45         ++extra[node[0]]; ++extra[node[1]];
46         solve(f);
47         fprintf(f.get(), "%d %d\n", node[0] + 1, node[1] + 1);
48     }
49 }
50
51 int main()
52 {
53     file_pointer f=fopen("conexidad.in", "r"), &fclose);
54     fscanf(f.get(), "%d%d", &N, &M);
55     iota(p, p + N, 0);
56     for (int _ = 0; _ < M; ++_)
57     {
58         int x, y; fscanf(f.get(), "%d%d", &x, &y); --x; --y;
59         join(x, y);
60     }
61
62     f.reset(fopen("conexidad.out", "w"));
63     solve(f);
64     return 0;
65 }
```

Listing 1.1.2: conexidad_lukacs100.cpp

```

1 // Autor Lukacs Sandor
2 #include <iostream>
3 #include <fstream>
4
5 using namespace std;
6
7 ifstream in("conexidad.in");
8 ofstream out("conexidad.out");
9
10 int n, m, k, a[102][5000]; //a - liste de adiacenta
11 int b[102][5000]; //b[i] - nodurile din componenta conexa i
12 int viz[102];
13 int c1[102], n1; //componente conexe cu 1 singur nod
14 int c2[102], n2; //componente conexe cu mai multe noduri
15
16 void dfs(int x)
17 {
```

```

18     viz[x] = 1; //marchez nodul x ca vizitat
19     b[k][0] = b[k][0] + 1; //adaug nodul x in componenta conexa k
20     b[k][b[k][0]] = x;
21     for(int i = 1; i <= a[x][0]; i++)
22         if(viz[a[x][i]] == 0) dfs(a[x][i]);
23 }
24
25 int main()
26 {
27     in >> n >> m;
28
29     if(n == 2 and m == 0) {out << "1\n1\n1 2"; return 0;}
30     int x, y;
31     for(int i = 1; i <= m; i++)
32     {
33         in >> x >> y;
34         a[x][0] = a[x][0] + 1; a[x][a[x][0]] = y;
35         a[y][0] = a[y][0] + 1; a[y][a[y][0]] = x;
36     }
37
38     for(int i = 1; i <= n; i++)
39     {
40         if(viz[i] == 0){
41             k = k + 1;
42             dfs(i);
43         }
44     }
45
46     int nr = k - 1;
47
48     for(int i = 1; i <= n; i++) viz[i] = 0;
49     for(int i = 1; i <= k; i++)
50     {
51         if(b[i][0] == 1)
52         {
53             n1 = n1 + 1;
54             c1[n1] = i;
55         }
56         else
57         {
58             n2 = n2 + 1; c2[n2] = i;
59             viz[i] = b[i][0];//retinem cate noduri are conexa
60         }
61     }
62
63     if(n2 == 0)
64     {//daca am numai noduri izolate
65         out << "2\n" << k - 1 << "\n";
66         for(int i = 1; i < n; i++)
67             out << i << " " << i+1 << "\n";//leg nodurile izolate intr-un lant
68     }
69
70     else
71     {
72         int max_elem;
73         if(k == 1)
74             max_elem = 0;// avem o singura componenta conexa
75         else
76         {
77             if(n1 == 0)
78                 max_elem = 1;// daca nu am noduri izolate pot folosi
79                 // cate un singur nod
80             else
81                 if(n1 > n - n1 - 2 - (n2 - 2) * 2)
82                     max_elem = 2;//daca nodurile izolate sunt mai multe decat
83                     // nodurile ramase libere dupa ce leg
84                     // componente conexe
85             else
86                 max_elem = 1;
87         }
88         out << max_elem << "\n" << k - 1 << "\n";
89
90         //legam componente conexe formate din mai multe noduri
91         for(int i = 1; i < n2; i++)
92         {
93             x = b[c2[i]][viz[c2[i]]];

```

```

94         y = b[c2[i + 1]][viz[c2[i + 1]]]; //nodurile din componente
95                                         //conexe diferite
96         out << x << " " << y << "\n";
97         nr = nr - 1;
98         viz[c2[i]] = viz[c2[i]] - 1;
99         viz[c2[i + 1]] = viz[c2[i + 1]] - 1; //nu mai luam in considerare
100                                         //nodul afisat
101     }
102
103    //legam nodurile izolate de celelalte componente conexe
104    int izo = 0; //nu avem noduri izolate adaugate
105    for(int i = 1; i <= n2 and nr > 0; i++)
106    {
107        for(int j = viz[c2[i]]; j > 0 and nr > 0; j--)
108            //luam nodurile nefolosite
109            izo = izo + 1;
110            out << b[c2[i]][j] << " " << b[c1[izo]][1] << "\n";
111            nr = nr - 1; viz[c2[i]] = viz[c2[i]] - 1;
112        }
113    }
114
115    //daca au mai ramas noduri legam nodurile izolate
116    while(nr > 0)
117    {
118        out << b[c1[izo]][1] << " " << b[c1[izo+1]][1] << "\n"; //legam nodurile
119                                         //izolate
120        nr = nr - 1;
121        izo = izo + 1;
122    }
123}
124 return 0;
125

```

Listing 1.1.3: conexidad_mihai100.cpp

```

1  /// Autor Mihai Calancea
2  #include <iostream>
3  #include <fstream>
4  #include <vector>
5  #include <queue>
6  #include <algorithm>
7
8  using namespace std;
9
10 int main()
11 {
12     ifstream cin("conexidad.in");
13     ofstream cout("conexidad.out");
14
15     int n, m; cin >> n >> m;
16
17     vector<vector<int>> g(n);
18
19     for(int i = 0; i < m; i += 1)
20     {
21         int a, b; cin >> a >> b;
22         a -= 1, b -= 1;
23         g[a].push_back(b);
24         g[b].push_back(a);
25     }
26
27     vector<int> seen(n, 0);
28
29     auto bfs = [&] (int start, int &min_node, int &max_node)
30     {
31         queue<int> Q;
32         Q.push(start);
33
34         while(not Q.empty())
35         {
36             int node = Q.front();
37             Q.pop();
38             seen[node] = 1;
39
40             min_node = min(node, min_node);

```

```

41         max_node = max(node, max_node);
42
43         for(auto v : g[node])
44             if(not seen[v])
45             {
46                 Q.push(v);
47                 seen[v] = 1;
48             }
49     }
50 }
51
52 vector<int> min_node(n, n + 1);
53 vector<int> max_node(n, - 1);
54 int components = 0, singletons = 0;
55
56 for(int i = 0; i < n; i += 1)
57     if(not seen[i])
58     {
59         bfs(i, min_node[i], max_node[i]);
60         components += 1;
61         if(min_node[i] == max_node[i])
62             singletons += 1;
63     }
64
65 auto usedInChain = [] (int nr)
66 {
67     if(nr <= 1)
68         return 0;
69     return 2 + (nr - 2) * 2;
70 };
71
72 int rem = n - singletons - usedInChain(components - singletons);
73 vector<int> used(n, 0);
74 vector<pair<int, int>> edges;
75
76 int ans = 0;
77
78 if(rem >= singletons)
79 {
80     int last_max = -1;
81     for(int i = 0; i < n; i += 1)
82         if(min_node[i] < max_node[i])
83         {
84             if(last_max >= 0)
85             {
86                 edges.push_back({min_node[i], last_max});
87                 used[min_node[i]] = used[last_max] = 1;
88             }
89             last_max = max_node[i];
90         }
91
92     for(int i = 0; i < n; i += 1)
93         if(min_node[i] == max_node[i])
94             for(int j = 0; j < n; j += 1)
95                 if(not used[j] and min_node[j] != max_node[j])
96                 {
97                     edges.push_back({i, j});
98                     used[j] = 1;
99                     break;
100                }
101 }
102 else
103 {
104     int last = -1;
105     for(int i = 0; i < n; i += 1)
106         if(min_node[i] <= max_node[i])
107         {
108             if(last >= 0)
109                 edges.push_back({i, last});
110
111             last = i;
112         }
113 }
114
115 vector<int> cnt(n, 0);
116 for(auto edge : edges) {

```

```

117         cnt[edge.first] += 1;
118         cnt[edge.second] += 1;
119     }
120
121     ans = *max_element(cnt.begin(), cnt.end());
122
123     cout << ans << "\n";
124     cout << edges.size() << "\n";
125     for(auto edge : edges)
126         cout << edge.first + 1 << " " << edge.second + 1 << "\n";
127 }
```

Listing 1.1.4: conexidad_radu100.cpp

```

1 // autor Radu Muntean
2 #include <iostream>
3 #include <vector>
4 #include <iostream>
5
6 using namespace std;
7
8 ifstream in ("conexidad.in");
9 ofstream out ("conexidad.out");
10
11 const int Q = 100;
12
13 int n, m;
14 int lst[Q + 1], val[2 * Q * Q + 1], nxt[2 * Q * Q + 1];
15
16 void add_edge(int a, int b)
17 {
18     static int cont = 0;
19     val[++cont] = b;
20     nxt[cont] = lst[a];
21     lst[a] = cont;
22 }
23
24 int viz[Q + 1];
25
26 void dfs(int nod, int dad, vector<int> &elems)
27 {
28     viz[nod] = 1;
29     elems.push_back(nod);
30
31     for (int p = lst[nod]; p; p = nxt[p])
32     {
33         if (viz[val[p]] == 1)
34             continue;
35         dfs(val[p], nod, elems);
36     }
37 }
38
39 int main()
40 {
41     in >> n >> m;
42     int a, b;
43     for (int i = 0; i < m; i++)
44     {
45         in >> a >> b;
46         add_edge(a, b);
47         add_edge(b, a);
48     }
49
50     int last = 0;
51
52     vector<pair<int, int> > rez;
53     vector<int> isolated;
54     vector<int> normals;
55
56     vector<int> actual;
57
58     for (int i = 1; i <= n; i++)
59     {
60         if (viz[i] == 1)
61             continue;
```

```

62         actual.clear();
63         dfs(i, 0, actual);
64
65         if (actual.size() == 1)
66         {
67             isolated.push_back(actual[0]);
68             continue;
69         }
70
71         if (normals.size() == 0)
72         {
73             normals = actual;
74             continue;
75         }
76
77         rez.push_back(make_pair(i, normals.back()));
78         normals.pop_back();
79         normals.insert(normals.end(), actual.begin() + 1, actual.end());
80     }
81
82     if (normals.size() >= isolated.size())
83     {
84         out << "1\n";
85         for (int i = 0; i < isolated.size(); i++)
86             rez.push_back(make_pair(isolated[i], normals[i]));
87     }
88     else
89     {
90         if (n == 2)
91             out << "1\n";
92         else
93             out << "2\n";
94
95         int last, start;
96         if (normals.size() > 0)
97         {
98             for (int i = 0; i < normals.size() - 1; i++)
99                 rez.push_back(make_pair(isolated[i], normals[i]));
100
101             last = normals.back();
102             start = normals.size() - 1;
103         }
104         else
105         {
106             last = isolated[0];
107             start = 1;
108         }
109
110         for (int i = start; i < isolated.size(); i++)
111         {
112             rez.push_back(make_pair(last, isolated[i]));
113             last = isolated[i];
114         }
115     }
116 }
117
118 out << rez.size() << "\n";
119 for(int i = 0; i < rez.size(); i++)
120     out << rez[i].first << " " << rez[i].second << "\n";
121
122 return 0;
123 }
```

1.1.3 *Rezolvare detaliată

1.2 rufe

Problema 2 - rufe

90 de puncte

Alex vrea să își usuce rufelete pe balcon. El a spălat K tricouri și o șosetă. Uscătorul lui Alex are N niveluri, iar fiecare nivel are M locuri unde poate atârna câte un singur obiect de îmbrăcăminte.

Alex usucă hainele într-un mod specific: începe prin a pune şoseta pe nivelul A , locul B , iar apoi aduce coşul de rufe cu cele K tricouri şi le aşază pe rând, mereu alegând o poziţie liberă cât mai depărtată de locul unde a pus şoseta. Metrica pe care o găseşte ca fiind cea mai potrivită când vine vorba de uscatul rufelor este *distanța Manhattan*, astfel încât distanţa de la nivelul $r1$, locul $c1$ la nivelul $r2$, locul $c2$ are valoarea expresiei $|r1 - r2| + |c1 - c2|$.

Cerinţe

Aflaţi distanţa dintre poziţia unde a atârnat ultimul tricou şi poziţia unde se usucă şoseta.

Date de intrare

Pe prima linie a fişierului de intrare **rufe.in** se vor afla 5 numere întregi N, M, A, B , şi K , cu semnificaţia din enunţ, separate prin câte un spaţiu.

Date de ieşire

În fişierul de ieşire **rufe.out** se va afla o singură linie care să conţină valoarea cerută.

Restricţii şi precizări

- $1 \leq N, M \leq 10^9$
- $1 \leq A \leq N$
- $1 \leq B \leq M$
- $1 \leq K \leq N * M - 1$
- Pentru teste în valoare de 13 puncte se garantează că $N, M \leq 10^3$.
- Pentru alte teste în valoare de 12 puncte se garantează că $N \leq 10^6$.
- Pentru alte teste în valoare de 12 puncte se garantează că $M \leq 10^6$.
- Pentru alte teste în valoare de 18 puncte se garantează că $K \leq 10^6$.
- Pentru alte teste în valoare de 7 puncte se garantează că $A = B = 1$.

Exemple

rufe.in	rufe.out	Explicaţii
5 6 3 3 4	4	Uscătorul are 5 niveluri cu câte 6 locuri pe nivel. Şoseta se pune pe nivelul 3, locul 3. Primele 2 tricouri vor fi atârnate la distanţă 5 în colţurile ușcătorului. Următoarele 2 tricouri pot fi puse numai la distanţă 4.
3476 53410 438 9217 1000000	45818	-
10000000000 10000000000 1 1 7	1999999995	-
654321 123456 5454 1212 10000000000	628395	În acest caz Alex usucă 10^{10} tricouri. Acordaţi atenţie citirii unei astfel de valori din fişier.

Timp maxim de executare/test: 1.0 secunde

Memorie: total 64 MB din care pentru stivă 32 MB

Dimensiune maximă a sursei: 20 KB

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

1.2.1 Indicaţii de rezolvare

Autor: Bogdan Ciobanu

Soluţie pentru 13 puncte (complexitate $O((m + n)^2)$)

Cel mai îndepărtat punct de la şosete este unul din cele 4 colţuri ale dreptunghiului, pe care o vom nota cu d . Vom parcurge toate romburile de distanţă $d, d - 1, d - 2, \dots$ şi numărând poziţiile valide, ne vom opri la distanţa respectivă.

Soluție pentru 45 - 55 de puncte (complexitate $O(\sqrt{k})$)

Observăm că fiecare linie este alcătuită dintr-o secvență de numere strict descrescătoare, cuprinsă între coloana 1 și coloana B , și o secvență de numere strict crescătoare cuprinsă între coloana $B + 1$ și coloana M .

Să notăm distanțele din cele 4 colțuri ale matricei $d1$ (stânga-sus), $d2$ (stânga-jos), $d3$ (dreapta-sus), $d4$ (dreapta jos). Vom avea sirurile descrescătoare $[d1 \dots a]$, $[d1 - 1 \dots a - 1]$... $[d1 - a + 1 \dots 1]$ $[d2 \dots n - a]$, $[d2 - 1 \dots n - a - 1]$... $[d2 - n - a + 1 \dots 1]$ și sirurile crescătoare $[a \dots d3]$, $[a - 1, d3 - 1]$... $[1 \dots m - b]$...

Se începe cu distanța maximă. Cât timp numărul de tricouri așezate nu depășește k se determină câte siruri încep cu valoarea d , se adună aceste valori, apoi se încearcă distanța $d - 1$.

Soluție pentru 90 de puncte (complexitate $O(\log(n + m))$)

Din punct de vedere grafic, observăm, că toate punctele la distanță k de punctul (a, b) se află așezate în formă de *romb*. Coordonatele acestor puncte sunt $(a + x, b + y)$, cu $\text{abs}(x) + \text{abs}(y) = k$.

Observăm că dacă nu luăm în calcul restricțiile dreptunghiului, atunci numărul de puncte la distanță k este egal cu $4 * k$. Deci numărul punctelor la distanță mai mică sau egală cu k va fi $4 * (1 + 2 + 3 + \dots + k) = 2 * k * (k + 1)$.

Nu toate punctele ale rombului sunt poziții valide. Dacă ținem cont de restricțiile impuse de dimensiunea dreptunghiului, vom observa, că numărul locurilor "neadmise" de o latură a dreptunghiului este *suma de numere impare*: $1 + 3 + 5 + \dots + (2p - 1) = p^2$. Reprezentând grafic, obținem forma unui triunghi dreptunghic isoscel așezat pe ipotenuză.

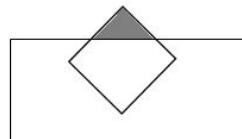


Figura 1.1: rufe

În cazul în care punctele rombului sunt limitate de două laturi ale dreptunghiului. Atunci locurile "neadmise" de cele două laturi pot avea elemente comune. Reprezentând grafic, aceste puncte comune, formează un triunghi dreptunghic așezat pe o catetă. Numărul punctelor din această zonă este de forma: $1 + 2 + 3 + \dots + q = q * (q + 1)/2$.

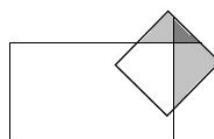


Figura 1.2: rufe

Aceste zone se identifică foarte ușor. Folosind *principiul includerii și excluderii* putem afla numărul punctelor ce se află la distanță mai mică sau egală cu k în $O(1)$ cu ajutorul formulelor de mai sus.

Întrucât problema cere numărul punctelor la distanță cât mai îndepărtată, vom folosi *căutare binară* pentru a găsi punctele definite mai sus, iar $m * n - k - 1$ este numărul punctelor la care trebuie să ne referim pentru a da răspuns problemei.

Rezolvarea problemei nu necesită folosirea tablourilor.

Soluție pentru 90 de puncte (complexitate $O(\log(n + m))$)

O altă abordare este să numărăm complementul soluției precedente, adică punctele din afară rombului, dar care se află în dreptunghi. Pentru ușurime, vom împărți dreptunghiul în 4 cadrane în funcție de poziția şoşetelor, aşa cum arată în imagine:

Legenda:

- rosu = celula şoşetelor
- albastru = rombul determinat de variabila căutată binar, k

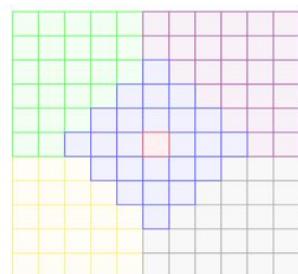


Figura 1.3: rufe

- verde = cadranul 1
- violet = cadranul 2
- gri = cadranul 3
- galben = cadranul 4

Cu puțină atenție, putem rezolva independent cadranele. În cadrul unui cadran vom număra celulele care nu fac parte din rombul albastru. Se observă că figura după ce eliminăm acele celule va fi un dreptunghi sau trapez, dar cel din urmă este doar un caz particular al ceiluilalt.

Pentru a calcula numărul de celule din trapez, îl vom împărți ca în urmatorul grafic:

Regiunile verzi și albastre vor fi două dreptunghiuri, ale căror număr de celule este ușor de aflat, iar regiunea roșie este un triunghi dreptunghic; pentru acesta trebuie să calculam suma primelor p numere naturale.

1.2.2 Cod sursă

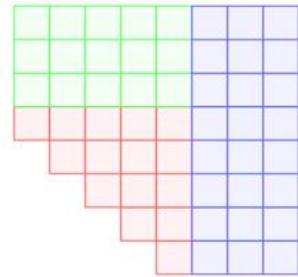


Figura 1.4: rufe

Listing 1.2.1: rufe_bc.cpp

```

1 // autor Ciobanu Bogdan
2 #include <stdio.h>
3 #include <algorithm>
4 #include <memory>
5
6 using namespace std;
7
8 long long n, m, x, y, k;
9
10 long long unwrap(long long a)
11 {
12     return a > OLL ? a : OLL;
13 }
14
15 long long sum(long long from, long long to, long long i0)
16 {
17     if (from > to) return OLL;
18     return (to - from + 2 * i0) * (to - from + 1) / 2;
19 }
20
21 long long count_corner(long long x, long long y, long long d)
22 {
23     return sum(unwrap(x + d - n), min(d, m - y), unwrap(n - x - d) + 1)
24         + unwrap(m - y - d) * (n - x + 1);
25 }
26
27 long long count_greater_equal(long long d)
28 {
29     if (d == 0) return n * m;
30
31     return count_corner(x, y, d) + count_corner(n - x + 1, m - y + 1, d)
32         + count_corner(n - x + 1, y, d) + count_corner(x, m - y + 1, d)
33         - unwrap(x - d) - unwrap(y - d) - unwrap(n - (x + d - 1))
34         - unwrap(m - (y + d - 1));
35 }
36
37 int main()
38 {
39     unique_ptr<FILE, decltype(&fclose)> f(fopen("rufe.in", "r"), &fclose);
40     fscanf(f.get(), "%lld%lld%lld%lld%lld", &n, &m, &x, &y, &k);
41
42     long long l = -1, r = n + m - 1;
43     while (r - l > 1LL)
44     {
45         long long p = (l + r) / 2;
46         ((count_greater_equal(p) < k) ? r : l) = p;
47     }
48
49     f.reset(fopen("rufe.out", "w"));
50     fprintf(f.get(), "%lld\n", l);

```

```

51     return 0;
52 }
```

Listing 1.2.2: rufe_LS_LogN.cpp

```

1 // autor Lukacs Sandor
2 #include <bits/stdc++.h>
3
4 using namespace std;
5
6 ifstream in("rufe.in");
7 ofstream out("rufe.out");
8
9 long long n, m, k, a, b;
10
11 long long pozitii_libere(long long a, long long b, long long d)
12 { //numarul de pozitii libere daca ultimul tricou s-a asezat la d
13     long long total = 2 * d * (d - 1) + 1; //numarul total de pozitii aflate la
14                                         //distantele 1, 2, ... d - 1,
15                                         //nu toate se gasesc pe uscator
16     long long poz;
17     //scadem ce este in plus in sus
18     if(d > a)
19     {
20         poz = d - a; //triunghi isoscel cu inaltimea poz
21         total = total - poz * (poz + 1) + poz; //de 2 ori suma gauss pana la poz,
22                                         //din care scadem poz, deoarece apare de 2 ori
23     }
24
25     //in jos
26     if(d > n - a + 1)
27     {
28         poz = d - (n - a + 1);
29         total = total - poz * (poz + 1) + poz;
30     }
31
32     //stanga
33     if(d > b)
34     {
35         poz = d - b;
36         total = total - poz * (poz + 1) + poz;
37     }
38
39     //dreapta
40     if(d > m - b + 1)
41     {
42         poz = d - (m - b + 1);
43         total = total - poz * (poz + 1) + poz;
44     }
45
46     //adaugam ceea ce am scazut de 2 ori
47     //stanga - sus
48     if(d > a + b - 1)
49     {
50         poz = d - (a + b - 1) - 1;
51         total = total + poz * (poz + 1) / 2;
52     }
53
54     //dreapta - sus
55     if(d > a + m - b)
56     {
57         poz = d - (a + m - b) - 1;
58         total = total + poz * (poz + 1) / 2;
59     }
60
61     //stanga - jos
62     if(d > n - a + b)
63     {
64         poz = d - (n - a + b) - 1;
65         total = total + poz * (poz + 1) / 2;
66     }
67
68     //dreapta jos
69     if(d > n - a + m - b + 1)
70     {
```

```

71         poz = d - (n - a + m - b + 1) - 1;
72         total = total + poz * (poz + 1) / 2;
73     }
74
75     return total;
76 }
77
78 int main()
79 {
80     in >> n >> m >> a >> b >> k;
81     if(k == 1)
82         out << max(max(a - 1 + b - 1, a - 1 + m - b),
83                     max(n - a + b - 1, n - a + m - b));
84     else
85         if(k >= n * m - 4)
86             out << 1;
87         else
88         {
89             long long l = 1, r = n + m, mid, sol;
90             bool stop = false;
91             while(l <= r and stop == false)
92             {
93                 mid=l+(r-l)/2;//distanta la care se aseaza ultimul tricou
94
95                 if(n * m - pozitii_libere(a, b, mid) >= k)
96                     {//daca a pus la uscat mai mult de k tricouri cand ultimul
97                     //tricou e la dist mid
98                     if(n * m - pozitii_libere(a, b, mid + 1) < k or mid == r)
99                         {//daca ultimul tricou e cu o unitate mai departe
100                         // de sosete, dar asa nu incap k tricouri
101                         sol = mid; stop = true;
102                     }
103                 else
104                     l = mid + 1;//punem tricouri la distanta mai mare
105                 }
106             else
107                 r = mid - 1;//tricouri mai aproape de sosete
108             }
109
110             out << sol;
111         }
112
113     return 0;
114 }
```

Listing 1.2.3: rufe_LS_NplusM.cpp

```

1  /// autor Lukacs Sandor
2  #include <bits/stdc++.h>
3
4  using namespace std;
5
6  ifstream in("rufe.in");
7  ofstream out("rufe.out");
8
9  long long n, m, k, a, b;
10
11 long long pozitii_libere(long long a, long long b, long long d)
12 { //numarul de pozitii libere daca ultimul tricou s-a asezat la d
13     long long total = 2 * d * (d - 1) + 1; //numarul total de pozitii aflate la
14     //distantele 1, 2, ... d - 1, nu toate se gasesc pe uscator
15     long long poz;
16     //scadem ce este in plus in sus
17     if(d > a) {
18         poz = d - a; //triunghi isoscel cu inaltimea poz
19         total = total - poz * (poz + 1) + poz; //de 2 ori suma gauss pana la poz,
20                                         //din care scadem poz, deoarece apare de 2 ori
21     }
22
23     //in jos
24     if(d > n - a + 1)
25     {
26         poz = d - (n - a + 1);
27         total = total - poz * (poz + 1) + poz;
28     }

```

```

29
30     //stanga
31     if(d > b)
32     {
33         poz = d - b;
34         total = total - poz * (poz + 1) + poz;
35     }
36
37     //dreapta
38     if(d > m - b + 1)
39     {
40         poz = d - (m - b + 1);
41         total = total - poz * (poz + 1) + poz;
42     }
43
44     //adaugam ceea ce am scazut de 2 ori
45     //stanga - sus
46     if(d > a + b - 1)
47     {
48         poz = d - (a + b - 1) - 1;
49         total = total + poz * (poz + 1) / 2;
50     }
51
52     //dreapta - sus
53     if(d > a + m - b)
54     {
55         poz = d - (a + m - b) - 1;
56         total = total + poz * (poz + 1) / 2;
57     }
58
59     //stanga - jos
60     if(d > n - a + b)
61     {
62         poz = d - (n - a + b) - 1;
63         total = total + poz * (poz + 1) / 2;
64     }
65
66     //dreapta jos
67     if(d > n - a + m - b + 1)
68     {
69         poz = d - (n - a + m - b + 1) - 1;
70         total = total + poz * (poz + 1) / 2;
71     }
72
73     return total;
74 }
75
76 int main()
77 {
78     in >> n >> m >> a >> b >> k;
79     long long d = max(max(a - 1 + b - 1, a - 1 + m - b),
80                         max(n - a + b - 1, n - a + m - b));
81     while(k > n * m - pozitii_libere(a, b, d))
82         d = d - 1;
83
84     out << d;
85     return 0;
86 }
```

Listing 1.2.4: rufe_PA_log.cpp

```

1  /// Autor Adrian Panaete
2  #include <bits/stdc++.h>
3
4  using namespace std;
5
6  ifstream f("rufe.in");
7  ofstream g("rufe.out");
8
9  int64_t lo,hi,mi,r,R,a[4],b[4];
10
11 void read(),solve();
12
13 int64_t drep(int64_t A,int64_t B,int64_t D)
14 {
```

```

15     return D<=A ?
16             D*(D+1)/2:D<=B?A*(A+1)/2+(D-A)*A :
17             D>=A+B?A*B:A*B-(A+B-D)*(A+B-D-1)/2;
18 }
19
20 int main()
21 {
22     read();solve();
23     return 0;
24 }
25 void read()
26 {
27     int64_t n,m,i,j,k;
28     f>>n>>m>>i>>j>>k;
29     hi=n+m+1,R=m*n-k-1;
30     a[0]=i;
31     b[0]=j-1;
32     a[1]=n-i;
33     b[1]=j;
34     a[2]=i-1;
35     b[2]=m-j+1;
36     a[3]=n-i+1;
37     b[3]=m-j;
38     for(int p=0;p<4;p++)
39         if(a[p]>b[p])
40             swap(a[p],b[p]);
41 }
42
43 void solve()
44 {
45     while(hi-lo>1)
46     {
47         mi=(lo+hi)/2,r=R;
48         for(int p=0;p<4;p++)
49             r-=drep(a[p],b[p],mi);
50         if(r>=0)
51             lo=mi;
52         else
53             hi=mi;
54     }
55     g<<hi;
56 }
```

Listing 1.2.5: rufe_SzZ_brut.cpp

```

1 // Autor Szabo Zoltan
2 #include <bits/stdc++.h>
3
4 using namespace std;
5
6 ifstream fin("rufe.in");
7 ofstream fout("rufe.out");
8
9 long long distanta(long long n, long long m,
10                     long long a, long long b, long long k)
11 {
12     if (m*n==k) return 0;
13     long long d,sus,st,jos,dr,l1,l2;
14     sus=a-1;
15     st=b-1;
16     jos=n-a;
17     dr=m-b;
18     long long contor=0;
19     long long lmax=max(sus+st,
20                         max(st+jos, max(jos+dr, dr+sus)));
21     for (d=lmax; d>0; d--)
22         for (l1=d; l1>0; l1--)
23         {
24             l2=d-l1;
25             if (a+l1<=n and b+l2<=m)
26             {
27                 contor++;
28                 if (contor==k)
29                     return d;
30             }
31         }
32 }
```

```

31             if (b+11<=m and a-12>0)
32             {
33                 contor++;
34                 if (contor==k)
35                     return d;
36             }
37         }
38     }
39     if (a-11>0 and b-12>0)
40     {
41         contor++;
42         if (contor==k)
43             return d;
44     }
45     if (b-11>0 and a+12<=n)
46     {
47         contor++;
48         if (contor==k)
49             return d;
50     }
51 }
52 }
53 }
54 int main()
55 {
56     long long n,m,t,a,b,k;
57     fin>>n>>m>>a>>b>>k;
58     fout<<distanta(n,m,a,b,k)<<"\n";
59     return 0;
60 }
```

Listing 1.2.6: rufe_SzZ_log.cpp

```

1 // autor Szabo Zoltan
2 #include <bits/stdc++.h>
3
4 using namespace std;
5
6 ifstream fin("rufe.in");
7 ofstream fout("rufe.out");
8
9 long long distanta(long long n, long long m,
10                     long long a, long long b, long long k)
11 {
12     if (m*n==k) return 0;
13     long long v;
14     long long stanga;
15     long long dreapta;
16     long long mij,d,sus,st,jos,dr,k1;
17     sus=a-1;
18     st=b-1;
19     jos=n-a;
20     dr=m-b;
21     stanga=1;      // limita stanga pt cautare binara
22     dreapta=max(sus+st,
23                  max(st+jos,
24                      max(jos+dr, dr+sus))); // limita dreapta pt cautare binara
25
26     while (stanga<=dreapta)
27     {
28         mij=(stanga+dreapta)/2;
29         d=mij-1;
30         k1= 2*(d+1)*d; // calculez numarul tricourilor la distanta <=d
31
32         // metoda includerii si excluderii
33         if (d>sus)
34             {v=d-sus;k1=k1-v*v;} // elimin locurile de deasupra dreptunghiului
35
36         if (d>st)
37             {v=d-st;k1=k1-v*v;} // elimin locurile din stanga dreptunghiului
38
39         if (d>jos)
40             {v=d-jos;k1=k1-v*v;} // elimin locurile de sub dreptunghi
```

```

42     if (d>dr)
43     {v=d-dr;k1=k1-v*v;}; // elimin locurile din dreapta dreptunghiului
44
45     if (d>st+sus+1)
46     {v=d-st-sus-1;k1=k1+v*(v+1)/2;}//adaug locurile eliminate de doua ori
47
48     if (d>st+jos+1)
49     {v=d-st-jos-1;k1=k1+v*(v+1)/2;}
50
51     if (d>dr+jos+1)
52     {v=d-dr-jos-1;k1=k1+v*(v+1)/2;}
53
54     if (d>dr+sus+1)
55     {v=d-dr-sus-1;k1=k1+v*(v+1)/2;}
56
57     k1=m*n-k1-1; // transform k1 in numarul tricourilor la distanta >d
58     if (k1<k)
59         dreapta=mij-1;
60     else
61         stanga=mij+1;
62     }
63     return dreapta;
64 }
65
66 int main()
67 {
68     long long n,m,a,b,k;
69     fin>>n>>m>>a>>b>>k;
70     fout<<distanta(n,m,a,b,k)<<"\n";
71     return 0;
72 }
```

1.2.3 *Rezolvare detaliată

1.3 tairoș

Problema 3 - tairoș

90 de puncte

Se dă un arbore cu N noduri, numerotate de la 1 la N .

Arboarele se va transforma astfel: la oricare etapă fiecare nod de gradul 1 diferit de rădăcină din arborele actual se înlocuiește cu un arbore identic cu cel dat inițial, iar la următoarea etapă procedeul se va relua pentru arborele obținut, formându-se astfel un arbore infinit. În următoarele 3 imagini se prezintă un exemplu de arbore dat inițial, arborele obținut după prima etapă de prelungire a frunzelor și arborele obținut după 2 etape de prelungire a frunzelor.

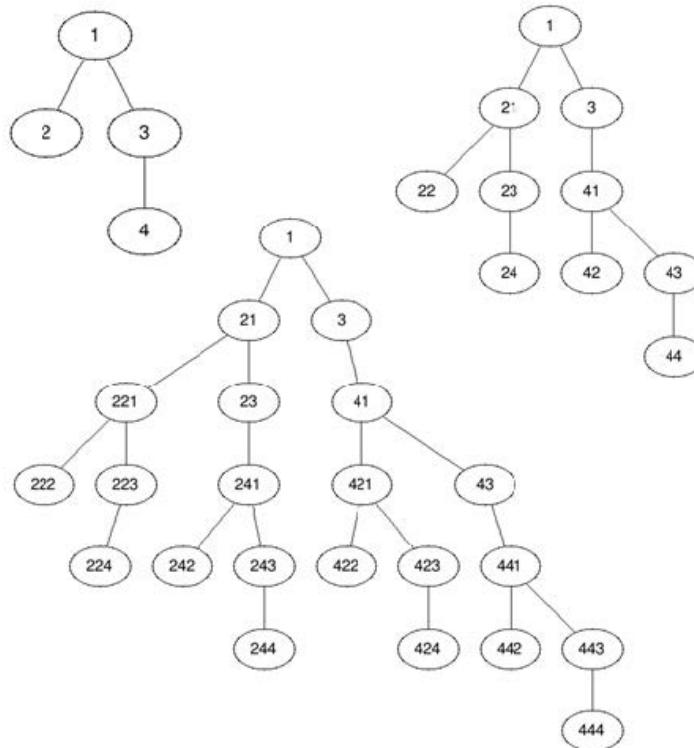


Figura 1.5: tairoso

Cerințe

Să se determine câte noduri se află la distanță D de rădăcina arborelui infinit.

Date de intrare

Pe prima linie a fișierului de intrare **tairoso.in** se va afla un număr natural N , reprezentând numărul de noduri din arborele dat inițial. Pe a doua linie se va afla numărul întreg D , cu semnificația de mai sus, iar fiecare dintre următoarele $N - 1$ linii conține câte 2 numere întregi x și y cu semnificația că în arborele dat inițial există muchia $[x, y]$.

Date de ieșire

Fișierul de ieșire **tairoso.out** va conține un singur număr, și anume restul împărțirii numărului de noduri cerut la numărul 1.000.000.007.

Restricții și precizări

- $2 \leq N \leq 100$
- $1 \leq D \leq 10.000$
- Un arbore este un graf neorientat, conex și fără cicluri.
- Distanța dintre două noduri x și y ale unui arbore este egală cu numărul de muchii ale unui lanț cu extremitățile în nodurile x și y , lanț format din noduri distincte.
 - Rădăcina va fi considerată ca fiind nodul 14;
 - Pentru teste în valoare de 17 puncte avem $N = 3$
 - Pentru teste în valoare de alte 22 puncte răspunsul este ≤ 10000 ;

Exemple

tairoso.in	tairoso.out	Explicații
4		Arborele dat în fișierul de intrare are 4 noduri. Se cere numărul nodurilor aflate la distanță 3 față de rădăcină.
3		
1 2		Urmărind imaginile din exemplele de mai sus, la distanța 3 avem
3 1		următoarele 5 noduri: 222, 223, 241, 421 și 43
3 4		

5 3 1 2 3 1 3 5 4 3	8	-
5 25 2 1 2 3 1 4 5 2	33554432	-

Timp maxim de executare/test: 1.0 secunde

Memorie: total 64 MB din care pentru stivă 32 MB

Dimensiune maximă a sursei: 20 KB

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

1.3.1 Indicații de rezolvare

Autor: Radu Muntean

Subtask-ul cu $N = 3$ în valoare de 17 puncte

Pentru $N = 3$ structura arborelui initial poate fi:

- Avem cele 2 muchii legate de nodul 1 (muchiile 1 – 2 și 1 – 3). Numărul de noduri la distanța D de rădăcina va fi întotdeauna $2D$, din modul de exapandare al arborilor.
- Avem cele 2 muchii legate în lanț: 1 – 2 și 2 – 3 (ori, similar 1 – 3 și 3 – 2). Numărul de noduri la distanța D de rădăcină va fi întotdeauna 1, deoarece permanent lanțul doar se va lungi în jos.

Simularea efectiva a extinderii arborelui, soluție parțială în valoare de 26–30 puncte

O soluție de punctaj parțial ar fi să facem un *DFS*, unde să reținem permanent adâncimea la care se află nodul procesat. Astfel, atunci când suntem într-o frunză la adâncimea H , să continuăm *DFS*-ul cu nodul 1 și adâncimea H . Prin această procedură simulăm prelungirea arborilor doar atunci când avem nevoie. Vom da return din *DFS* doar atunci cand adâncimea actuală este egală cu adâncimea din cerință D , moment în care vom și incrementa rezultatul final.

Putem observa că numărul total de noduri parcuse de această abordare este $T[1] + T[2] + T[3] + \dots + T[D]$, unde $T[i]$ reprezintă numărul de noduri la distanța i de rădăcina 1. (practic, răspunsul la cerință ar fi $T[D]$). Grosolan putem aproxima complexitatea la $O(raspuns^2)$ și de aceasta soluția va obține punctajul doar pentru subtask-ul "raspuns ≤ 10000" și puțin din subtask-ul $N = 3$, mai exact cazul *lanț*.

Soluția de 90 de puncte

Soluția problemei constă în numărarea frunzelor în arborele initial.

Fie $FR[i]$ = numărul de frunze la distanța i de rădăcina 1 în arborele inițial

Fie $NT[i]$ = numărul de noduri la distanța i de rădăcina 1 în arborele inițial care nu sunt frunze.

Știm că arborele inițial are doar 100 de noduri, deci acești 2 vectori de mai sus FR și NT nu vor fi deloc mari.

În continuare vom defini $V[i]$ ca fiind numărul de noduri la distanța i care au fost cândva frunze în procesul de extindere al arborelui inițial.

Initial $V[0] = 1$ și $V[i] = 0$ oricare $i \neq 0$ (caz particular de pornire. Dacă extindem un nod izolat folosind tehnica din enunț vom obține exact arborele inițial).

Putem parcurge de la 0 la D vectorul V pentru a genera frunze la dreapta în vector folosind *tehnica programării dinamice*. Mai exact, o *recurență înainte* de forma:

find la poziția i și având deja calculată valoarea $V[i]$

$$\begin{aligned}
 V[i+1] &= V[i] * FR[1] \\
 V[i+2] &= V[i] * FR[2] \\
 \dots \\
 V[i + adancime_arbore_initial] &= V[i] * FR[adancime_arbore_initial]
 \end{aligned}$$

În cuvinte, dacă la distanța i avem $V[i]$ noduri și știm că în arborele inițial aveam $FR[k]$ frunze la distanța k de rădăcina 1, atunci prelungind toate cele $V[i]$ noduri aflate la distanța i , vom obține $V[i] * FR[k]$ noi frunze la distanța $V[i+k]$.

Acum, pentru a afla exact câte noduri sunt la distanța D de rădăcina 1, trebuie să nu uităm de nodurile din arborele inițial care nu sunt frunze, cele reținute în vectorul NT .

Răspunsul = $V[D] + V[D-1]*NT[1] + V[D-2]*NT[2] + \dots + V[D-a_a_i]*NT[a_a_i]$, unde a_a_i este *adâncimea arborelui* inițial.

Clarificare pentru formula de mai sus:

$V[D]$ reprezintă numărul de noduri foste frunze aflate la distanța D de rădăcina (acestea clar trebuie numărate).

$V[D-1]*NT[1]$ reprezintă nodurile-funze aflate la distanța $D-1$ de rădăcina, pe care le prelungim pentru ultima dată. Pentru fiecare arbore astfel prelungit trebuie să numaram câte noduri nefrunze (deoarece frunzele au fost deja numărate) are la distanța 1 - informație retinută în $NT[1]$.

$V[D-k]*NT[k]$ exact ca mai sus.

Complexitate timp finală $O(MAX_D * MAX_N)$, adică $D * adancime_arbore_initial$. Aveți grijă permanent să nu uitați de *modulo*.

1.3.2 Cod sursă

Listing 1.3.1: tairo_s_bogdan.cpp

```

1 //autor Ciobanu Bogdan
2 #include <stdio.h>
3 #include <algorithm>
4 #include <numeric>
5 #include <memory>
6
7 using namespace std;
8
9 const int MAX_N = 100, MAX_D = 10000;
10 const int MOD = int(1e9) + 7;
11
12 bool A[MAX_N][MAX_N];
13 int p[MAX_N], d[MAX_N];
14 int dp[2][MAX_D + 1];
15 int N;
16
17 bool is_leaf(int node)
18 {
19     if (node == 0) return false;
20     bool has_adj = false;
21     for (int i = 0; i < N; ++i) if (A[node][i])
22     {
23         if (has_adj) return false;
24         has_adj = true;
25     }
26     return true;
27 }
28
29 int main()
30 {
31     unique_ptr<FILE, decltype(&fclose)> f(fopen("tairo_s.in", "r"), fclose);
32     int D;
33     fscanf(f.get(), "%d%d", &N, &D);
34     for (int _ = 0; _ < N - 1; ++_)
35     {
36         int x, y;
37         fscanf(f.get(), "%d%d", &x, &y);
38         --x;

```

```

39         --y;
40         A[x][y] = A[y][x] = true;
41     }
42
43     fill(p + 1, p + N, -1);
44
45     while (true)
46     {
47         bool changed = false;
48         for (int i = 0; i < N; ++i) if (p[i] != -1)
49             for (int j = 0; j < N; ++j) if (A[i][j] && p[j] == -1)
50             {
51                 p[j] = i;
52                 d[j] = d[i] + 1;
53                 changed = true;
54             }
55
56         if (!changed) break;
57     }
58
59     dp[0][0] = 1;
60     for (int s = 0; s <= D; ++s)
61     {
62         for (int i = 0; i < N; ++i) if (s + d[i] <= D)
63         {
64             (dp[!is_leaf(i)][s + d[i]] += dp[0][s]) %= MOD;
65         }
66     }
67
68     f.reset(fopen("tairos.out", "w"));
69     fprintf(f.get(), "%d\n", dp[1][D]);
70     return 0;
71 }
```

Listing 1.3.2: tairos_Lukacs.cpp

```

1 // Autor Lukacs Sandor
2 #include <bits/stdc++.h>
3
4 #define M 10000000007
5
6 using namespace std;
7
8 ifstream in("tairos.in");
9 ofstream out("tairos.out");
10
11 int n, d;
12 int a[101][101];//listele de adiacenta
13 int viz[101];//viz[i] = 1, daca nodul i a fost vizitat, 0 in caz contrar
14 int h;//inaltimea arborelui initial
15
16 long long f[100000], nf[100000];//f[i] - cate frunze exista la distanta i,
17 //nf[i] - cate nefrunze exista la distanta i
18
19 int dmin=101;//nivelul minim pe care se gaseste o frunza in arborele initial.
20
21 long long sol[100002], solf[100002], solnf[100002];
22 //sol[i] - cate noduri sunt la distanta i,
23 //solf[i] - cate frunze sunt la distanta i,
24 //solnf[i] - cate nefrunze sunt la dist i
25
26 void dfs(int k, int niv)
27 {
28     viz[k] = 1;//marchez ca vizitat
29     if(h < niv) h = niv;//actualizez inaltimea arborelui
30     f[niv] = f[niv] + 1;//consider ca nodul k e frunza
31     bool ok = false;
32     for(int i = 1; i <= a[k][0]; i++)
33     {
34         if(viz[a[k][i]] == 0)
35             {//daca exista noduri nevizitate
36                 ok = true;//retin ca nu e frunza
37                 dfs(a[k][i], niv + 1);
38             }
39     }
40 }
```

```

40
41     if(ok == true)
42     { //daca a avut descendenti
43         f[niv] = f[niv] - 1; //nu e frunza
44         nf[niv] = nf[niv] + 1; //il trecem la nefrunze
45     }
46     else //daca e frunza
47         if(niv < dmin)
48             dmin = niv; //actualizez nivelul minim
49
50 }
51
52 int main()
53 {
54     in >> n >> d;
55     for(int i = 1; i < n; i++)
56     {
57         int x, y;
58         in >> x >> y;
59         a[x][0] = a[x][0] + 1;
60         a[x][a[x][0]] = y; //adaug muchia (x, y)
61
62         a[y][0] = a[y][0] + 1;
63         a[y][a[y][0]] = x; //adaug muchia (y, x)
64     }
65
66     dfs(1, 0);
67
68     if(dmin == h and f[h] == 1 and nf[h] == 0)
69     { //daca avem un arbore lant
70         out << 1;
71         return 0;
72     }
73
74     if(h == 1)
75     { //daca avem un arbore stea fr^d
76         long long ans = 1;
77         for(int i = 1; i <= d; i++)
78             ans = (ans * f[h]) % M;
79         out << ans;
80         return 0;
81     }
82
83     if(dmin == h)
84     { //daca toate frunzele sunt la acelasi nivel
85         for(int i = 0; i <= h; i++)
86             sol[i] = f[i] + nf[i];
87         long long p = f[h];
88         for(int i = h + 1; i <= d; i++)
89             sol[i] = (p * sol[i - h]) % M;
90
91         out << sol[d];
92         return 0;
93     }
94
95
96     for(int i = 0; i <= h; i++) {
97         solf[i] = f[i]; solnf[i] = nf[i];
98     }
99
100    for(int i = dmin; i <= d; i++)
101    { //construiesc arborele cu radacina in frunzele de la distanta i
102        for(int j = 1; j <= h; j++)
103        {
104            solf[i + j] = (solf[i + j] + (solf[i] * f[j]) % M) % M; //daca
105                //in arborele initial aveam frunze la distanta j in nou
106                //arbore voi avea frunze la dist i + j
107            solnf[i+j]=(solnf[i+j]+(solf[i]*nf[j])) % M; //nefrunze
108        }
109
110        solnf[i] = (solnf[i] + solf[i]) % M;
111        solf[i] = 0;
112    }
113
114    out << (solf[d] + solnf[d]) % M;
115

```

```
116     return 0;
117 }
```

Listing 1.3.3: tairosmihai.cpp

```

1 // Autor Mihai Calancea
2 #include <iostream>
3 #include <fstream>
4 #include <vector>
5
6 const int MOD = 1e9 + 7;
7
8 using namespace std;
9
10 vector<int> mem;
11 vector<vector<int>> g;
12
13 void dfs(int, int, vector<int>&, int&, int);
14 int solve(int);
15
16 void dfs(int node, int target, vector<int> &seen, int &ans, int depth)
17 {
18     seen[node] = 1;
19     if(depth == target)
20     {
21         ans = (ans + 1) % MOD;
22         return;
23     }
24
25     bool any = false;
26
27     for(auto v : g[node])
28     {
29         if(seen[v])
30             continue;
31         any = true;
32         dfs(v, target, seen, ans, depth + 1);
33     }
34
35     if(not any)
36     {
37         ans += solve(target - depth);
38         ans %= MOD;
39     }
40 }
41
42 int solve(int d)
43 {
44     if(mem[d] >= 0)
45         return mem[d];
46     int ans = 0;
47     vector<int> seen(g.size(), 0);
48     dfs(0, d, seen, ans, 0);
49     return (mem[d] = ans);
50 }
51
52 int main()
53 {
54     ifstream cin("tairosmihai.in");
55     ofstream cout("tairosmihai.out");
56
57     int n; cin >> n;
58     int d; cin >> d;
59
60     g = vector<vector<int>> (n);
61
62     for(int i = 0; i < n - 1; i += 1)
63     {
64         int a, b; cin >> a >> b;
65         a -= 1, b -= 1;
66         g[a].push_back(b);
67         g[b].push_back(a);
68     }
69
70     mem = vector<int> (d + 1, -1);

```

```

71     cout << solve(d) << "\n";
72 }

```

Listing 1.3.4: tairosl_radu.cpp

```

1  /// autor Radu Muntean
2  #include <iostream>
3  #include <assert.h>
4
5  using namespace std;
6
7  ifstream in ("tairosl.in");
8  ofstream out ("tairosl.out");
9
10 const int N_MAX = 100, D_MAX = 10000, R = 1000000007;
11
12 int N, D;
13 int fst[N_MAX + 1], val[2 * N_MAX + 1], nxt[2 * N_MAX + 1];
14 int leafs[N_MAX + 1];
15 int not_leafs[N_MAX + 1];
16
17 // Vom folosi vectorul sol, unde sol[i] va reprezenta numarul de fuste frunze
18 //aflate la distanta i de radacina modulo R.
19
20 long long sol[N_MAX * D_MAX + 1];
21
22 void add_edge(int x, int y)
23 {
24 // "Static int" spune ca variabila 'contor' va fi initializata doar la primul
25 // apel al functie 'add_edge()', iar, in rest, se va prelua valoarea
26 //lui 'contor' lasata de apelul precedent al functiei noastre 'add_edge()' .
27     static int contor = 0;
28     val[++contor] = y;
29     nxt[contor] = fst[x];
30     fst[x] = contor;
31 }
32
33 int no_nodes_visited = 0;
34
35 void dfs(int node, int dad, int depth)
36 {
37     no_nodes_visited++;
38     bool is_leaf = true;
39     for (int p = fst[node]; p != 0; p = nxt[p])
40     {
41         if (val[p] == dad)
42             continue;
43
44         is_leaf = false;
45         dfs(val[p], node, depth + 1);
46     }
47
48     if (is_leaf)
49         leafs[depth]++;
50     else
51         not_leafs[depth]++;
52 }
53
54 int main()
55 {
56     int a, b;
57     in >> N >> D;
58     assert (N > 1 && N <= 100);
59     assert (D > 0 && D <= 10000);
60
61     for (int i = 0; i < N - 1; i++)
62     {
63         in >> a >> b;
64         assert (a > 0 && a <= N);
65         assert (b > 0 && b <= N);
66         add_edge(a, b);
67         add_edge(b, a);
68     }
69     // Facem un DFS ca sa gasim numarul de frunze de la

```

```

70     // fiecare nivel de adancime in arbore.
71     dfs(1, 0, 0);
72     assert (no_nodes_visited == N);
73
74     // Avand o singura radacina, sol[0] = 1;
75     sol[0] = 1;
76     long long solution = 0;
77
78     for (int depth = 0; depth <= D; depth++)
79     {
80         sol[depth] %= R;
81         if (D - depth < N)
82         {
83             solution += sol[depth] * not_leafs[D - depth];
84             solution %= R;
85         }
86
87         for (int forward = 1; forward <= N; forward++)
88             sol[depth + forward] += sol[depth] * leafs[forward];
89     }
90
91     out << solution;
92     return 0;
93 }
```

Listing 1.3.5: tairoz_radu_brut.cpp

```

1  /// Autor Radu Muntean
2  #include <iostream>
3  #include <assert.h>
4
5  using namespace std;
6
7  ifstream in ("tairoz.in");
8  ofstream out ("tairoz.out");
9
10 const int N_MAX = 100, D_MAX = 10000, R = 1000000007;
11
12 int N, D;
13 int fst[N_MAX + 1], val[2 * N_MAX + 1], nxt[2 * N_MAX + 1];
14
15 void add_edge(int x, int y)
16 {
17     // "Static int" spune ca variabila 'contor' va fi initializata doar la
18     // primul apel al functie 'add_edge', iar, in rest, se va prelua valoarea
19     // lui 'contor' lasat de apelul precedent al functiei noastre 'add_edge'.
20     static int contor = 0;
21     val[++contor] = y;
22     nxt[contor] = fst[x];
23     fst[x] = contor;
24 }
25
26 int solution = 0;
27 void dfs(int node, int dad, int depth)
28 {
29     if (depth == D)
30     {
31         solution++;
32         return;
33     }
34
35     bool is_leaf = true;
36     for (int p = fst[node]; p != 0; p = nxt[p])
37     {
38         if (val[p] == dad)
39             continue;
40
41         is_leaf = false;
42         dfs(val[p], node, depth + 1);
43     }
44     if (is_leaf)
45         dfs(1, 0, depth);
46 }
47
48 int main()
```

```

49 {
50     int a, b;
51     in >> N >> D;
52     assert (N > 1 && N <= 100);
53     assert (D > 0 && D <= 10000);
54
55     for (int i = 0; i < N - 1; i++)
56     {
57         in >> a >> b;
58         assert (a > 0 && a <= N);
59         assert (b > 0 && b <= N);
60         add_edge(a, b);
61         add_edge(b, a);
62     }
63     // Facem un DFS ca sa gasim numarul de frunze de la fiecare nivel
64     // de adancime in arbore.
65     dfs(1, 0, 0);
66
67     out << solution % R;
68     return 0;
69 }
```

Listing 1.3.6: tairoz_zoli.cpp

```

1  /// Autor Szabo Zoltan
2  #include <iostream>
3  #include <fstream>
4
5  using namespace std;
6
7  ifstream fin("tairoz.in");
8  ofstream fout("tairoz.out");
9
10 const long long R=1000000007;
11
12 int N, D, mat[105][105], tata[105], lungime[105],sterile[105],
13     frunze[105],nrfrunze[100],distanta[105], nfr, maxadanc;
14 bool viz[105], steril[105];    // un nod este steril daca nu e frunza
15 long long nrsol[10005];
16
17 void dfs(int nod, int adancime, int &maxadanc)
18 {
19     if (maxadanc<adancime)
20         maxadanc=adancime; // calculam adancimea maxima al arborelui
21     lungime[nod]=adancime;
22     for (int i=1;i<=mat[nod][0];i++)
23         if(!viz[mat[nod][i]])           // daca un nod nu a fost vizitat,
24                                         // atunci este fiul nodului curent
25         {
26             steril[nod]=true;          // nodul curent nu poate fi frunza
27                                         // daca are un descendant
28             viz[mat[nod][i]]=true;
29             tata[mat[nod][i]]=nod;
30             dfs(mat[nod][i], adancime+1,maxadanc);
31         }
32     }
33
34 void citire(int &N, int &D, int tata[105])
35 {
36     int x,y;
37     fin>>N>>D;
38     for(int i=1; i<=N-1; i++)
39     {
40         fin>>x>>y;
41         mat[x][0]++;mat[x][mat[x][0]]=y;           // lista de adiacenta
42         mat[y][0]++;mat[y][mat[y][0]]=x;
43     }
44     viz[1]=true;           // se stie ca nodul 1 este radacina
45     dfs(1,0,maxadanc);   // dfs porneste cu nivel 0, radacina 1
46                                         // si va calcula adancimea maxima a arborelui
47 }
48
49 int main()
50 {
51     citire(N,D,tata);
```

```

52     for(int i=1;i<=N; i++)
53         if (steril[i])
54             sterile[lungime[i]]++; // creste numarul nodurilor sterile de o
55                                         // anumita distanta
56         else
57             frunze[lungime[i]]++; // creste numarul frunzelor de o
58                                         // anumita distanta
59
60 nfr=0;
61 for(int i=1;i<=N;i++)
62     if(frunze[i]) // daca exista frunze la distanta i de radacina
63     {
64         nfr++;
65         nrfrunze[nfr]=frunze[i]; // numarul frunzelor la
66         distanta[nfr]=i; // distanta data
67     }
68
69 nrsol[0]=1;
70 for (int i=1;i<=maxadanc;i++)
71     for (int j=1; j<=nfr; j++)
72         if (distanta[j]<=i)
73             nrsol[i]=(nrsol[i]+(nrsol[i-distanta[j]]*nrfrunze[j])%R)%R;
74                                         // construiesc valorile initiale pana la maxadanc
75
76 for (int i=maxadanc+1; i<=D; i++)
77     for (int j=1; j<=nfr; j++)
78         nrsol[i]=(nrsol[i]+(nrsol[i-distanta[j]]*nrfrunze[j])%R)%R;
79                                         // construiesc valorile initiale pana la maxadanc
80
81 for (int i=1;i<=N;i++)
82 {
83     if (D - i >= 0)
84         nrsol[D]=(nrsol[D]+(nrsol[D-i]*sterile[i])%R)%R;
85                                         // copletam cu nodurile sterile
86 }
87
88 fout<<nrsol[D];
89 return 0;
90 }
```

1.3.3 *Rezolvare detaliată

Capitolul 2

OJI 2018

2.1 galeti

Problema 1 - galeti

90 de puncte

Avem n găleți, numerotate de la stânga la dreapta cu numere de la 1 la n . Fiecare găleată conține inițial 1 litru de apă. Capacitatea fiecărei găleți este nelimitată. Vărsăm gălețile una în alta, respectând o anumită regulă, până când toată apa ajunge în prima găleată din stânga. Vărsarea unei găleți presupune un anumit efort.

Regula după care se răstoarnă gălețile este următoarea: se aleg două găleți astfel încât orice găleată situată între ele să fie goală. Se varsă apa din găleata din dreapta în găleata din stânga. Efortul depus este egal cu volumul de apă din găleata din dreapta (cea care se varsă).

Formal, dacă notăm a_i volumul de apă conținut în găleata cu numărul i , regula de vărsare a acestei găleți în găleata cu numărul j poate fi descrisă astfel:

1. $j < i$
2. $a_k = 0$ pentru orice k astfel încât $j < k < i$
3. efortul depus este a_i
4. după vărsare $a_j = a_j + a_i$ și $a_i = 0$

Cerințe

Cunoscând numărul de găleți n și un număr natural e , să se determine o succesiune de vărsări în urma căreia toată apa ajunge în găleata cea mai din stânga și efortul total depus este exact e .

Date de intrare

Fișierul de intrare **galeti.in** conține pe prima linie două numere naturale, n și e , în această ordine, separate prin spațiu. Primul număr n reprezintă numărul de găleți. Al doilea număr e reprezintă efortul care trebuie depus pentru a vărsa toată apa în găleata din stânga.

Date de ieșire

Fișierul de ieșire **galeti.out** trebuie să conțină $n - 1$ linii care descriu vărsările, în ordinea în care acestea se efectuează, pentru a vărsa toată apa în găleata din stânga cu efortul total e . Fiecare dintre aceste linii trebuie să conțină două numere i și j , separate prin spațiu, cu semnificația că apa din găleata cu numărul i se varsă în găleata cu numărul j .

Restricții și precizări

- $1 \leq n \leq 100.000$
- $1 \leq e \leq 5.000.000.000$
- Se asigură că pentru datele de test există cel puțin o soluție posibilă,
- Dacă există mai multe soluții se poate afișa oricare dintre acestea.
- Punctajul maxim al problemei este de 100 de puncte dintre care 10 puncte din oficiu.
- Pentru teste în valoare de 18 puncte datele de intrare sunt cunoscute. Mai precis:
 - Testul 0 : $n = 91, e = 90$
 - Testul 1 : $n = 30, e = 435$
 - Testul 2 : $n = 7, e = 16$
- Pentru alte teste în valoare de 15 puncte $n \leq 9$.

Exemple

galeti.in	galeti.out	Explicații
4 4	2 1 4 3 3 1	<p>Inițial fiecare găleată conține câte un litru de apă.</p> <p>1 1 1 1.</p> <p>Prima dată vărsăm un litru de apă din găleata 2 în găleata 1 cu efort 1 => 2 0 1 1.</p> <p>Apoi vărsăm un litru de apă din găleata 4 în găleata 3 cu efort 1 => 2 0 2 0.</p> <p>În final vărsăm cei doi litri de apă din găleata 3 în găleata 1 cu efort 2 => 4 0 0 0</p> <p>O altă variantă corectă ar fi fost:</p> <p>4 3 2 1 3 1</p> <p>Observați că următoarea succesiune de vărsări este greșită:</p> <p>4 2 2 1 3 1</p> <p>Deși efortul depus este 4 și cei 4 litri ajung în prima găleată, la primul pas vărsarea unui litru de apă din găleata 4 în găleata 2 nu este permisă deoarece între acestea se găsește găleata 3 care conține apă.</p>

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

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

Dimensiune maximă a sursei: **20 KB**

2.1.1 Indicații de rezolvare

prof. Adrian Panaete - Colegiul Național "A. T. Laurian" Botosani

Vom demonstra prin inducție că efortul depus pentru a vărsa toată apa în prima găleată din cele n este o valoare E_n cu proprietatea că $n - 1 \leq E_n \leq \frac{n(n-1)}{2}$.

Fie e_i efortul depus când vărsăm găleata cu numărul i . În momentul vărsării găleata coține cel puțin apa conținută inițial (1 litru) și cel mult toată apa conținută în toate gălețile de la găleata i la găleata n ($n - i + 1$ litri) .

$$1 \leq e_i \leq n - i + 1$$

Obținem

$$E_n = e_2 + e_3 + \dots + e_{n-1} + e_n \leq (n - 1) + (n - 2) + \dots + 2 + 1 = \frac{n(n-1)}{2}$$

Să demonstrăm în continuare că pentru orice valoare E_n cu proprietatea că $n - 1 \leq E_n \leq \frac{n(n-1)}{2}$ există cel puțin o soluție.

Pentru $n = 1$ efortul depus este $E_1 = 0$ dar $1 - 1 \leq 0 \leq \frac{1(1-1)}{2}$.

Pentru $n = 2$ efortul depus este $E_2 = 1$ dar $2 - 1 \leq 1 \leq \frac{2(2-1)}{2}$.

Să presupunem că pentru $n - 1$ găleți știm deja că putem vărsa toată apa în prima găleată cu orice efort E_{n-1} cu proprietatea că $n - 2 \leq E_{n-1} \leq \frac{(n-1)(n-2)}{2}$.

Pentru n găleți considerăm două cazuri particulare:

Cazul 1.

Vărsăm primele $n - 1$ găleți în prima și la final vărsăm ultima găleată în prima.

Efortul depus la ultima vărsare este 1 și pentru celelalte este E_{n-1} și astfel putem vărsa toată apa în prima găleată pentru

$$n - 1 = n - 2 + 1 \leq E_n \leq \frac{(n-1)(n-2)}{2} + 1$$

Cazul 2.

Vărsăm ultimele $n - 1$ în a doua și la final vărsăm a doua găleată în prima găleată pentru $2n - 3 = n - 2 + n - 1 \leq E_n \leq \frac{(n-1)(n-2)}{2} + n - 1 = \frac{n(n-1)}{2}$

Pentru a arăta că în cele două cazuri acoperim toate valorile posibile ale lui E_n între $n - 1$ și $\frac{n(n-1)}{2}$ este sufficient să dovedim că valoarea minimă din al doilea caz nu poate depăși valoarea maximă din primul caz cu mai mult de o unitate adică $\frac{(n-1)(n-2)}{2} + 1 \leq 2n - 3 - 1$

Ultima inegalitate este echivalentă cu $(n - 3)(n - 4) \leq 0$ și este evident adevărată pentru orice valoare întreagă n .

Demostrația de până aici ne sugerează și o metodă prin care se poate realiza succesiunea de vârsări construind această succesiune. Reluăm raționamentul pe oricare interval de găleți $[st, dr]$ conținând $m = dr - st + 1$ găleți. Ne propunem să aducem toată apa în st cu un efort cunoscut ef .

Dacă $ef \geq 2m - 3$ atunci procedăm ca în cazul 2. Mai precis efectuăm toate operațiile de vârsare care aduc apa din intervalul $[st + 1, dr]$ în $st + 1$ cu efortul $ef - m + 1$ după care efectuăm vârsarea din $st + 1$ în st cu efortul $m - 1$.

În caz contrar procedăm ca în cazul 1 și mutăm toată apa din intervalul $[st, dr - 1]$ în st cu efort $ef - 1$ după care vârsăm dr în st cu efort 1.

Implementarea se poate face în două moduri:

Recursiv: implementăm o funcție cu parametrii st, dr, ef care va fi apelată recursiv în funcție de valoarea lui ef fie cu valorile $st + 1, dr, ef - m + 1$ fie cu valorile $st, dr - 1, ef - 1$. La revenire se afișează $st + 1$ și st în cazul primului apel respectiv dr și st în cazul celuilalt.

Iterativ: în loc să implementăm funcția recursivă simulăm *stiva* de apeluri memorând vârsările în ordine inversă și trecând de la un pas la altul prin modificarea a 3 variabile st, dr și ef care au inițial valorile $st = 1, dr = n, ef = e$ și rulam o instrucțiune repetitivă până când $dr = st$ sau echivalent $ef = 0$.

Algoritmul este liniar în ambele cazuri atât ca timp de executare cât și ca memorie utilizată

Notă:

Structuri de date utilizate:

In cazul implementării recursive nu se utilizează structuri de date.

In cazul implementării *iterative* este necesară implementarea *stivei* care conține răspunsurile. Pentru aceasta se pot utiliza doi vectori standard iar ca alternativă se pot utiliza structuri specializate din STL (vector<pair<int,int>>, stack<pair<int,int>>)

Tipul problemei:

Ad-hoc, algoritm constructiv, greedy

In funcție de rezolvarea abordată - recursivitate, two-pointers, structuri de date (stiva)

Gradul de dificultate: 2

2.1.2 Cod sursă

Listing 2.1.1: galeti_rec_PA.cpp

```

1 #include <bits/stdc++.h>
2 #define RECURSIV 1
3
4 using namespace std;
5
6 ifstream f("galeti.in");
7 ofstream g("galeti.out");
8
9 void solve_recursiv(int, int, long long);
10 void solve_iterativ();
11
12 int N;
13 long long E;
14
15 int main()
16 {
17     f>>N>>E;
18     if(RECURSIV)
19     {

```

```

21         solve_recurziv(1,N,E);
22         return 0;
23     }
24     solve_iterativ();
25     return 0;
26 }
27
28 void solve_recurziv(int ST,int DR,long long E)
29 {
30     int n=DR-ST+1;
31     if(n==1) return;
32     if(E>=2LL*n-3LL)
33     {
34         solve_recurziv(ST+1,DR,E-1LL*n+1LL);
35         g<<ST+1<<' '<<ST<<'\n';
36         return;
37     }
38     solve_recurziv(ST,DR-1,E-1LL);
39     g<<DR<<' '<<ST<<'\n';
40 }
41
42 void solve_iterativ()
43 {
44     int ST=1,DR=N;
45     stack<pair<int,int>> sol;
46
47     while(ST<DR)
48     {
49         int n=DR-ST+1;
50         if(E>2LL*n-3LL)
51         {
52             sol.push(make_pair(ST+1,ST));
53             ST++;
54             E-=1LL*n-1LL;
55         }
56         else
57         {
58             sol.push(make_pair(ST,DR));
59             DR--,E-=1LL;
60         }
61     }
62
63     for(;sol.size();sol.pop())
64         g<<sol.top().first<<' '<<sol.top().second<<'\n';
65 }
```

Listing 2.1.2: galeti_SzZ_ite.cpp

```

1 #include <iostream>
2
3 #define Int long long
4
5 using namespace std;
6
7 ifstream fin("galeti.in");
8 ofstream fout("galeti.out");
9
10 Int N,C,a[1000010],prec[1000010];
11
12 void galeata(Int poz,Int cost,Int dest)
13 {
14     if (poz==N)
15     {
16         fout<<poz<<" "<<dest<<"\n";
17     }
18     else
19     {
20         if (N+N-poz-poz+1<=cost)
21         {
22             galeata(poz+1,cost-N+poz-1,poz);
23             fout<<poz<<" "<<dest<<"\n";
24         }
25         else
26         {
27             fout<<poz<<" "<<dest<<"\n";
28             galeata(poz+1,cost-1,dest);
29         }
30     }
31 }
```

```

28         }
29     }
30
31     int main()
32     {
33         fin>>N>>C;
34         Int st=1, dr=N-1;
35
36         for (Int i=2; i<=N; i++)
37             if (N-i+1+N-i<=C)
38             {
39                 a[dr--]=i;
40                 C-=N-i+1;
41             }
42             else
43             {
44                 a[st++]=i;
45                 C-=1;
46             }
47
48         for(Int i=2;i<=N;i++)
49             prec[i]=i-1;
50
51         for(Int i=1;i<=N-1;i++)
52         {
53             fout<<a[i]<<" "<<prec[a[i]]<<"\n";
54             prec[a[i]+1]=prec[a[i]];
55         }
56
57         return 0;
58     }

```

Listing 2.1.3: galeti_SzZ_rec.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 ifstream fin("galeti.in");
6 ofstream fout("galeti.out");
7
8 int N,C;
9
10 void galeata(int poz,long long cost,int dest)
11 {
12     if (poz==N)
13     {
14         fout<<poz<<" "<<dest<<"\n";
15     }
16     else
17     {
18         if (N+N-poz-poz+1<=cost)
19         {
20             galeata(poz+1,cost-N+poz-1,poz);
21             fout<<poz<<" "<<dest<<"\n";
22         }
23         else
24         {
25             fout<<poz<<" "<<dest<<"\n";
26             galeata(poz+1,cost-1,dest);
27         }
28     }
29
30     int main()
31     {
32         fin>>N>>C;
33         galeata(2,C,1);
34
35         return 0;
36     }

```

Listing 2.1.4: galeti100_PA.cpp

```

1 #include <bits/stdc++.h>
2

```

```

3 #define RECURSIV 0
4
5 using namespace std;
6
7 ifstream f("galeti.in");
8 ofstream g("galeti.out");
9
10 void solve_recursiv(int,int,long long);
11 void solve_iterativ();
12
13 int N,From[1000010],To[1000010];
14 long long E;
15
16 int main()
17 {
18     f>>N>>E;
19     if(RECURSIV){solve_recursiv(1,N,E);return 0;}
20     solve_iterativ();
21     return 0;
22 }
23
24
25 void solve_recursiv(int ST,int DR,long long E)
26 {
27     int n=DR-ST+1;
28     if(n==1) return;
29     if(E>=2LL*n-3LL)
30     {
31         solve_recursiv(ST+1,DR,E-1LL*n+1LL);
32         g<<ST+1<<' '<<ST<<'\n';
33         return;
34     }
35     solve_recursiv(ST,DR-1,E-1LL);
36     g<<DR<<' '<<ST<<'\n';
37 }
38
39 void solve_iterativ()
40 {
41     int ST=1,DR=N,k=0;
42     while(ST<DR)
43     {
44         int n=DR-ST+1;
45         if(E>=2LL*n-3LL)
46         {
47             From[++k]=ST+1;
48             To[k]=ST;
49             ST++;E-=1LL*n-1LL;
50         }
51         else
52         {
53             From[++k]=DR;
54             To[k]=ST;
55
56             DR--,E-=1LL;
57         }
58     }
59
60     for(;k;k--)
61         g<<From[k]<<' '<<To[k]<<'\n';
62 }
```

Listing 2.1.5: mihai-100.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 int main()
6 {
7     ifstream cin("galeti.in");
8     ofstream cout("galeti.out");
9
10    int n; cin >> n;
11    int64_t c; cin >> c;
```

```

13     int chain_length = 1;
14     int64_t current = n - 1;
15
16     while(current < c)
17     {
18         current += chain_length;
19         chain_length += 1;
20     }
21
22     int extra = -1;
23
24     if(current > c)
25     {
26         chain_length -= 1;
27         current -= chain_length;
28         extra = c - current;
29     }
30
31     int now = 2 + chain_length - 1;
32
33     while(now != 2 + extra - 1 and now != 1)
34     {
35         cout << now << " " << now - 1 << "\n";
36         now -= 1;
37     }
38
39     if(extra > 0)
40         cout << 2 + chain_length << " " << now << "\n";
41
42     while(now != 1)
43     {
44         cout << now << " " << now - 1 << "\n";
45         now -= 1;
46     }
47
48     int where = 2 + chain_length;
49     if(extra > 0)
50         where += 1;
51
52     while(where <= n)
53     {
54         cout << where << " " << 1 << "\n";
55         where += 1;
56     }
57 }
```

2.1.3 *Rezolvare detaliată

2.2 ramen

Problema 2 - ramen

90 de puncte

Ai deschis recent un restaurant cu specific japonez, iar lucrurile nu merg grozav. Uneori clienții ajung să aștepte foarte mult mâncarea comandată, iar acum crezi că ai înțeles de ce se întâmplă acest lucru.

Restaurantul nu are mese, ci un singur bar foarte lung dotat cu o bandă rulantă care transportă porțiile de mâncare de la bucătărie la client. Barul are 500.000.000 de scaune numerotate în ordine crescătoare, scaunul 1 fiind cel mai apropiat de bucătărie. Uneori clienții fac noi comenzi. O comandă făcută la secunda T de către clientul aflat pe scaunul cu numărul P va ajunge instant la bucătărie. Prepararea mâncării va dura D secunde, iar apoi mâncarea va fi pusă pe bandă și va dura exact P secunde ca aceasta să ajungă la client. În acest timp, mâncarea va trece prin fața scaunelor $1, 2, \dots, P-1$. Dacă dintr-un anumit motiv clientul nu își ridică mâncarea de pe bandă, aceasta va continua să se deplaseze. În caz contrar, clientul în cauză se așteaptă ca mâncarea să ajungă la scaunul său la secunda $T + D + P$.

Deocamdată restaurantul servește un singur fel de mâncare: ramen. Astfel, comenziile făcute de clienți ajung să fie ușor interschimbabile, iar aceștia se arată foarte deschiși la a profita de pe urma acestui fapt.

Se cunosc următoarele:

- Un client poate avea zero sau mai multe comenzi în aşteptare.
- Un client care are zero comenzi în aşteptare este complet inactiv.
- Numărul de comenzi în aşteptare ale unui client care face o comandă la secunda T va crește cu o unitate exact la secunda T .
- Un client care are în aşteptare cel puțin o comandă va ridica de pe bandă prima porție de ramen care trece prin fața sa, indiferent dacă aceasta îi era destinată sau nu. Dacă va face acest lucru la momentul T , numărul său de comenzi în aşteptare va scădea cu o unitate exact la momentul T .

Spre exemplu, analizăm situația următoare cu două comenzi:

Durata de preparare a ramenului este $D = 2$ secunde. Această valoare este o constantă și se aplică identic fiecărei comenzi.

La secunda $T_1 = 10$, persoana de pe scaunul cu numarul $P_1 = 8$ (să o numim A) comandă o porție de ramen. La secunda $T_1 + D = 12$, porția sa este pusă pe bandă.

La secunda $T_2 = 16$, persoana de pe scaunul cu numarul $P_2 = 6$ (să o numim B) comandă o porție de ramen. La secunda $T_2 + D = 18$, porția sa este pusă pe bandă.

La secunda 19 porția destinată clientului A trece prin fața scaunului 6, iar clientul B, fiind el însuși în aşteptarea unei comenzi o va lua și o va mâncă. El va mâncă deci la secunda 19 și va ignora apoi propria sa comandă, care va trece pe lângă el.

La secunda 28 porția destinată clientului B va ajunge la clientul A, iar acesta o va lua și o va mâncă. El va mâncă deci la secunda 28.

Se observă că în general, în ciuda întârzierilor create, fiecare client va consuma exact câte porții a comandat.

Cerințe

Pentru a evalua impactul acestui obicei asupra timpilor de aşteptare, ai obținut date despre toate comenziile date în ziua curentă. Îți propui să află, pentru fiecare comandă următoarea valoare: dacă respectiva comandă este a NR -a făcută de clientul respectiv, care este secunda la care clientul în cauză va mâncă pentru a NR -a oară?

Date de intrare

Fișierul de intrare **ramen.in** va conține pe prima linie numărul de comenzi N , respectiv timpul de preparare a unei porții de ramen D . Următoarele N linii vor conține câte o pereche de numere: $T[i]$, secunda la care este făcută a i -a comandă, respectiv $P[i]$, numărul scaunului de la care s-a făcut a i -a comandă. Se garantează că timpii de comandă sunt distincți și sunt crescători în ordinea citirii lor.

Date de ieșire

Fișierul de ieșire **ramen.out** va conține N linii, fiecare conținând o singură valoare naturală: a i -a valoare va reprezenta răspunsul cerut pentru a i -a comandă în ordinea citirii.

Restricții și precizări

1. $1 \leq N \leq 100.000$
2. $0 \leq D, T[i] \leq 500.000.000$, pentru orice $1 \leq i \leq N$
3. $1 \leq P[i] \leq 500.000.000$, pentru orice $1 \leq i \leq N$
4. Se garantează că $T[i] < T[i + 1]$, pentru orice $1 \leq i < N$.
5. Pentru teste în valoare totală de 22 de puncte, se garantează în plus față de restricțiile generale că $N \leq 2000$ și $D, T[i], P[i] \leq 5000$
6. Pentru alte teste în valoare totală de 25 de puncte, se garantează în plus față de restricțiile generale că $N \leq 2000$.

Exemple

ramen.in	ramen.out	Explicații
2 2	28	Exemplul descris în enunț.
10 8	19	
16 6		

3 2 5 4 6 4 7 3	12 13 10	Observați că în acest exemplu clientul de pe scaunul cu numărul 4 face două comenzi. Răspunsul corespunzător primei comenzi este secunda în care clientul mănâncă pentru prima oară, iar răspunsul corespunzător comenzi cu numărul doi este secunda în care clientul mănâncă pentru a doua oară.
3 0 0 6 3 3 4 5	10 3 8	Observați că în acest exemplu clientul de pe scaunul cu numărul 3 face o comandă la secunda 3. Exact în același moment, îi apare o porție de ramen în față, iar el o consumă imediat.

Timp maxim de executare/test: **1.8** secunde

Memorie: total **128 MB** din care pentru stivă **32 MB**

Dimensiune maximă a sursei: **20 KB**

2.2.1 Indicații de rezolvare

????? - ?????

Se pot obține 32 de puncte simulând comenzi și mișcarea porțiilor pe bandă secundă cu secundă.

Încercând să obținem o soluție mai rapidă, observăm că procesarea comenzi în ordinea timpilor nu este cea mai oportună: odată ce se primesc comenzi noi, acestea pot afecta orice altă comandă precedentă. Ne putem pune problema găsirii unei ordini de procesare care să aibă proprietatea că odată rezolvată o comandă, răspunsul pentru această comandă rămâne același până la finalul programului.

Oordonare care se pretează natural la această cerință este sortarea în ordine crescătoare după poziția clienților. Odată ce am aflat răspunsurile pentru acele NR comenzi care implică clienții cei mai apropiati de bucătărie, putem afla răspunsul și pentru a $(NR + 1)$ -a comandă iar acesta va fi permanent, fiindcă în general o anumită comandă poate fi afectată doar de comenzi făcute de pe poziții mai mici.

Să vedem deci cum determinăm răspunsul pentru comanda de poziție minimă. Dacă această comandă este definită de valorile TT , PP , atunci ea ar putea fi satisfăcută cu orice comandă care are $T(i) + D + PP \geq TT$ (orice porție cu $T(i)$ mai mic ar fi trecut deja de scaunul țintă înainte ca acesta să fi făcut comanda). Dintre acestea, răspunsul va fi dat de comanda cu $T(i)$ minim. Cu alte cuvinte avem nevoie să găsim cel mai mic timp $T(i)$ cu proprietatea că:

$$T(i) \geq TT - D - PP$$

Răspunsul va fi apoi $T(i) + D + PP$. Apoi va trebui să stergem această comandă din mulțimea comenzi care mai pot constitui răspunsuri.

Soluțiile care caută și sterg aceste valori în timp liniar pot obține 57 de puncte. Pentru o soluție mai rapidă este necesar ca aceste operații să se efectueze în timp logaritmice. Acest lucru se poate realiza cu un *arbore de intervale* sau cu container-ul STL `std::set<>`.

O altă soluție, într-un sens duală celei precedente, presupune crearea a două tipuri de evenimente: apariția unei comenzi pe bandă, respectiv începerea disponibilității unui client de a lua orice porție din fața sa. Aceste evenimente vor fi apoi parcuse în ordine crescătoare a timpilor, iar pentru fiecare comandă se va afla la ce client va ajunge. În general, o comandă va ajunge la clientul cu indice minim dintre cei disponibili la momentul respectiv. Apoi, clientul respectiv trebuie sters din mulțimea clienților disponibili.

Soluțiile care caută acest minim și îl sterg în timp liniar pot obține deasemenea 57 de puncte. O soluție mai rapidă se poate obține efectuând aceste operații cu ajutorul unui *heap* sau cu container-ul STL `std::set<>`.

2.2.2 Cod sursă

Listing 2.2.1: mihai100.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int MAXX = 1e3;
6
7 class Event
8 {
9     public:
10     int type;
11     int time;
12     pair<int, int> who;
13
14     Event(int _type, int _time, pair<int, int> _who = pair<int, int>()):
15         type(_type),
16         time(_time),
17         who(_who)
18     {}
19
20     bool operator<(Event other) const
21     {
22         if(time == other.time)
23             return type > other.type;
24         return time < other.time;
25     }
26 };
27
28 vector<int64_t> solveFast(vector<int> t, vector<int> p, int T)
29 {
30     int n = t.size();
31     vector<Event> events;
32
33     for(int i = 0; i < n; ++i)
34     {
35         events.push_back(Event(0, t[i] + T));
36         events.push_back(Event(1, t[i] - p[i], {p[i], i}));
37     }
38
39     sort(events.begin(), events.end());
40
41     set<pair<int, int>> waiting;
42
43     vector<int64_t> sol(n, 0);
44
45     for(const auto& event : events)
46     {
47         if(event.type == 0)
48         {
49             auto client = *(waiting.begin());
50             sol[client.second] = event.time + client.first;
51             waiting.erase(waiting.begin());
52         }
53         else
54             waiting.insert(event.who);
55     }
56
57     return sol;
58 }
59
60 int main()
61 {
62     ifstream cin("ramen.in");
63     ofstream cout("ramen.out");
64
65     int n, T;
66     cin >> n >> T;
67
68     vector<int> t(n, 0), p(n, 0);
69
70     set<int> times;
71
72     for(int i = 0; i < n; ++i)
73     {
74         cin >> t[i] >> p[i];

```

```

75         times.insert(t[i]);
76     }
77
78     assert(int(times.size()) == n);
79
80     vector<int64_t> sol = solveFast(t, p, T);
81
82     for(int i = 0; i < n; ++i)
83         cout << sol[i] << "\n";
84 }
```

Listing 2.2.2: mihaiSmallN.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int MAXX = 1e3;
6
7 class Event
8 {
9     public:
10     int type;
11     int time;
12     pair<int, int> who;
13
14     Event(int _type, int _time, pair<int, int> _who = pair<int, int>()):
15         type(_type),
16         time(_time),
17         who(_who)
18     {}
19
20     bool operator<(Event other) const
21     {
22         if(time == other.time)
23             return type > other.type;
24         return time < other.time;
25     }
26 };
27
28 vector<int64_t> solveFast(vector<int> t, vector<int> p, int T)
29 {
30     int n = t.size();
31
32     vector<Event> events;
33     map<int, vector<int>> orders;
34     map<int, vector<int64_t>> answers;
35
36     for(int i = 0; i < n; ++i)
37     {
38         events.push_back(Event(0, t[i] + T));
39         orders[p[i]].push_back(i);
40         events.push_back(Event(1, t[i] - p[i], {p[i], t[i]}));
41     }
42
43     sort(events.begin(), events.end());
44     vector<pair<int, int>> waiting;
45
46     for(const auto& event : events)
47     {
48         if(event.type == 0)
49         {
50             auto client = *(waiting.begin());
51             answers[client.first].push_back(event.time + client.first);
52             waiting.erase(waiting.begin());
53         }
54         else
55         {
56             waiting.push_back(event.who);
57             sort(waiting.begin(), waiting.end());
58         }
59     }
60
61     vector<int64_t> sol(n, 0);
```

```

63     for(auto elem : orders)
64     {
65         int who = elem.first;
66         for(int i = 0; i < int(orders[who].size()); ++i)
67             sol[orders[who][i]] = answers[who][i];
68     }
69
70     return sol;
71 }
72
73 int main()
74 {
75     ifstream cin("ramen.in");
76     ofstream cout("ramen.out");
77
78     int n, T;
79     cin >> n >> T;
80
81     vector<int> t(n, 0), p(n, 0);
82
83     set<int> times;
84     for(int i = 0; i < n; ++i)
85     {
86         cin >> t[i] >> p[i];
87         times.insert(t[i]);
88     }
89
90     assert(int(times.size()) == n);
91
92     vector<int64_t> sol = solveFast(t, p, T);
93
94     for(int i = 0; i < n; ++i)
95         cout << sol[i] << "\n";
96 }
```

Listing 2.2.3: mihaiSmallT.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int MAXX = 1e5;
6
7 vector<int64_t> solveSlow(vector<int> t, vector<int> p, int T)
8 {
9     int64_t time = 0;
10    int n = t.size();
11
12    vector<int> wait(MAXX + 1, 0);
13    vector<int> food;
14
15    map<int, vector<int>> orders;
16    map<int, vector<int64_t>> answers;
17
18    for(int i = 0; i < n; ++i)
19        orders[p[i]].push_back(i);
20
21    int rem = n;
22
23    while(rem)
24    {
25        for(int i = 0; i < n; ++i)
26            if(t[i] == time)
27            {
28                food.push_back(-T);
29                wait[p[i]] += 1;
30            }
31
32        vector<int> redo;
33        for(auto& elem : food)
34            if(elem >= 0 and wait[elem] > 0)
35            {
36                wait[elem] -= 1;
37                rem -= 1;
38                answers[elem].push_back(time);
39            }
40    }
41}
```

```

39         }
40     else
41         redo.push_back(elem + 1);
42
43     food = redo;
44     time += 1;
45 }
46
47 vector<int64_t> sol(n, 0);
48
49 for(auto elem : orders)
50 {
51     int who = elem.first;
52     for(int i = 0; i < int(orders[who].size()); ++i)
53         sol[orders[who][i]] = answers[who][i];
54 }
55
56 return sol;
57 }
58
59 int main()
60 {
61     ifstream cin("ramen.in");
62     ofstream cout("ramen.out");
63
64     int n, T;
65     cin >> n >> T;
66
67     vector<int> t(n, 0), p(n, 0);
68     for(int i = 0; i < n; ++i)
69         cin >> t[i] >> p[i];
70
71     auto sol = solveSlow(t, p, T);
72     for(auto temp : sol)
73         cout << temp << "\n";
74 }
```

Listing 2.2.4: nostl_PA.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 ifstream f("ramen.in");
6 ofstream g("ramen.out");
7
8 const int N = (1<<17)+10;
9 const int infinit = 1<<30;
10
11 int n, T, tp[N], po[N], x[N], i, z, sol[N], tmax[2*N], tmin[2*N], cauta(int, int);
12
13 bool crit(int u, int v) {return make_pair(po[u], tp[u]) <= make_pair(po[v], tp[v]);}
14
15 inline void upd(int nod)
16 {
17     int st, dr; st=2*nod; dr=st+1;
18     if(!tmin[st]) st=dr; else if(!tmin[dr]) dr=st;
19     tmin[nod]=tmin[st]; tmax[nod]=tmax[dr];
20 }
21
22 int main()
23 {
24     f >> n >> T;
25
26     for(i = 1; i <= n; i++)
27     {
28         f >> tp[i] >> po[i];
29         x[i]=i;
30     }
31
32     sort(x+1, x+n+1, crit);
33     z=1;
34     while(z<n) z*=2;
35     z--;
36 }
```

```

37     for(i=1;i<=n;i++) tmax[z+i]=tp[i]+T+1;
38
39     sort(tmax+z+1,tmax+z+n+1);
40     copy(tmax+z+1,tmax+z+n+1,tmin+z+1);
41
42     for(i=z;i>=1;i--)
43         upd(i);
44
45     for(i=1;i<=n;i++)
46         sol[x[i]]=cauta(1,tp[x[i]]-po[x[i]]+1)+po[x[i]]-1;
47
48     for(i=1;i<=n;i++)
49         g<<sol[i]<<'\\n';
50
51     return 0;
52 }
53
54 int cauta(int nod,int timp)
55 {
56     int st,dr,ret;
57     if(!tmin[nod]) return infinit;
58     if(tmax[nod]==tmin[nod])
59     {
60         ret=tmax[nod];
61         tmax[nod]=tmin[nod]=0;
62         return ret;
63     }
64
65     st=2*nod;dr=st+1;
66     if(!tmin[st])
67         ret=cauta(dr,timp);
68     else
69     if(!tmin[dr])
70         ret=cauta(st,timp);
71     else
72     if(tmax[st]>=timp)
73         ret=cauta(st,timp);
74     else
75         ret=cauta(dr,timp);
76
77     upd(nod);
78     return ret;
79 }
```

Listing 2.2.5: ramen_set_PA.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 ifstream f("ramen.in");
6 ofstream g("ramen.out");
7
8 const int N = 100010;
9
10 int n, T, tp, po, i, sol[N];
11
12 set< tuple < int, int, int > > clienti;
13 set< int > farfurii;
14
15 int main()
16 {
17     f >> n >> T;
18
19     for(i = 1; i <= n; i++)
20     {
21         f >> tp >> po;
22         clienti.insert(make_tuple(po, tp, i));
23         farfurii.insert(tp);
24     }
25
26     for(auto it:clienti)
27     {
28         tie(po, tp, i)=it;
29         set<int>::iterator IT = farfurii.lower_bound(tp - po - T);
```

```

30         sol[i] = *IT + T + po;
31         farfurii.erase( *IT );
32     }
33
34     for(i = 1; i <= n; i++)
35         g << sol[i] << '\n';
36
37     return 0;
38 }
```

2.2.3 *Rezolvare detaliată

2.3 aquapark

Problema 3 - aquapark

90 de puncte

Pentru a atrage turiștii, primăria unui oraș a hotărât că va construi un parc acvatic imens cu n piscine. Parcul va avea o zonă acoperită și va fi înconjurat de un spațiu deschis pentru activități în aer liber.

Zona închisă va fi acoperită de o singură clădire de forma unui poligon, iar piscinele vor fi proiectate în vârfurile poligonului și vor putea comunica între ele prin m căi de acces care nu se vor interseca. Căile de acces între două piscine pot fi de tipul 1: canal umplut cu apă în interiorul clădirii, sau de tipul 2: o alei în afara clădirii.

În exemplul alăturat prin linie punctată se delimită zonă acoperită a parcului. Avem 5 piscine, există 6 căi de acces: (1,2), (2,5), (1,4), (1,3), (3,4), (3,5), dintre care 2 sunt canale (tipul 1): (1,3) și (1,4), respectiv 4 sunt alei (tipul 2): (1,2), (2,5), (3,4) și (3,5).

Un alt proiect ce păstrează aceleși căi de acces, dar diferă prin tipul acestora, este să construim 4 canale: (1,2), (3,4), (3,5), (2,5) respectiv 2 alei: (1,3), (1,4).

În total putem construi 8 proiecte distincte cu aceste căi de acces. Două proiecte se consideră distincte dacă există cel puțin o cale de acces ce are tipuri diferite pe cele două proiecte.

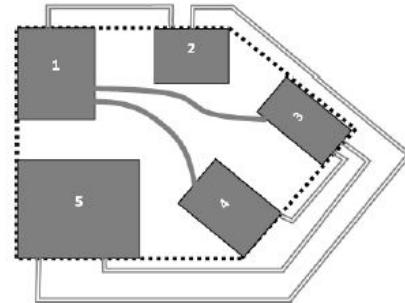


Figura 2.1: aquapark

Cerințe

Cunoscând căile de acces între piscine, să se determine una dintre cerințele următoare:

1. - o modalitate de construire a căilor de acces, precizând tipul fiecăreia;
2. - numărul proiectelor distincte.

Date de intrare

Fișierul de intrare **aquapark.in** conține pe prima linie trei numere separate prin câte un spațiu c n m reprezentând în această ordine tipul cerinței, numărul piscinelor respectiv numărul căilor de acces. Următoarele m linii conțin câte două numere x și y , reprezentând o cale de acces între piscina x și piscina y .

Date de ieșire

Fișierul de ieșire **aquapark.out** va conține în funcție de valoarea lui c următoarele informații:

- dacă $c = 1$: pe m linii se vor tipări câte trei numere separate prin câte un spațiu x y t , semnificând că între piscina x și piscina y există o cale de acces de tipul t (1-canal, 2-alee). Fiecare mulțime citită din fișierul de intrare va trebui să apară exact o dată în fișierul de ieșire, indiferent de ordinea citirii.

- dacă $c = 2$: se va tipări un singur număr ce va semnifica numărul proiectelor distincte *modulo* 666013.

Restricții și precizări

- $1 \leq n \leq 70000$
- $1 \leq m \leq 100000$
- Între două piscine există cel mult o cale de acces
- Nu există o cale de acces de la o piscină la ea însăși
- Se asigură că pentru datele de test există cel puțin o soluție,
- Dacă există mai multe soluții se poate afișa oricare dintre acestea.
- Pentru teste în valoare de 16 puncte $n, m \leq 15$
- Pentru alte teste în valoare de 49 de puncte $n \leq 1000, m \leq 1500$
- Punctajul maxim al problemei este de 100 de puncte dintre care 10 puncte din oficiu.

Exemple

aquapark.in	aquapark.out	Explicații																											
1 5 6 1 2 2 5 1 4 3 1 4 3 5 3	1 2 1 1 3 1 1 4 1 2 5 2 3 4 1 3 5 2	c=1, se cere o modalitate de construcție a căilor de acces: Avem cale de acces de tip 1 (canale) între piscinele (1,2), (1,3), (1,4) și (3,4). Avem cale de acces de tip 2 (alee) între piscinele (2,5) și (3,5).. Vezi desenul de mai sus.																											
2 5 6 1 2 2 5 1 4 3 1 4 3 5 3	8	Avem 8 modalități distincte de a construi căile parcului acvatic:																											
		<table border="1"> <thead> <tr> <th>Solutie</th><th>căi de tipul 1</th><th>căi de tipul 2</th></tr> </thead> <tbody> <tr><td>1</td><td>(1,2) (1,3) (1,4) (3,4)</td><td>(2,5) (3,5)</td></tr> <tr><td>2</td><td>(1,3) (1,4) (3,4)</td><td>(1,2) (2,5) (3,5)</td></tr> <tr><td>3</td><td>(1,2) (1,3) (1,4)</td><td>(2,5) (3,5) (3,4)</td></tr> <tr><td>4</td><td>(1,3) (1,4)</td><td>(1,2) (2,5) (3,5) (3,4)</td></tr> <tr><td>5</td><td>(2,5) (3,5)</td><td>(1,2) (1,3) (1,4) (3,4)</td></tr> <tr><td>6</td><td>(1,2) (2,5) (3,5)</td><td>(1,3) (1,4) (3,4)</td></tr> <tr><td>7</td><td>(2,5) (3,5) (3,4)</td><td>(1,2) (1,3) (1,4)</td></tr> <tr><td>8</td><td>(1,2) (2,5) (3,5) (3,4)</td><td>(1,3) (1,4)</td></tr> </tbody> </table>	Solutie	căi de tipul 1	căi de tipul 2	1	(1,2) (1,3) (1,4) (3,4)	(2,5) (3,5)	2	(1,3) (1,4) (3,4)	(1,2) (2,5) (3,5)	3	(1,2) (1,3) (1,4)	(2,5) (3,5) (3,4)	4	(1,3) (1,4)	(1,2) (2,5) (3,5) (3,4)	5	(2,5) (3,5)	(1,2) (1,3) (1,4) (3,4)	6	(1,2) (2,5) (3,5)	(1,3) (1,4) (3,4)	7	(2,5) (3,5) (3,4)	(1,2) (1,3) (1,4)	8	(1,2) (2,5) (3,5) (3,4)	(1,3) (1,4)
Solutie	căi de tipul 1	căi de tipul 2																											
1	(1,2) (1,3) (1,4) (3,4)	(2,5) (3,5)																											
2	(1,3) (1,4) (3,4)	(1,2) (2,5) (3,5)																											
3	(1,2) (1,3) (1,4)	(2,5) (3,5) (3,4)																											
4	(1,3) (1,4)	(1,2) (2,5) (3,5) (3,4)																											
5	(2,5) (3,5)	(1,2) (1,3) (1,4) (3,4)																											
6	(1,2) (2,5) (3,5)	(1,3) (1,4) (3,4)																											
7	(2,5) (3,5) (3,4)	(1,2) (1,3) (1,4)																											
8	(1,2) (2,5) (3,5) (3,4)	(1,3) (1,4)																											

Timp maxim de executare/test: **2.0** secunde

Memorie: total **128 MB** din care pentru stivă **32 MB**

Dimensiune maximă a sursei: **20 KB**

2.3.1 Indicații de rezolvare

Szabo Zoltan - ISJ Mures Tg Mures
Budau Adrian - Universitatea București

Solutie pentru 75 de puncte

Inițial să considerăm, că toate muchiile sunt desenate în interior.

Vor exista perechi de muchii care se intersectează iar alte perechi nu.

Două muchii (x, y) respectiv (u, v) , cu $x < y$ și $u < v$, se intersectează dacă și numai dacă:

- $x < u < y < v$ sau
- $u < x < v < y$.

Dacă două muchii interioare se intersectează, una dintre ele trebuie să devină exterioară.

În acest fel unel muchii vor fi condiționate de alte muchii. Putem construi un *graf bipartit* din muchiile grafului.

Pornind de la o muchie, cu ajutorul unei *cozi* putem eticheta fiecare muchie. Prima va fi etichetată cu 1, toate cele care se intersectează cu aceasta vor fi etichetate cu 2, cu ajutorul unei cozi vom eticheta toate muchiile care se intersectează cu o muchie deja etichetată. Enunțul problemei ne garantează că de fiecare dată muchiile nou etichetate nu vor intra în conflict cu muchiile deja etichetate.

Numărul proiectelor discrete este egal cu 2 la puterea numărului de *componente conexe* ale grafului bipartit.

Această rezolvare are complexitate $O(m^2)$

Soluție pentru 100 de puncte

Soluția de 100 de puncte se bazează pe aceleași observații ca și cea de 75 de puncte, anume că este nevoie de construcția aceluia *graf bipartit*. Ce este necesar de observat este că acest graf bipartit are prea multe muchii, multe din ele nenecesare.

De exemplu dacă avem în acest graf bipartit 2 vârfuri deosebite și 2 de altă parte, atunci nu sunt necesare niciodată toate cele 4 muchii, sunt suficiente oricare 3.

Se poate folosi o *baleiere* pe baza evenimentelor de forma

- La poziția x apare o muchie (x, y)
- La poziția y dispără o muchie (x, y)

La orice moment se mențin muchiile care au apărut și nu au dispărut însă, și deci când dispăr o muchie (x, y) se poate itera prin această listă de muchii de la cea mai recentă aparută spre cea mai puțin recentă.

Orice muchie de forma (z, t) care a apărut dar nu a dispărut încă are cu siguranță $t > y$, deci dacă $z > x$ atunci aceste două muchii se intersectează dacă sunt amândouă în interior sau în exterior (deci se poate trage o muchie în graful bipartit).

Implementarea directă a acestei idei funcționează în complexitate $O(\text{numrul de intersecții a muchiilor})$, și nu este încă suficient de rapid, însă o mică observație va reduce complexitatea la $O(M \log M)$.

Dacă atunci când dispăr muchia (x, y) , se descoperă că ea se intersectează cu muchiile $(z_1, t_1), (z_2, t_2), \dots, (z_k, t_k)$ cu $z_1 \leq z_2 \leq \dots \leq z_k$ atunci $t_k \leq t_{k-1} \leq \dots \leq t_2 \leq t_1$ altfel nu ar exista soluție. Deci orice altă muchie ar mai dispărea ulterior (x, y) (evident $y \geq y$) și care se intersectează cu un (z_i, t_i) se intersectează și cu (z_1, t_1) . Dar se știe deja în graful bipartit că (z_1, t_1) și (z_i, t_i) sunt de aceeași parte, deci nu mai merită considerată muchia (z_i, t_i) dacă $i \geq 2$. Astfel putem elimina toate muchiile (z_i, t_i) cu $i \geq 2$ atunci când procesăm muchia (x, y) .

Aceste muchii (z, t) pot fi ținute într-o *stivă*, operațiile reducându-se la *eliminarea unor elemente din vârful stivei*. Cum pentru orice eveniment în care apare o muchie (x, y) doar se adaugă în capul *stivei* muchia, iar un element poate fi șters din stivă cel mult o dată înseamnă că după *sortarea* evenimentelor (pentru a se putea folosi *baleierea*) complexitatea este $O(M)$.

Din cauza sortării complexitatea finală devine $O(M \log M)$.

2.3.2 Cod sursă

Listing 2.3.1: Adrian-100-set.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <algorithm>
5 #include <cassert>
6 #include <set>
7 #include <limits>
8
9 using namespace std;
10
11 struct Edge
12 {
13     int from, to, index;
14 };
15
16 struct Event
17 {
18     bool opening;
19     Edge edge;
20
21     int point() const
22     {
23         if (opening)
24             return edge.from;
25     }
26 };

```

```

25         return edge.to;
26     }
27
28     bool operator<(const Event& that) const
29     {
30         if (point() != that.point())
31             return point() < that.point();
32         if (!opening && that.opening)
33             return true;
34         if (opening && !that.opening)
35             return false;
36
37         return edge.from > that.edge.from;
38     }
39 };
40
41 static const int kModulo = 666013;
42
43 void dfs(const vector<vector<int>> &edges, vector<int> &colors, int node)
44 {
45     for (auto &neighbor : edges[node])
46     {
47         assert(colors[neighbor] != colors[node]);
48         if (colors[neighbor] == 0)
49         {
50             colors[neighbor] = 3 - colors[node];
51             dfs(edges, colors, neighbor);
52         }
53     }
54 }
55
56 int main()
57 {
58     ifstream cin("aquapark.in");
59     ofstream cout("aquapark.out");
60
61     int type; assert(cin >> type);
62     assert(1 <= type && type <= 2);
63
64     int N, M; assert(cin >> N >> M);
65     assert(1 <= N && N <= 75 * 1000);
66     assert(1 <= M && M <= 100 * 1000);
67
68     vector<Event> events;
69     events.reserve(M * 2);
70
71     vector<Edge> edges(M);
72     for (int i = 0; i < M; ++i)
73     {
74         int x, y;
75         assert(cin >> x >> y);
76         assert(1 <= x && x <= N);
77         assert(1 <= y && y <= N);
78         assert(x != y);
79         if (x > y)
80             swap(x, y);
81         edges[i] = Edge{x, y, i};
82     }
83
84     sort(edges.begin(), edges.end(), [&](const Edge& a, const Edge& b) {
85         return a.to > b.to;
86     });
87
88     for (int i = 0; i < M; ++i)
89     {
90         edges[i].index = i;
91         events.push_back(Event{true, edges[i]});
92         events.push_back(Event{false, edges[i]});
93     }
94
95     sort(edges.begin(), edges.end(), [&](const Edge& a, const Edge& b) {
96         if (a.from != b.from)
97             return a.from < b.from;
98         return a.to < b.to;
99     });
100

```

```

101     vector< vector<int> > conflict(M);
102     int answer = 0;
103     for (int i = 1; i < M; ++i)
104     {
105         if (edges[i].from == edges[i-1].from && edges[i].to == edges[i-1].to)
106         {
107             conflict[edges[i].index].push_back(edges[i-1].index);
108             conflict[edges[i-1].index].push_back(edges[i].index);
109             --answer;
110         }
111         if (i > 0)
112             assert(edges[i-1].from != edges[i].from || edges[i-1].to != edges[i].to);
113     }
114 }
115
116 sort(events.begin(), events.end());
117
118 set<pair<int, int> > S;
119 for (auto &event : events)
120 {
121     if (event.opening)
122         S.emplace(event.edge.from, event.edge.index);
123     else
124     {
125         bool kept = false;
126         for (auto it = S.upper_bound(
127             make_pair(event.edge.from, numeric_limits<int>::max())),
128             jt = it;
129             it != S.end();
130             it = jt)
131         {
132             jt = it;
133             ++jt;
134
135             conflict[event.edge.index].push_back(it->second);
136             conflict[it->second].push_back(event.edge.index);
137             if (kept)
138                 S.erase(it);
139             else
140                 kept = true;
141         }
142
143         S.erase(make_pair(event.edge.from, event.edge.index));
144     }
145 }
146
147 vector<int> colors(M, 0);
148
149 for (int i = 0; i < M; ++i)
150     if (colors[i] == 0)
151     {
152         colors[i] = 1;
153         dfs(conflict, colors, i);
154         ++answer;
155     }
156
157 if (type == 1)
158 {
159     for (auto &edge : edges)
160         cout << edge.from << " " << edge.to << " " <<
161             colors[edge.index] << "\n";
162 }
163 else
164 {
165     int as_power = 1;
166     for (int i = 0; i < answer; ++i)
167         as_power = (2 * as_power) % kModulo;
168     cout << as_power << "\n";
169 }
170 }
```

Listing 2.3.2: Adrian-100-stack.cpp

```

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

```

3 #include <vector>
4 #include <algorithm>
5 #include <cassert>
6 #include <limits>
7 #include <stack>
8
9 using namespace std;
10
11 struct Edge
12 {
13     int from, to, index;
14 };
15
16 struct Event
17 {
18     bool opening;
19     Edge edge;
20
21     int point() const
22     {
23         if (opening)
24             return edge.from;
25         return edge.to;
26     }
27
28     bool operator<(const Event& that) const
29     {
30         if (point() != that.point())
31             return point() < that.point();
32         if (!opening && that.opening)
33             return true;
34         if (opening && !that.opening)
35             return false;
36
37         if (opening)
38         {
39             if (edge.to != that.edge.to)
40                 return edge.to > that.edge.to;
41             return edge.index < that.edge.index;
42         }
43         else
44         {
45             if (edge.from != that.edge.from)
46                 return edge.from > that.edge.from;
47             return edge.index > that.edge.index;
48         }
49     }
50 };
51
52 static const int kModulo = 666013;
53
54 void dfs(const vector< vector<int> > &edges, vector<int> &colors, int node)
55 {
56     for (auto &neighbor : edges[node])
57     {
58         assert(colors[neighbor] != colors[node]);
59         if (colors[neighbor] == 0)
60         {
61             colors[neighbor] = 3 - colors[node];
62             dfs(edges, colors, neighbor);
63         }
64     }
65 }
66
67 int main()
68 {
69     ifstream cin("aquapark.in");
70     ofstream cout("aquapark.out");
71
72     int type; assert(cin >> type);
73     assert(1 <= type && type <= 2);
74
75     int N, M; assert(cin >> N >> M);
76     assert(1 <= N && N <= 75 * 1000);
77     assert(1 <= M && M <= 100 * 1000);
78 }
```

```

79     vector<Event> events;
80     events.reserve(M * 2);
81
82     vector<Edge> edges(M);
83     for (int i = 0; i < M; ++i)
84     {
85         int x, y; assert(cin >> x >> y);
86         assert(1 <= x && x <= N);
87         assert(1 <= y && y <= N);
88         assert(x != y);
89         if (x > y)
90             swap(x, y);
91         edges[i] = Edge{x, y, i};
92         events.push_back(Event{true, edges[i]});
93         events.push_back(Event{false, edges[i]});
94     }
95
96     sort(edges.begin(), edges.end(), [&](const Edge& a, const Edge& b) {
97         if (a.from != b.from)
98             return a.from < b.from;
99         return a.to < b.to;
100    });
101
102    vector<vector<int> > conflict(M);
103    int answer = 0;
104    for (int i = 1; i < M; ++i)
105    {
106        if (edges[i].from == edges[i-1].from && edges[i].to == edges[i-1].to)
107        {
108            conflict[edges[i].index].push_back(edges[i-1].index);
109            conflict[edges[i-1].index].push_back(edges[i].index);
110            --answer;
111        }
112
113        if (i > 1)
114            assert(edges[i-2].from != edges[i].from ||
115                  edges[i-2].to != edges[i].to);
116    }
117
118    sort(events.begin(), events.end());
119    stack< pair<int, int> > S;
120
121    for (auto &event : events)
122    {
123        if (event.opening)
124            S.emplace(event.edge.from, event.edge.index);
125        else
126        {
127            auto last_erased = make_pair(-1, -1);
128
129            while (!S.empty() && S.top().first > event.edge.from)
130            {
131                last_erased = S.top();
132                S.pop();
133                conflict[event.edge.index].push_back(last_erased.second);
134                conflict[last_erased.second].push_back(event.edge.index);
135            }
136
137            if (!S.empty() &&
138                S.top() == make_pair(event.edge.from, event.edge.index))
139            {
140                S.pop();
141            }
142
143            if (last_erased.first != -1)
144                S.emplace(last_erased);
145        }
146    }
147
148    vector<int> colors(M, 0);
149    for (int i = 0; i < M; ++i)
150        if (colors[i] == 0)
151        {
152            colors[i] = 1;
153            dfs(conflict, colors, i);
154            ++answer;

```

```

155         }
156
157     if (type == 1)
158     {
159         for (auto &edge : edges)
160             cout << edge.from << " " << edge.to << " "
161             << colors[edge.index] << "\n";
162     }
163     else
164     {
165         int as_power = 1;
166         for (int i = 0; i < answer; ++i)
167             as_power = (2 * as_power) % kModulo;
168         cout << as_power << "\n";
169     }
170 }
```

Listing 2.3.3: aquapark75_SzZ.cpp

```

1 #include <iostream>
2 #include <algorithm>
3 #define mod 666013;
4
5 using namespace std;
6
7 ifstream fin("aquapark.in");
8 ofstream fout("aquapark.out");
9
10 int caz,a[1000],k,n,m,x,y,p;
11 bool viz[4000];
12
13 struct muchie
14 {
15     int x,y,cost;
16 } b[4000];
17
18 bool intersect(muchie a, muchie b)
19 {
20     if (a.x < b.x and b.x<a.y and a.y<b.y)
21         return true;
22     if (b.x<a.x and a.x<b.y and b.y<a.y)
23         return true;
24     return false;
25 }
26
27 bool comp(muchie a, muchie b)
28 {
29     if (a.x<b.x)
30         return true;
31     else if (a.x==b.x and a.y<b.y)
32         return true;
33     else
34         return false;
35 }
36
37 void tipar(muchie b[], int k)
38 {
39     for (int i=1; i<=k; ++i)
40         fout<<b[i].x<<" "<<b[i].y<<" "<<b[i].cost<<"\n";
41 }
42
43 int main()
44 {
45     p=1;
46     fin>>caz>>n>>m;
47     k=0;
48     for (int i=1; i<=m; ++i)
49     {
50         fin>>x>>y;
51         if (x>y) swap(x,y);
52         b[++k].x=x;
53         b[k].y=y;
54     }
55
56     sort(b+1,b+k+1,comp);
```

```

57     int st=1,dr=0;
58     bool ok;
59     int c[4000];
60     for(int i=1; i<=k; i++)
61         if (!viz[i])
62         {
63             ok=false;
64             ++dr;
65             viz[i]=true;
66             c[dr]=i;
67             b[i].cost=1;
68             if (b[i].x==b[i+1].x and b[i].y==b[i+1].y)
69                 /// daca avem doua muchii identice atunci
70                 /// una va fi interna cealalta externa
71                 {
72                     ++dr;
73                     viz[i+1]=true;
74                     c[dr]=i+1;
75                     b[i+1].cost=2;
76                     st+=2;
77                 }
78         }
79         if (st<=dr) ok=true;
80     while (st<=dr)
81     {
82         for (int j=1; j<=k; j++)
83             if (!viz[j])
84                 if (intersect(b[c[st]],b[j]))
85                 {
86                     ++dr;
87                     viz[j]=true;
88                     c[dr]=j;
89                     b[c[dr]].cost=3-b[c[st]].cost;
90                     /// muchia ce se intersecteaza va fi
91                     /// pe malul celalalt
92                 }
93         st++;
94     }
95     if (ok) p=(p*2)% mod;
96 }
97
98 if (caz==2)
99     fout<<p<<"\n";
100 else
101     tipar(b,k);
102 return 0;
103 }
```

2.3.3 *Rezolvare detaliată

Capitolul 3

OJI 2017

3.1 armonica

Problema 1 - armonica

100 de puncte

Spunem că trei numere a , b , c sunt în progresie armonică dacă b este media armonică a numerelor a și c , adică

$$b = \frac{2}{\frac{1}{a} + \frac{1}{c}} = \frac{2ac}{a+c}$$

Cerințe

Cunoscând un număr natural b să se determine toate perechile de numere naturale (a, c) pentru care a , b , c sunt în progresie armonică.

Date de intrare

Fișierul de intrare **armonica.in** conține pe prima linie un număr natural b .

Date de ieșire

Fișierul de ieșire **armonica.out** va conține pe prima linie un număr natural n reprezentând numărul de perechi de numere naturale (a, c) pentru care b este media armonică. Pe următoarele n linii se vor afișa perechile de numere cerute. Astfel fiecare dintre următoarele n linii vor conține căte două numere a și c separate printr-un spațiu cu semnificația că b este medie armonică a numerelor a și c .

Restricții și precizări

- $1 \leq b \leq 1000000000$;
- Pentru teste în valoare de 40 de puncte avem $b \leq 1000000$;
- Perechile de numere din fișierul de ieșire pot fi afișate în orice ordine;
- Dacă b este medie armonică între două numere diferite a și c atunci perechile (a, c) și (c, a) sunt considerate soluții distințe.
- Problema va fi evaluată pe teste în valoare de 90 de puncte.
- Se vor acorda 10 puncte din oficiu.

Exemple

armonica.in	armonica.out	Explicații
3	3 3 3 2 6 6 2	Numărul 3 este medie armonică a numerelor 3 și 3. Avem progresia armonică (3,3,3) Numărul 3 este medie armonică a numerelor 2 și 6. Avem progresiile armonice (2,3,6) și (6,3,2)

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

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

Dimensiune maximă a sursei: **25 KB**

3.1.1 Indicații de rezolvare

Prof. Adrian Panaete - Colegiul Național "A.T.Laurian" Botoșani.

Se observă că pentru (a, b, c) progresie armonică se obține $2ac = b(a + c)$.

Se deduce că $2ac - ba - bc = 0$. Înmulțind cu 2 și adunând b^2 (b la patrat) se deduce:

$$(2a - b)(2c - b) = b^2$$

Dacă b este impar atunci pentru orice descompunere $b^2 = uv$ se obține soluția

$$a = (b + u)/2c = (b + v)/2$$

Dacă b este par atunci $b = 2k$ și se deduce $(2a - 2k)(2c - 2k) = 4 * k^2$ adică $(a - k)(c - k) = k^2$.

În acest caz pentru orice decompunere $k^2 = uv$ se obține soluția

$$a = k + u, c = k + v.$$

Observație finală: numărul de soluții este numarul divizorilor lui b^2 pentru b impar respectiv numarul divizorilor lui $(b/2)^2$ pentru b par.

Soluția 2: Bazată pe observație directă (Szabo Zoltan)

Cu o sursă brută generăm primele soluții ale problemei.

Se observă cu ușurință că perechile (a, c) din fiecare soluție sunt fie multipli unul la celălalt, fie formează un raport al cărui numărator și numitor are legătură cu divizorii numărului b .

Astfel fractia a/c va putea avea toate posibilitățile de valori p/q unde p și q sunt divizori ai lui b , iar p/q este fracție ireductibilă.

Din generarea soluțiilor se observă că pentru b număr impar algoritmul va genera soluții pentru divizorii lui b . Iar pentru b număr par, algoritmul va genera soluții pentru divizorii lui $b/2$.

Pentru fiecare astfel de fracție găsită cu p și q cunoscute avem pe deosebire ecuația $a/c = p/q$, sau $a * q = p * c$.

Pe de altă parte b este număr cunoscut cu

$$b = 2ac/(a + c)$$

de unde, rezolvând sistemul de ecuații, obținem soluțiile:

$$a = b(p + q)/(2 * q) \text{ respectiv } c = b(p + q)/2q$$

Observăm că fiecare pereche (a, c) se poate tipări și ca pereche (c, a) , dacă a este diferit de c .

De aceasta ținem cont în memorarea soluțiilor.

De exemplu pentru $b = 15 = 3 * 5$ avem următoarele 5 rapoarte pentru a/c : 1, 3, 5, 15, 3 / 5.
De aici vor rezulta 9 soluții distințe.

Cele 9 soluții ale problemei sunt:

solutie	raport
15 15	1/1
10 30	1/3
30 10	3/1
9 45	1/5
45 9	5/1
8 120	1/15
120 8	15/1
20 12	5/3
12 20	3/5

Complexitatea algoritmului este $O(\sqrt{b})$.

3.1.2 Cod sursă

Listing 3.1.1: armonica_eugen.cpp

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 ifstream f("armonica.in");
5 ofstream g("armonica.out");
6
7 long long c, a, b, A[10001], C[10001];
8
9 int main()
10 {
11     int i, k = 0;
12
13     for (int i = 1; i <= 10000; i++)
14     {
15         for (int j = 1; j <= 10000; j++)
16         {
17             if (i * j == 15)
18             {
19                 cout << i << " " << j << endl;
20             }
21         }
22     }
23 }
```

```

13     f >> b;
14     for(a=1; a<=b; ++a)
15     {
16         if (2*a - b > 0)
17             if ((b*a) % (2*a-b) == 0)
18             {
19                 c = (b*a) / (2*a - b);
20                 if (c <= 0) break;
21                 if (a == c) A[++k] = a, C[k] = a;
22                     else A[++k] = a, C[k] = c, A[++k] = c, C[k] = a;
23             }
24     }
25     g << k << endl;
26     for(i=1; i<=k; ++i)
27         g << A[i] << " " << C[i] << endl;
28     return 0;
29 }
```

Listing 3.1.2: armonica_eugen0.cpp

```

1 //100 puncte
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 ifstream f("armonica.in");
6 ofstream g("armonica.out");
7
8 long long a, b, c;
9 vector <long long > A;
10
11 void divizori(long long x)
12 {
13     long long d = 2, y = x*x, p, z = x;
14     A.push_back(1);
15     while(x > 1)
16     {
17         int e = 0;
18         while(x % d == 0)
19         {
20             ++e;
21             x /= d;
22         }
23         p = 1;
24         for(int i=1; i<=2*e; ++i)
25         {
26             p *= d;
27             if (p <= z)
28                 if ((y % p == 0))
29                 {
30                     if (binary_search(A.begin(), A.end(), p)) A.push_back(p);
31                 }
32         }
33         int m = A.size();
34         while(p > 1)
35         {
36             for(int i=0; i < m; ++i)
37             {
38                 a = A[i] * p;
39                 if (a > 0 && a <= z)
40                     if (y%a == 0) A.push_back(a);
41             }
42             p /= d;
43         }
44         d++;
45         if (d*d > x) d = x;
46     }
47 }
48
49 int main()
50 {
51     int i, k, ok = 0;
52     long long K;
53     f >> b;
54     if (b%2 == 0)
```

```

56     {
57         ok = 1;
58         K = b/2;
59         divizori(K);
60     }
61     else divizori(b);
62
63     k = A.size();
64     g << 2*k - 1 << '\n';
65     for(i=0; i<(int)A.size(); ++i){
66         if (ok)
67         {
68             a = K + A[i];
69             c = K + (K*K) / A[i];
70             g << a << " " << c << '\n';
71             if (a != c) g << c << " " << a << '\n';
72         }
73         else
74         {
75             a = (b + A[i]) / 2;
76             c = (b + (b*b) / A[i]) / 2;
77             g << a << " " << c << '\n';
78             if (a != c) g << c << " " << a << '\n';
79         }
80     }
81
82     return 0;
83 }
```

Listing 3.1.3: armonica_eugen1.cpp

```

1 //40 puncte
2 #include <bits/stdc++.h>
3
4 using namespace std;
5
6 ifstream f("armonica.in");
7 ofstream g("armonica.out");
8
9 long long c, a, b, A[10001], C[10001];
10
11 int main()
12 {
13     int i, k = 0;
14
15     f >> b;
16     for(a=1; a<=b; ++a)
17     {
18         if (2*a - b > 0)
19             if ((b*a) % (2*a-b) == 0)
20             {
21                 c = (b*a) / (2*a - b);
22                 if (c <= 0) break;
23                 if (a == c) A[++k] = a, C[k] = a;
24                 else A[++k] = a, C[k] = c, A[++k] = c, C[k] = a;
25             }
26     }
27
28     g << k << endl;
29     for(i=1; i<=k; ++i)
30         g << A[i] << " " << C[i] << endl;
31     return 0;
32 }
```

Listing 3.1.4: armonica_eugen2.cpp

```

1 //40 puncte
2 #include <bits/stdc++.h>
3
4 using namespace std;
5
6 ifstream f("armonica.in");
7 ofstream g("armonica.out");
8
```

```

9  long long a, b, c;
10 vector <long long> A;
11
12
13 void divizori(long long x)
14 {
15     for(long long d=1; d*d <= x; ++d)
16         if (x % d == 0)
17         {
18             A.push_back(d);
19             if (x/d <= b && x/d != d) A.push_back(x/d);
20         }
21 }
22
23 int main()
24 {
25     int i, k, ok = 0;
26     long long K;
27
28     f >> b;
29     if (b%2 == 0)
30     {
31         ok = 1;
32         K = b/2;
33         b /= 2;
34         divizori(K*K);
35     }
36     else divizori(b*b);
37
38     k = A.size();
39     g << 2*k - 1 << '\n';
40     for(i=0; i<(int)A.size(); ++i){
41         if (ok)
42         {
43             a = K + A[i];
44             c = K + (K*K) / A[i];
45             g << a << " " << c << '\n';
46             if (a != c) g << c << " " << a << '\n';
47         }
48         else
49         {
50             a = (b + A[i]) / 2;
51             c = (b + (b*b) / A[i]) / 2;
52             g << a << " " << c << '\n';
53             if (a != c) g << c << " " << a << '\n';
54         }
55     }
56
57     return 0;
58 }
```

Listing 3.1.5: armonica_VG.cpp

```

1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 int n;
7 vector<long long> divisors;
8 vector<int> primes, powers;
9
10 void back(int pos, long long val)
11 {
12     if (pos == primes.size())
13     {
14         divisors.push_back(val);
15         return;
16     }
17
18     for (int i = 0; i <= powers[pos]; ++i)
19     {
20         back(pos + 1, val);
21         val *= primes[pos];
22     }
}
```

```

23 }
24
25 void getDivisorsOfN2(long long n)
26 {
27     for (int i = 2; 1LL * i * i <= n; ++i)
28         if (n % i == 0)
29         {
30             int power = 0;
31             while (n % i == 0)
32             {
33                 n /= i;
34                 power += 2;
35             }
36             primes.push_back(i);
37             powers.push_back(power);
38         }
39
40     if (n > 1)
41     {
42         primes.push_back(n);
43         powers.push_back(2);
44     }
45
46     back(0, 1);
47 }
48
49 int main()
50 {
51     freopen("armonica.in", "r", stdin);
52     freopen("armonica.out", "w", stdout);
53
54     scanf("%d", &n);
55
56     if (n%2)
57     {
58         getDivisorsOfN2(n);
59         printf("%d\n", divisors.size());
60         for (auto dv : divisors)
61         {
62             long long d1 = dv;
63             long long d2 = 1LL*n*n/dv;
64
65             printf("%lld %lld\n", (n+d1)/2, (n+d2)/2);
66         }
67     }
68     else
69     {
70         getDivisorsOfN2(n/2);
71         printf("%d\n", divisors.size());
72         for (auto dv : divisors)
73         {
74             long long d1 = dv;
75             long long d2 = 1LL*n*n/4/dv;
76
77             printf("%lld %lld\n", n/2+d1, n/2+d2);
78         }
79     }
80
81     return 0;
82 }
```

Listing 3.1.6: armonica_Zoli.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 ifstream fin ("armonica.in");
6 ofstream fout("armonica.out");
7
8 struct fractie
9 {
10     long long x,y;
11 };
12
```

```

13 int main()
14 {
15     long long b,d,bb,i,nv,sf,p;
16     fractie v[10000];
17
18     fin>>b;
19     p=1;
20     v[0].x=v[0].y=1;
21     nv=1;
22
23     d=2;bb=b;
24     if (bb%2==0) bb/=2;
25     while(bb>1)
26     {
27         p=1;
28         while (bb%d==0)
29         {
30             p*=d;
31             bb/=d;
32         }
33
34         sf=nv;
35         while (p>1)
36         {
37             v[sf].x=v[0].x*p;
38             v[sf].y=v[0].y;
39             sf++;
40             for (i=1;i<nv;++i)
41             {
42                 v[sf].x=v[i].x*p;
43                 v[sf].y=v[i].y;
44                 sf++;
45                 v[sf].x=v[i].x;
46                 v[sf].y=v[i].y*p;
47                 sf++;
48             }
49             p/=d;
50         }
51     }
52
53     d++;
54     if (d*d>bb) d=bb;
55     nv=sf;
56 }
57
58 fout<<2*nv-1<<"\n";
59 fout<<b<<" "<<b<<"\n";
60
61 for (i=1;i<nv;++i)
62 {
63     fout<<b*(v[i].x+v[i].y)/(2*v[i].x)<<" "
64     <<b*(v[i].x+v[i].y)/(2*v[i].y)<<' \n';
65     fout<<b*(v[i].x+v[i].y)/(2*v[i].y)<<" "
66     <<b*(v[i].x+v[i].y)/(2*v[i].x)<<' \n';
67 }
68
69 return 0;
70 }
```

Listing 3.1.7: armonicaPA.cpp

```

1 #include <bits/stdc++.h>
2
3 #define tip long long
4
5 using namespace std;
6
7 ifstream f("armonica.in");
8 ofstream g("armonica.out");
9
10 tip b;
11
12 void factorizare(tip),bkt(tip,tip);
13
14 vector<pair<tip,tip>>F,sol;
```

```

15 vector<pair<int,int>> V;
16
17 int main()
18 {
19     f>>b;
20     if(b%2)
21         factorizare(b);
22     else
23         factorizare(b/2);
24
25     bkt(0,1);
26
27     g<<sol.size()<<'\\n';
28     for(auto it:sol)
29         g<<it.first<<' ' <<it.second<<'\\n';
30
31     return 0;
32 }
33
34 void factorizare(tip k)
35 {
36     for(tip i=2;i*i<=k;i++)
37         if(k%i==0)
38     {
39         tip e=0;
40         while(k%i==0)
41         {
42             e++;
43             k/=i;
44         }
45         F.push_back({i,2*e});
46     }
47
48     if(k>1)
49         F.push_back({k,2});
50 }
51
52 void bkt(tip p,tip d)
53 {
54     if(p==F.size())
55     {
56         tip a,c,x;
57         if(b%2)
58         {
59             x=b*b/d;
60             a=(b+d)/2;
61             c=(b+x)/2;
62         }
63         else
64         {
65             x=b*b/4/d;
66             a=b/2+d;
67             c=b/2+x;
68         }
69
70         //g<<d<<' ' <<x<<'\\n';
71         sol.push_back({a,c});
72         return;
73     }
74
75     tip i=F[p].first,j=F[p].second,m=1;
76
77     for(tip q=0;q<=j;q++)
78     {
79         bkt(p+1,d*m);
80         m*=i;
81     }
82 }
```

Listing 3.1.8: armonicaPAbrut.cpp

```

1 #include <bits/stdc++.h>
2
3 #define tip long long
4
```

```

5  using namespace std;
6
7  ifstream f("armonica.in");
8  ofstream g("armonica.out");
9
10 tip b,i,j,A[20000],B[20000];
11
12 int main()
13 {
14     f>>b;
15     for(i=b-1;2*i-b>0;i--)
16         if((i*b)%(2*i-b)==0)
17             A[++j]=i;
18
19     g<<2*j+1<<'\\n';
20     g<<b<<' ' <<b<<'\\n';
21
22     for(i=1;i<=j;i++)
23     {
24         g<<A[i]<<' ' <<b*A[i]/(2*A[i]-b)<<'\\n';
25         g<<b*A[i]/(2*A[i]-b)<<' ' <<A[i]<<'\\n';
26     }
27
28     return 0;
29 }
```

3.1.3 *Rezolvare detaliată

3.2 ninjago

Problema 2 - ninjago

100 de puncte

După ce eroii ninja l-au învins pe Nadakhan, de ziua celor dispăruti Zane trebuia să păzească cele n păpuși din muzeu. Între aceste păpuși există m coridoare pe care se poate circula în ambele sensuri. Se garantează faptul că pe cele m coridoare Zane poate ajunge la fiecare dintre cele n păpuși.

Skulkiu, având la dispoziție 5 tipuri de obstacole A, B, C, D, E , încearcă să-l opreasă pe Zane punând pe fiecare corridor câte 4 obstacole.

Zane poate distrugă obstacolele de tip A, B, C și D , dar nu poate să distrugă obstacolele de tipul E . Pentru a distrugă un obstacol de tipul A arma lui Zane are nevoie de 1 unitate de energie, pentru a distrugă un obstacol de tipul B de 2 unități de energie, pentru a distrugă un obstacol de tipul C de 3 unități de energie, iar pentru a distrugă un obstacol de tipul D de 4 unități de energie.

Datorită dispozitivului cu care Skulkiu amplasează obstacolele pe corridor, cele patru obstacole de pe același corridor au o adâncime din ce în ce mai mare, ceea ce implică faptul că pentru a distrugă al doilea obstacol amplasat pe corridor este nevoie de 5 ori mai multă energie decât cea obișnuită, pentru a distrugă cel de-al treilea obstacol amplasat pe corridor este nevoie de 25 ori mai multă energie decât cea obișnuită, iar pentru a distrugă al patrulea obstacol amplasat pe același corridor este nevoie de 125 de ori mai multă energie decât cea obișnuită.

Indiferent de sensul de parcurgere al corridorului de către Zane pentru a înlătura obstacolele, energia consumată este aceeași, aceasta depinzând doar de ordinea în care au fost amplasate obstacolele de către Skulkiu.

Zane nu va înlătura obstacolele de pe toate coridoarele ci doar strictul necesar pentru a avea acces la fiecare păpușă.

Zane dorește să-i lase pe ceilalți ninja să se antreneze aşa că face în aşa fel încât ajutorul pentru distrugerea obstacolelor de tip E să fie minim și apoi ca el să utilizeze un număr minim de unități de energie.

Pentru coridoarele pe care se află obstacole de tip E Zane consumă energie doar pentru obstacolele de tip A, B, C și D . Inițial Zane se află lângă păpușa 1.

Cerințe

- 1) Precizați la câte dintre cele n păpuși poate ajunge Zane înainte de a cere ajutorul celorlalți ninja.
- 2) Precizați pentru eliberarea cător coridoare trebuie să ceară ajutor extern pentru a reuși să ajungă la toate cele n păpuși și câte obstacole de tip E sunt în total pe aceste coridoare.
- 3) Precizați care este numărul minim de unități de energie utilizate.

Date de intrare

Fișierul **ninjago.in** conține pe prima linie un număr natural v care poate avea doar valorile 1, 2 sau 3 reprezentând cerința care va fi rezolvată. Fișierul **ninjago.in** conține pe a doua linie numerele naturale n și m separate printr-un spațiu, iar pe următoarele m linii pentru fiecare koridor două numere naturale separate printr-un spațiu reprezentând cele două păpuși între care se circulă pe koridorul respectiv urmate de un spațiu și patru litere corespunzătoare celor patru tipuri de obstacole în ordinea în care Skulkiu le-a amplasat pe koridorul respectiv. Între cele patru litere nu se află nici un spațiu.

Date de ieșire

Dacă valoarea lui v este 1 atunci fișierul de ieșire **ninjago.out** va conține pe prima linie numai numărul păpușilor la care poate ajunge Zane înainte de a cere ajutorul celorlalți ninja.

Dacă valoarea lui v este 2 atunci fișierul de ieșire **ninjago.out** va conține pe prima linie numărul de coridoare pe care nu le poate elibera singur și pe a doua linie numărul total de obstacole de tip E de pe aceste coridoare.

Dacă valoarea lui v este 3 atunci fișierul de ieșire **ninjago.out** va conține pe prima linie numai numărul minim de unități de energie utilizate.

Restricții și precizări

- $1 \leq N, M \leq 31200$;
- Pentru rezolvarea corectă a primei cerințe se acordă 20 de puncte;
- Pentru rezolvarea corectă a celei de-a două cerințe se acordă 40 de puncte;
- Pentru rezolvarea corectă a celei de-a treia cerințe se acordă 30 de puncte.
- Problema va fi evaluată pe teste în valoare de 90 de puncte.
- Se vor acorda 10 puncte din oficiu.

Exemple

ninjago.in	ninjago.out	Explicații
1 9 15	5	Zane va putea ajunge la nodurile 1, 2, 5, 6, 7 eliberând singur coridoarele (1,2), (1,5), (2,7) și (6,7)
1 2 CBAA		
1 5 ABAA		
2 6 CBEA		
2 7 ACBA		
2 5 CBEA		
3 4 ABAA		
3 7 AECE		
3 8 CBAC		
3 9 ECEE		
4 8 DBAD		
4 9 ECEB		
5 6 CBAD		
5 7 BAAD		
6 7 CBAA		
7 8 ECEB		

2 9 15 1 2 CBAA 1 5 ABAA 2 6 CBEA 2 7 ACBA 2 5 CBEA 3 4 ABAA 3 7 AECE 3 8 CBAC 3 9 ECEE 4 8 DBAD 4 9 ECEB 5 6 CBAD 5 7 BAAD 6 7 CBAA 7 8 ECEB	2 4	Zane trebuie să ceară ajutor pentru eliberarea a 2 coridoare pentru a putea ajunge la toate cele 9 păpuși Zane are nevoie de ajutor pentru a elibera coridoarele (3,7) și (4,9). Pe fiecare dintre aceste 2 coridoare se află câte 2 obstacole de tip E, deci în total 4 obstacole de tip E.
3 9 15 1 2 CBAA 1 5 ABAA 2 6 CBEA 2 7 ACBA 2 5 CBEA 3 4 ABAA 3 7 AECE 3 8 CBAC 3 9 ECEE 4 8 DBAD 4 9 ECEB 5 6 CBAD 5 7 BAAD 6 7 CBAA 7 8 ECEB	1593	Zane va consuma minim 1593 de unități de energie astfel: 163 pentru corridorul (1,2) 161 pentru corridorul (1,5) 191 pentru corridorul (2,7) 161 pentru corridorul (3,4) 76 pentru corridorul (3,7) 413 pentru corridorul (3,8) 265 pentru corridorul (4,9) 163 pentru corridorul (6,7) Pentru corridorul (4,9) obstacolele sunt ECEB, Deci Zane va consuma $0+3*5+0*25+2*125=265$ unități de energie

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

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

Dimensiune maximă a sursei: **25 KB**

3.2.1 Indicații de rezolvare

Prof. Mureșan Codruța - Colegiul Național "Horea, Cloșca și Crișan" Alba Iulia

Rezolvarea problemei presupune determinarea *arborelui parțial de cost minim* într-un graf neorientat cu n noduri și m muchii.

Costurile muchiilor vor fi numere naturale scrise în baza 5 formate din 4 sau 5 cifre, dar reținute în baza 10.

În momentul citirii datelor se transformă codificarea obstacolelor în numere naturale astfel: pentru fiecare dintre cele 4 obstacole pe poziția corespunzătoare se înlocuiesc literele A , B , C și D cu cifrele 1, 2, 3, 4, iar pentru muchiile corespunzătoare coridoarelor care conțin obstacole de tip E se asociază ca și cost un număr natural scris în baza 5 format din 5 cifre, prima cifra reprezentând numărul de obstacole de tip E , iar cifra corespunzătoare obstacolului de tip E se înlocuiește cu 0.

De exemplu pentru secvența de obstacole *ECEB* se va genera numărul 22030 scris în baza 5, deci costul reprezentat în baza 10 va fi 1515. Transformarea numărului format din ultimele 4 cifre din baza 5 în baza 10 (*restul împărțirii* numărului reținut la 625) va reprezenta numărul unităților de energie cosumate de arma lui Zane pentru a distruga obstacolele de pe corridorul corespunzător, iar *câțul* împărțirii costului la 6254 va reprezenta numărul de obstacole de tip *E*.

Cu aceste codificări cel mai mic cost posibil va fi 156 ($0AAAA \Rightarrow 01111 \Rightarrow 1 * 5^0 + 1 * 5^1 + 1 * 5^2 + 1 * 5^3 + 0 * 5^4 = 1 + 5 + 25 + 125 + 0 = 156$), iar cel mai mare cost posibil va fi 2500 ($4EEEE \Rightarrow 40000 \Rightarrow 0 * 5^0 + 0 * 5^1 + 0 * 5^2 + 0 * 5^3 + 4 * 5^4 = 4 * 625 = 2500$). Astfel se poate evita *ordonarea* muchiilor păstrând muchiile într-un tablou de *liste simplu înláñtuite* a cărui indice reprezintă costul muchiilor. Parcurgând acest vector vom avea muchiile în ordinea de care avem nevoie: cele care nu conțin obstacole de tip E la început ordonate crescător după cost (de la 156 la 624), iar la sfârșit cele care conțin obstacole de tip E ordonate după numărul de obstacole de tip E și apoi după cost.

Se începe construirea *arborelui parțial de cost minim* folosind *algoritmul lui Kruskal cu păduri de mulțimi disjuncte*, împărțit în două etape: muchiile fără obstacole de tip E și apoi muchiile ce conțin obstacole de tip E . După prima etapă se analizează muchiile alese în arboreal parțial de cost minim și se pot da ușor și rapid răspunsurile:

- la câte dintre cele n papuși poate ajunge Zane înainte de a cere ajutorul celorlalți ninja
- pentru eliberarea cător coridoare trebuie să ceară ajutor extern pentru a reuși să ajunga la toate cele n papuși

Apoi se termină determinarea arborelui parțial de cost minim folosind muchii ce conțin obstacole de tip E , calculând simultan numărul total de obstacole de tip E . Numărul minim de unități de energie utilizate se calculează în timp ce se construiesc arborele parțial de cost minim în cele două etape.

3.2.2 Cod sursă

Listing 3.2.1: eric_ninjago.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 int getCost(char c)
6 {
7     return c - 'A' + 1;
8 }
9
10 int getCost(string s)
11 {
12     int ret = 0;
13     int p = 1;
14
15     for(int i = 0; i < s.length(); ++i, p *= 5)
16     {
17         if (s[i] == 'E')
18             continue;
19         ret += p * getCost(s[i]);
20     }
21
22     return ret;
23 }
24
25 struct Edge
26 {
27     int f, t;
28     int cost;
29     int E;
30
31     Edge(int f, int t, string str):
32         f(f),
33         t(t),
34         cost(getCost(str)),
35         E(0)
36     {
37         for(auto c: str)
38             E += (c == 'E');
39     }
40
41     pair<int, int> asECost() const
42     {
43         return {E, cost};
44     }

```

```

45  };
46
47 int n, m;
48
49 vector<Edge> E;
50
51 vector<int> under;
52 vector<int> TT;
53
54 int getUp(int nod)
55 {
56     if (TT[nod] == nod)
57         return nod;
58     else
59         return TT[nod] = getUp(TT[nod]);
60 }
61
62 bool unite(int x, int y)
63 {
64     x = getUp(x);
65     y = getUp(y);
66
67     if (x == y)
68         return false;
69
70     if (under[x] >= under[y])
71     {
72         TT[y] = x;
73         under[x] += under[y];
74         return true;
75     }
76     else
77         return unite(y, x);
78 }
79
80 int countReachableFrom1()
81 {
82     int ret = 0;
83     for (int i = 1; i <= n; ++i)
84         ret += (getUp(i) == getUp(1));
85
86     return ret;
87 }
88
89 int main()
90 {
91     ifstream in("ninjago.in");
92     int v;
93
94     in >> v;
95     in >> n >> m;
96     E.reserve(m);
97
98     under.resize(n + 1);
99     TT.resize(n + 1);
100    for(int i = 1; i <= n; ++i)
101    {
102        under[i] = 1;
103        TT[i] = i;
104    }
105
106    for (int i = 0; i < m; ++i)
107    {
108        int f, t;
109        string s;
110        in >> f >> t >> s;
111
112        assert(1 <= f && f <= n);
113        assert(1 <= t && t <= n);
114
115        assert(s.length() == 4);
116
117        for (auto c: s)
118            assert(strchr("ABCDE", c) != NULL);
119
120        E.push_back({f, t, s});

```

```

121     }
122     in.close();
123
124     sort(E.begin(), E.end(), [] (const Edge &a, const Edge &b) -> bool
125     {
126         return a.asECost() < b.asECost();
127     });
128
129     int energyUsed = 0;
130
131     size_t at = 0;
132     while (at < E.size() && E[at].E == 0)
133     {
134         if (unite(E[at].f, E[at].t))
135             energyUsed += E[at].cost;
136
137         at++;
138     }
139
140
141     const int from1 = countReachableFrom1();
142
143     int badEdges = 0;
144     int takenEs = 0;
145     while (at < E.size())
146     {
147         bool used = unite(E[at].f, E[at].t);
148
149         badEdges += used;
150         takenEs += E[at].E * used;
151         energyUsed += used * E[at].cost;
152         at++;
153     }
154
155     ofstream out("ninja.out");
156
157     switch (v)
158     {
159         case 1:
160             out << from1 << "\n";
161             break;
162         case 2:
163             out << badEdges << "\n" << takenEs << "\n";
164             break;
165         case 3:
166             out << energyUsed << "\n";
167             break;
168         default:
169             assert(false);
170             break;
171     }
172
173     return 0;
174 }
```

Listing 3.2.2: fastlikeaninja_PA.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 ifstream f("ninja.in");
6 ofstream g("ninja.out");
7
8 const int N=31210;
9 int q,n,m,x,y,i,j,k,p,e,C,c,G,O,rx,ry,sol,root(int),DFS(int),t[N];
10 string s;
11 vector<pair<int,int>> M[5][N];
12 vector<int> v[N];
13
14 void APM();
15 bitset<40000> viz;
16
17 int main()
18 {
```

```

19     f>>q>>n>>m;
20     if(q==1)
21     {
22         for(;m;m--)
23         {
24             f>>x>>y>>s;
25             if(!strchr(s.c_str(),'E'))
26             {
27                 v[x].push_back(y);
28                 v[y].push_back(x);
29             }
30         }
31     }
32     g<<DFS(1);
33     return 0;
34 }
35
36 for(; m; m--)
37 {
38     f>>x>>y>>s;
39     p=1,e=0,c=0;
40     for(j=0; j<4; j++,p*=5)
41     {
42         if(s[j]=='E')
43             s[j] ='A'-1,e++;
44         c+=p*(s[j]-'A'+1);
45     }
46
47     M[e][c].push_back({x,y});
48 }
49
50 G=0,O=0,C=0;
51 for(i=1; i<=n; i++)
52     t[i]=i;
53 for(n--,e=0; e<=4&&n; e++)
54     for(j=e>0,c=0; c<625&&n; c++)
55         for(k=M[e][c].size()-1;k>=0&&n;k--)
56             if((x=root(M[e][c][k].first)) != (y=root(M[e][c][k].second)))
57                 t[x]=y,C+=c,G+=j,O+=e,n--;
58     q==2 ? g<<G<<' \n'<<O : g<<C;
59     return 0;
60 }
61
62 int DFS(int nod)
63 {
64     int ret=1;
65     viz[nod]=1;
66     for(auto vec:v[nod])
67         if(!viz[vec])
68             ret+=DFS(vec);
69
70     return ret;
71 }
72
73 int root(int nod)
74 {
75     return t[nod]==nod ? nod : t[nod]=root(t[nod]);
76 }

```

Listing 3.2.3: ninjago_eugen.cpp

```

1 //Kruskal cu pad. disj.
2 #include <bits/stdc++.h>
3
4 using namespace std;
5
6 #define NM 31210
7
8 ifstream f("ninjago.in");
9 ofstream g("ninjago.out");
10
11 struct edge
12 {
13     int x, y, c;
14     char e;

```

```

15     friend bool operator <(edge a, edge b) {
16         return a.c < b.c;
17     }
18 };
19
20 vector <int> G1[NM];
21 vector <edge> G, sol;
22
23 int n, m, n1, ctotal;
24 bool viz[NM];
25 int t[NM];
26
27 void DFS(int nod)
28 {
29     n1++;
30     for(int i=0; i < (int)G1[nod].size(); ++i)
31     if (! viz[G1[nod][i]]) {
32         viz[G1[nod][i]] = 1;
33         DFS(G1[nod][i]);
34     }
35 }
36
37 int Radacina(int i)
38 {
39     int k = i;
40     while (t[k] > 0)
41         k = t[k];
42
43     while (t[i] > 0)
44     {
45         t[i] = k;
46         i = t[i];
47     }
48
49     return i;
50 }
51
52 void Unifica(int i, int j)
53 {
54     t[i] += t[j];
55     t[j] = i;
56 }
57
58 void APM()
59 {
60     edge w;
61
62     sort(G.begin(), G.end(), less <edge>());
63
64     for (int i=1; i<=n; ++i)
65         t[i] = -1;
66     for (int k = 0; k < (int)G.size(); ++k)
67     {
68         w = G[k];
69         int rx = Radacina(w.x);
70         int ry = Radacina(w.y);
71         if (rx != ry)
72         {
73             ctotal += (w.c) % 625;
74             sol.push_back(w);
75             Unifica(rx, ry);
76         }
77     }
78 }
79
80 int main()
81 {
82     int i, j, v;
83     edge w;
84     string s;
85
86     f >> v >> n >> m;
87     for(i=1; i<=m; ++i)
88     {
89         f >> w.x >> w.y >> s;
90         int ct = 0, p = 1, e = 0, ok = 1;

```

```

91         for(j=0; j<4; ++j, p *= 5)
92     {
93         if (s[j] == 'E')
94         {
95             ok = 0; e++;
96             ct += 625;
97         }
98         else
99             ct += p * (s[j] - 'A' + 1);
100    }
101
102    w.e = e;
103    w.c = ct;
104    G.push_back(w);
105
106    if (ok)
107    {
108        G1[w.x].push_back(w.y);
109        G1[w.y].push_back(w.x);
110    }
111}
112
113 if (v == 1)
114 {
115     DFS(1);
116     if (n1 > 1) --n1;
117     g << n1 << '\n';
118     return 0;
119 }
120
121 APM();
122
123 if (v == 2)
124 {
125     int obs = 0, nre = 0;
126     for(i=0; i<(int) sol.size(); ++i)
127     {
128         if (sol[i].c >= 625)
129         {
130             obs++;
131             nre += sol[i].e;
132         }
133     }
134
135     g << obs << '\n' << nre << '\n';
136     return 0;
137 }
138
139 if (v == 3)
140 {
141     g << ctotal << '\n';
142     return 0;
143 }
144
145 return 0;
146 }
```

Listing 3.2.4: ninjago_primPA.cpp

```

1 //utilizez algoritmul prim
2 //creez o coada de prioritati in care inserez muchii in ordinea descrescatoare
3 // a costurilor
4 //initial in aceasta coada inserez muchiile care pleaca din 1
5 // aleg primul element din coada de prioritati si
6 // am in general urmatoarele cazuri:
7 // 1. Muchia este intre doua noduri marcate
8 //      -in acest caz elimin muchia din coada fara sa fac nimic
9 // 2. Muchia este intre un nod marcat si unul nemarcat
10 //     -in acest caz marchez nodul nemarcat
11 //     -adun costul acestei muchii la costul APM
12 //     -daca am muchii e adun o unitate la solutia 21 si numarul de e la
13 //       solutia 22
14 //       inserez muchiile din nodul proaspat marcat spre noduri inca nemarcate
15
16 #include <bits/stdc++.h>
```

```

17
18 using namespace std;
19
20 ifstream f("ninjago.in");
21 ofstream g("ninjago.out");
22
23 const int N=31210;
24 const int MASK30=(1<<30)-1;
25 const int MASK16=(1<<16)-1;
26 const int MASK10=(1<<10)-1;
27
28 int c,n,m,x,y,i,gal,obs,cost,sol,GLUE(int,int,int),root(int),used[N],t[N];
29 string s;
30 vector<int> v[N];
31
32 void DFS(int),APM(),edge(),SPLIT(int&,int&,int&,int);
33
34 bitset<40000> viz;
35
36 int main()
37 {
38     for(f>>c>>n>>m, i=1; i<=m; i++)
39         edge();
40     if(c==1)
41     {
42         DFS(1);
43         g<<sol;
44     }
45     else
46         APM();
47
48     return 0;
49 }
50
51 void APM()
52 {
53     priority_queue<int> Q;
54
55     used[1]=1;
56     for(auto it:v[1])
57         Q.push(MASK30-it);
58
59     for(int na=n-1;na;)
60     {
61         int e,cst,y;
62         SPLIT(e,cst,y,MASK30-Q.top());
63         Q.pop();
64         if(!used[y])
65         {
66             if(e) gal++,obs+=e;
67             cost+=cst;
68             used[y]=1;
69             na--;
70             for(auto it:v[y])
71                 if(!used[it&MASK16])
72                     Q.push(MASK30-it);
73         }
74     }
75
76     c==2 ? g<<gal<<'\\n'<<obs : g<<cost;
77 }
78
79 void DFS(int nod)
80 {
81     sol++;
82     viz[nod]=1;
83     for(auto it:v[nod])
84     {
85         int e,cst,vec;
86         SPLIT(e,cst,vec,it);
87         if(!e)
88             if(!viz[vec])
89                 DFS(vec);
90     }
91 }
92

```

```

93 int root(int nod)
94 {
95     return t[nod]==nod ? nod : t[nod]=root(t[nod]);
96 }
97
98 void edge()
99 {
100     int x,y,p=1,e=0,cst=0, j=0;
101     string s;
102
103     for(f>>x>>y>>s, j=0; j<4; j++, p*=5)
104         if(s[j]<'E')
105             cst+=p*(s[j]-'A'+1);
106         else
107             e++;
108
109     v[x].push_back(GLUE(e,cst,y));
110     v[y].push_back(GLUE(e,cst,x));
111 }
112
113 void SPLIT(int &gal,int &cst,int &nod,int val)
114 {
115     gal=val>>26;
116     cst=((val)>>16)&MASK10;
117     nod=val&MASK16;
118 }
119
120 int GLUE(int gal,int cst,int vec)
121 {
122     return (((gal<<10)|cst)<<16)|vec;
123 }
```

Listing 3.2.5: ninjagoBubbleS_Codruta.cpp

```

1 #include <iostream>
2 #include <fstream>
3
4 using namespace std;
5
6 short v;
7 int n,m;
8 struct muchie
9 {
10     int e1,e2,c;
11 };
12
13 muchie g[31202];
14
15 void citire()
16 {
17     ifstream fin("ninjago.in");
18     int i,j,p5;
19     char o;
20     fin >> v;
21     fin >> n >> m;
22
23     for (int i = 1; i <= m; i++)
24         { fin >> g[i].e1 >> g[i].e2;
25             g[i].c=0; p5=1;
26             for (int j = 0; j < 4; j++)
27                 {
28                     fin >> o;
29                     switch (o)
30                     {
31                         case 'A': g[i].c+=p5; break;
32                         case 'B': g[i].c+=2*p5; break;
33                         case 'C': g[i].c+=3*p5; break;
34                         case 'D': g[i].c+=4*p5; break;
35                         case 'E': g[i].c+=625;
36                     }
37                     p5*=5;
38                 }
39             }
40     fin.close();
41 }
```

```

42
43 void Sortare()
44 { int i,mf;
45   muchie aux;
46   do
47   {   mf=0;
48     for (i=1;i<m;i++)
49       if (g[i].c>g[i+1].c)
50         { aux=g[i];
51           g[i]=g[i+1];
52           g[i+1]=aux;
53           mf=1;
54         }
55   } while (mf);
56 }
57
58 int main()
59 { ofstream fout("ninja.out");
60
61   int i,j,nn, ma,obst,vf;
62   long cost;
63   int s[31202];
64   citire();
65   Sortare();
66   ma=0, cost=0, obst=0;
67   for (i = 1; i <= n; i++) s[i]=i;
68   i=1;
69   while (ma<n-1 && g[i].c<625)
70   {
71     if (s[g[i].el]!=s[g[i].e2])
72     {
73       ma++;
74       cost+=g[i].c;
75       vf=s[g[i].e2];
76       for (j = 1; j <= n; j++)
77       {
78         if (s[j]==vf)
79           s[j]=s[g[i].el];
80       }
81     }
82   }
83   i++;
84 }
85
86
87 if (ma==n-1)
88 switch (v)
89 { case 1:
90   fout<<n<<' \n'; break;
91   case 2:
92     fout<<"0" << endl <<"0"; break;
93   case 3: fout << cost;
94   }
95 else
96 {
97   if (v==1)
98   {
99     nn=0;
100
101   for (j = 1; j <= n; j++)
102     if (s[j]==s[1]) nn++;
103
104   fout << nn;
105   }
106 else
107   {
108     nn=n-1-ma;
109     while (ma<n-1)
110     {
111       if (s[g[i].el]!=s[g[i].e2])
112       {
113         ma++;
114         cost+=g[i].c% 625;
115         obst+=g[i].c/625;
116         vf=s[g[i].e2];
117         for (j = 1; j <= n; j++)

```

```

118                     if (s[j]==vf)
119                         s[j]=s[g[i].el];
120                     }
121                     i++;
122                 }
123             if (v==2) fout << nn << endl << obst;
124             else fout << cost;
125         }
126     }
127 }
128 fout.close();
129
130 return 0;
131 }

```

Listing 3.2.6: ninjagoCRadix2.cpp

```

1 #include <iostream>
2 #include <fstream>
3
4 using namespace std;
5
6 short v;
7 int n,m,e1[31202],e2[31202],c[31202];
8
9 void citire()
10 {
11     ifstream fin("ninjago.in");
12     int i,j,p5;
13     char o;
14     fin >> v;
15     fin >> n >> m;
16
17     for (int i = 1; i <= m; i++)
18     { fin >> e1[i] >> e2[i];
19         c[i]=0; p5=1;
20         for (int j = 0; j < 4; j++)
21         {
22             fin >> o;
23             switch (o)
24             {
25                 case 'A': c[i]=c[i]+p5; break;
26                 case 'B': c[i]=c[i]+2*p5; break;
27                 case 'C': c[i]=c[i]+3*p5; break;
28                 case 'D': c[i]=c[i]+4*p5; break;
29                 case 'E': c[i]=c[i]+625;
30             }
31             p5*=5;
32             //cout << endl;
33         }
34     }
35
36     fin.close();
37 }
38
39 void sortare()
40 {
41     int cat[5], b[5];
42     int y[31202],oe1[31202],oe2[31202], poz[31202];
43     short cif, j;
44     int p5;
45     p5=1;
46     for (cif = 1; cif <= 5; cif++)
47     { for (j = 0; j <= 4; j++)
48         cat[j]=0;
49         for (j = 1; j <= m; j++)
50             { cat[(c[j]/p5)%5]++;
51                 poz[j]=cat[(c[j]/p5)%5];
52             }
53         b[0]=0;
54         for (j = 1; j <= 4; j++)
55             b[j]=b[j-1]+cat[j-1];
56         for (j = 1; j <= m; j++)
57             { y[b[(c[j]/p5)%5]+poz[j]]=c[j];

```

```

58         oe1[b[(c[j]/p5)%5]+poz[j]]=e1[j];
59         oe2[b[(c[j]/p5)%5]+poz[j]]=e2[j];
60     }
61     for (j = 1; j <= m; j++)
62     {
63         c[j]=y[j];
64         e1[j]=oe1[j];
65         e2[j]=oe2[j];
66     }
67     p5*=5;
68 }
69 }
70
71 int main()
72 {
73     ofstream fout("ninjago.out");
74
75     int i,j,nn, ma, obst,vf;
76     long cost;
77     int s[31202];
78     citire();
79     sortare();
80     ma=0, cost=0, obst=0;
81     for (i = 1; i <= n; i++) s[i]=i;
82     i=1;
83     while (ma<n-1 && c[i]<625)
84     {
85         if (s[e1[i]]!=s[e2[i]])
86         {
87             ma++;
88             cost+=c[i];
89             vf=s[e2[i]];
90             for (j = 1; j <= n; j++)
91                 if (s[j]==vf)
92                     s[j]=s[e1[i]];
93         }
94         i++;
95     }
96
97     if (ma==n-1)
98         switch (v)
99         {
100             case 1:
101                 fout<<n<<'\\n'; break;
102             case 2:
103                 fout<<"0" << endl <<"0"; break;
104             case 3: fout << cost;
105         }
106     else
107     {
108         if (v==1)
109         {
110             nn=0;
111             for (j = 1; j <= n; j++)
112                 if (s[j]==s[1]) nn++;
113             fout << nn;
114         }
115         else
116         {
117             nn=n-1-ma;
118             while (ma<n-1)
119             {
120                 if(s[e1[i]]!=s[e2[i]])
121                 {
122                     ma++;
123                     cost+=c[i]% 625;
124                     obst+=c[i]/625;
125                     vf=s[e2[i]];
126                     for (j = 1; j <= n; j++)
127                         if (s[j]==vf)
128                             s[j]=s[e1[i]];
129             }
130         }
131         if (v==2) fout <<nn << endl << obst;
132         else fout << cost;;
133 }
```

```

134         }
135     }
136
137     fout.close();
138
139     return 0;
140 }
```

Listing 3.2.7: ninjagoCRadixS_Codruta.cpp

```

1 #include <iostream>
2 #include <fstream>
3
4 using namespace std;
5
6 short v;
7 int n,m,e1[31202],e2[31202],c[31202];
8
9 void citire()
10 {
11     ifstream fin("ninjago.in");
12     int i,j,p5;
13     char o;
14     fin >> v;
15     fin >> n >> m;
16
17     for (int i = 1; i <= m; i++)
18     { fin >> e1[i] >> e2[i];
19         c[i]=0; p5=1;
20         for (int j = 0; j < 4; j++)
21         {
22             fin >> o;
23             switch (o)
24             {
25                 case 'A': c[i]=c[i]+p5; break;
26                 case 'B': c[i]=c[i]+2*p5; break;
27                 case 'C': c[i]=c[i]+3*p5; break;
28                 case 'D': c[i]=c[i]+4*p5; break;
29                 case 'E': c[i]=c[i]+625;
30             }
31             p5*=5;
32             //cout << endl;
33         }
34     }
35     fin.close();
36 }
37
38 void sortare()
39 {
40     int cat[5], b[5];
41     int y[31202],oe1[31202],oe2[31202], poz[31202];
42     short cif, j;
43     int p5;
44     p5=1;
45     for (cif = 1; cif <= 5; cif++)
46     { for (j = 0; j <= 4; j++)
47         cat[j]=0;
48         for (j = 1; j <= m; j++)
49         { cat[(c[j]/p5)%5]++;
50             poz[j]=cat[(c[j]/p5)%5];
51         }
52         b[0]=0;
53         for (j = 1; j <= 4; j++)
54             b[j]=b[j-1]+cat[j-1];
55         for (j = 1; j <= m; j++)
56         { y[b[(c[j]/p5)%5]+poz[j]]=c[j];
57             oe1[b[(c[j]/p5)%5]+poz[j]]=e1[j];
58             oe2[b[(c[j]/p5)%5]+poz[j]]=e2[j];
59         }
60         for (j = 1; j <= m; j++)
61         {
62             c[j]=y[j];
63             e1[j]=oe1[j];
64             e2[j]=oe2[j];
65         }
}
```

```

66             p5*=5;
67         }
68     }
69
70     int main()
71     {
72         ofstream fout("nunjago.out");
73
74         int i,j,nn, ma, obst,vf;
75         long cost;
76         int s[31202];
77
78         citire();
79         sortare();
80
81         ma=0, cost=0, obst=0;
82         for (i = 1; i <= n; i++) s[i]=i;
83
84         i=1;
85         while (ma<n-1 && c[i]<625)
86         {
87             if (s[e1[i]]!=s[e2[i]])
88             {
89                 ma++;
90                 cost+=c[i];
91                 vf=s[e2[i]];
92                 for (j = 1; j <= n; j++)
93                     if (s[j]==vf)
94                         s[j]=s[e1[i]];
95             }
96             i++;
97         }
98
99         if (ma==n-1)
100         switch (v)
101         {
102             case 1:
103                 fout<<n<<'\\n'; break;
104             case 2:
105                 fout<<"0" << endl <<"0"; break;
106             case 3: fout << cost;
107         }
108         else
109         {
110             if (v==1)
111             {
112                 nn=0;
113                 for (j = 1; j <= n; j++)
114                     if (s[j]==s[1]) nn++;
115             }
116             else
117             {
118                 nn=n-1-ma;
119                 while (ma<n-1)
120                 {
121                     if(s[e1[i]]!=s[e2[i]])
122                     {
123                         ma++;
124                         cost+=c[i]% 625;
125                         obst+=c[i]/625;
126                         vf=s[e2[i]];
127                         for (j = 1; j <= n; j++)
128                             if (s[j]==vf)
129                                 s[j]=s[e1[i]];
130                         }
131                     i++;
132                 }
133
134                 if (v==2) fout <<nn << endl << obst;
135                 else fout << cost;;
136             }
137         }
138
139         fout.close();
140
141         return 0;

```

142 }

Listing 3.2.8: ninjagoPA.cpp

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 ifstream f("ninjago.in");
5 ofstream g("ninjago.out");
6
7 const int N=31210;
8
9 int c,n,m,x,y,i,rx,ry,sol,root(int),t[N];
10
11 string s;
12 tuple<int,int,int,int> M[N],edge();
13
14 vector<int> v[N];
15 void DFS(int),APM();
16
17 bitset<40000> viz;
18
19 int main()
20 {
21     for(f>>c>>n>>m,i=1;i<=m;i++) M[i]=edge();
22     if(c==1){DFS(1);g<<sol;} else APM();
23     return 0;
24 }
25
26 void APM()
27 {
28     int x,y,e,cst, gal=0,obs=0,capm=0;
29     string s;
30
31     for(i=1; i<=n; i++)
32         t[i]=i;
33
34     sort(M+1,M+m+1);
35
36     for(n--,i=1;n;i++)
37     {   // http://en.cppreference.com/w/cpp/utility/tuple/tie
38         tie(e,cst,x,y)=M[i];//A std::tuple object containing lvalue references
39         if((rx=root(x))!= (ry=root(y)))
40             t[rx]=ry,capm+=cst,gal+=e>0,obs+=e,n--;
41     }
42
43     c==2 ? g<<gal<<' \n'<<obs : g<<capm;
44 }
45
46 void DFS(int nod)
47 {
48     sol++;
49     viz[nod]=1;
50     for(auto vec:v[nod])
51         if(!viz[vec])
52             DFS(vec);
53 }
54
55 int root(int nod)
56 {
57     return t[nod]==nod ? nod : t[nod]=root(t[nod]);
58 }
59
60 tuple<int,int,int,int> edge()
61 {
62     int x,y,p=1,e=0,cst=0,j=0;
63     string s;
64
65     for(f>>x>>y>>s, j=0; j<4; j++,p*=5)
66         if(s[j]<' E')
67             cst+=p*(s[j]-' A'+1);
68         else
69             e++;
70
71     if(!e)

```

```

72     {
73         v[x].push_back(y);
74         v[y].push_back(x);
75     }
76
77     return make_tuple(e,cst,x,y);
78 }
```

Listing 3.2.9: ninjagoPMD_Codruta.cpp

```

1 #include <iostream>
2 #include <fstream>
3
4 using namespace std;
5
6 short v;
7 int n,m;
8
9 struct nod
10 { int e1,e2;
11   nod* urm;
12 };
13
14 nod* lm[2501];
15
16 int T[31202], H[31202];
17
18 void citire()
19 {
20   ifstream fin("ninjago.in");
21   int i,j,p5,c;
22   char o;
23   nod *p;
24   fin >> v;
25   fin >> n >> m;
26   for (i=156;i<=2500;i++)
27     lm[i]=NULL;
28
29   for (int i = 1; i <= m; i++)
30   { p=new nod;
31     fin >> p->e1 >> p->e2;
32     c=0;
33     p5=1;
34     for (int j = 0; j < 4; j++)
35     {
36       fin >> o;
37       switch (o)
38     {
39       case 'A': c+=p5;break;
40       case 'B': c+=2*p5;break;
41       case 'C': c+=3*p5;break;
42       case 'D': c+=4*p5;break;
43       case 'E': c+=625;
44     }
45     p5*=5;
46   }
47
48   p->urm=lm[c];
49   lm[c]=p;
50 }
51
52 fin.close();
53 }
54
55 int radacina(int x)
56 {
57   int r,y;
58   r=x;
59   while (T[r]!=r)
60     r=T[r];
61   while (T[x]!=x)
62   {
63     y=T[x];
64     T[x]=r;
65     x=y;
```

```

66         }
67     return r;
68 }
69
70 void UnesteArbore(int x, int y)
71 {
72     if (H[x]>H[y])
73         T[y]=x;
74     else T[x]=y;
75
76     if (H[x]==H[y])
77         H[y]++;
78 }
79
80 int main()
81 {   ofstream fout("ninja.out");
82
83     int i,j,nn, ma,obst,r1;
84     long cost;
85     nod *p;
86     citire();
87
88     for (i=1;i<=n;i++)
89     {
90         T[i]=i;
91         H[i]=1;
92     }
93
94     ma=0, cost=0, obst=0;
95
96     i=156;
97     while (ma<n-1 && i<625)
98     {   p=lm[i];
99         while (ma<n-1 && p)
100        {
101            if (radacina(p->e1)!=radacina(p->e2))
102            {
103                ma++;
104                cost+=i;
105                UnesteArbore(radacina(p->e1),radacina(p->e2));
106            }
107            p=p->urm;
108        }
109        i++;
110    }
111
112    if (ma==n-1)
113        switch (v)
114        {   case 1:
115            fout<<n<<' \n'; break;
116            case 2:
117            fout<<"0" << endl <<"0"; break;
118            case 3: fout << cost;
119        }
120    else
121    {
122        if (v==1)
123        {
124            nn=0;
125            r1=radacina(1);
126            for (j = 1; j <= n; j++)
127                if (radacina(j)==r1) nn++;
128            fout << nn;
129        }
130        else
131        {
132            nn=n-1-ma;
133            while (ma<n-1)
134            {
135                p=lm[i];
136                while (ma<n-1 && p)
137                {   if (radacina(p->e1)!=radacina(p->e2))
138                {
139                    ma++;
140                    cost+=i%625;
141                    obst+=i/625;

```

```

142                     UnesteArbore (radacina (p->e1), radacina (p->e2));
143                 }
144             p=p->urm;
145         }
146         i++;
147     }
148
149     if (v==2) fout << nn << endl << obst;
150     else fout << cost;
151 }
152 }
153
154 fout.close();
155
156 return 0;
157 }
```

Listing 3.2.10: ninjagoPrim_mat_Codruta.cpp

```

1 #include <iostream>
2 #include <fstream>
3
4 using namespace std;
5
6 short v;
7 int n,m;
8 int c[2000][2000];
9 const int Infinit = 3200;
10 int nuE,p1,ncE,noE;
11 long cost;
12
13 void citire()
14 {
15     ifstream fin("ninjago.in");
16     int i,j,p5,x,y;
17     char o;
18     fin >> v;
19     fin >> n >> m;
20
21     for(i=1;i<=n;i++)
22         for(j=1;j<=n;j++)
23             if(i==j)
24                 c[i][j]=0;
25             else
26                 c[i][j]=c[j][i]=Infinit;
27
28     for (int i = 1; i <= m; i++)
29     { fin >> x >> y;
30         c[x][y]=0; p5=1;
31         for (int j = 0; j < 4; j++)
32         {
33             fin >> o;
34             switch (o)
35             {
36                 case 'A': c[x][y]+=p5; break;
37                 case 'B': c[x][y]+=2*p5; break;
38                 case 'C': c[x][y]+=3*p5; break;
39                 case 'D': c[x][y]+=4*p5; break;
40                 case 'E': c[x][y]+=625;
41             }
42             p5*=5;
43         }
44
45         c[y][x]=c[x][y];
46     }
47
48     fin.close();
49 }
50
51 void adaug(int x, int y)
52 {
53     if (nuE&& c[x][y]<625) p1++;
54     if (c[x][y]>625)
55     {
```

```

56         nuE=0;
57         ncE++;
58         noE+=c[x][y]/625;
59     }
60     cost+=c[x][y] % 625;
61 }
63
64 void APMPrim()
65 { int i,j,k, Min;
66   int s[2000];
67
68   for(i=2;i<=n;i++)
69     s[i]=1;
70
71   for(k=1;k<=n-1;k++)
72   {
73     Min=Infinit;
74     for(i=1;i<=n;i++)
75       if(s[i])
76         if(Min>c[s[i]][i])
77         {
78           Min=c[s[i]][i];
79           j=i;
80         }
81
82     adaug(s[j],j);
83
84     for(i=1;i<=n;i++)
85       if(s[i] && c[i][s[i]]>c[i][j])
86         s[i]=j;
87
88     s[j]=0;
89   }
90 }
91
92 int main()
93 { ofstream fout("ninjago.out");
94
95   citire();
96   nuE=1;
97   p1=1;
98   ncE=0;
99   noE=0;
100  cost=0;
101  APMPrim();
102
103  switch (v)
104  { case 1:
105    fout<<p1<<'\\n'; break;
106    case 2:
107      fout<<ncE << endl <<noE; break;
108    case 3: fout << cost;
109  }
110
111  fout.close();
112
113  return 0;
114 }
```

Listing 3.2.11: ninjagoQuickS_Codruta.cpp

```

1 #include <iostream>
2 #include <fstream>
3
4 using namespace std;
5
6 short v;
7 int n,m;
8 struct muchie
9 {
10   int e1,e2,c;
11 };
12
13 muchie g[31202];
```

```

14
15 void citire()
16 {
17     ifstream fin("ninjago.in");
18     int i,j,p5;
19     char o;
20     fin >> v;
21     fin >> n >> m;
22
23     for (int i = 1; i <= m; i++)
24     { fin >> g[i].e1 >> g[i].e2;
25         g[i].c=0; p5=1;
26         for (int j = 0; j < 4; j++)
27         {
28             fin >> o;
29             switch (o)
30             {
31                 case 'A': g[i].c+=p5;break;
32                 case 'B': g[i].c+=2*p5;break;
33                 case 'C': g[i].c+=3*p5;break;
34                 case 'D': g[i].c+=4*p5;break;
35                 case 'E': g[i].c+=625;
36             }
37             p5*=5;
38         }
39     }
40     fin.close();
41 }
42
43 void quickSort(int left, int right)
44 {
45     int i = left, j = right;
46     muchie tmp;
47     int pivot = g[(left + right) / 2].c;
48
49     while (i <= j)
50     {
51         while (g[i].c < pivot)
52             i++;
53         while (g[j].c > pivot)
54             j--;
55
56         if (i <= j)
57         {
58             tmp = g[i];
59             g[i] = g[j];
60             g[j] = tmp;
61             i++;
62             j--;
63         }
64     };
65
66     if (left < j)
67         quickSort(left, j);
68     if (i < right)
69         quickSort(i, right);
70 }
71
72 int main()
73 {
74     ofstream fout("ninjago.out");
75
76     int i,j,nn, ma,obst,vf;
77     long cost;
78     int s[31202];
79     citire();
80     quickSort(1,m);
81
82     ma=0, cost=0, obst=0;
83     for (i = 1; i <= n; i++) s[i]=i;
84     i=1;
85     while (ma<n-1 && g[i].c<625)
86     {
87         if (s[g[i].e1]!=s[g[i].e2])
88         {
89             ma++;

```

```

90         cost+=g[i].c;
91         vf=s[g[i].e2];
92         for (j = 1; j <= n; j++)
93         {
94             if (s[j]==vf)
95                 s[j]=s[g[i].e1];
96
97         }
98
99     }
100    i++;
101}
102
103 if (ma==n-1)
104     switch (v)
105     { case 1:
106         fout<<n<<'\\n'; break;
107     case 2:
108         fout<<"0" << endl <<"0"; break;
109     case 3: fout << cost;
110     }
111 else
112 {
113     if (v==1)
114     {
115         nn=0;
116         for (j = 1; j <= n; j++)
117             if (s[j]==s[1]) nn++;
118         fout << nn;
119     }
120     else
121     {
122         nn=n-1-ma;
123         while (ma<n-1)
124         {
125             if(s[g[i].e1]!=s[g[i].e2])
126             {
127                 ma++;
128                 cost+=g[i].c% 625;
129                 obst+=g[i].c/625;
130                 vf=s[g[i].e2];
131                 for (j = 1; j <= n; j++)
132                     if (s[j]==vf)
133                         s[j]=s[g[i].e1];
134             }
135             i++;
136         }
137         if (v==2) fout <<nn << endl << obst;
138         else fout << cost;
139     }
140 }
141 }
142
143 fout.close();
144
145 return 0;
146 }
```

Listing 3.2.12: ninjagoSTL_Codruta.cpp

```

1 #include <iostream>
2 #include <fstream>
3
4 #include<bits/stdc++.h>
5
6 using namespace std;
7
8 short v;
9 int n,m;
10 struct muchie
11 {
12     int e1,e2,c;
13 };
14
15 muchie g[31202];
```

```

16
17 bool ComparMuchii (muchie m1, muchie m2)
18 {
19     return (m1.c<m2.c);
20 }
21
22 void citire()
23 {
24     ifstream fin("ninja.go.in");
25     int i,j,p5;
26     char o;
27     fin >> v;
28     fin >> n >> m;
29
30     for (int i = 1; i <= m; i++)
31     { fin >> g[i].e1 >> g[i].e2;
32         g[i].c=0; p5=1;
33         for (int j = 0; j < 4; j++)
34         {
35             fin >> o;
36             switch (o)
37             {
38                 case 'A': g[i].c+=p5; break;
39                 case 'B': g[i].c+=2*p5; break;
40                 case 'C': g[i].c+=3*p5; break;
41                 case 'D': g[i].c+=4*p5; break;
42                 case 'E': g[i].c+=625;
43             }
44             p5*=5;
45         }
46     }
47     fin.close();
48 }
49
50 int main()
51 {   ofstream fout("ninja.go.out");
52
53     int i,j,nn, ma,obst,vf;
54     long cost;
55     int s[31202];
56
57     citire();
58
59     sort(g+1,g+m+1,ComparMuchii);
60
61     ma=0, cost=0, obst=0;
62     for (i = 1; i <= n; i++) s[i]=i;
63     i=1;
64     while (ma<n-1 && g[i].c<625)
65     {
66         if (s[g[i].e1]!=s[g[i].e2])
67         {
68             ma++;
69             cost+=g[i].c;
70             vf=s[g[i].e2];
71             for (j = 1; j <= n; j++)
72             {
73                 if (s[j]==vf)
74                     s[j]=s[g[i].e1];
75
76             }
77         }
78         i++;
79     }
80
81     if (ma==n-1)
82         switch (v)
83         { case 1:
84             fout<<n<<' \n'; break;
85             case 2:
86                 fout<<"0" << endl <<"0"; break;
87                 case 3: fout << cost;
88             }
89         else
90         {
91             if (v==1)

```

```

92         {
93             nn=0;
94             for (j = 1; j <= n; j++)
95                 if (s[j]==s[1]) nn++;
96                 fout << nn;
97             }
98         else
99         {
100             nn=n-1-ma;
101             while (ma<n-1)
102             {
103                 if(s[g[i].e1]!=s[g[i].e2])
104                 {
105                     ma++;
106                     cost+=g[i].c% 625;
107                     obst+=g[i].c/625;
108                     vf=s[g[i].e2];
109                     for (j = 1; j <= n; j++)
110                         if (s[j]==vf)
111                             s[j]=s[g[i].e1];
112                     }
113                     i++;
114                 }
115             if (v==2) fout <<nn << endl << obst;
116             else fout << cost;
117         }
118     }
119 }
120 fout.close();
121
122 return 0;
123 }
```

3.2.3 *Rezolvare detaliată

3.3 permutare

Problema 3 - permutare

100 de puncte

Definim o *permutare dublă de ordin n* ca fiind un sir format din primele $2n$ numere naturale nenule:

$$(a_1, a_2, \dots, a_n, a_{n+1}, a_{n+2}, \dots, a_{2n}).$$

Această permutare dublă este *de trei ori în creștere*, dacă sunt adevărate următoarele trei proprietăți:

1. secvența formată din primele n elemente este crescătoare: $a_1 < a_2 < \dots < a_n$
2. secvența formată din ultimele n elemente este crescătoare: $a_{n+1} < a_{n+2} < \dots < a_{2n}$
3. perechile ordonate formate din elementele aflate pe poziții identice ale celor două secvențe sunt de asemenea în ordine crescătoare: $a_1 < a_{n+1}, a_2 < a_{n+2}, \dots, a_n < a_{2n}$.

De exemplu permutarea $(1, 3, 4, 2, 5, 6)$ este o permutare dublă de ordin 3, de trei ori în creștere, pentru că secvențele $(1, 3, 4)$ și $(2, 5, 6)$ formează siruri crescătoare, iar toate perechile formate din elementele de pe poziții identice: $(1, 2), (3, 5), (4, 6)$ formează deasemenea siruri crescătoare.

Următoarele permutări duble nu au proprietatea de trei ori în creștere:

$(1, 4, 3, 2, 5, 6)$ - secvența $(1, 4, 3)$ nu este crescătoare,

$(1, 3, 4, 2, 6, 5)$ - secvența $(2, 6, 5)$ nu este crescătoare,

$(1, 4, 5, 2, 3, 6)$ - perechea $(4, 3)$ nu este crescătoare.

Pentru simplificare în continuare *permutarea dublă de trei ori în creștere* se va numi *permutare*.

Vom considera toate permutările de ordin n ordonate lexicografic, numerotate începând cu 1. Tabelul de mai jos conține datele pentru $n = 3$:

poziție	permutare
1	1 2 3 4 5 6
2	1 2 4 3 5 6
3	1 2 5 3 4 6
4	1 3 4 2 5 6
5	1 3 5 2 4 6

Există două tipuri de întrebări:

1. Ce permutare se află pe o poziție dată?
2. Pe ce poziție se află o permutare dată?

Prima întrebare este codificată astfel: $1 \ n \ p$ și se compune din valorile

1 - tipul întrebării,

n - ordinul permutării,

p - poziția permutării cerute.

A doua întrebare este codificată astfel: $2 \ n \ a_1 \ a_2 \dots \ a_{2n}$ și se compune din valorile

2 - tipul întrebării,

n - ordinul permutării,

$a_1 \ a_2 \dots \ a_{2n}$ - elementele permutării.

Exemple

Întrebarea $1 \ 3 \ 2$ înseamnă:

"Ce permutare de ordin 3 se află pe poziția 2 în ordine lexicografică?" și are răspunsul: $1 \ 2 \ 4 \ 3 \ 5 \ 6$.

Întrebarea $2 \ 3 \ 1 \ 3 \ 5 \ 2 \ 4 \ 6$ înseamnă:

"Pe ce poziție se află permutarea de ordin 3: $1 \ 3 \ 5 \ 2 \ 4 \ 6$?" și are răspunsul: 5 .

Cerințe

Să se răspundă corect la un set de întrebări.

Date de intrare

Fișierul **permute.in** conține pe fiecare linie câte o întrebare de orice tip.

Date de ieșire

Fișierul **permute.out** va conține pe câte o linie câte un răspuns la fiecare întrebare din fișierul de intrare, în ordinea întrebărilor.

Restricții și precizări

- $2 < n < 1000$;
- $0 < p \leq 1000000000$ (în cazul întrebărilor de tip 1);
- răspunsul la întrebările de tip 2 este ≤ 1000000000 ;
- fișierele de intrare vor conține cel mult 2000 de întrebări;
- pentru teste în valoare de 20 de puncte numărul de întrebări va fi 1000 iar numerele de ordine ce intervin în calcule vor fi mai mici decât 5000;
- pentru teste în valoare de 30 de puncte întrebările vor fi de tipul 1;
- pentru teste în valoare de 30 de puncte întrebările vor fi de tipul 2;
- pentru teste în valoare de 30 de puncte întrebările vor fi mixte.
- problema va fi evaluată pe teste în valoare de 90 de puncte.
- se vor acorda 10 puncte din oficiu.

Exemple

permute.in	permute.out	Explicații
1 3 2	1 2 4 3 5 6	a doua permutare de ordin 3 (1,2,4,3,5,6)
2 3 1 3 5 2 4 6	5	permutarea (1,3,5,2,4,6) are poziția 5
1 4 1	1 2 3 4 5 6 7 8	prima permutare de ordin 4 (1,2,3,4,5,6,7,8)
2 4 1 2 3 4 5 6 7 8	1	Permutarea (1,2,3,4,5,6,7,8) are poziția 1

Timp maxim de executare/test: **1.5** secunde
Memorie: total **64 MB** din care pentru stivă **32 MB**
Dimensiune maximă a sursei: **25 KB**

3.3.1 Indicații de rezolvare

prof. Szabo Zoltan, Liceul Tehnologic "Petru Maior" Reghin

Pentru simplitate "permutarea dublă de trei ori în creștere" o vom numi "permutare". Prin definiția permutării observăm că pentru fiecare element din a doua secvență, există un element corespunzător din prima secvență cu valoare mai mică.

Astfel, dacă pentru fiecare permutare construim un sir de caractere în care pe pozițiile indicate de elementele primei secvențe punem caracterul '(' iar pentru pozițiile indicate de elementele celei de a doua secvențe punem caracterul ')', obținem o parantezare formată din n perechi de paranteze.

Ordinea lexicografică a tuturor parantezărilor coincide cu cea a permutărilor.

Problema se poate rezolva cu *programare dinamică* bazată pe formula de recurență

$$P(i, j) = P(i - 1, j) + P(i, j - 1)$$

Reconstruirea soluției pornește de la linia n catre linia 1 în matricea triunghiulară P .

Complexitatea unei căutări este $O(n^2)$.

3.3.2 Cod sursă

Listing 3.3.1: back_Eric_.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int MAX_N = 100;
6
7 ofstream out("permutare.out");
8
9 int fst[MAX_N];
10 int snd[MAX_N];
11
12 int type;
13 int n;
14 int at = 0;
15 int p;
16
17 bool check()
18 {
19     at++;
20     if (type == 1 && at == p)
21     {
22         for (int i = 1; i <= n; ++i)
23             out << fst[i] << " ";
24
25         for (int i = 1; i <= n; ++i)
26             out << snd[i] << " ";
27
28         out << "\n";
29         return true;
30     }
31
32     if (type == 2)
33     {
34         for (int i = 1; i <= n; ++i)
35             if (fst[i] != ffst[i] || snd[i] != fsnd[i])
36                 return false;
37     }
38 }
```

```

40         out << at << "\n";
41     return true;
42 }
43
44     return false;
45 }
46
47
48 bool back(int f, int s)
49 {
50     if (f == n + 1 && s == n + 1)
51         return check();
52
53     if (f <= n)
54     {
55         fst[f] = (f + s - 1);
56         if (back(f + 1, s))
57             return true;
58     }
59
60     if (s < f)
61     {
62         snd[s] = (f + s - 1);
63         if (back(f, s + 1))
64             return true;
65     }
66
67     return false;
68 }
69
70 void process(const string &s)
71 {
72     stringstream ss(s);
73     ss >> type;
74     at = 0;
75
76     if (type == 1)
77     {
78         ss >> n >> p;
79         back(1, 1);
80     }
81     else
82     {
83         ss >> n;
84         for(int i = 1; i <= n; ++i)
85             ss >> ffst[i];
86         for(int i = 1; i <= n; ++i)
87             ss >> fsnd[i];
88         back(1, 1);
89     }
90 }
91
92 int main()
93 {
94     ifstream in("permutare.in");
95     string s;
96
97     while (true)
98     {
99         getline(in, s);
100        if (in.eof())
101            break;
102
103        process(s);
104    }
105 }
```

Listing 3.3.2: permutare_Eric.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 ofstream out("permutare.out");
6
```

```

7  const int MAX_N = 1000;
8  const int MAX_V = 2e9 + 10;
9
10 unsigned DP[MAX_N + 2][MAX_N + 2];
11
12 void precalc()
13 {
14     for (int i = 1; i <= MAX_N; ++i)
15     {
16         DP[i][0] = 1;
17         for (int j = 1; j <= i; ++j)
18         {
19             DP[i][j] = DP[i - 1][j] + DP[i][j - 1];
20
21             if (DP[i][j] > MAX_V)
22                 DP[i][j] = MAX_V;
23         }
24     }
25 }
26
27 int fst[MAX_N + 1];
28 int snd[MAX_N + 1];
29
30 void solve1(int n, unsigned p)
31 {
32     int i = 1, j = 1;
33     int u = n, v = n;
34
35     int at = 1;
36
37     while (i <= n || j <= n)
38     {
39         if (p > DP[u][v - 1])
40         {
41             p -= DP[u][v - 1];
42
43             snd[j] = at;
44             j++;
45
46             u--;
47         }
48         else
49         {
50             fst[i] = at;
51             i++;
52
53             v--;
54         }
55
56         at++;
57     }
58 }
59
60     for (int i = 1; i <= n; ++i)
61         out << fst[i] << " ";
62     for (int i = 1; i <= n; ++i)
63         out << snd[i] << " ";
64     out << "\n";
65 }
66
67 void solve2(int n)
68 {
69     int i = 1, j = 1;
70     int at = 1;
71
72     int u = n, v = n;
73
74     int p = 1;
75
76     while (i <= n || j <= n)
77     {
78         if (i <= n && at == fst[i])
79         {
80             i++;
81             v--;
82         }

```

```

83         else
84         {
85             assert(at == snd[j]);
86
87             p += DP[u][v - 1];
88             u--;
89             j++;
90         }
91
92         at++;
93     }
94
95     out << p << "\n";
96 }
97
98 void process(const string &s)
99 {
100     stringstream ss(s);
101
102     int type;
103     ss >> type;
104
105     if (type == 1)
106     {
107         int n, p;
108         ss >> n >> p;
109         solve1(n, p);
110     }
111     else
112     {
113         int n;
114         ss >> n;
115
116         for(int i = 1; i <= n; ++i)
117             ss >> fst[i];
118
119         for(int i = 1; i <= n; ++i)
120             ss >> snd[i];
121         solve2(n);
122     }
123 }
124
125 int main()
126 {
127     precalc();
128
129     ifstream in("permutare.in");
130     string s;
131
132     while (true)
133     {
134         getline(in, s);
135         if (in.eof())
136             break;
137
138         process(s);
139     }
140
141     return 0;
142 }
```

Listing 3.3.3: permutare_VG_.cpp

```

1 #include <cstdio>
2 #include <algorithm>
3
4 using namespace std;
5
6 #define maxn 1010
7 #define inf 2000000000
8
9 int n, m;
10 int d[maxn][maxn];
11 int sol[2*maxn];
12
```

```

13 int main()
14 {
15     freopen("permutare.in", "r", stdin);
16     freopen("permutare.out", "w", stdout);
17
18     d[0][0]=1;
19
20     for(int i=1; i<=1000; ++i)
21     {
22         d[i][0]=1;
23         for(int j=1; j<=i; ++j)
24             d[i][j]=min(1LL*d[i][j-1]+d[i-1][j], 1LL*inf);
25     }
26
27     int tip, poz;
28
29     while(scanf("%d%d", &tip, &n) == 2)
30     {
31         int p1=1, p2=n+1;
32         int inc=n, des=n;
33         if(tip==1)
34         {
35             scanf("%d", &poz);
36             for(int i=1; i<=2*n; ++i)
37             {
38                 if(des==0)
39                 {
40                     sol[p2++]=i;
41                     --inc;
42                 }
43                 else
44                 if(d[inc][des-1]>=poz)
45                 {
46                     sol[p1++]=i;
47                     --des;
48                 }
49                 else
50                 {
51                     sol[p2++]=i;
52                     poz-=d[inc][des-1];
53                     --inc;
54                 }
55             }
56             for(int i=1; i<=2*n; ++i)
57                 printf("%d ", sol[i]);
58             printf("\n");
59         }
60         else
61         {
62             poz=1;
63             for(int i=1; i<=2*n; ++i)
64                 scanf("%d", &sol[i]);
65             for(int i=1; i<=2*n; ++i)
66             {
67                 if(sol[p1]==i)
68                 {
69                     --des;
70                     ++p1;
71                 }
72                 else
73                 {
74                     poz+=d[inc][des-1];
75                     --inc;
76                     ++p2;
77                 }
78             }
79             printf("%d\n", poz);
80         }
81     }
82
83     return 0;
84 }
```

Listing 3.3.4: permutarebrut_Zoli_.cpp

```

1 #include <iostream>
2 #include <fstream>
3
4 using namespace std;
5 long long p[3][1002],a[3][1002],k,t,n,nrsol;
6 bool gasit;
7
8 bool egal()
9 {
10     for (int i=1;i<=2;++i)
11         for (int j=1;j<=n;++j)
12             if (a[i][j]!=p[i][j])
13                 return 0;
14     return 1;
15 }
16
17 void back2(int p1, int p2, int curent)
18 {
19     if (p2>n and !gasit)
20     {
21         nrsol++;
22         if (egal())
23             gasit=true;
24     }
25
26     if (p1<=n and !gasit)
27     {
28         p[1][p1]=curent;
29         back2(p1+1,p2,curent+1);
30     }
31
32     if (p1>p2 and !gasit)
33     {
34         p[2][p2]=curent;
35         back2(p1,p2+1,curent+1);
36     }
37 }
38
39 void back1(int p1, int p2, int curent)
40 {
41     if (p2>n and !gasit)
42     {
43         nrsol++;
44         if (nrsol==k)
45             gasit=true;
46     }
47
48     if (p1<=n and !gasit)
49     {
50         a[1][p1]=curent;
51         back1(p1+1,p2,curent+1);
52     }
53
54     if (p1>p2 and !gasit)
55     {
56         a[2][p2]=curent;
57         back1(p1,p2+1,curent+1);
58     }
59 }
60
61 int main()
62 {
63     ifstream fin ("permutare.in");
64     ofstream fout ("permutare.out");
65
66     while (fin>>t)
67         if (t==2)
68         {
69             fin>>n;
70             for (int i=1;i<=2;++i)
71                 for (int j=1;j<=n;++j)
72                     fin>>a[i][j];
73             gasit=false;
74             nrsol=0;
75             back2(1,1,1);
76             fout<<nrsol<<"\n";

```

```

77         }
78     else
79     {
80         fin>>n>>k;
81         gasit=false;
82         nrsol=0;
83         back1(1,1,1);
84         for (int j=1; j<=n; ++j)
85             fout<<a[1][j]<<" ";
86         for (int j=1; j<=n; ++j)
87             fout<<a[2][j]<<" ";
88         fout<<endl;
89     }
90
91     return 0;
92 }
```

Listing 3.3.5: permutarePA.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 ifstream f("permutare3.in");
6 ofstream g("permutare.out");
7
8 const unsigned oo=2000000001;
9 unsigned i,k,n,m,c,u,v,p,q,x[2010],D[1000][1000];
10
11 int main()
12 {
13     D[0][0]=1;
14     for(i=1; i<1000; i++)
15     {
16         D[i][0]=1;
17         for(k=1; k<=i; k++)
18             D[i][k]=min(D[i-1][k]+D[i][k-1],oo);
19     }
20
21     while(f>>c>>n)
22     {
23         u=v=n, p=1, q=n+1, m=2*n;
24
25         for(i=0; i<=n; i++)
26         {
27             cout<<i<<" : ";
28             for(k=0; k<=i; k++) cout<<D[i][k]<<" ";
29             cout<<"\n";
30         }
31         cout<<"\n";
32
33         if(c==1)
34         {
35             cout<<"i : u v k D[u][v-1]\n"; // u=1 , v=0
36
37             f>>k;
38             for(i=1; i<=m; i++)
39             {
40                 cout<<i<<" : "<<u<<" "<<v<<"\t"<<k<<"\t"<<D[u][v-1]<<"\n";
41                 if(k <= D[u][v-1]) x[p++]=i, v--;
42                 else x[q++]=i, k-=D[u--][v-1];
43             }
44
45             for(i=1; i<=m; i++) g<<x[i]<<' ';
46             g<<"\n";
47
48             cout<<"\n";
49             for(i=1; i<=m; i++) cout<<x[i]<<' ';
50             cout<<"\n";
51
52             continue;
53         }
54
55         for(i=1; i<=m; i++) f>>x[i];
56
57         for(i=1; i<=m; i++) cout<<x[i]<<' ';
58         cout<<"\n\n";
59     }
60 }
```

```

57     cout<<"i : p   x[p]      u   v      k   D[u][v-1]\n"; // u=1 , v=0
58
59     for(i=1,k=1; v>0; i++) // i=1,2,...,n,...,2n-1
60     {
61         cout<<i<<" : "<<p<<"   "<<x[p]<<"\t"<<u<<"   "<<v<<"\t"<<k
62             <<"\t"<<D[u][v-1]<<"\n"; // u=1 , v=0
63         if(x[p]==i) v--, p++;
64         else {k+=D[u][v-1]; q++; u--;}
65     }
66
67     g<<k<<'\n';
68     cout<<"nk = "<<k<<"\n";
69 }
70
71     return 0;
72 }
```

Listing 3.3.6: permutareZoli.cpp

```

1 #include <iostream>
2 #include <fstream>
3
4 using namespace std;
5
6 unsigned p[1002][1002],a[3][1002],t,n,i,j,nr,m,d,
7     poz,linie,k,j1,j2,elementcurrent;
8
9 int main()
10 {
11     ifstream fin ("permutare.in");
12     ofstream fout ("permutare.out");
13
14     p[1][1]=1;
15     linie=1;
16     while (fin>>t)
17     {
18         if (t==2)
19         {
20             fin>>n;
21             for (i=1;i<=2;++i)
22                 for (j=1;j<=n;++)
23                     fin>>a[i][j];
24
25             for (i=linie+1;i<=n+1;++)
26             {
27                 p[i][1]=1;
28                 for (j=2;j<=i;++)
29                 {
30                     if (p[i-1][j]+p[i][j-1]>2000000005)
31                         p[i][j]=2000000005;
32                     else
33                         p[i][j]=p[i-1][j]+p[i][j-1];
34                 }
35             }
36
37             linie=n+1;
38             nr=0;
39             m=n+1; // cel mai mare prim-element posibil de pe linia a 2-a
40             d=0; // diferenta pentru prelucrare recursiva,
41                 // dupa fiecare pozitie scadem 2
42             poz=1; // pozitia elementului curent de pe linia a doua a matricei
43             while (1)
44             {
45                 if (a[2][poz]-d==m)
46                 {
47                     nr++;
48                     break;
49                 }
50                 nr+=p[n+1][m-a[2][poz]+d];
51                 ++poz;
52                 d+=2;
53                 m--;
54             }
55
56             fout<<nr<<"\n";

```

```

57         }
58     else
59     {
60         fin>>n>>k;
61         p[1][1]=1;
62         for (i=linie+1;i<=n;++i)
63         {
64             p[i][1]=1;
65             for (j=2;j<=i;++j)
66                 if (p[i-1][j]>2000000005-p[i][j-1])
67                     p[i][j]=2000000005;
68                 else
69                     p[i][j]=p[i-1][j]+p[i][j-1];
70         }
71
72         linie=n;
73         d=0;      // diferența pentru prelucrare recursiva,
74                  // se mărește din 2 în 2
75         j1=1;
76         j2=1;
77         elementcurrent=1;
78         for (i=n;i>0 and k;i--)
79         {
80             m=i+2;
81             nr=0;j=1;
82             while(nr+p[i][j]<k)
83             {
84                 nr+=p[i][j];
85                 //fout<<nr<<endl;
86                 m--;
87                 j++;
88             }
89             m--;
90             j++;
91             for(j=elementcurrent;j<m+d;++j)
92                 a[1][j1++]=elementcurrent++;
93                 a[2][j2++]=elementcurrent++;
94                 k=k-nr;
95                 d=d+2;
96         }
97
98         while (j2<=n)
99         {
100             a[2][j2]=a[2][j2-1]+1;
101             j2++;
102         }
103
104 //        fout<<2<<" "<<n<<" ";
105         for (j=1;j<=n;++j)
106             fout<<a[1][j]<<" ";
107             for (j=1;j<=n;++j)
108                 fout<<a[2][j]<<" ";
109                 fout<<endl;
110         }
111
112     return 0;
113 }
```

3.3.3 *Rezolvare detaliată

Capitolul 4

OJI 2016

4.1 elicoptere

Problema 1 - elicoptere

100 de puncte

Arhipelagul Zopopan este format din n insule de formă triunghiulară numerotate de la 1 la n . Fiecare insulă este localizată prin coordonatele carteziene ale vârfurilor.

Administrația dorește să cumpere elicoptere pentru a realiza transportul între insule. Un elicopter va putea să asigure o rută între două insule pe distanță minimă obținută pe orizontală sau verticală (paralel cu axele de coordonate). În plus, datorită capacitatei rezervorului o astfel de rută nu poate să depășească o valoare k - număr natural. Elicopterele parcurg rutele în ambele sensuri.

Investiția trebuie să îndeplinească următoarele condiții:

1. Numărul de elicoptere cumpărate să fie minim.
2. Numărul de perechi de insule între care se poate realiza transportul, folosind unul sau mai multe elicoptere să fie maxim.
3. Suma lungimii tuturor rutelor să fie minimă.

Cerințe

Să se scrie un program care pentru n , k și coordonatele vârfurilor insulelor cunoscute, determină:

1. numărul minim de elicoptere ce vor fi cumpărate de administrație;
2. numărul perechilor neordonate de insule între care se poate realiza transportul prin elicopter direct sau indirect;
3. suma distanțelor parcuse de toate elicopterele cumpărate (distanța parcursă de un elicopter se consideră distanța dintre insulele între care acesta asigură transportul).

Date de intrare

Fișierul de intrare **elicoptere.in** conține pe prima linie o valoare v ce poate fi 1, 2, sau 3, în funcție de cerință ce va fi rezolvată, pe linia a doua numerele naturale n și k separate printr-un spațiu, cu semnificația de mai sus, iar pe următoarele n linii se află câte șase numere naturale x_1 , y_1 , x_2 , y_2 , x_3 și y_3 separate prin spațiu reprezentând coordonatele celor trei vârfuri ale insulelor în formatul (abscisă, ordonată).

Date de ieșire

Dacă valoarea lui v este 1 atunci fișierul de ieșire **elicoptere.out** va conține pe prima linie numai numărul minim de elicoptere, ce vor fi cumpărate de administrație.

Dacă valoarea lui v este 2 atunci fișierul de ieșire **elicoptere.out** va conține pe prima linie numai numărul maxim de perechi de insule între care se poate realiza transportul prin elicopter.

Dacă valoarea lui v este 3 atunci fișierul de ieșire **elicoptere.out** va conține pe prima linie suma minimă a lungimii rutelor parcuse de elicoptere.

Restricții și precizări

- $1 \leq n \leq 100$;
- $1 \leq k \leq 1000$;
- coordonatele vârfurilor insulelor sunt numere naturale $0 \leq x_i, y_i \leq 10^6$;
- orice două insule nu au puncte comune;
- la cerința 2, dacă se poate ajunge din insula A în insula B atunci evident că se poate ajunge și din B în A , deci perechea formată din A și B se numără o singură dată;
- Distanța dintre două insule poate fi și număr real. La cerința 3 rezultatul se cere cu o aproximare de 0.001, adică rezultatul notat cu R se consideră corect, dacă față de rezultatul comisiei C îndeplinește condiția $|R - C| < 0.001$.
- Pentru a calcula și afișa un număr real x cu o precizie cât mai mare vă recomandăm folosirea tipului *double*
 - Pentru rezolvarea corectă a cerinței 1 se acordă 20% din punctaj;
 - Pentru rezolvarea corectă a cerinței 2 se acordă 40% din punctaj;
 - Pentru rezolvarea corectă a cerinței 3 se acordă 40% din punctaj.

Exemple

elicoptere.in	elicoptere.out	Explicații
1 6 11 100 20 100 30 105 30 20 20 30 30 20 30 200 20 200 30 205 30 100 40 100 50 105 40 10 40 5 40 10 50 10 20 5 30 10 30	3	Datele corespund figurilor anterioare: $v = 1$, deci se rezolvă NUMAI prima cerință. Perechile de insule cu transport direct cu elicoptere: (1,4) (2,6), (6,5) și obținem astfel 3 elicoptere.
2 6 11 100 20 100 30 105 30 20 20 30 30 20 30 200 20 200 30 205 30 100 40 100 50 105 40 10 40 5 40 10 50 10 20 5 30 10 30	4	Datele corespund figurilor anterioare: $v = 2$, deci se rezolvă NUMAI a doua cerință. Perechile de insule cu transport direct cu elicoptere: (1,4) (2,6), (6,5) și obținem astfel 3 elicoptere. Insula 3 rămâne izolată, astfel avem două grupuri de insule. Primul grup conține insulele 1 și 4, iar al doilea insulele 2, 5, 6. Din primul grup se numara perechea (1,4), iar din al doilea grup se numara perechile de insule (2,5), (2,6) și (5,6). În total 4 perechi de insule între care se poate deplasa cu elicopterul direct sau cu escala (indirect).
3 6 11 100 20 100 30 105 30 20 20 30 30 20 30 200 20 200 30 205 30 100 40 100 50 105 40 10 40 5 40 10 50 10 20 5 30 10 30	30	Datele corespund figurilor anterioare: $v = 3$, deci se rezolvă NUMAI a treia cerință. Perechile de insule cu transport direct cu elicoptere: (1,4) (2,6), (6,5) și obținem astfel 3 elicoptere. Insula 3 rămâne izolată, astfel avem două grupuri de insule. Primul grup conține insulele 1 și 4, iar al doilea insulele 2, 5, 6. Elicopetele asigură transportul direct între insule: 1 și 4 cu distanță verticală egală cu 10; 2 și 6 cu distanță orizontală egală cu 10; 5 și 6 cu distanță verticală egală cu 10; în total liniile de transport au distanță 30.

Timp maxim de executare/test: 0.2 secunde

Memorie: total 4 MB din care pentru stivă 4 MB

Dimensiune maximă a sursei: 10 KB

4.1.1 Indicații de rezolvare

Doru Anastasiu Popescu, Facultatea de Matematică-Informatică, Universitatea din Pitești

Mai întâi trebuie să determinăm distanța dintre orice două insule, adică distanța dintre două triunghiuri disjuncte folosind segmente orizontale sau verticale. Distanța se va calcula folosind pe rând câte un vîrf al fiecărui triunghi și orizontală prin el, respectiv verticală spre celălalt triunghi.

Folosind distanțele calculate se aleg cele care au valoarea mai mică sau egală cu k și se construiește *matricea costurilor* pentru un graf neorientat care conține în fiecare nod câte un triunghi.

Se determină *componentele conexe* ale grafului construit și *arborele parțial de cost minim* pentru fiecare dintre acestea.

n - numărul _de_ componente _conexe reprezintă valoarea ce va fi afișată pentru cerința 1).

Suma costurilor APM-urilor este numărul de la cerința 3).

Pentru cerința 2) se calculează numărul de perechi de triunghiuri din fiecare componentă conexă, adică $Nr = C_{nr}^2$, unde nr reprezintă numărul de triunghiuri din componentă conexă, suma acestor combinări reprezintă numărul cerut (dacă $nr = 1$, atunci $Nr = 0$).

4.1.2 Cod sursă

Listing 4.1.1: elicoptere100_DAP.cpp

```

1 #include <iostream>
2 #include <math.h>
3 #include <iomanip>
4
5 using namespace std;
6
7 ifstream fin("elicoptere.in");
8 ofstream fout("elicoptere.out");
9
10 struct punct{double x,y;};
11 struct tri{punct A,B,C;};
12
13 tri T[1001];
14 int n,k,c[1001],v[1001],u,nrc,w;
15 double a[105][105];
16 int b[105][105];
17 long long perechi;
18
19 void cit()
20 {
21     int i;
22     fin>>w;
23     fin>>n>>k;
24     for(i=1;i<=n;i++)
25         fin>>T[i].A.x>>T[i].A.y>>T[i].B.x>>T[i].B.y>>T[i].C.x>>T[i].C.y;
26     fin.close();
27 }
28
29 double min(double x, double y)
30 {
31     if(x<y) return x;
32     return y;
33 }
34
35 double distLaturaOrizontala(punct M, punct P, punct Q)
36 {
37     double a,b,c;
38     punct N;
39     a=Q.y-P.y;
40     b=P.x-Q.x;
41     c=P.y+Q.x-P.x*Q.y;
42     if(P.y==M.y && M.y==Q.y)
43     {
44         return min(fabs(P.x-M.x),fabs(Q.x-M.x));
45     }
46     if(P.y==Q.y)
47         return 1000000;
48     N.y=M.y;
49     N.y=M.y;
50 }
```

```

51 N.x=(-b*M.y-c)/a;
52 if((P.y<=M.y && M.y<=Q.y) || (Q.y<=M.y && M.y<=P.y))
53     return sqrt((M.x-N.x)*(M.x-N.x)+(M.y-N.y)*(M.y-N.y));
54 return 1000000;
55 }
56
57 double distLaturaVerticala(punct M, punct P, punct Q)
58 {
59 double a,b,c;
60 punct N;
61 a=Q.y-P.y;
62 b=P.x-Q.x;
63 c=P.y*Q.x-P.x*Q.y;
64 if(P.x==M.x && M.x==Q.x)
65     return min(fabs(P.y-M.y),fabs(Q.y-M.y));
66
67 if(P.x==Q.x)
68     return 1000000;
69
70 N.x=M.x;
71 N.y=(-a*M.x-c)/b;
72 if((P.x<=M.x && M.x<=Q.x) || (Q.x<=M.x && M.x<=P.x))
73     return sqrt((M.x-N.x)*(M.x-N.x)+(M.y-N.y)*(M.y-N.y));
74 return 1000000;
75 }
76
77 double distTriOrizontala(tri W, tri V)
78 {
79 double Min,d;
80 Min=1000000;
81 d=distLaturaOrizontala(W.A,V.A,V.B);
82 if(d<Min)
83     Min=d;
84 d=distLaturaOrizontala(W.A,V.A,V.C);
85 if(d<Min)
86     Min=d;
87 d=distLaturaOrizontala(W.A,V.B,V.C);
88 if(d<Min)
89     Min=d;
90 d=distLaturaOrizontala(W.B,V.A,V.B);
91 if(d<Min)
92     Min=d;
93 d=distLaturaOrizontala(W.B,V.A,V.C);
94 if(d<Min)
95     Min=d;
96 d=distLaturaOrizontala(W.B,V.B,V.C);
97 if(d<Min)
98     Min=d;
99 d=distLaturaOrizontala(W.C,V.A,V.B);
100 if(d<Min)
101     Min=d;
102 d=distLaturaOrizontala(W.C,V.A,V.C);
103 if(d<Min)
104     Min=d;
105 d=distLaturaOrizontala(W.C,V.B,V.C);
106 if(d<Min)
107     Min=d;
108 return Min;
109 }
110
111 double distTriVerticala(tri W, tri V){
112 double Min,d;
113 Min=1000000;
114 d=distLaturaVerticala(W.A,V.A,V.B);
115 if(d<Min)
116     Min=d;
117 d=distLaturaVerticala(W.A,V.A,V.C);
118 if(d<Min)
119     Min=d;
120 d=distLaturaVerticala(W.A,V.B,V.C);
121 if(d<Min)
122     Min=d;
123 d=distLaturaVerticala(W.B,V.A,V.B);
124 if(d<Min)
125     Min=d;
126 d=distLaturaVerticala(W.B,V.A,V.C);

```

```

127  if (d<Min)
128      Min=d;
129  d=distLaturaVerticala(W.B,V.B,V.C);
130  if (d<Min)
131      Min=d;
132  d=distLaturaVerticala(W.C,V.A,V.B);
133  if (d<Min)
134      Min=d;
135  d=distLaturaVerticala(W.C,V.A,V.C);
136  if (d<Min)
137      Min=d;
138  d=distLaturaVerticala(W.C,V.B,V.C);
139  if (d<Min)
140      Min=d;
141  return Min;
142 }
143
144 void matriceCost () {
145     int i,j;
146     double x1,x2,x;
147     for (i=1;i<=n;i++)
148         for (j=i+1;j<=n;j++)
149         {
150             x1=min(distTriOrizontala(T[i],T[j]),distTriVerticala(T[i],T[j]));
151             x2=min(distTriOrizontala(T[j],T[i]),distTriVerticala(T[j],T[i]));
152             x=min(x1,x2);
153             if (x<=k)
154                 a[j][i]=a[i][j]=x;
155             else
156                 a[j][i]=a[i][j]=1000000;
157         }
158     }
159
160 void ad(int k)
161 {
162     int i;
163     u++;c[u]=k;v[k]=1;
164     for (i=1;i<=n;i++)
165         if (v[i]==0 && a[k][i]<1000000 && a[k][i]!=0)
166             ad(i);
167 }
168
169 long long comb2(int n)
170 {
171     long long c=n;
172     return c*(c-1)/2;
173 }
174
175 void componente()
176 {
177     int i,j;
178     for (i=1;i<=n;i++)
179         if (v[i]==0)
180         {
181             u=0;nrc++;
182             ad(i);
183             b[nrc][0]=0;
184             for (j=1;j<=u;j++)
185             {
186                 b[nrc][0]++;
187                 b[nrc][b[nrc][0]]=c[j];
188             }
189             perechi+=comb2(b[nrc][0]);
190         }
191     }
192
193 double apm(int q)
194 {
195     int s[105],i,j,k;
196     double ct=0,d[105],Min;
197     s[b[q][1]]=1;
198     for (i=2;i<=b[q][0];i++)
199     {
200         s[b[q][i]]=0;
201         d[b[q][i]]=a[b[q][i]][b[q][1]];
202     }

```

```

203 for(k=1;k<b[q][0];k++)
204 {
205     Min=1000000;
206     for(i=1;i<=b[q][0];i++)
207         if(s[b[q][i]]==0 && Min>d[b[q][i]])
208             {
209                 Min=d[b[q][i]];
210                 j=i;
211             }
212     s[b[q][j]]=1;
213     ct+=Min;
214     for(i=1;i<=b[q][0];i++)
215         if(s[b[q][i]]==0 && d[b[q][i]]>a[b[q][i]][b[q][j]])
216             d[b[q][i]]=a[b[q][i]][b[q][j]];
217 }
218 return ct;
219 }
220
221 int main()
222 {
223     double s=0,t;
224     cit();
225     matriceCost();
226     componente();
227
228     int i,j;
229 /*
230     for(i=1;i<=nrc;i++) {
231         for(j=1;j<=b[i][0];j++)
232             fout<<b[i][j]<<" ";
233             fout<<'\'n';
234     }
235 */
236     for(i=1;i<=nrc;i++) {
237         t=apm(i);
238         s+=t;
239     }
240     if(w==1)
241         fout<<n-nrc<<'\'n';
242     if(w==2)
243         fout<<perechi<<'\'n';
244     if(w==3)
245         fout<<fixed<<setprecision(10)<<s<<'\'n';
246     fout.close();
247     return 0;
248 }
```

Listing 4.1.2: elicoptere100_PA.cpp

```

1 #include <iostream>
2 #include <vector>
3 #include <cmath>
4 #include <iomanip>
5 #include <algorithm>
6 #define re double
7 #define upd(x,y,a,b) L=min(L, \
8                         dst(x[a][u],y[a][u],x[b][v],y[b][v],x[b][w],y[b][w]))
9 #define fi first
10 #define se second
11 #define pb push_back
12 #define mp make_pair
13
14 using namespace std;
15
16 ifstream f("elicoptere.in");
17 ofstream g("elicoptere.out");
18
19 pair<re,pair<int,int>> M[5000];
20 vector<int> V[101];
21
22 re K,L,dst(int,int,int,int,int,int);
23 int p,n,k,i,j,m,x[101][3],y[101][3],v[101],st[201],e,c,s;
24
25 void conect(int,int), df(int);
26
```

```

27 int main()
28 {
29     f>>p>>n>>k;K=(re)k;
30     for(i=1;i<=n;i++) for(j=0;j<3;j++) f>>x[i][j]>>y[i][j];
31     for(i=1;i<n;i++) for(j=i+1;j<=n;j++) conect(i,j);
32     if(p<3)
33     {
34         for(i=1;i<=n;i++)
35             if(!v[i]) {s=0;df(i);e+=s-1;if(s>1)c+=s*(s-1)/2;}
36             g<<(p==1?e:c);return 0;
37     }
38     sort(M+1,M+m+1);K=0.0;
39     for(i=1;i<=n;i++) v[i]=i;
40     for(auto it:M)
41     {
42         L=it.fi;i=it.se.fi;j=it.se.se;
43         for();(st[+k]==i;if(v[i]==i)break;i=v[i];}
44         for();(st[+k]==j;if(v[j]==j)break;j=v[j];}
45         if(i!=j)K+=L;for(;k;k--) v[st[k]]=i;
46     }
47     g<<fixed<<setprecision(10)<<K;
48     return 0;
49 }
50
51 void conect(int a,int b)
52 {
53     L=K+1.0;
54     for(int u=0;u<3;u++) for(int v=0;v<2;v++) for(int w=v+1;w<3;w++)
55     {upd(x,y,a,b);upd(x,y,b,a);upd(y,x,a,b);upd(y,x,b,a);}
56     if(L>K) return;V[a].pb(b);V[b].pb(a);M[+m]=mp(L,mp(i,j));
57 }
58
59 re dst(int x0,int y0,int x1,int y1,int x2,int y2)
60 {
61     re X0=(re)x0,Y0=(re)y0,X1=(re)x1,Y1=(re)y1, X2=(re)x2,Y2=(re)y2;
62     if(x1>x2) {swap(x1,x2);swap(y1,y2);}
63     if(x0<x1||x0>x2) return K+1.0;
64     if(x1==x2) return (re)(min(abs(y0-y1),abs(y0-y2)));
65     Y0=fabs(Y0-(Y1*(X2-X0)+Y2*(X0-X1))/(X2-X1));
66     if(Y0>K) return K+1.0;
67     return Y0;
68 }
69
70 void df(int q){v[q]=1;s++;for(auto it:V[q]) if(!v[it]) df(it);}

```

Listing 4.1.3: elicoptere100_PA_naiv.cpp

```

1 #include <fstream>
2 #include <iomanip>
3 #include <cstdlib>
4 #include <cmath>
5
6 #define tip double
7 #define oo 1000000000.0
8
9 using namespace std;
10
11 ifstream f("elicoptere.in");
12 ofstream g("elicoptere.out");
13
14 int n,v,i,j,m,e,viz[102],L,R,q,Q[102],sol1,sol2,a[102][102],A[102],U,W;
15 tip k,d[102][102],x[602],cost[10010],u[10010],w[10010],sol3,
16     edge(int,int),dst1(tip*,tip*,tip*),dst2(tip*,tip*,tip*);
17 void solve12(),solve3();
18
19 int main()
20 {
21     f>>v>>n>>k;
22     for(i=1;i<=n;i++)
23         for(j=1;j<=6;j++)
24             f>>x[m++];
25
26     for(i=0;i<n;i++)
27     {
28         d[i][i]=oo;

```

```

29         for(j=i+1;j<n;j++)
30     {
31         d[i][j]=min(edge(i,j),edge(j,i));
32         if(d[i][j]>k)d[i][j]=oo;
33         d[j][i]=d[i][j];
34         if(d[i][j]!=oo) { e++; cost[e]=d[i][j];u[e]=i;w[e]=j; }
35     }
36 }
37
38 solve12();
39 if(v<3)
40 {
41     v==1?g<<sol1:g<<sol2;
42     return 0;
43 }
44
45 for(i=1;i<e;i++)
46 {
47     q=i;
48     for(j=i+1;j<=e;j++)
49         if(cost[j]<cost[q])
50             q=j;
51     if(q>i) {swap(cost[i],cost[q]);swap(u[i],u[q]);swap(w[i],w[q]);}
52 }
53
54 for(i=0;i<n;i++)
55     a[i][i]=1;
56 i=1;
57 while(sol1)
58 {
59     U=u[i];W=w[i];
60     if(!a[U][W])
61     {
62         sol3+=cost[i];sol1--;
63     }
64     for(j=0;j<n;j++)A[j]=a[U][j]|a[W][j];
65     for(j=0;j<n;j++)
66         if(A[j])
67             for(q=0;q<n;q++)
68                 if(A[q])
69                     a[j][q]=1;
70     i++;
71 }
72
73 g<<fixed<<setprecision(10)<<sol3;
74 return 0;
75 }
76
77 tip edge(int i,int j)
78 {
79     tip ret=oo;
80     int a,b,c;
81     for(a=0;a<3;a++)
82         for(b=0;b<3;b++)
83             for(c=b+1;c<3;c++)
84             {
85                 ret=min(ret,dst1(x+6*i+2*a,x+6*j+2*b,x+6*j+2*c));
86                 ret=min(ret,dst2(x+6*i+2*a,x+6*j+2*b,x+6*j+2*c));
87             }
88     return ret;
89 }
90
91 tip dst1(tip *A,tip *B,tip *C)
92 {
93     tip x0=A[0],y0=A[1],x1=B[0],y1=B[1],x2=C[0],y2=C[1];
94     if(x1>x2){swap(x1,x2);swap(y1,y2);}
95     if(x0<x1||x0>x2) return oo;
96     if(x1==x2) return min(fabs(y1-y0),fabs(y2-y0));
97     return abs((x0*y1+x1*y2+x2*y0-x0*y2-x2*y1-x1*y0)/(x2-x1));
98 }
99
100 tip dst2(tip *A,tip *B,tip *C)
101 {
102     tip x0=A[0],y0=A[1],x1=B[0],y1=B[1],x2=C[0],y2=C[1];
103     if(y1>y2){swap(x1,x2);swap(y1,y2);}
104     if(y0<y1||y0>y2) return oo;

```

```

105     if(y1==y2) return min(fabs(x1-x0), fabs(x2-x0));
106     return abs((x0*y1+x1*y2+x2*y0-x0*y2-x2*y1-x1*y0)/(y2-y1));
107 }
108
109 void solve12()
110 {
111     for(i=0;i<n;i++)
112     if(!viz[i])
113     {
114         L=R=1;Q[L]=i;viz[i]=1;
115         while(L<=R)
116         {
117             j=Q[L++];
118             for(q=0;q<n;q++)
119                 if(d[j][q]<oo&&!viz[q])
120                 {
121                     Q[++R]=q;viz[q]=1;
122                 }
123         }
124         sol1+=R-1;
125         sol2+=R*(R-1)/2;
126     }
127 }
```

4.1.3 *Rezolvare detaliată

4.2 summax

Problema 2 - summax

Avem o matrice triunghiulară cu n linii, cu elemente numere întregi. În această matrice putem construi un traseu după următoarea regulă:

- primul element al traseului este elementul $a_{1,1}$
- dacă elementul $a_{i,j}$ aparține traseului, atunci următorul element al traseului poate fi doar $a_{i+1,j}$ sau $a_{i+1,j+1}$, pentru orice $1 \leq i < n$.

Traseul se va codifica cu numerele de ordine ale coloanelor, parcugând liniile de la 1 la n . Valoarea traseului este egală cu suma elementelor ce îl formează.

Traseul evidențiat în exemplul din dreapta are valoarea $5 + 4 + 6 + 5 + 4 = 24$, și se codifică cu 1, 2, 3, 3, 4.

Fie mulțimea tuturor traseelor de valoare maximă generate în ordine lexicografică și numerotate. Pentru exemplul alăturat avem șase trasee de lungime maximă:

- traseul 1. 1 1 1 2 ($5+2+7+6+4=24$)
- traseul 2. 1 1 1 2 2 ($5+2+7+6+4=24$)
- traseul 3. 1 2 2 2 2 ($5+4+5+6+4=24$)
- traseul 4. 1 2 3 3 4 ($5+4+6+5+4=24$)
- traseul 5. 1 2 3 4 4 ($5+4+6+5+4=24$)
- traseul 6. 1 2 3 4 5 ($5+4+6+5+4=24$)

a ₁₁				
a ₂₁	a ₂₂			
a ₃₁	a ₃₂	a ₃₃		
a ₄₁	a ₄₂	a ₄₃	a ₄₄	
a ₅₁	a ₅₂	a ₅₃	a ₅₄	a ₅₅

Figura 4.1: summax

5				
2	4			
7	5	6		
6	6	5	5	
3	4	3	4	4

Figura 4.2: summax

Cerințe

Cunoscând dimensiunea și elementele unei matrice triunghiulare, respectiv două numere naturale st și dr ($st \leq dr$), se cere să se determine:

1. Numărul total al traseelor de valoare maximă. În cazul în care această valoare depășește 2000000000, se va tipări valoarea 2000000001;
2. Traseele cu numerele de ordine $st, st+1, \dots, dr$.

Date de intrare

Fișierul **summax.in** conține pe prima linie un număr natural v . Pentru toate testele de intrare, numărul v poate avea doar valoarea 1 sau 2.

A doua linie conține trei numere naturale n , st și dr , separate prin spațiu. Următoarele n linii conțin câte o linie a matricei triunghiulare astfel: linia i conține i elemente, și anume valorile $a_{i,1}$, $a_{i,2} \dots a_{i,i}$ pentru orice $1 \leq i \leq n$.

Date de ieșire

Dacă valoarea lui v este 1, se va rezolva numai punctul 1 din cerință.

În acest caz, în fișierul de ieșire **summax.out** se va scrie un singur număr natural ce reprezintă numărul traseelor de lungime maximă.

Dacă valoarea lui v este 2, se va rezolva numai punctul 2 din cerință.

În acest caz, în fișierul de ieșire **summax.out** se vor tipări pe câte o linie n numere naturale separate prin spațiu, reprezentând codificările traseelor de valoare maximă cu numerele de ordine st , $st + 1, \dots, dr$.

Restricții și precizări

- $1 \leq n \leq 2000$
- $1 \leq st \leq dr \leq 2000000000$
- $1 \leq dr - st \leq 1000$
- elementele matricei triunghiulare sunt numere naturale strict pozitive
- valoarea maximă a traseului nu depășește 1000000000

Exemple

summax.in	summax.out	Explicații
1 5 2 4 5 2 4 7 5 6 6 6 5 5 3 4 3 4 4	6	v=1 Numărul traseelor de valoare maximă este 6. (vezi exemplul de mai sus).
2 5 2 4 5 2 4 7 5 6 6 6 5 5 3 4 3 4 4	1 1 1 2 2 1 2 2 2 2 1 2 3 3 4	v=2 st=2 dr=4 S-au tipărit traseele cu numerele de ordine 2, 3 și 4. (vezi exemplul de mai sus).

Alte informații utile

- o variabilă de tip *int* (*C++*) respectiv *integer* (*Pascal*) ocupă 4 octeți (32 de biți)
- o matrice cu 1000 de linii și 1000 de coloane cu elemente întregi ocupă $(1000 * 1000 * 4) / (1024 * 1024) = 3,81$ MB
- Testele de intrare au următoarea configurație:

fișierul de intrare	v	n	st	dr
0	1	20	4	10
1	1	900	50	100
2	1	1300	2000	3000
3	1	1700	20000	21000
4	2	20	6	9
5	2	30	20 000 000	20 001 000
6	2	60	40 030 000	40 031 000
7	2	100	139 876 543	139 876 999
8	2	500	137 987 000	137 988 000
9	2	700	123 456 789	123 457 777
10	72	900	100 000 000	100 001 000

11	2	1000	1 999 999 999	2 000 000 000
12	2	1010	1 000 000	1 000 001
13	2	1020	10 123	11 111
14	2	1100	1 999 999 999	2 000 000 000
15	2	1200	1 000 000	1 000 001
16	2	1300	10 123	11 111
17	2	1500	1 999 999 999	2 000 000 000
18	2	1600	1 000 000	1 000 999
19	2	1700	10 123	11 123

Timp maxim de executare/test: **2.0** secunde

Memorie: total **16 MB** din care pentru stivă **16 MB**

Dimensiune maximă a sursei: **10 KB**

4.2.1 Indicații de rezolvare

Szabo Zoltan - Liceul Tehnologic "Petru Maior" Reghin

1. Soluția brută - 10 puncte

complexitate $O(2^n)$

Cu un algoritm *backtracking* generăm toate soluțiile și calculăm valoarea maximă a unui drum. Apoi lansăm un nou *backtracking* pentru a parurge drumurile în *ordine lexicografică*, și tipărim în fișier soluțiile cu numerele de ordine cerute.

2. Soluție backtrack îmbunătățită - 30-45 de puncte

complexitate $O(2^n)$

Valoarea maximă a drumului se calculează cu *programare dinamică* cu ajutorul unei matrice.

$b[n][j] = a[n][j]$ pentru orice $1 \leq j \leq n$

$b[i][j] = a[i][j] + \max(b[i+1][j], b[i+1][j+1]),$ pentru orice $1 \leq j \leq i < n$

Valoarea drumului maxim este $b[1][1]$, vom genera cu *backtracking* numărul drumurilor, respectiv drumurile cu numerele de ordine cerute.

3. Programare dinamică - 65 de puncte

complexitate $O(n^2),$

memorie pentru trei matrice

Numărul drumurilor de valoare maximă se calculează într-un tabel, cunoscând matricile a și $b.$

$c[n][j] = 1$ pentru orice $1 \leq j \leq n$

$c[i][j] = c[i+1][j],$ dacă $b[i+1][j] > b[i+1][j+1],$ pentru orice $1 \leq j \leq i < n$

$c[i][j] = c[i+1][j+1],$ dacă $b[i+1][j] < b[i+1][j+1],$ pentru orice $1 \leq j \leq i < n$

$c[i][j] = c[i+1][j] + c[i+1][j+1],$ dacă $b[i+1][j] = b[i+1][j+1],$ pentru orice $1 \leq j \leq i < n$

Dacă valoarea depășește 2000000000, atunci valoarea va fi 2000000001.

$c[1][1]$ va conține numărul drumurilor de valoare maximă

Cu ajutorul acestei matrice putem recalculta drumul cu un număr de ordine dat, știind către drumuri de valoare maximă se continuă din $a[i][j]$ către $a[i+1][j]$ și către $a[i+1][j+1].$

Memoria disponibilă de $16Mo$ permite valoarea maximă a lui $n = 1000 - 1100$

4. Programare dinamică - 80 de puncte

complexitate $O(n^2),$

memorie pentru două matrice

Observăm că putem economisi o matrice, pentru că valorile lui a se pot deduce din valorile lui $b.$

Memoria disponibilă de $16Mo$ permite valoarea maximă a lui $n = 1400 - 1400$

5. Programare dinamică - 100 de puncte

complexitate $O(n^2),$

memorie pentru o matrice

Atunci când declarăm două matrice pentru două triunghiuri, practic o jumătate de matrice este nefolositoare. Pentru a folosi memoria disponibilă în mod eficient, putem folosi două trucuri:

a. *linearizarea matricei* - memorarea elementelor într-un sir

b. cele două jumătăți de matrice se vor combina într-o singură matrice de dimensiuni $n * (n+1)$

Memoria disponibilă de $16Mo$ permite valoarea maximă a lui $n = 2000$

4.2.2 Cod sursă

Listing 4.2.1: summax_80.cpp

```

1 #include <iostream>
2
3 using namespace std;
4 unsigned int a[2001][2002], i, j, k, n, st, dr, dif, col, p;
5
6 int main()
7 {
8     ifstream fin("summax.in");
9     ofstream fout("summax.out");
10
11    fin>>p>>n>>st>>dr;
12    for (i=1; i<=n; ++i)
13        for(j=1; j<=i; ++j)
14            fin>>a[i][j];
15
16    for (j=1; j<=n; ++j)
17        a[j-1][n]=1;
18
19    for (i=n-1; i>0; --i)
20        for (j=1; j<=i; ++j)
21            if (a[i+1][j]>a[i+1][j+1])
22            {
23                a[i][j]=a[i][j]+a[i+1][j];
24                a[j-1][i]=a[j-1][i+1];
25            }
26            else
27            {
28                a[i][j]=a[i][j]+a[i+1][j+1];
29                a[j-1][i]=a[j][i+1];
30                if (a[i+1][j]==a[i+1][j+1])
31                    if (a[j-1][i]+a[j-1][i+1]>2000000000)
32                        a[j-1][i]=2000000001;
33                    else
34                        a[j-1][i]+=a[j-1][i+1];
35            }
36        if (p==1)
37            fout<<a[0][1]<<"\n";
38        else
39        {
40            for (k=st; k<=dr; ++k)
41            {
42                fout<<1;
43                dif=k;
44                col=1;
45                for (i=2; i<=n; ++i)
46                    if (a[col-1][i]>=dif and a[i][col]>=a[i][col+1])
47                    {
48                        fout<<" "<<col;
49                    }
50                    else
51                    {
52                        if (a[i][col]==a[i][col+1])
53                            dif=dif-a[col-1][i];
54                        col++;
55                        fout<<" "<<col;
56                    }
57                fout<<"\n";
58            }
59        }
60        return 0;
61 }
```

Listing 4.2.2: summax_100_PA.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 ifstream f("summax.in");
6 ofstream g("summax.out");
7
8 unsigned p,n,x,y,i,j,*a[2001],*b[2001],oo=2000000001,D,s[2001];
9
10 void afiseaza(unsigned,unsigned);
11
12 int main()
13 {
14     f>>p>>n>>x>>y;
15     for(i=0;i<=n;i++)
16     {
17         a[i]=new unsigned[i+3];
18         b[i]=new unsigned[i+3];
19         for(j=1;j<=i;j++)
20             f>>a[i][j];
21     }
22
23     for(j=1;j<=n;j++)
24         b[n][j]=a[n][j];
25
26     for(i=n-1;i>=1;i--)
27         for(j=1;j<=i;j++)
28             b[i][j]=a[i][j]+max(b[i+1][j],b[i+1][j+1]);
29
30     for(j=1;j<=n;j++)
31         a[n][j]=1;
32
33     for(i=n-1;i>=1;i--)
34         for(j=1;j<=i;j++)
35         {
36             if(b[i+1][j]>b[i+1][j+1])
37                 a[i][j]=a[i+1][j];
38             else
39                 if(b[i+1][j]<b[i+1][j+1])
40                     a[i][j]=a[i+1][j+1];
41             else
42                 a[i][j]=min(a[i+1][j]+a[i+1][j+1],oo);
43         }
44
45     if(p==1)
46     {
47         g<<a[1][1];
48         return 0;
49     }
50
51     afiseaza(1,1);
52     return 0;
53 }
54
55 void afiseaza(unsigned I,unsigned J)
56 {
57     if(D+a[I][J]<x)
58     {
59         D+=a[I][J];
60         return;
61     }
62
63     s[I]=J;
64     if(I==n)
65     {
66         D++;
67         for(unsigned k=1;k<=n;k++)
68             g<<s[k]<<' ';
69         g<<'\'n';
70         return;
71     }
72
73     if(b[I+1][J]>=b[I+1][J+1])
74     {
75         afiseaza(I+1,J);
76         if(D==y) return;

```

```
77      }
78      if(b[I+1][J+1]>=b[I+1][J])
79          afiseaza(I+1,J+1);
80
81 }
```

4.2.3 *Rezolvare detaliată

Capitolul 5

OJI 2015

5.1 2sah

Problema 1 - 2sah

100 de puncte

Se dă o tablă de șah cu $n + 1$ linii (numerotate de sus în jos începând cu 1) și $2n + 1$ coloane (numerotate de la stânga la dreapta începând cu 1). Pe prima linie pătratul din mijloc conține 1 gram de fân, iar celelalte pătrate de pe prima linie nu conțin nimic. Începând cu linia a doua fiecare pătrat conține o cantitate de fân obținută prin adunarea cantităților de fân din cele 3 pătrate ale liniei anterioare cu care se învecinează (pe verticală și diagonală). De exemplu dacă $n = 3$ tabla are 4 linii, 7 coloane și următoarea configurație.

	*		1			
		1	1*	1		
	1	2	3	2	1*	
1	3	6	7	6	3	1

Figura 5.1: 2sah

Un cal pleacă de pe prima linie, de pe o coloană $k \leq n$, să redea din orice poziție (i, j) în poziția $(i + 1, j + 2)$ atât timp cât este posibil și mănâncă tot fânul din pătratele prin care trece. De exemplu, pentru $n = 3$ și $k = 2$, pătratele prin care trece calul sunt marcate cu asterisc (*)

Cerințe

1. Cunoscând n și k , să se calculeze cantitatea de fân de pe linia k a tablei.
2. Cunoscând n și k , să se calculeze câte grame de fân mănâncă un cal care pleacă de pe prima linie, de pe coloana k . Întrucât aceste numere pot fi mari, se cere doar restul *modulo* 100003 ale acestor numere.

Date de intrare

Fișierul de intrare **2sah.in** va conține pe prima linie un număr t cu valoarea 1 sau 2. Pe a doua linie a fișierului de intrare se găsesc două numere naturale n și k separate printr-un spațiu.

Dacă $t = 1$ se va rezolva prima cerință, deci pentru valoarea n citită tabla are $n + 1$ linii și $2n + 1$ coloane, iar k reprezintă numărul liniei de pe care trebuie calculată cantitatea de fân.

Dacă $t = 2$ se va rezolva a doua cerință, deci pentru valoarea n citită tabla are $n + 1$ linii și $2n + 1$ coloane, iar k reprezintă numărul coloanei din prima linie de unde pleacă calul.

Date de ieșire

Dacă t din fișierul de intrare este 1 se va rezolva doar prima cerință.

În acest caz fișierul de ieșire **2sah.out** va conține un singur număr reprezentând cantitatea totală de fân din toate pătratele situate pe tablă pe linia k (trebuie afișat restul *modulo* 100003).

Dacă t din fișierul de intrare este 2 se va rezolva doar a doua cerință.

În acest caz fișierul de ieșire **2sah.out** va conține un singur număr reprezentând cantitatea totală de fân mânăcată de un cal care pleacă de pe linia 1 și coloana k (trebuie afișat restul *modulo* 100003).

Restricții și precizări

- $1 \leq k \leq n \leq 1000000000$ (un miliard)
- La cerința 1 pentru 80% dintre teste $k \leq n \leq 1000000$, iar pentru alte 20% din teste $k \leq n \leq 1000000000$
 - La cerința 2 pentru 30% dintre teste $k \leq n \leq 1000$, pentru alte 30% dintre teste $k \leq n \leq 1000000$, iar pentru restul de 40% dintre teste $k \leq n \leq 1000000000$.
 - Rezolvarea corectă a primei cerințe asigură 30% din punctajul testului respectiv.
 - Rezolvarea corectă a celei de a doua cerințe asigură 70% din punctajul testului respectiv.

Exemple

2sah.in	2sah.out	Explicații
1 3 2	3	t=1, deci se rezolvă prima cerință. Pe linia a doua există 3 pătrate care conțin fiecare câte un gram de fân.(vezi desenul din enunț)
2 3 2	2	t=2, deci se rezolvă doar a doua cerință. Traseul calului este: (1,2) -> (2,4) -> (3,6) adică exact pătrățele marcate cu asterisc în desenul din enunț. Prima poziție nu conține fân, iar celelalte două conțin câte un gram de fân. Deci calul mănâncă 2 grame de fân.

Timp maxim de executare/test: **0.3** secunde

Memorie: total **32 MB** din care pentru stivă **8 MB**

Dimensiune maximă a sursei: **20 KB**

5.1.1 Indicații de rezolvare

Prof. Adrian Panaete - Colegiul Național "A. T. Laurian" Botoșani

Notăm $f_{i,j}$ cantitatea de fân din pătratul de pe linia i și coloana j .

Avem $f_{i,j} = f_{i-1,j-1} + f_{i-1,j} + f_{i-1,j+1}$ pentru orice i, j cu $1 \leq i \leq n+1, 1 \leq j \leq 2n+1$

Pentru prima cerință evident că orice cantitate nenulă de pe o linie va fi adunată la exact 3 cantități de pe linia următoare. Mai precis cantitatea $f_{i,j}$ de pe linia i se adună la cantitățile $f_{i+1,j-1}, f_{i+1,j}$ și $f_{i+1,j+1}$. Astfel deducem că suma pe o linie este triplă față de suma liniei anterioare. Deoarece sumele formează o progresie geometrică de rație 3 cu primul termen 1 (adică suma pe prima linie este 1) obținem că soluția este 3^{k-1} .

Pentru a obține maximul de punctaj la această cerință trebuie calculată această valoare folosind *ridicare la putere în timp logaritmic*.

Pentru a doua cerință observăm că traseul calului conține pozițiile $(1, k), (2, k+2), (3, k+4), \dots, (i, k+2i-2) \dots$ unde $k+2i-2 \leq 2n+1$ adică $i \leq (2n+3-k)/2 (\leq n+1)$.

Exprimând cantitățile din fiecare pozitie cu formula de recurență se observă că aceasta cantitate se descompune exact în cantitățile pe care le-ar consuma caii care pleacă de pe prima linie de pe coloanele $k+1, k+2$ și $k+3$; mai precis notând F_k soluția căutată vom avea formula de recurență:

$$F_k = F_{k+1} + F_{k+2} + F_{k+3}$$

Pentru un calcul mai simplu vom considera $j = n+1-k$ obținând sirul $T_j = S_k$ sir care respectă relația de recurență

$$T_j = T_{j-1} + T_{j-2} + T_{j-3}$$

cu termenii inițiali

$$T_0 = T_1 = 1, T_2 = 2$$

Acest sir mai este cunoscut și sub denumirea de *sir Tribonacci* iar pentru calculul termenilor săi se pot aplica tehnici similare calculului pentru termenii sirului Fibonacci (sau mai general pentru oricare siruri definite prin formule de *recurență liniară*).

Pentru punctaj parțial se poate folosi generarea termenilor folosind liniar formula de recurență.

Pentru punctaj maxim trebuie aplicată o metodă cu timp de execuție logaritmic. Observăm că din formula de recurență se poate deduce că (în termeni de operații cu matrice) avem:

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} T_j \\ T_{j+1} \\ T_{j+2} \end{pmatrix} = \begin{pmatrix} T_{j+1} \\ T_{j+2} \\ T_{j+3} \end{pmatrix}$$

Deci folosind matricea

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Avem

$$\begin{pmatrix} T_j \\ T_{j+1} \\ T_{j+2} \end{pmatrix} = A^j \cdot \begin{pmatrix} T_0 \\ T_1 \\ T_2 \end{pmatrix}$$

Iar matricea A^j se va calcula folosind *ridicarea la putere în timp logaritmic*.

5.1.2 Cod sursă

Listing 5.1.1: 2sah_brut.cpp

```

1 #include <iostream>
2
3 #define MOD 100003
4
5 using namespace std;
6
7 ifstream f("2sah.in");
8 ofstream g("2sah.out");
9
10 int t,n,k,i,j,m,s,a[1005][2010];
11
12 int main()
13 {
14     f>>t>>n>>k;
15     a[1][n+1]=1;
16
17     if(t==1)
18     {
19         for(i=2;i<=k;i++)
20             for(j=1;j<=2*n+1;j++)
21                 a[i][j]=(a[i-1][j-1]+a[i-1][j]+a[i-1][j+1])%MOD;
22         s=0;
23         for(j=1;j<=2*n+1;j++)
24             s=(s+a[k][j])%MOD;
25         g<<s;
26         return 0;
27     }
28
29     for(i=2;i<=n+1;i++)
30         for(j=1;j<=2*n+1;j++)
31             a[i][j]=(a[i-1][j-1]+a[i-1][j]+a[i-1][j+1])%MOD;
32
33     i=1;
34     j=k;
35     s=0;
36     m=2*n+1;
37     n++;
38     while(i<=n&&j<=m)
39     {
40         s=(s+a[i][j])%MOD;
41         i++; j+=2;
42     }
43
44     g<<s;
45     return 0;
46 }
```

Listing 5.1.2: 2sah_clasic.cpp

```

1 #include <fstream>
2
3 #define MOD 100003
4
5 using namespace std;
6
7 ifstream f("2sah.in");
8 ofstream g("2sah.out");
9
10 long long t,n,k,sol,i,j,m,A[3][3],
11     P[3][3]={{0,1,0},{0,0,1},{1,1,1}},
12     S[3][3]={{1,0,0},{0,1,0},{0,0,1}};
13
14 int main()
15 {
16     f>>t>>n>>k;
17     if(t==1)
18     {
19         sol=1;
20         k--;
21         k%=(MOD-1); //utilizand teorema Fermat
22         for(;k;k--)
23         {
24             sol=(3*sol)%MOD;
25         }
26         g<<sol;
27         return 0;
28     }
29
30     k=n+2-k;
31     for(;k;k>>=1)
32     {
33         if(k&1)
34         {
35             for(i=0;i<3;i++)
36                 for(j=0;j<3;j++)
37                     for(m=0;m<3;m++)
38                         A[i][j]+=S[i][m]*P[m][j];
39             for(i=0;i<3;i++)
40                 for(j=0;j<3;j++)
41                 {
42                     S[i][j]=A[i][j]%MOD;
43                     A[i][j]=0;
44                 }
45         }
46
47         for(i=0;i<3;i++)
48             for(j=0;j<3;j++)
49                 for(m=0;m<3;m++)
50                     A[i][j]+=P[i][m]*P[m][j];
51
52         for(i=0;i<3;i++)
53             for(j=0;j<3;j++)
54             {
55                 P[i][j]=A[i][j]%MOD;
56                 A[i][j]=0;
57             }
58
59     }
60
61     g<<S[2][0]<<'\\n';
62     return 0;
63 }
```

Listing 5.1.3: 2sah_liniar.cpp

```

1 #include <fstream>
2
3 using namespace std;
4
5 ifstream f("2sah.in");
6 ofstream g("2sah.out");
7
```

```

8 const int MOD=100003;
9 int n,t,k,i,ans,t0,t1,t2,t3;
10
11 int main()
12 {
13     f>>t>>n>>k;
14     if(t==1)
15     {
16         ans=1;
17         for(k--;k;k--)
18             ans=(ans*3)%MOD;
19         g<<ans;
20         return 0;
21     }
22
23     k=n+2-k;
24     if(k<3) {g<<1;return 0;}
25
26     t1=t2=1;
27     for(i=3;i<=k;i++)
28     {
29         t3=(t0+t1+t2)%MOD;
30         t0=t1;t1=t2;t2=t3;
31     }
32
33     g<<t3;
34     return 0;
35 }
```

Listing 5.1.4: 2sah_oficiala.cpp

```

1 //Problema 2sah - solutia oficiala O(log N)
2 //Sursa: Panaete Adrian
3 #include <fstream>
4
5 #define tip long long
6 #define MOD 100003
7
8 using namespace std;
9
10 ifstream f("2sah.in");
11 ofstream g("2sah.out");
12
13 tip t,n,k,A[3][3],S[3][3],C[3][3];
14
15 void sol1(),sol2();
16
17 int main()
18 {
19     f>>t>>n>>k;
20     if(t==1)sol1();
21     else sol2();
22     return 0;
23 }
24
25 void sol1()
26 {
27     tip ans=1,p=3;
28     for(k--;k;k>=1)
29     {
30         if(k%2) ans=(ans*p)%MOD;
31         p=(p*p)%MOD;
32     }
33
34     g<<ans;
35 }
36
37 void sol2()
38 {
39     int i,j,q;
40     k=n+2-k;
41     for(i=0;i<3;i++)
42     {
43         S[i][i]=1;
44         A[2][i]=1;
```

```

45     }
46
47     A[0][1]=A[1][2]=1;
48     for(;k;k>=1)
49     {
50         if(k%2)
51         {
52             for(i=0;i<3;i++)
53                 for(j=0;j<3;j++)
54                 {
55                     C[i][j]=0;
56                     for(q=0;q<3;q++)
57                         C[i][j]=(C[i][j]+S[i][q]*A[q][j])%MOD;
58                 }
59             for(i=0;i<3;i++)
60                 for(j=0;j<3;j++)
61                     S[i][j]=C[i][j];
62         }
63
64         for(i=0;i<3;i++)
65             for(j=0;j<3;j++)
66             {
67                 C[i][j]=0;
68                 for(q=0;q<3;q++)
69                     C[i][j]=(C[i][j]+A[i][q]*A[q][j])%MOD;
70             }
71
72         for(i=0;i<3;i++)
73             for(j=0;j<3;j++)
74                 A[i][j]=C[i][j];
75     }
76
77     g<<S[1][2];
78 }

```

Listing 5.1.5: 2sah_smart.cpp

```

1 #include <iostream>
2 #include <map>
3 #include<set>
4
5 #define tip long long
6 #define MOD 100003
7
8 using namespace std;
9
10 ifstream f("2sah.in");
11 ofstream g("2sah.out");
12
13 map<int,int> T;
14 set<int> S;
15
16 void solve(int);
17 long long t,n,k,sol,p;
18
19 int main()
20 {
21     f>>t>>n>>k;
22     if(t==1)
23         for(k--,sol=1,p=3;k;k/=2)
24         {
25             if(k&1)
26                 sol=(sol*p)%MOD;
27             p=(p*p)%MOD;
28         }
29     else
30     {
31         T[0]=0;
32         T[1]=1;
33         T[2]=1;
34         T[3]=2;
35         for(int i=0;i<4;i++)
36             S.insert(i);
37         n=n+2-k;

```

```

38         solve(n);
39         sol=T[n];
40     }
41
42     g<<sol<<'\n';
43     return 0;
44 }
45
46 void solve(int M)
47 {
48     if(S.find(M) != S.end()) return;
49
50     int m=M/2;
51
52     solve(m+1);
53     solve(m);
54     solve(m-1),
55     solve(m-2);
56
57     S.insert(M);
58     if(M%2)
59         T[M]=(1LL*T[m]*T[m-1]+1LL*T[m]*T[m-1] +
60                 1LL*T[m]*T[m]+1LL*T[m+1]*T[m+1])%MOD;
61     else
62         T[M]=(1LL*T[m]*T[m+1]+1LL*T[m-1]*T[m] +
63                 1LL*T[m-2]*T[m]+1LL*T[m-1]*T[m-1])%MOD;
64 }
```

5.1.3 *Rezolvare detaliată

5.2 Dragoni

Problema 2 - Dragoni

100 de puncte

Supărați că lansarea părții a treia a filmului lor preferat s-a amânat până în iunie 2018, Henry și Hetty s-au gândit la propriul scenariu pentru finalul trilogiei:

Într-o lume în care vikingii pot zbura cu dragonii există N insule. Hiccup, șeful tribului de vikingi aflat pe insula 1, știe M rute directe de zbor bidirectionale între insule. Pentru fiecare j între 1 și M , ruta j unește insulele A_j și B_j și are lungimea D_j .

Pe fiecare insulă i , ($1 \leq i \leq n$) există dragoni din specia i care pot zbura fără a se opri pentru odihnă o distanță maximă $D_{max,i}$. Cu alte cuvinte, dragonii de pe insula i vor putea parcurge orice rută j , ($1 \leq j \leq m$) pentru care $D_j \leq D_{max,i}$, indiferent de ce alte drumuri au făcut anterior.

Hiccup dorește să ajungă de pe insula 1 pe insula N pentru a-l salva pe Toothless, dragonul lui. Pentru a ajunge acolo, el va lua inițial un dragon din specia 1 (de pe insula 1). Apoi, dacă la un moment dat Hiccup se află pe o insulă i , ($1 \leq i \leq n$) având cu el un dragon din specia t , el poate:

1. Să zboare de pe insula i pe o altă insulă x cu dragonul pe care îl are, folosind o rută directă j între insulele i și x , bineînțeles doar dacă $D_j \leq D_{max,t}$.
2. Să schimbe dragonul din specia t pe care îl are cu un dragon din specia i aflat pe insula respectivă.

Cerințe

a. Să se determine distanța maxima $D_{max,i}$ caracteristică unui dragon la care Hiccup poate ajunge fără a schimba dragonul pe care l-a luat inițial de pe insula 1.

b. Să se determine distanța minimă pe care Hiccup trebuie să o parcurgă pentru a ajunge de pe insula 1 pe insula N .

Date de intrare

Fișierul de intrare **dragoni.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 se găsesc două numere naturale N și M reprezentând numărul de insule, respectiv numărul de rute directe între insule. Pe a treia linie se găsesc N numere naturale, al i -lea dintre acestea reprezentând distanța maximă $D_{max,i}$ pe care o poate zbura un dragon de pe insula i . Pe următoarele M linii sunt

descrise cele M rute directe. Pe fiecare dintre aceste linii se găsesc câte trei numere naturale A , B și D cu semnificația că există rută bidirectională de lungime D între insulele A și B .

Date de ieșire

În fișierul de ieșire **dragoni.out** se va afișa un singur număr natural.

Dacă valoarea lui p este 1, se rezolvă numai cerința **a**.

În acest caz numărul afișat va reprezenta distanța maximă D_{max_i} a unui dragon i la care Hiccup poate ajunge fără a schimba dragonul pe care l-a luat inițial de pe insula 1.

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

În acest caz numărul afișat va reprezenta distanța minima pe care Hiccup trebuie să o parcurgă pentru a ajunge de pe insula 1 pe insula N .

Restricții și precizări

- $1 \leq N \leq 800$
- $1 \leq M \leq 6000$
- $1 \leq D_{max_i} \leq 50000$, pentru orice $1 \leq i \leq N$.
- $1 \leq A_j, B_j \leq N$, pentru orice $1 \leq j \leq M$.
- $1 \leq D_j \leq 50000$, pentru orice $1 \leq j \leq M$.
- Se garantează că Hiccup poate ajunge pe insula N .
- Se garantează că răspunsul oricărei cerințe este un număr natural mai mic decât 10^9 .
- Pentru rezolvarea corectă a primei cerințe se acordă 20% din punctajul testului respectiv.
- Pentru rezolvarea corectă a celei de-a doua cerințe se acordă 80% din punctajul testului respectiv.

Exemple

dragoni.in	dragoni.out	Explicații
1 5 6 6 3 13 20 26 1 2 5 1 3 7 1 5 10 2 3 6 3 4 5 3 5 14	20	<p>$P = 1$ deci se va rezolva cerința a).</p> <p>Există $N = 5$ insule și $M = 6$ rute între ele. Hiccup pornește de pe insula 1 având un dragon care poate zbura o distanță de maxim 6. Cu acest dragon poate ajunge doar pe insulele 1, 2, 3 și 4, întrucât pentru a ajunge pe insula 5 el ar fi obligat să parcurgă o rută de lungime mai mare decât 6.</p> <p>Distanța maximă pe care o poate zbura un dragon aflat pe insulele 1, 2, 3 sau 4 este deci 20 (dragonul de pe insula 4). Se observă că dragonul care poate zbura o distanță de 26 se află pe insula 5 și este inaccesibil.</p>
2 5 6 6 3 13 20 26 1 2 5 1 3 7 1 5 10 2 3 6 3 4 5 3 5 14	28	<p>$P = 2$ deci se va rezolva cerința b).</p> <p>Există $N = 5$ insule și $M = 6$ rute între ele. Pentru a parcurge o distanță minimă de 28 între insulele 1 și N, Hiccup face următorii pași:</p> <p>Zboară de pe insula 1 pe insula 2 o distanță de 5 cu dragonul din specia 1.</p> <p>Zboară de pe insula 2 pe insula 3 o distanță de 6 cu dragonul din specia 1.</p> <p>Schimbă dragonul din specia 1 cu dragonul aflat pe insula 3, care poate zbura o distanță maximă de 13.</p> <p>Zboară de pe insula 3 pe insula 1 o distanță de 7 cu dragonul din specia 3.</p> <p>Zboară de pe insula 1 pe insula 5 o distanță de 10 cu dragonul din specia 3.</p> <p>În total el parcurge o distanță de $5 + 6 + 7 + 10 = 28$.</p>

Timp maxim de executare/test: **1.5** secunde
Memorie: total **32 MB** din care pentru stivă **8 MB**
Dimensiune maximă a sursei: **10 KB**

5.2.1 Indicații de rezolvare

Vlad-Alexandru Gavrilă, Universitatea Cambridge

Soluție oficială - Vlad-Alexandru Gavrilă

Pentru a rezolva doar cerința a), se vor păstra în graf doar acele muchii care pot fi parcuse de dragonul aflat pe insula 1 (care au lungimea mai mică decât distanța maximă $Dmax[1]$). Apoi se va face o *parcursere BFS* sau *DFS* din nodul 1 pe acest graf pentru a se determina multimea M de noduri care pot fi vizitate pornind din nodul 1 folosind doar dragonul 1. Răspunsul va fi reprezentat de valoarea $Dmax[x]$ maximă pentru care x aparține M .

Pentru a rezolva cerința b), vom construi un nou graf în care putem reprezenta corect mișările pe care le poate face Hiccup. Astfel, un nod va fi reprezentat de o pereche (i, j) ($1 \leq i, j \leq N$), cu următoarea semnificație: *Hiccup se află în nodul i având la el un dragon de tipul j* . Apoi vom trasa muchiile în acest nou graf corespunzător cu mișările pe care le poate face Hiccup:

1. Zbor:

Pentru fiecare muchie din graful inițial $A[i]$, $B[i]$, $D[i]$, și fiecare j între 1 și N , avem muchie bidirectională în graful nou între nodurile $(A[i], j)$, $(B[i], j)$ de lungime $D[i]$ doar dacă $D[i] \leq Dmax[j]$. Această muchie corespunde unui zbor între nodurile $A[i]$ și $B[i]$ cu dragonul j .

2. Schimbarea dragonului:

Pentru fiecare nod i din graful original și fiecare j între 1 și N , avem muchie unidirecțională între nodurile (i, j) și (i, i) de cost 0 în graful nou. Această muchie corespunde schimbării unui dragon arbitrar j cu un dragon din specia aflată în nodul i .

Pe acest graf nou construit se va aplica *algoritmul lui Dijkstra* pornind din nodul $(1, 1)$. Soluția va fi distanța minimă în care putem accesa unul din nodurile $(N, 1)$, $(N, 2)$, ..., (N, n) , deoarece nu ne interesează cu ce dragon ajungem în nodul N .

De menționat este că graful nu trebuie reținut efectiv în memorie, deoarece această abordare nu se încadrează în limitele problemei. Observăm că pentru orice i între 1 și M și orice j între 1 și N , avem $(A[i], j) \rightarrow (B[i], j)$ în graful nou doar dacă $D[i] \leq Dmax[j]$ - această verificare putând fi făcută în cadrul algoritmului lui Dijkstra când încercăm să vedem dacă o muchie poate îmbunătăți vreo distanță. Muchiile de tip $(i, j) \rightarrow (i, i)$ pot fi, din nou, generate pe parcurs, fără a trebui să fie reținute.

Complexitate timp: $O(M * N \log N)$ timp, $O(N + M)$ memorie.

Soluție alternativă (pentru cerința 2) - Adrian Panaete

Se consideră multimea tuturor perechilor insula-dragon. Pentru fiecare pereche se calculează distanța minimă parcursă pentru a ajunge pe insula respectivă cu dragonul respectiv. Inițial avem doar dragonul 1 pe insula 1 cu distanță 0. În continuare vom alege mereu perechea de distanță minimă disponibilă. Aceasta va corespunde unei insule și unui dragon.

Se alege cel mai bun între dragonul respectiv și dragonul corespunzător insulei și se va încerca îmbunătățirea distanțelor la vecinii insulei care sunt situați la distanță mai mică decât $Dmax$ pentru dragonul respectiv. Pentru menținerea sortată a perechilor se poate folosi o structură de tip *set* sau *heap*. Pentru a evita utilizarea unei astfel de structuri poate fi folosit principiul din *algoritmul Bellman-Ford*. Mai precis se menține o *coadă* cu toate perechile în care a avut loc o îmbunătățire a distanței și se procesează elementele din coadă. Pentru eficientizare o pereche poate fi marcată la intrarea în coadă și demarcată la părăsirea ecsteia.

Dacă insula de destinație este chiar insula N , în loc să introducem perechea în coadă se updatează soluția.

5.2.2 Cod sursă

```

1 #include <cstdio>
2 #include <algorithm>
3 #include <vector>
4
5 using namespace std;
6
7 #define maxn 810
8 #define maxm 6010
9 #define inf (1LL*1000000000*1000000000)
10
11 int cer, n, m, a[maxm], b[maxn], d[maxn], dmax[maxn];
12 int f[maxn*maxn];
13 long long dist[maxn*maxn];
14
15 vector<int> v[maxn], w[maxn];
16 vector<pair<long long, int>> h;
17
18 int getNode(int lvl, int nod)
19 {
20     return (lvl-1)*n+nod;
21 }
22
23 void dijkstra()
24 {
25     for(int i=1; i<=n*n; ++i)
26         dist[i]=inf;
27
28     dist[1]=0;
29     h.push_back(make_pair(0, 1));
30
31     int nod;
32     long long cdist;
33
34     while(!h.empty())
35     {
36         nod=h[0].second;
37
38         if(nod==n*n)
39             break;
40
41         cdist=-h[0].first;
42         pop_heap(h.begin(), h.end());
43         h.pop_back();
44
45         if(f[nod])
46             continue;
47
48         f[nod]=1;
49         int fiu, dragon = (nod-1)/n+1, node = (nod-1)%n+1;
50
51         fiu = getNode(node, node);
52         if(dist[fiu]>cdist)
53         {
54             dist[fiu]=cdist;
55             h.push_back(make_pair(-dist[fiu], fiu));
56             push_heap(h.begin(), h.end());
57         }
58
59         for(int i=0; i<v[node].size(); ++i)
60         {
61             fiu = getNode(dragon, v[node][i]);
62             if(dist[fiu]>cdist+w[node][i] && w[node][i]<=dmax[dragon])
63             {
64                 dist[fiu]=cdist+w[node][i];
65                 h.push_back(make_pair(-dist[fiu], fiu));
66                 push_heap(h.begin(), h.end());
67             }
68         }
69     }
70 }
71
72 int cerintal()
73 {
74     int best=dmax[1];
75 }
```

```

76     for(int i=2; i<=n; ++i)
77         if(dist[i]!=inf && dmax[i]>best)
78             best=dmax[i];
79
80     return best;
81 }
82
83 int main()
84 {
85     freopen("dragoni.in", "r", stdin);
86     freopen("dragoni.out", "w", stdout);
87
88     scanf("%d", &cer);
89     scanf("%d%d", &n, &m);
90     for(int i=1; i<=n; ++i)
91         scanf("%d", &dmax[i]);
92
93     for(int i=1; i<=m; ++i)
94     {
95         scanf("%d%d%d", &a[i], &b[i], &d[i]);
96         v[a[i]].push_back(b[i]);
97         v[b[i]].push_back(a[i]);
98         w[a[i]].push_back(d[i]);
99         w[b[i]].push_back(d[i]);
100    }
101
102    dijkstra();
103
104    if(cer==1)
105        printf("%d\n", cerintal());
106    else
107        printf("%lld\n", dist[n*n]);
108
109    return 0;
110 }
```

Listing 5.2.2: Dragoni_GD.cpp

```

1 #include <iostream>
2 #include <vector>
3 #include <utility>
4 #include <deque>
5
6 #define tip long long
7 #define mp make_pair
8 #define IN first
9 #define DR second
10
11 using namespace std;
12
13 ifstream f("dragoni.in");
14 ofstream g("dragoni.out");
15
16 int t,n,m,P[1005][1005],D[1005],Q1[1005],q1[1005],b,LH,T[1005][1005];
17
18 vector<int> V[1005];
19 vector<pair<int,int> > v[1005];
20
21 deque<int> Q;
22
23 void cerintal(),cerinta2();
24
25 int main()
26 {
27     f>>t>>n>>m;
28     for(int i=1;i<=n;i++) f>>D[i];
29     if(t==1)
30         cerintal();
31     else
32         cerinta2();
33     return 0;
34 }
35
36 void cerintal()
37 {
```

```

38     int x,y,d,sol;
39     for(;m;m--)
40     {
41         f>>x>>y>>d;
42         if(d<=D[1])
43         {
44             V[x].push_back(y);
45             V[y].push_back(x);
46         }
47     }
48
49     Q1[1]=1;
50     q1[1]=1;
51     t=b=1;
52     for(;b<=t;)
53     {
54         x=Q1[b++];
55         for(vector<int>::iterator it=V[x].begin();it!=V[x].end();it++)
56             if(!q1[*it])
57             {
58                 Q1[++t]=*it;
59                 q1[*it]=1;
60             }
61     }
62
63     sol=D[1];
64     for(int i=1;i<=n;i++)
65         if(q1[i])
66             sol=max(sol,D[i]);
67     g<<sol;
68 }
69
70 void cerinta2()
71 {
72     int x,y,d,i,j,k,oo=1000000000;
73     pair<int,int> Per;
74     for(int i=1;i<=m;i++)
75     {
76         f>>x>>y>>d;
77         v[x].push_back(make_pair(d,y));
78         v[y].push_back(make_pair(d,x));
79     }
80
81     for(i=1;i<=n;i++)
82     {
83         for(j=1;j<=n;j++) T[i][j]=oo;
84         T[i][i]=0;
85         Q.push_back(i);
86         q1[i]=1;
87         for(;Q.size();)
88         {
89             j=Q.front();
90             Q.pop_front();
91             q1[j]=0;
92             for(vector<pair<int,int> >::iterator it=v[j].begin();
93                  it!=v[j].end();
94                  it++)
95                 if(D[i]>=it->first)
96                     if(T[i][it->second]>T[i][j]+it->first)
97                     {
98                         T[i][it->second]=T[i][j]+it->first;
99                         if(!q1[it->second])
100                         {
101                             q1[it->second]=1;
102                             Q.push_back(it->second);
103                         }
104                     }
105     }
106 }
107
108 for(i=1;i<=n;i++) D[i]=oo;
109 D[1]=0;
110 Q.push_back(1);
111 q1[1]=1;
112 for(;Q.size();)
113 {

```

```

114         j=Q.front();
115         Q.pop_front();
116         q1[j]=0;
117         for(k=1;k<=n;k++)
118             {
119                 if(D[k]>D[j]+T[j][k])
120                     D[k]=D[j]+T[j][k];
121                     if(!q1[k])
122                     {
123                         q1[k]=1;
124                         Q.push_back(k);
125                     }
126             }
127         }
128     g<<D[n];
129 }

```

Listing 5.2.3: dragoni_noGraph_int.cpp

```

1 #include <cstdio>
2 #include <algorithm>
3 #include <vector>
4
5 using namespace std;
6
7 #define maxn 810
8 #define maxm 6010
9 #define inf 1000000000
10
11 int cer, n, m, a[maxm], b[maxm], d[maxm], dmax[maxn];
12 int f[maxn*maxn];
13 int dist[maxn*maxn];
14
15 vector<int> v[maxn], w[maxn];
16 vector<pair<int, int> > h;
17
18 int getNode(int lvl, int nod)
19 {
20     return (lvl-1)*n+nod;
21 }
22
23 void dijkstra()
24 {
25     for(int i=1; i<=n*n; ++i)
26         dist[i]=inf;
27
28     dist[1]=0;
29     h.push_back(make_pair(0, 1));
30
31     int nod;
32     long long cdist;
33
34     while(!h.empty())
35     {
36         nod=h[0].second;
37
38         if(nod==n*n)
39             break;
40
41         cdist=-h[0].first;
42         pop_heap(h.begin(), h.end());
43         h.pop_back();
44
45         if(f[nod])
46             continue;
47
48         f[nod]=1;
49         int fiu, dragon = (nod-1)/n+1, node = (nod-1)%n+1;
50
51         fiu = getNode(node, node);
52         if(dist[fiu]>cdist)
53         {
54             dist[fiu]=cdist;
55             h.push_back(make_pair(-dist[fiu], fiu));

```

```

56             push_heap(h.begin(), h.end());
57         }
58
59         for(int i=0; i<v[node].size(); ++i)
60         {
61             fiu = getNode(dragon, v[node][i]);
62             if(dist[fiu]>cdist+w[node][i] && w[node][i]<=dmax[dragon])
63             {
64                 dist[fiu]=cdist+w[node][i];
65                 h.push_back(make_pair(-dist[fiu], fiu));
66                 push_heap(h.begin(), h.end());
67             }
68         }
69     }
70 }
71
72 int cerintal()
73 {
74     int best=dmax[1];
75
76     for(int i=2; i<=n; ++i)
77         if(dist[i]!=inf && dmax[i]>best)
78             best=dmax[i];
79
80     return best;
81 }
82
83 int main()
84 {
85     freopen("dragoni.in", "r", stdin);
86     freopen("dragoni.out", "w", stdout);
87
88     scanf("%d", &cer);
89     scanf("%d%d", &n, &m);
90     for(int i=1; i<=n; ++i)
91         scanf("%d", &dmax[i]);
92
93     for(int i=1; i<=m; ++i)
94     {
95         scanf("%d%d%d", &a[i], &b[i], &d[i]);
96         v[a[i]].push_back(b[i]);
97         v[b[i]].push_back(a[i]);
98         w[a[i]].push_back(d[i]);
99         w[b[i]].push_back(d[i]);
100    }
101
102    dijkstra();
103
104    if(cer==1)
105        printf("%d\n", cerintal());
106    else
107        printf("%d\n", dist[n*n]);
108
109    return 0;
110}

```

Listing 5.2.4: dragoni_WithGraphMLE.cpp

```

1 #include <iostream>
2 #include <algorithm>
3 #include <vector>
4
5 using namespace std;
6
7 #define maxn 810
8 #define maxm 6010
9 #define inf 1000000000
10
11 int cer, n, m, a[maxm], b[maxm], d[maxm], dmax[maxn];
12 int f[maxn*maxn];
13 int dist[maxn*maxn];
14
15 vector<int> v[maxn*maxn], w[maxn*maxn];
16 vector<pair<int, int> > h;
17

```

```

18 int getNode(int nod, int lvl)
19 {
20     return (lvl-1)*n+nod;
21 }
22
23 void addEdge(int a, int l1, int b, int l2, int dst)
24 {
25     int n1=getNode(a, l1), n2=getNode(b, l2);
26     v[n1].push_back(n2);
27     w[n1].push_back(dst);
28 }
29
30 void buildGraph()
31 {
32     for(int i=1; i<=m; ++i)
33         for(int j=1; j<=n; ++j)
34             if(d[i]<=dmax[j])
35             {
36                 addEdge(a[i], j, b[i], j, d[i]);
37                 addEdge(b[i], j, a[i], j, d[i]);
38             }
39
40     for(int i=1; i<=n; ++i)
41         for(int j=1; j<=n; ++j)
42             addEdge(i, j, i, j, 0);
43 }
44
45 void dijkstra()
46 {
47     for(int i=1; i<=n*n; ++i)
48         dist[i]=inf;
49
50     dist[1]=0;
51     h.push_back(make_pair(0, 1));
52
53     int nod;
54     long long cdist;
55
56     while(!h.empty())
57     {
58         nod=h[0].second;
59
60         if(nod==n*n)
61             break;
62
63         cdist=-h[0].first;
64         pop_heap(h.begin(), h.end());
65         h.pop_back();
66
67         if(f[nod])
68             continue;
69
70         f[nod]=1;
71         int fiu;
72
73         for(int i=0; i<v[nod].size(); ++i)
74         {
75             fiu=v[nod][i];
76             if(dist[fiu]>cdist+w[nod][i])
77             {
78                 dist[fiu]=cdist+w[nod][i];
79                 h.push_back(make_pair(-dist[fiu], fiu));
80                 push_heap(h.begin(), h.end());
81             }
82         }
83     }
84 }
85
86 int cerintal()
87 {
88     int best=dmax[1];
89
90     for(int i=2; i<=n; ++i)
91         if(dist[i]!=inf && dmax[i]>best)
92             best=dmax[i];
93 }
```

```

94     return best;
95 }
96
97 int main()
98 {
99     freopen("dragoni.in", "r", stdin);
100    freopen("dragoni.out", "w", stdout);
101
102    scanf("%d", &cer);
103    scanf("%d%d", &n, &m);
104    for(int i=1; i<=n; ++i)
105        scanf("%d", &dmax[i]);
106    for(int i=1; i<=m; ++i)
107        scanf("%d%d%d", &a[i], &b[i], &d[i]);
108
109    buildGraph();
110    dijkstra();
111
112    if(cer==1)
113        printf("%d\n", cerintal());
114    else
115        printf("%d\n", dist[n*n]);
116
117    return 0;
118 }
```

Listing 5.2.5: dragoniBeFo.cpp

```

1 #include <iostream>
2 #include <vector>
3 #include <deque>
4 #include <utility>
5 #include <algorithm>
6
7 #define tip long long
8
9 using namespace std;
10
11 ifstream f("dragoni.in");
12 ofstream g("dragoni.out");
13
14 const int N=1005;
15 int n,m,i,j,x,y,d,ic,t,b,q[N][N],Q1[N],q1[N];
16 tip Dist[N][N],dist,oo,dc,D[N],sol;
17
18 vector<pair<tip,int>> v[N];
19 vector<int> V[N];
20
21 deque<pair<int,int>> Q;
22
23 void cerintal(),cerinta2();
24
25 int main()
26 {
27     f>>t>>n>>m;
28     for(i=1;i<=n;i++)
29         f>>D[i];
30     if(t==1)cerintal();
31     else cerinta2();
32     return 0;
33 }
34
35 void cerintal()
36 {
37     for(;m;m--)
38     {
39         f>>x>>y>>d;
40         if(d<=D[1])
41         {
42             V[x].push_back(y);
43             V[y].push_back(x);
44         }
45     }
46
47     Q1[1]=1;q1[1]=1;t=b=1;
```

```

48     for(;b<=t;)
49     {
50         x=Q1[b++];
51         for(vector<int>::iterator it=V[x].begin();it!=V[x].end();it++)
52             if(!q1[*it])
53             {
54                 Q1[++t]=*it;
55                 q1[*it]=1;
56             }
57     }
58 }
59
60 sol=D[1];
61 for(i=1;i<=n;i++)
62     if(q1[i])
63         sol=max(sol,D[i]);
64 g<<sol;
65 }
66
67 void cerinta2()
68 {
69     //creez liste de vecini ca perechi (distanta,vecin)
70     for(i=1;i<=m;i++)
71     {
72         f>>x>>y>>d;
73         v[x].push_back(make_pair(d,y));
74         v[y].push_back(make_pair(d,x));
75     }
76
77     //sortez liste de vecini dupa distante
78     for(i=1;i<=n;i++)
79         sort(v[i].begin(),v[i].end());
80
81     //initializez matricea distanteelor
82     oo=1LL*1000000000*1000000000;
83     for(i=1;i<=n;i++)
84         for(j=1;j<=n;j++)
85             Dist[i][j]=oo;
86
87     //initializez coada Bellman-Ford
88     Q.push_back(make_pair(1,1));
89     q[1][1]=1;
90     Dist[1][1]=0;
91     sol=oo;
92     for(;Q.size();)
93     {
94         //scot primul element din coada , ii memorez insula, dragonul si
95         //iau distanta corespunzatoare
96         ic=Q.front().first;
97         dc=Q.front().second;
98         q[ic][dc]=0;
99         dist=Dist[ic][dc];
100        Q.pop_front();
101
102        //daca gasesc dragon mai bun schimb dragonul
103        if(D[ic]>D[dc]) dc=ic;
104
105        //parcurg vecinii pe care dragonul ii poate atinge
106        for(vector<pair<tip,int>>::iterator it=v[ic].begin();
107            it!=v[ic].end()&&D[dc]>=it->first;
108            it++)
109        {
110            //daca obtin o distanta mai proasta decat solutia
111            //sau mai proasta decaat distanta pentru insula vecina cu
112            //dragonul curent
113            //atunci renunt sa ma deplasez acolo
114            if(dist+it->first>=sol||dist+it->first>=Dist[it->second][dc])
115                continue;
116
117            //daca vecinul e destinatia updatez solutia
118            if(it->second==n)
119            {
120                sol=dist+it->first;
121                continue;
122            }
123

```

```

124         //altfel updatez distanta pentru vecin cu dragonul curent
125         Dist[it->second][dc]=dist+it->first;
126
127         //daca prerechea vecin,dragon nu e in coada o adaug
128         if(!q[it->second][dc])
129         {
130             q[it->second][dc]=1;
131             Q.push_back(make_pair(it->second,dc));
132         }
133     }
134 }
135 g<<sol;
136 }
```

Listing 5.2.6: dragoniHeap.cpp

```

1 #include <iostream>
2 #include <vector>
3 #include <utility>
4 #include <algorithm>
5
6 #define tip long long
7 #define mp make_pair
8 #define IN first
9 #define DR second
10
11 using namespace std;
12
13 ifstream f("dragoni.in");
14 ofstream g("dragoni.out");
15
16 int t,n,m,P[1005][1005],D[1005],Q1[1005],q1[1005],b,LH,T[1005][1005];
17
18 vector<int> V[1005];
19 vector<pair<int,int> > v[1005];
20 pair<int,int> H[1000005];
21
22 void cerinta1(),cerinta2(),Hd(int),Hu(int);
23
24 int main()
25 {
26     f>>t>>n>>m;
27     for(int i=1;i<=n;i++) f>>D[i];
28     if(t==1)
29         cerinta1();
30     else
31         cerinta2();
32     return 0;
33 }
34
35 void cerinta1()
36 {
37     int x,y,d,sol;
38     for(;m;m--)
39     {
40         f>>x>>y>>d;
41         if(d<=D[1])
42         {
43             V[x].push_back(y);
44             V[y].push_back(x);
45         }
46     }
47
48     Q1[1]=1;
49     q1[1]=1;
50     t=b=1;
51     for(;b<=t;)
52     {
53         x=Q1[b++];
54         for(vector<int>::iterator it=V[x].begin();it!=V[x].end();it++)
55             if(!q1[*it])
56             {
57                 Q1[++t]=*it;
58                 q1[*it]=1;
59             }
60     }
61 }
```

```

60      }
61
62      sol=D[1];
63      for(int i=1;i<=n;i++)
64      {
65          if(q1[i])
66              sol=max(sol,D[i]);
67      g<<sol;
68  }
69 void cerinta2()
70 {
71     int x,y,d,ins,dra,oo=2000000000;
72     pair<int,int> Per;
73     for(int i=1;i<=m;i++)
74     {
75         f>>x>>y>>d;
76         v[x].push_back(make_pair(d,y));
77         v[y].push_back(make_pair(d,x));
78     }
79
80     for(ins=1;ins<=n;ins++)
81         sort(v[ins].begin(),v[ins].end());
82
83     for(ins=1;ins<=n;ins++)
84         for(dra=1;dra<=n;dra++)
85         {
86             LH++;
87             H[LH].IN=ins;
88             H[LH].DR=dra;
89             P[ins][dra]=LH;
90             T[ins][dra]=oo;
91         }
92
93     T[1][1]=0;
94     for(;;)
95     {
96         Per=H[1];
97         if(Per.IN==n)
98         {
99             g<<T[Per.IN][Per.DR];
100            return;
101        }
102
103        P[Per.IN][Per.DR]=n+1;
104        H[1]=H[LH--];
105        P[H[1].IN][H[1].DR]=1;
106        Hd(1);
107        for(int i=0; i<v[Per.IN].size(); ++i)
108        {
109            pair<int, int> it = v[Per.IN][i];
110            if(D[Per.DR]<it.first) break;
111            ins=it.second;
112            dra=D[Per.DR]>D[ins] ? Per.DR : ins;
113            if(P[ins][dra]>LH||(T[ins][dra]<=T[Per.IN][Per.DR]+it.first))
114                continue;
115            T[ins][dra]=T[Per.IN][Per.DR]+it.first;
116            Hu(P[ins][dra]);
117        }
118    }
119 }
120
121 void Hu(int son)
122 {
123     int dad=son/2;
124     if(!dad) return;
125     if(T[H[dad]].IN[H[dad].DR]>T[H[son]].IN[H[son].DR])
126     {
127         pair<int,int>PerAux=H[son];
128         H[son]=H[dad];
129         H[dad]=PerAux;
130         P[H[son].IN][H[son].DR]=son;
131         P[H[dad].IN][H[dad].DR]=dad;Hu(dad);
132     }
133 }
134
135 void Hd(int dad)

```

```

136    {
137        int son=2*dad;
138        if(son>LH) return;
139        if(son<LH)
140            if(T[H[son].IN][H[son].DR]>T[H[son+1].IN][H[son+1].DR])
141                son++;
142        if(T[H[dad].IN][H[dad].DR]>T[H[son].IN][H[son].DR])
143        {
144            pair<int,int>PerAux=H[son];
145            H[son]=H[dad];
146            H[dad]=PerAux;
147            P[H[son].IN][H[son].DR]=son;
148            P[H[dad].IN][H[dad].DR]=dad; Hd(son);
149        }
150    }

```

Listing 5.2.7: dragoniRF.cpp

```
1 #include <fstream>
2 #include <vector>
3 #include <utility>
4 #include <deque>
5
6 #define tip long long
7 #define mp make_pair
8 #define IN first
9 #define DR second
10
11 using namespace std;
12
13 ifstream f("dragoni.in");
14 ofstream g("dragoni.out");
15
16 int t,n,m,P[1005][1005],D[1005],Q1[1005],q1[1005],b,LH,T[1005][1005];
17
18 vector<int> V[1005];
19 vector<pair<int,int>> v[1005];
20
21 deque<int> Q;
22
23 void cerintal(),cerinta2();
24
25 int main()
26 {
27     f>>t>>n>>m;
28     for(int i=1;i<=n;i++) f>>D[i];
29     if(t==1)cerintal();else cerinta2();
30     return 0;
31 }
32 void cerintal()
33 {
34     int x,y,d,sol;
35     for(;m;m--)
36     {
37         f>>x>>y>>d;
38         if(d<=D[1])
39         {
40             V[x].push_back(y);
41             V[y].push_back(x);
42         }
43     }
44
45     Q1[1]=1;
46     q1[1]=1;
47     t=b=1;
48     for(;b<=t;)
49     {
50         x=Q1[b++];
51         for(vector<int>::iterator it=V[x].begin();it!=V[x].end();it++)
52             if(!q1[*it])
53             {
54                 Q1[++t]=*it;
55                 q1[*it]=1;
56             }
57     }
58 }
```

```

58
59     sol=D[1];
60     for(int i=1;i<=n;i++)
61         if(q1[i])
62             sol=max(sol,D[i]);
63     g<<sol;
64 }
65
66 void cerinta2()
67 {
68     int x,y,d,i,j,k,oo=1000000000;
69     pair<int,int> Per;
70     for(int i=1;i<=m;i++)
71     {
72         f>>x>>y>>d;
73         v[x].push_back(make_pair(d,y));
74         v[y].push_back(make_pair(d,x));
75     }
76
77     for(i=1;i<=n;i++)
78     {
79         for(j=1;j<=n;j++) T[i][j]=oo;
80         T[i][i]=0;
81         Q.push_back(i);
82         q1[i]=1;
83         for(;Q.size();)
84         {
85             j=Q.front();
86             Q.pop_front();
87             q1[j]=0;
88             for(vector<pair<int,int> >::iterator it=v[j].begin();
89                  it!=v[j].end();
90                  it++)
91                 if(D[i]>=it->first)
92                     if(T[i][it->second]>T[i][j]+it->first)
93                     {
94                         T[i][it->second]=T[i][j]+it->first;
95                         if(!q1[it->second])
96                         {
97                             q1[it->second]=1;
98                             Q.push_back(it->second);
99                         }
100                     }
101     }
102 }
103
104 for(k=1;k<=n;k++)
105     for(i=1;i<=n;i++)
106         for(j=1;j<=n;j++)
107             if(i!=k&&i!=j&&k!=j&&T[i][j]>T[i][k]+T[k][j])
108                 T[i][j]=T[i][k]+T[k][j];
109
110     g<<T[1][n];
111 }
```

5.2.3 *Rezolvare detaliată

Capitolul 6

OJI 2014

6.1 cartite

Problema 1 - cartite

100 de puncte

Cârtițele sunt animale de dimensiuni mici care își duc traiul pe suprafețe de teren deschis, având ca dușman principal vulpea. Lângă o pădure se află o zonă agricolă în formă dreptunghiulară, împărțită în pătrățele de aceeași dimensiune. Zona agricolă este reprezentată printr-un tablou bidimensional cu m linii și n coloane, având liniile și coloanele numerotate începând cu 1. În această zonă agricolă trăiește o cârtiță și k vulpi.

Pentru cârtiță cunoaștem coordonatele ei (linia și coloana) pe care se găsește, la fel și pentru vulpi, care stau la pândă pentru a ataca cârtița în momentele ei de neatenție.

Pe suprafața terenului cârtița se poate deplasa din pătrățelul în care se află doar într-unul dintre cele 4 pătrățele vecine pe direcțiile nord, sud, est sau vest.

Vulpile pot ataca instantaneu pe o raza de acțiune de lungime 0, 1 sau 2 pe orizontală și verticală, inclusiv în poziția unde se găsesc, după care tot instantaneu se întorc în pozițiile inițiale. În figura de mai jos sunt desenate pătrățele unde poate ataca o vulpe poziționată în pătrățelul cu cifra reprezentând raza de acțiune.

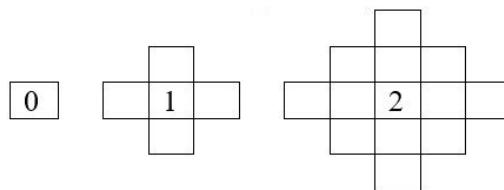


Figura 6.1: cartite

Pentru a micșora riscul de deplasare în zona agricolă cârtița sapă în pământ un sistem de g galerii, care leagă între ele pătrățele din zona agricolă. Aceste galerii nu se intersectează sub pământ, ci doar la suprafață, trecerea dintr-o galerie în alta, care se intersectează în același pătrățel făcându-se printr-un sistem ce nu îi pune viața în pericol.

Galeriile sunt indicate prin coordonatele pătrățelor pe care le unesc. Acestea sunt săpate astfel încât, dacă pornim dintr-un capăt al unei galerii le putem parcurge pe toate. Nu există două galerii care să pornească din același pătrățel și să ajungă tot în același pătrățel (galeriile sunt distințe).

Cârtița dorește să se plimbe prin toate galeriile de sub teren trecând o singură dată prin fiecare, dar pentru acest lucru trebuie să ajungă nevătămată mergând la suprafața terenului la un pătrățel de unde să intre în sistemul de galerii.

Cerințe

Determinații:

- cel mai apropiat pătrățel de poziția inițială a cârtiței prin care ea poate să intre în galerie pentru a se plimba, precum și lungimea traseului parcurs la suprafață astfel încât fiecare pătrățică de pe traseu să nu fie atacată de nicio vulpă;

b) traseul de plimbare numai prin galerii, specificat prin coordonatele pătrățelor care constituie capetelor acestora.

Date de intrare

Fișierul de intrare **cartite.in** conține următoarele date:

- pe prima linie un număr natural p , pentru toate testeile de intrare, numărul p poate avea doar valoarea 1 sau valoarea 2;
- pe a doua linie m, n dimensiunile zonei agricole;
- pe a treia linie coordonatele cărtiței;
- pe linia a patra numărul de vulpi k urmată de k linii cu câte trei numere naturale reprezentând coordonatele pătrățelor unde se găsesc vulpi și raza lor de acțiune (0, 1 sau 2);
- următoarea linie conține numărul de galerii g ;
- fiecare din următoarele g linii conține câte 4 numere naturale separate prin câte un spațiu, reprezentând coordonatele capetelor unei galerii.

Date de ieșire

Dacă valoarea lui p este 1, se va afișa numai rezultatul de la punctul a) din cerință. În acest caz, în fișierul de ieșire **cartite.out** se vor scrie trei numere naturale separate între ele prin câte un spațiu, reprezentând coordonatele pătrățelului unde va intra cărtița în galerii și lungimea traseului parcurs la suprafață.

Dacă valoarea lui p este 2, se va afișa numai rezultatul de la punctul b) din cerință. În acest caz, fișierul de ieșire **cartite.out** va conține coordonatele pătrățelor traseului de plimbare prin galerii (coordonatele câte unui pătrățel pe câte o linie, începând cu prima linie a fișierului de ieșire). Pătrățelul de pornire trebuie să fie același cu cel în care ajunge cărtița la sfârșitul plimbării și nu este obligatoriu să fie același cu cel în care ea intră în galerii de la suprafața terenului.

Restricții și precizări

- $1 \leq m, n \leq 200$.
- $1 \leq g \leq 100$
- $0 \leq k \leq 50$
- Lungimea traseului parcurs la suprafață este egală cu numărul de pătrățele prin care aceasta trece, dar diferite de cel din care pleacă.
- Fiecare dintre cerințe reprezintă 50% din punctaj.
- Cărtița nu poate intra în galerii printr-un pătrățel din raza de acțiune a unei vulpi.
- Pentru toate testeile există soluție la cerința a), adică există un traseu sigur de la cărtiță la o intrare într-o galerie.
- Soluția, nu este unică, însă, orice soluție corectă va obține punctajul maxim pe test.
- Inițial, cărtița se găsește pe o poziție în care nu este atacată de nicio vulpă.

Exemple

cartite.in	cartite.out	Explicații
1	4 2 3	$p = 1$
6 4		Deplasarea cărtiței pe suprafața agricolă se va face pe traseul de lungime 3 ce trece prin pătrățelele (6,3) (6,2) (5,2) (4,2).
6 3		
3		
5 1 0		
3 4 1		Coordonatele pătrățelului de intrare în galerii sunt (4,2).
4 3 0		
7		
1 1 3 2		
1 3 1 4		
1 1 3 3		
1 4 4 2		
4 2 3 3		
4 2 1 3		
4 2 3 2		

Atenție! Pentru acest test se va afișa doar rezultatul la cerința a).

2	1 1	p = 2
6 4	3 2	Traseul de plimbare prin galerii este următorul: (1,1) (3,2) (4,2) (1,3) (1,4) (4,2)(3,3)(1,1), cu observația că se putea alege și alt pătrâtel de plecare.
6 3	4 2	
3	1 3	
5 1 0	1 4	
3 4 1	4 2	Atenție! Pentru acest test se va afișa doar rezultatul la cerința b).
4 3 0	3 3	
7	1 1	
1 1 3 2		Se observă că în acest caz trebuie să se citească toate datele din fișier.
1 3 1 4		
1 1 3 3		
1 4 4 2		
4 2 3 3		
4 2 1 3		
4 2 3 2		

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

Memorie: total **64 MB** din care pentru stivă **16 MB**

Dimensiune maximă a sursei: **5 KB**

6.1.1 Indicații de rezolvare

Lector Univ. Dr. Popescu Doru Anastasiu - Universitatea din Pitești, Facultatea de Matematică-Informatică

Cerința a)

Folosind pozițiile vulpilor și razele lor de acțiune se construiește un tablou cu elemente 0 și 1, 0 - poziție accesibilă, 1 - poziție inaccesibilă cărtiței. Pentru acest tablou se determină distanța minimă de la o poziție dată (cea a cărtiței) la o mulțime de poziții mergând în cele 4 direcții (Nord, Sud, Est, Vest) cu o poziție la fiecare pas (*algoritmul lui Lee*). Dimensiunile reduse ale datelor de intrare și timpul maxim de executie/test nu conduc la probleme de memorie s-au timp de execuție.

Cerinta b)

În rezolvarea acestei cerințe este necesar să se construiască un graf neorientat ce are drept noduri, capetele galeriilor, iar drept muchii galeriile. Pentru acest lucru va trebui să se determine pătratele distincte ce sunt capete de galerie și să li se asocieze noduri în graf. Apoi, problema se reduce la determinarea unui *ciclu eulerian* într-un graf neorientat. Graful are un număr mic de noduri (maxim 200), datorită numărului mic de galerii (cel mult 100) și *algoritmul din teorema lui Euler* (pentru ciclu eulerian) este suficient pentru a obține punctajul maxim.

6.1.2 Cod sursă

Listing 6.1.1: AdrianPanaeteFleury2.cpp

```

1 #include <fstream>
2 #include <vector>
3
4 #define N 1<<16
5 #define I(x,y) (x<<8)|y
6 #define Afis(x) printf("%d %d\n",x>>8,x&255)
7
8 using namespace std;
9
10 int n,m,i,j,X,Y,vulpe=-1,capat=-2,baza,top,vulpi[N],capete[N],Lee[N],Q[N],
11 k,p,e,D[5]={-256,256,-1,1,0},*d,x[1001],y[1001],used[1001],cx,cy,lg,caz;
12
13 vector<int> v[N];
14
15 void fill(int,int),DF(int);
16
17 int main()
18 {

```

```

19     freopen("cartite.in", "r", stdin);
20     freopen("cartite.out", "w", stdout);
21     scanf("%d", &caz);
22     scanf("%d%d", &n, &m);
23     m++;
24     n++;
25     scanf("%d%d", &i, &j);
26     X=I(i, j);
27
28     for(j=0; j<=m; j++) vulpi[I(0, j)]=vulpi[I(n, j)]=1;
29     for(i=0; i<=n; i++) vulpi[I(i, 0)]=vulpi[I(i, m)]=1;
30
31     scanf("%d", &k);
32     for(; k; k--)
33     {
34         scanf("%d%d%d", &i, &j, &p);
35         fill(I(i, j), p);
36     }
37
38     scanf("%d", &e);
39     for(k=1; k<=e; k++)
40     {
41         scanf("%d%d", &i, &j);
42         x[k]=I(i, j);
43         v[x[k]].push_back(k);
44         scanf("%d%d", &i, &j);
45         y[k]=I(i, j);
46         v[y[k]].push_back(k);
47         capete[x[k]]=capete[y[k]]=1;
48     }
49
50     Q[1]=X;
51     top=1;
52     Lee[X]=1;
53     for(baza=1; baza<=top; baza++)
54     {
55         i=Q[baza];
56         for(d=D; *d; d++)
57         {
58             if(vulpi[i+d]) continue;
59             if(Lee[i+d]) continue;
60             Lee[i+d]=Lee[i]+1;
61             Q[++top]=i+d;
62             if(capete[i+d]) break;
63         }
64
65         if(capete[i+d])
66         {
67             cx=(i+d)>>8;
68             cy=(i+d)&255;
69             lg=Lee[i+d]-1;
70             break;
71         }
72     }
73
74     if(caz==1)
75     {
76         printf("%d %d %d\n", cx, cy, lg);
77         return 0;
78     }
79
80     for(i=1; i<=e; i++)
81     {
82         if(!vulpi[x[i]]){DF(x[i]); break;}
83         if(!vulpi[y[i]]){DF(x[i]); break;}
84     }
85     return 0;
86 }
87
88 void DF(int nod)
89 {
90     for(; v[nod].size();)
91     {
92         int edge=v[nod].back();
93         if(used[edge]) {v[nod].pop_back(); continue;}
94         used[edge]=1;

```

```

95         DF(x[edge]+y[edge]-nod);
96     }
97     Afis(nod);
98 }
99
100 void fill(int x,int y)
101 {
102     int *q;
103     if(x<0||y<0) return;
104     if(vulpi[x]) return;
105
106     vulpi[x]=1;
107
108     for(q=D; *q; q++) fill(x+*q,y-1);
109 }
```

Listing 6.1.2: AdrianPanaeteRecursiv1.cpp

```

1 #include <iostream>
2 #include <vector>
3
4 #define N 1<<16
5 #define I(x,y) (x<<8)|y
6 #define Afis(x) printf("%d %d\n",x>>8,x&255)
7
8 using namespace std;
9
10 int n,m,i,j,X,Y,vulpe=-1,capat=-2,baza,top,
11     vulpi[N],capete[N],Lee[N],Q[N],k,p,e,
12     D[5]={-256,256,-1,1,0},*d,x[1001],y[1001],used[1001],cx,cy,lg,caz;
13
14 vector<int> v[N];
15 void fill(int,int),DF(int);
16
17 int main()
18 {
19     freopen("cartite.in","r",stdin);
20     freopen("cartite.out","w",stdout);
21
22     scanf("%d",&caz);
23     scanf("%d%d",&n,&m);
24
25     m++;
26     n++;
27
28     scanf("%d%d",&i,&j);
29     X=I(i,j);
30
31     for(j=0;j<=m;j++)
32         vulpi[I(0,j)]=vulpi[I(n,j)]=1;
33     for(i=0;i<=n;i++)
34         vulpi[I(i,0)]=vulpi[I(i,m)]=1;
35
36     scanf("%d",&k);
37     for(;k;k--)
38     {
39         scanf("%d%d%d",&i,&j,&p);
40         fill(I(i,j),p);
41     }
42
43     scanf("%d",&e);
44     for(k=1;k<=e;k++)
45     {
46         scanf("%d%d",&i,&j);
47         x[k]=I(i,j);
48         v[x[k]].push_back(k);
49         scanf("%d%d",&i,&j);
50         y[k]=I(i,j);
51         v[y[k]].push_back(k);
52         capete[x[k]]=capete[y[k]]=1;
53     }
54
55     Q[1]=X;
56     top=1;
57     Lee[X]=1;
```

```

58     for (baza=1;baza<=top;baza++)
59     {
60         i=Q[baza];
61         for(d=D;*d;d++)
62         {
63             if(vulpi[i+d]) continue;
64             if(Lee[i+d]) continue;
65             Lee[i+d]=Lee[i]+1;
66             Q[++top]=i+d;
67             if(capete[i+d]) break;
68         }
69         if(capete[i+d])
70         {
71             cx=(i+d)>>8;
72             cy=(i+d)&255;
73             lg=Lee[i+d]-1;
74             break;
75         }
76     }
77 }
78
79 if(caz==1)
80 {
81     printf("%d %d %d\n",cx,cy,lg);
82     return 0;
83 }
84
85 for(i=1;i<=e;i++)
86 {
87     if(!vulpi[x[i]]){DF(x[i]);break;}
88     if(!vulpi[y[i]]){DF(y[i]);break;}
89 }
90
91 return 0;
92 }
93
94 void DF(int nod)
95 {
96     for(;v[nod].size();)
97     {
98         int edge=v[nod].back();
99         if(used[edge])
100         {
101             v[nod].pop_back();
102             continue;
103         }
104         used[edge]=1;
105         DF(x[edge]+y[edge]-nod);
106     }
107     Afis(nod);
108 }
109
110 void fill(int x,int y)
111 {
112     int *q;
113     if(x<0||y<0) return;
114     if(vulpi[x]) return;
115     vulpi[x]=1;
116     for(q=D;*q;q++)
117         fill(x+*q,y-1);
118 }
```

Listing 6.1.3: DoruAnastasiuPopescucartite.cpp

```

1 #include <cstdio>
2 #include <vector>
3
4 #define N 1<<16
5 #define I(x,y) (x<<8)|y
6 #define Afis(x) printf("%d %d\n",x>>8,x&255)
7
8 using namespace std;
9
10 int n,m,i,j,X,Y,vulpe=-1,catap=-2,baza,top,
11     vulpi[N],capete[N],Lee[N],Q[N],k,p,e,
```

```

12      D[5]={-256,256,-1,1,0},*d,x[1001],y[1001],used[1001],cx,cy,lg,caz;
13
14  vector<int> v[N];
15  void fill(int,int),DF(int);
16
17 int main()
18 {
19     freopen("cartite.in","r",stdin);
20     freopen("cartite.out","w",stdout);
21
22     scanf("%d",&caz);
23     scanf("%d%d",&n,&m);
24
25     m++;
26     n++;
27
28     scanf("%d%d",&i,&j);
29     X=I(i,j);
30
31     for(j=0;j<=m;j++)
32         vulpi[I(0,j)]=vulpi[I(n,j)]=1;
33     for(i=0;i<=n;i++)
34         vulpi[I(i,0)]=vulpi[I(i,m)]=1;
35
36     scanf("%d",&k);
37     for(;k;k--)
38     {
39         scanf("%d%d%d",&i,&j,&p);
40         fill(I(i,j),p);
41     }
42
43     scanf("%d",&e);
44     for(k=1;k<=e;k++)
45     {
46         scanf("%d%d",&i,&j);
47         x[k]=I(i,j);
48         v[x[k]].push_back(k);
49         scanf("%d%d",&i,&j);
50         y[k]=I(i,j);
51         v[y[k]].push_back(k);
52         capete[x[k]]=capete[y[k]]=1;
53     }
54
55     Q[1]=X;
56     top=1;
57     Lee[X]=1;
58     for(baza=1;baza<=top;baza++)
59     {
60         i=Q[baza];
61         for(d=D;*d;d++)
62         {
63             if(vulpi[i+d]) continue;
64             if(Lee[i+d]) continue;
65             Lee[i+d]=Lee[i]+1;
66             Q[++top]=i+d;
67             if(capete[i+d]) break;
68         }
69
70         if(capete[i+d])
71         {
72             cx=(i+d)>>8;
73             cy=(i+d)&255;
74             lg=Lee[i+d]-1;
75             break;
76         }
77     }
78
79     if(caz==1)
80     {
81         printf("%d %d %d\n",cx,cy,lg);
82         return 0;
83     }
84
85     for(i=1;i<=e;i++)
86     {
87         if(!vulpi[x[i]]){DF(x[i]);break;}

```

```

88         if (!vulpi[y[i]]) {DF(x[i]);break;}
89     }
90
91     return 0;
92 }
93
94 void DF(int nod)
95 {
96     for (; v[nod].size();)
97     {
98         int edge=v[nod].back();
99         if (used[edge])
100         {
101             v[nod].pop_back();
102             continue;
103         }
104         used[edge]=1;
105         DF(x[edge]+y[edge]-nod);
106     }
107     Afis(nod);
108 }
109
110 void fill(int x,int y)
111 {
112     int *q;
113     if (x<0||y<0) return;
114     if (vulpi[x]) return;
115     vulpi[x]=1;
116     for (q=D; *q; q++)
117         fill(x+*q,y-1);
118 }
```

6.1.3 *Rezolvare detaliată

6.2 fractii2

Problema 2 - fractii2

100 de puncte

Numărul 1 poate fi scris în diverse moduri ca sumă de fracții cu numărătorul 1 și numitorul o putere a lui 2. De exemplu:

$$1 = \frac{1}{2} + \frac{1}{2} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} = \frac{1}{8} + \frac{1}{4} + \frac{1}{2} + \frac{1}{8}$$

Două scrieri nu sunt considerate distincte dacă folosesc aceleași fracții scrise în altă ordine. În exemplul de mai sus ultimele două scrieri nu sunt distincte.

Cerințe

Pentru N - număr natural nenul să se determine:

- O modalitate de scriere a numărului 1 ca sumă de exact N fracții cu numărătorul 1 și numitorul o putere a lui 2.
- Numărul de scrieri distincte a numărului 1 ca sumă de exact N fracții cu numărătorul 1 și numitorul o putere a lui 2. Deoarece acest număr poate fi foarte mare acest număr trebuie calculat *modulo* 100003.

Date de intrare

Fișierul de intrare **fractii2.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 se găsește un singur număr N natural - reprezentând numărul de fracții.

Date de ieșire

Dacă valoarea lui p este 1, se va rezolva numai punctul a) din cerință. În acest caz, în fișierul de ieșire **fractii2.out** se vor scrie, pe o singură linie, N numere naturale separate prin căte un

spațiu reprezentând cei N exponenti ai lui 2 din scrierea solicitată în prima cerință. Astfel, dacă numerele afișate sunt m_1, m_2, \dots, m_N atunci există scrierea

$$1 = \frac{1}{2^{m_1}} + \frac{1}{2^{m_2}} + \dots + \frac{1}{2^{m_N}}.$$

Dacă valoarea lui p este 2, se va rezolva numai punctul b) din cerință. În acest caz, în fișierul de ieșire **fractii2.out** se va scrie un număr natural reprezentând răspunsul la a doua cerință, adică numărul de scrieri distințe a numărului 1 ca sumă de N fracții cu numărătorul 1 și numitorul o putere a lui 2 (*modulo* 100003).

Restricții și precizări

- $2 \leq N \leq 2000$
- Pentru prima cerință se acordă 20% din punctaj.
- Pentru a doua cerință de acordă 80% din punctaj.
- Rezultatul pentru a doua cerință trebuie afișat *modulo* 100003

Exemple

fractii2.in	fractii2.out	Explicații
1 4	2 2 2 2	<p>$p=1$ $1 = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} = \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4}$</p> <p>Răspunsul corespunde celei de-a doua scrieri dar există și alte variante corecte de răspuns. De exemplu, 3 1 2 3 se consideră răspuns corect.</p> <p>Atenție! Pentru acest test se va afișa doar rezultatul la cerința a).</p>
2 4	2	<p>$p=2$ $1 = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} = \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4}$</p> <p>Acestea sunt singurele scrieri distințe.</p> <p>Atenție! Pentru acest test se va afișa doar rezultatul la cerința b).</p>

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

Memorie: total **64 MB** din care pentru stivă **16 MB**

Dimensiune maximă a sursei: **5 KB**

6.2.1 Indicații de rezolvare

Prof. Adrian Panaete - Colegiul Național "A. T. Laurian" Botoșani

Pentru prima cerință o soluție corectă este

$$1 = \frac{1}{2^1} + \frac{1}{2^2} + \dots + \frac{1}{2^{N-3}} + \frac{1}{2^{N-2}} + \frac{1}{2^{N-1}} + \frac{1}{2^{N-1}}$$

Deci se pot afișa pe prima linie numerele 1, 2, 3, ..., $N-3$, $N-2$, $N-1$, $N-1$ în orice ordine și separate prin spațiu. Evident că aceasta nu este singura soluție corectă.

Orice scriere va fi de forma

$$1 = \frac{a_1}{2^1} + \frac{a_2}{2^2} + \dots + \frac{a_{k-1}}{2^{k-1}} + \frac{a_k}{2^k} \quad \text{cu } a_k \neq 0.$$

Prin înmulțire cu 2^k se deduce imediat că a_k este număr par deci există p natural nenul cu $a_k = 2p$ deci avem

$$\frac{a_k}{2^k} = \frac{2p}{2^k} = \frac{p}{2^{k-1}}.$$

și avem în cele din urmă o altă descompunere sub forma

$$1 = \frac{a_1}{2^1} + \frac{a_2}{2^2} + \dots + \frac{a_{k-1} + p}{2^{k-1}}.$$

Deci se poate spune că descompunerea va fi obținută alegând o altă descompunere în care fracția de numitor maxim este $\frac{1}{2^{k-1}}$ de cel puțin p ori prin înlocuirea a p astfel de fracții cu $2p$ fracții $\frac{1}{2^k}$ și se observă că descompunerea astfel obținută conține cu exact p fracții mai mult.

Observăm că astfel avem o metodă standard prin care se poate forma scrierea dorită.

Initial plecăm de la scrierea unică $1 = \frac{1}{2} + \frac{1}{2} = \frac{2}{2^1}$ apoi pentru fiecare i , $1 \leq i < k$ păstrăm exact a_i fracții $\frac{1}{2^i}$ iar pe fiecare dintre cele rămase le transformăm în câte două fracții $\frac{1}{2^{i+1}}$.

Numărarea descompunerilor se poate face prin *metoda programării dinamice* plecând de la observația de mai sus.

Notăm $D[m][i]$ numărul de descompuneri formate din m fracții dintre care $2i$ au *numitor maxim*.

Pentru oricare astfel de descompunere putem alege acum orice j , $1 \leq j \leq 2i$ și scriem j dintre fracțiile de *numitor maxim* ca sumă de *două fracții* cu numitorul dublat. Se va obține o nouă descompunere formată din $m + j$ fracții dintre care $2j$ au *numitor maxim*.

Aplicăm *dinamică înainte* adunând $D[m][i]$ la $D[m+j][j]$ pentru orice j cu $1 \leq j \leq 2i$ dacă $m + j < N$

Soluția finală se va obține prin adunarea valorii $D[m][i]$ de fiecare dată când obținem $m + j = N$.

6.2.2 Cod sursă

Listing 6.2.1: fractii2.cpp

```

1 #include <iostream>
2
3 #define MOD 100003
4
5 using namespace std;
6
7 int n, i, j, m, cod, left, right, from, sol[2010][1010];
8
9 int main()
10 {
11     freopen("fractii2.in", "r", stdin);
12     freopen("fractii2.out", "w", stdout);
13
14     scanf("%d%d", &cod, &n);
15     if(cod==1)
16     {
17         for(i=1; i<n; i++) printf("%d ", i);
18         printf("%d\n", n-1);
19         return 0;
20     }
21
22     sol[2][1]=1;
23     for(i=3; i<=n; i++)
24     {
25         m=i/2;
26         for(j=1; j<=m; j++)
27         {
28             from=i-j;
29             left=(j+1)/2-1;
30             right=(i-j)/2;
31             sol[i][j]=sol[i][j-1]+sol[i-j][right]-sol[i-j][left];
32             if(sol[i][j]<0) sol[i][j]+=MOD;
33             if(sol[i][j]>=MOD) sol[i][j]-=MOD;
34         }
35         //printf("Total %d = %d\n", i, sol[i][m]);
36     }
37     printf("%d", sol[n][n/2]);
38     return 0;
39 }
```

Listing 6.2.2: fractii2_back.cpp

```

1 #include <cstdio>
2 #include <vector>
3 #include <utility>
4
5 #define MOD 100003
6
7 using namespace std;
8
9 int n,cod,Sol,Contor[10000];
10 void back(int,int);
11
12 int main()
13 {
14     freopen("fractii2.in","r",stdin);
15     freopen("fractii2.out","w",stdout);
16
17     scanf("%d%d",&cod,&n);
18     if(cod==1)
19     {
20         printf("%d ",n-1);
21         for(int i=n-1;i;i--)
22             printf("%d ",i);
23         return 0;
24     }
25
26     Contor[1]=2;
27     back(2,1);
28     printf("%d",Sol);
29     return 0;
30 }
31
32 void back(int Fraction_Number,int Last_Exponent)
33 {
34     if(Fraction_Number>=n)
35     {
36         if(Fraction_Number>n) return;
37         //for(int i=1;i<=Last_Exponent;i++)
38         //    for(int j=Contor[i];j;j--)
39         //        printf("%4d",i);
40         //printf("\n");
41         Sol=Sol+1==MOD?0:Sol+1;
42         return;
43     }
44
45     int Next_Exponent=Last_Exponent+1;
46     int Steps=Contor[Last_Exponent];
47
48     for(int i=1;i<=Steps;i++)
49     {
50         Contor[Last_Exponent]=Steps-i;
51         Contor[Next_Exponent]=2*i;
52         back(Fraction_Number+i,Next_Exponent);
53     }
54 }
```

Listing 6.2.3: fractii2_var2.cpp

```

1 #include <cstdio>
2 #include <algorithm>
3
4 #define MOD 100003
5 #define N 2002
6
7 using namespace std;
8
9 int n,i,j,k,m,cod,U[N],D[N][N];
10
11 int main()
12 {
13     freopen("fractii2.in","r",stdin);
14     freopen("fractii2.out","w",stdout);
15
16     scanf("%d%d",&cod,&n);
17     if(cod==1)
18     {
```

```
19         for(i=1;i<n;i++)printf("%d ",i);printf("%d\n",n-1);
20     return 0;
21 }
22
23 n-=2;
24 U[n+1]=2;
25 for(i=n+1;i>=2;i--)
26     for(m=U[i],j=1;j<=m;j++)
27         U[i-j]=max(U[i-j],min(2*j,i-j));
28
29 D[0][0]=D[1][1]=1;
30 for(i=2;i<=n;i++)
31 {
32     m=U[i];
33     for(j=1;j<=m;j++)
34     {
35         k=min(2*j,i-j);
36         D[i][j]=D[i][j-1]+D[i-j][k]>=MOD ?
37             D[i][j-1]+D[i-j][k]-MOD :
38             D[i][j-1]+D[i-j][k];
39     }
40 }
41
42 printf("%d",D[n][2]);
43 return 0;
44 }
```

6.2.3 *Rezolvare detaliată

Capitolul 7

OJI 2013

7.1 biperm

Problema 1 - biperm

100 de puncte

Pentru un număr natural nenul n , să considerăm toate numerele naturale nenule mai mici sau egale cu n , luând fiecare număr de câte două ori: 1, 1, 2, 2, 3, 3, ..., n , n . Aceste numere le amestecăm aleator, și le aranjăm pe două linii a către n elemente. Structura astfel obținută o vom numi o *bipermutare*. În figurile 1, 2 și 3 avem câte un exemplu de bipermutare pentru $n = 5$.

O bipermutare este *perfectă*, dacă ambele linii ale structurii reprezintă câte o permutare (vezi figurile 2 și 3).

Prin *mutare pe poziția p* , înțelegem interschimbarea elementelor de pe aceeași coloană p . În exemplele de mai jos, bipermutarea perfectă din figura 2 s-a obținut din bipermutarea din figura 1, aplicând o *mutare pe poziția 2*. Bipermutarea perfectă din figura 3 s-a obținut din bipermutarea din figura 1, aplicând mutări pe pozițiile 1, 2, 4 și 5.

1	5	5	3	4
3	2	2	4	1

Figura 1

1	2	5	3	4
3	5	2	4	1

Figura 2

3	2	5	4	1
1	5	2	3	4

Figura 3

Figura 7.1: biperm

Cerințe

Cunoscând o bipermutare, determinați:

- numărul bipermutărilor perfecte distințe ce se pot obține prin mutări;
- numărul minim de mutări prin care se poate obține o bipermutare perfectă;
- o bipermutare perfectă obținută din bipermutarea inițială.

Date de intrare

Fișierul de intrare **biperm.in** conține pe prima linie valoarea lui n . Următoarele două linii conțin, fiecare, câte n elemente separate prin câte un spațiu, formând o bipermutare.

Date de ieșire

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

- pe prima linie două numere naturale separate printr-un spațiu, reprezentând numărul bipermutărilor perfecte distințe ce se pot obține din bipermutarea dată, respectiv numărul minim de mutări prin care se poate obține o bipermutare perfectă;
- pe următoarele două linii se vor tipări câte n numere separate prin spațiu, reprezentând o bipermutare perfectă obținută din bipermutarea dată.

Restricții și precizări

- $2 < n \leq 10000$;
- calculul corect al numărului bipermutărilor perfecte distințe valorează 30% din punctaj;
- calculul corect al numărului minim de mutări valorează 10% din punctaj;

- tipărirea unei bipermutări perfecte valorează 60% din punctaj. Pot exista mai multe soluții, se va admite orice soluție corectă;
- se garantează că numărul bipermutărilor perfecte distincte nu depășește 2 000 000 000 pentru niciun test.
- acordarea punctajului la un răspuns corect este condiționată de existența răspunsurilor anterioare, indiferent de corectitudinea lor;
- pentru 40% din teste $n \leq 20$
- pentru 40% din teste $20 < n \leq 400$
- pentru 20% din teste $400 < n \leq 10000$

Exemple

biperm.in	biperm.out	Explicații
5 1 5 5 3 4 3 2 2 4 1	4 1 1 2 5 3 4 3 5 2 4 1	Sunt 4 permutări perfecte. Numărul minim de mutări este 1 și există două soluții cu număr minim de mutări: 1 2 5 3 4 1 5 2 3 4 3 5 2 4 1 și 3 2 5 4 1 Celelalte două soluții, ce nu se obțin din număr minim de mutări sunt: 3 2 5 4 1 3 5 2 4 1 1 5 2 3 4 și 1 2 5 3 4 Pentru a treia cerință oricare dintre cele 4 soluții este acceptată.

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

Memorie: total **128 MB** din care pentru stivă **64 MB**

Dimensiune maximă a sursei: **10 KB**

7.1.1 Indicații de rezolvare

Să construim grafurile orientate corespunzătoare bipermutărilor din exemple:

Putem observa, că în graful orientat asociat unei permutări duble, în fiecare ciclu, toate arcele își păstrează sensul. În același timp într-o bipermutare ce nu e permutare dublă, există cel puțin un *ciclu* care nu e *circuit* (adică arcele nu își păstrează sensul).

Deci trebuie să reorientăm arcele ciclurilor, astfel încât să devină circuite. Pentru a efectua un număr minim de operații (interschimbări) vom număra câte arce sunt într-o direcție, și câte în direcția opusă, vom alege varianta ce afectează mai puține arce.

Numărul total de permutări duble este egal cu $2^{n_{\text{arce}}}$.

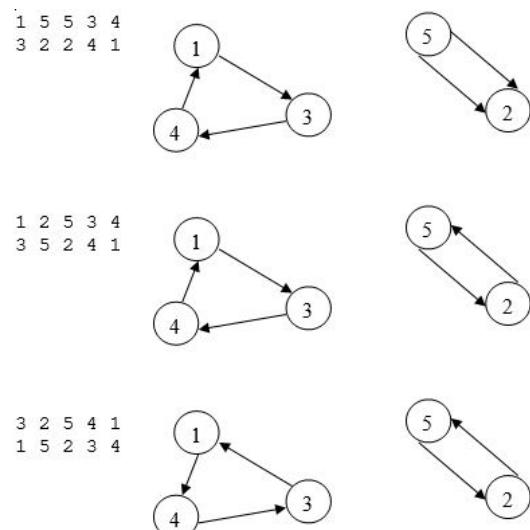


Figura 7.2: biperm

7.1.2 Cod sursă

Listing 7.1.1: biperm.cpp

```

1 // prof. Szabo Zoltan - Mures
2 // 100 puncte
3 #include <iostream>
4 #include <iostream>
5
6 using namespace std;
7
8 int viz[2][10003], a[2][10003], b[2][10003];
9
10 int main()
11 {
12     int n, i, j, p, nrmin, invers, l, poz, ant, aux, urmviz;
13
14     ifstream fin("biperm.in");
15     ofstream fout("biperm.out");

```

```

16
17     fin>>n;
18     for (i=0;i<2;++i)           // matricea viz[x][i] are semnificatia:
19         for (j=1;j<=n;++j)
20         {
21             fin>>a[i][j];      // numarul i apare in bipermutare pe pozitiile
22             if (viz[0][a[i][j]]==0)    //          viz[0][i] si viz[1][i]
23                 viz[0][a[i][j]]=j;
24             else
25                 viz[1][a[i][j]]=j;
26         }
27
28 p=1;           // p - numarul solutiilor distincte, produs, putere a lui 2
29 nrmin=0; // nrmin - va numara interschimbarile efectuate pentru cazul optim
30
31     for (i=1;i<=n;++i)
32         if (b[0][i]==0)
33         {
34             l=1; invers=0;poz=i; // reconstruim ciclurile: l lungimea ciclului
35                         // invers= cate arce sunt in ciclu cu sensul invers
36             b[0][poz]=a[0][poz]; // poz = pozitia curenta in ciclu
37             b[1][poz]=a[1][poz]; // ant = pozitia anterioara in ciclu
38             ant=poz;           // in b se construieste permutarea dubla ceruta
39             urmviz=b[1][poz];   // urmatorul nod spre vizitare in lant
40
41             if (viz[0][urmviz]==poz)
42                 poz=viz[1][urmviz];
43             else
44                 poz=viz[0][urmviz]; // lant continua este muchia a[0][poz] si
45                                         // a[1][poz]
46             while (!b[0][poz])
47             {
48                 if (a[0][poz]==b[1][ant])
49                 {
50                     b[0][poz]=a[0][poz];
51                     b[1][poz]=a[1][poz];
52                     l++;
53                     ant=poz;
54                     urmviz=b[1][poz];
55                     if (viz[0][urmviz]==poz)
56                         poz=viz[1][urmviz];
57                     else
58                         poz=viz[0][urmviz];
59                 }
60             else
61                 {
62                     b[0][poz]=a[1][poz];
63                     b[1][poz]=a[0][poz];
64                     l++;invers++; // am gasit un nou arc invers
65                     ant=poz;        //
66                     urmviz=b[1][poz];
67                     if (viz[0][urmviz]==poz)
68                         poz=viz[1][urmviz];
69                     else
70                         poz=viz[0][urmviz];
71                 }
72             }
73
74             if (l>1)
75                 p*=2;
76
77             if (invers>l-invers)
78                 nrmin+=l-invers;
79             else
80                 nrmin+=invers;
81         }
82
83     fout<<p<<" "<<nrmin<<"\n";
84     for(i=0;i<2;++i)
85     {
86         for(j=1;j<n;++j)
87             fout<<b[i][j]<<" ";
88         fout<<b[i][n]<<"\n";
89     }
90
91     fin.close();

```

```

92     fout.close();
93     return 0;
94 }
```

Listing 7.1.2: biperm_exp.cpp

```

1 //profesor Szabo Zoltan - ISJ Mures
2 #include <fstream>
3
4 using namespace std;
5
6 int a[2][60],b[2][60];
7
8 int verif(int a[2][60], int n)
9 {
10     int perm[60],j;
11     for (j=1;j<=n;++j)
12         perm[j]=0;
13
14     for (j=1;j<=n;++j)
15         if(perm[a[0][j]]==0)
16             perm[a[0][j]]=1;
17         else
18             return 0;
19
20     for (j=1;j<=n;++j)
21         perm[j]=0;
22
23     for (j=1;j<=n;++j)
24         if(perm[a[1][j]]==0)
25             perm[a[1][j]]=1;
26         else
27             return 0;
28
29     return 1;
30 }
31
32 int main()
33 {
34     int n,i,j,k,sup,nr,p,aux, nrsol=0,min,mutari,egal;
35
36     ifstream fin ("biperm.in");
37     ofstream fout ("biperm.out");
38     fin>>n;
39
40     if (n<30)
41     {
42         for (i=0;i<2; ++i)
43             for (j=1;j<=n; ++j)
44                 fin>>a[i][j];
45
46         sup=1<<n;
47         min=sup;
48         for (i=0;i<sup; ++i)
49         {
50             mutari=0;
51             nr=i;
52             k=1;
53             egal=0;
54             while(nr>0)
55             {
56                 if (a[0][k]==a[1][k] && nr%2)
57                     egal=1;
58                 if (nr%2)
59                 {
60                     aux=a[0][k];
61                     a[0][k]=a[1][k];
62                     a[1][k]=aux;
63                     ++mutari;
64                 }
65             }
66
67             if (!egal and verif(a,n))
68             {
69                 ++nrsol;
```

```

70             if (mutari<min)
71             {
72                 min=mutari;
73                 for (p=0;p<2; ++p)
74                     for (j=1;j<=n;++j)
75                         b[p][j]=a[p][j];
76             }
77         }
78
79         nr=i;
80         k=1;
81         while(nr>0)
82         {
83             if (nr%2)
84             {
85                 aux=a[0][k];
86                 a[0][k]=a[1][k];
87                 a[1][k]=aux;
88             }
89             k++;
90             nr/=2;
91         }
92     }
93 }
94
95 fout<<nrsol<<" "<<min<<"\n";
96 for (i=0;i<2;++i)
97 {
98     for (j=1;j<n;++j)
99         fout<<b[i][j]<<" ";
fout<<b[i][n]<<"\n";
100 }
101
102 fin.close();
103 fout.close();
104 return 0;
105 }
```

Listing 7.1.3: biperm1.cpp

```

1 //profesor Szabo Zoltan - ISJ Mures
2 #include <fstream>
3
4 using namespace std;
5
6 int a[2][60],b[2][60];
7
8 int verif(int a[2][60], int n)
9 {
10     int perm[60],j;
11     for (j=1;j<=n;++j)
12         perm[j]=0;
13
14     for (j=1;j<=n;++j)
15         if(perm[a[0][j]]==0)
16             perm[a[0][j]]=1;
17         else
18             return 0;
19
20     for (j=1;j<=n;++j)
21         perm[j]=0;
22
23     for (j=1;j<=n;++j)
24         if(perm[a[1][j]]==0)
25             perm[a[1][j]]=1;
26         else
27             return 0;
28
29     return 1;
30 }
31
32 int main()
33 {
34     int n,i,j,k,sup,nr,p,aux, nrsol=0,min,mutari,egal;
35
36     ifstream fin ("biperm.in");

```

```

37     ofstream fout ("biperm.out");
38     fin>>n;
39
40     if (n<30)
41     {
42         for (i=0;i<2; ++i)
43             for (j=1; j<=n; ++j)
44                 fin>>a[i][j];
45         sup=1<
46         min=sup;
47         for (i=0;i<sup;++i)
48         {
49             mutari=0;
50             nr=i;
51             k=1;
52             egal=0;
53             while(nr>0)
54             {
55                 if (a[0][k]==a[1][k] && nr%2)
56                     egal=1;
57                 if (nr%2)
58                 {
59                     aux=a[0][k];
60                     a[0][k]=a[1][k];
61                     a[1][k]=aux;
62                     ++mutari;
63                 }
64                 k++;
65                 nr/=2;
66             }
67             if (!egal and verif(a,n))
68             {
69                 ++nrsol;
70                 if (mutari<min)
71                 {
72                     min=mutari;
73                     for (p=0;p<2; ++p)
74                         for (j=1; j<=n; ++j)
75                             b[p][j]=a[p][j];
76                 }
77             }
78             nr=i;
79             k=1;
80             while(nr>0)
81             {
82                 if (nr%2)
83                 {
84                     aux=a[0][k];
85                     a[0][k]=a[1][k];
86                     a[1][k]=aux;
87                 }
88                 k++;
89                 nr/=2;
90             }
91         }
92     }
93 }
94
95 fout<<nrsol<<" "<<min<<"\n";
96 for (i=0;i<2;++i)
97 {
98     for (j=1; j<n; ++j)
99         fout<<b[i][j]<<" ";
100     fout<<b[i][n]<<"\n";
101 }
102 fin.close();
103 fout.close();
104 return 0;
105 }
```

7.1.3 *Rezolvare detaliată

7.2 subsecvențe

Problema 2 - subsecvențe

100 de puncte

Fie n un număr natural și $M = \{S_1, S_2, \dots, S_n\}$ o mulțime de siruri de caractere nevide. Fie S_k un sir de caractere din M . Atunci, orice caracter al lui S_k aparține mulțimii $\{a', b'\}$. Notăm prin $|S_k|$ numărul caracterelor sirului S_k sau, echivalent, lungimea sa. O subsecvență $S_k[i : j]$ a lui S_k este formată din caracterele situate pe pozițiile consecutive $i, i+1, \dots, j$. Astfel, dacă $S_k = 'abbaababa'$, atunci $S_k[3 : 6] = 'bbaa'$ sau subsecvența evidențiată: $'abbaaaab'$.

Cerințe

Fiind dată o mulțime M , se cere să se determine lungimea maximă a unei subsecvențe care se găsește în toate sirurile din M .

Date de intrare

Pe prima linie a fișierului de intrare **subsecvențe.in** se găsește un număr natural n egal cu cardinalul mulțimii M . Pe fiecare din următoarele n linii se găsește câte un sir din mulțimea M .

Date de ieșire

Pe prima linie a fișierului de ieșire **subsecvențe.out** se va scrie un singur număr natural egal cu lungimea subsecvenței găsite.

Restricții și precizări

- $1 < n < 5$
- Dacă $|S| = |S_1| + |S_2| + \dots + |S_n|$, atunci $|S| < 50001$
- Se garantează că va exista întotdeauna soluție
- Se garantează că rezultatul nu va depăși 60
- Pentru 30% din teste: $|S| < 101$
- Pentru 55% din teste: $|S| < 3501$
- Pentru 80% din teste: $|S| < 10001$

Exemple

subsecvențe.in	subsecvențe.out	Explicații
4 abbabaaaabb aaaababab bbbbaaaab aaaaaaabaaab	5	Lungimea unei subsecvențe comune de lungime maximă este 5. În exemplu subsecvența comună de lungime 5 este <i>aaaab</i> : <i>abbabaaaabb</i> , <i>aaaababab</i> , <i>bbbbaaaab</i> , <i>aaaaaaaaabaaab</i> .

Timp maxim de executare/test: **0.6** secunde

Memorie: total **128 MB** din care pentru stivă **64 MB**

Dimensiune maximă a sursei: **20 KB**

7.2.1 Indicații de rezolvare

Student Marius Laurențiu Stroe - Universitatea București

Notăm prin $|S| = |S_1| + \dots + |S_{|M|}|$, prin S' sirul de lungime minimă din M iar prin L lungimea maximă posibilă a unei subsecvențe.

Soluție 30p

Soluția naivă constă în a considera toate subsecvențele celui mai scurt sir S' , în $O(L * |S'|)$, și a verifica dacă există în celelalte, în $O(|S| * L)$. Complexitatea finală: $O(L^2 * |S| * |S'|)$. Cazul defavorabil se obține când există două siruri de lungimi aproximativ egale, caz în care complexitatea finală este $O(L^2 * |S|^2)$. Însă, în practică, dacă această abordare este optimizată efectuând cât mai puține comparații atunci se pot obține punctaje mai mari.

Căutarea unui subșir în celelalte siruri poate profita de restricția că lungimea maximă a unei subsecvențe nu poate depăși 60. Înlocuind caracterele a și b cu 0 și 1, vom obține siruri ale căror

subsecvențe pot fi reprezentate ca numere întregi pe cel mult 64 de biți. Astfel, putem considera toate lungimile unei subsecvențe, în complexitate $O(L)$, consideră apoi toate subsecvențele de această lungime din cel mai scurt sir, în complexitate $O(|S'|)$, și căuta în restul sirurilor această subsecvență, în complexitate $O(|S| + L)$. O subsecvență, reprezentată de un număr întreg, se poate căuta într-un sir considerând toate subsecvențele acestuia consecutiv, deplasând biții săi la stânga și înălțându-i pe cei mai semnificativi, operații ce se pot efectua în $O(1)$. Astfel, complexitatea totală este $O(L * |S| * |S'|)$, care în cazul devarobabil, menționat anterior, devine $O(L * |S|^2)$. În practică, căutarea curentă se comportă puțin mai slab decât căutarea anterioară datorită operațiilor pe numere de 64 biți.

Soluție 40p

O altă restricție a problemei ce ne poate ajuta este cardinalul mulțimii M ce poate fi maxim 4. În cazul în care $|M| = 2$, problema se reduce la a determina *cea mai lungă subsecvență comună a două siruri*. O soluție a acestei probleme se poate obține în complexitate $O(|S|^2)$ cu *metoda programării dinamice*. Recurența poate fi următoarea:

```
d[i][j] = {
    d[i - 1][j - 1] + 1, dacă a S1[i] = S2[j]
    0, altfel
}
```

Soluția este $\max(d[i][j])$, iterând după toți indicii i, j . Analog, se mai poate adăuga o dimensiune dacă avem trei siruri. În cazul general, complexitatea este $O(|S|^{1+M})$. Implementarea recurenței cu număr variabil de siruri vă va aduce acest punctaj.

Soluție 55p

Observăm că dacă o subsecvență S' se găsește în toate sirurile, atunci orice subsecvență a sa, se găsește de asemenea în toate sirurile. În particular, orice *prefix* al lui S' se găsește în toate sirurile. Se poate astfel căuta binar lungimea unei subsecvențe. Iar dacă extindem prima soluție înlocuind căutarea liniară, obținem o soluție cu complexitatea finală în cazul defavorabil: $O(\log(L) * |S|^2)$.

Soluție 55p, Adrian Panaete

Putem folosi o *metodă constructivă* începând cu subsecvența vidă corespunzătoare lungimii 0 și extinzând orice subsecvență comună în cele două moduri posibile (adăugând la final caracterul '*a*' sau caracterul '*b*'). Dacă prin extindere se obține subsecvență comună, atunci se realizează o îmbunătățire a soluției curente și se păstrează subsecvența extinsă. Algoritmul se încheie dacă pentru subsecvențele de o anumită lungime L nu se mai poate aplica nicio extindere și L va fi exact soluția problemei. Deoarece la fiecare pas corespunzător subsecvențelor de lungime $0, 1, \dots, L - 1$ vom avea de procesat numai subsecvențe comune, atunci numărul acestora nu va putea depăși $|S'|$. În realitate acest număr va fi mai mic deoarece subsecvențele identice ale celui mai scurt string vor fi luate o singură dată. Complexitatea supraestimată a acestui algoritm este $O(L * |S| * |S'|)$.

Soluție 80p

Tinând cont că pentru 80% din teste o complexitate de memorie $O(L * |S|)$ este acceptabilă, putem menține într-o *structură de date* toate subsecvențele lui S' de lungime cel mult L . Subsecvențele trebuie tinute pe numere întregi de 64 de biți pentru a nu consuma memorie în exces și a compara două subsecvențe în timp constant. Astfel, iterând celelalte siruri, vom calcula din nou toate subsecvențele, dar vom trece mai departe doar cele care se găsesc în sirul anterior. Astfel, setul de subsecvențe descrește semnificativ. Condiția pentru a obține acest punctaj este ca structura de date ce menține subsecvențele să permită operații de *inserare* și *ștergere* în *temp logaritmic*. Atunci complexitatea finală este $O(L * |S| \log(L * |S|))$.

Soluție 100p, Adrian Panaete

Se concatenează cele n stringuri obținând astfel un string de lungime $|S|$. Pentru fiecare poziție din sirul concatenat memorăm un cod binar de cel mult 4 biți care ne indică din ce string inițial face parte poziția respectivă (pentru primul sir codul 1 = 0001 pentru al doilea codul 2 = 0010 pentru al treilea codul 4 = 0100 și pentru al patrulea codul 8 = 1000).

Vom începe să sortăm pozițiile utilizând drept criteriu *ordinea lexicografică* a subsecvențelor ce încep din poziția respectivă. Pentru sortare vom folosi ideea algoritmului *radix-sort* grupând pozițiile în *buchete* care conțin toate pozițiile cu o aceeași subsecvență de o anumită lungime

și descompunând fiecare buchet în cate două buchete corespunzătoare prelungirii subsecvenței respective fie cu caracterul '*a*' fie cu caracterul '*b*'.

Odată cu introducerea unei poziții în unul dintre buchete adăugăm la un cod binar al buchetului și codul poziției din care provine subsecvența corespunzătoare. În acest fel putem urmări dacă un buchet mai merită să fie descompus (în cazul în care codul buchetului are la final toți cei *n* biți deci conține subsecvențe din toate cele *n* stringuri inițiale).

Tot codurile binare ale pozițiilor ne semnalizează dacă la un moment dat încercăm să prelungim o subsecvență cu un caracter care nu mai aparține stringului. În acel caz poziția respectivă nu va mai fi plasată în niciunul dintre cele două buchete. Cum fiecare buchet descrie de fapt o subsecvență de o anumită lungime odată cu definitivarea unui buchet dacă se constată că el conține poziții din toate stringurile se poate folosi lungimea respectivă pentru a îmbunătăți eventual soluția problemei.

Deoarece fiecare poziție poate fi utilizată de cel mult *L* ori (*L*=soluția testului) vom obține un algoritm de complexitate $O(L * |S|)$.

Soluție 100p, Adrian Panaete

Construim un *arbore binar*, în care fiul stâng este '*a*' iar fiul dreaptă este '*b*'. Considerăm toate subsecvențele de lungime *L* și le inserăm în acest arbore binar. Inserarea se face pornind din rădăcină și mergând spre stânga dacă primul caracter al subsecvenței este '*a*' sau în dreapta dacă acesta este '*b*'. Recursiv se procedează pentru restul nodurilor din arbore și restul subsecvenței. Complexitatea acestei operații este $O(L * |S|)$.

În acest arbore binar, fiecare nod memoră câte o subsecvență de lungime egală cu distanța de la nod la rădăcină. Pentru fiecare nod din arbore memorăm pe un număr de cel mult $|M|$ biți din ce string face parte subsecvența corespunzătoare. Luăm ulterior celealte siruri și parcurgem arborele cu toate subsecvențele sale de lungime cel mult 60. Însă, de data aceasta, avem două situații de *întrerupere* a parcurgerii:

1: parcurgerea se întrerupe deoarece un nod nu are continuare pe o literă; acest lucru înseamnă de fapt că subsecvența corespunzătoare nu e în primul sir, deci implicit, orice continuare a ei nu va fi în primul sir;

2: parcurgerea se întrerupe pentru că informația din nod îmi dovedește că un alt string anterior, altul decât primul sir, nu conține subsecvența corespunzătoare și, implicit, nici continuările ei.

Dacă am orice altceva însemnată că de fapt subsecvența nu a fost marcată într-un sir anterior. În cazul preferabil, când informația mă asigură că sunt pe o cale bună, actualizez informația din nod adăugând un bit corespunzător sirului curent.

Soluția se actualizează în timp real odată cu introducerea subsecvențelor nouului sir în arboarele binar în funcție de cât de adânc reușesc acestea să pătrundă fără ca eventual să apară o întrerupere de cele două tipuri. Soluția de lungime maximă a ultimului sir este răspunsul corect. Complexitatea finală este $O(L * |S|)$.

7.2.2 Cod sursă

Listing 7.2.1: subsecvențe_ans_build.cpp

```

1 #include <cstdio>
2 #include<iostream>
3 #include<algorithm>
4 #include<vector>
5
6 using namespace std;
7
8 char A[50010];
9 vector<char> S[5],aux;
10 vector<vector<char> > V[2];
11 int n,i,c,v,L;
12
13 int main()
14 {
15     freopen("subsecvențe.in","r",stdin);
16     freopen("subsecvențe.out","w",stdout);
17
18     int n;  cin >> n;

```

```

19     for (int i = 0; i < n; ++ i)
20     {
21         cin >> A;
22         for(int j=0;A[j];j++)
23             S[i].push_back(A[j]);
24     }
25
26     V[0].push_back(aux);
27     c=0;
28     v=1;
29     for(L=0;;)
30     {
31         int i;
32         vector<vector<char> >::iterator it;
33         V[v].resize(0);
34
35         for(it=V[c].begin();it!=V[c].end();it++)
36         {
37             aux=*it;
38             aux.push_back('a');
39
40             for(i=0;i<n;i++)
41                 if(search(S[i].begin(),
42                           S[i].end(),
43                           aux.begin(),aux.end())
44                           ==S[i].end())
45                     break;
46
47             if(i==n)
48                 V[v].push_back(aux);
49
50             aux=*it;
51             aux.push_back('b');
52             for(i=0;i<n;i++)
53                 if(search(S[i].begin(),
54                           S[i].end(),aux.begin(),
55                           aux.end())
56                           ==S[i].end())
57                     break;
58
59             if(i==n)
60                 V[v].push_back(aux);
61         }
62
63         if(!V[v].size())break;
64         L++;
65         c=1-c;
66         v=1-v;
67     }
68
69     cout<<L;
70     return 0;
71 }
```

Listing 7.2.2: subsecvente_arb.cpp

```

1 #include<cstdio>
2 #include<cstdlib>
3 #include<ctime>
4 #include<cstring>
5
6 using namespace std;
7
8 FILE *fin;
9
10 struct nod
11 {
12     int inf;
13     nod *F[2];
14     nod()
15     {
16         inf=0;
17         F[0]=F[1]=0;
18     }
19 };
```

```

20
21 nod *root,*q,*aux;
22 int pow,mask,lg,i,j,sol,start;
23 char a[50010],*p;
24
25 int main()
26 {
27     freopen("subsecvente.in","r",stdin);
28     freopen("subsecvente.out","w",stdout);
29
30     root=new nod;
31     pow=mask=1;
32     int n;
33
34     scanf("%d\n", &n);
35     scanf("%s",a+1);
36
37     lg=strlen(a+1);
38     for(i=1;i<=lg;i++)
39     {
40         for(j=0,p=a+i,q=root; j<60 && *p; j++,p++)
41         {
42             if(!q->F[*p-'a'])
43             {
44                 aux=new nod;
45                 aux->inf=1;
46                 q->F[*p-'a']=aux;
47             }
48             q=q->F[*p-'a'];
49         }
50     for(;scanf("%s",a+1)+1;)
51     {
52         pow<=<1;
53         mask|=pow;
54         sol=0;
55         lg=strlen(a+1);
56
57         for(i=1;i<=lg;i++)
58             for(j=0,p=a+i,q=root;j<60&&*p;j++,p++)
59             {
60                 if(!q->F[*p-'a']) break;
61                 q=q->F[*p-'a'];
62                 q->inf|=pow;
63                 if(q->inf!=mask) break;
64                 if(j+1>sol)
65                 {
66                     start=i;
67                     sol=j+1;
68                 }
69             }
70             if(!sol) break;
71     }
72
73     printf("%d\n",sol);
74 //a[start+sol]=0;printf("%s",a+start);
75     return 0;
76 }

```

Listing 7.2.3: subsecvente_arb_static_arrays.cpp

```

1 #include<cstdio>
2 #include<cstring>
3 #include <algorithm>
4
5 #define NODES 2600100
6
7 using namespace std;
8
9 int p2,lg,mask,i,j,k,p,q,n,sol(int),s[2][NODES];
10 char cod[NODES];
11 char a[50010];
12
13 int main()
14 {

```

```

15     freopen("subsecvente.in", "r", stdin);
16     freopen("subsecvente.out", "w", stdout);
17
18     int n;
19     scanf("%d\n", &n);
20     for(p2=1,n=1; scanf("%s",a+1)+1; p2<<=1)
21     {
22         lg=strlen(a+1);
23         mask|=p2;
24         for(i=1;i<=lg;i++)
25         {
26             j=min(i+59,lg);
27
28             for(k=i,p=1;k<=j;k++)
29             {
30                 q=a[k]-'a';
31                 if(!s[q][p])
32                     s[q][p]=++n;
33
34                 p=s[q][p];
35                 cod[p]|=p2;
36             }
37         }
38     }
39
40     cod[1]=mask;
41     printf("%d\n", sol(1)-1);
42     return 0;
43 }
44
45 int sol(int nod)
46 {
47     int sa,sb;
48     if(!nod||cod[nod]!=mask) return 0;
49     sa=sol(s[0][nod]);
50     sb=sol(s[1][nod]);
51     return max(sa,sb)+1;
52 }
```

Listing 7.2.4: subsecvente_arb_stl.cpp

```

1 #include<cstdio>
2 #include<cstring>
3 #include<vector>
4 #include <algorithm>
5
6 using namespace std;
7
8 int p2,lg,mask,i,j,k,p,q,n,sol(int);
9 vector<int>s[2],cod;
10 char a[50010];
11
12 int main()
13 {
14     freopen("subsecvente.in", "r", stdin);
15     freopen("subsecvente.out", "w", stdout);
16
17     int n;
18     scanf("%d\n", &n);
19     s[0].push_back(0);
20     s[1].push_back(0);
21     cod.push_back(0);
22     s[0].push_back(0);
23     s[1].push_back(0);
24     cod.push_back(0);
25     for(p2=1,n=1; scanf("%s",a+1)+1; p2<<=1)
26     {
27         lg=strlen(a+1);
28         mask|=p2;
29         for(i=1;i<=lg;i++)
30         {
31             j=min(i+59,lg);
32
33             for(k=i,p=1;k<=j;k++)
34             {
```

```

35             q=a[k]-'a';
36             if(!s[q][p])
37             {
38                 s[0].push_back(0);
39                 s[1].push_back(0);
40                 cod.push_back(0);
41                 s[q][p]=++n;
42             }
43
44             p=s[q][p];
45             cod[p]|=p2;
46         }
47     }
48 }
49
50 cod[1]=mask;
51 printf("%d\n",sol(1)-1);
52 return 0;
53 }
54
55 int sol(int nod)
56 {
57     int sa,sb;
58     if(!nod||cod[nod]!=mask) return 0;
59     sa=sol(s[0][nod]);
60     sb=sol(s[1][nod]);
61     return max(sa,sb)+1;
62 }
```

Listing 7.2.5: subsecvente_binary_search.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <algorithm>
5
6 using namespace std;
7
8 const char iname[] = "subsecvente.in";
9 const char oname[] = "subsecvente.out";
10
11 typedef unsigned long long i64;
12
13 string main_str;
14 vector <string> strs;
15
16 void read_in()
17 {
18     ifstream fin(iname);
19     int n;
20     fin >> n;
21
22     fin >> main_str;
23     for (string str; fin >> str; )
24         strs.push_back(str);
25
26     fin.close();
27 }
28
29 void write_out(int length)
30 {
31     ofstream out(oname);
32     out << length << "\n";
33     out.close();
34 }
35
36 i64 get_number(string& str, size_t i, size_t length)
37 {
38     i64 number = 0;
39
40     for (size_t step = 0; step < length; ++ step)
41         number = (number << 1) | int(str[i + step] - 'a');
42
43     return number;
44 }
```

```

45
46 int search_number(i64 number, size_t length)
47 {
48     i64 number_mask = (1LL << length) - 1; // Length should be less than 63.
49
50     for (size_t i = 0; i < strs.size(); ++ i)
51     {
52         if (strs[i].size() < length)
53             return 0;
54
55         i64 number_str = get_number(strs[i], 0, length);
56         bool numbers_match = (number_str == number);
57         for (size_t j = length; j < strs[i].size() && !numbers_match; ++ j)
58         {
59             number_str = (number_str << 1) | int(strs[i][j] - 'a');
60             number_str = number_str & number_mask;
61             numbers_match |= (number_str == number);
62         }
63
64         if (!numbers_match)
65             return 0;
66     }
67
68     return 1;
69 }
70
71 int main(void)
72 {
73     read_in();
74
75     size_t k = 0, delta = 64;
76     for (k = 0; delta; delta >>= 1)
77     {
78         if (k + delta >= 62) continue;
79
80         size_t length = k + delta;
81         i64 number_mask = (1LL << length) - 1;
82
83         if (main_str.size() >= length)
84         {
85             i64 number = get_number(main_str, 0, length);
86             if (search_number(number, length))
87                 k = length;
88
89             for (size_t i = length; i < main_str.size() && k < length; ++ i)
90             {
91                 number = (number << 1) | int(main_str[i] - 'a');
92                 number = number & number_mask;
93                 if (search_number(number, length))
94                     k = length;
95             }
96         }
97     }
98
99     write_out(k);
100
101    return 0;
102 }
```

Listing 7.2.6: subsecvente_binary_search_naive.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <string>
5 #include <algorithm>
6
7 using namespace std;
8
9 const char iname[] = "subsecvente.in";
10 const char oname[] = "subsecvente.out";
11
12 typedef unsigned long long i64;
13
14 void read_in(string &main_str, vector <string> &strs)
```

```

15  {
16      ifstream fin(iname);
17      int n; fin >> n;
18
19      fin >> main_str;
20      for (string str; fin >> str; )
21          strs.push_back(str);
22      fin.close();
23  }
24
25 void write_out(int length)
26 {
27     ofstream out(oname);
28     out << length << "\n";
29     out.close();
30 }
31
32 bool search_string(string& str, vector <string>& strs)
33 {
34     for (size_t i = 0; i < strs.size(); ++ i)
35         if (strs[i].find(str) == string::npos)
36             return false;
37     return true;
38 }
39
40 int main(void)
41 {
42     string main_str; vector <string> strs;
43     read_in(main_str, strs);
44
45     int delta = 64, k = 0;
46     for (; delta > 0; delta >>= 1)
47     {
48         int length = k + delta;
49         if (length > 61) continue;
50
51         for (int i = 0; i <= (int)main_str.size() - length; ++ i)
52         {
53             string number_str = main_str.substr(i, length);
54             if (search_string(number_str, strs))
55             {
56                 k = length;
57                 break;
58             }
59         }
60     }
61     write_out(k);
62
63     return 0;
64 }

```

Listing 7.2.7: subsecvente_dp.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <string>
5 #include <algorithm>
6 #include <cstdarg>
7 #include <cassert>
8
9 using namespace std;
10
11 typedef unsigned long long u64;
12
13 const char iname[] = "subsecvente.in";
14 const char oname[] = "subsecvente.out";
15
16 const int max_length = 3005;
17 const u64 hash_size = 666013;
18
19 vector <int> dp;
20 u64 dp_base;
21 int dp2[max_length] [max_length];

```

```

22
23 vector < string > strs;
24 vector < pair < u64, int > > hashh[hash_size];
25 int hash_obj_index;
26
27
28 void read_in(vector <string> &strs)
29 {
30     ifstream fin(iname);
31     int n;
32     fin >> n;
33     for (string str; fin >> str; )
34         strs.push_back(str);
35     fin.close();
36     assert(n == (int)strs.size());
37 }
38
39 void write_out(int length)
40 {
41     ofstream out(ename);
42     out << length << "\n";
43     out.close();
44 }
45
46 inline u64 find_value(u64 value)
47 {
48     u64 list = value % hash_size;
49     vector < pair < u64, int > ::iterator it;
50
51     for (it = hashh[list].begin(); it != hashh[list].end(); ++ it)
52         if (it->first == value)
53             return it->second;
54     return 0;
55 }
56
57 inline int hash_array(vector <int>& array)
58 {
59     u64 value = 0;
60
61     for (size_t i = 0; i < array.size(); ++ i)
62         value = value * dp_base + array[i];
63
64     int index = find_value(value);
65     if (!index)
66     {
67         index = (++ hash_obj_index);
68         hashh[value % hash_size].push_back(make_pair(value, index));
69     }
70     return index;
71 }
72
73 int call_dp(vector <int> array);
74
75 int dp_func(int num, ...)
76 {
77     // Parse arguments.
78     va_list args_list;
79     va_start(args_list, num);
80
81     vector <int> args(num);
82     for (size_t i = 0; i < args.size(); ++ i)
83         args[i] = va_arg(args_list, int);
84
85     va_end(args_list);
86
87     // Get current state from args list.
88     int curr_state = hash_array(args);
89
90     if (dp[curr_state] == -1)
91     {
92         // Call all other states.
93         for (size_t i = 0; i < args.size(); ++ i) if (args[i] > 0)
94         {
95             args[i] -= 1;
96             call_dp(args);
97             args[i] += 1;

```

```

98         }
99
100        // Apply dynamic programming.
101        bool eq = true;
102        bool gtz = args[0] > 0;
103        for (size_t i = 1; i < args.size(); ++ i)
104        {
105            eq = eq & strs[i][ args[i] ] == strs[i - 1][ args[i - 1] ];
106            gtz = gtz & (args[i] > 0);
107        }
108
109        if (gtz && eq)
110        {
111            for (size_t i = 0; i < args.size(); ++ i)
112                args[i] -= 1;
113
114            dp[curr_state] = max(dp[curr_state], call_dp(args) + 1);
115        }
116
117        if (dp[curr_state] == -1)
118            dp[curr_state] = eq ? 1 : 0;
119    }
120
121    return dp[curr_state];
122}
123
124 int call_dp(vector <int> array)
125{
126    if (array.size() == 2)
127        return dp_func(2, array[0], array[1]);
128    if (array.size() == 3)
129        return dp_func(3, array[0], array[1], array[2]);
130    if (array.size() == 4)
131        return dp_func(4, array[0], array[1], array[2], array[3]);
132    if (array.size() == 5)
133        return dp_func(4, array[0], array[1], array[2], array[3], array[4]);
134}
135
136 int call_2dim_dp(void)
137{
138    int ans = 0;
139
140    dp2[0][0] = 1;
141    for (size_t i = 0; i < strs[0].size(); ++ i)
142        for (size_t j = 0; j < strs[1].size(); ++ j)
143            if (strs[0][i] == strs[1][j])
144            {
145                dp2[i + 1][j + 1] = max(dp2[i + 1][j + 1], dp2[i][j] + 1);
146                ans = max(ans, dp2[i + 1][j + 1]);
147            }
148            else
149                dp2[i + 1][j + 1] = 0;
150
151    return ans;
152}
153
154 int main(void)
155{
156    read_in(strs);
157
158    if (strs.size() > 2)
159    {
160        // Build dp matrix.
161        vector <int> dp_indexes; u64 dp_size = 1;
162        for (size_t i = 0; i < strs.size(); ++ i)
163        {
164            dp_size *= strs[i].size();
165            dp_base = max(dp_base, (u64)strs[i].size() + 1);
166            dp_indexes.push_back((int)strs[i].size() - 1);
167        }
168
169        dp.resize(dp_size + 5);
170        dp.assign(dp.size(), -1);
171
172        // Apply dynamic programming.
173        call_dp(dp_indexes);

```

```

174         // Get answer.
175         int ans = 0;
176         for (int i = 1; i <= hash_obj_index; ++ i)
177             ans = max(ans, dp[i]);
178
179         write_out(ans);
180     }
181     else
182         write_out(call_2dim_dp());
183
184     return 0;
185 }

```

Listing 7.2.8: subsecvete_linear_search.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <algorithm>
5
6 using namespace std;
7
8 const char iname[] = "subsecvete.in";
9 const char oname[] = "subsecvete.out";
10
11 typedef long long i64;
12
13 void read_in(string &main_str, vector <string> &strs)
14 {
15     ifstream fin(iname);
16     int n;
17     fin >> n;
18     fin >> main_str;
19     for (string str; fin >> str; )
20         strs.push_back(str);
21     fin.close();
22 }
23
24 void write_out(int length)
25 {
26     ofstream out(oname);
27     out << length << "\n";
28     out.close();
29 }
30
31 i64 get_number(string& str, size_t i, size_t length)
32 {
33     i64 number = 0;
34
35     for (size_t step = 0; step < length; ++ step)
36         number = (number << 1) | int(str[i + step] - 'a');
37     return number;
38 }
39
40 int search_number(i64 number, size_t length, vector <string>& strs)
41 {
42     i64 number_mask = (1LL << length) - 1; // Length should be less than 63.
43
44     for (size_t i = 0; i < strs.size(); ++ i)
45     {
46         if (strs[i].size() < length)
47             return 0;
48
49         i64 number_str = get_number(strs[i], 0, length);
50         bool numbers_match = (number_str == number);
51         for (size_t j = length; j < strs[i].size() && !numbers_match; ++ j)
52         {
53             number_str = (number_str << 1) | int(strs[i][j] - 'a');
54             number_str = number_str & number_mask;
55             numbers_match |= (number_str == number);
56         }
57
58         if (!numbers_match)
59             return 0;

```

```

60      }
61
62      return 1;
63  }
64
65 int main(void)
66 {
67     string main_str; vector <string> strs;
68     read_in(main_str, strs);
69
70     size_t length_ans = 0;
71     for (size_t length = 62; length >= 1 && !length_ans; -- length)
72     {
73         i64 number_mask = (1LL << length) - 1;
74
75         if (main_str.size() >= length)
76         {
77             i64 number = get_number(main_str, 0, length);
78             if (search_number(number, length, strs))
79                 length_ans = length;
80             for (size_t i = length; i < main_str.size() && !length_ans; ++ i)
81             {
82                 number = (number << 1) | int(main_str[i] - 'a');
83                 number = number & number_mask;
84                 if (search_number(number, length, strs))
85                     length_ans = length;
86             }
87         }
88     }
89
90     write_out(length_ans);
91
92     return 0;
93 }
```

Listing 7.2.9: subsecvinte_log.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <string>
5 #include <set>
6 #include <algorithm>
7 #include <cassert>
8
9 using namespace std;
10
11 const char iname[] = "subsecvinte.in";
12 const char oname[] = "subsecvinte.out";
13
14 typedef unsigned long long u64;
15
16 vector <string> strs;
17
18 set < pair <u64, int> > numbers[2];
19
20 void read_in(vector <string> &strs)
21 {
22     ifstream fin(iname);
23     int n; fin >> n;
24
25     for (string str; fin >> str; )
26         strs.push_back(str);
27     fin.close();
28     assert(n == (int)strs.size());
29 }
30
31 void write_out(int length)
32 {
33     ofstream out(oname);
34     out << length << "\n";
35     out.close();
36 }
37
38 u64 get_number(string& str, int index, int length)
```

```

39  {
40      u64 number = 0;
41
42      for (int i = index; i < index + length; ++ i)
43          number = (number << 1) | (str[i] - 'a');
44      return number & ((1LL << length) - 1);
45  }
46
47 void add_number(u64 number, int length, int index, bool init)
48 {
49     if (init ||
50         numbers[1 ^ index].find(make_pair(number, length)) !=
51         numbers[1 ^ index].end())
52         numbers[index].insert(make_pair(number, length));
53 }
54
55 int main(void)
56 {
57     read_in(strs);
58
59     int step_ans = 0;
60
61     for (int length = 61; length >= 1; -- length)
62     {
63         int step = 0;
64         u64 number_mask = (1LL << length) - 1;
65
66         for (int i = 0; i < strs.size(); ++ i)
67         {
68             step ^= 1;
69             numbers[step].clear();
70             if (length > strs[i].size())
71                 break;
72
73             u64 curr_number = get_number(strs[i], 0, length);
74             add_number(curr_number, length, step, !i);
75
76             for (int j = length; j < strs[i].size(); ++ j)
77             {
78                 curr_number = (curr_number << 1) | int(strs[i][j] - 'a');
79                 curr_number = curr_number & number_mask;
80                 add_number(curr_number, length, step, !i);
81             }
82
83             if (numbers[step].size() == 0)
84                 break;
85         }
86
87         if (numbers[step].size() > 0)
88         {
89             step_ans = step;
90             break;
91         }
92     }
93
94     set<pair<u64, int>>::iterator it;
95     int max_length = 0;
96
97     for (it = numbers[step_ans].begin(); it != numbers[step_ans].end(); ++ it)
98     {
99         max_length = max(max_length, it->second);
100    }
101
102    write_out(max_length);
103
104    return 0;
105}

```

Listing 7.2.10: subsecvinte_naive.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <string>
5 #include <algorithm>

```

```

6
7 using namespace std;
8
9 const char iname[] = "subsecvente.in";
10 const char oname[] = "subsecvente.out";
11
12 typedef unsigned long long i64;
13
14 void read_in(string &main_str, vector <string> &strs)
15 {
16     ifstream fin(iname);
17     int n; fin >> n;
18     fin >> main_str;
19     for (string str; fin >> str; )
20         strs.push_back(str);
21     fin.close();
22 }
23
24 void write_out(int length)
25 {
26     ofstream out(oname);
27     out << length << "\n";
28     out.close();
29 }
30
31 bool search_string(string& str, vector <string>& strs)
32 {
33     for (size_t i = 0; i < strs.size(); ++ i)
34         if (strs[i].find(str) == string::npos)
35             return false;
36     return true;
37 }
38
39 int main(void)
40 {
41     string main_str; vector <string> strs;
42     read_in(main_str, strs);
43
44     string number_ans;
45     for (size_t i = 0; i < main_str.size(); ++ i)
46         for (size_t j = i; j < main_str.size(); ++ j)
47         {
48             string number_str = main_str.substr(i, j-i+1);
49
50             if (search_string(number_str, strs))
51                 if (number_ans.size() < number_str.size())
52                     number_ans = number_str;
53         }
54
55     write_out(number_ans.size());
56     return 0;
57 }
```

Listing 7.2.11: subsecvente_naive_optimized.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <string>
5 #include <algorithm>
6
7 using namespace std;
8
9 const char iname[] = "subsecvente.in";
10 const char oname[] = "subsecvente.out";
11
12 typedef unsigned long long i64;
13
14 void read_in(string &main_str, vector <string> &strs)
15 {
16     ifstream fin(iname);
17     int n; fin >> n;
18     fin >> main_str;
19     for (string str; fin >> str; )
20         strs.push_back(str);
```

```

21     fin.close();
22 }
23
24 void write_out(int length)
25 {
26     ofstream out(pname);
27     out << length << "\n";
28     out.close();
29 }
30
31 bool search_string(string& str, vector <string>& strs)
32 {
33     for (size_t i = 0; i < strs.size(); ++ i)
34         if (strs[i].find(str) == string::npos)
35             return false;
36
37     return true;
38 }
39
40 int main(void)
41 {
42     string main_str; vector <string> strs;
43     read_in(main_str, strs);
44
45     string number_ans;
46     for (int length = 61; length >= 1 && !number_ans.size(); -- length)
47         for (int i=0; i<=(int)main_str.size()-length&&!number_ans.size();++i)
48         {
49             string number_str = main_str.substr(i, length);
50             if (search_string(number_str, strs))
51                 number_ans = number_str;
52         }
53
54     write_out(number_ans.size());
55     return 0;
56 }
```

Listing 7.2.12: subsecvente_radix.cpp

```

1 #include <cstdio>
2 #include<algorithm>
3
4 using namespace std;
5
6 char A[50100],B[50100];
7 int n,C[50100],V[2][50100],x,M,k,S,SOL;
8
9 void radix(int,int,int,int);
10
11 int main()
12 {
13     freopen("subsecvente.in","r",stdin);
14     freopen("subsecvente.out","w",stdout);
15
16     scanf("%d",&n);
17     for(int i=0;i<n;i++)
18     {
19         scanf("%s",B);
20         x=1<<i;M|=x;
21         for(k=0;B[k];k++)
22         {
23             A[++S]=B[k];
24             C[S]=x;
25             V[0][S]=S;
26         }
27     }
28
29     radix(1,S,0,0);
30     printf("%d",SOL);
31     return 0;
32 }
33
34 void radix(int ST,int DR,int LG,int p)
35 {
36     int MA=0,MB=0,DA=ST-1,SA=ST,DB=DR,SB=DR+1,q=1-p,w,u;
```

```

37     for (w=ST; w<=DR; w++)
38     {
39         u=V[p][w];
40         if (C[u] !=C[u+LG]) continue;
41
42         if (A[u+LG]=='a')
43         {
44             DA++;
45             MA|=C[u];
46             V[q][DA]=u;
47         }
48         else
49         {
50             SB--;
51             MB|=C[u];
52             V[q][SB]=u;
53         }
54     }
55
56     if (MA==M)
57     {
58         SOL=max(SOL, LG+1);
59         radix(SA, DA, LG+1, q);
60     }
61
62     if (MB==M)
63     {
64         SOL=max(SOL, LG+1);
65         radix(SB, DB, LG+1, q);
66     }
67 }
68 }
```

7.2.3 *Rezolvare detaliată

Capitolul 8

OJI 2012

8.1 blis

Problema 1 - blis

100 de puncte

Se consideră un sir de biți și un număr natural K . Sirul se împarte în secvențe astfel încât fiecare bit din sir să aparțină unei singure secvențe și fiecare secvență să aibă lungimea cel puțin 1 și cel mult K . După împărțire, fiecare secvență de biți se convertește în baza 10, obținându-se un sir de valori zecimale. De exemplu, pentru sirul de biți 1001110111101010011 și $K = 4$, se poate obține 1 0011 101 111 0 1010 011, apoi în baza 10: 1, 3, 5, 7, 0, 10, 3. O altă împărțire poate fi 1 00 1 1 10 11 110 1010 011, adică 1, 0, 1, 1, 2, 3, 6, 10, 3.

Cerințe

Scrieți un program care:

- determină valoarea maximă (în baza 10) care se poate obține dintr-o secvență de cel mult K biți
- împarte sirul inițial în secvențe de cel mult K biți astfel încât sirul zecimal obținut să conțină un subșir strict crescător de lungime maximă posibilă.

Date de intrare

Prima linie a fișierului de intrare **blis.in** conține numărul natural K , iar pe linia a doua se află sirul de biți, sirul neconținând spații.

Date de ieșire

Fișierul de ieșire **blis.out** va conține pe prima linie un număr natural reprezentând valoarea maximă care se poate obține dintr-o secvență de cel mult K biți, iar pe linia a doua un singur număr natural reprezentând lungimea maximă a subșirului strict crescător care se poate obține din sirul de biți prin împărțirea sa în secvențe de cel mult K biți.

Restricții și precizări

- $3 \leq$ lungimea sirului de biți ≤ 100000
- pentru 70% din teste, lungimea sirului de biți ≤ 1000
- $1 \leq K \leq 30$
- Un subșir se obține dintr-un sir prin eliminarea a zero, unul, două sau mai multe elemente;
- O secvență este formată din elemente aflate pe poziții consecutive în sir;
- Pentru rezolvarea corectă a primei cerințe se acordă 20% din punctaj, iar pentru rezolvarea corectă a celei de-a doua se acordă 80

Exemple

blis.in	blis.out	Explicații
4 1001110111101010011	15 6	Secvența de valoare maximă de cel mult 4 biți este 1111, adică 15 în baza 10. Pentru cerința a doua, sirul binar se împarte astfel: 1 00 1 1 10 11 110 1010 011, Se obține sirul zecimal: 1, 0, 1, 1, 2, 3, 6, 10, 3. Subșirul strict crescător maximal de lungime 6 este 0, 1, 2, 3, 6, 10

Timp maxim de executare/test: **1.5** secunde
Memorie: total **32 MB** din care pentru stivă **32 MB**
Dimensiune maximă a sursei: **10 KB**

8.1.1 Indicații de rezolvare

prof. Dan Pracsu

Pentru rezolvarea primei cerințe, există două situații:

a) $k \geq$ lungimea sirului de biți. În acest caz numărul zecimal maxim este dat de întreg sirul convertit în baza 10.

b) $k <$ lungimea sirului. În acest caz, evident că numărul maxim va avea exact k biți. Se construiește mai întâi numărul zecimal x cu primii k biți din sir, apoi se parcurge restul de $N - k$ biți eliminându-se din x cel mai din stânga bit (din cei k) și adăugându-se la dreapta bitul curent. La fiecare pas se actualizează maximul.

Complexitatea este $O(N)$, unde N este lungimea sirului de biți.

În continuare sunt prezentate două soluții de rezolvare pentru cerința a doua.

Soluția 1 (aproximativ 70 puncte, Complexitate $O(N \times N \times k)$)

Vom construi o matrice A , în care $A(i, j) = k$ înseamnă: k este cea mai mică valoare ce poate fi capătul din dreapta unui subșir strict crescător de lungime j care s-a construit cu primii i biți.

Pentru fiecare i între 1 și N , se construiește numărul zecimal x cu ultimii j biți ($i \geq j \geq i-k+1$), apoi se caută în stânga poziției j posibile valori care actualizează valorile minime ale capetelor subșirurilor strict crescătoare de lungimi 1, 2, 3, ...

De remarcat faptul că în matricea A fiecare coloană va avea mereu elementele ordonate strict crescător, la fel cum și la soluția clasică de determinare a *subșirului strict crescător de lungime maximă* în complexitate $O(N \log N)$ se construiește un vector care este mereu ordonat strict crescător.

Soluția 2 (100 puncte, Complexitate $O(N \times k \times (\log N + k))$)

Folosim *metoda programării dinamice "înainte"*. Vom utiliza un vector $BST[i]$ cu semnificația "cea mai mică valoare în care se termină un subșir strict crescător de lungime i ".

Inițializăm $BST[0] = -\text{infinit}$ și $BST[i] = +\text{infinit}$ pentru $1 \leq i \leq Nr_biti$.

Semnificația lui $BST[i] = +\text{infinit}$ este aceea că nu s-a obținut un subșir strict crescător de lungime i . Să presupunem că am ajuns la un bit P și cunoaștem sirul BST construit până la bitul $P - 1$.

Construim toate numerele de cel mult k biți începând cu bitul P . Fie N_j numărul format cu j biți (cei de pe pozițiile $P \dots P + j - 1$). Fie L cea mai mare lungime unde avem $BST[L] < N_j$.

Dacă $BST[L + 1] > N_j$ atunci cu siguranță se poate realiza o îmbunătățire la lungimea $L + 1$. Totuși momentul potrivit pentru a face aceasta îmbunătățire este abia când ajungem să procesăm bitul de pe poziția $P + j - 1$ de aceea doar memorăm că la poziția $P + j - 1$ valoarea N_j ar avea posibilitatea să facă un update pentru lungimea $L + 1$. După ce se generează toate aceste update-uri pentru pozițiile $P, P + 1, \dots, P + k - 1$, se updatează BST cu toate update-urile primite de poziția P .

După ce s-au parcurs toți cei N biți soluția L_{max} poate fi depistată ușor ca fiind cea mai mare poziție unde $BST[L] < \text{infinit}$.

Analiza complexității: Pentru fiecare dintre cele N poziții (factor N) avem de efectuat următoarele operații:

1. Generarea celor k numere (factor k) ; pentru fiecare este necesară determinarea poziției L (factor $\log N$ prin *căutare binară*). Deci avem $O(N \times K \times \log N)$

2. Aplicare update-urilor (cel mult $K * (K - 1)/2$) deci avem $O(N \times k \times k)$

Cumulat avem $O(N \times k \times (\log N + k))$

8.1.2 Cod sursă

Listing 8.1.1: blis70p.cpp

```

1  /*
2   *      Complexitate O(n x n x k)
3   *      70 puncte
4  */
5 #include<iostream>
6
7 #define inFile "blis.in"
8 #define outFile "blis.out"
9 #define dim 1005
10
11 using namespace std;
12
13 char s[dim];
14 int t[dim], n, k, valm;
15 int a[dim][dim], d[dim];
16
17 void Citire()
18 {
19     ifstream fin(inFile);
20     fin >> k;
21     fin >> s;
22     fin.close();
23
24     for (n = 0; s[n]; n++)
25         t[n + 1] = s[n] - '0';
26 }
27
28 void Calcul()
29 {
30     int i, j, x, p;
31
32     // prima cerinta
33     // determin valoarea maxima de cel mult k biti
34     if (k >= n) // iau toti bitii
35     {
36         valm = 0;
37         for (i = 1; i <= n; i++)
38             valm |= (t[i] << (n - i));
39     }
40     else
41     {
42         // iau primii k biti
43         x = 0;
44         p = 0;
45         for (i = 1; i <= k; i++)
46         {
47             x |= (t[i] << (k - i));
48             p = (p << 1) + 1;
49         }
50         p >>= 1;
51         valm = x;
52         // iau pe rand urmatorii biti de la k+1 la n
53         for (i = k + 1; i <= n; i++)
54         {
55             x |= (t[i] << (k - i));
56             x = (x << 1) + t[i];
57             if (valm < x) valm = x;
58         }
59     }
60
61     // a doua cerinta
62     a[0][1] = 100001;
63     d[0] = 1;
64     a[1][1] = t[1];
65     d[1] = 1;
66
67     for (i = 2; i <= n; i++)
68     {
69         for (j = 1; j <= d[i - 1]; j++)
70             a[i][j] = a[i - 1][j];

```

```

71         d[i] = d[i - 1];
72         a[i][d[i] + 1] = 100001;
73         x = 0;
74         for (j = i; (j >= 1) && (j >= i - k + 1); j--)
75         {
76             x = x | (t[j] << (i - j));
77             for (p = 1; p <= d[j-1] && x > a[j-1][p]; p++)
78                 ;
79             if (p <= d[j-1])
80             {
81                 if (x < a[i][p])
82                     a[i][p] = x;
83             }
84             else
85             {
86                 if (p > d[i])
87                 {
88                     d[i] = p;
89                     a[i][p] = x;
90                 }
91                 else if (x < a[i][p]) a[i][p] = x;
92             }
93         }
94     }
95
96     ofstream fout(outFile);
97     fout << valm << "\n" << d[n] << "\n";
98     fout.close();
99 }
100
101 int main()
102 {
103     Citire();
104     Calcul();
105
106     return 0;
107 }
```

Listing 8.1.2: blis100p.cpp

```

1 //Solutia oficiala - implementare STL
2 //100 puncte
3 #include<cstdio>
4 #include<cstring>
5 #include<algorithm>
6 #include<utility>
7 #include<vector>
8
9 #define N 100010
10
11 using namespace std;
12
13 int n,k,i,j,v,big,ST,DR,MI,BST[N];
14 int oo=(1<<30)+100;
15
16 vector< pair<int,int> > UPD[N];
17 char B[N];
18
19 int main()
20 {
21     freopen("7-blis.in", "r", stdin);
22     freopen("blis.out", "w", stdout);
23
24     scanf("%d%s", &k, B+1);
25     n=strlen(B+1);
26     for(i=1;i<=n+1;i++)
27         BST[i]=oo;
28     BST[0]=-1;
29     for(i=1;i<=n;i++)
30     {
31         v=0;
32         K=min(n,i+k-1);
33         for(j=i;j<=K;j++)
34         {
35             v=(v<<1) | (B[j]-'0');
36         }
37         BST[i]=v;
38     }
39
40     for(i=1;i<=n;i++)
41     {
42         for(j=i;j<=n;j++)
43         {
44             if(BST[i]<=BST[j])
45                 cout << BST[i] << " ";
46             else
47                 cout << BST[j] << " ";
48         }
49         cout << endl;
50     }
51 }
```

```

36         big=max(big,v);
37         for(ST=0,DR=i+1;DR-ST-1;)
38         {
39             MI=(ST+DR)/2;
40             if(BST[MI]<v)
41                 ST=MI;
42             else
43                 DR=MI;
44         }
45         if(BST[DR]>v)
46             UPD[j].push_back(make_pair(DR,v));
47     }
48
49     for(auto it=UPD[i].begin(); it!=UPD[i].end(); it++)
50         if(BST[it->first]>it->second)
51             BST[it->first]=it->second;
52     }
53
54     printf("%d\n",big);
55
56     for(ST=0,DR=n+1;DR-ST-1;)
57     {
58         MI=(ST+DR)/2;
59         if(BST[MI]<oo)
60             ST=MI;
61         else
62             DR=MI;
63     }
64
65     printf("%d\n",ST);
66
67     return 0;
68 }
```

Listing 8.1.3: PA_n2blis.cpp

```

1 // Sursa oficiala modificata - cautare secventiala in loc de cautare binara
2 //85 puncte
3 #include<iostream>
4 #include<cstring>
5
6 using namespace std;
7
8 struct nod{int val,lg; nod *urm;};
9 nod *P[100010],*paux;
10 int k,n,i,v,j,K,ST,DR,MI,LMAX,sol,BST[100010],oo=2000000000;
11 char B[100010];
12
13 int main()
14 {
15     freopen("blis.in","r",stdin);
16     freopen("blis.out","w",stdout);
17
18     scanf("%d",&k);
19     scanf("%s",B+1);
20     n=strlen(B+1);
21     for(i=1;i<=n+1;i++)
22         BST[i]=oo;
23     BST[0]=-oo;
24     LMAX=0;
25     for(i=1;i<=n;i++)
26     {
27         v=0;
28         K=i+k-1<n?i+k-1:n;
29         for(j=i;j<=K;j++)
30         {
31             v<<=1;
32             v|=B[j]-'0';
33             sol=v>sol?v:sol;
34             for(DR=LMAX+1;DR>=0;DR--)
35                 if(BST[DR-1]<v)
36                     break;
37                 if(v<BST[DR])
38                 {
39                     paux=new nod;
```

```

40             paux->val=v;
41             paux->lg=DR;
42             paux->urm=P[j];
43             P[j]=paux;
44         }
45     }
46
47     for(;P[i];)
48     {
49         paux=P[i];
50         P[i]=P[i]->urm;
51         if(BST[paux->lg]>paux->val)
52             BST[paux->lg]=paux->val;
53         delete paux;
54     }
55
56     while(BST[LMAX+1]<oo) LMAX++;
57 }
58 printf("%d\n%d",sol,LMAX);
59 return 0;
60 }
```

Listing 8.1.4: PA_pblis.cpp

```

1 //sursa oficiala - liste implementate cu pointeri
2 //100 puncte
3 #include<cstdio>
4 #include<cstring>
5
6 using namespace std;
7
8 struct nod
9 {
10     int val,lg;
11     nod *urm;
12 };
13
14 nod *P[100010],*paux;
15 int k,n,i,v,j,K,ST,DR,MI,LMAX,sol,BST[100010],oo=2000000000;
16 char B[100010];
17
18 int main()
19 {
20     freopen("blis.in","r",stdin);
21     freopen("blis.out","w",stdout);
22
23     scanf("%d",&k);
24     scanf("%s",B+1);
25
26     n=strlen(B+1);
27     for(i=1;i<=n+1;i++)
28         BST[i]=oo;
29     BST[0]=-oo;
30     LMAX=0;
31     for(i=1;i<=n;i++)
32     {
33         v=0;
34         K=i+k-1<n?i+k-1:n;
35         for(j=i;j<=K;j++)
36         {
37             v<<=1;
38             v|=B[j]-'0';
39             sol=v>sol?v:sol;
40             for(ST=0,DR=LMAX+1;DR-ST-1;)
41             {
42                 MI=(ST+DR)>>1;
43                 if(BST[MI]<v)
44                     ST=MI;
45                 else
46                     DR=MI;
47             }
48             if(v<BST[DR])
49             {
50                 paux=new nod;
```

```

52         paux->val=v;
53         paux->lg=DR;
54         paux->urm=P[j];
55         P[j]=paux;
56     }
57 }
58
59     for(;P[i];)
60     {
61         paux=P[i];
62         P[i]=P[i]->urm;
63         if(BST[paux->lg]>paux->val)
64             BST[paux->lg]=paux->val;
65
66         delete paux;
67     }
68
69     while(BST[LMAX+1]<oo)
70         LMAX++;
71 }
72
73 printf("%d\n%d",sol,LMAX);
74 return 0;
75 }
```

8.1.3 *Rezolvare detaliată

8.2 parc

Problema 2 - parc

Un parc de formă dreptunghiulară este format din zone pietonale și pistele de biciclete. Reprezentând harta parcului într-un sistem cartezian, cu coordonata colțului stânga-jos $(0, 0)$, pistele de biciclete sunt reprezentate prin dungi orizontale sau verticale colorate cu gri, iar zonele pietonale au culoarea albă, ca în figura din dreapta.

Vizitatorii parcului se pot plimba liber pe zonele pietonale în orice direcție, însă pistele de biciclete se vor traversa, în linie dreaptă, paralel cu axele. În figura alăturată avem un parc de dimensiuni 10×8 , cu pistele de biciclete verticale între 2 și 4 respectiv 5 și 8, și orizontale între

0 și 1 respectiv între 2 și 4. Gigel se află în punctul $A(1, 1)$ și poate să ajungă pe drumul cel mai scurt la prietenul lui, în punctul $B(8, 7)$ deplasându-se astfel: pornește din punctul $(1, 1)$ și parcurge un traseu format din segmente cu extremitățile în punctele de coordonate $(1.5, 2)$ $(1.5, 4)$ $(2, 5)$ $(4, 5)$ $(5, 7)$ și în final ajunge în punctul de coordonate $(8, 7)$.

Lungimea totală a drumului va fi aproximativ 11.4721359.

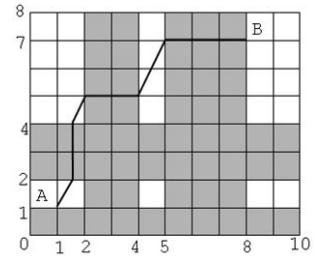


Figura 8.1: parc

Cerințe

Cunoscând dimensiunile parcului, coordonatele lui Gigel, coordonatele prietenului lui și pozițiile pistelor de biciclete, să se calculeze lungimea drumului minim și numărul drumurilor distințe de lungime minimă.

Date de intrare

Fișierul **parc.in** conține pe prima linie două numere naturale X_{parc} și Y_{parc} separate prin spațiu, reprezentând dimensiunile parcului în direcțiile Ox respectiv Oy . Linia a doua va conține patru numere separate prin spațiu xG , yG , xpr și ypr ce reprezintă coordonatele lui Gigel și coordonatele prietenului lui. Linia a treia va conține un număr natural m , reprezentând numărul pistelor verticale. Următoarele m linii vor conține perechi de valori de pe axa Ox ce delimităază câte o pistă de biciclete verticală. Următoarea linie va conține un număr natural n , reprezentând numărul pistelor orizontale. Următoarele n linii vor conține perechi de valori de pe axa Oy ce delimităază câte o pistă de biciclete orizontală.

Date de ieșire

Fișierul **parc.out** va conține pe prima linie lungimea minimă a drumului cerut de problemă, un **număr real**. Linia a doua va conține numărul drumurilor minime distințe, un **număr natural**.

Restricții și precizări

- $0 \leq xG, xpr \leq X_{parc} \leq 30000, 0 \leq yG, ypr \leq Y_{parc} \leq 30000$;
- $0 < m, n < 2000$;
- perechile de numere naturale ce definesc o pistă nu sunt ordonate; - pistele orizontale, și cele verticale nu sunt ordonate în fișierul de intrare;
- două piste de aceeași direcție nu se suprapun;
- Gigel și prietenului lui sunt pe zone pietonale (inclusând și marginile acestora);
- două drumuri sunt distințe dacă diferă prin cel puțin un punct;
- numărul de drumuri distințe nu va depăși 1 000 000 000;
- lungimea drumului din fișierul de ieșire este un număr real ce se acceptă cu eroare maxima de 0.01;
- nu se admite formatul să fie specific pentru afișarea numerelor reale;
- prima cerință valorează 40% din punctaj, iar a doua valorează 60% din punctaj.

Exemple

parc.in	parc.out	Explicații
10 8	11.472136	- lungimea drumului minim a fost calculată în exemplul de mai sus, rezultatul se poate tipări cu oricâte zecimale, diferența absolută față de rezultatul oficial să nu difere cu mai mult de 0.01
1 1 8 7	1	
2		
5 8		
2 4		
2		- există un singur drum de lungime minimă
4 2		
0 1		

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

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

Dimensiune maximă a sursei: **10 KB**

8.2.1 Indicații de rezolvare

prof. Zoltan Szabo

Cel mai scurt drum dintre două puncte în plan se obține pe linie dreaptă. Astfel dacă eliminăm porțiunile de traversare a pistelor de biciclete, traseul lui Gigel către prietenul lui trebuie să fie un segment. Această lungime se calculează cu *teorema lui Pitagora*, la care se adaugă lungimile de traversare orizontală și verticală.

Soluția nu este unică de fiecare dată! Dacă traseul lui Gigel intersectează punctul de întâlnire a două piste de biciclete, numărul soluțiilor se dublează, fiindcă Gigel poate traversa zona în două moduri: orizontal-vertical sau vertical-horizontal. În ambele cazuri două laturi alăturate ale aceluiași dreptunghi, deci ambele sunt soluții corecte. Astfel, numărul soluțiilor distințe pentru orice teste de intrare este o putere a lui 2.

8.2.2 Cod sursă

Listing 8.2.1: PA_parc2.cpp

```

1 //Problema parc
2 //Autor : Zoltan Szabo
3 //Implementare alternativa : A. Panaete
4 //Complexitate O( N log N + M log M ) (de la sortare ca prima a doua e liniara)
5 #include<cstdio>
6 #include<algorithm>
7 #include<cmath>
8
9 #define MOD 31313

```

```

10
11  using namespace std;
12
13  int XP,YP,xg,yg,xp,yp,xc,yc,XL,XR,YD,YU,m,n,i,j,M,N,L,R,U,D,X[20010],Y[20010];
14  double SOL;
15
16  void read(),solve();
17
18  int main()
19  {
20      read();
21      solve();
22      return 0;
23  }
24
25  void read()
26  {
27      freopen("parc.in","r",stdin);
28      freopen("parc.out","w",stdout);
29
30      scanf("%d%d",&XP,&YP);
31      XP++;
32      YP++;
33      scanf("%d%d",&xg,&yg);
34      scanf("%d%d",&xp,&yp);
35
36      int aux;
37      scanf("%d",&m);
38      m*=2;
39      X[++R]=0;
40      for(;m;m--)
41      {
42          scanf("%d",&aux);
43          X[++R]=aux;
44      }
45
46      X[+R]=XP;
47      L=1;
48      scanf("%d",&m);
49      m*=2;
50      Y[+U]=0;
51      for(;m;m--)
52      {
53          scanf("%d",&aux);
54          Y[+U]=aux;
55      }
56
57      Y[+U]=YP;
58      D=1;
59  }
60
61  void solve()
62  {
63      if(xp==xg) {YP=yg>yp?yg-yp:yp-yg;printf("%d\n1\n",YP);return;}
64      if(yp==yg) {XP=xg>xp?xg-xp:xp-xg;printf("%d\n1\n",XP);return;}
65
66      if(xg>xp) {xg=XP-xg;xp=XP-xp;for(i=L;i<=R;i++)X[i]=XP-X[i];}
67      if(yg>yp) {yg=YP-yg;yp=YP-yp;for(i=D;i<=U;i++)Y[i]=YP-Y[i];}
68
69      sort(X+L,X+R+1);
70      for(;;L+=2) if(X[L]<=xg&&xg<=X[L+1])break;
71      for(;;R-=2) if(X[R-1]<=xp&&xp<=X[R])break;
72
73      X[L]=xg;X[R]=xp;
74
75      sort(Y+D,Y+U+1);
76      for(;D+=2) if(Y[D]<=yg&&yg<=Y[D+1])break;
77      for(;;U-=2) if(Y[U-1]<=yp&&yp<=Y[U])break;
78
79      Y[D]=yg;Y[U]=yp;
80      int CX=0,CY=0;double cx,cy;
81
82      for(i=L;i<R;i+=2)CX+=X[i+1]-X[i];
83      for(i=D;i<U;i+=2)CY+=Y[i+1]-Y[i];
84
85      SOL+=(double)(X[R]-X[L]-CX);

```

```

86     SOL+=(double) (Y[U]-Y[D]-CY);
87     cx=(double) CX; cy=(double) CY;
88     SOL+=sqrt(cx*cx+cy*cy);
89
90     printf("%lf\n", SOL);
91
92     for(n=0, i=L+1; i<=R; i+=2, n++) X[n+1]=X[n]+X[i]-X[i-1];
93     for(m=0, i=D+1; i<=U; i+=2, m++) Y[m+1]=Y[m]+Y[i]-Y[i-1];
94     for(D=X[n], U=Y[m]; U;) {R=D%U; D=U; U=R; }
95
96     xg=X[n]/D; yg=Y[m]/D;
97
98     for(i=1, j=1, U=1; i<n&&j<m;)
99     {
100         if(X[i]%xg){i++; continue;}
101         if(Y[j]%yg){j++; continue;}
102         if(X[i]/xg<Y[j]/yg){i++; continue;}
103         if(X[i]/xg>Y[j]/yg){j++; continue;}
104         i++; j++;
105         U*=2;
106     }
107
108     printf("%d\n", U);
109 }

```

Listing 8.2.2: parc_Zoli_double.cpp

```

1 //Solutia oficiala
2 //100 puncte
3 #include <iostream>
4 #include<cstdio>
5 #include<cmath>
6
7 using namespace std;
8
9 void scufundare(int p, int a[2][10000],int n)
10 {
11     int pozmax=p;
12     if (2*p<=n && a[0][pozmax]<a[0][2*p]) pozmax=2*p;
13     if (2*p+1<=n && a[0][pozmax]<a[0][2*p+1]) pozmax=2*p+1;
14     if (p!=pozmax)
15     {    swap(a[0][p],a[0][pozmax]);
16         swap(a[1][p],a[1][pozmax]);
17         scufundare(pozmax,a,n);
18     }
19 }
20
21 // pistele de biciclete le sortez in ordine crescatoare
22 void heapsort(int a[2][10000],int n)
23 {
24     int i;
25     for (i=n/2;i>=1;--i)
26         scufundare(i,a,n);
27     for (i=n;i>1;--i)
28     {
29         swap(a[0][1],a[0][i]);
30         swap(a[1][1],a[1][i]);
31         scufundare(1,a,i-1);
32     }
33 }
34 int main()
35 {
36     int pv[2][10000],po[2][10000],xp,yp,xg,yg,xpr,ypr,m,n,i,j,p2,
37     sv[10000],so[10000];
38     int startv, finishv,starto,finisho,xinv=0,yinv=0,d_oriz,d_vert,
39     cx,cy,semnx=1,semny=1;
40     int kx,ky,segmx[10000],segmy[10000];
41     double lung;
42
43     freopen("parc.in","r",stdin);
44     freopen("parc.out","w",stdout);
45
46     scanf("%d %d\n",&xp,&yp);
47     scanf("%d %d %d %d\n",&xg,&yg,&xpr,&ypr);

```

```

48     if (xpr<xg)      // voi prelucra doar un singur caz:
49         // cand gigel este in stanga-jos
50     {
51         xinv=1;        // iar prietenul este in dreapta-sus
52         xg=xp-xg;    // pentru a rezolva acest lucru, voi folosi simetria
53             // pe OX si OY daca este necesar
54         xpr=xp-xpr;
55         semnx=-1;
56     }
57
58     if (ypr<yg)
59     {
60         yinv=1;
61         semny=-1;
62         yg=yp-yg;
63         ypr=yp-ypr;
64     }
65
66     if (xpr==xg)
67     {
68         printf("%d\n",ypr-yg);
69         return 0;
70     }
71     if (ypr==yg)
72     {
73         printf("%d\n",xpr-xg);
74         return 0;
75     }
76
77     scanf("%d\n",&m);
78     if (xinv)
79     {
80         for (i=1;i<=m;++i)
81         {
82             scanf("%d %d\n",&pv[0][i],&pv[1][i]);
83             if (pv[0][i]<pv[1][i]) swap(pv[0][i],pv[1][i]);
84             pv[0][i]=xp-pv[0][i]; // citesc pistele verticale (x-urile)
85                 // si daca e nevoie
86             pv[1][i]=xp-pv[1][i]; // transform in simetricul lor
87                 // sa le prelucrez de la st. da dr.
88         }
89     }
90     else
91     {
92         for (i=1;i<=m;++i)
93         {
94             scanf("%d %d\n",&pv[0][i],&pv[1][i]);
95             if (pv[0][i]>pv[1][i]) swap(pv[0][i],pv[1][i]);
96         }
97
98     heapsort(pv,m);           // sortez punctele in ordine crescatoare
99     scanf("%d\n",&n);
100    if (yinv)
101    {
102        for (i=1;i<=n;++i)
103        {
104            // citesc pistele orizontale (y-urile) si daca e nevoie
105            scanf("%d %d\n",&po[0][i],&po[1][i]);
106
107            // le transform in simetricul lor sa le parcurg de jos in sus
108            if (po[0][i]<po[1][i]) swap(po[0][i],po[1][i]);
109            po[0][i]=yp-po[0][i];
110            po[1][i]=yp-po[1][i];
111        }
112    }
113    else
114    {
115        for (i=1;i<=n;++i)
116        {
117            scanf("%d %d\n",&po[0][i],&po[1][i]);
118            if (po[0][i]>po[1][i]) swap(po[0][i],po[1][i]);
119        }
120
121    heapsort(po,n);           // sortez punctele in ordine crescatoare
122    sv[0]=0;
123    for(i=1;i<=m;++i)
124        sv[i]=sv[i-1]+pv[1][i]-pv[0][i];      // memorez grosimile pistelor,
125                                            // ca si sume partiale
126    so[0]=0;
127    for(i=1;i<=n;++i)
128        so[i]=so[i-1]+po[1][i]-po[0][i];      // idem
129
130    startv=1;
131    while (xg>pv[0][startv]) ++startv; // ma uit cate pistele verticale
132                                            // sunt intre Gigel si prietenul lui
133    finishv=m;
134    while (xpr<pv[1][finishv]) --finishv;

```

```

124
125     starto=1;
126     while (yg>po[0][starto])
127         ++starto;           // ma uit cate piste orizontale sunt intre Gigel si
128                           // prietenul lui
129     finisho=n;
130     while (ypr<po[1][finisho]) --finisho;
131     d_oriz=sv[finishv]-sv[startv-1];           // grosimea pistelor verticale
132     d_vert=so[finisho]-so[starto-1];           // grosimea pistelor verticale
133     cx=xpr-xg-d_oriz;                         // cea mai scurta distanta dintre doua puncte
134                           // se calculeaza dintr-un triunghi
135     cy=ypr-yg-d_vert;                          // dreptunghic cu catele cx si cy
136                           // cx si cy reprezinta lungimile catetelor triunghiului dreptunghic
137                           // fara pistele orizontale si verticale
138     lung=sqrt((float)(cx*cx+cy*cy))+d_oriz+d_vert; // la lungimea ipotenuzei
139                           // se adauga distantele orizontale si verticale
140     segmx[0]=pv[0][startv]-xg;kx=0;
141     for (i=startv+1;i<=finishv;++i)
142     {
143         ++kx;
144         segmx[kx]=segmx[kx-1]+pv[0][i]-pv[1][i-1];
145     }
146     segmy[0]=po[0][starto]-yg;ky=0;
147     for (i=starto+1;i<=finisho;++i)
148     {
149         ++ky;
150         segmy[ky]=segmy[ky-1]+po[0][i]-po[1][i-1];
151     }
152
153     p2=1;
154     i=j=0;
155     while(i<=kx && j<=ky)
156         if (segmx[i]*cy==segmy[j]*cx)
157         {
158             p2*=2;
159             ++i;++j;
160         }
161         else
162             if (segmx[i]*cy>segmy[j]*cx)
163                 ++j;
164             else
165                 ++i;
166
167     printf("%lf\n",lung);
168     printf("%d\n",p2);
169     return 0;
170 }
```

8.2.3 *Rezolvare detaliată

Capitolul 9

OJI 2011

9.1 suma

Problema 1 - suma

Constructorii angajați de faraonul Keops au terminat construirea piramidei în trepte mult visată. Măreața piramidă are n camere identice de formă cubică, numerotate de la 1 la n , dispuse pe m niveluri astfel:

- camera din vârful piramidei formează nivelul 1 și are numărul 1;
- nivelul 2 al piramidei este format din următoarele 4 camere care în secțiune cu un plan paralel cu baza au aspectul unei matrice cu 2 linii și 2 coloane; camerele de pe nivelul 2 sunt numerotate de la 2 la 5 în ordinea crescătoare a liniilor matricei, iar pe aceeași linie în ordinea crescătoare a coloanelor matricei;
-
- nivelul m al piramidei este format din $m * m$ camere și au, în secțiune cu un plan paralel cu baza, aspectul unei matrice cu m linii și m coloane; camerele de pe nivelul m sunt numerotate în continuarea celor de pe nivelurile 1, 2, ..., $m - 1$, în ordinea crescătoare a liniilor matricei de secțiune, iar pe aceeași linie în ordinea crescătoare a coloanelor matricei. De exemplu, piramida din desenul de mai sus are $n = 30$, $m = 4$ iar camerele sunt numerotate și dispuse pe niveluri astfel:

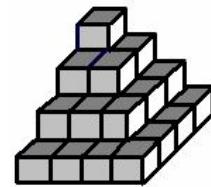


Figura 9.1: suma

Nivel 1	Nivel 2	Nivel 3	Nivel 4
1	2 3 4 5	6 7 8 9 10 11 12 13 14	15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
1 cameră	2x2 camere	3x3 camere	4x4 camere

Figura 9.2: suma

Nivelurile de camere sunt poziționate astfel încât camerele de pe prima linie și prima coloană a fiecărui nivel să se suprapună. Pentru exemplul dat, camerele 1, 2, 6 și 15 sunt situate una sub alta, în această ordine.

Accesul în oricare din camerele piramidei, situate pe diferite niveluri, se realizează prin drumuri construite astfel:

- intrarea în piramidă se face doar prin camera din vârful ei, cea cu numărul 1;
- din camera cu numărul k de pe un drum se poate intra într-o din cele patru camere situate pe nivelul imediat următor al piramidei și anume: camera situată sub cea cu numărul k sau una din cele trei camere vecine acesteia în secțiune (în direcțiile Est, Sud-Est, Sud, considerând secțiunile poziționate ca în imaginile de mai sus). De exemplu, din camera cu numărul 10 se poate intra într-o din camerele cu numerele: 20, 21, 24 sau 25.

Faraonul privește cu mândrie și tristețe la frumoasa piramidă. Banii din visterie s-au împuținat iar camerele piramidei trebuie finisate și decorate. Scribul său favorit a refăcut toate calculele, a eliminat obiectele inutile și a stabilit pentru fiecare cameră k un cost c_k aferent finisării și decorării ei ($1 \leq k \leq n$).

Însă, suma totală necesară fiind încă mare, faraonul i-a cerut scribului să aleagă un drum, dintre cele construite, care să treacă prin toate nivelurile piramidei astfel încât suma s a tuturor costurilor aferente finisării și decorării camerelor de pe acest drum să fie minimă. Deocamdată, doar aceste camere vor fi aranjate...

Cerințe

Scrieți un program care să determine numărul m de niveluri ale piramidei, suma minimă s a tuturor costurilor aferente finisării și decorării camerelor de pe un drum ce trece prin toate nivelurile piramidei, construit în modul descris în enunț, precum și un astfel de drum pentru care se obține suma minimă, putând fi ales de scrib.

Date de intrare

Fișierul de intrare **suma.in** conține pe prima linie numărul natural nenul n reprezentând numărul de camere din piramidă. A doua linie conține n numere naturale nenule c_1, c_2, \dots, c_n , separate prin câte un spațiu, reprezentând costurile aferente finisării și decorării camerelor, în ordinea numerotării lor.

Date de ieșire

Fișierul de ieșire **suma.out** va conține pe prima linie două numere naturale m și s , separate printr-un singur spațiu, cu semnificația din enunț. Cea de-a doua linie va conține, separate prin câte un spațiu, în ordinea parcurgerii lor, numerele camerelor de pe un drum ce trece prin toate nivelurile piramidei, drum pentru care se obține suma minimă s .

Restricții și precizări

- $1 \leq n \leq 63365$
- Pentru fiecare valoare n citită se poate construi în modul descris în enunț o piramidă în trepte cu n camere
 - $1 \leq c_1, c_2, \dots, c_n < 100$
 - Dacă există mai multe drumuri ce trec prin toate nivelurile piramidei și pentru care se obține suma minimă s , atunci drumul ales va fi cel mai mic drum din punct de vedere lexicografic.
 - Drumul $a_1, a_2, a_3, \dots, a_m$ este mai mic, din punct de vedere lexicografic, decât drumul $b_1, b_2, b_3, \dots, b_m$ dacă există un indice j ($1 \leq j \leq m$) astfel încât $a_1 = b_1, a_2 = b_2, \dots, a_{j-1} = b_{j-1}$ și $a_j < b_j$.
 - Se acordă:
 - 10% din punctaj pentru determinarea corectă a numărului m de niveluri ale piramidei
 - 30% din punctaj pentru determinarea corectă a sumei minime s
 - 60% din punctaj pentru determinarea corectă a drumului cerut.

Exemple

suma.in	suma.out	Explicații						
14 7 8 4 5 5 8 4 2 7 7 8 3 1 6	3 13 1 3 8	<p>Piramida conține 14 camere dispuse pe $m = 3$ trei niveluri. Nivelurile conțin valorile:</p> <table style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">Nivelul 1</td> <td style="text-align: center;">Nivelul 2</td> <td style="text-align: center;">Nivelul 3</td> </tr> <tr> <td style="text-align: center;">7</td> <td style="text-align: center;">8 4 5 5</td> <td style="text-align: center;">8 4 2 7 7 8 3 1 6</td> </tr> </table> <p>Suma minimă s a tuturor costurilor aferente finisării și decorării camerelor de un drum ce trece prin toate cele 3 niveluri ale piramidei este 13. Există mai multe drumuri pentru care se poate obține suma minimă: [1,3,8], [1,4,13], [1,5,13]. Din punct de vedere lexicografic, cel mai mic drum dintre aceste drumuri este: [1,3,8].</p>	Nivelul 1	Nivelul 2	Nivelul 3	7	8 4 5 5	8 4 2 7 7 8 3 1 6
Nivelul 1	Nivelul 2	Nivelul 3						
7	8 4 5 5	8 4 2 7 7 8 3 1 6						

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

Memorie: total **2 MB** din care pentru stivă **1.5 MB**

Dimensiune maximă a sursei: **5 KB**

9.1.1 Indicații de rezolvare

prof. Carmen Mincă, Colegiul Național de Informatică "Tudor Vianu" București

O soluție se poate obține aplicând *metoda programării dinamice, metoda înainte*.

Drumul căutat va trece prin m camere din piramidă, câte una de pe fiecare nivel.

Numărul m de niveluri se poate determina utilizând relația

$$n = 1 + 4 + 9 + \dots + m^2 = \frac{m(m+1)(2m+1)}{6}$$

Pentru a memora costurile asociate camerelor precum și sumele minime asociate drumurilor construite pentru fiecare cameră, drumuri ce ajung pe ultimul nivel al piramidei, se pot folosi două tablouri tridimensionale.

Fie A masivul ce memorează costurile camerelor pe secțiuni și T masivul sumelor minime. Pentru orice cameră situată pe un nivel k al piramidei, linia i și coloana j , suma minimă a costurilor asociate unui drum ce pornește din această cameră și se oprește pe ultimul nivel al piramidei se poate calcula astfel:

$T[m][i][j] = A[m][i][j]$, pentru $i=1, 2, \dots, m$ și $j=1, 2, 3, \dots, m$
 $T[k][i][j] = A[k][i][j] + \min T[k+1][i][j], T[k+1][i][j+1], T[k+1][i+1][j], T[k+1][i+1][j+1]$,
 pentru $k=m-1, m-2, \dots, 1; i=1, 2, \dots, k; j=1, 2, \dots, k$.

Suma minimă va fi memorată în $T[1][1][1]$. Pentru a afișa numerele camerelor se află pe cel mai mic drum, din punct de vedere lexicografic, ale căror costuri formează suma minimă $T[1][1][1]$ refacem "traseul" prin care a fost obținută această sumă în T , pornind de la $T[1][1][1]$ sau se memorează coordonatele k, i și j ale camerelor în timpul construirii sumelor parțiale minime.

9.1.2 Cod sursă

Listing 9.1.1: sumal_100.cpp

```

1 //prof.Carmen Minca
2 #include <iostream>
3
4 using namespace std;
5
6 int a[58][58][58], t[58][58][58];
7 int n,k;
8
9 ifstream f("suma.in");
10 ofstream g("suma.out");
11
12 void citeste()
13 {
14     int i,l,c,x;
15     k=0;
16     f>>n;
17
18     for(i=1;i<=n;i++)
19         while (n>0)
20         { k++;
21             for(l=1;l<=k;l++)
22                 for(c=1;c<=k;c++)
23                     { f>>x;
24                         t[k][l][c]=a[k][l][c]=x;
25                     }
26             n=n-k*k;
27         }
28
29 g<<k<<' ';//nr niveluri
30 f.close();
31 }
32
33 void suma()
34 { int i,l,c,x;
35     for(i=k-1;i>=1;i--)
36         for(l=1;l<=i;l++)
37             for(c=1;c<=i;c++)

```

```

38         { if(t[i+1][l][c]<=t[i+1][l][c+1]) x=t[i+1][l][c];
39             else x=t[i+1][l][c+1];
40
41             if(x>t[i+1][l+1][c])x=t[i+1][l+1][c];
42             if(x>t[i+1][l+1][c+1])x=t[i+1][l+1][c+1];
43             t[i][l][c]=t[i][l][c]+x;
44         }
45     }
46
47 void drum()
48 {int c,l,i=1,s,nr=0;
49   s=t[1][1][1];
50   g<<s;
51   g<<endl<<1;
52   l=c=1;
53   do
54   { nr=nr+i*i;
55     s=s-a[i][l][c]; i++;
56
57     if(s)
58     {if(t[i][l][c]==s) {l=l;c=c;}
59       else
60       if(t[i][l][c+1]==s)c=c+1;
61       else
62       if(s==t[i][l+1][c])l=l+1;
63       else {l=l+1;c=c+1;}
64       g<<' '<<nr+(l-1)*i+c; }
65   } while(s);
66
67   g<<endl;
68   g.close();
69 }
70
71
72 int main()
73 {citere(); 
74 suma();
75 drum();
76 }
```

Listing 9.1.2: suma2_100.cpp

```

1 //stud. Filip Buruiana
2 #include <stdio.h>
3
4 #define INF 999999999
5 #define NMax 63366
6 #define MAXLEV 58
7
8 int N, C[NMax];
9 int pd[NMax], next[NMax], from[MAXLEV], to[MAXLEV], lev;
10
11 inline void update(int i, int nxt)
12 {
13     if (pd[nxt] + C[i] < pd[i])
14     {
15         pd[i] = pd[nxt] + C[i];
16         next[i] = nxt;
17     }
18 }
19
20 int main()
21 {
22     int i, j, sub, l, c;
23
24     freopen("suma.in", "r", stdin);
25     freopen("suma.out", "w", stdout);
26
27     scanf("%d", &N);
28     for (i = 1; i <= N; ++i)
29         scanf("%d", &C[i]);
30
31     for (i = 1; i * (i+1) * (2*i+1) / 6 < N; ++i);
32     lev = i;
33 }
```

```

34     from[lev] = N-lev*lev+1; to[lev] = N;
35     for (i = lev-1; i; --i)
36         to[i]=from[i+1]-1, from[i]=to[i]-i*i+1;
37
38     for (i = from[lev]; i <= to[lev]; ++i)
39         pd[i] = C[i];
40     for (i = lev-1; i; --i)
41     {
42         for (j = from[i]; j <= to[i]; ++j)
43         {
44             l = (j-from[i])/i + 1;
45             c = (j-from[i])%i + 1;
46             sub = from[i+1]+(l-1)*(i+1)+c-1;
47
48             pd[j] = INF;
49             update(j, sub);
50             update(j, sub+1);
51             update(j, sub+i+1);
52             update(j, sub+i+2);
53         }
54     }
55
56     printf("%d %d\n", lev, pd[1]);
57     for (i = 1; i; i = next[i])
58         printf("%d ", i);
59     printf("\n");
60
61     return 0;
62 }
```

Listing 9.1.3: suma3_st.cpp

```

1 //prof. Carmen Minca
2 #include <stdio.h>
3
4 int a[58][58][58],t[58][58][58];
5 int n,k;
6
7 void citeste()
8 {
9     freopen("suma.in", "r", stdin);
10    int i,l,c,x;
11    k=0;
12    scanf("%d",&n);
13
14    for(i=1;i<=n;i++)
15        while (n>0)
16        { k++;
17            for(l=1;l<=k;l++)
18                for(c=1;c<=k;c++)
19                { scanf("%d",&x);
20                  t[k][l][c]=a[k][l][c]=x;
21                }
22            n=n-k*k;
23        }
24    }
25
26 void suma()
27 { int i,l,c,x;
28    for(i=k-1;i>=1;i--)
29        for(l=1;l<=i;l++)
30            for(c=1;c<=i;c++)
31            { if(t[i+1][l][c]<=t[i+1][l][c+1]) x=t[i+1][l][c];
32              else x=t[i+1][l][c+1];
33              if(x>t[i+1][l+1][c]) x=t[i+1][l+1][c];
34              if(x>t[i+1][l+1][c+1]) x=t[i+1][l+1][c+1];
35              t[i][l][c]=t[i][l][c]+x;
36            }
37    }
38
39 void drum()
40 { int c,l,i=1,s,nr=0;
41    freopen("suma.out", "w", stdout);
42    s=t[1][1][1];
43    printf("%d ",k); //nr niveluri
```

```

44     printf("%d\n", s);
45     l=c=1;
46 do
47 { nr=nr+i*i;
48   s=s-a[i][l][c]; i++;
49   if(s)
50   {if(t[i][l][c]==s) {l=l;c=c;}
51   else
52   if(t[i][l][c+1]==s)c=c+1;
53   else
54   if(s==t[i][l+1][c])l=l+1;
55   else {l=l+1;c=c+1;}
56   printf(" %d",nr+(l-1)*i+c); }
57
58 } while(s);
59
60 printf("\n");
61 }
62
63 int main()
64 {citeste();
65 suma();
66 drum();
67 }
```

9.1.3 *Rezolvare detaliată

9.2 ubuntzei

Problema 2 - ubuntzei

100 de puncte

Trei ubuntzei au hotărât ca anul acesta să petreacă ziua de 1 Mai pe malul Mării Negre împreună cu prietenii lor, motiv pentru care au pus la cale o excursie pe un traseu care să plece din orașul lor Cluj-Napoca spre Vama Veche, unde nisipul îi aşteaptă.

În țara ubuntzelor există N localități, numerotate de la 1 la N , legate între ele prin M șosele bidirectionale de diferite lungimi. Localitatea de plecare a ubuntzelor, orașul Cluj-Napoca, este numerotată cu 1, iar localitatea destinație, Vama Veche, cu N . Între oricare două localități există cel mult o șosea. Fiecare șosea unește două localități distincte și se poate călători între oricare două localități circulând numai pe șosele.

Prietenii ubuntzelor locuiesc în K localități distincte, diferite de Cluj-Napoca și Vama Veche. Pentru a nu călători singuri, cei trei ubuntzei vor să treacă prin cele K localități în care locuiesc prietenii lor, și apoi, împreună cu aceștia, să-și continue excursia către mare.

Nerăbdători să ajungă cât mai repede la destinație, ubuntzeii s-au hotărât să își stabilească un traseu de lungime minimă care să treacă prin toate cele K localități.

Cerințe

Scrieți un program care să determine, pentru ubuntzei, lungimea minimă L a unui traseu de la Cluj-Napoca la Vama Veche ce trece prin toate cele K localități.

Date de intrare

Prima linie a fișierului de intrare **ubuntzei.in** conține două numere naturale N M , separate printr-un spațiu, cu semnificația din enunț. A doua linie a fișierului conține $K + 1$ numere naturale distincte: K C_1 $C_2 \dots C_K$, separate prin câte un spațiu, K având semnificația din enunț iar C_1 , C_2, \dots, C_K reprezentând cele K localități în care locuiesc prietenii. Fiecare din următoarele M linii conține câte trei numere naturale x y z , separate prin câte un spațiu, reprezentând o șosea care leagă localitatea x de localitatea y și are lungimea z .

Date de ieșire

Fișierul de ieșire **ubuntzei.out** va conține numărul natural L reprezentând lungimea minimă căutată.

Restricții și precizări

- $1 \leq N \leq 2000$
- $1 \leq M \leq 10000$
- $0 \leq K \leq \min\{15, N - 2\}$
- $2 \leq C_1, C_2, \dots, C_K \leq N - 1$
- Traseul poate trece de mai multe ori prin oricare localitate.
- Costul unei muchii va fi cuprins între 1 și 10^5 .
- Pentru primele 20% din teste $K = 0$.
- Pentru primele 50% din teste $K \leq 10$.
- Pentru primele 70% din teste $N \leq 200$.

Exemple

ubuntzei.in	ubuntzei.out	Explicații
4 5	4	Există un singur traseu de lungime minimă de la localitatea 4 la localitatea 4 și care trece prin localitatea 2, și anume: [1,2,3,4]. Lungimea L a acestui traseu este 4.
1 2		
1 2 1		
1 3 1		
2 3 1		
2 4 4		
3 4 2		

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

Memorie: total **32** MB din care pentru stivă **20 MB**

Dimensiune maximă a sursei: **20 KB**

9.2.1 Indicații de rezolvare

stud. Marius Stroe, Universitatea "Babeș Bolyai", Facultatea de Informatică, Cluj-Napoca

Soluție 100 puncte

Fie $G = (V, E)$ graful neorientat ce modelează țara din enunțul problemei. Cerința constă în a găsi un drum de lungime minimă în graful G de la nodul de start la nodul destinație trecând cel puțin o dată prin cele K orașe date.

În acest sens, construim un nou graf $G' = (V', E')$, unde V' este o mulțime de perechi (u, s) , astfel încât u este un nod din V , iar s este o submulțime a celor K orașe. Perechea (u, s) semnifică faptul că suntem în nodul u și am vizitat nodurile din mulțimea s .

Astfel, vom aplica *algoritmul lui Dijkstra* pe acest graf căutând drumul de lungime minimă de la nodul $(1, \{\})$ la nodul $(N, \{C_1, C_2, \dots, C_K\})$. În implementare, mulțimile se pot reține cu ajutorul *biților* și se pot manipula cu ajutorul *măștilor pe biți*.

Complexitate finală: $O(K * M \log_2(N) + 2^K K^2 \log_2(K))$.

Se observă că graful G' este aciclic. Astfel, o soluție alternativă folosește *metoda programării dinamice*, calculând $dp(i, s)$ ca fiind costul minim de a ajunge în orașul i și să fi trecut cel puțin o dată prin orașele din mulțimea s . Recurența este

$$dp(i, s) = \min\{dp(j, s - \{i\}) + distG(i, j) : j \in s \text{ și } j \neq i\}$$

unde $distG(i, j)$ reprezintă distanța în graful G , graful inițial, de la nodul i la nodul j .

Soluție 70 de puncte

Această soluție se bazează pe ideea anterioară, doar că drumurile minime dintre oricare două noduri din cele K se calculează cu *algoritmul lui Floyd-Warshall* în complexitate $O(N^3)$. Iar drumul minim în graful G' se calculează cu oricare din variantele prezentate la soluția anterioară, inclusiv *algoritmul lui Bellman Ford cu coadă*.

Soluție 50 puncte

Vom precacula cu *algoritmul lui Floyd-Warshall* distanțele dintre oricare două noduri din graful G . Fie d matricea distanțelor obținută la terminarea algoritmului.

Ulterior, enumerăm toate *permutările* P ale celor K noduri, calculăm pentru fiecare distanță $d(1, P_1) + d(P_1, P_2) + \dots + d(P_{K-1}, P_K) + d(P_K, N)$ și alegem cea mai mică valoare, ce va fi răspunsul.

Complexitate finală: $O(N^3 + K!)$.

Soluție 20 de puncte

Se va calcula lungimea drumului minim de la nodul 1 la nodul N , utilizând orice algoritm de drum minim cunoscut.

9.2.2 Cod sursă

Listing 9.2.1: bellmanford.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <queue>
5 #include <bitset>
6 #include <cassert>
7 #include <memory.h>
8 #include <algorithm>
9
10 using namespace std;
11
12 const char iname[] = "ubuntzei.in";
13 const char oname[] = "ubuntzei.out";
14
15 const int MAX_N = 205;
16 const int MAX_K = 15;
17 const int INF = 0x3F3F3F3F;
18
19 int dist[MAX_N][1 << MAX_K];
20
21 bitset<(1 << MAX_K)> inque[MAX_N];
22 queue < pair <int, int> > que;
23
24 int rf[MAX_N][MAX_N];
25
26 vector <int> adj[MAX_N], subset, idxSubset; bitset <MAX_N> inSubset;
27
28 int main(void)
29 {
30     int nodeCount, edgeCount, subsetCount, u, v, cost;
31
32     ifstream in(iname);
33     in >> nodeCount >> edgeCount >> subsetCount;
34
35     subset.resize(subsetCount);
36     idxSubset.resize(nodeCount);
37     idxSubset.assign(idxSubset.size(), -1);
38     for (int i = 0; i < subsetCount; ++ i)
39     {
40         in >> u; -- u;
41         assert(0 < u && u < nodeCount - 1);
42         subset[i] = u;
43         idxSubset[u] = i;
44         inSubset[u] = true;
45     }
46
47     memset(rf, INF, sizeof rf);
48     for (int i = 0; i < nodeCount; ++ i)
49         rf[i][i] = 0;
50     for (int i = 0; i < edgeCount; ++ i)
51     {
52         in >> u >> v >> cost;
53         -- u, -- v;
54         assert(0 <= u && u < nodeCount);
55         assert(0 <= v && v < nodeCount);
56         rf[u][v] = rf[v][u] = cost;
57     }
58     in.close();
59

```

```

60     for (int k = 0; k < nodeCount; ++ k)
61     {
62         for (int i = 0; i < nodeCount; ++ i)
63         {
64             for (int j = 0; j < nodeCount; ++ j)
65             {
66                 rf[i][j] = min(rf[i][j], rf[i][k] + rf[k][j]);
67             }
68         }
69     }
70
71     if (subsetCount > 0)
72     {
73         for (int i = 0; i < nodeCount; ++ i)
74         {
75             for (int j = i + 1; j < nodeCount; ++ j)
76             {
77                 assert(rf[i][j] < INF);
78                 if ((inSubset[i] || !i || i == nodeCount - 1) &&
79                     (inSubset[j] || !j || j == nodeCount - 1))
78                 {
80                     adj[i].push_back(j);
81                     adj[j].push_back(i);
82                 }
83             }
84         }
85     }
86
87     memset(dist, INF, sizeof dist);
88     dist[0][0] = 0;
89     que.push(make_pair(0, 0)), inque[0][0] = true;
90
91     while (!que.empty())
92     {
93         int u = que.front().first;
94         int s = que.front().second;
95         que.pop();
96         inque[u][s] = false;
97
98         for (vector<int>::iterator it = adj[u].begin();
99              it != adj[u].end();
100             ++ it)
101         {
102             int v = *it;
103
104             if (!inSubset[v])
105             {
106                 if (dist[v][s] > dist[u][s] + rf[u][v])
107                 {
108                     dist[v][s] = dist[u][s] + rf[u][v];
109                     if (inque[v][s] == false)
110                     {
111                         que.push(make_pair(v, s));
112                         inque[v][s] = true;
113                     }
114                 }
115             }
116             else if (!(s & (1 << idxSubset[v])))
117             {
118                 int t = s | (1 << idxSubset[v]);
119
120                 if (dist[v][t] > dist[u][s] + rf[u][v])
121                 {
122                     dist[v][t] = dist[u][s] + rf[u][v];
123                     if (inque[v][t] == false)
124                     {
125                         que.push(make_pair(v, t));
126                         inque[v][t] = true;
127                     }
128                 }
129             }
130         }
131     }
132
133     int sink = nodeCount - 1;
134     int conf = (1 << subset.size()) - 1;
135     assert(dist[sink][conf] < INF);

```

```

136         ofstream( fname ) << dist[ sink ][ conf ] << "\n";
137     }
138     else
139         ofstream( fname ) << rf[ 0 ][ nodeCount - 1 ] << "\n";
140
141     return 0;
142 }

```

Listing 9.2.2: dijkstra.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <queue>
5 #include <bitset>
6 #include <set>
7 #include <cassert>
8 #include <memory.h>
9 #include <algorithm>
10
11 using namespace std;
12
13 const char iname[] = "ubuntzei.in";
14 const char fname[] = "ubuntzei.out";
15
16 const int MAX_N = 10005;
17 const int MAX_K = 17;
18 const int INF = 0x3F3F3F3F;
19
20 vector < pair < int, int > > adj[MAX_N], adjK[MAX_K];
21 int dist[MAX_K][1 << MAX_K];
22
23 priority_queue < pair < int, int > > que;
24
25 vector < int > go_dijkstra(int src,
26                           int nodeCount,
27                           vector < pair < int, int > >* adj)
28 {
29     vector < int > d(nodeCount, INF);
30     vector < pair < int, int > ::iterator it;
31
32     d[src] = 0;
33     for (que.push(make_pair(-d[src], src)); !que.empty(); )
34     {
35         int u = que.top().second;
36         int dq = -que.top().first;
37         que.pop();
38         if (d[u] < dq) continue ;
39
40         for (it = adj[u].begin(); it != adj[u].end(); ++ it)
41         {
42             int v = ( *it ).first;
43             int cost = ( *it ).second;
44             if (d[v] > d[u] + cost)
45             {
46                 d[v] = d[u] + cost;
47                 que.push(make_pair(-d[v], v));
48             }
49         }
50     }
51     return d;
52 }
53
54 priority_queue < pair < int, pair < int, int > > > queK;
55
56 int go_dijkstraK(int src, int sink, int nodeCountK,
57                   vector < pair < int, int > >* adjK)
58 {
59     memset(dist, INF, sizeof dist);
60
61     dist[src][1 << src] = 0;
62     for (queK.push(make_pair(0, make_pair(src, 1 << src))); !queK.empty(); )
63     {
64         int u = queK.top().second.first;

```

```

65         int s = queK.top().second.second;
66         int d = -queK.top().first;
67         queK.pop();
68         if (dist[u][s] < d)  continue;
69
70         for (vector < pair <int, int> >::iterator it = adjK[u].begin();
71             it != adjK[u].end();
72             ++ it)
73     {
74         int v = ( *it ).first;
75         int cost = ( *it ).second;
76
77         if (!(s & (1 << v)))
78     {
79             int t = s | (1 << v);
80             if (dist[v][t] > dist[u][s] + cost)
81             {
82                 dist[v][t] = dist[u][s] + cost;
83                 queK.push(make_pair(-dist[v][t], make_pair(v, t)));
84             }
85         }
86     }
87 }
88
89 return dist[sink][(1 << nodeCountK) - 1];
90 }
91
92 int main(void)
93 {
94     int nodeCount, edgeCount, subsetCount, u, v, cost;
95
96     ifstream in(iname);
97     in >> nodeCount >> edgeCount >> subsetCount;
98     int src = 0;
99     int sink = nodeCount - 1;
100    vector <int> subset(subsetCount), idxSubset(nodeCount);
101    bitset <MAX_N> inSubset;
102
103    for (int i = 0; i < subsetCount; ++ i)
104    {
105        in >> u; -- u;
106        assert(0 < u && u < nodeCount - 1);
107        idxSubset[subset[i] = u] = i;
108        inSubset[u] = true;
109    }
110
111    subset.push_back(src),
112    idxSubset[src] = (int) subset.size() - 1,
113    inSubset[src] = true;
114
115    subset.push_back(sink),
116    idxSubset[sink] = (int) subset.size() - 1,
117    inSubset[sink] = true;
118
119    for (int i = 0; i < edgeCount; ++ i)
120    {
121        in >> u >> v >> cost;
122        -- u, -- v;
123        assert(0 <= u && u < nodeCount);
124        assert(0 <= v && v < nodeCount);
125
126        adj[u].push_back(make_pair(v, cost));
127        adj[v].push_back(make_pair(u, cost));
128    }
129
130    in.close();
131
132    vector <int> d;
133
134    d = go_dijkstra(src, nodeCount, adj);
135    for (size_t i = 0; i < d.size(); ++ i) if (inSubset[i])
136    {
137        assert(d[i] < INF);
138        if ((int) i != src)
139            adjK[ idxSubset[src] ].push_back(make_pair(idxSubset[i], d[i]));
140    }

```

```

141
142     if (subsetCount > 0)
143     {
144         for (size_t i = 0; i < subset.size(); ++ i)
145             if (subset[i] != src && subset[i] != sink)
146             {
147                 d = go_dijkstra(subset[i], nodeCount, adj);
148                 for (size_t j = 0; j < d.size(); ++ j) if (inSubset[j])
149                 {
150                     assert(d[j] < INF);
151                     if (subset[i] != (int) j)
152                         adjK[i].push_back(make_pair(idxSubset[j], d[j]));
153                 }
154             }
155
156         int answer = go_dijkstraK(idxSubset[src],
157                                     idxSubset[sink],
158                                     subset.size(),
159                                     adjK);
160         assert(answer < INF);
161         ofstream(pname) << answer << "\n";
162     }
163     else
164         ofstream(pname) << d[nodeCount - 1] << "\n";
165
166     return 0;
167 }
```

Listing 9.2.3: dp.cpp

```

1 // Filip Buruiana
2 #include <stdio.h>
3 #include <vector>
4 #include <set>
5
6 using namespace std;
7
8 #define minim(a, b) ((a < b) ? a : b)
9 #define INF 999999999
10 #define KMAX 15
11 #define NMAX 10005
12 #define pii pair<int, int>
13 #define f first
14 #define s second
15 #define mp make_pair
16 #define pb push_back
17
18 int N, M, K, C[KMAX];
19 int source[NMAX], path[KMAX][NMAX];
20 vector<pii> G[NMAX];
21 int pd[1<<KMAX][KMAX];
22
23 set<pii> q;
24
25 inline int bit(int x, int nr)
26 { return (x & (1<<nr)) != 0; }
27
28 void dijkstra(int sursa, int *sol)
29 {
30     int i, vec;
31     for (i = 1; i <= N; ++i)
32         sol[i] = INF;
33
34     sol[sursa] = 0;
35     q.clear();
36
37     for (i = 1; i <= N; ++i)
38         q.insert(mp(sol[i], i));
39
40     for (i = 1; i < N; ++i)
41     {
42         pii top = *q.begin();
43         q.erase(q.begin());
44         int nod = top.s;
45         if (sol[nod] < top.f) continue;
```

```

46     for (size_t j = 0; j < G[nod].size(); ++j)
47     {
48         if (sol[vec] = G[nod][j].f) > sol[nod] + G[nod][j].s)
49         {
50             sol[vec] = sol[nod] + G[nod][j].s;
51             q.insert(mp(sol[vec], vec));
52         }
53     }
54
55 int main()
56 {
57     int i, j, k, newCost;
58
59     freopen("ubuntzei.in", "r", stdin);
60     freopen("ubuntzei.out", "w", stdout);
61
62     scanf("%d %d %d", &N, &M, &K);
63     for (i = 0; i < K; ++i)
64         scanf("%d", &C[i]);
65
66     for ( ; M; --M)
67     {
68         scanf("%d %d %d", &i, &j, &k);
69         G[i].pb(mp(j,k));
70         G[j].pb(mp(i,k));
71     }
72
73     dijkstra(1, source);
74     if (K == 0)
75     {
76         printf("%d\n", source[N]);
77         return 0;
78     }
79
80     for (i = 0; i < K; ++i)
81         dijkstra(C[i], path[i]);
82
83     for (i = 1; i < (1<<K); ++i)
84     {
85         for (j = 0; j < K; ++j)
86             if (i == (1<<j))
87                 break;
88             if (j < K && i == (1<<j))
89             {
90                 pd[i][j] = source[C[j]];
91                 continue;
92             }
93
94             for (j = 0; j < K; ++j)
95             {
96                 pd[i][j] = INF;
97                 if (bit(i,j))
98                     for (k = 0; k < K; ++k)
99                         if (k != j && bit(i, k))
100                             {
101                                 newCost = pd[i-(1<<j)][k] + path[k][C[j]];
102                                 pd[i][j] = minim(pd[i][j], newCost);
103                             }
104             }
105     }
106
107     bst = INF;
108     for (i = 0; i < K; ++i)
109         bst = minim(bst, pd[(1<<K)-1][i] + path[i][N]);
110
111     printf("%d\n", bst);
112
113     return 0;
114 }
```

Listing 9.2.4: royfloyd.cpp

```

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

```

4 #include <queue>
5 #include <bitset>
6 #include <cassert>
7 #include <memory.h>
8 #include <algorithm>
9
10 using namespace std;
11
12 const char iname[] = "ubuntzei.in";
13 const char oname[] = "ubuntzei.out";
14
15 const int MAX_N = 205;
16 const int INF = 0x3F3F3F3F;
17
18 vector <int> subset;
19 int d[MAX_N][MAX_N];
20
21 int main(void)
22 {
23     int nodeCount, edgeCount, subsetCount, u, v, cost;
24
25     ifstream in(iname);
26     in >> nodeCount >> edgeCount >> subsetCount;
27     subset.resize(subsetCount);
28     for (int i = 0; i < subsetCount; ++ i)
29     {
30         in >> subset[i];
31         subset[i] --;
32     }
33
34     memset(d, INF, sizeof d);
35
36     for (int i = 0; i < nodeCount; ++ i)
37         d[i][i] = 0;
38     for (int i = 0; i < edgeCount; ++ i)
39     {
40         in >> u >> v >> cost;
41         -- u, -- v;
42         d[u][v] = d[v][u] = cost;
43     }
44     in.close();
45
46     for (int k = 0; k < nodeCount; ++ k)
47     {
48         for (int i = 0; i < nodeCount; ++ i)
49         {
50             for (int j = 0; j < nodeCount; ++ j)
51                 d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
52         }
53     }
54
55     if (subsetCount > 0)
56     {
57         sort(subset.begin(), subset.end());
58         int src = 0, sink = nodeCount - 1;
59         int answer = INF;
60         do {
61             int temp = d[src][subset.front()] + d[subset.back()][sink];
62
63             for (int i = 0; i < (int) subset.size() - 1; ++ i)
64                 temp += d[subset[i]][subset[i + 1]];
65
66             answer = min(answer, temp);
67         } while (next_permutation(subset.begin(), subset.end()));
68
69         ofstream(oname) << answer << "\n";
70     }
71     else
72         ofstream(oname) << d[0][nodeCount - 1] << "\n";
73
74     return 0;
75 }
```

9.2.3 *Rezolvare detaliată

Capitolul 10

OJI 2010

10.1 immortal

Problema 1 - immortal

100 de puncte

Cei care au văzut filmul Nemuritorul, știu că fraza cu care nemuritorii încep lupta este "Nu poate să rămână decât unul singur". Să încercăm să simulăm povestea nemuritorilor.

Într-o zonă dreptunghiulară formată din n linii (numerotate de la 1 la n) și m coloane (numerotate de la 1 la m) se află maxim $n \times m - 1$ nemuritori.

Doi nemuritori vecini se "luptă" între ei și cel care pierde lupta este eliminat. "Luptă" constă în săritura unuia dintre nemuritori peste celălalt, dacă această săritură se poate face. Săritura se poate face pe orizontală sau verticală și nemuritorul peste care s-a sărit dispare.

Prin vecin al nemuritorului din poziția (i, j) înțelegem un nemuritor din una dintre pozițiile $(i - 1, j)$, $(i + 1, j)$, $(i, j - 1)$, $(i, j + 1)$. Deci, după luptă nemuritorul din câmpul (i, j) se va găsi în una dintre pozițiile: $(i - 2, j)$, $(i + 2, j)$, $(i, j - 2)$ sau $(i, j + 2)$, dacă această poziție este liberă și este în interiorul zonei.

Cerințe

Se cere să se determine o succesiune a luptelor ce pot fi purtate, astfel încât la final să rămână un singur nemuritor.

Date de intrare

Fișierul de intrare **immortal.in** conține pe prima linie trei valori naturale n m I , separate prin câte un spațiu, reprezentând numărul de linii, numărul de coloane ale zonei descrise și respectiv numărul de nemuritori existenți inițial.

Următoarele I linii conțin fiecare câte două numere naturale x y separate printr-un spațiu, reprezentând pozițiile unde se găsesc inițial cei I nemuritori (linia și coloana).

Date de ieșire

Fișierul de ieșire **immortal.out** va conține $I - 1$ linii, fiecare linie descriind o "luptă". Luptele vor fi scrise în ordinea în care au avut loc.

O linie va conține 4 numere naturale care indică: primele două poziția de pe care pleacă un nemuritor la "luptă", ultimele două poziția pe care acesta ajunge după "luptă".

Pentru ca "luptă" să fie corectă, în poziția peste care nemuritorul "sare" trebuie să existe un nemuritor care va "muri". O poziție va fi specificată prin indicele de linie urmat de indicele de coloană.

Valorile scrise pe aceeași linie vor fi separate prin spații.

Restricții și precizări

$$1 < n, m \leq 20$$

$$1 < I \leq \min\{15, n * m - 1\}$$

Pentru datele de test există întotdeauna soluție.

Exemple

immortal.in	immortal.out	Explicații
3 4 4	3 3 3 1	
1 2	3 1 1 1	1 2 3 4 =====
2 1	1 1 1 3	(3,3) --> (3,1) dispare (3,2) (3,1) --> (1,1) dispare (2,1) (1,1) --> (1,3) dispare (1,2)
3 2		
3 3		

Timp maxim de executare/test: 0.5 secunde

Memorie: total 2 MB din care pentru stivă 1 MB

Dimensiune maximă a sursei: 20 KB

10.1.1 Indicații de rezolvare

autor: ??? - ???

Problemă clasică de *backtracking*.

Se pot obține puncte și abordând problema cu o strategie *greedy*: se parcurge zona plecând de la (1, 1) și se alege prima luptă posibilă. După consumarea ei se reia parcurgerea zonei (deoarece este posibil să apară o nouă luptă într-o poziție anterioară precedentei lupte) și se caută o nouă luptă. Dacă după $NrNemuritori - 1$ pași a rămas un nemuritor, înseamnă că luptele au condus la soluție.

Perfecționarea strategiei anterioare se poate realiza prin parcurgerea zonei de luptă în sens contrar (de la (n, n) la $(1, 1)$) în cazul în care parcurgerea de la $(1, 1)$ la (n, n) nu a dus la soluție - deci se fac 2 verificări. Evident pot exista și alte modalități de parcurgere a zonei de luptă.

Pentru testele date soluția *backtracking* ia 100 puncte, prima strategie *greedy* obține 10 puncte, iar a doua 30 puncte.

10.1.2 Cod sursă

Listing 10.1.1: immortalc.c

```

1 //Emanuela Cerchez
2 //timp maxim: 0.17
3 #include <stdio.h>
4
5 #define InFile "immortal.in"
6 #define OutFile "immortal.out"
7 #define NMax 54
8 #define LgMax NMax * NMax
9
10 int T[NMax][NMax];
11 int n, m, nr;
12 struct Poz {int x, y;} P[LgMax];
13 struct Lupta {struct Poz v, m;} sol[LgMax];
14
15 int ok=0;
16 int mort[LgMax];
17 int dx[]={-1, 0, 1, 0};
18 int dy[]={0, 1, 0, -1};
19
20 void citire(void);
21 void bkt(int);
22 void afisare(void);
23
24 int main()
25 {
26     citire();
27     bkt(0);
28     return 0;

```

```

29 }
30
31 void citire()
32 {
33 int i, j, x, y, k=0;
34 FILE * fin=fopen(InFile, "r");
35 fscanf(fin, "%d %d %d", &n, &m, &nr);
36 for (i=1; i<=nr; i++)
37     fscanf(fin, "%d %d", &x, &y);
38     T[x+1][y+1]=i;
39 for (i=2; i<=n+1; i++)
40     for (j=2; j<=m+1; j++)
41         if (T[i][j])
42             {P[++k].x=i; P[k].y=j; T[i][j]=k;}
43 for (i=0; i<n+4; i++) T[i][0]=T[i][1]=T[i][m+3]=T[i][m+2]=-1;
44 for (i=0; i<m+4; i++) T[0][i]=T[1][i]=T[n+2][i]=T[n+3][i]=-1;
45 }
46
47 void bkt(int k)
48 //sau desfasurat k lupte sol[0], ..., sol[k-1]
49 {
50 int i, d, x, y, moare;
51 if (!ok)
52     if (k==nr-1)
53         {ok=1; afisare();}
54     else
55         for (i=1; i<=nr; i++)
56             if (!mort[i])
57             {
58                 //poate sari i?
59                 x=P[i].x; y=P[i].y;
60                 for (d=0; d<4; d++)
61                     if (T[x+dx[d]][y+dy[d]]>0 && T[x+2*dx[d]][y+2*dy[d]]==0)
62                     {
63                         //un vecin peste care poate sari
64                         moare=T[x+dx[d]][y+dy[d]];
65                         sol[k].m.x=x+2*dx[d]; sol[k].m.y=y+2*dy[d];
66                         sol[k].v.x=x; sol[k].v.y=y;
67                         P[i].x=x+2*dx[d]; P[i].y=y+2*dy[d];
68                         T[x][y]=0; T[x+2*dx[d]][y+2*dy[d]]=i;
69                         mort[T[x+dx[d]][y+dy[d]]]=1;
70                         T[x+dx[d]][y+dy[d]]=0;
71
72                         bkt(k+1);
73
74                         P[i].x=x; P[i].y=y;
75                         T[x][y]=i; T[x+2*dx[d]][y+2*dy[d]]=0;
76                         T[x+dx[d]][y+dy[d]]=moare;
77                         mort[T[x+dx[d]][y+dy[d]]]=0;
78
79                     }
80             }
81 }
82
83 void afisare()
84 {
85 int i;
86 FILE * fout=fopen(OutFile, "w");
87 for (i=0; i<nr-1; i++)
88     fprintf(fout, "%d %d %d %d\n",
89             sol[i].v.x-1,sol[i].v.y-1,sol[i].m.x-1,sol[i].m.y-1);
90 fclose(fout);
91 }

```

Listing 10.1.2: immortal.cpp

```

1 //Nistor Mot
2 //Timp maxim: 17 secunde
3 #include <stdio.h>
4 #define M 23
5
6 /* backtracking neoptimizat */
7 int T[M][M], I, x[M], v[M], px[M], py[M], mx[M], my[M], sens[M], sf;
8
9 void scrie()

```

```

10 {FILE *f;
11 int i;
12 f=fopen("immortal.out","wt");
13 if(I==1) {fprintf(f,"0\n"); fclose(f); return;}
14 for(i=1;i<I;i++)
15 {fprintf(f,"%d %d ",mx[i],my[i]);
16 switch(sens[i])
17 {case 1: my[i]+=2; break;
18 case 2: my[i]-=2; break;
19 case 3: mx[i]+=2; break;
20 case 4: mx[i]-=2; break;}
21 fprintf(f,"%d %d\n",mx[i],my[i]);}
22 fclose(f);
23 return;}
24
25 void back(int k)
26 {int i,X,Y,z;
27 if(sf) return;
28 if(k==I) {sf=1; scrie(); return;}
29 for(i=1;i<=I;i++) if(v[i]==0)
30 {X=px[i]; Y=py[i];
31 if((z=T[X][Y-1])>0 && T[X][Y+1]==0)
32 {v[i]=1; sens[k]=1; mx[k]=X; my[k]=Y-1; py[z]+=2;
33 T[X][Y]=0; T[X][Y-1]=0; T[X][Y+1]=z; back(k+1);
34 v[i]=0; py[z]-=2; T[X][Y]=i; T[X][Y+1]=0; T[X][Y-1]=z;}
35
36 if((z=T[X][Y+1])>0 && T[X][Y-1]==0)
37 {v[i]=1; sens[k]=2; mx[k]=X; my[k]=Y+1; py[z]-=2; T[X][Y]=0;
38 T[X][Y+1]=0; T[X][Y-1]=z; back(k+1);
39 v[i]=0; py[z]+=2; T[X][Y]=i; T[X][Y-1]=0; T[X][Y+1]=z;}
40
41 if((z=T[X-1][Y])>0 && T[X+1][Y]==0)
42 {v[i]=1; sens[k]=3; mx[k]=X-1; my[k]=Y; px[z]+=2; T[X][Y]=0;
43 T[X-1][Y]=0; T[X+1][Y]=z; back(k+1);
44 v[i]=0; px[z]-=2; T[X][Y]=i; T[X+1][Y]=0; T[X-1][Y]=z;}
45
46 if((z=T[X+1][Y])>0 && T[X-1][Y]==0)
47 {v[i]=1; sens[k]=4; mx[k]=X+1; my[k]=Y; px[z]-=2; T[X][Y]=0;
48 T[X+1][Y]=0; T[X-1][Y]=z; back(k+1);
49 v[i]=0; px[z]+=2; T[X][Y]=i; T[X-1][Y]=0; T[X+1][Y]=z;}
50 }
51 }
52
53 int main()
54 {FILE *f;
55 int i,j,k,n,m;
56 f=fopen("immortal.in","r");
57 fscanf(f,"%d %d %d",&n,&m,&I);
58 for(i=0;i<=n+1;i++) T[i][0]=T[i][m+1]=-1;
59 for(i=1;i<=m;i++) T[0][i]=T[n+1][i]=-1;
60 for(i=1;i<=I;i++)
61 {fscanf(f,"%d %d",&j,&k); T[j][k]=i; px[i]=j; py[i]=k;}
62 fclose(f);
63 back(1);
64 return 0;
65 }

```

10.1.3 *Rezolvare detaliată

10.2 joc

Problema 2 - joc

100 de puncte

Jocul nostru presupune parcurgerea unui tablou bidimensional cu două linii și n coloane, format din $2 \times n$ celule pătratice.

Fiecare celulă are asociată câte o valoare întreagă v care nu se modifică pe durata desfășurării jocului. Jucătorii trebuie să găsească un drum de la celula de plecare la celula de sosire care respectă următoarele condiții:

- celula de plecare este cea din linia 1 și coloana 1, iar celula de sosire este cea din linia 2 și coloana n .

- nu trebuie decât cel mult odată prin oricare celulă.

- deplasarea se poate face din celula curentă spre oricare altă celulă învecinată cu ea pe orizontală sau verticală.

- conține cel mult k celule consecutive aflate pe aceeași linie.

Pentru un astfel de drum se calculează punctajul acestuia ca fiind egal cu suma valorilor asociate celulelor prin care trece drumul.

Cerințe

Cunoscând valorile asociate celulelor tabloului, scrieți un program care determină punctajul maxim care poate fi obținut în acest joc.

Date de intrare

Fișierul de intrare **joc.in** va conține pe prima linie două numere naturale n și k separate printr-un spațiu cu semnificațiile din enunț. Pe fiecare dintre următoarele două linii se găsesc câte n numere întregi, reprezentând valorile asociate celor $2 \times n$ celule ale tabloului.

Date de ieșire

Fișierul de ieșire **joc.out** va conține pe prima linie numărul întreg p , reprezentând punctajul maxim care se poate obține.

Restricții și precizări

- $2 \leq n \leq 5000$
- $2 \leq k \leq 10, k \leq n$
- $-1000 \leq v \leq 1000$
- Pentru 40% dintre cazurile de test $n \leq 40$

Exemple

joc.in	joc.out	Explicații												
6 3 0 -2 5 4 -9 -1 -1 3 2 7 0 1	21	Jucătorul va parcurge în ordine celulele cu valorile: 0, -1, 3, 2, 5, 4, 7, 0, 1 <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>0</td><td>-2</td><td>5</td><td>4</td><td>-9</td><td>-1</td></tr> <tr> <td>0</td><td>-2</td><td>5</td><td>4</td><td>-9</td><td>-1</td></tr> </table>	0	-2	5	4	-9	-1	0	-2	5	4	-9	-1
0	-2	5	4	-9	-1									
0	-2	5	4	-9	-1									
5 5 0 0 4 2 10 2 -3 -8 6 -2	14	Jucătorul va parcurge în ordine celulele cu valorile: 0, 0, 4, 2, 10, -2 <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>0</td><td>0</td><td>4</td><td>2</td><td>10</td></tr> <tr> <td>2</td><td>-3</td><td>-8</td><td>6</td><td>-2</td></tr> </table>	0	0	4	2	10	2	-3	-8	6	-2		
0	0	4	2	10										
2	-3	-8	6	-2										
5 4 -3 0 5 4 10 -2 3 -2 7 0	22	Jucătorul va parcurge în ordine celulele cu valorile: -3,-2, 3, 0, 5, -2, 7, 4, 10, 0 <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>-3</td><td>0</td><td>5</td><td>4</td><td>10</td></tr> <tr> <td>2</td><td>3</td><td>-2</td><td>7</td><td>0</td></tr> </table>	-3	0	5	4	10	2	3	-2	7	0		
-3	0	5	4	10										
2	3	-2	7	0										

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

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

Dimensiune maximă a sursei: **20 KB**

10.2.1 Indicații de rezolvare

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

Observăm că starea curentă este caracterizată de parametrii (s, i, j, d, c) , unde i și j reprezintă linia, respectiv coloana, d este direcția din care s-a intrat în celula actuală, c este numărul de celule consecutive parcuse pe linia i până la coloana j , iar s este suma maximă obținută. Observând că o parcurgere exhaustivă a tuturor drumurilor duce inevitabil la trecerea repetată prin anumite stări, se va evita recalcularea acelorași valori prin *memorarea lor*.

Alternativele sunt *programare dinamică* sau o abordare *recursivă cu memoizare*. Pentru aceste tipuri de soluție se primesc 100 puncte. Complexitatea așteptată: $O(N * K)$

Soluții bazate pe *backtracking*, primesc între 20 și 40 de puncte, în funcție de optimizări.

10.2.2 Cod sursă

Listing 10.2.1: joc_back1.cpp

```

1  /*
2   * Backtracking - 30 puncte
3   * Constantin Galatan
4  */
5  #include <fstream>
6  #include <iostream>
7  #include <climits>
8  #include <algorithm>
9
10 using namespace std;
11
12 #define sus 0
13 #define dr 1
14 #define jos 2
15 ifstream fin("joc.in");
16 ofstream fout("joc.out");
17
18 const int di[] = { -1, 0, 1 },
19     dj[] = { 0, 1, 0 };
20
21 int a[2][5001];
22 int sol[2][5001];
23 int Smax = -99999;
24 int n;
25 int K, k;
26
27 void Read();
28 void Rec(int l, int c, int s);
29 int OK(int i, int j, int d);
30
31 int main()
32 {
33     Read();
34     k = 1;
35     Rec(0, 0, 0);
36     fout << Smax << '\n';
37     fout.close();
38 }
39
40 void Rec(int i, int j, int s)
41 {
42     int temp = k;
43     if ( i == 1 && j == n - 1 )
44     {
45         if ( s + a[1][n-1] > Smax )
46             Smax = s + a[1][n-1];
47         return;
48     }
49
50     int iv, jv;
51     sol[i][j] = 1;
52     for (int d = 0; d < 3; ++d )
53     {
54         iv = i + di[d];
55         jv = j + dj[d];
56         if ( OK(iv, jv, d) )
57         {
58             if ( d == 1 ) k++;
59             if ( d == 0 || d == 2 ) k = 1;
60             Rec(iv, jv, s + a[i][j]);
61             if ( d == 0 || d == 2 ) k = temp;
62         }
63     }
64     sol[i][j] = 0;
65 }
66
67 int OK(int i, int j, int d)
68 {
69     if ( d == 1 && k + 1 > K) return 0;
70     if ( i < 0 || i >= n || j < 0 || j >= n ) return 0;

```

```

71     if ( sol[i][j] == 1 ) return 0;
72     return 1;
73 }
74
75 void Read()
76 {
77     fin >> n >> K;
78     for ( int i = 0; i < 2; ++i )
79         for ( int j = 0; j < n; ++j )
80             fin >> a[i][j];
81     fin.close();
82 }
83 }
```

Listing 10.2.2: joc_dinamical.cpp

```

1 // prof. Nistor Mot
2 #include <stdio.h>
3 #define M 5001
4 /* dinamica O(n) */
5
6 int main()
7 {int n,k,i,j,j1,s,max,a[2][M],b[2][M];
8
9 FILE *f1,*f2;
10 f1=fopen("9-joc.in","r");
11 fscanf(f1,"%d %d",&n,&k);
12
13 for(i=0;i<2;i++)
14     for(j=0;j<n;j++) fscanf(f1,"%d",&a[i][j]);
15 fclose(f1);
16
17 b[0][0]=a[0][0]; b[1][0]=0;
18 for(i=1;i<n;i++)
19     for(j=0;j<2;j++)
20         { max=-11111; s=a[j][i];
21             for(j1=1;j1<k&&j1<=i;j1++)
22                 { s+=a[j][i-j1];
23                     if(s+b[1-j][i-j1]>max) max=s+b[1-j][i-j1];
24                 b[j][i]=max; }
25
26 f2=fopen("9-joc.out","wt");
27 if(b[0][n-1]+a[1][n-1]>b[1][n-1]) b[1][n-1]=b[0][n-1]+a[1][n-1];
28 fprintf(f2,"%d\n",b[1][n-1]);
29
30 fclose(f2);
31 return 0;
32 }
```

Listing 10.2.3: joc_dinamica2.cpp

```

1 //Code by Patcas Csaba
2 //Time complexity: O(n * k)
3 //Space complexity: O(n * k)
4 //Method: DP
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 FORI(it, X) for(__typeof((X).begin()) it = (X).begin(); it !=(X).end(); ++it)
45 #define REPEAT do{
46 #define UNTIL(x) }while(! (x));
47
48 #define SZ size()
49 #define BG begin()
50 #define EN end()
51 #define CL clear()
52 #define X first
53 #define Y second
54 #define RS resize
55 #define PB push_back
56 #define MP make_pair
57 #define ALL(x) x.begin(), x.end()
58
59 #define IN_FILE "9-joc.in"
60 #define OUT_FILE "joc.out"
61 ifstream fin(IN_FILE);
62 ofstream fout(OUT_FILE);
63
64 #define INF 10000001
65
66 int n, k;
67 VVI a;
68 vector <VVI> dp;
69
70 int Solve()
71 {
72     dp.RS(3, VVI(n + 1, VI(k + 1)));
73     dp[1][1][1] = a[1][1];
74     dp[2][1][1] = a[1][1] + a[2][1];
75     FOR(j, 2, n)
76     {
77         FOR(i, 1, 2)
78             FOR(q, 2, k)
79             {
80                 if (q > j) break;
81                 dp[i][j][q] = dp[i][j - 1][q - 1] + a[i][j];
82             }
83         FOR(i, 1, 2)
84         {
85             int maxVal = -INF;
86             FOR(q, 2, k)
87             {
88                 if (q > j) break;
89                 maxVal = max(maxVal, dp[3 - i][j][q]);
90             }
91             dp[i][j][1] = maxVal + a[i][j];
92         }
93     }
94     int ans = -INF;
95     FOR(q, 1, k) ans = max(ans, dp[2][n][q]);
96     return ans;
97 }
98
99 int main()

```

```

100 {
101     //Read data
102     fin >> n >> k;
103     a.RS(3, VI(n + 1));
104     FOR(i, 1, n) fin >> a[1][i];
105     FOR(i, 1, n) fin >> a[2][i];
106     fin.close();
107
108     //Write data
109     fout << Solve();
110     fout.close();
111
112     return 0;
113 }
```

Listing 10.2.4: joc_greedy.cpp

```

1 //Code by Patcas Csaba
2 //Time complexity: O(n)
3 //Space complexity: O(n)
4 //Method: Greedy
5 //Implementation time: 10 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 FORI(it, X) for(__typeof((X).begin()) it = (X).begin(); \
45                                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 "9-joc.in"
61 #define OUT_FILE "joc.out"
62 ifstream fin(IN_FILE);
63 ofstream fout(OUT_FILE);
64
65 int n, k;
66 VVI a;
67
68 int Solve()
69 {
70     int ans = 0, i = 1, j = 1, nCons = 1;
71     bool changeRow = false;
72     while (j < n)
73     {
74         ans += a[i][j];
75         if ((a[i][j + 1] > a[3 - i][j] || changeRow) && (nCons < k))
76         {
77             ++j;
78             ++nCons;
79             changeRow = false;
80         }
81         else
82         {
83             i = 3 - i;
84             nCons = 1;
85             changeRow = true;
86         }
87     }
88     ans += a[i][n];
89     if (i == 1) ans += a[2][n];
90     return ans;
91 }
92
93 int main()
94 {
95     //Read data
96     fin >> n >> k;
97     a.RS(3, VI(n + 1));
98     FOR(i, 1, n) fin >> a[1][i];
99     FOR(i, 1, n) fin >> a[2][i];
100    fin.close();
101
102    //Write data
103    fout << Solve();
104    fout.close();
105
106    return 0;
107 }
```

Listing 10.2.5: joc_memoizare.cpp

```

1 /*
2     Recursie cu memoizare
3     Complexitate: O(n * k)
4     Constantin Galatan
5 */
6 #include <iostream>
7 #include <climits>
8
9 using namespace std;
10
11 #define INF INT_MAX / 2
12 #define sus 0
13 #define dr 1
14 #define jos 2
15
16 ifstream fin("9-joc.in");
17 ofstream fout("joc.out");
18
19 int a[2][5001];
20 int n;
21 int K;
22 int s[2][5001][3][11]; // s[1][c][dir][k] - suma maxima pana la linia l,
23 // coloana c, daca s-a intrat in
```

```

24         // celula din directia dir, cu k celule consecutive pe linia l
25 void Read();
26 int Sum(int l, int c, int d, int k);
27 int max(int a, int b);
28
29 int main()
30 {
31     Read();
32
33     fout << Sum(l, n - 1, dr, 1) << '\n';
34
35     fout.close();
36     return 0;
37 }
38
39 int Sum(int l, int c, int dir, int k)
40 {
41     int& ret = s[l][c][dir][k];      // ret - alias pentru s[l][c][dir][k];
42
43     if (ret != -INF) // Daca se trece printr-o stare anterior calculata
44         return ret;           // se returneaza valoarea din matrice
45
46     if (l == 0 && c == 0) return ret = a[0][0];
47     if (l == 1 && c == 0) return ret = a[l][c] + a[0][0];
48
49     // Suma maxima se calculeaza in functie de linia curenta,
50     // directia din care s-a intrat in celula
51     // si numarul de celule consecutive parcuse pe linie
52     int r(0);
53     if (l == 1)
54     {
55         if (dir == dr)
56             if (k < K)
57                 r = max(Sum(l, c - 1, dr, k + 1), Sum(l - 1, c, jos, 1));
58             else
59                 r = Sum(l - 1, c, jos, 1);
60
61         if (dir == sus)
62             r = Sum(l, c - 1, dr, k + 1);
63     }
64
65     if (l == 0)
66     {
67         if (dir == dr)
68             if (k < K)
69                 r = max(Sum(l, c - 1, dr, k + 1), Sum(l + 1, c, sus, 1));
70             else
71                 r = Sum(l + 1, c, sus, 1);
72
73         if (dir == jos) r = Sum(l, c - 1, dr, k + 1);
74     }
75
76     return ret = r + a[l][c];
77 }
78
79 int max(int a, int b)
80 {
81     if (a >= b) return a;
82     return b;
83 }
84
85 void Read()
86 {
87     int i(0), j(0);
88     fin >> n >> K;
89     for (i = 0; i < 2; ++i)
90         for (j = 0; j < n; ++j)
91             fin >> a[i][j];
92
93     for (i = 0; i < 2; ++i)
94         for (j = 0; j < n; ++j)
95             for (int d = 0; d < 3; ++d)
96                 for (int k = 0; k <= K; ++k)
97                     s[i][j][d][k] = -INF;
98
99     fin.close();

```

100 }

10.2.3 *Rezolvare detaliată

Capitolul 11

OJI 2009

11.1 cerc

Problema 1 - cerc

100 de puncte

Se desenează n cercuri distincte în plan, numerotate cu numerele de la 1 la n . Pentru fiecare cerc k ($k \in \{1, 2, \dots, n\}$) se cunosc: raza cercului, r_k , și coordonatele (x_k, y_k) ale centrului cercului, coordonate referitoare la reperul cartezian xOy cu originea în punctul O din plan. Din punctul O , se desenează m drepte distincte, astfel încât pentru fiecare dreaptă, dintre cele m desenate, să existe cel puțin un cerc, dintre cele n , al cărui centru să fie situat pe această dreaptă și pentru fiecare cerc desenat, să existe o singură dreaptă, dintre cele m desenate, care să treacă prin centrul lui.

Cerințe

Să se scrie un program care să determine:

- numărul m de drepte distincte;
- cel mai mare număr q de cercuri, dintre cele n , exterioare două câte două, ale căror centre sunt situate pe o aceeași dreaptă care trece prin punctul O , dintre cele m desenate;
- numărul p al dreptelor distincte, dintre cele m desenate, pe care sunt situate centrele a câte q cercuri, dintre cele n , exterioare două câte două.

Date de intrare

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

n	- pe prima linie, o valoare naturală nenulă n , reprezentând numărul de cercuri
$x_1 y_1 r_1$	- următoarele n linii conțin câte trei numere naturale nenule, separate prin câte un
.....	spațiu, care reprezintă coordonatele centrului (x_1, y_1) și raza r_1 ale primului cerc, ...,
$x_n y_n r_n$	coordonatele centrului (x_n, y_n) și raza r_n ale celui de-al n -lea cerc

Date de ieșire

Fișierul de ieșire **cerc.out** va conține o singură linie pe care se vor scrie cele trei numere naturale m , q și p , separate prin câte un spațiu.

Restricții și precizări

- $1 \leq n \leq 2000$; $1 \leq x_1, x_2, \dots, x_n \leq 1000$; $1 \leq y_1, y_2, \dots, y_n \leq 1000$; $1 \leq r_1, r_2, \dots, r_n \leq 70$
- dacă două cercuri, dintre cele n , au centrele în același punct din plan, atunci razele lor sunt distincte
 - două cercuri sunt exterioare dacă nu au niciun punct comun și nici interioarele lor nu au puncte comune
 - Pentru rezolvarea cerinței a) se acordă 20% din punctaj, pentru cerința b) 50% din punctaj și pentru cerința c) 30% din punctaj.

Exemple

cerc.in	cerc.out	Explicații
12	4 3 2	Sunt $m = 4$ drepte distincte care conțin centrele celor 12 cercuri. Dreapta d_1 trece printr-un singur centru de cerc, d_4 trece prin 2 centre de cercuri exterioare.
2 6 1		Dreptele d_2 și d_3 trec prin câte 3 centre de cercuri exterioare.
3 9 1		Numărul maxim de cercuri exterioare două căte două este $q = 3$ iar centrele lor sunt situate pe d_2 sau pe d_3 ($p = 2$).
4 12 3		
4 4 2		
9 9 2		
10 10 6		
12 12 1		
6 3 1		
10 5 1		
14 7 2		
14 7 1		
12 4 2		

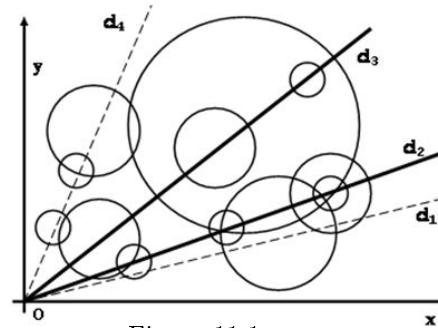


Figura 11.1: cerc

Timp maxim de executare/test: 1.0 secunde

11.1.1 Indicații de rezolvare

1. Descrierea soluției

prof. Constantin Gălățan (sursa cerct.cpp - 100p)

Cercurile situate cu centrele pe o aceeași dreaptă determină un *șir de intervale închise*. Se rețin coordonatele centrelor cercurilor în două siruri x și y de numere reale. Se sortează cercurile crescător după unghiul pe care îl fac cu axa Ox , iar pentru unghiuri egale, se sortează crescător după capetele din dreapta ale intervalelor. Apoi, pentru fiecare dreaptă distinctă (corespunzătoare unei secvențe consecutive în tablourile x și y), se aplică o metodă de tip *greedy* asemănătoare *problemelor spectacolelor*. Complexitate $O(n * \log n)$.

2. Descrierea soluției

drd.Pătcaș Csaba (sursa cerc.pas - 100p)

Două cercuri vor avea centrul pe aceeași dreaptă, dacă tangentă unghiului față de Ox a dreptelor determinate de origine și centrul cercurilor este egală. Pentru a evita problemele cu precizia numerelor reale, vom *reține tangentele în forma unor fracții ireducibile*. Determinarea dreptelor se poate face în timp $O(m * n)$.

Dacă pentru fiecare dreaptă considerăm numai punctele de intersecție cu cercurile a căror centru se află pe dreaptă, problema devine echivalentă cu *determinarea unui set independent maximal de intervale*, numit și *problemă spectacolelor*: sortăm intervalele crescător după capătul din dreapta, după care printr-o parcurgere a sirului sortat determinăm soluția. Complexitatea acestui pas depinde de complexitatea sortării, deci se poate face cu ușurință în $O(n * \log n)$ folosind un algoritm clasic de sortare, cum ar fi *QuickSort*.

Deoarece acest pas se repetă pentru fiecare dreaptă în parte, complexitatea totală va fi $O(m * n * \log n)$

3. Descrierea soluției

prof. Carmen Mincă

(sursele cerc_c.cpp, cerc_p.pas, cerc100.cpp cerc_g.cpp - 100p)

Se pot construi soluții cu complexitate $O(n * n)$ sau $O(n * \log n)$, aplicând *metoda programării dinamice* sau *metoda Greedy*.

O soluție obținută prin aplicarea *metodei programării dinamice*, *metoda înainte*, poate utiliza un tablou c unidimensional de tip înregistrare declarat astfel:

```
struct cerc { int x,y,r,d,e; } c[500];
```

pentru a memora în $c[k]$ ($k = 1, 2, \dots, n$) toate informațiile referitoare la cercul k (coordonatele x și y , raza r). Câmpul d al structurii va memora numărul dreptei, dintre cele m , care trece prin centru cercului k . Câmpul e va memora numărul cercurilor cu centrele pe dreapta d , exterioare două căte două, și cu abscisa centrului mai mare decât abscisa cercului k .

Se poate utiliza un tablou unidimensional de tip întreg v în care $v[k]$ va memora numărul cercurilor ale căror centre sunt situate pe dreapta k și sunt exterioare două căte două. Numărul valorilor distincte din tabloul v este egal cu numărul m de drepte distincte cerute.

Se sortează tabloul c , crescător după valorile câmpului x (abscisele centrelor cercurilor) și după tangentă unghiului dintre axa Ox și dreapta care trece prin O și centrele cercurilor prin *Qsort*.

Plecând de la primul cerc, stabilim care dintre cercurile memorate în c la dreapta celui curent, au centru situat pe dreapta care trece prin centrul primului cerc. Două cercuri, de raze r_1 respectiv r_2 , au centrele (x_1, y_1) și (x_2, y_2) situate pe aceeași dreaptă d care trece prin punctul O dacă este satisfăcută relația:

$$y_2 = \frac{y_1}{x_1} \cdot x_2 \Leftrightarrow y_2 \cdot x_1 = y_1 \cdot x_2$$

Pentru toate aceste cercuri se va memora în câmpul d același număr care reprezintă numărul dreptei curente (pentru primul cerc, numărul este 1). Se procedează analog pentru toate cercurile care nu au inițializat câmpul d . La final, fiecare $c[k].d$ va memora un număr natural nenul cel mult egal cu m .

Două cercuri, cu centrele (x_1, y_1) și (x_2, y_2) , și razele r_1 , respectiv r_2 , sunt *exterioare* dacă este satisfăcută relația:

$$\text{dist}((x_1, y_1), (x_2, y_2)) > r_1 + r_2 \Leftrightarrow (x_1 - x_2)^2 + (y_1 - y_2)^2 > (r_1 + r_2)^2$$

Începând cu penultimul cerc, se determină pentru fiecare cerc i numărul maxim al cercurilor j ($j > i$) cu centrele pe dreapta d care conține și centrul cercului i și sunt exterioare două câte două. Acest număr se va memora în $c[i].e$ (inițial $c[i].e = 1$, $i = 1, 2, \dots, n$):

Cel mai mare număr memorat în câmpul e al tabloului c reprezintă numărul q maxim de cercuri exterioare două câte două ale căror centre aparțin aceleiași drepte care trece prin punctul O . Numărul componentelor tabloului c care memorează în câmpul e valoarea q reprezintă numărul p al dreptelor care conțin centrele a câte q cercuri exterioare două câte două. (sursele cerc_c.cpp și cerc_p.pas obțin 100p).

O altă soluție se poate obține reducând cercurile la intervalele închise rezultate prin intersecțarea cerurilor cu dreptele pe care se află centrele lor, obținându-se pentru fiecare dreaptă un *șir de intervale închise*. Problema se reduce la a determina numărul maxim de intervale disjuncte două câte două pentru fiecare dreaptă. Cu o abordare gen "problema subșirului maximal", se obține punctajul maxim (sursa cerc100.cpp). O determinare prin *Greedy*, abordare gen "problema spectacolelor", obține punctajul maxim (sursa cerc_g.cpp).

11.1.2 *Cod sursă

11.1.3 *Rezolvare detaliată

11.2 Project management

Problema 2 - Project management

100 de puncte

La o firmă de software se lucrează la un mare proiect. Proiectul constă în executarea a n ($n \in \mathbb{N}$) faze de dezvoltare, numerotate cu numerele $1, 2, \dots, n$. Unele faze pot fi executate în paralel (în același timp), însă executarea altor faze nu poate fi începută până când nu se finalizează executarea anumitor faze.

Cerințe

Să se scrie un program care să se determine:

- a) timpul minim t în care se poate finaliza executarea proiectului
- b) pentru fiecare fază k ($k \in \{1, 2, \dots, n\}$), momentul de timp c_k la care poate începe fază k cel mai devreme, respectiv momentul de timp d_k la care poate începe fază k cel mai târziu, fără a influența durata totală de executare a proiectului.

Date de intrare

Fisierul de intrare **pm.in** conține:

- pe prima linie, un număr natural n , reprezentând numărul fazelor proiectului
- pe a doua linie, n numere naturale, separate prin câte un spațiu, reprezentând timpul necesar finalizării fiecărei faze
- pe fiecare linie k dintre următoarele n linii, un număr natural m_k și un sir a format din m_k numere naturale: a_1, a_2, \dots, a_{m_k} , cele $m_k + 1$ numere din linie fiind separate prin câte un spațiu, m_k reprezentând numărul de faze ce trebuie finalizate înaintea începerii fazei k , iar numerele din sirul a reprezentând numerele de ordine ale fazelor ce trebuie finalizate înaintea începerii fazei k .

Date de ieșire

Fisierul de ieșire **pm.out** va conține $n + 1$ linii. Pe prima linie se va scrie numărul natural t , iar pe fiecare linie k dintre următoarele n linii, se vor scrie cele două numere naturale c_k și d_k , separate prin câte un spațiu.

Restricții și precizări

- $0 \leq n \leq 100$; $n \in \mathbb{N}$
- Timpul necesar finalizării executării oricărei faze nu va depăși 1.000.000
- Se consideră că executarea proiectului începe la momentul de timp 0
- Nu vor exista dependențe circulare (proiectul întotdeauna se poate finaliza)
- Pentru rezolvarea cerinței a) se acordă 40% din punctaj, iar pentru cerința b) 30% respectiv 30% din punctaj.

Exemple

pm.in	pm.out
7	11
2 3 5 3 3 3 2	0 3
0	0 0
0	3 3
1 2	2 5
1 1	2 5
1 1	8 8
3 3 4 5	8 9
1 3	

11.2.1 Indicații de rezolvare

drd. Pătcaș Csaba

Algoritmul ce trebuie folosit pentru rezolvarea problemei se numește în literatură *"Critical Path Method"*.

Prima dată se construiește *graful dependentelor* și se adaugă două noduri auxiliare, care vor reprezenta momentul de *început* și momentul de *sfârșit* al proiectului. Aceste noduri le putem numerota, de exemplu, cu 0 și $n + 1$.

Al doilea pas este nivelarea grafului: va trebui să aranjăm nodurile pe nivele, astfel încât o muchie să arate întotdeauna dinspre un nivel inferior spre un nivel superior. Acest pas se poate rezolva prin multe metode, de exemplu algoritmul clasic al *sortării topologice*.

Dacă se omite acest pas și se consideră nodurile în ordinea $1, 2, \dots, n$ se pot obține 50 de puncte.

Ultimul pas este calcularea valorilor cerute. Dacă notăm cu $earliest_k$ momentul *minim* în care poate începe faza k și $latest_k$ momentul *maxim* în care poate începe faza k fără să schimbe timpul total al proiectului, recurențele sunt ușor de dedus:

$$\begin{aligned} earliest_0 &= 0 \\ earliest_k &= \max(earliest_j + time_j), k \text{ depinde de } j \\ latest_n &= earliest_n \\ latest_k &= \min(latest_j - time_k), j \text{ depinde de } k \end{aligned}$$

Valorile *earliest* se vor calcula în ordinea crescătoare a nivelelor, iar cele *latest* în ordinea descrescătoare a nivelelor. Folosind reprezentarea grafului cu *matrice de adiacență*, complexitatea

finală este $O(n^2)$. Dacă se folosesc *liste de adiacență*, trebuie reținut și *graful transpus*, astfel complexitatea finală se poate reduce la $O(m)$. Ambele metode obțin punctaj maxim.

11.2.2 *Cod sursă

11.2.3 *Rezolvare detaliată

Capitolul 12

OJI 2008

12.1 iepuri

Problema 1 - iepuri

100 de puncte

Un gospodar are N iepuri (pe care i-a numerotat de la 1 la N) și foarte mulți morcovi. Ce e mai deosebit în această gospodărie este că iepurii sunt organizați ierarhic, în funcție de vârstă, autoritate și nevoile nutriționale. Astfel, fiecare iepure are exact un șef direct (exceptându-l pe Rilă Iepurilă, care este șeful cel mare, șeful tuturor iepurilor). Orice iepure poate avea 0, 1 sau mai mulți subordonați direcți. Orice iepure-șef va mâncă cel puțin un morcov mai puțin decât fiecare dintre subordonații săi direcți.

Gospodarul nu se poate hotărî câți morcovi să dea fiecărui iepure și ar vrea să știe în câte moduri poate împărți morcovi la iepuri știind că fiecare iepure poate să mănânce minim un morcov și maxim K morcovi.

Cerințe

Scrieți un program care calculează numărul de posibilități *modulo* 30011 de a împărți morcovi la cei N iepuri știind că orice iepure poate mâncă între 1 și K morcovi și trebuie să mănânce cu cel puțin un morcov mai puțin decât fiecare dintre iepurii care îi sunt subordonați direcți.

Date de intrare

Fișierul de intrare *iepuri.in* conține:

- pe prima linie două numere naturale N și K , separate printr-un spațiu, reprezentând numărul de iepuri, respectiv numărul maxim de morcovi ce pot fi mâncăți de un iepure.
- pe fiecare din următoarele $N - 1$ linii se află câte două numere naturale distințe a și b , cuprinse între 1 și N , separate printr-un spațiu, cu semnificația că iepurele a este șeful direct al iepurelui b .

Date de ieșire

Fișierul de ieșire *iepuri.out* va conține numărul de moduri de a împărți morcovii conform condițiilor specificate în enunț, *modulo* 30011.

Restricții și precizări

- $1 \leq N, K \leq 100$
- Numărul ce trebuie scris în fișierul de ieșire va fi afișat *modulo* 30011.

Exemplu

<i>iepuri.in</i>	<i>iepuri.out</i>
9 4	9
7 2	
7 3	
7 4	
3 5	
3 6	
5 8	
5 9	
6 1	

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

12.1.1 Indicații de rezolvare

Iolanda Popa

Soluția se bazează pe *metoda programării dinamice* și are complexitatea $O(N * K)$.

Observăm că putem deduce numărul de posibilități de a împărți un anumit număr j de morcovi la un iepure i și oricăți morcovi la iepurii subordonați direct sau indirect dacă cunoaștem numărul de posibilități de a împărți $j + 1, j + 2, \dots, j + K$ morcovi la fiecare fiu al său și oricăți morcovi iepurilor ce aparțin subarborelor desemnați de fiecare fiu. Astfel, *structura de date* folosită pentru implementarea *recurenței* devine evidentă:

$T[i][j] =$ numărul de posibilități de a împărți morcovi la iepurii ce aparțin subarborelui cu rădăcina în iepurele i șiind că iepurele i ia j morcovi

Astfel definită structura de date, luăm numărul de posibilități de a împărți morcovi la iepurii din subarborele desemnat de primul fiu pentru fiecare din cazurile: primul fiu ia $j + 1, j + 2, \dots, j + K$ morcovi. La sfârșit, adunăm toate aceste valori iar rezultatul îl înmulțim cu valorile pentru al doilea fiu, al treilea fiu etc. calculate în mod similar.

$T[i][j] = 1$, dacă iepurele i corespunde unui nod terminal

$T[i][j] = (T[f1][j+1]+T[f1][j+2]+\dots+T[f1][j+K]) *$

$(T[f2][j+1]+T[f2][j+2]+\dots+T[f2][j+K]) * \dots * (T[fp][j+1]+T[fp][j+2]+\dots+T[fp][j+K])$, unde $f1, f2, \dots, fp$ sunt numerele de ordine a celor p iepuri fi ai iepurelui i .

Rezultatul problemei va fi dat de $T[R][1]+T[R][2]+\dots+T[R][K]$ unde R reprezintă numărul de ordine al lui Rilă-Iepurilă, adică al iepurelui care nu are nici un *șef* (rădăcina arborelui).

Soluție alternativă - Stelian Ciurea

1. reprezentăm *arborele* sub forma *listelor de adiacență* implementate dinamic; la citire construim și *vectorul de predecesori* (legături de tip tată - aceasta ne permite să determinăm rădăcina arborelui și deasemenea să parcurgem arborele în adâncime fără să utilizăm un vector pentru nodurile vizitate)

2. determinăm rădăcina

3. facem o *parcugere în adâncime* a arborelui plecând din rădăcină: această funcție de parcugere are 2 parametri: nd = nodul curent și i care reprezintă numărul minim de morcovi pe care îi poate mâncă iepurele din nd .

În această funcție, pentru o valoare j a numărului de morcovi, dacă notăm cu $nrp1$ numărul de posibilități pe care le avem în *fiul_1* al lui nd cu cel puțin $j + 1$ morcovi, cu $nrp2$ numărul de posibilități pe care le avem în *fiul_2* cu cel puțin $j + 1$ morcovi etc, numărul de posibilități pe care le avem pentru nd este $nrp1 * nrp2 * \dots$

Numărul total de posibilități la nivelul lui nd se obține ca o însumare a valorilor pe care le obținem pentru toate valorile pe care le ia j (de la i la k).

Această soluție obtine 60 de puncte; pentru 40% dintre teste nu se încadrează în limita de rulare de o secundă.

12.1.2 Cod sursă

Listing 12.1.1: IEPURI.cpp

```

1 #include <stdio.h>
2
3 #define NMAX 101
4 #define KMAX 103
5 #define MOD 30011
6
7 unsigned N, K, arb[NMAX][NMAX],
8     rad, p[NMAX];
9
10 void readinput()
11 {
12     unsigned a, b;
13

```

```

14     freopen( "iepuri9.in", "rt", stdin );
15     scanf( "%u%u", &N, &K );
16
17     for( int i = 0; i < N - 1; i++ )
18     {
19         scanf( "%u%u", &a, &b );
20         arb[a][ ++arb[a][0] ] = b;
21         p[b]=a;
22     }
23     for( int i = 1; i <= N; i++ )
24         if( !p[i] ) rad = i;
25 }
26
27 unsigned m[NMAX][KMAX];
28
29 void go( unsigned rad )
30 {
31     for( int i = 1; i <= arb[rad][0]; i++ )
32         go( arb[rad][i] );
33
34     for( int k = K; k >= 1; k-- )
35     {
36         m[rad][k] = 1;
37         for( int i = 1; i <= arb[rad][0]; i++ )
38             {
39                 m[ arb[rad][i] ][ k + 1 ] += m[ arb[rad][i] ][ k + 2 ];
40                 m[ arb[rad][i] ][ k + 1 ] %= MOD;
41
42                 long temp = m[rad][k];
43                 temp *= m[ arb[rad][i] ][ k + 1 ];
44                 m[rad][k] = temp % MOD;
45             }
46     }
47 }
48
49 void writeoutput()
50 {
51     unsigned sum = 0;
52
53     for( int i = 1; i <= K; i++ )
54     {
55         sum += m[rad][i];
56         sum %= MOD;
57     }
58
59     freopen( "iepuri.out", "wt", stdout );
60     printf( "%u\n", sum );
61 }
62
63 int main()
64 {
65     readinput();
66     go( rad );
67     writeoutput();
68     return 0;
69 }
```

Listing 12.1.2: IEPURI60.cpp

```

1 //problema iepuri - Stelian Ciurea
2 //solutie alternativa = 60 puncte
3 #include<iostream>
4 #include<math.h>
5
6 using namespace std;
7
8 #define nmax 101
9 #define nrprim 30011
10
11 ifstream f("iepuri.in");
12 ofstream fout("iepuri.out");
13
14 struct nod
15 {
16     int inf;
```

```

17     nod * urm;
18 }
19
20 nod * prim[nmax];
21 int t[nmax], i,n,k,rad;
22 long rez;
23
24 void citire()
25 { int a,b;
26
27     f>>n>>k;
28     for(i=1;i<n;i++)
29     {
30         f>>a>>b;
31         nod *p= new nod;
32         p->inf=a;
33         p->urm=prim[b];
34         prim[b]=p;
35         nod * q=new nod;
36         q->inf=b;
37         q->urm=prim[a];
38         prim[a]=q;
39         t[b]=a;
40     }
41 }
42
43 long df (int nd, int i) //i=numarul de morcovi pe care il haleste nd
44 {
45     long sum=0,rez,aux,j;
46     for (j=i;j<=k;j++)
47     {
48         rez=1;
49         for(nod * p=prim[nd]; p!=NULL; p=p->urm)
50         if(p->inf!=t[nd])
51         {
52             aux=df(p->inf,j+1);
53             rez = (rez*aux)%nrprim;
54         }
55         if (rez==0) break;
56         sum = (rez+sum) % nrprim;
57     }
58     return sum;
59 }
60 int main()
61 {
62     citire();
63     rez = 0;
64     for (rad=1;rad<=n;rad++)
65     if (t[rad]==0)
66     break;
67     rez = df(rad,1);
68     fout<<rez<<endl;
69     return 0;
70 }

```

12.1.3 *Rezolvare detaliată

12.2 numar

Problema 2 - numar

90 de puncte

Presupunem că avem n numere prime notate a_1, a_2, \dots, a_n sortate strict crescător. Formăm un sir strict crescător b ale cărui elemente sunt toți multiplii acestor n numere prime astfel încât, multiplii comuni apar o singură dată. Presupunem că numerotarea pozițiilor elementelor din sirul b începe tot cu 1.

Cerințe

Scrieți un program care citește din fișierul de intrare valoarea lui n și apoi cele n elemente ale sirului a , determină elementul de pe poziția m din sirul b și afișează în fișierul de ieșire valoarea acestuia.

Date de intrare

Fișierul de intrare **numar.in** conține

- pe prima linie două numere naturale separate printr-un spațiu care reprezintă primul valoarea lui n și al doilea valoarea lui m ;
- pe a doua linie n numere naturale prime separate prin câte un spațiu care reprezintă valorile elementelor sirului a . Aceste valori sunt dispuse în ordine strict crescătoare iar ultima dintre ele este mai mică decât un milion.

Date de ieșire

Fișierul de ieșire **numar.out** va conține pe prima linie o singură valoare care reprezintă termenul de pe poziția m din sirul b .

Restricții și precizări

- Pentru 30% din teste $n \leq 20, m \leq 1000, a_1 \leq 50$
- Pentru celelalte 70% din teste $21 \leq n \leq 100, 1001 \leq m \leq 15000, 51 \leq a_1 \leq 1000$
- $a_n < 1\ 000\ 000$

Exemple

numar.in	numar.out	Explicații
3 10 2 3 5	14	sirul b este format din valorile: 2, 3, 4, 5, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 22 ... Pe poziția 10 se află numărul 14
4 20 7 23 37 131	98	
3 11111 977 1009 1031	3726237	

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

12.2.1 Indicații de rezolvare

autor: ??? - ???

Rezolvare problemei se bazează pe următoarea observație: b_m nu poate fi mai mare decât $m * a_1$

Fie x un număr natural și $m =$ numărul de multiplii ai celor n numere prime din intervalul $[1; x]$. Avem (din *principiul includerii și al excluderii*):

$$m = [x/a_1] + [x/a_2] + \dots + [x/a_n] - [x/(a_1 a_2)] - \dots - [x/(a_{n-1} a_n)] + [x/(a_1 a_2 a_3)] + \dots$$

Noi îl știm pe m și trebuie să îl determinăm pe x . Aceasta se poate face prin *căutarea binară* a lui x în intervalul $[1..m * a_1]$.

Totuși ne lovim de următoarea problemă: trebuie să generăm produsele de la numitorii care apar în expresia de mai sus. E evident că acestea se pot calcula generând o singură dată submulțimile multimi $\{a_1, a_2, \dots, a_n\}$ și reținând valorile corespunzătoare produselor. Nu este însă necesar să generăm toate submulțimile ci doar pe cele care reprezintă produse mai mici decât $m * a_1$!!! Se poate demonstra sau observa că și în cazul cel mai defavorabil, când sirul a este format din cele mai mici numere prime iar $m = 15000$, numărul de produse care îndeplinesc acest criteriu este mai mic decât 10000. În consecință, vom utiliza un *backtracking* optimizat.

În final, trebuie să avem în vedere că prin *căutarea binară* s-ar putea să determinăm un număr $x_1 > x$, care să conducă la aceeași valoare a expresiei la care ne conduce și x , dar care să nu fie în sirul b !!

De exemplu, pentru $n = 2$, $a_1 = 2$ și $a_2 = 3$, $[11/2]+[11/3]-[11/6] = [10/2]+[10/3]-[10/6]$, dar 11 nu se află în sirul multiplilor numerelor 2 și 3 !!

Pentru a determina rezultatul problemei, vom face un for descreșător plecând de la valoarea pe care ne-o furnizează căutarea binară și ne vom opri la primul număr care este multiplu a cel puțin unuia dintre numerele prime date prin sirul a . (o sursă care nu va face această determinare "prinde" doar o parte din teste).

Numărul de pași în cazul cel mai nefavorabil, nu depășește: $10000(\text{backtracking}) + 10000*24(\text{căutarea binară și calculul expresiei}) + 1000*1000 (= a_1 * n \text{ pentru cele două for-uri de după căutarea binară: ecartul între doi termeni din sirul } b \text{ nu poate fi mai mare decât } a_1)$.

Sursa care implementează acest algoritm se află în subdirectorul sol în fișierul numar100.cpp .

Soluții alternative:

1) interclasarea sirurilor de multiplii (optimizată: începînd cu multiplii numerelor a_n și a_{n-1} etc) = 50 puncte - numar50.cpp

2) generare directă a multiplilor și marcarea lor într-un vector de contorizare = 30 puncte (numar30.cpp)

(Aceste două metode "au probleme" cu spațiul de memorie disponibil și cu limita de timp la testele pe care nu le iau).

3) generare directă a multiplilor și reținerea lor într-un arboare binar echilibrat (avl) = până la 40 puncte; (Depășește limita de timp la testele pe care nu le ia. Pot să apară probleme din cauza alocării memoriei) - numar40.cpp

12.2.2 Cod sursă

Listing 12.2.1: numar30.cpp

```

1 //problema numar - Stelian Ciurea
2 //generare directă, vector de contorizare = 30 puncte
3 #include <iostream>
4
5 using namespace std;
6
7 long a[1001];
8 long n,m,i,j,k,h,nrl,lim;
9 char ct[255001];
10
11 ifstream fi("numar.in");
12 ofstream fout("numar.out");
13
14 int main()
15 { //cout<<endl;
16     fi>>n>>m;
17     for (i=1;i<=n;i++)
18         fi>>a[i];
19
20     lim = m*a[1]; //limita pana la care pot fi valori in sir
21
22     for (k=1;k<=n;k++)
23         for (i=1;i*a[k]<=lim;i++)
24             ct[i*a[k]]=1;
25     j=0;
26     for (i=1;j!=m;i++)
27         if (ct[i]==1)
28         {
29             cout<<i<<' ';
30             j++;
31         }
32
33     fout<<i-1<<endl;
34
35     return 0;
36 }
```

Listing 12.2.2: numar40.cpp

```

1 //problema numar - stelian ciurea
2 //avl = 40 puncte
3 #include<fstream>
4 #include<stdlib.h>
5
6 using namespace std;
7
8 struct nod
9 {
10     int info;
11     int hi;
12     nod *st, *dr;
13 };
14
15 long a[1000],ct,n,m,sol,lim,mult;
16 int max2(int a, int b)
17 { if (a>b) return a; return b; }
18
19 int high(nod *p)
20 { if (p==NULL) return -1;
21     return p->hi;
22 }
23
24 void rotst(nod*&x)
25 { nod *y = x->dr;
26     x->dr = y->st;
27     y->st = x;
28     x->hi = 1+max2(high(x->st),high(x->dr));
29     y->hi = 1+max2(high(y->st),high(y->dr));
30     x = y;
31 }
32
33 void rotdr(nod*&y)
34 { nod *x = y->st;
35     y->st = x->dr;
36     x->dr = y;
37     y->hi = 1+max2(high(y->st),high(y->dr));
38     x->hi = 1+max2(high(x->st),high(x->dr));
39     y = x;
40 }
41
42 void rotdst(nod*&x)
43 {
44     rotdr(x->dr);
45     rotst(x);
46 }
47
48 void rotddr(nod*&y)
49 {
50     rotst(y->st);
51     rotdr(y);
52 }
53
54 void ins(nod*&p, int vl)
55 {
56     if (p==NULL)
57     { p = new nod;
58         p->info = vl;
59         p->st=p->dr=NULL;
60         p->hi = 0;
61         return;
62     }
63     if (p->info == vl) return;
64     if (p->info < vl)
65     { ins(p->dr,vl);
66         if (high(p->dr) - high(p->st) > 1)
67             if ( high(p->dr->dr) >= high(p->dr->st) )
68                 rotst(p);
69             else
70                 rotdst(p);
71             else
72                 p->hi = 1 + max2(high(p->st), high(p->dr));
73     }
74 }
```

```

75     else
76     { ins(p->st,v1);
77
78     if (high(p->st) - high(p->dr) > 1)
79       if ( high(p->st->st) >= high(p->st->dr) )
80         rotdr(p);
81     else
82       rotddr(p);
83     else
84       p->hi = 1 + max2(high(p->st), high(p->dr));
85
86   }
87 }
88
89 int ad(nod *p)
90 { if (p==NULL) return -1;
91   int a1 = ad(p->st);
92   int a2 = ad(p->dr);
93   return 1 + max2(a1,a2);
94 }
95
96 void inord(nod* p)
97 { if (p!=NULL)
98   { inord(p->st);
99     ct++;
100    // cout<<p->info<<' ';
101    if (ct==m)
102      sol = p->info;
103    inord(p->dr);
104   }
105 }
106
107 int main()
108 { int i;
109   nod *root=NULL;
110   ifstream fi("numar.in");
111   ofstream fout("numar.out");
112   fi>>n>>m;
113   for (i=0;i<n;i++)
114     f>>a[i];
115   lim = a[1]*m;
116   for (i=0;i<n;i++)
117     { mult = a[i];
118       while (mult<=lim)
119         { ins(root,mult);
120           mult += a[i];
121         }
122     }
123
124   inord(root);
125   fout<<sol<<endl;
126   fout.close();
127
128   return 0;
129 }
```

Listing 12.2.3: numar50.cpp

```

1 //problema numar - stelian ciurea
2 //interclasare optimizata - 50 puncte
3 #include <fstream>
4
5 using namespace std;
6
7 long a[1001];
8 long n,m,i,j,k,h,nrl,lim;
9 long sir1[10001],sir2[10001];
10
11 ifstream fi("numar.in");
12 ofstream fout("numar.out");
13
14 int main()
15 { //cout<<endl;
16   fi>>n>>m;
17   for (i=1;i<=n;i++)
```

```

18     fi>>a[i];
19
20     lim = m*a[1]; //limita pana la care pot fi valori in sir
21
22     k = n;           //interclasare de la n la 1
23     while (a[k]>lim) k--;
24
25     for (i=1;i*a[k]<=lim;i++)
26         sir1[i]=i*a[k];
27     nrl = i;
28     for (k=k-1;k>=1;k--)
29     {i=1;
30         j=1;
31         h=1;
32         while (h<=m && j*a[k]<=lim)
33             if (sir1[i]<j*a[k])
34                 {sir2[h]=sir1[i];i++;h++;}
35             else
36                 if (sir1[i]==j*a[k])
37                     {sir2[h]=sir1[i];j++;i++;h++;}
38             else
39                 {sir2[h]=j*a[k];j++;h++;}
40
41         nrl = h-1;
42         for (i=1;i<=nrl;i++)
43             sir1[i] = sir2[i];
44     }
45 /*   for (i=1;i<=m;i++)
46     cout<<sir1[i]<<' ';
47 */
48     fout<<sir1[m]<<endl;
49     fout.close();
50
51     return 0;
52 }
```

Listing 12.2.4: numar100.cpp

```

1 //problema numar - stelian ciurea
2 //solutia "oficiala" 100 puncte
3 #include <fstream>
4 #include <iostream>
5
6 using namespace std;
7
8 long a[1001];
9 long st,dr,mj,vl,n,m,i,j,k,h,nrl,lim,ok,pro,np;
10 long prod[13000];
11 int s[1001];
12
13
14 long f(long x)
15 { long vl = 0;
16   for (i=1;i<=np;i++)
17     if (prod[i]!=0)
18       vl = vl + x/prod[i];
19     else
20       cout<<"upsi";
21   return vl;
22 }
23
24 void back(int p,long pr)
25 { int i,j;
26   for (i=s[p-1]+1;i<=n-k+p;i++)
27     {s[p]=i;
28      if (a[i] > lim / pr) return;
29      if (p==1)
30        { pro=1;
31          for (j=1;j<=k;j++)
32            { if (pro <= lim/a[i+j-1])
33                pro = pro*a[i+j-1];
34              else
35                return;
36            }
37        }

```

```

38     if (p==k)
39     { pro=1;
40     for (j=1;j<=k;j++)
41     { if (pro <= lim/a[s[j]])
42       pro = pro*a[s[j]];
43     else return;
44     //if (pro>lim) return;
45   }
46   if (pro<=lim)
47   { np++;
48   //cout<<np<<' ';
49   if (k%2==1)
50     prod[np]=pro;
51   else
52     prod[np]=-pro;
53   }
54   else return;
55   }
56   else
57   back(p+1,pr*a[i]);
58   }
59 }
60 }
61 ifstream fi("numar.in");
62 ofstream fout("numar.out");
63
64 int main()
65 { cout<<endl;
66   fi>>n>>m;
67   for (i=1;i<=n;i++)
68     fi>>a[i];
69
70   lim = m*a[1]; //limita pana la care pot fi valori in sir
71
72   np=0;
73
74   for (k=1;k<=n;k++)
75     back(1,1);
76
77   st = 1;
78   dr = lim;
79   do
80   { mj = (st+dr) / 2;
81     vl = f(mj);
82     if (vl==m)
83     { //cout<<mj<<endl;
84     break;
85     }
86     if (vl<m) st = mj+1;
87     else dr = mj-1;
88   }
89   while (st<=dr);
90
91   for (k=mj;k>=0;k--)
92   {
93     for (i=1;i<=n;i++)
94     if (k%a[i]==0)
95     { fout<<k<<endl;
96     return 0;
97     }
98   }
99 }
100
101 return 0;
102 }
```

12.2.3 *Rezolvare detaliată

Capitolul 13

OJI 2007

13.1 Numere

Fie a și b două numere naturale nenule.

Cerință

Scrieți un program care citește din fișierul de intrare două valori a și b , determină numărul de numere naturale formate din exact a cifre care au fiecare produsul cifrelor egal cu b și afișează în fișierul de ieșire restul împărțirii valorii determinate la numărul 9973.

Date de intrare

Fișierul de intrare **numere.in** conține pe prima linie numerele a și b despărțite printr-un spațiu.

Date de ieșire

Fișierul de ieșire **numere.out** va conține pe prima linie o singură valoare care reprezintă restul împărțirii numărului de numere naturale formate din exact a cifre care au produsul cifrelor egal cu b la 9973.

Restricții și precizări

- Pentru 10% din teste $1 \leq a \leq 6, 1 \leq b \leq 1000$
- Pentru 20% din teste $7 \leq a \leq 150, 1 \leq b \leq 100$
- Pentru 30% din teste $151 \leq a \leq 1000, 1 \leq b \leq 100$
- Pentru 40% din teste $1001 \leq a \leq 9000, 100 \leq b \leq 9000$

Exemple:

numere.in	numere.out	Explicații - numerele sunt:
3 9	6	119 133 191 313 331 911
4 15	12	1135 1153 1315 1351 1513 1531 3115 3151 3511 5113 5131 5311
1000 210	833	...

Timp maxim de execuție/test: 1 secundă

13.1.1 Indicații de rezolvare

Soluția 1 - Programare dinamică. (Ilie Vieru) -100 puncte

- Dacă a are divizori primi mai mari decât 9 atunci scrie 0
- Altfel:

$$nr[1][d] = \begin{cases} 0, & \text{dacă } b \% d \neq 0 \\ 1, & \text{în rest} \end{cases}$$

$$nr[i][j] = \sum_{\substack{1 \leq p \leq 9 \\ p|b, p|j}} nr[i-1][j/p], \text{ unde } 2 \leq i \leq a, 1 \leq j \leq b$$

Se afișează $nr[a][b](mod)$.

Problema se rafinează ajungând la complexitatea $O(a * nrdiv(b))$ înlocuind calculul matriceal cu cel cu vectori.

Soluția 2 - backtracking (Stelian Ciurea) -100 puncte

Problema se poate rezolva și prin metoda backtracking!

Pentru aceasta:

- se face descompunerea numărului b în factori primi (se observă că puterea maximă la care poate să apară un factor prim în această descompunere este 13 în cazul factorului prim 2);
- în funcție de această descompunere se reține pentru fiecare cifră care este numărul maxim de câte ori poate să apară în numărul a (în vectorul f);
- se determină prin backtracking, succesiv, câte o configurație posibilă a numărului a astfel încât produsul ciferelor lui să fie b : astfel $s[2]$ va reține câte cifre de 2 apar în a , $s[3]$ va reține câte cifre de 3 apar în a etc;
- se determină numărul de cifre de 1 care apare într-o astfel de configurație;
- pentru o configurație posibilă se calculează toate numerele care au respectiva configurație a ciferelor, folosind formula *aranjamentelor cu repetiție*:

$$A_b^{s[1], s[2], \dots, s[9]} = \frac{b}{s[1]! s[2]! \dots s[9]!}$$

unde $s[1]$ este numărul de cifre de 1 care apar în a , $s[2]$ este numărul de cifre de 2 care apar în a etc.

- plecând de la observația evidentă că numărul de cifre de 1 este mult mai mare în comparație cu numărul celorlalte cifre (mai ales pentru valori mari ale lui b), se simplifică înainte de a începe calculul propriu-zis în formula anterioară $b!$ cu $s[1]!$.

- pentru a evita calcule repetate, înainte de apelarea subprogramului care implementează backtracking-ul descris anterior, se precalculează în vectorul *fact* factorialele numerelor de la 2 la 13 (13 fiind factorialul maxim care apare la numitorul aranjamentelor cu repetiții după simplificarea descrisă mai sus), precum și puterile ciferelor de la 1 la 13 în matricea *pow*.

13.1.2 Cod sursă

Listing 13.1.1: numerev1.cpp

```

1 //numarul de numere de n cifre si prod cifrelor k
2 //complexitate : O(n * nrdiv(k))
3 #include <stdio.h>
4 #include <string.h>
5
6 #define MAX 9001
7 #define CONST 9973
8 #define INPUT "numere.in"
9 #define OUTPUT "numere.out"
10
11 int N, K, ndiv;
12 int nrpos1[MAX], nrpos2[MAX], diviz[MAX];
13
14 FILE *f = fopen(INPUT, "r");
15 FILE *g = fopen(OUTPUT, "w");
16
17 int prim(int nr)
18 {
19     if(nr % 2 == 0) return 0;
20     int i;
21     for(i = 3; i*i <= nr; i += 2)
22         if(nr%i == 0) return 0;
23     return 1;
24 }
25
26 int main()
27 {
28     fscanf(f, "%d %d", &N, &K);
29     int ok = 1, i, j, p;
30
31     for(i = 11; i <= K && ok; ++i)
32         if(K % i == 0 && prim(i)) ok = 0;
33
34     for(i = 1; i <= K; ++i)
35         if(K%i == 0) diviz[++ndiv] = i;
36
37     if(ok)
38     {

```

```

39     for(i = 1; i <= ndiv && diviz[i] <= 9 ; ++i)
40         nrpos1[diviz[i]] = 1;
41     for(i = 2; i <= N; ++i)
42     {
43         for(j = 1; j <= ndiv; ++j)
44             nrpos2[diviz[j]] = 0;
45         for(j = 1; j <= ndiv; ++j)
46         {
47             for(p = 1; p <= ndiv && diviz[p] <= 9; ++p)
48                 if(diviz[j] % diviz[p] == 0)
49                     nrpos2[diviz[j]] = (nrpos2[diviz[j]] +
50                                         nrpos1[diviz[j]/diviz[p]]) % CONST;
51         }
52         for(j = 1;j <= ndiv; ++j)
53             nrpos1[diviz[j]] = nrpos2[diviz[j]] % CONST;
54
55     }
56
57     fprintf(g, "%d\n", nrpos1[K]);
58 }
59 else fprintf(g, "0\n");
60
61     return 0;
62 }
```

13.1.3 Rezolvare detaliată

Varianta 1:

Listing 13.1.2: numerel.java

```

1 %\begin{verbatim}
2 import java.io.*;
3 class numerel
4 {
5     static StreamTokenizer st;
6     static PrintWriter out;
7
8     static final int MAXAB=9000;
9
10    static int nc,pc,ndiv=0,rez=-1; // nc=nr cifre, pc=produs cifre
11
12    static int[] nrpos1=new int[MAXAB+1];
13    static int[] nrpos2=new int[MAXAB+1];
14    static int[] div =new int[MAXAB+1];
15
16
17    static void citire() throws IOException
18    {
19        st.nextToken(); nc=(int)st.nval;
20        st.nextToken(); pc=(int)st.nval;
21    } // citire(...)
22
23    static boolean prim(int nr)
24    {
25        int d;
26        if((nr%2)==0) return false;
27        for(d=3;d*d<=nr;d+=2) if((nr%d)==0) return false;
28        return true;
29    }
30
31    static void solutia()
32    {
33        int i,j,p;
34        boolean ok=true;
35
36        for(i=11;i<=pc;++i)
37            if((pc%i)==0 && prim(i))
38            {
39                ok=false;
40                break;
41            }
42    }

```

```

43     for(i=1;i<=pc;++i)  if((pc%i)==0)  div[++ndiv]=i;
44
45     if(ok)
46     {
47         for(i=1; i<=ndiv && div[i]<=9 ; ++i) nrpos1[div[i]]=1;
48
49         for(i=2;i<=nc;++i)
50         {
51             for(j=1;j<=ndiv;++j)      nrpos2[div[j]]=0;
52
53             for(j=1;j<=ndiv;++j)
54             {
55                 for(p=1;p<=ndiv && div[p]<=9;++p)
56                 if(div[j]%div[p]==0)
57                     nrpos2[div[j]]=(nrpos2[div[j]]+nrpos1[div[j]/div[p]])%9973;
58             }
59
60             for(j=1;j<=ndiv;++j) nrpos1[div[j]]=nrpos2[div[j]]%9973;
61         }
62
63         rez=nrpos1[pc];
64     }
65     else rez=0;
66 }
67
68 public static void main(String[] args) throws IOException
69 {
70     st=new StreamTokenizer(new BufferedReader(new FileReader("numere.in")));
71     out=new PrintWriter(new BufferedWriter(new FileWriter("numere.out")));
72
73     citire();
74     solutia();
75
76     //System.out.println(rez);
77     out.println(rez);
78     out.close();
79 } // main
80 } // class
81 %\end{verbatim}

```

Varianta 2:

Listing 13.1.3: numere2.java

```

1  %\begin{verbatim}
2  import java.io.*;
3  class numere2
4  {
5      static StreamTokenizer st;
6      static PrintWriter out;
7
8      static int nc,pc; // nc=nr cifre, pc=produs cifre
9
10     static int[] s=new int[10];
11     static int[] f=new int[10];
12     static int[] fact=new int[21];
13     static int[][] pow=new int[10][14];
14     static int k2,k3,k5,k7;
15
16     static int j,b1,a,b,i;
17     static int x,prod1,rez1,rez,prod;
18
19     static void citire() throws IOException
20     {
21         st.nextToken(); a=(int)st.nval;
22         st.nextToken(); b=(int)st.nval;
23     }// citire(...)
24
25     static int cmmdc(int a, int b)
26     {
27         while(a!=b) if(a>b) a-=b; else b-=a;
28         return a;
29     }
30

```

```

31
32     static void print(int p)
33     {
34         int g13,j1,i,j,k;
35
36         k=0; g13=0; prod1=1;
37
38         for(j=2;j<=p;j++)
39         {
40             k=k+s[j];
41             if(s[j]==13) g13=1;
42             prod1=prod1*fact[s[j]];
43         }
44
45         s[1]=a-k;
46         rez1=1;
47         for(j=s[1]+1;j<=a;j++)
48         {
49             j1=j;
50             if(prod1!=1)
51             {
52                 x=cmmdc(j,prod1);
53                 j1=j1/x;
54                 prod1=prod1/x;
55                 if(g13==1)
56                     if(j1%13==0) { g13=0; j1=j1/13; }
57                 rez1=(rez1*j1)%9973;
58             }
59         }
60
61         rez=(rez+rez1)%9973;
62     }
63
64
65     static void back(int p)
66     {
67         int i;
68         for(i=0;i<=f[p];i++)
69         {
70             s[p]=i;
71             prod=prod*pow[p][i];
72             if(prod==b1) print(p);
73             if(prod<b1) if(p<9) back(p+1);
74             prod=prod/pow[p][i];
75         }
76     }
77
78
79     static void solutia()
80     {
81         b1=b;
82         fact[0]=1;
83         for(i=1;i<=12;i++) fact[i]=fact[i-1]*i ;
84         fact[13]=fact[12];
85
86         for(i=2;i<=9;i++) { pow[i][1]=i; pow[i][0]=1; }
87         for(i=2;i<=9;i++)
88             for(j=2;j<=13;j++)
89                 if(pow[i][j-1]*i<10000) pow[i][j]=pow[i][j-1]*i;
90
91         while(b%2==0) { b=b/2; k2++; }
92         while(b%3==0) { b=b/3; k3++; }
93         while(b%5==0) { b=b/5; k5++; }
94         while(b%7==0) { b=b/7; k7++; }
95
96         f[2]=k2; f[3]=k3; f[4]=k2/2; f[5]=k5;
97         f[7]=k7; f[8]=k2/3; f[9]=k3/2;
98         if(k2<k3) f[6]=k2; else f[6]=k3;
99
100        prod=1;
101        back(2);
102    }
103
104    public static void main(String[] args) throws IOException
105    {
106        st=new StreamTokenizer(new BufferedReader(new FileReader("numere.in")));

```

```

107     out=new PrintWriter(new BufferedWriter(new FileWriter("numere.out")));
108
109     citire();
110     solutia();
111
112     System.out.println(rez);
113     out.println(rez);
114     out.close();
115 } // main
116 } // class
117 %\end{verbatim}

```

13.2 Cezar

În Roma antică există n așezări senatoriale distințe, câte una pentru fiecare dintre cei n senatori ai Republicii. Așezările senatoriale sunt numerotate de la 1 la n , între oricare două așezări existând legături directe sau indirecte. O legătură este directă dacă ea nu mai trece prin alte așezări senatoriale intermediare. Edilii au pavat unele dintre legăturile directe dintre două așezări (numind o astfel de legătură pavată "stradă"), astfel încât între oricare două așezări senatoriale să existe o singură succesiune de străzi prin care se poate ajunge de la o așezare senatorială la cealaltă.

Toți senatorii trebuie să participe la ședințele Senatului. În acest scop, ei se deplasează cu lectica. Orice senator care se deplasează pe o stradă plătește 1 ban pentru că a fost transportat cu lectica pe acea stradă.

La alegerea sa ca prim consul, Cezar a promis că va dota Roma cu o lectică gratuită care să circule pe un număr de k străzi ale Romei astfel încât orice senator care va circula pe străzile respective, să poată folosi lectica gratuită fără a plăti. Străzile pe care se deplasează lectica gratuită trebuie să fie legate între ele (zborul, metroul sau teleportarea nefind posibile la acea vreme).

În plus, Cezar a promis să stabilească sediul sălii de ședințe a Senatului într-o sală dintre așezările senatoriale aflate pe traseul lecticii gratuite. Problema este de a alege cele k străzi și amplasarea sediului sălii de ședințe a Senatului astfel încât, prin folosirea transportului gratuit, senatorii, în drumul lor spre sala de ședințe, să facă economii cât mai însemnante. În calculul costului total de transport, pentru toți senatorii, Cezar a considerat că fiecare senator va călători exact o dată de la așezarea sa până la sala de ședințe a Senatului.

Cerință

Scriți un program care determină costul minim care se poate obține prin alegerea adecvată a celor k străzi pe care va circula lectica gratuită și a locului de amplasare a sălii de ședință a Senatului.

Date de intrare

Fișierul **cezar.in** conține

- pe prima linie două valori n și k separate printr-un spațiu reprezentând numărul total de senatori și numărul de străzi pe care circulă lectica gratuită
- pe următoarele $n - 1$ linii se află câte două valori i și j separate printr-un spațiu, reprezentând numerele de ordine a două așezări senatoriale între care există stradă.

Date de ieșire

Pe prima linie a fișierului **cezar.out** se va scrie costul total minim al transportării tuturor senatorilor pentru o alegere optimă a celor k străzi pe care va circula lectica gratuită și a locului unde va fi amplasată sala de ședințe a Senatului.

Restricții și precizări

- $1 < n \leq 10000$, $0 < k < n$
- $1 \leq i, j \leq n$, $i \neq j$
- Oricare două perechi de valori de pe liniile $2, 3, \dots, n$ din fișierul de intrare reprezintă două străzi distințe.
 - Perechile din fișierul de intrare sunt date astfel încât respectă condițiile din problemă.
 - Pentru 25% din teste $n \leq 30$, pentru 25% din teste $30 < n \leq 1000$, pentru 25% din teste $1000 < n \leq 3000$, pentru 10% din teste $3000 < n \leq 5000$, pentru 10% din teste $5000 < n \leq 10000$.

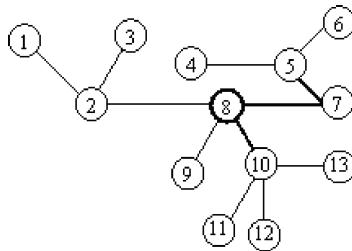
Exemplu

Figura 13.1: Cezar

cezar.in	cezar.out	Explicație
13 3	11	Costul minim se obține, de exemplu, pentru alegerea celor 3 străzi între așezările 5-7, 7-8, 8-10 și a sălii de ședințe a Senatului în așezarea 8 (după cum este evidențiat în desen).
1 2		
2 3		
2 8		
7 8		
7 5		
5 4		
5 6		
8 9		
8 10		
10 11		
10 12		
10 13		

Timp maxim de execuție/test: 0.5 secunde

13.2.1 Indicații de rezolvare

O implementare posibilă utilizează metoda GREEDY, $O(n * (n - k))$ sau $O((n - k) * \log(n))$.

Se elimină succesiv, dintre frunzele existente la un moment dat, frunza de cost minim. Toate nodurile au costul inițial 1. La eliminarea unei frunze, se incrementează cu 1 costul tatălui acesteia. Validitatea metodei rezultă din observația că, la eliminarea unei frunze oarecare, tatăl acesteia poate deveni frunză la rândul lui, dar cu un cost strict mai mare decât al frunzei eliminate.

Se poate reține *arborele* cu ajutorul listelor de adiacență (liniare sau organizate ca *arbori de căutare*), iar frunzele se pot memora într-un *minheap* de costuri, structură care se actualizează în timp logaritmic.

13.2.2 Cod sursă

Listing 13.2.1: cezar.cpp

```

1 #include <fstream>
2
3 using namespace std;
4
5 struct NOD
6 {
7     int nod;
8     NOD* next;
9 };
10
11 NOD *dp[10000];
12 long s; int c,n,k,h[10001],lh;
13
14 void add(NOD*& p, int a)

```

```

15 {NOD* q;
16 q=new(NOD);q->next=p;
17 q->nod=a;p=q;
18 }
19
20 void remove(NOD*& p, int a)
21 {NOD *q,*r;
22 if (p->nod==a)
23 {
24 q=p;p=p->next;
25 delete q;
26 }
27 else
28 {
29 q=p;
30 while (q->next->nod!=a) q=q->next;
31 r=q->next;q->next=q->next->next;
32 delete r;
33 }
34 }
35
36 void citire()
37 {int i,x,y;
38 ifstream f("cezar.in");
39 f>>n>>k;
40
41 c=new int[10000];
42 for (i=0;i<n;i++) {
43 *(c+i)=1;dp[i]=0;h[i]=n;
44 }
45
46 for (i=0;i<n-1;i++) {
47 f>>x>>y;
48 add(dp[x-1],y-1);add(dp[y-1],x-1);
49 }
50
51 for (i=0;i<n;i++)
52 if (!dp[i]->next) h[++lh]=i;
53 *(c+n)=20000;h[n]=n;
54
55 f.close();
56 }
57
58 void desfrunzire()
59 { int ii,i,j,pmin;
60 long min;
61 s=0;
62 for (ii=n-1;ii>=k+1;ii--)
63 {
64 pmin=h[1];s+=*(c+pmin);
65 j=dp[pmin]->nod;
66 *(c+j)+=*(c+pmin);
67 remove(dp[j],pmin);
68 if (!dp[j]->next) h[1]=j;
69 else {h[1]=h[lh];h[lh]=n;lh--;}
70 i=1;
71 while (2*i<=lh)
72 {
73 if ( *(c+h[2*i]) < *(c+h[2*i+1])) j=2*i; else j=2*i+1;
74 if ( *(c+h[i]) > *(c+h[j])) {
75 int aux=h[i];h[i]=h[j];h[j]=aux;i=j;
76 else i=n;
77 }
78 }
79 }
80
81 int main()
82 { citire();
83 desfrunzire();
84 ofstream f("cezar.out");
85 f<<s<<'\'n';
86 f.close();
87
88 return 0;
89 }

```

13.2.3 Rezolvare detaliată

Varianta 1a: Cu test incorct, dar care obține 17 rezultate corecte din 20.

Listing 13.2.2: cezar1a.java

```

1 import java.io.*;           // teste: 7(1070,1072), 17(17825,17853) si 20(17154,17191) ...
2   ???
3 class cezarla             // eliminat <=> g[i]=0 ... !!!
4 {
5   static StreamTokenizer st;
6   static PrintWriter out;
7
8   static int n, ns;
9
10  static int[] v1 = new int[10001];
11  static int[] v2 = new int[10001];
12  static int[] g = new int[10001];           // gradele
13  static int[] ca = new int[10001];          // ca[k]=costul acumulat in nodul k !!!
14  static int[] nde = new int[10001];          // nde[k]=nr "descendentii" eliminati pana in
15    k
16
17  static void afisv(int[] v,int i1, int i2)
18  {
19    int i;
20    for(i=i1;i<=i2;i++)
21    {
22      System.out.print(v[i]+" ");
23      if(i%50==0) System.out.println();
24    }
25    System.out.println();
26
27  static void citire() throws IOException
28  {
29    int i,j,k;
30
31    st.nextToken(); n=(int)st.nval;
32    st.nextToken(); ns=(int)st.nval;
33
34    for(i=1;i<=n;i++) // curatenie ...
35    {
36      g[i]=0;
37      ca[i]=0;
38      nde[i]=0;
39    }
40
41    for(k=1;k<=n-1;k++)
42    {
43      st.nextToken(); i=(int)st.nval;
44      st.nextToken(); j=(int)st.nval;
45
46      v1[k]=i;
47      v2[k]=j;
48      g[i]++; g[j]++;
49    }
50
51    //afisv(v1,1,n-1);
52    //afisv(v2,1,n-1);
53    //afisv(g,1,n);
54 } // citire(...)

55 static int tata(int i) // mai bine cu lista de adiacenta ...
56 {
57   int k,t;
58   t=-1;           // initializarea aiurea ...
59   for(k=1;k<=n-1;k++)
60   {
61     if( (v1[k]==i) && (g[v2[k]] > 0)) {t=v2[k]; break;}
62     else
63       if( (v2[k]==i) && (g[v1[k]] > 0)) {t=v1[k]; break;}
64   }
65   return t;
66 }
67

```

```

68     static void eliminm()      // frunze(g=1) cu cost=min
69     {
70         int i, imin, timin;
71         int min;
72
73         min=Integer.MAX_VALUE;
74         imin=-1;           // initializare aiurea ...
75         for(i=1;i<=n;i++) // mai bine cu heapMIN ...
76             if(g[i]==1)    // i=frunza
77                 if(ca[i]+nde[i]<min) // aici este testul INCORECT ...
78                 {
79                     min=ca[i]+nde[i];
80                     imin=i;
81                 }
82
83         timin=tata(imin);
84
85         g[imin]--; g[timin]--;
86
87         ca[timin]=ca[timin]+ca[imin]+nde[imin]+1;
88         nde[timin]=nde[timin]+nde[imin]+1;
89
90         //System.out.println(" Elimin nodul "+imin+" timin = "+timin+
91         //                      " ca = "+ca[timin]+" nde = "+nde[timin]);
92     }// elimin()
93
94     public static void main(String[] args) throws IOException
95     {
96         int k, senat=0, cost=0;
97         st=new StreamTokenizer(new BufferedReader(new FileReader("cezar20.in")));
98         out=new PrintWriter(new BufferedWriter(new FileWriter("cezar.out")));
99
100        citire();
101
102        for(k=1;k<=n-1-ns;k++)
103        {
104            //System.out.print(k+" : ");
105            eliminm();
106        }
107
108        //afisv(c,1,n);
109
110        for(k=1;k<=n;k++)
111            if(g[k]>0)
112            {
113                cost+=ca[k];
114                senat=k;
115            }
116
117        System.out.println("\n"+cost+" "+senat);
118
119        out.println(cost+" "+senat);
120        out.close();
121    } // main
122 } // class

```

Varianta 1b: Cu testul corect!

Listing 13.2.3: cezar1b.java

```

1 import java.io.*; // OK: "f" cu cat incarca "in plus" = nde[i]+1 (fara ca[i] !!!)
2 class cezar1b      // eliminat <==> g[i]=0 ... !!!
3 {
4     static StreamTokenizer st;
5     static PrintWriter out;
6
7     static int n, ns;
8
9     static int[] v1 = new int[10001];
10    static int[] v2 = new int[10001];
11    static int[] g = new int[10001];      // gradele
12    static int[] ca = new int[10001];      // ca[k]=costul acumulat in nodul k !!!
13    static int[] nde = new int[10001];      // nde[k]=nr "descendenți" eliminati pana in k
14

```

```

15     static void citire() throws IOException
16     {
17         int i,j,k;
18
19         st.nextToken(); n=(int)st.nval;
20         st.nextToken(); ns=(int)st.nval;
21
22         for(i=1;i<=n;i++) g[i]=ca[i]=nde[i]=0; // curatenie ...
23
24         for(k=1;k<=n-1;k++)
25         {
26             st.nextToken(); i=(int)st.nval;
27             st.nextToken(); j=(int)st.nval;
28
29             v1[k]=i;
30             v2[k]=j;
31             g[i]++; g[j]++;
32         }
33     } // citire(...)

34
35     static int tata(int i) // mai bine cu lista de adiacenta ...
36     {
37         int k,t;
38         t=-1; // initializarea aiurea ...
39         for(k=1;k<=n-1;k++) // n-1 muchii
40         {
41             if( (v1[k]==i) && (g[v2[k]] > 0) ) {t=v2[k]; break;}
42             else
43                 if( (v2[k]==i) && (g[v1[k]] > 0) ) {t=v1[k]; break;}
44         }
45         return t;
46     }

47
48     static void eliminm() // frunze(g=1) cu cost=min
49     {
50         int i, imin, timin;
51         int min;
52
53         min=Integer.MAX_VALUE;
54         imin=-1; // initializare aiurea ...
55         for(i=1;i<=n;i++) // mai bine cu heapMIN ...
56             if(g[i]==1) // i=frunza
57                 if(nde[i]+1<min) // ... aici este testul CORECT ... !!!
58                 {
59                     min=nde[i]+1;
60                     imin=i;
61                 }
62
63         timin=tata(imin);
64         g[imin]--; g[timin]--;
65         ca[timin]=ca[timin]+ca[imin]+nde[imin]+1;
66         nde[timin]=nde[timin]+nde[imin]+1; // nr descendenti eliminati
67     } // elimin()

68
69     public static void main(String[] args) throws IOException
70     {
71         int k,senat=0,cost=0;
72         st=new StreamTokenizer(new BufferedReader(new FileReader("cezar20.in")));
73         out=new PrintWriter(new BufferedWriter(new FileWriter("cezar.out")));
74
75         citire();
76
77         for(k=1;k<=n-1-ns;k++) eliminm();
78
79         for(k=1;k<=n;k++)
80             if(g[k]>0)
81             {
82                 cost+=ca[k];
83                 senat=k;
84             }
85
86         System.out.println("\n"+cost+" "+senat);
87         out.println(cost+" "+senat);
88         out.close();
89         //afisv(g,1,n);
90     } // main

```

```
91 } // class
```

Varianta 2:

Listing 13.2.4: cezar2.java

```

1 import java.io.*;           // ok toate teste
2 class cezar2                // cost initial = 1 pentru toate nodurile ...
3 {
4     static StreamTokenizer st;
5     static PrintWriter out;
6
7     static int n, ns,s=0;
8
9     static int[] v1 = new int[10001];
10    static int[] v2 = new int[10001];
11    static int[] g = new int[10001];          // gradele
12    static int[] ca = new int[10001];         // ca[k]=costul acumulat in nodul k !!!
13
14    static void afisv(int[] v,int i1, int i2)
15    {
16        int i;
17        for(i=i1;i<=i2;i++)
18        {
19            System.out.print(v[i]+" ");
20            if(i%50==0) System.out.println();
21        }
22        System.out.println();
23    }
24
25    static void citire() throws IOException
26    {
27        int i,j,k;
28
29        st.nextToken(); n=(int)st.nval;
30        st.nextToken(); ns=(int)st.nval;
31
32        for(i=1;i<=n;i++)                  // curatenie ...
33        {
34            g[i]=0;
35            ca[i]=1;                      // cost initial in nodul i
36        }
37
38        for(k=1;k<=n-1;k++)
39        {
40            st.nextToken(); i=(int)st.nval;
41            st.nextToken(); j=(int)st.nval;
42
43            v1[k]=i;
44            v2[k]=j;
45            g[i]++; g[j]++;
46        }
47
48        //afisv(v1,1,n-1);
49        //afisv(v2,1,n-1);
50        //afisv(g,1,n);
51    } // citire(...)

52
53    static int tata(int i) // mai bine cu liste de adiacenta ...
54    {
55        int k,t;
56        t=-1;                   // initializarea aiurea ...
57        for(k=1;k<=n-1;k++)    // este mai bine cu lista de adiacenta ?
58        {
59            if( (v1[k]==i) && (g[v2[k]]>0)) {t=v2[k]; break;}
60            else
61                if( (v2[k]==i) && (g[v1[k]]>0)) {t=v1[k]; break;}
62        }
63        return t;
64    }

65    static void eliminm() // frunze(g=1) cu cost=min
66    {
67        int i, imin, timin;
```

```

69     int min;
70
71     min=Integer.MAX_VALUE;
72     imin=-1;           // initializare aiurea ...
73     for(i=1;i<=n;i++) // mai bine cu heapMIN ...
74         if(g[i]==1)    // i=frunza
75             if(ca[i]<min) // cost acumulat
76             {
77                 min=ca[i];
78                 imin=i;
79             }
80
81     timin=tata(imin);
82
83     g[imin]--; g[timin]--;
84
85     ca[timin]=ca[timin]+min;
86     s+=min;
87
88     //System.out.println(" Elimin nodul "+imin+
89     //                      " timin = "+timin+" ca = "+ca[timin]);
90 } // elimin()
91
92 public static void main(String[] args) throws IOException
93 {
94     int k,senat=0;
95     st=new StreamTokenizer(new BufferedReader(new FileReader("cezar20.in")));
96     out=new PrintWriter(new BufferedWriter(new FileWriter("cezar.out")));
97
98     citire();
99
100    for(k=1;k<=n-1-ns;k++)
101    {
102        eliminm();
103    }
104
105    //afisv(c,l,n);
106
107    for(k=1;k<=n;k++)
108        if(g[k]>0)
109        {
110            senat=k;
111            break;
112        }
113
114    System.out.println("\n"+s+" "+senat);
115
116    out.println(s+" "+senat);
117    out.close();
118 } // main
119 } // class

```

Capitolul 14

OJI 2006

14.1 Graf

Victor Manz

Se știe că într-un graf neorientat conex, între oricare două vârfuri există cel puțin un lanț iar lungimea unui lanț este egală cu numărul muchiilor care-l compun. Definim noțiunea *lanț optim între două vârfuri X și Y* ca fiind un lanț de lungime minimă care are ca extremități vârfurile X și Y. Este evident că între oricare două vârfuri ale unui graf conex vom avea unul sau mai multe lanțuri optime, depinzând de configurația grafului.

Cerință:

Fiind dat un graf neorientat conex cu N vârfuri etichetate cu numerele de ordine 1,2, ...,N și două vârfuri ale sale notate X și Y ($1 \leq X, Y \leq N, X \neq Y$), se cere să scrieți un program care determină vârfurile care aparțin tuturor lanțurilor optime dintre X și Y.

Date de intrare:

Fișierul **graf.in** conține

- pe prima linie patru numere naturale reprezentând: N (numărul de vârfuri ale grafului), M (numărul de muchii), X și Y (cu semnificația din enunț).

- pe următoarele M linii câte două numere naturale nenule A_i, B_i ($1 \leq A_i, B_i \leq N, A_i \neq B_i$, pentru $1 \leq i \leq M$) fiecare dintre aceste perechi de numere reprezentând câte o muchie din graf.

Date de ieșire

Fișierul **graf.out** va conține

- pe prima linie, numărul de vârfuri comune tuturor lanțurilor optime dintre X și Y;
- pe a doua linie, numerele corespunzătoare etichetelor acestor vârfuri, dispuse în ordine crescătoare; între două numere consecutive de pe această linie se va afla câte un spațiu.

Restricții

- $2 \leq N \leq 7500; 1 \leq M \leq 14000$
- pentru 50% din teste $N \leq 200$

Exemple

graf.in	graf.out	graf.in	graf.out
6 7 1 4	3	3 2 1 3	2
1 2	1 4 5	1 2	1 3
1 3		3 1	
1 6			
2 5			
3 5			
5 6			
5 4			

Timp maxim de execuție/test: 1 secundă.

14.1.1 Indicații de rezolvare

Soluția oficială

I) Dacă definim distanța între două vârfuri ale unui graf neorientat ca fiind lungimea celui mai scurt lanț dintre lanțurile care au drept capete vârfurile, atunci putem să observăm că un vârf oarecare Z se află pe un lanț de lungime minimă dintre X și Y dacă și numai dacă $d(X, Z) + d(Z, Y) = d(X, Y)$, pentru cazul în care considerăm lungimea lanțului ca fiind numărul muchiilor și $d(X, Z) + d(Z, Y) = d(X, Y) + 1$, pentru cazul în care considerăm lungimea ca fiind numărul vârfurilor.

Stabilim prin câte o *parcursere în lățime* distanțele tuturor vârfurilor față de X și respectiv Y (capetele *lanțului*, citite din fișier). Vedem care dintre vârfurile ce aparțin cel puțin unui lanț de lungime minimă între X și Y au proprietatea că sunt singurele aflate la o anumită distanță de X . Acestea sunt vârfurile care aparțin tuturor lanțurilor de lungime minimă dintre X și Y .

Algoritmul are complexitate $O(n + m)$.

II) Facem o parcursere în lățime din X și o parcursere în lățime pornind din Y , în urma cărora determinăm pentru fiecare vârf z distanța dintre X și z , și Y și z - notate $d(X, z)$ și respectiv $d(Y, z)$ și numărul de drumuri optime dintre X și z , notat $nr(X, z)$ și dintre Y și z - $nr(Y, z)$. Un vârf z are proprietatea de a apartine tuturor drumurilor optime dintre X și Y dacă și numai dacă:

$$d(X, z) + d(Y, z) = d(X, Y) \text{ și } nr(X, z) * nr(Y, z) = nr(X, y).$$

Această soluție este însă dificil de implementat pentru grafuri cu un numar mare de noduri - se poate ajunge la operații cu numere mari;

III) Calculăm lungimea minimă a unui lanț de la X la Y . Eliminam apoi succesiv câte un nod din arbore (cu excepția nodurilor X și Y) și recalcăm lungimea minimă a unui lanț de la X la Y . Dacă această lungime diferă față de cea inițială, rezultă că toate lanțurile de lungime minimă trec prin nodul eliminat. Soluție de complexitate $O(n(n + m))$ care rezolvă corect 50% din teste.

IV) Se pot da și solutii bazate pe backtracking, de exemplu:

- se determină (printr-o parcursere în lățime a grafului) numărul de vârfuri aflate pe lanțul de lungime minimă dintre X și Y ; fie K numărul de vârfuri "intermediare" (fară X și Y);

- generăm toate lanțurile de lungime minimă dintre X și Y (inclusiv capetele) (algoritm de generare a aranjamentelor de n luate câte K cu verificările corespunzatoare de adiacență) contorizăm pentru fiecare vârf numărul de lanțuri în care apare;

- vârfurile care formează soluția vor avea contorul egal cu contorii vârfurilor X și Y .

În funcție de implementare și de optimizările aduse algoritmilor de backtracking, programele bazate pe astfel de soluții pot primi maxim 40 puncte.

14.1.2 Cod sursă

Listing 14.1.1: GRAF40.cpp

```

1 #include <iostream>
2 #include <string.h>
3
4 using namespace std;
5
6 #define nmax 220
7
8 char a[nmax][nmax];
9 int s[nmax+1], ct[nmax+1], n, k, m, lung, coada[nmax+1],
10     beginn, endd, v1, v2, dist[nmax+1];
11
12 void in(int val)
13 {coada[endd]=val; endd++;}
14
15 int out()
16 {beginn++; return coada[beginn-1];}
17 int isempty()
18 {return beginn==endd;}
19
20
21 void parc()
22 {int i;
23     int vf = out();
24 //   cout<<vf<<' ';
25     for(i=1;i<=n;i++)
26         if (a[vf][i]==1 && ct[i]==0)

```

```

27         { dist[i]=dist[vf]+1;
28         ct[i]++;in(i);
29     }
30     if (!isempty())
31     parc();
32 }
33
34 void print()
35 { int i;
36     for (i=0;i<=k;i++)
37     { //cout<<s[i]<<' ';
38     ct[s[i]]++;
39     }
40 // cout<<v2<<endl;
41 }
42
43 int valid(int p)
44 { int i;
45     if(a[s[p]][s[p-1]]==0) return 0;
46     for(i=0;i<p;i++)
47     if (s[p]==s[i]) return 0;
48     if (p==k && a[s[p]][v2]==0) return 0;
49     return 1;
50 }
51 void back(int p)
52 { int i;
53     for (i=1;i<=n;i++)
54     {s[p]=i;
55     if (valid(p))
56     if (p==k)
57     print();
58     else
59     back(p+1);
60     }
61 }
62
63 int main()
64 { ifstream f("graf.in");
65     int i,i1,i2;
66     f>>n>>m>>v1>>v2;
67     for (i=1;i<=m;i++)
68     { f>>i2>>i1;
69     a[i1][i2]=a[i2][i1]=1;
70     }
71     in(v1);
72     ct[v1]=1;
73     parc();
74 // for (i=1;i<=n;i++)
75 // cout<<pred[i]<<' ';
76 // cout<<endl;
77     k=dist[v2];
78 // cout<<k<<endl;
79     k--;
80     memset(ct,0,nmax*sizeof(int));
81     s[0]=v1;
82     back(1);
83     ofstream fo("graf.out");
84     int kt=0;
85     ct[v2]=ct[v1];
86     for (i=1;i<=n;i++)
87     if(ct[v1]==ct[i])
88     kt++;
89     fo<<kt<<endl;
90     for (i=1;i<=n;i++)
91     if(ct[v1]==ct[i])
92     fo<<i<<' ';
93     return 0;
94 }

```

Listing 14.1.2: graf100.cpp

```

1 #include <fstream>
2 #include <cstring>
3
4 using namespace std;

```

```

5
6 const int nmax = 7501;
7
8 struct node { int x; node * next;};
9
10 typedef int sir[nmax];
11 typedef node* nod;
12
13 nod v[nmax];
14 int *dx, *dy, *num, *vertex, *at, l[nmax];
15 int n,m,x,y,i,j,xx,yy,nr;
16
17 nod p;
18
19 ifstream fi("graf.in");
20
21 void bfs (int x, int d[])
22 { int tail, ttail, head, i;
23   nod p,q;
24   memset(d,(n+1)*sizeof(int),0);
25   head=1; tail=1; ttail=1;
26   l[1]=x; d[x]=1;
27   do
28   { for (i=head; i<=tail; i++)
29     {
30       p = v[l[i]];
31       while (p!=NULL)
32       { if (d[p->x]==0)
33       { ttail++;
34         l[ttail] = p->x;
35         d[p->x] = d[l[i]]+1;
36       }
37       p = p->next;
38     }
39   }
40   if (tail==ttail) break;
41   head = tail+1;
42   tail = ttail;
43 }
44 while (1);
45 }
46
47 int main()
48 {
49   fi>>n>>m>>x>>y;
50   for (i=1;i<=m;i++)
51   { fi>>xx>>yy;
52     p = new node;
53     p->x = yy;
54     p->next = v[xx];
55     v[xx] = p;
56     p = new node;
57     p->x = xx;
58     p->next = v[yy];
59     v[yy] = p;
60   }
61
62 dx = new int[n+1];
63 bfs(x, dx);
64 dy = new int[n+1];
65 bfs(y,dy);
66 num = new int[n+1];
67 for (i=0;i<=n;i++)
68   num[i]=0;
69
70 vertex = new int[n+1];
71
72 for (i=0;i<=n;i++)
73   vertex[i]=0;
74
75 for (i=1;i<=n;i++)
76   if (dx[i] + dy[i] == dx[y]+1)
77   { num[dx[i]]++;
78     vertex[dx[i]] = i;
79   }
80

```

```

81 at = new int[n+1];
82 for (i=0;i<=n;i++)
83     at[i]=0;
84
85 nr = 0;
86 for (i=1;i<=n;i++)
87     if (num[i]==1)
88         { nr++;
89          at[vertex[i]]=1;
90         }
91
92 ofstream fo("graf.out");
93 fo<<nr<<endl;
94 for (i=1;i<=n;i++)
95     if (at[i]==1)
96         fo<<i<<' ';
97
98 return 0;
99 }
```

14.1.3 Rezolvare detaliată

Varianta 1:

Listing 14.1.3: graf1.java

```

1 import java.io.*;
2 class graf1           // cu vectori de muchii, sortati (heapsort)
3 {
4     static StreamTokenizer st;
5     static PrintWriter out;
6
7     static int n,m,x,y,nv; // nv = nr varfuri (solutie)
8
9     static int[] v1;    // v1[k] = v1 muchia k
10    static int[] v2;   // v2[k] = v2 muchia k
11    static int[] q;    // coada
12
13    static int[] fdx;  // fdx[k] = frecventa distantei k fata de x
14    static int[] vdm; // vdm[i] = 1 daca i se afla pe un drum minim x...y
15
16    static int[] dx;   // dx[i] = dist(x,i)
17    static int[] dy;   // dy[i] = dist(i,y)
18
19    static int[] ai;   // ai[k] = adresa inceput lista adiacenta pentru k
20    static int[] ne;   // ne[k] = nr elemente din lista de adiacenta a lui k
21
22    static void afisv(int[] v,int i1, int i2)
23    {
24        int i;
25        for(i=i1;i<=i2;i++) System.out.print(v[i]+" ");
26        System.out.println();
27    }
28
29    static void citire() throws IOException
30    {
31        int i,j,k;
32
33        st.nextToken(); n=(int)st.nval;
34        st.nextToken(); m=(int)st.nval;
35        st.nextToken(); x=(int)st.nval;
36        st.nextToken(); y=(int)st.nval;
37
38        v1=new int[2*m+1];
39        v2=new int[2*m+1];
40
41        q=new int[n+1];
42
43        fdx=new int[n+1];
44        vdm=new int[n+1];
45
46        dx=new int[n+1];
47        dy=new int[n+1];
```

```

48
49     ai=new int[2*m+1];
50     ne=new int[n+1];
51
52     for(k=1;k<=m;k++)
53     {
54         st.nextToken(); i=(int)st.nval;
55         st.nextToken(); j=(int)st.nval;
56
57         v1[2*k-1]=i; v2[2*k-1]=j;
58         v1[2*k]=j; v2[2*k]=i;
59     }
60 }
61
62 static void calculd(int[] d, int nods) // nodstart
63 {
64     int ic, sc, k, z;
65
66     for(k=1;k<=n;k++) d[k]=-1;
67
68     q[0]=nods;
69     d[nods]=0;
70
71     ic=0;
72     sc=1;
73
74     while(ic != sc)
75     {
76         z=q[ic];
77         q[ic]=0;
78         ic++;
79
80         for(k=ai[z]; k<=ai[z]+ne[z]-1; k++)
81         {
82             if(d[v2[k]] == -1)
83             {
84                 q[sc]=v2[k];
85                 sc++;
86                 d[v2[k]]=1+d[z];
87             } // if
88         } // for
89     } // while
90 } // calculd()

92
93 static void hs(int[] x, int[] y, int n)
94 {
95     int t,f,fs,fd,aux,k;
96
97     // construiesc heap
98
99     for(k=2;k<=n;k++)
100    {
101        // urc ...
102        f=k;
103        t=f/2;
104        while(t>0)
105        {
106            if(x[t]<x[f])
107            {
108                aux=x[t]; x[t]=x[f]; x[f]=aux;
109                aux=y[t]; y[t]=y[f]; y[f]=aux;
110                f=t;
111                t=f/2;
112            }
113            else break;
114        } //while
115    }
116
117    // sortez: 1) permut varaf<-->ultimul; 2) cobor ...
118
119    for(k=n-1;k>=1;k--) // k = dim heap dupa x[1]<-->x[k+1]
120    {
121        // max <--> la coada ...
122        aux=x[1]; x[1]=x[k+1]; x[k+1]=aux;
123        aux=y[1]; y[1]=y[k+1]; y[k+1]=aux;

```

```

124
125      // cobor ... f=max(fs,fd) ... daca exista fd ...
126      t=1;
127      fs=2; fd=3;
128
129      f=fs;
130      if (fd<=k) if (x[fs]<x[fd]) f=fd;
131
132      while (f<=k)
133      {
134          if (x[t]<x[f])
135          {
136              aux=x[t]; x[t]=x[f]; x[f]=aux;
137              aux=y[t]; y[t]=y[f]; y[f]=aux;
138
139              t=f;
140
141              if (t>0x7fffffff/2) break; // fs > MAXINT ...
142
143              fs=2*t; fd=fs+1;
144              f=fs;
145
146              if (fs<0x7fffffff) // ca sa existe fd ...
147              if (fd<=k) if (x[fs]<x[fd]) f=fd;
148          }
149          else break;
150      } // while
151  } // for k
152 }
153
154 static void calcaine()
155 {
156     int i,j,k;
157
158     ai[v1[1]]=1;
159     ne[v1[1]]=1;
160
161     for(k=2;k<=2*m;k++)
162         if (v1[k]==v1[k-1]) ne[v1[k]]++;
163         else
164         {
165             ai[v1[k]]=k;
166             ne[v1[k]]=1;
167         }
168     }
169
170 static void solutia()
171 {
172     int i;
173
174     fdx=new int[n];           // distante = 0,1,...,n-1 !!!
175     vdm=new int[n+1];         // varf pe drum minim x...y
176
177     for(i=0;i<=n-1;i++) fdx[i]=0;
178     for(i=1;i<=n;i++) vdm[i]=0;
179
180     for(i=1;i<=n;i++)
181         if (dx[i]+dy[i] == dx[y])
182         {
183             fdx[dx[i]]++;
184             vdm[i]=1;
185         }
186
187     //afisv(fdx,0,n-1);
188     //afisv(vdm,1,n);
189
190     nv=0;
191     for(i=0;i<=n-1;i++) if (fdx[i]==1) nv++;
192
193     out.println(nv);
194
195     for(i=1;i<=n;i++)
196         if ((vdm[i]==1)&&(fdx[dx[i]]==1))
197         {
198             out.print(i+" ");
199         }

```

```

200
201     out.println();
202 }
203
204 public static void main(String[] args) throws IOException
205 {
206     st=new StreamTokenizer(new BufferedReader(new FileReader("graf.in")));
207     out=new PrintWriter(new BufferedWriter(new FileWriter("graf.out")));
208
209     citire();
210
211     //afisv(v1,1,2*m);
212     //afisv(v2,1,2*m);
213
214     hs(v1,v2,2*m);
215
216     //afisv(v1,1,2*m);
217     //afisv(v2,1,2*m);
218
219     calcaine();
220
221     //afisv(ai,1,n);
222     //afisv(ne,1,n);
223
224     q=new int[n];
225
226     calculd(dx,x);
227     calculd(dy,y);
228
229     //afisv(dx,1,n);
230     //afisv(dy,1,n);
231
232     solutia();
233
234     out.close();
235 } // main
236 } // class

```

Varianta 2:

Listing 14.1.4: graf2.java

```

1 import java.io.*;
2 class graf2
3 {                                     // lista de adiacenta - o singura citire din fisier !!!
4     static StreamTokenizer st;
5     static PrintWriter out;
6
7     static final int nmax=7500;
8     static final int mmax=14000;
9
10    static int n,m,x,y,nv;           // nv = nr varfuri (solutie)
11
12    static int[] v1;                // v1[k] = v1 muchia k
13    static int[] v2;                // v2[k] = v2 muchia k
14    static int[] q;                 // coada
15
16    static int[] fdx;              // fdx[k] = frecventa distantei k fata de x
17    static int[] vdm;              // vdm[i] = 1 daca i se afla pe un drum minim x...y
18
19    static int[] dx=new int[nmax+1]; // dx[i] = dist(x,i)
20    static int[] dy=new int[nmax+1]; // dy[i] = dist(i,y)
21
22    static int[] gi;               // gi[k] = grad interior nod k
23    static int[] ge;               // ge[k] = grad exterior nod k
24
25    static int[][] a;              // liste de adiacenta ...
26
27    static void afisv(int[] v,int i1, int i2)
28    {
29        int i;
30        for(i=i1;i<=i2;i++) System.out.print(v[i]+" ");
31        System.out.println();
32    }

```

```

33
34     static void citire() throws IOException
35     {
36         int i,j,k;
37
38         st.nextToken(); n=(int)st.nval;
39         st.nextToken(); m=(int)st.nval;
40         st.nextToken(); x=(int)st.nval;
41         st.nextToken(); y=(int)st.nval;
42
43         v1=new int[2*m+1];
44         v2=new int[2*m+1];
45
46         gi=new int[n+1];
47         ge=new int[n+1];
48
49         for(i=1;i<=n;i++) gi[i]=ge[i]=0;
50
51         for(k=1;k<=m;k++)
52         {
53             st.nextToken(); i=(int)st.nval;
54             st.nextToken(); j=(int)st.nval;
55             v1[k]=i; v2[k]=j;
56             gi[j]++; ge[i]++;
57         }
58
59         a=new int[n+1][1];           // apoi modific 1 ...
60
61         for(i=1;i<=n;i++)
62         {
63             a[i]=new int[gi[i]+ge[i]+1];    // am modificat ...
64             a[i][0]=0;
65         }
66
67         for(k=1;k<=m;k++)
68         {
69             i=v1[k]; j=v2[k];
70             a[i][0]++; a[j][0]++;
71
72             a[i][a[i][0]]=j;
73             a[j][a[j][0]]=i;
74         }
75     } // citire(...)

76
77     static void calculd(int[] d, int nods) // nodstart
78     {
79         int ic, sc, k, z;
80
81         for(k=1;k<=n;k++) d[k]=-1;
82
83         q[0]=nods;
84         d[nods]=0;
85
86         ic=0;
87         sc=1;
88
89         while(ic != sc)
90         {
91             z=q[ic];
92             q[ic]=0;
93             ic++;
94
95             for(k=1; k<=a[z][0]; k++)
96             {
97                 if(d[a[z][k]] == -1)
98                 {
99                     q[sc]=a[z][k];
100                     sc++;
101                     d[a[z][k]]=1+d[z];
102                 } // if
103             } // for
104         } // while
105     } // calculd()

106
107     static void solutia()
108     {

```

```

109     int i;
110
111     fdx=new int[n];           // distante = 0,1,...,n-1 !!!
112     vdm=new int[n+1];        // varf pe drum minim x...y
113
114     for(i=0;i<=n-1;i++) fdx[i]=0;
115     for(i=1;i<=n;i++) vdm[i]=0;
116
117     for(i=1;i<=n;i++)
118     {
119         if(dx[i]+dy[i] == dx[y])
120             {
121                 //System.out.println("i = "+i+"  dx[i] = "+dx[i]);
122                 fdx[dx[i]]++;
123                 vdm[i]=1;
124             }
125
126         //afisv(fdx,0,n-1);
127         //afisv(vdm,1,n);
128
129         nv=0;
130         for(i=0;i<=n-1;i++) if(fdx[i]==1) nv++;
131
132         out.println(nv);
133
134         if((vdm[i]==1)&&(fdx[dx[i]]==1))
135         {
136             out.print(i+" ");
137         }
138
139         out.println();
140     }
141
142     public static void main(String[] args) throws IOException
143     {
144         st=new StreamTokenizer(new BufferedReader(new FileReader("graf.in")));
145         out=new PrintWriter(new BufferedWriter(new FileWriter("graf.out")));
146
147         citire();
148
149         //afisv(v1,1,2*m);
150         //afisv(v2,1,2*m);
151
152         q=new int[n];
153
154         calculd(dx,x);
155         calculd(dy,y);
156
157         //afisv(dx,1,n);
158         //afisv(dy,1,n);
159
160         solutia();
161
162         out.close();
163     } // main
164 } // class

```

Varianta 3:

Listing 14.1.5: graf3.java

```

1 import java.io.*;
2 class graf3                // lista de adiacenta - doua citiri din fisier !!! !!!
3 {
4     static StreamTokenizer st;
5     static PrintWriter out;
6
7     static final int nmax=7500;
8     static final int mmax=14000;
9
10    static int n,m,x,y,nv;    // nv = nr varfuri (solutie)
11
12    static int[] q;          // coada
13

```

```

14     static int[] fdx;           // fdx[k] = frecventa distantei k fata de x
15     static int[] vdm;          // vdm[i] = 1 daca i se afla pe un drum minim x...y
16
17     static int[] dx=new int[nmax+1];    // dx[i] = dist(x,i)
18     static int[] dy=new int[nmax+1];    // dy[i] = dist(i,y)
19
20     static int[] gi;             // gi[k] = grad interior nod k
21     static int[] ge;             // ge[k] = grad exterior nod k
22
23     static int[][] a;           // liste de adiacenta ...
24
25     static void afisv(int[] v, int i1, int i2)
26     {
27         int i;
28         for(i=i1;i<=i2;i++) System.out.print(v[i]+" ");
29         System.out.println();
30     }
31
32     static void citire() throws IOException
33     {
34         int i,j,k;
35
36         st.nextToken(); n=(int)st.nval;
37         st.nextToken(); m=(int)st.nval;
38         st.nextToken(); x=(int)st.nval;
39         st.nextToken(); y=(int)st.nval;
40
41         gi=new int[n+1];
42         ge=new int[n+1];
43
44         for(i=1;i<=n;i++) gi[i]=ge[i]=0;
45
46         for(k=1;k<=m;k++)
47         {
48             st.nextToken(); i=(int)st.nval;
49             st.nextToken(); j=(int)st.nval;
50             gi[j]++; ge[i]++;
51         }
52
53         a=new int[n+1][1];           // apoi modific 1 ...
54
55         for(i=1;i<=n;i++)
56         {
57             a[i]=new int[gi[i]+ge[i]+1];
58             a[i][0]=0;
59         }
60
61         // "reset" pentru a doua citire din fisier ... !!!
62         st=new StreamTokenizer(new BufferedReader(new FileReader("graf.in")));
63
64         st.nextToken(); n=(int)st.nval;
65         st.nextToken(); m=(int)st.nval;
66         st.nextToken(); x=(int)st.nval;
67         st.nextToken(); y=(int)st.nval;
68
69         for(k=1;k<=m;k++)
70         {
71             st.nextToken(); i=(int)st.nval;
72             st.nextToken(); j=(int)st.nval;
73             a[i][0]++; a[j][0]++;
74
75             a[i][a[i][0]]=j;
76             a[j][a[j][0]]=i;
77         }
78     }// citire...
79
80     static void calculd(int[] d, int nods)      // nodstart
81     {
82         int ic, sc, k, z;
83
84         for(k=1;k<=n;k++) d[k]=-1;
85
86         q[0]=nods;
87         d[nods]=0;
88
89         ic=0;

```

```

90     sc=1;
91
92     while(ic != sc)
93     {
94         z=q[ic];
95         q[ic]=0;
96         ic++;
97
98         for(k=1; k<=a[z][0]; k++)
99         {
100             if(d[a[z][k]] == -1)
101             {
102                 q[sc]=a[z][k];
103                 sc++;
104                 d[a[z][k]]=1+d[z];
105             } // if
106         } // for
107     } // while
108 } // calculd()

109
110 static void solutia()
111 {
112     int i;
113
114     fdx=new int[n];           // distante = 0,1,...,n-1 !!!
115     vdm=new int[n+1];         // varf pe drum minim x...y
116
117     for(i=0;i<=n-1;i++) fdx[i]=0;
118     for(i=1;i<=n;i++) vdm[i]=0;
119
120     for(i=1;i<=n;i++)
121         if(dx[i]+dy[i] == dx[y])
122         {
123             //System.out.println("i = "+i+" dx[i] = "+dx[i]);
124             fdx[dx[i]]++;
125             vdm[i]=1;
126         }
127
128     //afisv(fdx,0,n-1);
129     //afisv(vdm,1,n);
130
131     nv=0;
132     for(i=0;i<=n-1;i++) if(fdx[i]==1) nv++;
133
134     out.println(nv);
135
136     for(i=1;i<=n;i++)
137         if((vdm[i]==1)&&(fdx[dx[i]]==1))
138         {
139             out.print(i+" ");
140         }
141
142     out.println();
143 }

144
145 public static void main(String[] args) throws IOException
146 {
147     st=new StreamTokenizer(new BufferedReader(new FileReader("graf.in")));
148     out=new PrintWriter(new BufferedWriter(new FileWriter("graf.out")));
149
150     citire();
151
152     //afisv(v1,1,2*m);
153     //afisv(v2,1,2*m);
154
155     q=new int[n];
156
157     calculd(dx,x);
158     calculd(dy,y);
159
160     //afisv(dx,1,n);
161     //afisv(dy,1,n);
162
163     solutia();
164
165     out.close();

```

166 }
167 }

14.2 Cifru

Stelian Ciurea

Un criptolog amator își propune să construiască o mașină de cifrat care să cripteze un text alcătuit din exact N simboluri distincte. Cifrarea se realizează prin permutarea simbolurilor ce formează textul.

Criptologul nostru dorește ca reconstituirea textului inițial să poată fi realizată trecând textul cifrat încă de $K - 1$ ori prin procedura de cifrare. Cu alte cuvinte, dacă textul rezultat din prima cifrare este cifrat încă o dată, rezultatul este cifrat din nou și aşa mai departe, plecând de la textul inițial și aplicând în total K operații de cifrare successive, trebuie să obțină textul inițial.

Criptologul nostru ar vrea să afle, cunoscând N și K , numărul de moduri distincte în care poate fi realizată mașina de cifrat. Două moduri de realizare a mașinii diferă dacă, există cel puțin un text în urma cifrării căruia, în cele două texte obținute există cel puțin o poziție în care se află simboluri diferite.

Cerință

Scrieți un program care determină restul împărțirii numărului de moduri distincte în care poate fi realizată mașina de cifrat la 19997.

Date de intrare

Fișierul **cifru.in** conține pe prima (și singura) linie, două valori numerice naturale separate printr-un spațiu, N și K (cu semnificația din enunț).

Date de ieșire

Fișierul **cifru.out** va conține pe prima linie, numărul de moduri distincte de realizare a mașinii de cifrat modulo 19997.

Restricții

- $1 \leq N \leq 2000$; $2 \leq K \leq 1000000000$
- pentru 30% din teste $N, K < 13$
- pentru 50% din teste $N, K \leq 100$

Exemple

cifru.in	cifru.out	cifru.in	cifru.out	cifru.in	cifru.out
3 3	3	9 6	11560	100 200	13767

Timp maxim de execuție/test: 2 secunde.

14.2.1 Indicații de rezolvare

Soluția oficială

I) Problema se poate reformula astfel: fie mulțimea $\{1, 2, \dots, n\}$ și fie σ o permutare a acestei mulțimi care îndeplinește condiția:

$$\underbrace{\sigma(\sigma(\sigma(\dots\sigma(x)\dots)))}_{k \text{ ori}} = x \quad \text{pentru orice } x \in \{1, 2, \dots, n\}$$

- Pentru început, vom da soluția pentru un caz particular al problemei și anume pentru $K = 3$: să notăm cu $f(n)$ numărul acestor permutări, și calculăm în câte moduri 1 (primul element al mulțimii) poate să revină pe prima poziție după K permutări. Există două posibilități și anume:

1) $\sigma(1) = 1$ și atunci evident că și după K permutări successive 1 va rămâne pe prima poziție; în acest caz, pentru celelalte $n - 1$ elemente ale mulțimii vor exista $f(n - 1)$ permutări care îndeplinesc condiția cerută;

2) elementul 1 formează un ciclu de lungime trei (evident cu încă alte două elemente din mulțime, fie acestea a și b . Astfel $\sigma(1) = a$, $\sigma(a) = b$, $\sigma(b) = 1$ și atunci aceste trei elemente revin pe pozițiile lor după trei permutări successive. În acest caz, cele două elemente a și b putem să le

alegem în $(n-1)(n-2) = A_n^2$ moduri, iar pentru restul de $n-3$ elemente ale multimii vom avea $f(n-3)$ permutări care corespund cerinței din enunț.

Având în vedere aceste două posibilități putem scrie următoarea relație de recurență:

$$f(n) = f(n-1) + (n-1)(n-2)f(n-3) \text{ și } f(n) = 1 \text{ pentru } n \leq 1$$

- Dacă K este număr prim, un raționament asemănător ne conduce la următoarea relație de recurență:

$$f(n) = f(n-1) + (n-1)(n-2)\dots(n-k+1)f(n-k) \text{ și } f(n) = 1 \text{ pentru } n \leq 1$$

- Dacă K nu este număr prim, atunci raționamentul anterior trebuie efectuat pentru fiecare divizor a lui K .

De exemplu, dacă $K = 6$, atunci elementul 1 poate să revină pe prima poziție după K permutări successive dacă

- $\sigma(1) = 1$;

- elementul 1 formează un ciclu de lungime 2 (atunci revine pe prima poziție după două permutări, după patru permutări și în final după șase permutări); celălalt element al acestui ciclu poate fi ales în A_n^1 iar pentru mulțimea formată din celelalte $n-2$ elemente avem $f(n-2)$ permutări;

- elementul 1 formează un ciclu de lungime 3 (atunci revine pe prima poziție după trei permutări și după șase permutări);

- elementul 1 formează un ciclu de lungime 6 (atunci revine pe prima poziție după șase permutări);

Obținem astfel pentru cazul $n = 6$, relația:

$$f(n) = f(n-1) + (n-1)f(n-2) + (n-1)(n-2)f(n-3) + (n-1)(n-2)\dots(n-5)f(n-6)$$

sau

$$f(n) = f(n-1) + A_n^1 f(n-2) + A_n^2 f(n-3) + A_n^5 f(n-6)$$

și

$$f(n) = 1 \text{ pentru } n \leq 1$$

Generalizând, putem scrie recurența

$$f(n) = f(n-1) + \sum_{d=2}^k A_{n-1}^{d-1} f(n-d)$$

pentru toții divizorii d ai lui K care sunt mai mici sau egali cu n .

Programul constă în implementarea acestei formule. Pentru a obține punctaj maxim, trebuie făcute câteva optimizări cum ar fi:

- calculul A_{n-1}^{d2} unde $d2$ este un divizor a lui K trebuie făcut plecând de la valoarea A_{n-1}^{d1} calculată în prealabil, unde $d1 < d2$ și $d1$ divizor al lui K ;

- reținerea valorilor calculate pentru funcția f într-un vector, pentru a evita calcularea în mod repetat a acestora.

Fără aceste optimizări, un program care implementează soluția descrisă primește 50 puncte.

II) O soluție bazată pe *backtracking*, care generează toate permutările și verifică pentru fiecare dacă îndeplinește condiția din enunț este corectă, dar se încadrează în timpul maxim de rulare/test doar pentru valori mici ale lui n și k . Pentru datele de test propuse, o astfel de soluție primește 20 puncte;

III) Se pot lua 30 puncte cu o astfel de soluție dacă se rulează programul în timpul concursului pentru toate combinațiile posibile sugerate de observația că

$$\bullet \text{pentru } 30\% \text{ din teste } N, K < 13$$

și se rețin rezultatele într-o matrice de constante.

14.2.2 Cod sursă

Listing 14.2.1: stelian.cpp

```

1 #include <iostream>
2 #include <fstream>
3
4 using namespace std;
5
6 const long prim = 19997;
7
8 ifstream fi("cifru.in");
9 ofstream fo("cifru.out");
10
11 long k,n;
12 long m[5005];
13
14 long f(int n)
15 { long d,i,rez=0,t;
16   if (n<k)
17     {if (n<=1) return 1;
18      if (m[n]!=0) return m[n];
19      long da = 1, p=1;
20      for (d=2;d<=n;d++)
21        if (k%d==0)
22          {for(i=n-da;i>=n-d+1;i--)
23            p = (p*i) % prim;
24            t = p*f(n-d);
25            rez = (rez + t) % prim;
26            da = d;
27          }
28        m[n] = (f(n-1) + rez) % prim;
29      }
30
31   if (m[n]!=0) return m[n];
32   long da = 1, p = 1;
33   for (d=2;d<=k;d++)
34     if (k%d==0)
35       {for(i=n-da;i>=n-d+1;i--)
36         p = (p*i) % prim;
37         t = (p*f(n-d)) % prim;
38         rez = (rez + t) % prim;
39         da = d;
40       }
41
42   return m[n] = (f(n-1) + rez) % prim;
43 }
44
45 int main()
46 {
47   fi>>n>>k;
48   fo<<f(n);
49   return 0;
50 }
```

14.2.3 Rezolvare detaliată

Varianta 1:

Listing 14.2.2: cifrul.java

```

1 import java.io.*;
2 class cifrul
3 {
4   static StreamTokenizer st;
5   static PrintWriter out;
6
7   static int[] x=new int[2001];
8   static int n, k;
9
10  public static void main(String[] args) throws IOException
11  {
12    st=new StreamTokenizer(new BufferedReader(new FileReader("cifru.in")));
13    out=new PrintWriter(new BufferedWriter(new FileWriter("cifru.out")));
14
15    citire();
16  }
17
18  void citire()
19  {
20    n=st.nextToken();
21    k=st.nextToken();
22
23    for(int i=1;i<=n;i++)
24      x[i]=st.nextToken();
25  }
26
27  void scrie()
28  {
29    for(int i=1;i<=n;i++)
30      out.print(x[i]);
31  }
32
33  void f(int n)
34  {
35    if(n<k)
36      out.print(x[n]);
37    else
38      f(n-1);
39
40    if(n>1)
41    {
42      int rez=0;
43
44      for(int i=k;i>n;i--)
45        rez=(rez+x[i])%19997;
46
47      f(n-1);
48
49      if(rez!=0)
50        out.print(rez);
51    }
52  }
53
54  void f()
55  {
56    f(n);
57  }
58
59  void f(int n,int k)
60  {
61    if(n<k)
62      out.print(x[n]);
63    else
64      f(n-1,k);
65
66    if(n>1)
67    {
68      int rez=0;
69
70      for(int i=k;i>n;i--)
71        rez=(rez+x[i])%19997;
72
73      f(n-1,k);
74
75      if(rez!=0)
76        out.print(rez);
77    }
78  }
79
80  void f(int n,int k,int rez)
81  {
82    if(n<k)
83      out.print(x[n]+rez);
84    else
85      f(n-1,k,rez);
86
87    if(n>1)
88    {
89      int rez2=0;
90
91      for(int i=k;i>n;i--)
92        rez2=(rez2+x[i])%19997;
93
94      f(n-1,k,rez+rez2);
95
96      if(rez2!=0)
97        out.print(rez2);
98    }
99  }
100 }
```

```
16         solutia();
17
18         out.println(x[n]);
19         out.close();
20     }
21
22     static void citire() throws IOException
23     {
24         st.nextToken(); n=(int)st.nval;
25         st.nextToken(); k=(int)st.nval;
26     }
27
28     static void solutia()
29     {
30         int i,j,a;
31
32         x[0]=1;
33         for(i=1;i<=n;i++)
34         {
35             j=1;
36             a=1;
37             while((j<=i)&&(j<=k))
38             {
39                 if(k%j==0) x[i]=(x[i]+a*x[i-j])%19997;
40                 a=(a*(i-j))%19997;
41                 j++;
42             } // while
43         } //for
44     } // solutia
45 } // class
```

Varianta 2:

Listing 14.2.3: cifru2.java

```
1 import java.io.*;
2 class cifru2
3 {
4     static StreamTokenizer st;
5     static PrintWriter out;
6
7     static int[] x=new int[5005];
8     static int n, k, rezsol, prim=19997;
9
10    public static void main(String[] args) throws IOException
11    {
12        st=new StreamTokenizer(new BufferedReader(new FileReader("cifru.in")));
13        out=new PrintWriter(new BufferedWriter(new FileWriter("cifru.out")));
14
15        citire();
16        solutia();
17
18        out.println(rezsol);
19        out.close();
20    }
21
22    static void citire() throws IOException
23    {
24        st.nextToken(); n=(int)st.nval;
25        st.nextToken(); k=(int)st.nval;
26    }
27
28    static int f(int n)
29    {
30        int d,i,rez=0,t;
31
32        if (n<k)
33        {
34            if (n<=1) return 1;
35            if (x[n]!=0) return x[n];
36
37            int da=1, p=1;
38            for (d=2;d<=n;d++)
39                if (k%d==0)
40                {
41                    if (da>p)
42                        rez+=x[d];
43                    da=p;
44                    p=d;
45                }
46            if (da>p)
47                rez+=x[d];
48        }
49        return rez;
50    }
51
52    static void solutia()
53    {
54        int i;
55
56        for (i=1;i<=n;i++)
57            if (f(i)>rezsol)
58                rezsol=f(i);
59
60        out.println(rezsol);
61    }
62
63    static void print()
64    {
65        int i;
66
67        for (i=1;i<=n;i++)
68            if (f(i)>rezsol)
69                rezsol=f(i);
70
71        out.println(rezsol);
72    }
73
74    static void print()
75    {
76        int i;
77
78        for (i=1;i<=n;i++)
79            if (f(i)>rezsol)
80                rezsol=f(i);
81
82        out.println(rezsol);
83    }
84
85    static void print()
86    {
87        int i;
88
89        for (i=1;i<=n;i++)
90            if (f(i)>rezsol)
91                rezsol=f(i);
92
93        out.println(rezsol);
94    }
95
96    static void print()
97    {
98        int i;
99
100       for (i=1;i<=n;i++)
101           if (f(i)>rezsol)
102               rezsol=f(i);
103
104       out.println(rezsol);
105   }
106 }
```

```
41         for(i=n-da;i>=n-d+1;i--) p=(p*i)%prim;
42         t=p*f(n-d);
43         rez=(rez+t)%prim;
44         da = d;
45     }
46     return x[n]=(f(n-1)+rez)%prim;
47 } // if
48
49 if(x[n]!=0) return x[n];
50
51 int da=1, p=1;
52 for(d=2;d<=k;d++)
53 if (k%d==0)
54 {
55     for(i=n-da;i>=n-d+1;i--) p=(p*i)%prim;
56     t=(p*f(n-d))%prim;
57     rez=(rez+t)%prim;
58     da=d;
59 }
60 return x[n]=(f(n-1)+rez)%prim;
61 } // f(...)
62
63 static void solutia()
64 {
65     rezsol=f(n);
66 } // solutia
67 } // class
```

Capitolul 15

OJI 2005

15.1 Lanț

Emanuela Cerchez

Ion este un lingvist pasionat. Recent el a descoperit un text scris într-o limbă necunoscută. Textul este scris pe mai multe linii și este format din cuvinte scrise cu litere mici din alfabetul latin, separate prin spații sau/și semne de punctuație (,; !? -).

Ion a fost frapat că există multe similitudini între cuvintele din text. Fiind foarte riguros, Ion definește similitudinea a două cuvinte după cum urmează.

Fie c_1 și c_2 două cuvinte. Cuvântul c_1 poate fi obținut din cuvântul c_2 printr-o succesiune de operații elementare. Operațiile elementare ce pot fi folosite sunt:

Operația	Efect	Exemplu
$move(c_1, c_2)$	Mută primul caracter din c_1 la sfârșitul lui c_2	Dacă $c_1 = "alba"$ și $c_2 = "neagra"$, după efectuarea operației c_1 va fi "lba", iar c_2 va fi "neagraa"
$insert(c_1, x)$	Inserează caracterul x la începutul lui c_1	Dacă $c_1 = "alba"$ și $x = "b"$, după efectuarea operației c_1 va fi "balba"
$delete(c_1)$	Șterge primul caracter din c_1	Dacă $c_1 = "alba"$ după efectuarea operației c_1 va fi "lba"

Definim similitudinea dintre c_1 și c_2 ca fiind numărul **minim** de operații $insert$ și $delete$ ce trebuie să fie executate pentru a transforma cuvântul c_1 în cuvântul c_2 (operațiile $move$ nu se numără).

Fie c_0 primul cuvânt din text. Începând cu c_0 putem construi lanțuri de k -similitudine.

Un lanț de k -similitudine este o succesiune de cuvinte distincte din text cu următoarele proprietăți:

- dacă cuvântul x apare în lanț înaintea cuvântului y , atunci prima apariție a lui x în text precedă prima apariție a lui y în text;
- dacă x și y sunt cuvinte consecutive în lanț (în ordinea $x \ y$), atunci similitudinea dintre x și y este k ;
- lanțul este maximal (adică nu putem adăuga încă un cuvânt la sfârșitul acestui lanț, astfel încât să fie respectate proprietățile precedente).

Cerință

Scrieți un program care să determine numărul de lanțuri de k -similitudine care încep cu c_0 .

Date de intrare

Fișierul de intrare **lant.in** conține pe prima linie valoarea k . Pe următoarele linii se află textul dat.

Date de ieșire

Fișierul de ieșire **lant.out** va conține o singură linie pe care va fi scris numărul de lanțuri de k -similitudine care încep cu c_0 .

Restricții

Lungimea unei linii din text nu depășește 1000 de caractere.

Lungimea unui cuvânt nu depășește 30 de caractere.

Numărul total de cuvinte ≤ 150 .

Pentru datele de test, numărul de lanțuri de k -similitudine care încep cu c_0 va fi $\leq 2.000.000.000$.

Exemplu

lant.in	lant.out	Explicație
5 ana are mere, banane, pere si castane.	6	Lanțurile de 5-similitudine care se pot forma sunt: ana are mere pere ana are mere ana are banane castane ana are si ana banane castane ana si

Timp maxim de execuție/test: 1 secundă.

15.1.1 Indicații de rezolvare

Soluția oficială

1. Se citește textul și se memorează cuvintele distincte din text într-un tablou c .

Fie nc numărul de cuvinte distincte determinate. Fiecare cuvânt este numerotat de la 0 la $nc - 1$ (indicii din tabloul c). Observați că numerotarea cuvintelor respectă ordinea primei apariții în text a cuvintelor.

2. Asociem problemei un graf orientat astfel:

- nodurile grafului sunt cuvintele distincte din text;
- există arc de la nodul i la nodul j ($i < j$) dacă numărul minim de operații **insert** și **delete** necesare pentru a transforma cuvântul $c[i]$ în cuvântul $c[j]$ este $\leq k$.

Observați că graful asociat problemei nu conține circuite.

Pentru a determina arcele grafului trebuie să rezolvăm următoarea subproblemă: să se determine numărul minim de operații **delete** și **insert** necesare pentru a transforma cuvântul x în cuvântul y .

Rezolvăm această subproblemă prin *programare dinamică*.

Fie $d[i][j] =$ numărul minim de operații **insert** și **delete** necesare pentru a transforma sufixul lui x care începe la poziția i în sufixul lui y care începe la poziția j .

Fie $n =$ lungimea cuvântului x și $m =$ lungimea cuvântului y .

- $d[n][j] = m - j$, pentru orice $j = 0, m$
- $d[i][m] = n - i$, pentru orice $i = 0, n$
- $d[i][j] = \min\{d[i + 1][j + 1],$ dacă $p[i] == q[j];$ - **move**
 $1 + d[i][j + 1] - \text{insert}$
 $1 + d[i + 1][j] - \text{delete}\}$

Soluția este $d[0][0]$.

3. Numărul de lanțuri de k -similitudine este egal cu numărul de drumuri care încep cu nodul 0 și se termină într-un nod terminal al grafului (nod cu gradul exterior 0).

Să notăm: $nr[i] =$ numărul de lanțuri de k -similitudine care încep cu cuvântul i .

Determinăm numărul folosind următoarea relație de recurență:

- $nr[i] = 1$, dacă nodul i este terminal
- $nr[i] = nr[i_1] + nr[i_2] + \dots + nr[i_k]$, unde i_1, i_2, \dots, i_k sunt noduri din graf cu proprietatea că există arc de la i la i_j , pentru $j = 1, k$.

15.1.2 Cod sursă

Listing 15.1.1: LANT.cpp

```

1 #include <iostream>
2 #include <string.h>
3 #include <stdlib.h>
4
5 using namespace std;
6
7 #define InFile "lant.in"
8 #define OutFile "lant.out"
9 #define LgMaxC 31

```

```

10 #define NrMaxC 151
11
12 typedef char Cuvant[LgMaxC];
13
14 Cuvant c[NrMaxC];
15 /* contine cuvintele distincte din text, in ordinea primei aparitii */
16
17 int nc, k;
18 int a[NrMaxC][NrMaxC];
19 int d[LgMaxC+1][LgMaxC+1];
20 long int nr[NrMaxC];
21 /* nr[i]= numarul de lanturi de k similitudine care incep cu cuvantul i */
22
23 void Citire();
24 void ConstrGraf();
25 void Numara(int);
26
27 int main()
28 {
29     int i;
30     ofstream fout(OutFile);
31     Citire();
32     ConstrGraf();
33     Numara(0);
34     fout<<nr[0]<<endl;
35     fout.close();
36     return 0;
37 }
38
39 void Adauga_Cuvant(char *p)
40 {
41     int i;
42     /* for (i=0; p[i]; i++)
43         if (!(p[i]>='a' && p[i]<='z')) cout<<"caractere ilegale "<<p<<endl; */
44     for (i=0; i<nc && strcmp(p,c[i]); i++);
45     if (i==nc) strcpy(c[nc++],p);
46     // if (nc>NrMaxC) cout<<"Prea multe cuvinte\n";
47 }
48
49 void Citire()
50 {
51     ifstream fin(InFile);
52     char s[1001], *p;
53     fin>>k; fin.get();
54     while (!fin.eof())
55     {
56         fin.getline(s,1001);
57         if (fin.good())
58             {p=strtok(s, " ,.;?!-");
59             while (p)
60                 {Adauga_Cuvant(p);
61                  p=strtok(NULL, " ,.;?!-"); }
62             }
63     }
64     fin.close();
65 }
66
67 int dist (char *p, char *q)
68 //determina distanta de editare dintre p si q
69 {
70     int n, m, i, j;
71     /*
72     d[i][j]=distanta de editare de la cuvantul p+i la cuvantul q+j
73     fie n= strlen(p) si m=strlen(q);
74     d[n][j]=m-j, pentru orice j=0,m
75     d[i][m]=n-i, pentru orice i=0,n
76     d[i][j]=min {d[i+1][j+1], daca p[i]==q[j]; - move
77                   1+d[i][j+1] - insert
78                   1+d[i+1][j] - delete
79     */
80     Solutia este d[0][0] */
81     n=strlen(p); m=strlen(q);
82     for (i=0; i<=n; i++) d[i][m]=n-i;
83     for (j=0; j<=m; j++) d[n][j]=m-j;
84     for (i=n-1; i>=0; i--)
85         for (j=m-1; j>=0; j--)
86             {d[i][j]=1+d[i][j+1];
87              if (d[i][j]>1+d[i+1][j])
88                  d[i][j]=1+d[i+1][j];
89              if (p[i]==q[j] && d[i][j]>d[i+1][j+1])
90                  d[i][j]=d[i+1][j+1];}

```

```

86     return d[0][0];
87 }
88
89 /*
90 void ConstrGraf()
91 {int i, j;
92   for (i=0; i<nc; i++)
93     {a[i]=(int *) malloc(sizeof(int));
94      a[i][0]=0;}
95   for (i=0; i<nc; i++)
96     for (j=i+1; j<nc; j++)
97       if (dist(c[i],c[j])<=k)
98       {
99         a[i][0]++;
100        a[i]=(int *)realloc(a[i],(a[i][0]+1)*sizeof(int));
101        a[i][a[i][0]]=j;
102      }
103 } */
104
105 void ConstrGraf()
106 {int i, j;
107   for (i=0; i<nc; i++)
108     for (j=i+1; j<nc; j++)
109       if (dist(c[i],c[j])<=k)
110       {
111         a[i][0]++;
112         a[i][a[i][0]]=j;
113       }
114 }
115
116 void Numara(int vf)
117 {
118   int i;
119   if (!a[vf][0]) {nr[vf]=1; return;}
120   long int s=0;
121   for (i=1; i<=a[vf][0]; i++)
122   {
123     if (nr[a[vf][i]]==0)
124       Numara(a[vf][i]);
125     s+=nr[a[vf][i]];
126     // if (s<0) cout<<"Prea multe lanturi\n";
127   }
128   nr[vf]=s;
129 }
```

15.1.3 Rezolvare detaliată

Prima variantă:

Listing 15.1.2: Lant1.java

```

1 import java.io.*;
2 class Lant1
3 {
4   static final int NMAX=150;
5   static String[] c=new String[NMAX+1];
6   static int k,nc=0,nlks=0;
7   static int[][] cs=new int[NMAX+1][NMAX+1]; // costuri editare
8   static int[] x=new int[NMAX+1]; // solutie in recursivitate
9
10  public static void main(String[] args) throws IOException
11  {
12    long t1,t2;
13    t1=System.currentTimeMillis();
14
15    citire();
16    solutie();
17    scrieSol();
18
19    t2=System.currentTimeMillis();
20    System.out.println("Timp = "+(t2-t1));
21  } // main()
22
```

```

23     static void citire() throws IOException
24     {
25         StreamTokenizer st=new StreamTokenizer(
26             new BufferedReader(new FileReader("lant.in")));
27
28         st.nextToken(); k=(int)st.nval;
29
30         st.whitespaceChars('/', '/');
31         st.whitespaceChars(':', ':');
32         st.whitespaceChars(';', ';');
33         st.whitespaceChars('.', '.');
34         st.whitespaceChars('!', '!');
35         st.whitespaceChars('?', '?');
36         st.whitespaceChars('-', '-');
37
38         while(StreamTokenizer.TT_EOF!=st.nextToken())
39         {
40             c[++nc]=st.sval;
41             if(c[nc]==null) {--nc; continue;} // "EOL" returneaza "null"
42             if(primaPozitie(c[nc])<nc) nc--;
43             else System.out.println(nc+" : "+c[nc]);
44         }
45         System.out.println();
46     }// citire()
47
48     static void solutie()
49     {
50         int i,j;
51
52         for(i=1;i<=nc-1;i++)
53             for(j=i+1;j<=nc;j++)
54                 cs[i][j]=dist(c[i],c[j]);
55
56         afism(cs); // matricea costurilor similitudine
57
58         x[1]=1; // lanturi care incep cu primul cuvant
59         for(j=2;j<=nc;j++)
60             if(cs[1][j]<=k) cuvant(2,j); // (pozitie,cuvant)
61
62         System.out.println();
63     }// solutie()
64
65     static int dist(String x, String y)
66     {
67         int i,j,n,m;
68         n=x.length();
69         m=y.length();
70         int[][] d=new int[n+1][m+1]; // este initializata cu zero
71         for(i=0;i<=n;i++) d[i][m]=n-i;
72         for(j=0;j<=m;j++) d[n][j]=m-j;
73
74         for(i=n-1;i>=0;i--)
75             for(j=m-1;j>=0;j--)
76             {
77                 d[i][j]=min(1+d[i][j+1],1+d[i+1][j]);
78                 if(x.charAt(i)==y.charAt(j)) d[i][j]=min(d[i][j],d[i+1][j+1]);
79             }
80         return d[0][0];
81     }// dist()
82
83     static void cuvant(int pozi, int cuvi) // plasez cuvi pe pozi si ...
84     {
85         int i,cuvj;
86         boolean esteMaximal=true;
87
88         x[pozi]=cuvi;
89         for(cuvj=cuvi+1;cuvj<=nc;cuvj++)
90             if(cs[cUVI][cuvj]<=k)
91             {
92                 esteMaximal=false;
93                 cuvant(pozi+1,cuvj);
94             }
95
96             if(esteMaximal)
97             {
98                 nlks++;

```

```

99         System.out.print(nlks+" : ");
100        for(i=1;i<=pozi;i++) System.out.print(c[x[i]]+" ");
101        System.out.println();
102    }
103 } // cuvant(...)

104 static int primaPozitie(String s) // in care apare s in text
105 {
106     int i,j,pp=-1,ns=s.length();
107     boolean ok=false;
108     for(i=1;i<nc;i++)
109     {
110         pp=i;
111         if(c[i].length()!=ns) continue; // difera prin lungime
112         ok=true;
113         for(j=0;j<ns;j++) // s[0] ... s[ns-1]
114             if(c[i].charAt(j)!=s.charAt(j)) { ok=false; break; }
115         if(ok) break;
116     }
117     if(ok) return pp; else return nc;
118 } // primaPozitie(...)

119 static int min(int a, int b) { if(a<b) return a; else return b; }

120 static void afism(int[][] a)
121 {
122     int i,j;
123     for(i=1;i<=nc;i++)
124     {
125         for(j=1;j<=nc;j++) System.out.print(a[i][j]+" ");
126         System.out.println();
127     }
128     System.out.println();
129 } // afism(...)

130 static void scrieSol() throws IOException
131 {
132     PrintWriter out = new PrintWriter(
133         new BufferedWriter(new FileWriter("lant.out")));
134     out.println(nlks);
135     out.close();
136 } // scrieSol()
137 } // class
138 -----
139 Lant.in      5
140           ana are mere,banane,
141           pere si castane.
142
143 Lant.out:   6
144
145 Ecran:
146
147 1 : ana
148 2 : are
149 3 : mere
150 4 : banane
151 5 : pere
152 6 : si
153 7 : castane
154
155 0 1 0 3 0 0 2      0 4 7 3 7 5 6      1 : ana are mere pere
156 0 0 2 2 2 0 2      0 0 3 5 3 5 6      2 : ana are banane castane
157 0 0 0 1 3 0 1      0 0 0 8 2 6 9      3 : ana are pere
158 0 0 0 0 1 0 4      0 0 0 0 8 8 5      4 : ana are si
159 0 0 0 0 0 0 1      0 0 0 0 0 6 9      5 : ana banane castane
160 0 0 0 0 0 0 1      0 0 0 0 0 0 7      6 : ana si
161
162 0 0 0 0 0 0 0      0 0 0 0 0 0 0      -----
163
164
165
166 -----

```

A doua variantă:

Listing 15.1.3: Lant2.java

```

1 import java.io.*;
2 class Lant2
3 {

```

```

4   static final int NMAX=150;
5   static String[] c=new String[NMAX+1];
6   static int k,nc=0;
7   static int[][] cs=new int[NMAX+1][NMAX+1];      // costuri editare
8   static int[] x=new int[NMAX+1];                  // x[i]=nr cuvinte care incep cu c[i]
9
10  public static void main(String[] args) throws IOException
11  {
12      long t1,t2;
13      t1=System.currentTimeMillis();
14
15      citire();
16      solutie();
17      scrieSol();
18
19      t2=System.currentTimeMillis();
20      System.out.println("Timp = "+(t2-t1));
21  } // main()
22
23  static void citire() throws IOException
24  {
25      StreamTokenizer st=new StreamTokenizer(
26          new BufferedReader(new FileReader("lant.in")));
27
28      st.nextToken(); k=(int)st.nval;
29
30      st.whitespaceChars('/', '/');
31      st.whitespaceChars(':', ':');
32      st.whitespaceChars(';', ';');
33      st.whitespaceChars('.', '.');
34      st.whitespaceChars('!', '!');
35      st.whitespaceChars('?', '?');
36      st.whitespaceChars('-', '-');
37
38      while(StreamTokenizer.TT_EOF!=st.nextToken())
39      {
40          c[++nc]=st.sval;
41          if(c[nc]==null) {--nc; continue;}      // "EOL" returneaza "null"
42          if(primaPozitie(c[nc])<nc) nc--;
43      }
44  } // citire()
45
46  static void solutie()
47  {
48      int i,j;
49      boolean esteTerminal;
50
51      for(i=1;i<=nc-1;i++)
52          for(j=i+1;j<=nc;j++)
53              cs[i][j]=dist(c[i],c[j]);
54
55      for(i=2;i<=nc;i++)
56      {
57          esteTerminal=true;
58          for(j=i+1;j<=nc;j++)
59              if(cs[i][j]<=k) {esteTerminal=false; break;}
60          if(esteTerminal) x[i]=1;
61      }
62
63      for(i=nc-1;i>=1;i--)
64          for(j=i+1;j<=nc;j++)
65              if(cs[i][j]<=k) x[i]+=x[j];
66  } // solutie()
67
68  static int dist(String x, String y)
69  {
70      int i,j,n,m;
71      n=x.length();
72      m=y.length();
73      int[][] d=new int[n+1][m+1]; // este initializata cu zero
74      for(i=0;i<=n;i++) d[i][m]=n-i;
75      for(j=0;j<=m;j++) d[n][j]=m-j;
76
77      for(i=n-1;i>=0;i--)
78          for(j=m-1;j>=0;j--)

```

```

79      {
80          d[i][j]=min(1+d[i][j+1],1+d[i+1][j]);
81          if(x.charAt(i)==y.charAt(j)) d[i][j]=min(d[i][j],d[i+1][j+1]);
82      }
83      return d[0][0];
84  } // dist(...)

85
86  static int primaPozitie(String s) // in care apare s in text
87  {
88      int i,j,pp=-1,ns=s.length();
89      boolean ok=false;
90      for(i=1;i<ns;i++)
91      {
92          pp=i;
93          if(c[i].length()!=ns) continue; // difera prin lungime
94          ok=true;
95          for(j=0;j<ns;j++) // s[0] ... s[ns-1]
96          if(c[i].charAt(j)!=s.charAt(j)) { ok=false; break; }
97          if(ok) break;
98      }
99      if(ok) return pp; else return ns;
100 } // primaPozitie(...)

101 static int min(int a, int b) { if(a<b) return a; else return b; }

102 static void scrieSol() throws IOException
103 {
104     PrintWriter out = new PrintWriter(
105         new BufferedWriter(new FileWriter("lant.out")));
106     out.println(x[1]);
107     out.close();
108 } // scrieSol()
109 } // class

```

15.2 Scara

Stelian Ciurea

Domnul G are de urcat o scară cu n trepte. În mod normal, la fiecare pas pe care îl face, el urcă o treaptă. Pe k dintre aceste trepte se află câte o sticlă cu un număr oarecare de decilitri de apă, fie acesta x . Dacă bea toată apa dintr-o astfel de sticlă, forță și mobilitatea lui G cresc, astfel încât, la următorul pas el poate urca până la x trepte, după care, dacă nu bea din nou ceva, revine la "normal". Sticlele cu apă nu costă nimic. Cantitatea de apă conținută de aceste sticle poate să difere de la o treaptă la alta.

Pe j trepte se află câte o sticlă cu băutură energizantă. Și pentru aceste sticle, cantitatea de băutură energizantă poate să difere de la o treaptă la alta. Să presupunem că într-una dintre aceste sticle avem y decilitri de băutură energizantă. Dacă bea q ($q \leq y$) decilitri dintr-o astfel de sticlă, la următorul pas G poate urca până la $2q$ trepte, după care și în acest caz, dacă nu bea din nou ceva, el revine la "normal". Însă băutura energizantă costă: pentru o cantitate de q decilitri consumată, G trebuie să plătească q lei grei.

Pot exista trepte pe care nu se află nici un pahar, dar și trepte pe care se află atât o sticlă cu apă cât și una cu băutură energizantă. În astfel de situații, nu are rost ca G să bea ambele băuturi deoarece efectul lor nu se cumulează; el poate alege să bea una dintre cele două băuturi sau poate să nu bea nimic.

Cerință

Determinați p , numărul minim de pași pe care trebuie să îi facă G pentru a urca scara, precum și suma minimă pe care trebuie să o cheltuiască G pentru a urca scara în p pași.

Date de intrare

Fișierul text de intrare **scara.in** conține:

- pe prima linie un număr natural n , reprezentând numărul total de trepte;
- pe cea de a doua linie un număr natural k , reprezentând numărul de trepte pe care se află sticle cu apă;
- pe fiecare dintre următoarele k linii câte două numere naturale separate printr-un spațiu, reprezentând numărul de ordine al treptei pe care se află o sticlă cu apă și respectiv cantitatea de apă din acea sticlă exprimată în decilitri;
- pe următoarea linie un număr natural j , reprezentând numărul de trepte pe care se află sticle cu băutură energizantă;

- pe fiecare dintre următoarele j linii căte două numere naturale separate printr-un spațiu, reprezentând numărul de ordine al treptei pe care se află o sticlă cu băutură energizantă și respectiv cantitatea de băutură energizantă din acea sticlă exprimată în decilitri.

Date de ieșire

Fisierul text de ieșire **scara.out** va conține o singură linie pe care vor fi scrise două numere naturale p și c separate printr-un spațiu, p reprezentând numărul minim de pași, iar c suma minimă cheltuită.

Restricții

$$n \leq 120$$

$$0 \leq k \leq n$$

$$0 \leq j \leq n$$

Cantitatea de apă aflată în oricare sticlă este $1 \leq x \leq 100$

Cantitatea de băutură energizantă aflată în oricare sticlă este $1 \leq y \leq 100$

Exemple

scara.in	scara.out	scara.in	scara.out
6	3 2	6	4 1
1		1	
1 2		1 2	
2		2	
4 1		4 1	
1 2		1 1	

Timp maxim de execuție: 1 secundă/test.

15.2.1 Indicații de rezolvare

Soluția oficială

Se construiește un graf orientat cu n noduri (care corespund celor n trepte) la arcele căruia se atașează costuri urmărind ca:

– între două noduri costul să fie cu atât mai mic cu cât nodurile sunt "mai depărtate"

– costul pentru salturi efectuate după ce se bea băutura energizantă să fie mai mare decât costul salturilor făcute după ce se bea apă între aceleași noduri, dar acest cost să nu depășească costul saltului între două noduri mai apropiate.

În calculul costurilor între două noduri am folosit formulele:

$$\text{cost}[i][i+1] = 999000$$

$$\text{cost}[i][j] = 1000000 - (j - i) \text{ pentru salturi când se bea apă } j > i$$

$$\text{cost}[i][j] = 1000000 - (j - i) + q \text{ pentru salturi când se bea energizantă } j > i, 1 \leq q \leq y$$

Pe acest graf se aplică apoi un algoritm de aflare a drumului minim de sursă unică (*Dijkstra*) sau având în vedere caracterul aciclic al grafului rezultat, algoritmul Dijkstra adaptat pentru astfel de grafuri.

15.2.2 Cod sursă

Listing 15.2.1: scara.cpp

```

1 #include <iostream>
2 #include <limits.h>
3
4
5 using namespace std;
6
7 const int nmax=121;
8
9 unsigned long cost[nmax+1][nmax+1];
10 unsigned long c[nmax+1], pd[nmax+1], d[nmax+1], minn;
11 int eng[nmax+1];
12
13 int main()
14 {ifstream fi("scara.in");
15  ofstream fo("scara.out");
16
17  int n,k,j,i,x,cx,i1,nod1,nod2,y,p,v;
18

```

```

19   fi>>n;
20   for (i=0;i<n+1;i++)
21     for (j=0;j<n+1;j++)
22       cost[i][j]=LONG_MAX;
23
24 //costul sariturii pe prima treapta
25 cost[0][1]=999000;
26
27 //costurile cand merge din 1 in 1
28 for (i=1;i<=n;i++)
29   cost[i][i+1]=999000;
30
31 fi>>k;
32 for (i=0;i<k;i++)
33 { fi>>p>>x;
34   if (! fi.good() || x>100) cout<<"apa incorrect\n";
35   nod1=p;
36   for (nod2=nod1+2,i1=2;i1<=x;i1++,nod2=nod2+1)
37     if (nod2<=n)
38       cost[nod1][nod2]=1000000-100*(nod2-nod1);
39 }
40
41 //costurile cand bea energizanta
42 fi>>j;
43 for (i=0;i<j;i++)
44 { fi>>p>>y;
45   if (! fi.good() || y>100) cout<<"baut incorrect\n";
46   eng[p]=y;
47 }
48
49 for (i=1;i<=n;i++)
50   if (eng[i]!=0)
51   {
52     nod1=i;
53     for (nod2=nod1+1,i1=1;i1<=eng[i];i1++,nod2=nod2+2)
54       if (nod2<=n)
55         if (cost[nod1][nod2]>1000000-100*(nod2-nod1)+i1)
56           cost[nod1][nod2]=1000000-100*(nod2-nod1)+i1;
57       if (nod2+1<=n)
58         if (cost[nod1][nod2+1]>1000000-100*(nod2-nod1+1)+i1)
59           cost[nod1][nod2+1]=1000000-100*(nod2-nod1+1)+i1;
60   }
61 }
62
63 /* cout<<endl;
64 for (i=0;i<n;i++)
65 { for (j=0;j<n+1;j++)
66   if (cost[i][j]<MAXLONG) fo<<cost[i][j]<<"  ";
67   else fo<<-1<<" ";
68   fo<<endl;
69 cout<<endl;
70 */
71
72 //Dijkstra
73 for (i=1;i<n+1;i++)
74 { c[i]=1;
75   d[i]=cost[0][i];
76 }
77 pd[1]=0;
78 for (j=0;j<n-1;j++)
79 { minn=LONG_MAX;
80 //   for (i=1;i<=n;i++)
81 //     fo<<d[i]<<' ';
82 //   fo<<endl;
83 //   for (i=1;i<=n;i++)
84 //     fo<<pd[i]<<' ';
85 //   fo<<endl;
86
87   for (i=1;i<n+1;i++)
88     if (c[i]!=0)
89       if (minn>d[i])
90         {minn=d[i];
91          v=i;
92        }
93       c[v]=0;
94 //   fo<<minn<<' '<<v<<endl;

```

```

95     for (i=1;i<n+1;i++)
96         if (c[i]!=0)
97             if (d[i]>d[v]+cost[v][i])
98                 {
99                     d[i]=d[v]+cost[v][i];
100                    pd[i]=v;
101                }
102            }
103
104    /*
105     cout<<d[n+1]<<endl;
106     for (i=0;i<=n;i++)
107         cout<<pd[i]<<' ';
108     cout<<endl;
109 */
110
111    i=n;
112    int ct=0,cst=0;
113    while (i!=0)
114        {j=pd[i];
115 //    cout<<j<<' ';
116        ct++;
117        if (cost[j][i]%100!=0)
118            cst = cst+cost[j][i]%100;
119        i=j;
120    }
121
122    fo<<ct<<' '<<cst<<endl;
123    fo.close();
124
125    return 0;
126 }
```

Listing 15.2.2: scaraDP.cpp

```

1 #include <stdio.h>
2
3 #define MAX_N 121
4
5 #define FIN "scara.in"
6 #define FOUT "scara.out"
7
8 #define INF 10000
9
10 #define cmp(a, b, c, d) (((a) < (c)) || ((a) == (c) && (b) < (d)))
11
12 int N, apa[MAX_N], suc[MAX_N];
13 int nrpasi[MAX_N], sum[MAX_N];
14
15 void citire(void);
16 void afisare(void);
17 void dinamic(void);
18
19 int main (void)
20 {
21     citire ();
22     dinamic ();
23     afisare();
24     return 0;
25 }
26
27 void citire ()
28 {
29     int i, j, K;
30     FILE * fin=fopen (FIN, "r");
31     fscanf (fin, "%d", &N);
32     fscanf (fin, "%d", &K);
33
34     for( K>0; K--)
35     {
36         fscanf(fin, "%d %d",&i, &j);
37         apa[i] = j;
38     }
39
40     fscanf(fin, "%d", &K);
```

```

41     for( ; K>0; K--)
42     {
43         fscanf(fin, "%d %d", &i, &j);
44         suc[i] = j;
45     }
46
47     fclose (fin);
48 }
49
50 void afisare ()
51 {
52     FILE * fout=fopen(FOUT, "w");
53     fprintf(fout, "%d %d\n", nrpasi[N], sum[N]);
54     fclose (fout);
55 }
56
57 void dinamic ()
58 {
59     int i, q;
60     nrpasi[1]=1;
61     for (i=2; i<=N; i++) nrpasi[i]=INF;
62
63     for (i=1; i<=N; i++)
64     {
65         // fac un pas "normal"
66         if(i<N && cmp(nrpasi[i]+1, sum[i], nrpasi[i+1], sum[i+1]))
67         {
68             nrpasi[i+1]=nrpasi[i]+1;
69             sum[i+1] = sum [i];
70         }
71
72         //daca este apa pe treapta i
73         if (apa[i])
74             for (q=1; q<=apa[i] && i+q <= N; q++)
75                 //beau q dl de apa
76                 if(cmp(nrpasi[i]+1,sum[i],nrpasi[i+q],sum[i+q]))
77                 {
78                     nrpasi[i+q]=nrpasi[i]+1;
79                     sum[i+q]=sum [i];
80                 }
81
82         // daca exista suc pe treapta i
83         if(suc[i])
84             for(q=1; q<=2*suc[i] && i+q<=N; q++)
85                 // beau q dl de suc
86                 if(cmp(nrpasi[i]+1,sum[q]+((q+1)/2),
87                         nrpasi[i+q],sum[i+q]))
88                 {
89                     nrpasi[i+q]=nrpasi[i]+1;
90                     sum[i+q]=sum[i]+((q+1)/2);
91                 }
92     }
93 }
```

15.2.3 Rezolvare detaliată

Varianta 1a:

Listing 15.2.3: Scarala.java

```

1 import java.io.*;      // numai nr pasi !
2 class Scarala
3 {
4     static int n;
5     static int[] a;      // apa
6     static int[] e;      // energizant
7     static int[] p;      // nr pasi
8     static int[] c;      // cost
9
10    public static void main(String[] args) throws IOException
11    {
12        long t1,t2;
13        t1=System.currentTimeMillis();
```

```

14
15     int k,j;
16     int i,treapta, cantitate;
17     int pmin;
18     StreamTokenizer st=new StreamTokenizer(
19         new BufferedReader(new FileReader("scara.in")));
20     PrintWriter out=new PrintWriter(
21         new BufferedWriter(new FileWriter("scara.out")));
22
23     st.nextToken(); n=(int)st.nval;
24     a=new int[n+1];
25     e=new int[n+1];
26     p=new int[n+1];
27
28     st.nextToken(); k=(int)st.nval;
29     for(i=1;i<=k;i++)
30     {
31         st.nextToken(); treapta=(int)st.nval;
32         st.nextToken(); cantitate=(int)st.nval;
33         a[treapta]=cantitate;
34     }
35
36     st.nextToken(); j=(int)st.nval;
37     for(i=1;i<=j;i++)
38     {
39         st.nextToken(); treapta=(int)st.nval;
40         st.nextToken(); cantitate=(int)st.nval;
41         e[treapta]=cantitate;
42     }
43
44     p[1]=1;
45     for(k=2;k<=n;k++)
46     {
47         pmin=k;
48         for(i=1;i<=k-1;i++)
49         {
50             pmin=min(pmin,p[i]+(k-i)); // in cel mai rau caz !
51             if(i+a[i]>=k) pmin=min(pmin,p[i]+1);
52             if(i+2*e[i]>=k) pmin=min(pmin,p[i]+1);
53         }
54         p[k]=pmin;
55     }
56     out.println(p[n]);
57     out.close();
58
59     t2=System.currentTimeMillis();
60     System.out.println("Timp = "+(t2-t1));
61 } // main
62
63     static int min(int a, int b)
64     {
65         if(a<b) return a; else return b;
66     }
67 } // class

```

Varianta 1b:

Listing 15.2.4: Scara1b.java

```

1 import java.io.*; // nr pasi si cost ... cu programare dinamica !
2 class Scara1b // traseul = !!!!!!!!!!!!!!!!!!!!!!!!
3 {
4     static int n;
5     static int[] a; // apa
6     static int[] e; // energizant
7     static int[] p; // nr pasi
8     static int[] c; // cost
9
10    public static void main(String[] args) throws IOException
11    {
12        int k,j;
13        int i,treapta, cantitate;
14        int pmin,cmin;
15        StreamTokenizer st=new StreamTokenizer(
16            new BufferedReader(new FileReader("scara.in")));
17        PrintWriter out=new PrintWriter(

```

```

18         new BufferedWriter(new FileWriter("scara.out")));
19
20     st.nextToken(); n=(int)st.nval;
21     a=new int[n+1];
22     e=new int[n+1];
23     p=new int[n+1];
24     c=new int[n+1];
25
26     st.nextToken(); k=(int)st.nval;
27     for(i=1;i<=k;i++)
28     {
29         st.nextToken(); treapta=(int)st.nval;
30         st.nextToken(); cantitate=(int)st.nval;
31         a[treapta]=cantitate;
32     }
33
34     st.nextToken(); j=(int)st.nval;
35     for(i=1;i<=j;i++)
36     {
37         st.nextToken(); treapta=(int)st.nval;
38         st.nextToken(); cantitate=(int)st.nval;
39         e[treapta]=cantitate;
40     }
41
42     p[1]=1;
43     c[1]=0;
44     for(k=2;k<=n;k++)
45     {
46         pmin=p[k-1]+1; // in cel mai rau caz !
47         cmin=c[k-1]; // in cel mai rau caz !
48
49         for(i=1;i<=k-1;i++)
50         {
51             pmin=min(pmin,p[i]+(k-i)); // in cel mai rau caz !
52
53             if(i+a[i]>=k)
54                 if(p[i]+1<pmin) {pmin=p[i]+1; cmin=c[i];} // apa=free !
55                 else if((p[i]+1==pmin)&&(a[i]>0)) cmin=min(cmin,c[i]);
56
57             if(i+2*e[i]>=k)
58                 if(p[i]+1<pmin) { pmin=p[i]+1; cmin=c[i]+(k-i+1)/2; }
59                 else if((p[i]+1==pmin)&&(e[i]>0)) cmin=min(cmin,c[i]+(k-i+1)/2);
60         }
61         p[k]=pmin;
62         c[k]=cmin;
63     }
64
65     out.println(p[n]+" "+c[n]);
66     out.close();
67 } // main
68
69 static int min(int a, int b)
70 {
71     if(a<b) return a; else return b;
72 }
73 } // class

```

Varianta 2a:

Listing 15.2.5: Scara2a.java

```

1 import java.io.*; // cu "propagare" numai nr pasi
2 class Scara2a
3 {
4     static int n;
5     static int[] a; // apa
6     static int[] e; // energizant
7     static int[] p; // nr pasi
8     static int[] c; // cost
9
10    public static void main(String[] args) throws IOException
11    {
12        long t1,t2;
13        t1=System.currentTimeMillis();
14
15        int k,j;

```

```

16     int i,treapta, cantitate;
17     int pmin,cmin;
18     StreamTokenizer st=new StreamTokenizer(
19         new BufferedReader(new FileReader("scara.in")));
20     PrintWriter out=new PrintWriter(
21         new BufferedWriter(new FileWriter("scara.out")));
22
23     st.nextToken(); n=(int)st.nval;
24     a=new int[n+1];
25     e=new int[n+1];
26     p=new int[n+1];
27     c=new int[n+1];
28
29     st.nextToken(); k=(int)st.nval;
30     for(i=1;i<=k;i++)
31     {
32         st.nextToken(); treapta=(int)st.nval;
33         st.nextToken(); cantitate=(int)st.nval;
34         a[treapta]=cantitate;
35     }
36
37     st.nextToken(); j=(int)st.nval;
38     for(i=1;i<=j;i++)
39     {
40         st.nextToken(); treapta=(int)st.nval;
41         st.nextToken(); cantitate=(int)st.nval;
42         e[treapta]=cantitate;
43     }
44
45     for(k=1;k<=n;k++) p[k]=k+1; // initializare pentru min !!
46
47     p[1]=1; c[0]=0;
48     for(k=1;k<=n-1;k++) // propag informatia de pe treapta k in sus !
49     {
50         if(a[k]>0)
51             for(i=k+1;i<=min(k+a[k],n);i++) // propag apa in sus !
52                 if(p[k]+1<p[i]) p[i]=p[k]+1;
53
54         for(i=k+a[k]+1;i<=n;i++) // pas cu pas !
55             if(p[i-1]+1<p[i]) p[i]=p[i-1]+1;
56
57         if(e[k]>0)
58             for(j=k+1;j<=min(k+2*e[k],n);j++) // propag energizant in sus
59                 if(p[k]+1<p[j]) p[j]=p[k]+1;
60
61         for(j=k+2*e[k]+1;j<=n;j++) // pas cu pas !
62             if(p[j-1]+1<p[j]) p[j]=p[j-1]+1;
63     }
64
65     out.println(p[n]+" "+c[n]); out.close();
66     t2=System.currentTimeMillis(); System.out.println("Timp = "+(t2-t1));
67 } // main
68
69     static int min(int a, int b)
70     {
71         if(a<b) return a; else return b;
72     }
73 } // class

```

Varianta 2b:

Listing 15.2.6: Scara2b.java

```

1 import java.io.*; // cu "propagare" pasi si cost
2 class Scara2b // traseul = ?????????!!!!!!!
3 {
4     static int n;
5     static int[] a; // apa
6     static int[] e; // energizant
7     static int[] p; // nr pasi
8     static int[] c; // cost
9
10    public static void main(String[] args) throws IOException
11    {
12        long t1,t2;
13        t1=System.currentTimeMillis();

```

```

14
15     int k,j;
16     int i,treapta, cantitate;
17     int pmin,cmin;
18     StreamTokenizer st=new StreamTokenizer(
19         new BufferedReader(new FileReader("scara.in")));
20     PrintWriter out=new PrintWriter(
21         new BufferedWriter(new FileWriter("scara.out")));
22
23     st.nextToken(); n=(int)st.nval;
24     a=new int[n+1];
25     e=new int[n+1];
26     p=new int[n+1];
27     c=new int[n+1];
28
29     st.nextToken(); k=(int)st.nval;
30     for(i=1;i<=k;i++)
31     {
32         st.nextToken(); treapta=(int)st.nval;
33         st.nextToken(); cantitate=(int)st.nval;
34         a[treapta]=cantitate;
35     }
36
37     st.nextToken(); j=(int)st.nval;
38     for(i=1;i<=j;i++)
39     {
40         st.nextToken(); treapta=(int)st.nval;
41         st.nextToken(); cantitate=(int)st.nval;
42         e[treapta]=cantitate;
43     }
44
45     for(k=1;k<=n;k++) p[k]=k+1; // initializare pentru min !!
46
47     p[1]=1; c[0]=0;
48     for(k=1;k<=n-1;k++) // propag informatia de pe treapta k in sus !
49     {
50         if(a[k]>0)
51             for(i=k+1;i<=min(k+a[k],n);i++) // propag apa in sus
52                 if(p[k]+1<=p[i]) { p[i]=p[k]+1; c[i]=c[k]; } //apa=free
53                 else
54                     if(p[k]+1==p[i]) c[i]=min(c[i],c[k]);
55
56         for(i=k+a[k]+1;i<=n;i++) // pas cu pas !
57             if(p[i-1]+1<=p[i]) { p[i]=p[i-1]+1; c[i]=c[i-1]; }
58             else
59                 if(p[i-1]+1==p[i]) c[i]=min(c[i],c[i-1]); // ???
60
61         if(e[k]>0)
62             for(j=k+1;j<=min(k+2*e[k],n);j++) // propag energizant in sus
63                 if(p[k]+1<=p[j]) { p[j]=p[k]+1; c[j]=c[k]+(j-k+1)/2; }
64                 else
65                     if(p[k]+1==p[j]) c[j]=min(c[j],c[k]+(j-k+1)/2);
66
67         for(j=k+2*e[k]+1;j<=n;j++) // pas cu pas !
68             if(p[j-1]+1<=p[j]) { p[j]=p[j-1]+1; c[j]=c[j-1]; }
69             else
70                 if(p[j-1]+1==p[j]) c[j]=min(c[j],c[j-1]);
71     }
72
73     out.println(p[n]+" "+c[n]); out.close();
74     t2=System.currentTimeMillis(); System.out.println("Timp = "+(t2-t1));
75 } // main
76
77 static int min(int a, int b)
78 {
79     if(a<b) return a; else return b;
80 }
81 } // class

```

Capitolul 16

OJI 2004

16.1 Moșia

Păcală a primit, aşa cum era învoiala, un petec de teren de pe moşia boierului. Terenul este împrejmuit complet cu segmente drepte de gard ce se sprijină la ambele capete de câte un par zdravăn. La o nouă prinsoare, Păcală ieșe iar în câștig și primește dreptul să strămute niște pari, unul câte unul, cum i-o fi voia, astfel încât să-și extindă suprafața de teren. Dar învoiala prevede că fiecare par poate fi mutat în orice direcție, dar nu pe o distanță mai mare decât o valoare dată (scrisă pe fiecare par) și fiecare segment de gard, fiind cam ţuubred, poate fi rotit și prelungit de la un singur capăt, celălalt rămânând nemîșcat.

Cunoscând pozițiile inițiale ale parilor și valoarea înscrișă pe fiecare par, se cere suprafața maximă cu care poate să-și extindă Păcală proprietatea. Se știe că parii sunt dați într-o ordine oarecare, pozițiile lor inițiale sunt date prin numere întregi de cel mult 3 cifre, distanțele pe care fiecare par poate fi deplasat sunt numere naturale strict pozitive și figura formată de terenul inițial este un poligon neconcov.

Date de intrare

Fișierul MOSIA.IN conține $n + 1$ linii cu următoarele valori:

n - numărul de pari

$x_1 \ y_1 \ d_1$ - coordonatele inițiale și distanța pe care poate fi mutat parul 1

$x_2 \ y_2 \ d_2$ - coordonatele inițiale și distanța pe care poate fi mutat parul 2

...

$x_n \ y_n \ d_n$ - coordonatele inițiale și distanța pe care poate fi mutat parul n

Date de ieșire

În fișierul MOSIA.OUT se scrie un număr real cu 4 zecimale ce reprezintă suprafața maximă cu care se poate mări moșia.

Restricții și observații:

$3 < N \leq 200$ număr natural

$-1000 < x_i, y_i < 1000$ numere întregi

$0 < d_i \leq 20$ numere întregi

poligonul neconcov se definește ca un poligon convex cu unele vârfuri coliniare
pozițiile parilor sunt date într-o ordine oarecare

poligonul obținut după mutarea parilor poate fi concav

pozițiile finale ale parilor nu sunt în mod obligatoriu numere naturale

Exemplu

Pentru fișierul de intrare

4		
-3	0	2
3	0	3
0	6	2
0	-6	6

se va scrie în fișierul de ieșire valoarea 30.0000

Explicație: prin mutarea parilor 1 și 2 cu câte 2 și respectiv 3 unități, se obține un teren având suprafața cu 30 de unități mai mare decât terenul inițial.

Timp limită de executare: 1 sec./test

16.1.1 *Indicații de rezolvare

16.1.2 Cod sursă

Listing 16.1.1: POLIGON.pas

```

1 {Solutia problemei MOSIA LUI PACALA - Rodica}
2 {Se ordoneaza punctele folosind alg. Hill pentru infasuratoare convexa
3 se calculeaza pentru fiecare par i aria d[i]*dist(par[i-1],par[i+1])/2
4 si se determina cu un alg. elementar de prog. dinamica secventa de pari
5 nealaturati care conduce la suma maxima a ariilor}
6 const maxi=251;fi='mosia.in';fo='mosia.out';
7 type punct=record x,y:longint end;
8 puncte=array[0..maxi+1]of punct;
9 sir=array[1..maxi]of integer;
10 var f:text; p:puncte;
11 s,o:sir;
12 n:integer;
13
14 procedure citire;
15 var i:integer;
16 begin
17     assign(f,fi);reset(f);
18     readln(f,n);
19     for i:=1 to n do readln(f,p[i].x,p[i].y,p[i].d);
20     for i:=1 to n do o[i]:=i;
21     close(f)
22 end;
23
24 procedure qsort(l, r: Integer);
25 var
26     i,j,mx,my,ax:punct;aux:punct;
27 begin
28     i:=l; j:=r;
29     my:=p[(l+r) div 2].y;
30     mx:=p[(l+r) div 2].x;
31     repeat
32         while (p[i].y<my)or((p[i].y=my)and(p[i].x<mx)) do inc(i);
33         while (p[j].y>my)or((p[j].y=my)and(p[j].x>mx)) do dec(j);
34         if i<=j then begin
35             aux:=p[i];p[i]:=p[j];p[j]:=aux;
36             ax:=o[i];o[i]:=o[j];o[j]:=ax;
37             inc(i);dec(j)
38         end;
39         until i>j;
40         if l<j then qsort(l, j);
41         if i<r then qsort(i, r);
42     end;
43
44 function sign(a,b,c:punct):shortint;
45 begin
46     if c.x*(a.y-b.y)+c.y*(b.x-a.x)+a.x*b.y-b.x*a.y>=0 then
47         sign:=1
48     else sign:=-1
49 end;
50
51 procedure convex;
52 var free:array[1..maxi]of boolean;
53     i,vf:integer;
54 begin
55     for i:=1 to n do free[i]:=true;
56     s[1]:=1;s[2]:=2;free[2]:=false;
57     vf:=2;
58     for i:=3 to n do begin
59         while sign(p[s[vf-1]],p[s[vf]],p[i])=-1 do
60             begin free[s[vf]]:=true;dec(vf) end;
61             inc(vf);s[vf]:=i;free[i]:=false
62         end;
63         for i:=n-1 downto 1 do
64             if free[i] then begin
65                 while sign(p[s[vf-1]],p[s[vf]],p[i])=-1 do dec(vf);

```

```

66           inc(vf);s[vf]:=i
67       end;
68   if vf-1<>n then begin n:=vf-1;writeln('Date eronate') end
69 end;
70
71 procedure ssort;
72 var i:integer;
73   p1:puncte;o1:sir;
74 begin
75   for i:=1 to n do begin
76     p1[i]:=p[s[i]];o1[i]:=o[s[i]]
77   end;
78   p:=p1;o:=o1
79 end;
80
81 function dist(a,b:punct):real;
82 begin
83   dist:=sqrt(sqr(a.x-b.x)+sqr(a.y-b.y))
84 end;
85
86 procedure dinamic;
87 var i,j,k:integer;sum:real;
88   ds:array[1..maxi] of real;
89   d:array[1..2,1..maxi] of record s:real;i:byte end;
90 begin
91   for i:=1 to n do ds[i]:=p[i].d/2*dist(p[i-1],p[i+1]);
92   d[1,n].s:=ds[n];d[1,n].i:=1;
93   d[1,n+1].s:=0;d[1,1].i:=0;
94   for i:=n-1 downto 2 do
95     if d[1,i+2].s+ds[i]>d[1,i+1].s then begin
96       d[1,i].s:=d[1,i+2].s+ds[i];d[1,i].i:=1
97     end
98     else begin
99       d[1,i].s:=d[1,i+1].s;d[1,i].i:=0
100    end;
101   d[1,1].s:=d[1,2].s;
102   d[2,n].s:=0;d[2,n].i:=0;
103   d[2,n+1].s:=0;
104   for i:=n-1 downto 1 do
105     if d[2,i+2].s+ds[i]>d[2,i+1].s then begin
106       d[2,i].s:=d[2,i+2].s+ds[i];d[2,i].i:=1
107     end
108     else begin
109       d[2,i].s:=d[2,i+1].s;d[2,i].i:=0
110     end;
111   if d[1,1].s>d[2,1].s then j:=1 else j:=2;
112   i:=1;k:=0;
113   while i<=n do
114     if d[j,i].i=0 then inc(i)
115     else begin inc(k);s[k]:=i;inc(i,2) end;
116     for i:=1 to k do write(o[s[i]],' ');
117     writeln;writeln(d[j,1].s:20:6);
118     assign(f,fo);rewrite(f);write(f,d[j,1].s:20:6);close(f)
119 end;
120
121 begin
122   citire;
123   qsort(1,n);
124   convex;
125   ssort;
126   p[0]:=p[n];p[n+1]:=p[1];
127   dinamic;
128 end.

```

16.1.3 *Rezolvare detaliată

16.2 Lanterna

Un agent secret are o hartă pe care sunt marcate N obiective militare. El se află, inițial, lângă obiectivul numerotat cu 1 (baza militară proprie) și trebuie să ajungă la obiectivul numerotat cu

N (baza militară inamică). În acest scop, el va folosi drumurile existente, fiecare drum legând 2 obiective distincte. Fiind o misiune secretă, deplasarea agentului va avea loc noaptea; de aceea, el are nevoie de o lanternă. Pentru aceasta, el are de ales între K tipuri de lanterne - o lanternă de tipul W ($1 \leq W \leq K$) are baterii care permit consumul a W wați, după consumul acestor wați, lanterna nu mai luminează. Din fericire, unele dintre obiective sunt baze militare prietene, astfel că, o dată ajuns acolo, el își poate reîncărca bateriile complet. Agentul trebuie să aibă grijă ca, înainte de a merge pe un drum între două obiective, cantitatea de wați pe care o mai poate consuma să fie mai mare sau egală cu cantitatea de wați pe care o va consuma pe drumul respectiv.

Cunoscând drumurile dintre obiective și, pentru fiecare drum, durata necesară parcurgerii drumului și numărul de wați consumați de lanternă, determinați tipul de lanternă cu numărul cel mai mic, astfel încât durata deplasării sa fie minimă (dintre toate tipurile de lanternă cu care se poate ajunge în timp minim la destinație, interesează lanternă cu consumul cel mai mic).

Date de intrare

Pe prima linie a fișierului **lanterna.in** se află numerele întregi N și K , separate printr-un spațiu. Pe următoarea linie se află N numere întregi din multimea $\{0, 1\}$. Dacă al i -lea număr este 1, aceasta înseamnă că obiectivul cu numărul i este o bază militară prietenă (adică agentul își poate reîncărca bateriile lanternei dacă ajunge la acest obiectiv); dacă numărul este 0, agentul nu își va putea reîncărca bateriile. Primul număr din linie este 1, iar ultimul este 0. Pe cea de-a treia linie a fișierului se află numărul M de drumuri dintre obiective. Fiecare din următoarele M linii conțin câte 4 numere întregi separate prin spații: a b T W , având semnificația că există un drum bidirectional între obiectivele a și b ($a \neq b$), care poate fi parcurs într-un timp T și cu un consum de W wați.

Date de ieșire

În fișierul **lanterna.out** se vor afișa două numere întregi, separate printr-un spațiu: T_{min} și W_{min} . T_{min} reprezentând durata minimă posibilă a deplasării de la obiectivul 1 la obiectivul N , iar W_{min} reprezintă tipul de lanternă cu numărul cel mai mic pentru care se obține acest timp.

Restricții și precizări

$$2 \leq N \leq 50$$

$$1 \leq K \leq 1000$$

$$1 \leq M \leq N * (N - 1) / 2$$

Între două obiective diferite poate exista maximum un drum direct.

Pentru fiecare drum, durata parcurgerii este un număr întreg între 1 și 100, iar numărul de wați consumați este un număr întreg între 0 și 1000.

Se garantează că există cel puțin un tip de lanternă pentru care deplasarea să fie posibilă.

Punctajul pentru un test se va acorda în felul următor:

- 30% : dacă este determinat corect T_{min}
- 100% : dacă sunt determinate corect atât T_{min} , cât și W_{min}

Exemplu:

lanterna.in	lanterna.out
7 10	
1 0 1 0 0 0 0	
7	
1 2 10 3	
1 4 5 5	
2 3 10 3	
4 3 15 1	
3 6 4 3	
6 5 2 2	
5 7 1 0	
	27 6

Timp maxim de executare: 1 secundă/test

16.2.1 *Indicații de rezolvare

16.2.2 Cod sursă

Listing 16.2.1: LANT2.pas

```

1 { Mugurel Ionut Andreica - Bucuresti, ROMANIA }
2
3 { Time Complexity : O(log(K) * (N*K)^3 ? }
4
5 { Cautare binara + Bellman Ford (cu coada) }
6
7 Program _lanterna_;
8
9 const filein = 'lanterna.in';
10    fileout = 'lanterna.out';
11    MAXN = 50;
12    MAXK = 1000;
13    infinit = 30000;
14    maxdim1 = 63;
15
16 type linie = array [0..MAXK] of integer;
17    plinie = ^linie;
18    pcoada = ^coada;
19    coada = record
20        i : byte;
21        k : integer;
22        urm : pcoada;
23    end;
24
25 var T, W : array [1..MAXN, 1..MAXN] of integer;
26    Tmin : array [1..MAXN] of plinie;
27    charge : array [1..MAXN] of boolean;
28    inq : array[1..MAXN, 0..MAXK] of boolean;
29    i, j, p, M, N, K, li, ls : integer;
30    TOK, bestK : integer;
31    cst, cfin, aux : pcoada;
32
33 procedure readdata;
34 begin
35 assign(input, filein);
36 reset(input);
37 read(N, K);
38
39 for i := 1 to N do
40 begin
41     read(p);
42     charge[i] := (p = 1);
43 end;
44
45 read(M);
46
47 for i := 1 to N do
48   for j := 1 to N do
49   begin
50       T[i,j] := infinit;
51       W[i,j] := MAXK + 1;
52   end;
53
54 for p := 1 to m do
55 begin
56     read(i, j);
57     read(T[i,j], W[i,j]);
58     T[j,i] := T[i,j];
59     W[j,i] := W[i,j];
60 end;
61
62 close(input);
63
64 for i := 1 to N do
65 begin
66     new(Tmin[i]);
67 end;
68 end;
69
70 procedure writedata;
71 begin
72 assign(output, fileout);
73 rewrite(output);
74 writeln(TOK, ' ', bestK);

```

```

75  close(output);
76  end;
77
78  function MinT(kmax : integer) : integer;
79  var i, j, k, p, q, tN : integer;
80
81  begin
82    for i := 1 to N do
83      for j := 0 to kmax do
84        begin
85          Tmin[i]^ [j] := infinit;
86        end;
87
88    Tmin[1]^ [0] := 0;
89    inq[1, 0] := true;
90
91    new(cst);
92    cst^.i := 1;
93    cst^.k := 0;
94    cst^.urm := nil;
95    cfin := cst;
96
97    tN := infinit;
98
99    while (cst <> nil) do
100      begin
101        i := cst^.i;
102        k := cst^.k;
103
104        if (Tmin[i]^ [k] < tN) then
105          for j := 1 to N do
106            if (T[i,j] < infinit) and (k + W[i,j] <= kmax) then
107              begin
108                p := Tmin[i]^ [k] + T[i,j];
109                q := k + W[i,j];
110
111                if (charge[j]) then
112                  q := 0;
113
114                if (p < Tmin[j]^ [q]) then
115                  begin
116                    Tmin[j]^ [q] := p;
117
118                    if (j = N) and (Tmin[j]^ [q] < tN) then
119                      tN := Tmin[j]^ [q];
120
121                    if (not inq[j, q]) and (Tmin[j]^ [q] < tN) then
122                      begin
123                        inq[j, q] := true;
124
125                        new(aux);
126                        aux^.i := j;
127                        aux^.k := q;
128                        aux^.urm := nil;
129                        cfin^.urm := aux;
130                        cfin := aux;
131                      end;
132                    end;
133                  end;
134
135                inq[i, k] := false;
136                aux := cst;
137                cst := cst^.urm;
138                dispose(aux);
139              end;
140
141    MinT := tN;
142  end;
143
144  begin
145    readdata;
146
147    TOK := MinT(K);
148    bestK := K;
149
150    li := 1; ls := K-1;

```

```
151 while (li <= ls) do
152   begin
153     p := (li + ls) shr 1;
154
155     if (MinT(p) = TOK) then
156       begin
157         bestK := p;
158         ls := p-1;
159       end
160     else
161       li := p+1;
162   end;
163
164 writedata;
165 end.
```

16.2.3 *Rezolvare detaliată

Capitolul 17

OJI 2003

17.1 Compus

La ultima expediție pe Marte a fost descoperit un compus organic necunoscut. Acest compus este acum studiat în laboratoarele NASA. Cercetătorii au descoperit că acest compus este constituit numai din atomi de hidrogen (H), ixigen (I) și carbin (C) și are masa moleculară M .

Se știe că regulile de formare a compușilor organici pe Marte sunt următoarele:

- un atom de carbin se poate lega de oricare dintre atomii de C , H și I cu oricâte dintre cele 4 legături pe care le are (astfel, în combinația $H - C = C$ primul atom de carbin se leagă prin două legături de alt atom de carbin și cu o legătură de alt atom de hidrogen)

- un atom de hidrogen se poate lega numai de un atom de carbin cu singura legătură pe care o posedă

- un atom de ixigen se poate lega numai de atomi de carbin cu cele două legături pe care le posedă

- un compus este un ansamblu cu proprietatea că toți atomii de carbin sunt legați conex între ei și nu există vreun atom cu una sau mai multe legături libere (nelegate de un alt atom).

Combinația $H - C = C$ nu este un compus deoarece atomii de carbin mai au legături libere.

Cercetătorii au în vedere studiul categoriilor de compuși, făcând distincție între doi compuși numai dacă aceștia diferă prin numărul de atomi de carbin, de ixigen sau de hidrogen.

Cerință

Scrieți un program care să determine câți compuși distincți formați din atomi de carbin, hidrogen și ixigen (cel puțin unul din fiecare) și care au masa moleculară M există.

Date de intrare

Fișierul de intrare **compus.in** conține pe prima linie masa moleculară a compusului.

Date de ieșire

Fișierul de ieșire **compus.out** conține o singură linie pe care se află numărul de compuși determinat.

Restricții și precizări

$30 \leq M \leq 1000000$

Masa atomului de H este 1, masa atomului de C este 5, iar masa atomului de I este 3. Masa moleculară a unui compus este egală cu suma maselor atomilor din care este constituit compusul respectiv.

Ordinea în care sunt "utilizate" legăturile unui atom nu contează. De asemenea, nici ordinea atomilor sau legăturile interne dintre ei nu contează atâtă timp cât respectă regulile de formare enunțate.

Exemple

Există un singur compus cu masa moleculară 10: cel format cu un atom de C , doi atomi de H și un atom de I ($5 + 2 * 1 + 3 = 10$), compus ale căruia legături pot fi reprezentate astfel:

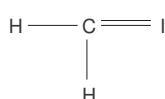


Figura 17.1: Compus1

Se pot obține 3 compuși cu masa moleculară 40: $(5C, 6H, 3I)$, $(6C, 4H, 2I)$, $(7C, 2H, 1I)$:

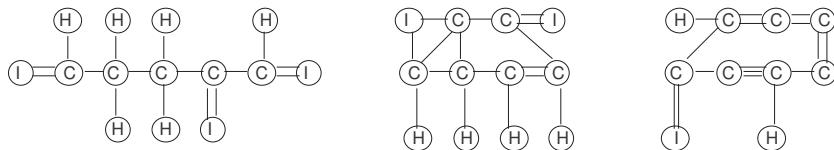


Figura 17.2: Compus2

Reprezentarea cu legături a oricărui dintre compuși nu este unică. Orice altă combinație corespunzătoare aceluiași triplet nu se consideră un compus distinct.

Exemple

compus.in	compus.out	compus.in	compus.out
40	3	125	28

Timp maxim de execuțare/test: 1 secundă

17.1.1 Indicații de rezolvare

Solutia optima si explicatiile pentru problema compus au fost realizate de Irina Dumitrascu (studenta, Automatica, Politehnica Bucuresti)

Idei de baza:

Fiecare legatura consuma 2 capete de legaturi posibile -> grupul de atomi de c va avea totdeauna un nr par de legaturi de oferit.

I consuma 2 capete -> nu pot sa am numar impar de H.

Numarul de legaturi ale atomilor de carbon: $4c$

Numarul minim de legaturi intre atomii de carbon: $2c - 2$

-> Avem:

maxim $2c + 2$ capete de legaturi disponibile pentru o si h
minim = 4 capete (trebuie sa punem obligatoriu un I si 2 H)

Ideeaa 1

c intre cMin = $(m - 3) / 8$ si cMax = $(m - 4) / 5$

Ideeaa 2

$$M = 5*c + 3*i + h = 5 * c + 3 * i' + 3 + h' + 2$$

(am scos separat un I si 2 H care trebuie sa existe oricum in orice compus)
acum $i' \geq 0$, $h' \geq 0$

$$M = 5 * c + 3 * i' + h' + 5$$

Presupunem c cunoscut. Atunci

$$3 * i' + h' = M - 5 * c - 5 = N \text{ (notatie)}$$

dar din condi'iile impuse de legaturile oferite de grupul conex de C,

$$4 \leq 2 * i + h \leq 2 * c + 2$$

$$4 \leq 2 * i' + 2 + h' + 2 \leq 2 * c + 2$$

$$0 \leq 2 * i' + h' \leq 2 * c - 2$$

$$\text{dar } 3 * i' + h' = N, \text{ deci}$$

$$0 \leq N - i' \leq 2 * c - 2$$

```

Deci putem stabili limitele pentru i din inegalitatatile
N - 2 * c + 2 <= i' <= N
Notam oMin = max (0, N - 2 * c + 2)
oMax = min (N, c)

```

Algoritmul devine
c intre cMin si cMax
daca n > 0
i intre oMin si oMax
verific daca h = M - 5c - 3i este par si pozitiv

Ideea 3 (compus.pas)

```

-----
h = N - 3*i trebuie sa fie par;
deci N are aceeasi partitate cu 3 * i -> N are aceeasi paritate cu i;
pot sa stabilesc paritatea la inceput si apoi sa cresc din 2 in 2;
asa nu trebuie sa mai verific paritatea lui h
if ((N mod 2) <> (i mod 2)) then inc(i);
h := N - i * 3;

while ((h >= 0) and (i <= oMax)) do
begin
    inc(sol);
    i := i + 2;
    h := h - 6;
end;

```

Ideea 4 (compus1.pas)

```

-----
In momentul asta pot sa observ ca while-ul e de fapt degeaba, si se reduce la
if ((N mod 2) <> (i mod 2)) then inc(o);
h := N - o * 3;

if ((h >= 0) and (i <= oMax)) then begin
    sol := sol + min ( h div 6 + 1, (oMax - i + 2) div 2);
end

```

Am ajuns in O(N) !!

17.1.2 Cod sursă

Listing 17.1.1: compus.pas

```

1 var m, n,
2     c, h, o, oMax, sMax,
3     sol           :longint;
4
5 function max(a : longint; b : longint) : longint;
6 begin
7     if (a > b) then max := a else max := b;
8 end;
9
10 function min(a : longint; b : longint) : longint;
11 begin
12     if (a < b) then min := a else min := b;
13 end;
14
15
16 begin
17     readln(m);
18     sol := 0;
19     for c := (m-3) div 8 + 1 to (m-4) div 5 do begin
20         n := m - c * 5 - 5;
21         if (n >= 0) then begin

```

```

23          o := max(0, N - 2 * c + 2);
24          oMax := min(c, N);
25          sMax := 2 * (c - 1);
26
27          if ((N mod 2) <> (o mod 2)) then inc(o);
28          h := N - o * 3;
29
30          while ((h >= 0) and (o <= oMax)) do
31          begin
32              inc(sol);
33              o := o + 2;
34              h := h - 6;
35          end
36      end;
37  writeln(sol)
38
39 end.
```

Listing 17.1.2: COMPUS1.pas

```

1 var m, n,
2     c, h, i, oMax,
3     sol           :longint;
4
5 function max(a : longint; b : longint) : longint;
6 begin
7     if (a > b) then max := a else max := b;
8 end;
9
10 function min(a : longint; b : longint) : longint;
11 begin
12     if (a < b) then min := a else min := b;
13 end;
14
15 function calcul (m : longint):longint;
16 begin
17     sol := 0;
18     for c := (m-3) div 8 + 1 to (m-4) div 5 do begin
19         n := m - c * 5 - 5;
20         if (n >= 0) then begin
21             i := max(0, N - 2 * c + 2);
22             oMax := min(c, N);
23
24             if ((N mod 2) <> (i mod 2)) then inc(i);
25             h := N - i * 3;
26
27             if ((h >= 0) and (i <= oMax)) then
28                 sol := sol + min ( h div 6 + 1, (oMax - i) div 2 + 1);
29
30         end;
31     end;
32     calcul := sol;
33 end;
34
35 begin
36     readln(m);
37     writeln(calcul(m));
38 end.
```

17.1.3 *Rezolvare detaliată

17.2 Zmeu

Un zmeu cu n capete călătorește din poveste în poveste, iar în poveștile tradiționale întâlnescă câte un Făt Frumos care-l mai surtează de câteva capete, în timp ce în poveștile moderne salvează omenirea mâncând în timp record, cu toate capetele lui, insecte ucigașe apărute prin mutații genetice. Într-o seară, el își planifică o secesiune de povești cărora să le dea viață. El știe p povești numerotate de la 1 la p , durata fiecareia și numărul de capete pe care le pierde în fiecare

poveste. Mai știe o multime de k perechi de povesti, semnificând faptul că a doua poveste din pereche nu poate fi spusă după prima poveste din pereche.

Cerință

Știind că trebuie să înceapă cu povestea 1 și să încheie succesiunea cu povestea p , ajutați bietul zmeu să aleagă una sau mai multe povesti intermediare astfel încât durata totală să fie minimă și să rămână cu cel puțin un cap la sfârșitul tuturor povestilor.

Date de intrare

Fișierul de intrare **zmeu.in** conține pe prima linie numerele n , p și k despărțite prin câte un spațiu. Pe fiecare din următoarele p linii se află câte o pereche de numere d_i și c_i (separate prin câte un spațiu) ce reprezintă durata și numărul de capete tăiate pentru fiecare poveste. Iar pe ultimele k linii se află câte o pereche de numere p_i și p_j (separate prin câte un spațiu) ce semnifică faptul că povestea p_j nu poate fi spusă după povestea p_i .

Date de ieșire

Fișierul de ieșire **zmeu.out** conține o singură linie pe care se află un număr natural reprezentând durata (minimă) a succesiunii de povesti sau valoarea -1 dacă nu există o astfel de succesiune.

Restricții și precizări

$$2 \leq N \leq 500$$

$$1 \leq P \leq 200$$

$$1 \leq k \leq 30000$$

Valorile reprezentând duratele și numărul de capete sunt numere naturale (duratele fiind strict pozitive), nedepășind valoarea 10.

Exemple

zmeu.in	zmeu.out
10 4 2	9
2 6	
4 0	
1 3	
3 3	
3 2	
4 3	

Timp maxim de executare/test: 1 secundă

17.2.1 Indicații de rezolvare

Problema propusa de prof. Dana Lica (Ploiesti) \index[person]{Dana Lica}

Se lucreaza pe un graf orientat in care nodurile reprezinta povesti si din care s-au eliminat niste arce (date in zmeu.in, i->1, p->i si 1->p)

Rezolvarea are la baza un algoritm de tip Lee cu memorarea in fiecare nod a tuturor timpilor minimi cu care se ajunge in nodul respectiv cu $1, 2, \dots, c$ capete netaiate.

Pentru testelete mari sunt necesare tehnici de alocare si optimizari legate de parcurgerea in latime a grafului.

17.2.2 Cod sursă

Listing 17.2.1: zmeu.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 pmax=200;nmax=500;
4     fi='zmeu.in';fo='zmeu.out';
5 type pnod=^nod;pcnod=^cnod;
6     nod=record time,cap:word;
7         next:pnod
8     end;

```

```

9      cnod=record pov:byte;next:pcnod end;
10     var a:array[1..pmax,1..pmax]of byte;
11     d:array[1..pmax]of pnod;
12     t:array[1..2,1..pmax]of word;
13     n,p,min:integer;
14     q:pnod;
15
16   procedure init;
17   var i,j:integer;
18   begin
19     for i:=1 to p do
20       for j:=1 to p do a[i,j]:=1;
21     for i:=1 to p do begin a[i,i]:=0;a[i,1]:=0;a[p,i]:=0 end;
22     a[1,p]:=0;
23     for i:=1 to p do begin
24       new(d[i]);d[i]^ .cap:=0;
25       d[i]^ .next:=nil
26     end
27   end;
28
29   procedure citeste;
30   var i,x,y,k:integer;
31   f:text;
32   begin
33     assign(f,fi);reset(f);
34     readln(f,n,p,k);
35     for i:=1 to p do readln(f,t[1,i],t[2,i]);
36     init;
37     for i:=1 to k do begin
38       readln(f,x,y);a[x,y]:=0
39     end;
40     close(f)
41   end;
42
43   procedure scrie;
44   var f:text;
45   begin
46     assign(f,fo);rewrite(f);
47     if min=maxint then min:=-1;
48     writeln(f,min) ;
49     close(f)
50   end;
51
52   function inser(j,i:byte):boolean;
53   var p,q,r:pnod;time,cap:word;
54   begin
55     p:=d[j];
56     inser:=false;
57     while p^.next<>nil do begin
58       p:=p^.next;
59       if p^.cap>t[2,i] then begin
60         time:=p^.time+t[1,i];
61         cap:=p^.cap-t[2,i];
62         q:=d[i];
63         if (cap<q^.next^.cap)and(time>q^.next^.time) then continue;
64         while (q^.next<>nil)and(q^.next^.cap<cap) do q:=q^.next;
65         if (q^.next=nil)or(q^.next^.cap>cap) then begin
66           new(r);r^.next:=q^.next;
67           r^.cap:=cap;r^.time:=time;
68           q^.next:=r;inser:=true
69         end
70         else if time<q^.next^.time then begin
71           q^.next^.time:=time;
72           inser:=true
73         end
74       end
75     end
76   end;
77
78   procedure calc;
79   var i,pov:integer;
80   c,u,r:pcnod;
81   pc:array[1..pmax]of boolean;
82   begin
83     if n<=t[2,1] then exit;
84     for i:=1 to p do pc[i]:=false;

```

```

85      new(d[1]^ .next); d[1]^ .next^ .next:=nil;
86      d[1]^ .next^ .time:=t[1,1];
87      d[1]^ .next^ .cap:=n-t[2,1];
88      new(c); c^ .next:=nil;
89      c^ .pov:=1; pc[1]:=true;
90      u:=c;
91      while c<>nil do begin
92          pov:=c^ .pov;
93          for i:=1 to p do
94              if a[pov,i]=1 then begin
95                  if inser(pov,i) and not pc[i] then begin
96                      new(u^ .next); u:=u^ .next;
97                      u^ .pov:=i; u^ .next:=nil; pc[i]:=true
98                  end
99              end;
100         pc[pov]:=false;
101         r:=c; c:=c^ .next; dispose(r)
102     end
103 end;
104
105 begin
106     citeste;
107     calc;
108     min:=maxint;
109     q:=d[p]^ .next;
110     while q<>nil do begin
111         if q^ .time<min then min:=q^ .time;
112         q:=q^ .next
113     end;
114     scrie
115 end.

```

17.2.3 *Rezolvare detaliată

Capitolul 18

OJI 2002

18.1 Urgență

Autoritățile dintr-o zonă de munte intenționează să stabilească un plan de urgență pentru a reacționa mai eficient la frecvențele calamități naturale din zonă. În acest scop au identificat N puncte de interes strategic și le-au numerotat distinct de la 1 la N . Punctele de interes strategic sunt conectate prin M căi de acces având priorități în funcție de importanță. Între oricare două puncte de interes strategic există cel mult o cale de acces ce poate fi parcursă în ambele sensuri și cel puțin un drum (format din una sau mai multe căi de acces) ce le conectează.

În cazul unei calamități unele căi de acces pot fi temporar întrerupte și astfel între anumite puncte de interes nu mai există legătură. Ca urmare pot rezulta mai multe grupuri de puncte în aşa fel încât între oricare două puncte din același grup să existe măcar un drum și între oricare două puncte din grupuri diferite să nu existe drum.

Autoritățile estimează gravitatea unei calamități ca fiind suma priorităților căilor de acces distruse de aceasta și doresc să determine un scenariu de gravitate maximă, în care punctele de interes strategic să fie împărțite într-un număr de K grupuri.

Date de intrare

Fișierul de intrare URGENTA.IN are următorul format:

$N \ M \ K$

$i_1 \ j_1 \ p_1$ - între punctele i_1 și j_1 există o cale de acces de prioritate p_1

$i_2 \ j_2 \ p_2$ - între punctele i_2 și j_2 există o cale de acces de prioritate p_2

...

$i_M \ j_M \ p_M$ - între punctele i_M și j_M există o cale de acces de prioritate p_M

Date de ieșire

Fișierul de ieșire URGENTA.OUT va avea următorul format:

gravmax - gravitatea maximă

C - numărul de căi de acces întrerupte de calamitate

$k_1 \ h_1$ - între punctele k_1 și h_1 a fost întreruptă calea de acces

$k_2 \ h_2$ - între punctele k_2 și h_2 a fost întreruptă calea de acces

...

$k_C \ h_C$ - între punctele k_C și h_C a fost întreruptă calea de acces

Restricții și precizări

$0 < N < 256$

$N - 2 < M < 32385$

$0 < K < N + 1$

Prioritățile căilor de acces sunt întregi strict pozitivi mai mici decât 256.

Un grup de puncte poate conține între 1 și N puncte inclusiv.

Dacă există mai multe soluții, programul va determina una singură.

Exemplu

URGENTA.IN	URGENTA.OUT
7 11 4	27
1 2 1	8
1 3 2	1 3
1 7 3	1 7
2 4 3	2 4
3 4 2	3 4
3 5 1	3 7
3 6 1	4 5
3 7 5	5 6
4 5 5	6 7
5 6 4	
6 7 3	

Timp maxim de executare: 1 secundă / test

18.1.1 *Indicații de rezolvare

18.1.2 *Codul sursă

18.1.3 *Rezolvare detaliată

18.2 Nunta

În fața palatului Printesei Mofturoase se află N peștori așezăți la coadă, unul în spatele celuilalt. Fiecare poartă sub mantie un număr de pietre prețioase pe care dorește să le ofere prințesei ca dar de nuntă. Pentru a nu semăna vrajbă în rândurile lor, prințesa a decis să-i determine ca $N - 1$ dintre ei să renunțe în chip pașnic, peștorul rămas devenind alesul prințesei (indiferent de numărul de pietre prețioase deținute de acesta).

Doi peștori vecini la coadă se pot înțelege între ei astfel: cel care are mai puține pietre prețioase pleacă de la coadă primind de la celălalt un număr de pietre astfel încât să plece acasă cu un număr dublu de pietre față de câte avea. Dacă doi peștori au același număr de pietre, unul din ei (nu contează care) pleacă luând toate pietrele vecinului său.

Un peștor se poate înțelege la un moment dat cu unul singur dintre cei doi vecini ai săi. După plecarea unui peștor, toți cei din spatele lui avansează.

De exemplu: pentru configurația alăturată de 5 peștori, un sir posibil de negocieri care conduc la reducerea cozii la un singur peștor este: se înțeleg vecinii 4 cu 5 și pleacă 4, se înțeleg apoi 1 cu 2 și pleacă 1, se înțeleg apoi 3 cu 2 și pleacă 3, se înțeleg 2 cu 5 și pleacă 5. Astfel peștorul 2 câștigă mâna preafrumoasei prințese, oferindu-i 0 pietre prețioase ca dar de nuntă.

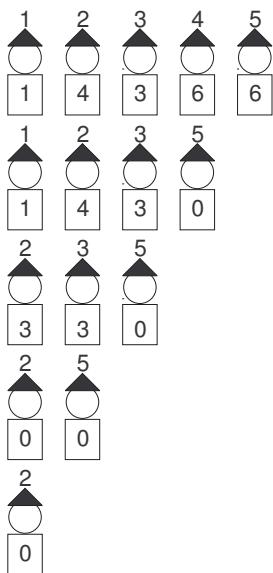


Figura 18.1: Nunta

Fie P numărul de pietre prețioase pe care le are peștorul care va deveni alesul prințesei. Se cer valorile distincte ale lui P la care se poate ajunge prin toate succesiunile de negocieri posibile.

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

- pe prima linie numărul de peștori: n ($1 \leq n \leq 50$).
- pe a doua linie, n numere naturale din intervalul $[0, 20]$, reprezentând numărul de pietre prețioase pe care le dețin peștorii, în ordinea în care stau la coadă.

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

- pe prima linie numărul m de valori distincte ce pot fi obținute
- pe a doua linie cele m valori ordonate crescător, reprezentând valorile care se pot obține.

Exemplu:

nunta.in	nunta.out
4	3
1 4 2 6	1 3 5

Timp maxim de executare: 1 secundă / test

18.2.1 *Indicații de rezolvare

18.2.2 *Codul sursă

18.2.3 *Rezolvare detaliată

Partea II

ONI - Olimpiada națională de informatică

Capitolul 19

ONI 2019

19.1 lexicografic

Problema 1 - lexicografic

100 de puncte

Se dă un sir v format din N elemente naturale nenule nu neapărat distințe.

Asupra sirului putem aplica un singur tip de operație: interschimbarea a două elemente aflate pe poziții consecutive.

Cerințe

Dându-se un număr natural K , se cere sirul minim lexicografic ce se poate obține prin aplicarea a cel mult K interschimbări de elemente de pe poziții consecutive.

Date de intrare

În fișierul **lexicografic.in** se află pe prima linie T , reprezentând numărul de teste.

Urmează cele T teste, fiecare pe câte 2 linii. Pe prima linie din cadrul unui test se află două numere N și K separate prin spațiu. Pe linia a doua din cadrul unui test se află cele N elemente ale sirului v separate prin spații.

Date de ieșire

În fișierul **lexicografic.out** se vor afișa cele T linii, câte una corespunzătoare răspunsului pe fiecare test. Linia corespunzătoare unui test va conține cele N elemente separate prin spații ale sirului minim lexicografic ce s-a obținut din sirul inițial, după aplicarea a cel mult K interschimbări de elemente de pe poziții consecutive.

Restricții și precizări

- $1 \leq N \leq 250.000$;
- $T \leq 2500$;
- Într-un fișier de intrare suma totală a lungimilor sirurilor corespunzătoare celor T teste nu va depăși 250.000;
 - $1 \leq K \leq N * (N - 1)/2$;
 - $1 \leq v[i] \leq N$, pentru $1 \leq i \leq N$;
 - Vă rugăm să acordați atenție tipului de date necesar pentru a citi valoarea lui K ;
 - Pentru acordarea punctajului pe un fișier de test este necesară rezolvarea corectă a tuturor celor T teste;
 - Pentru teste în valoare de 5 puncte se garantează $K = N * (N - 1)/2$;
 - Pentru alte teste în valoare de 7 puncte se garantează $K = 1$;
 - Pentru alte teste în valoare de 23 de puncte se garantează $T \leq 10, N \leq 50$;
 - Pentru alte teste în valoare de 4 puncte se garantează $T \leq 10, N \leq 100$;
 - Pentru alte teste în valoare de 12 puncte se garantează $T \leq 10, N \leq 500$;
 - Pentru alte teste în valoare de 24 de puncte se garantează $T \leq 10, N \leq 2000$;
 - Un sir a_1, a_2, \dots, a_n este mai mic lexicografic decât un alt sir b_1, b_2, \dots, b_n dacă există un număr întreg P mai mic sau egal cu N astfel încât: $a_1 = b_1, a_2 = b_2, \dots, a_{P-1} = b_{P-1}$, iar $a_P < b_P$.

Exemple

lexicografic.in	lexicografic.out	Explicații
3	2 3 4 1 1	Pentru primul test:
5 2	1 2 3 4	șirul este format din $N = 5$ elemente, și anume $v = (4, 2, 3, 1, 1)$. Putem efectua $K = 2$ interschimbări.
4 2 3 1 1	3 3 5 4 5 6	Interschimbând elementele $v[1]$ și $v[2]$ obținem șirul $(2, 4, 3, 1, 1)$, apoi după interschimbarea elementelor $v[3]$ și $v[2]$ se obține șirul minim lexicografic $(2, 3, 4, 1, 1)$.
4 3		
2 1 3 4		
6 4		
5 3 5 3 4 6		

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

Memorie: total **128 MB**

Dimensiune maximă a sursei: **20 KB**

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

19.1.1 Indicații de rezolvare

Constantinescu Andrei Costin

$$1. K = \frac{N(N-1)}{2}$$

În acest caz, operațiile se pot folosi pentru a sorta perfect numerele. Sortarea poate fi executată prin orice metodă, spre exemplu utilizând funcția *sort* din STL.

$$2. K = 1$$

Există o singură operație. Se observă că pentru a face această operație utilă, ea trebuie aplicată pe două poziții i și $i + 1$ astfel încât $v_i > v_{i+1}$. Pentru a obține șirul minim lexicografic trebuie aleasă poziția i minimă cu această proprietate.

3. $N = 2000$ Se va parcurge șirul în ordine crescătoare a indicilor. Pentru fiecare poziție i de la 1 la N se va alege cel mai mic element dintre $v_i, v_{i+1}, \dots, v_{\min(n; i+K)}$ și se va aduce pe poziția i prin operații succesive de interschimbare.

În caz de egalitate se va alege primul dintre aceste elemente. La sfârșitul pasului curent se va scădea din K numărul de operații efectuate.

Se poate demonstra că acest algoritm duce la soluția optimă.

Argumentul central este urmatorul: dacă un element v_i oarecare trebuie adus mai la stânga decât alt element v_j în soluția finală, atunci v_i merită mutat până la poziția lui finală înaintea lui v_j .

Complexitatea acestei soluții este $O(N^2)$.

4. $N = 250000$ Soluția anterioară se poate optimiza la $O(N \log N)$ folosind *structuri de date* eficiente pentru a îmbunătăți căutarea elementului care trebuie adus pe poziția i . Spre exemplu, se poate menține un *arbore de intervale* care stochează minimul pe interval și numărul de elemente încă nemutate din interval. Cu ajutorul parametrului din urmă și al lui K , se caută minimul în intervalul $[1; \min(n; K+1)]$ de **numere nemutate**. Se șterge elementul din structură, se adaugă la finalul unui vector în care se ține soluția, se scade K cu numărul de operații necesare și se trece la găsirea următorului element.

19.1.2 Cod sursă

Listing 19.1.1: lexicografic_100p_1.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int inf = 1 << 30;
6 const int NMAX = 250005;
7
8 int t, n, a[NMAX];
9 int aint_mn[4 * NMAX], aint_sum[4 * NMAX];
10 long long k;
11

```

```

12 void Update_sum(int node, int st, int dr, int poz, int val)
13 {
14     if (st == dr)
15         aint_sum[node] = val;
16     else
17     {
18         int mij = (st + dr) / 2;
19         if (poz <= mij) Update_sum(2 * node, st, mij, poz, val);
20         if (poz > mij) Update_sum(2 * node + 1, mij + 1, dr, poz, val);
21         aint_sum[node] = aint_sum[2 * node] + aint_sum[2 * node + 1];
22     }
23 }
24
25 void Update_mn(int node, int st, int dr, int poz, int val)
26 {
27     if (st == dr)
28         aint_mn[node] = val;
29     else
30     {
31         int mij = (st + dr) / 2;
32         if (poz <= mij) Update_mn(2 * node, st, mij, poz, val);
33         if (poz > mij) Update_mn(2 * node + 1, mij + 1, dr, poz, val);
34         aint_mn[node] = a[aint_mn[2 * node]] <= a[aint_mn[2 * node + 1]] ?
35             aint_mn[2 * node] :
36             aint_mn[2 * node + 1];
37     }
38 }
39
40 int Query_sum(int node, int st, int dr, int l, int r)
41 {
42     if (l <= st && dr <= r)
43         return aint_sum[node];
44     else
45     {
46         int mij = (st + dr) / 2, val = 0;
47         if (l <= mij) val += Query_sum(2 * node, st, mij, l, r);
48         if (r > mij) val += Query_sum(2 * node + 1, mij + 1, dr, l, r);
49
50         return val;
51     }
52 }
53
54 int Query_poz(int node, int st, int dr, long long many)
55 {
56     if (st == dr)
57         return st;
58     else
59     {
60         int mij = (st + dr) / 2;
61         if (aint_sum[2 * node] >= many)
62             return Query_poz(2 * node, st, mij, many);
63         else
64             return Query_poz(2 * node + 1, mij + 1, dr, many - aint_sum[2 * node]);
65     }
66 }
67
68 int Query_mn(int node, int st, int dr, int l, int r)
69 {
70     if (l <= st && dr <= r)
71         return aint_mn[node];
72     else
73     {
74         int mij = (st + dr) / 2, val = 0;
75         if (l <= mij)
76         {
77             int ans = Query_mn(2 * node, st, mij, l, r);
78             val = a[val] <= a[ans] ? val : ans;
79         }
80         if (r > mij)
81         {
82             int ans = Query_mn(2 * node + 1, mij + 1, dr, l, r);
83             val = a[val] <= a[ans] ? val : ans;
84         }
85
86         return val;
87     }
}

```

```

88 }
89
90 int main()
91 {
92     freopen("lexicografic.in", "r", stdin);
93     freopen("lexicografic.out", "w", stdout);
94
95     cin.sync_with_stdio(false);
96
97     cin >> t;
98     while (t--)
99     {
100         cin >> n >> k;
101         a[0] = inf;
102         for (int i = 1; i <= n; i++)
103         {
104             cin >> a[i];
105             Update_sum(1, 1, n, i, 1);
106             Update_mn(1, 1, n, i, i);
107         }
108
109         for (int i = 1; i <= n; i++)
110         {
111             // find k + 1 unvisited
112             int poz = Query_poz(1, 1, n, k + 1);
113
114             // take min from [1, poz]
115             int mn_poz = Query_mn(1, 1, n, 1, poz);
116
117             // mark it as visited
118             cout << a[mn_poz] << " ";
119             k -= Query_sum(1, 1, n, 1, mn_poz) - 1;
120             Update_sum(1, 1, n, mn_poz, 0);
121             Update_mn(1, 1, n, mn_poz, 0);
122         }
123
124         cout << "\n";
125
126         // clear
127         for (int i = 1; i <= n * 4; i++)
128             aint_mn[i] = aint_sum[i] = 0;
129     }
130     return 0;
131 }
```

Listing 19.1.2: lexicografic_100p_2.cpp

```

1 #include <algorithm>
2 #include <fstream>
3 #include <iostream>
4 #include <vector>
5 #include <list>
6 #include <set>
7
8 using namespace std;
9
10#define SZ(x) ((int) (x).size())
11
12const int INF = 0x3f3f3f3f;
13
14class SegmentTree
15{
16public:
17    SegmentTree(const vector<int>& m):
18        sum(SZ(m) * 4 + 5), vmin(SZ(m) * 4 + 5), n(SZ(m))
19    {
20        build(0, 0, n - 1, m);
21    }
22    pair<int, int> getMin(int lim)
23    {
24        return getMin(0, 0, n - 1, lim);
25    }
26    int remove(int pos)
27    {
28        return remove(0, 0, n - 1, pos);
29    }
30}
```

```

29     }
30
31 private:
32     vector<int> sum;
33     vector<pair<int, int>> vmin;
34     int n;
35
36     void build(int node, int left, int right, const vector<int>& m)
37     {
38         sum[node] = right - left + 1;
39         if (left == right)
40         {
41             vmin[node] = make_pair(m[left], left);
42         }
43         else
44         {
45             int mid = (left + right) / 2;
46             build(2 * node + 1, left, mid, m);
47             build(2 * node + 2, mid + 1, right, m);
48             vmin[node] = min(vmin[2 * node + 1], vmin[2 * node + 2]);
49         }
50     }
51
52     pair<int, int> getMin(int node, int left, int right, int lim)
53     {
54         if (sum[node] <= lim)
55         {
56             return vmin[node];
57         }
58         else
59         {
60             int mid = (left + right) / 2;
61             auto ret = getMin(2 * node + 1, left, mid, lim);
62             if (lim > sum[2 * node + 1])
63             {
64                 ret = min(ret, getMin(2 * node + 2,
65                                     mid + 1, right, lim - sum[2 * node + 1]));
66             }
67             return ret;
68         }
69     }
70
71     int remove(int node, int left, int right, int pos)
72     {
73         if (left == right)
74         {
75             vmin[node] = make_pair(INF, INF);
76             sum[node] = 0;
77             return 0;
78         }
79         else
80         {
81             int mid = (left + right) / 2;
82             int ret;
83             if (pos <= mid)
84             {
85                 ret = remove(2 * node + 1, left, mid, pos);
86             }
87             else
88             {
89                 ret = remove(2 * node + 2, mid + 1, right, pos);
90                 ret += sum[2 * node + 1];
91             }
92
93             sum[node] = (sum[2 * node + 1] + sum[2 * node + 2]);
94             vmin[node] = min(vmin[2 * node + 1], vmin[2 * node + 2]);
95             return ret;
96         }
97     }
98 };
99
100    int main()
101    {
102        ifstream fin("lexicografic.in");
103        ofstream fout("lexicografic.out");
104

```

```

105     int T;
106     fin >> T;
107
108     while (T-- > 0)
109     {
110         int n;
111         int64_t k;
112         fin >> n >> k;
113
114         vector<int> numbers(n);
115         for (int i = 0; i < n; ++i)
116         {
117             fin >> numbers[i];
118         }
119
120         SegmentTree t(numbers);
121         for (int i = 0; i < n; ++i)
122         {
123             auto vmin = t.getMin((int) min(k + 1, (int64_t) n));
124             fout << vmin.first << ' ';
125             k -= t.remove(vmin.second);
126         }
127
128         fin.close();
129         fout.close();
130     }
131 }
```

Listing 19.1.3: lexicografic_O(n^2).cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 using lint = long long int;
6 const int NMAX = 250000;
7 // const int KMAX = ?";
8
9 void run_test()
10 {
11     // Read and asserts
12     int N;
13     cin >> N;
14
15     int K;
16     cin >> K;
17
18     assert(1 <= N && N <= NMAX);
19
20     // TODO: Add assert for K
21     vector <int> v(N);
22     for (int i = 0; i < N; ++i)
23         cin >> v[i];
24
25     // Run greedy
26     for (int i = 0; i < N; ++i)
27     {
28         const int limit = min(static_cast <lint>(N - 1), i + K);
29         int minimum = v[i], pos = i;
30         for (int j = i + 1; j <= limit; ++j)
31         {
32             if (v[j] < minimum)
33             {
34                 minimum = v[j];
35                 pos = j;
36             }
37         }
38
39         K -= (pos - i);
40         while (pos > i)
41         {
42             v[pos] = v[pos - 1];
43             --pos;
44         }
45         v[i] = minimum;
```

```

46     /*for (int j = 0; j < N; ++j) {
47         cerr << v[j] << ' ';
48     }
49     cerr << endl;*/
50 }
51
52 // Print answer
53 for (int i = 0; i < static_cast <int>(v.size()); ++i)
54     cout << v[i] << " \n"[i + 1 == v.size()];
55 }
56
57 int main()
58 {
59     freopen("lexicografic.in", "r", stdin);
60     freopen("lexicografic.out", "w", stdout);
61     int T = 0;
62     cin >> T;
63     while (T--)
64         run_test();
65
66     return 0;
67 }
```

19.1.3 *Rezolvare detaliată

19.2 oracol

Problema 2 - oracol

100 de puncte

Gustavo, după ce a realizat că posedă abilitatea de a vedea în viitor, a decis că a venit momentul să treacă la următorul nivel și să-și valorifice capacitatele extrasenzoriale. Pentru a câștiga prestigiu și a deveni mai cunoscut în rândurile magicienilor profesioniști, acesta a ales să debuteze la Olimpiada Națională de Informatică prin prezicerea datelor de intrare pentru anumite probleme propuse în concurs.

Primul client al lui Gustavo, Alfredo, ar dori să afle într-un mod inedit conținutul unui fișier de intrare aferent unei probleme de concurs, în care sunt scrise elementele unui sir p de N numere întregi. Pentru a face lucrurile mai interesante, Gustavo îi percepă o taxă de $C(i, j)$ bănuți pentru a-i divulga suma numerelor din sirul p cu indicii în intervalul $[i, j]$, anume $p_i + p_{i+1} + \dots + p_j$.

Cerințe

Dându-se valoarea lui N și toate valorile $C(i, j)$ cu $1 \leq i \leq j \leq N$, determinați costul total minim pe care trebuie să-l plătească Alfredo pentru a afla toate elementele sirului p .

Date de intrare

În fișierul **oracol.in** se află pe prima linie numărul natural N . Pe următoarele N linii se află taxele percepute de Gustavo astfel: pe linia $i + 1$ se vor afla $N - i + 1$ numere naturale separate prin câte un spatiu, reprezentând în ordine costurile $C(i, i)$, $C(i, i + 1)$, ..., $C(i, N)$.

Date de ieșire

În fișierul **oracol.out** trebuie să se găsească un singur număr care reprezintă costul total minim pe care trebuie să-l plătească Alfredo pentru a afla sirul p .

Restricții și precizări

- $1 \leq N \leq 1000$;
- pentru orice $1 \leq i \leq j \leq N$ se garantează $0 \leq C(i, j) \leq 1.000.000$;
- pentru teste în valoare de 48 puncte $1 \leq N \leq 250$.

Exemple

oracol.in	oracol.out	Explicații
3	6	Costul total minim este 6 și se obține astfel:
4 5 1		Cu un cost de valoare $C(1, 3) = 1$ putem afla suma $p_1 + p_2 + p_3$.
6 3		Cu un cost de valoare $C(3, 3) = 2$ putem afla valoarea lui p_3 .
2		Cu un cost de valoare $C(2, 3) = 3$ putem afla suma $4p_2 + p_3$.
		Din acestea putem afla exact toate elementele sirului p.

Timp maxim de executare/test: **0.3** secunde

Memorie: total **128 MB**

Dimensiune maximă a sursei: **20 KB**

19.2.1 Indicații de rezolvare

Oncescu Costin Andrei

- Problema determinării sirului p este echivalentă cu problema determinării sumelor sale parțiale:

$$s_j = \sum_{i=1}^j p_i$$

- Achitarea taxei pentru suma subsecvenței $[i, j]$ ne permite să determinăm s_j dacă îl cunoaștem pe s_{i-1} sau, analog, să-l determinăm pe s_{i-1} dacă îl cunoaștem pe s_j .
- Singura sumă parțială cunoscută inițial este $s_0 = 0$.
- Ne dorim ca fiecare sumă parțială să poată fi dedusă din s_0 printr-un sir de subsecvențe care "leagă" cele două sume.
- Pentru a formaliza această idee, putem construi un *graf neorientat complet* G în care nodurile sunt reprezentate de cele $N + 1$ sume parțiale, iar valoarea $C(i, j)$ determină costul muchiei (i, j) .
- În acest graf dorim să selectăm o submulțime de muchii de cost total minim care conectează nodul 0 cu toate celelalte noduri. Fiindcă toate costurile sunt nenegative, această problemă este echivalentă cu problema găsirii unui *arbore parțial de cost minim*.
- Atât *algoritmul lui Kruskal* implementat în timp $O(N^2 \log N)$ cât și *algoritmul lui Prim* în timp $O(N^2)$ pot obține punctaj maxim.

19.2.2 Cod sursă

Listing 19.2.1: oracol_kruskal_100p.cpp

```

1 #include <algorithm>
2 #include <fstream>
3 #include <vector>
4
5 using namespace std;
6
7 struct Edge
8 {
9     int from, to, cost;
10    bool operator<(const Edge& e) const
11    {
12        return cost < e.cost;
13    }
14 };
15
16 class DSU
17 {
18 public:
19     DSU(int n):
20         f(n)
21     {
22         for (int i = 0; i < n; ++i)
23         {
24             f[i] = i;
25         }
26     }
27 }
```

```

28     int& operator[](int x)
29     {
30         int y, p;
31         for (y = x; y != f[y]; y = f[y]);
32         for (; x != y; x = p)
33         {
34             p = f[x];
35             f[x] = y;
36         }
37         return f[y];
38     }
39
40 private:
41     vector<int> f;
42 };
43
44 int main()
45 {
46     ifstream fin("oracol.in");
47     ofstream fout("oracol.out");
48
49     int n;
50     fin >> n;
51
52     vector<Edge> edges;
53     for (int i = 0; i < n; ++i)
54     {
55         for (int j = i + 1; j <= n; ++j)
56         {
57             int cost;
58             fin >> cost;
59             edges.push_back(Edge{i, j, cost});
60         }
61     }
62
63     sort(edges.begin(), edges.end());
64
65     DSU f(n + 1);
66
67     int64_t ans = 0;
68     for (const Edge& e: edges)
69     {
70         if (f[e.from] != f[e.to])
71         {
72             ans += e.cost;
73             f[e.from] = f[e.to];
74         }
75     }
76
77     fout << ans << '\n';
78 }
```

Listing 19.2.2: oracol_prim_100p.cpp

```

1 #include <iostream>
2 #include <vector>
3 #include <limits>
4 #include <iterator>
5 #include <numeric>
6
7 using namespace std;
8
9 int64_t SpanningTree(const vector<vector<int>>& cost)
10 {
11     int n = static_cast<int>(cost.size());
12     vector<int> prim_cost(n);
13     vector<int> available(n);
14     iota(available.begin(), available.end(), 0);
15     fill(next(prim_cost.begin()), prim_cost.end(),
16          numeric_limits<int>::max());
17     int64_t total = 0;
18     for (int i = 0; i < n; ++i)
19     {
20         int best_node = 0;
21         for (int j = 1; j < static_cast<int>(available.size()); ++j)
```

```

22         {
23             if (prim_cost[available[j]] < prim_cost[available[best_node]])
24             {
25                 best_node = j;
26             }
27         }
28
29         int extracted_node = available[best_node];
30         total += prim_cost[extracted_node];
31
32         swap(available[best_node], available.back());
33         available.pop_back();
34
35         for (auto&& node: available)
36         {
37             if (cost[extracted_node][node] < prim_cost[node])
38             {
39                 prim_cost[node] = cost[extracted_node][node];
40             }
41         }
42     }
43
44     return total;
45 }
46
47 int main()
48 {
49     ifstream fin("oracol.in");
50     int n; fin >> n;
51     vector<vector<int>> cost(n + 1, vector<int>(n + 1));
52     for (int i = 1; i <= n; ++i)
53     {
54         for (int j = i; j <= n; ++j)
55         {
56             fin >> cost[i - 1][j];
57             cost[j][i - 1] = cost[i - 1][j];
58         }
59     }
60
61     ofstream("oracol.out") << SpanningTree(cost) << endl;
62 }
```

19.2.3 *Rezolvare detaliată

19.3 treegcd

Problema 3 - treegcd

100 de puncte

Când era în vizită la bunici, Cătălin a descoperit în garaj o consolă Nintendo din anul 1970. Din fericire, această consolă avea și un joc. Acest joc se chemea "TreeGCD". Cătălin primește un arbore cu N noduri și un număr M . El trebuie să spună în câte moduri poate să atribuie fiecărui nod un număr de la 1 la M , astfel încât oricare două noduri adiacente să nu aibă atribuite numere prime între ele (adică cel mai mare divizor comun este strict mai mare decât 1).

Cerințe

Determinați care este numărul de moduri în care putem atribui fiecărui nod câte un număr de la 1 la M , astfel încât oricare două noduri care sunt legate printr-o muchie să nu aibă atribuite numere prime între ele. Numărul trebuie afișat modulo 1.000.000.007.

Date de intrare

În fișierul **treegcd.in** se află pe prima linie două numere naturale nenule N și M , separate printr-un spațiu. Pe fiecare din următoarele $N - 1$ linii se află câte două numere naturale nenule x și y , separate printr-un spațiu, cu semnificația că nodurile x și y sunt adiacente.

Date de ieșire

În fișierul **treegcd.out** trebuie să se găsească un singur număr. Acest număr reprezintă în câte moduri se poate atribui fiecărui nod o valoare de la 1 la M , astfel încât pentru oricare două noduri adiacente, valorile asociate să nu fie prime între ele. Deoarece rezultatul poate fi foarte mare, se va afișa restul $modulo (10^9 + 7)$ pentru numărul cerut.

Restricții și precizări

- $2 \leq N \leq 100$;
- $2 \leq M \leq 10.000$;
- pentru teste în valoare de 4 puncte avem $N = 2$ și $M \leq 1.000$;
- pentru alte teste în valoare de 13 puncte avem $N \leq 6$ și $M \leq 10$;
- pentru teste în valoare de 40 de puncte avem $N \leq 100$ și $M \leq 100$;
- pentru alte teste în valoare de 43 de puncte avem $N \leq 100$ și $M \leq 10.000$

Exemple

treegcd.in	treegcd.out	Explicații
2 6 1 2	13	Perechile de valori asociate nodurilor 1, 2 sunt: (2,2), (2,4), (2,6), (3,3), (3,6), (4,2), (4,4), (4,6), (5,5), (6,2), (6,3), (6,4), (6,6).
5 6 5 3 3 1 5 4 3 2	397	Rezultatul este 397.
10 6 7 1 2 1 3 2 4 2 5 2 6 2 7 5 8 5 9 7 1 0	534323877	Rezultatul este 6315455578532062, iar $6315455578532062 \% 1000000007 = 534323877$

Timp maxim de executare/test: 0.4 secunde

Memorie: total **128 MB**

Dimensiune maximă a sursei: **20 KB**

19.3.1 Indicații de rezolvare

student Banu Denis - Universitatea "Al. I. Cuza" Iași

4 puncte

Putem să atribuim fiecare valoare de la 2 la M celor două noduri și verificăm câte perechi de valori au *CMMDC*-ul mai mare ca 1.

Complexitate: $O(m^2)$

17 puncte

Folosind metoda *backtracking* încercam toate posibilitățile de a atribui numere de la 2 la M celor N noduri și verificăm că cel mai mare divizor comun între oricare două noduri adiacente să fie mai mare decât 1.

Complexitate: $O(m^n * n)$

53 puncte

Vom ține $DP[x][val]$ numărul de moduri de a atribui valori de la 2 la M în subarborele nodului x și nodul x să aibă valoarea val , astfel încât să nu avem noduri adiacente care au *CMMDC*-ul 1.

Dacă ne aflăm într-un nod x și am construit deja dinamica pentru toți fiii, luăm pe rând câte un fiu și pentru fiecare fiu y : fixăm valoarea i pe care vrem să o punem în nodul curent, fixăm valoarea j pe care vrem să o aibă fiul y și dacă $CMMDC(i, j) > 1$ atunci putem aduna la $aux[i]$ valoarea $DP[y][j]$, care înseamnă că pentru nodul x ca el să aibă valoarea i putem seta fiul y să aibă valoarea j , iar acest lucru se poate face în $DP[y][j]$ moduri. După ce am terminat cu fiul y , în $aux[i]$ se află toate modurile în care putem pune valoarea i în nodul x și valori de la 2 la M în subarborele y astfel încât arborele să fie completat corect.

Toate valorile $aux[i]$ pentru fiecare fiu y trebuie înmulțite iar acesta este rezultatul pentru $DP[nod][j]$.

Rezultatul problemei este suma valorilor din $DP[1]$.

Complexitate: $O(n * m * m)$

100 puncte

Putem face o dinamică asemănătoare cu cea de mai sus, dar trebuie să optimizam tranziția valorilor între fiu și nodul curent.

Dacă suntem într-un nod x și vrem să adăugăm valorile din fiul y , vom încerca pe rând să fixăm *CMMDC*-ul.

Prima dată consideram *CMMDC*-ul 2, adunăm toate valorile din $DP[y][2 * k]$ (toți multiplii de 2 din nodul y), deoarece pentru a avea *CMMDC*-ul 2 sigur y trebuie să aibă o valoare multiplu de 2.

Din același raționament vom aduna suma rezultată la $DP[x][2 * k]$, adică toți multiplii de 2 din nodul curent.

Pentru 3 putem proceda exact la fel.

Pentru 4 observăm că orice multiplu de 4 este și multimplu de 2 așa că nu vom face nimic.

Pentru 5 procedăm la fel ca la 2 și 3.

Pentru 6 observăm că 6 este $2 * 3$ și toți multiplii de 6 au fost deja numărăți de *două ori* așa că trebuie să facem suma tuturor valorilor $DP[y][6 * k]$, iar această sumă trebuie să o scădem din $DP[x][6 * k]$ §. a. m. d.

Ceea ce aplicăm mai sus este *principiul includerii și excluderii*, iar formula care ne spune când trebuie să adunăm și când să scădem este următoarea:

- Daca i are un factor prim care apare la o putere mai mare de 1 nu facem nimic cu el.
- Daca i are un număr *impar* de factori primi atunci îl adunăm.
- Daca i are un număr *par* de factori primi atunci îl scădem.

Deoarece parcurgem toate numerele de la 2 la M și pentru fiecare număr parcurgem doar multiplii lui, complexitatea în care se face transferul de la un fiu la nodul tată este $O(M \log M)$, iar complexitatea finală este $O(NM \log M)$.

19.3.2 *Cod sursă

19.3.3 *Rezolvare detaliată

19.4 compact

Problema 4 - compact

100 de puncte

Se dă un sir a de N numere naturale nenule mai mici sau egale cu M . Se dorește partitōnarea sirului în cât mai multe grupe stabile. O *grupă stabilă* se definește ca fiind o secvență continuă și nevidă de numere $a_s, a_{(s+1)}, a_{(s+2)}, \dots, a_{(d-1)}, a_d$, respectând următoarea condiție:

- orice alt element din afara intervalului $[s, d]$ este ori strict mai mare, ori strict mai mic decât **toate** valorile din $[s, d]$.

Mai exact, pentru orice $i \notin [s, d]$, doar una din următoarele condiții este satisfăcută:

1. $a[i] < a[j]$, oricare ar fi $s \leq j \leq d$;
2. $a[i] > a[j]$, oricare ar fi $s \leq j \leq d$.

Partitōnarea sirului presupune ca fiecare număr să facă parte din **exact o grupă**.

Cerințe

Dându-se N, M și sirul a de N numere, să se găsească o partitōie a sirului a în cât mai multe grupe stabile.

Date de intrare

În fișierul **compact.in** pe prima linie se află două numere N și M separate prin spațiu. Pe a doua linie se află cele N elemente ale șirului a separate prin câte un spațiu.

Date de ieșire

În fișierul **compact.out** se vor afișa două linii. Pe prima linie se va afla numărul maxim de grupe stable G , iar pe a doua linie se vor afla G valori, reprezentând poziția ultimului element al fiecarei grupe stable în **ordine crescătoare**.

Restricții și precizări

- $1 \leq N \leq 1.000.000$.
- $1 \leq M \leq N$.
- $1 \leq a[i] \leq M$, pentru $1 \leq i \leq N$.
- Pentru teste în valoare de 21 puncte $N \leq 100$.
- Pentru alte teste în valoare de 28 de puncte $N \leq 3000$.
- Se garantează că fiecare număr natural de la 1 la M apare cel puțin o dată.
- Ultimul indice din fiecare soluție va fi întotdeauna N .
- În cazul în care există mai multe soluții cu număr maxim de grupe stable, se va afișa soluția minimă lexicografic.
- Un șir a_1, a_2, \dots, a_n este mai mic lexicografic decât un alt șir b_1, b_2, \dots, b_n dacă există un număr întreg P mai mic sau egal cu N astfel încât: $a_1 = b_1, a_2 = b_2, \dots, a_{P-1} = b_{P-1}$, iar $a_P < b_P$.

Exemple

compact.in	compact.out
6 5	5
1 4 2 3 5 5	1 2 3 4 6
7 5	1
1 3 2 1 5 2 4	7
14 10	5
5 8 6 7 5 2 1 2 3 3 4 10 9 10	5 8 10 11 14
4 3	2
3 1 2 1	1 4

Timp maxim de executare/test: **0.6** secunde

Memorie: total **128 MB**

Dimensiune maximă a sursei: **20 KB**

19.4.1 Indicații de rezolvare

Alex Tatomir

1 $N \leq 100$ (21 de puncte)

O soluție este calcularea valorilor $d[i] =$ numărul maxim de grupe în care se poate împărți prefixul de lungime i sau -1 încaz că este imposibil. Dinamica se poate calcula prin formula $d[i] = \max(d[j] + 1)$ cu $0 \leq j < i$ astfel încât subsecvența $[j + 1; i]$ poate forma o grupă corectă. Dacă pentru fiecare j se verifică în $O(N)$ această condiție, complexitatea soluției este $O(N^3)$.

2 $N \leq 1000$ (49 de puncte)

O optimizare a soluției anterioare este verificarea unei secvențe $[j + 1; i]$ în timp $O(1)$. Pentru acest lucru se poate varia j în ordine descrescătoare, actualizând valoarea maximă și minimă. Pentru ca secvența să fie corectă, trebuie să se respecte condiția ca valoarea $i - j$ să fie egală cu suma frecvențelor numerelor dintre minim și maxim. Această valoare se poate calcula prin menținerea unor *sume parțiale* pe vectorul de frecvențe.

Complexitatea devine $O(N^2)$.

3 $N \leq 1000000$ (100 de puncte)

O soluție în timp $O(N)$ este următoarea.

Se pleacă cu o poziție oarecare i și se extinde grupa inițial formată doar din acest element în stânga și în dreapta. Se observă că dacă într-o grupă în construcție se află un număr x , atunci grupa trebuie să conțină toate numerele situate în sir între prima și ultima apariție a lui x , ceea ce impune o "necesitate" pe un interval de poziții. De asemenea, dacă într-o grupă în construcție există elementele $x \leq y$, atunci grupa trebuie să conțină toate elementele între x și y , ceea ce impune o "necesitate" pe intervalul de valori.

Se pot menține 4 valori

av_poz_stanga;
av_poz_dreapta;
av_val_stanga;
av_val_dreapta

care reprezintă faptul că s-au procesat pozițiile din intervalul

[av_poz_stanga; av_poz_dreapta]

și valorile din intervalul

[av_val_stanga; av_val_dreapta],

și alte 4 valori

nec_poz_stanga;
nec_poz_dreapta;
nec_val_stanga;
nec_val_dreapta

reprezentând că este nevoie să se proceseze pozițiile din intervalul

[nec_poz_stanga; nec_poz_dreapta]

și valorile din intervalul

[nec_val_stanga; nec_val_dreapta].

Se poate incrementa/decremenata fiecare din valorile care încep cu *av_* atât timp cât nu satisfac intervalele date de valorile *nec_*.

- procesarea unei poziții va actualiza valorile

nec_val_stanga și
nec_val_dreapta

dacă valoarea de pe poziția respectivă schimbă minimul sau maximul.

- procesarea unei valori va actualiza valorile

nec_poz_stanga și
nec_poz_dreapta

cu cea mai din stânga apariție a valorii, respectiv cu cea mai din dreapta apariție a valorii.

Această soluție are complexitate $O(N^2)$. Pentru a obține complexitate $O(N)$ este necesară procesarea tuturor elementelor dintr-o grupă construită anterior o singură dată. Mai exact, în pasul de incrementare/decrementare a unei valori *av_* se va procesa toată grupa poziției/valorii respective, în caz că aceasta a fost construită anterior.

Precizăm că soluții în complexitate $O(N \log N)$ pot lua 100 de puncte deasemenea.

19.4.2 *Cod sursă

19.4.3 *Rezolvare detaliată

19.5 hipersimetrie

Problema 5 - hipersimetrie

100 de puncte

O matrice hipersimetrică este o matrice pătratică definită recursiv astfel

1. Matricele de dimensiune 1×1 sunt hipersimetrice.
2. O matrice de dimensiune $N \times N$ ($N > 1$) este hipersimetrică, dacă îndeplinește simultan următoarele două condiții:
 - (a) Este simetrică vertical, orizontal, față de diagonala principală și față de diagonala secundară.

(b) Submatricele de dimensiuni $N/2 \times N/2$ (cu $N/2$ rotunjit în jos) situate în cele patru colțuri ale matricei sunt la rândul lor hipersimetrice.

O **matrice binară** este o matrice ale cărei elemente sunt 0 sau 1. **Valoarea** unei matrice binare hipersimetrice este numărul în baza 2 cu N^2 biți obținut prin concatenarea elementelor din matrice citite pe linii de la stânga la dreapta, de sus în jos.

Cerințe

Cunoscând N și K , să se calculeze a K -a valoare în ordine crescătoare dintre toate valorile matricelor binare hipersimetrice de dimensiune $N \times N$.

Date de intrare

Fișierul de intrare **hipersimetrie.in** conține pe prima linie numărul N . A doua linie conține un sir de caractere 0 sau 1 reprezentând valoarea lui K în baza 2 (se garantează că primul caracter al sirului este 1).

Date de ieșire

În fișierul de ieșire **hipersimetrie.out** afișați a K -a valoare în ordine crescătoare dintre toate valorile matricelor binare hipersimetrice de dimensiune $N \times N$. Deoarece această valoare poate fi foarte mare, se cere să afișați doar restul *modulo* 1.000.000.007 al acesteia.

Restricții și precizări

- $1 \leq N \leq 1.000.000.000$;
- $1 \leq K \leq 21.000.000$;
- Se garantează că pentru valoarea N dată există cel puțin K matrice binare hipersimetrice;
- Pentru teste în valoare de 27 puncte se garantează că $N \leq 1.500$
- Pentru alte teste în valoare de 62 puncte se garantează că $N \leq 1.000.000$
- Pentru alte teste în valoare de 11 puncte $N \leq 1.000.000.000$

Exemple

hipersimetrie.in	hipersimetrie.out	Explicații
3 100	186	$K = 100_2 = 4$. A 4-a matrice în ordinea crescătoare a valorii este 0 1 0 1 1 1 0 1 0 Valoarea sa este 010111010 în baza 2, adică 186 în baza 10.

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

Memorie: total **128 MB**

Dimensiune maximă a sursei: **20 KB**

19.5.1 Indicații de rezolvare

Adrian Panaete - C.N. "A.T. Laurian" Botoșani,
 Adrian Budau - Universitatea București

Ideea de bază este că datorită hipersimetriei asociind o valoare 0 sau 1 în anumite poziții această valoare se va regăsi în alte poziții. Se crează astfel o partiție a pozițiilor din matrice care împărtășesc prin proprietățile de simetrie o aceeași valoare. De exemplu să luăm $N = 13$. Atunci partitōnarea poate fi descrisă sugestiv astfel:

Oricare două poziții pe care avem aceeași valoare (de la 1 la 10) vom avea pe o matrice hipersimetrică același bit (0 sau 1). În total partitōa are 10 submulțimi. Pentru fiecare submulțime se va acorda fie bitul 0 fie bitul 1.

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

Observăm că numărul total de matrice este în acest caz 2^{10} .

Se observă că valoarea unei matrice depinde doar de valoarile setate pe prima poziție întâlnită în matrice, a unui reprezentant pentru fiecare dintre cele 10 submulțimi (marcate pe matrice cu gri). Este ca și cum am asocia valorii matricei, valoarea formată doar cu biți din aceste poziții speciale. În mod formal putem să ordonăm submulțimile de poziții după pozițiile speciale folosind drept criteriu ordinea între perechile (*linie, coloana*) ale acestor poziții.

Acum valoarea matricei va fi cu atât mai mare cu cât numărul binar asociat pozițiilor speciale este mai mare.

Pentru a calcula soluția cerută se vor procesa biții numărului K (decrementat) care reprezintă de fapt biții asociați fiecărei submulțimi în aşa fel încât să luăm în considerare toate pozițiile în care aceștia apar.

Pentru toate subtask-urile, cu excepția celui maximal, biți pot fi procesați pentru toate pozițiile speciale. De observat că K poate să aibă mai puțini biți decât numărul de submulțimi, dar asta este ca și cum ar avea biți de 0 la început. La subtask-ul maximal trebuie folosită restricția care spune că numărul K va avea cel mult 1000000 de biți astfel că pentru submulțimile care au pozițiile speciale situate pe primele linii ale matricei bitul corespunzător este 0 și putem ignora acele poziții.

Pentru ultimul task se pot procesa submulțimile de la cea mai mare ca indice (corespunzătoare ultimului bit al lui K) spre cele cu indice mai mici. Atunci când se ajunge la un indice care nu are un bit reprezentant în K (acest bit este implicit 0), ne putem opri din procesare.

În calculul efectiv al soluției se utilizează definiția recursivă a matricei depistând dimensiunile successive (din ce în ce mai mici) ale colțurilor care se obțin prin înjumătățirea dimensiunii.

Vom calcula valoarea corespunzătoare a soluției pentru aceste dimensiuni și vom folosi aceste valori pentru calculul valorii pentru matricea de dimensiune imediat superioară. Atunci când dimensiunea superioară este impară trebuie să avem grijă să utilizăm biții suplimentari ai lui K pe care urmează să îi adăugăm pe linia / coloana centrală. Calcularea noii valori se folosește de simetrie pentru a obține contribuția la soluție a colțurilor folosindu-ne de valoarea pe care o avem calculată pentru primul colț.

Valoarea din primul colț trebuie înmulțită cu puteri convenabile ale lui 2 pentru a obține contribuția celorlalte colțuri la valoare pentru noua dimensiune.

19.5.2 *Cod sursă

19.5.3 *Rezolvare detaliată

19.6 linegraph

Problema 6 - linegraph

100 de puncte

Cătălin, după ce s-a jucat destul la buncii, a ajuns acasă, dar nu a venit cu mâna goală, ci a adus cu el cel mai frumos arbore pe care l-a avut în jocul "TreeGCD", desenat pe o hârtie. Acasă, fratele lui mai mic a găsit această foaie și s-a gândit să își încerce talentul la desen. El vrea să transforme arborele într-un graf în următorul fel: fiecare muchie din arborele inițial devine un nod în noul graf, două noduri din noul graf sunt legate printre ele dacă și numai dacă cele două muchii corespunzătoare din arborele inițial au un nod în comun.

După ce a construit acest graf, el a aruncat hârtia cu arborele inițial. Când s-a întors de la școală, Cătălin, văzând ceea ce s-a întâmplat, nu a fost prea fericit. Din fericire, a aflat că voi puteți să reconstruiți arborele inițial, dacă vă dă graful.

Cerințe

Se dă numărul N de noduri, numărul M de muchii și cele M muchii din graf. Reconstruji arborele inițial. Este posibil ca fratele lui Cătălin să fi desenat greșit graful și să nu existe un arbore asociat.

Date de intrare

În fișierul de intrare **linegraph.in** pe prima linie se află un număr T , ce reprezintă numărul de teste din fișier. Pentru fiecare test pe prima linie se află două numere naturale N și M separate prin spațiu cu semnificațiile din enunț, iar pe următoarele M linii se află câte două numere separate prin spațiu, ce reprezintă nodurile care au o muchie între ele.

Date de ieșire

În fișierul de ieșire **linegraph.out** pe prima linie trebuie să se afișeze fie cuvântul *NU*, dacă nu există un arbore asociat grafului dat, fie cuvântul *DA*, dacă există arbore asociat, și în acest caz pe următoarea linie se va afișa un număr E , ce reprezintă numărul de noduri din arbore, și pe următoarele $E - 1$ linii se vor afișa câte două numere ce reprezintă perechile de noduri din arbore, care au o muchie între ele.

Restricții și precizări

- $1 \leq T \leq 10.000$;
- $1 \leq N \leq 1.000$;
- $0 \leq M \leq \frac{N(N-1)}{2}$;
- suma pătratelor tuturor N -urilor din fișierul de intrare nu depășește 1.000.000;
- pentru teste în valoare de 15 puncte, se garantează că există soluție și că arborele din care s-a construit graful are fie formă de *lanț*, fie are $N - 1$ frunze;
- pentru alte teste în valoare de 55 de puncte, se garantează că $N \leq 100$ și suma pătratelor tuturor N -urilor din fișierul de intrare nu depășește 10.000;
 - dacă există mai multe soluții, se poate afișa oricare dintre ele;
 - arborele din fișierul de ieșire cu E noduri va avea nodurile numerotate cu $1, 2, \dots, E$;
 - numerotarea efectivă a nodurilor din fișierul de ieșire nu este importantă - orice soluție ce renumeotează nodurile va fi considerată un răspuns corect.

Exemple

linegraph.in	linegraph.out	Explicații
2	DA	În fișierul de intrare avem un graf. Fiecare muchie din acest graf îi corespunde un nod din arborele din fișierul de ieșire. Astfel:
5 7	6	
3 2	1 2	
3 5	1 3	muchia (1,3) devine nodul 1, muchia (1,2) devine nodul 4,
3 1	3 4	muchia (3,4) devine nodul 3, muchia (3,5) devine nodul 2
2 5	3 5	și muchia (3,6) devine nodul 5.
2 1	3 6	Muchiile (1,3), (3,4), (3,5), (3,6) au toate nodul comun
1 5	NU	3, deci nodurile lor corespunzătoare din graf (1,3,2,5) au toate muchii între ele.
1 4		
3 1		În al doilea test nodul 3 este izolat și graful nu poate proveni din niciun arbore.
1 2		

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

Memorie: total **128 MB**

Dimensiune maximă a sursei: **20 KB**

19.6.1 Indicații de rezolvare

Andrei Costin Constantinescu - Universitatea Oxford Anglia

Solutie 15p

Dacă arborele din care s-a construit graful dat este *lanț*, atunci și graful este tot un lanț.

Dacă arborele este de tip stea (are $N-1$ frunze), atunci *graful dat este complet* (are $N*(N-1)/2$ muchii).

Se pot verifica cele două cazuri în timp liniar.

Solutie 55p / 100p

Se observă că graful dat trebuie să fie format din *clici* (graf complet) care să se intersecteze în noduri. Mai mult, un nod nu trebuie să facă parte din mai mult de două clici. În plus, graful trebuie să fie *conex*.

Dacă se respectă toate cerințele de mai sus, răspunsul este *DA*, și se poate reconstrui arborele inițial. În caz contrar, răspunsul este *NU*.

Există mai multe modalități de a rezolva cerințele. Una dintre ele este găsirea *componentelor biconexe*, după care se verifică dacă fiecare componentă este o clică și dacă un nod face parte din cel mult două componente biconexe.

Pentru reconstruirea arborelui vom eticheta fiecare clică cu un indice, adăugând o muchie între doi indici dacă cele două clici au un *nod critic* în comun.

Mai știm că fiecare clică trebuie să fie gradul în arbore egal cu dimensiunea ei în graful dat. Astfel, pentru clicile unde nu am adăugat destule muchii, vom crea noi indici și vom adăuga muchie între indicele clicii și noul indice.

Diferența dintre 55 de puncte și 100 de puncte o face complexitatea timp a implementării. Pentru 55 de puncte se poate implementa $O(n^3)$ iar pentru 100 de puncte $O(n^2)$.

19.6.2 *Cod sursă

19.6.3 *Rezolvare detaliată

Capitolul 20

ONI 2018

20.1 aranjare

Problema 1 - aranjare

100 de puncte

Ion are o stivă cu N elemente și vrea să le sorteze în ordine crescătoare de la bază spre vârf. Pentru a realiza acest lucru, el poate să achiziționeze M stive suplimentare și să efectueze K operații. O operație constă în a lua un element din vârful unei stive și a-l insera în vârful unei alte stive. Ion poate alege convenabil valorile lui M și K . Ajutați-l pe Ion să sorteze elementele astfel încât $M * K$ să aibă valoare cât mai mică și toate valorile să ajungă pe stiva inițială în ordine crescătoare de la bază spre vârf.

Cerințe

Afișați o secvență de K operații care folosesc M stive suplimentare în aşa fel încât să sortați elementele de pe stiva inițială în ordine crescătoare de la bază spre vârf.

Date de intrare

Fișierul de intrare **aranjare.in** va conține pe primul rând un număr natural nenul N . Pe al doilea rând se află o permutare a multimii $\{1, 2, \dots, N\}$ ce reprezintă valorile inițiale pe stiva lui Ion. Ultimul element din permutare este cel aflat în vârful stivei.

Date de ieșire

Fișierul de ieșire **aranjare.out** va conține pe primul rând numerele naturale M și K .

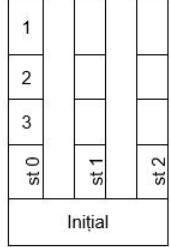
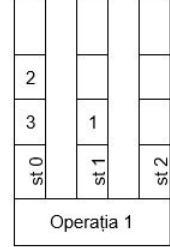
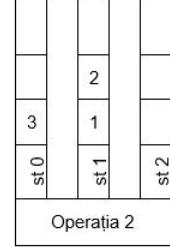
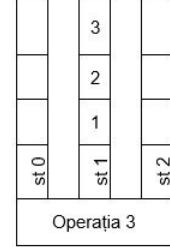
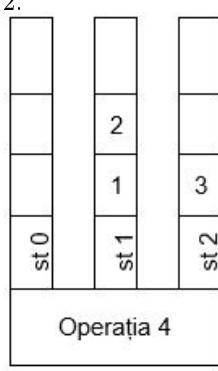
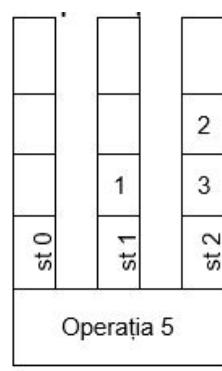
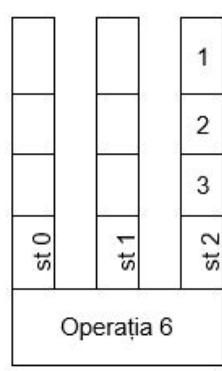
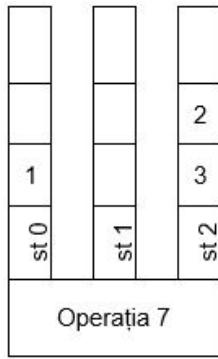
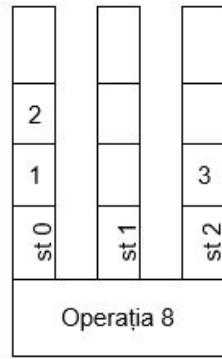
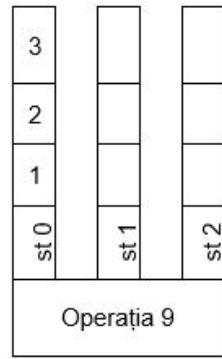
Pe următoarele K rânduri se vor scrie perechi de numere s t (câte o pereche pe fiecare rând) reprezentând mutarea elementului din vârful stivei s în vârful stivei t . Se consideră că stiva inițială a lui Ion are indicele 0, iar cele M stive suplimentare au indicii 1, 2, ..., M .

Pentru acordarea punctelor este necesar ca după executarea tuturor operațiilor indicate în fișierul de ieșire, elementele din stiva 0 să fie ordonate crescător de la bază spre vârf.

Restricții și precizări

- Pentru teste în valoare de 25 de puncte avem $N = 980$. Pentru celealte teste avem $N = 10\ 000$.
- Pentru testele cu $N = 980$, punctajele se acordă în modul următor:
 - Dacă $M * K \leq 60\ 000$, se acordă 100% din punctajul testului respectiv
 - Dacă $60\ 000 < M * K \leq 200\ 000$, se acordă 60% din punctajul testului respectiv
 - Dacă $200\ 000 < M * K \leq 3\ 000\ 000$ se acordă 20% din punctajul testului respectiv
- Pentru testele cu $N = 10\ 000$, punctajele se acordă în modul următor:
 - Dacă $M * K \leq 800\ 000$, se acordă 100% din punctajul testului respectiv
 - Dacă $800\ 000 < M * K \leq 6\ 000\ 000$, se acordă 60% din punctajul testului respectiv
 - Dacă $6\ 000\ 000 < M * K \leq 300\ 000\ 000$ se acordă 20% din punctajul testului respectiv

Exemple:

aranjare.in	aranjare.out	Explicații
3 3 2 1	2 9 0 1 0 1 0 1 1 2 1 2 1 2 2 0 2 0 2 0	S-au cumpărat 2 stive suplimentare, și s-au efectuat 9 operații. Primele trei operații mută elementele de pe stiva 0 pe stiva 1.    
		Următoarele trei operații mută elementele de pe stiva 1 pe stiva 2.
		  
		Ultimele trei operații mută elementele înapoi pe stiva 0, în ordinea sortată.
		  

Timp maxim de executare/test: **1.2** secunde

Memorie: total **128 MB** din care pentru stivă **32 MB**

Dimensiune maximă a sursei: **20 KB**

20.1.1 Indicații de rezolvare

Autor problema: Tamio-Vesa Nakajima

Soluție cu $M * K = O(N^2)$ - 20 puncte - Tamio-Vesa Nakajima

Luăm $M = N$. Mutăm fiecare element din stiva 0 pe una dintre celelalte stive. Apoi, le mutăm înapoi pe stiva inițială în ordinea corectă.

Soluție cu $M * K = O(N * \sqrt{N})$ - 60 puncte - Tamio-Vesa Nakajima

Luăm $M = \sqrt{N} + 1$. Împărțim elementele de pe stiva 0 cât mai egal pe celelalte stive. Se observă că avem destule stive să aplicăm algoritmul de 20 de puncte asupra fiecărei stive suplimentare, sortând astfel toate stivele suplimentare în ordine inversă. *Interclasăm* acum toate elementele

de pe stivele suplimentare, mutând mereu elementul minim de pe vârful stivelor suplimentare pe stiva inițială.

Soluție cu $M * K = O(N * \log(N))$ - 100 puncte - Constantinescu Andrei Costin

Luăm $M = 2$. Definim o procedură care sortează primele h elemente din vârful unei stive cu ajutorul a 2 stive auxiliare, pe care o vom folosi să sortăm elementele de pe stiva inițială. Pentru a face acest lucru, inițial distribuim cele h elemente din vârful stivei cât mai egal la cele 2 stive auxiliare, apoi apelăm recursiv procedura pentru a sorta aceste elemente descrescător, apoi *interclasăm* (ca la soluția de 60 de puncte) elementele acestea, mutându-le înapoi pe prima stivă. Această soluție este analogă algoritmului *merge sort*.

Soluție cu $M * K = O(N * \log(N))$ - 100 puncte - Daniel Posdarascu

Luăm $M = 2$. Definim o procedură care sortează primele h elemente din vârful unei stive cu ajutorul a 2 stive auxiliare, pe care o vom folosi să sortăm elementele de pe stiva inițială. Pentru a face acest lucru, inițial mutăm cele mai mici $h/2$ elemente din vârful stivei pe una dintre stivele auxiliare, apoi mutăm restul elementelor pe cealaltă stivă auxiliară, apoi apelăm recursiv procedura pentru a sorta aceste elemente distribuite descrescător, apoi *interclasăm* (ca la soluția de 60 de puncte) elementele acestea, mutându-le înapoi pe prima stivă. Această soluție este analogă algoritmului *quicksort*.

Soluție cu $M * K = O(N * \log(N))$ - 100 puncte - Bogdan Ciobanu

Luăm $M = 2$. Considerăm scrierea în baza 2 a numerelor. Pentru fiecare bit care apare în vreun număr din stivă, vom parta elementele de pe stiva inițială în funcție de acel bit (adică vom muta toate elementele care conțin acel bit pe stiva 1, toate cele care nu conțin acel bit pe stiva 2, vom muta toate elementele din stiva 1 pe stiva 0, și apoi vom muta toate elementele din stiva 2 pe stiva 0). Această soluție este analogă algoritmului *radix sort*.

20.1.2 Cod sursă

Listing 20.1.1: bogd_ciob_aranjare100.cpp

```

1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4
5 using namespace std;
6
7 int main()
8 {
9     //ifstream cin("aranjare.in");
10    //ofstream cout("aranjare.out");
11
12    int n;
13    cin >> n;
14
15    vector<int> stk(n);
16
17    for (auto& iter : stk)
18    {
19        cin >> iter;
20        --iter;
21    }
22
23    vector<pair<int, int>> operations;
24
25    for (int bit = 0; (1 << bit) < n; ++bit)
26    {
27        int num_set_bits = 0;
28        for (int i = n - 1; i >= 0; --i)
29        {
30            const int stk_idx = (stk[i] >> bit & 1);
31            operations.emplace_back(0, 1 + stk_idx);
32            num_set_bits += stk_idx;
33        }
34
35        for (int i = 0; i < n - num_set_bits; ++i)
36            operations.emplace_back(1, 0);

```

```

37         for (int i = 0; i < num_set_bits; ++i)
38             operations.emplace_back(2, 0);
39
40         stable_partition(stk.begin(), stk.end(), [&] (const int el) {
41             return !(el >> bit & 1);
42         });
43     }
44
45     cout << 2 << " " << operations.size() << '\n';
46
47     for (auto&& operation : operations)
48         cout << operation.first << " " << operation.second << '\n';
50 }
```

Listing 20.1.2: mapa_radix.cpp

```

1 #include <iostream>
2 #include <cassert>
3 #include <vector>
4
5 using namespace std;
6 using Matrix = vector<vector<int>>;
7
8 const int NMAX = 5e5;
9
10 int main()
11 {
12     int n;
13     cin >> n;
14     assert(n <= NMAX);
15
16     Matrix st(3, vector<int>(n));
17     vector<int> top(3, -1);
18     vector<bool> f(n + 1);
19
20     for (int i = 0; i < n; ++i)
21     {
22         cin >> st[0][i];
23         assert(1 <= st[0][i] && st[0][i] <= n);
24         assert(f[st[0][i]] == false);
25         f[st[0][i]] = true;
26     }
27
28     int maxBit;
29     for (maxBit = 1; (1 << maxBit) < n; ++maxBit);
30
31     vector<pair<int, int>> ans;
32 //    cerr << "MAXBIT = " << maxBit << "\n";
33     for (int b = 0; b <= maxBit; ++b) {
34         for (int i = n - 1; i >= 0; --i) {
35             if (st[0][i] & (1 << b)) {
36                 ans.emplace_back(0, 1);
37                 st[1][++top[1]] = st[0][i];
38             }
39             else {
40                 ans.emplace_back(0, 2);
41                 st[2][++top[2]] = st[0][i];
42             }
43         }
44
45         top[0] = -1;
46         for (int j = 2; j >= 1; --j) {
47             for (int i = top[j]; i >= 0; --i) {
48                 ans.emplace_back(j, 0);
49                 --top[j];
50                 st[0][++top[0]] = st[j][i];
51             }
52         }
53     }
54
55     cout << 2 << " " << ans.size() << '\n';
56     for (auto move : ans) {
57         cout << move.first << " " << move.second << '\n';
58     }
```

```

59     return 0;
60 }
```

Listing 20.1.3: tamio_100.cpp

```

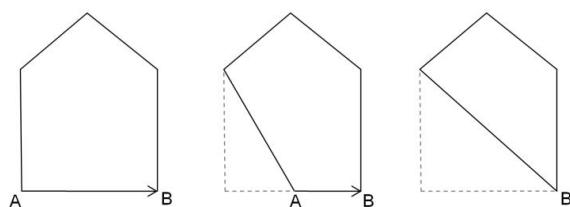
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 vector<int> v[3] = {};
6 vector<pair<int, int>> sol;
7
8 void do_op(int x, int y)
9 {
10     v[y].push_back(v[x].back());
11     v[x].pop_back();
12     sol.emplace_back(x, y);
13 }
14
15 void sort_stack(int s, int depth, int a1, int a2, int mult = 1)
16 {
17     if(depth == 1) return;
18
19     for(int i = 0; i < depth; ++i)
20         do_op(s, i < depth / 2 ? a1 : a2);
21
22     sort_stack(a1, depth / 2, s, a2, -mult);
23     sort_stack(a2, depth - depth / 2, s, a1, -mult);
24
25     int i1 = depth / 2, i2 = depth - depth / 2;
26
27     while(i1 || i2)
28     {
29         if((i1 && i2 && v[a1].back() * mult < v[a2].back() * mult) || !i2)
30         {
31             do_op(a1, s);
32             --i1;
33         }
34         else
35         {
36             do_op(a2, s);
37             --i2;
38         }
39     }
40 }
41
42 int main()
43 {
44     //ifstream f("aranjamente.in");
45     //ofstream g("aranjamente.out");
46
47     auto& f = cin;
48     auto& g = cout;
49
50     int n;
51     f >> n;
52     for(int i = 1, x; i <= n; ++i)
53     {
54         f >> x;
55         v[0].push_back(x);
56     }
57
58     sort_stack(0, n, 1, 2);
59
60
61     g << 2 << ' ' << sol.size() << endl;
62     for(auto& x : sol) g << x.first << ' ' << x.second << endl;
63
64     assert(v[0].size() == n);
65     assert(is_sorted(begin(v[0]), end(v[0])));
66
67     return 0;
68 }
```

20.1.3 *Rezolvare detaliată

20.2 poligon

Problema 2 - poligon

Se consideră un poligon convex cu N laturi. Se vor efectua $N - 1$ mutări. O mutare constă în alegerea a două puncte A și B vecine pe poligon și mutarea punctului A în B (vezi figura). Costul mutării este egal cu distanța euclidiană dintre A și B . După mutare punctul A este assimilat de B , iar procesul se reia pe noul poligon. Se cere costul total minim al unei succesiuni de $N - 1$ mutări care reduce poligonul la un singur punct, precum și o modalitate de a obține acest cost.



100 de puncte

Cerințe

Dându-se T poligoane convexe, să se determine:

1. Costul minim ans al unei succesiuni de mutări care reduce poligonul la un singur punct;
2. O succesiune de mutări de cost minim.

Date de intrare

Fisierul de intrare **poligon.in** conține pe prima linie un număr întreg p , reprezentând numărul cerinței ce se cere a fi rezolvată.

Pe a doua linie a fișiereului de intrare se va afla T , reprezentând numărul de poligoane ce urmează să fie citite. Apoi, urmează cele T teste. Fiecare test are următoarea structură:

- pe prima linie numărul natural N , reprezentând numărul de laturi ale poligonului;
- pe următoarele N linii câte 2 numere întregi x și y , separate printr-un spațiu, reprezentând coordonatele vârfurilor poligonului curent. Vârfurile sunt date în ordine trigonometrică.

Date de ieșire

Fisierul de ieșire **poligon.out** va conține, în funcție de valoarea lui p , următoarele informații:

1. Dacă $p = 1$ se rezolvă doar cerința 1. Pentru fiecare dintre cele T teste se va afișa câte un număr real ans pe o linie, cu semnificația din enunț.
2. Dacă $p = 2$ se rezolvă doar cerința 2. Pentru fiecare din cele T teste se vor afișa câte $N - 1$ linii, fiecare dintre aceste fiind de forma $A\ B$, reprezentând mutările în ordinea în care acestea se efectuează.

Restricții și precizări

- $1 \leq T \leq 5$
- $1 \leq N \leq 2\ 000$
- Pentru toate vârfurile poligonului $-1\ 000\ 000 \leq x, y \leq 1\ 000\ 000$
- Nu vor exista 2 vârfuri ale poligonului aflate la aceleași coordonate.
- Poligonul nu este neapărat strict convex. Cu alte cuvinte, pot exista oricâte vârfuri consecutive coliniare.
 - Pentru teste în valoare de 5 puncte, $N \leq 7$;
 - Pentru alte teste în valoare de 10 puncte $N \leq 15$;
 - Pentru alte teste în valoare de 15 puncte $N \leq 50$;
 - Pentru alte teste în valoare de 15 puncte $N \leq 100$;
 - Pentru alte teste în valoare de 15 puncte $N \leq 500$;
 - Pentru alte teste în valoare de 40 puncte $N \leq 2\ 000$;
 - Pentru rezolvarea cerinței 1. se acordă 80% din punctajul asociat testului.
 - Pentru rezolvarea cerinței 2. se acordă 20% din punctajul asociat testului.
 - Valoarea lui ans se va considera corectă dacă aceasta diferă față de răspunsul corect prin maxim 10^{-6} .
 - ATENȚIE! După o mutare $A\ B$ (în urma căreia vârful A a fost assimilat de vârful B), o mutare de forma $A\ C$ sau $C\ A$ va fi considerată invalidă.

Exemplu:

poligon.in	poligon.out	Explicații
1 2 4 0 0 1 0 1 1 0 1 5 0 0 8 0 8 10 4 20 0 10	3 36.770329614269	În acest caz $p = 1$, deci se va rezolva doar cerința 1. Fișierul conține $T = 2$ poligoane. Vârfurile primului poligon sunt $(0, 0), (1, 0), (1, 1), (0, 1)$. Costul minim asociat unei succesiuni de mutări de cost minim este 3. Vârfurile celui de-al doilea poligon sunt $(0, 0), (8, 0), (8, 10), (4, 20), (0, 10)$. Costul minim asociat unei succesiuni de mutări de cost minim este 36.770329614269.
2 2 4 0 0 1 0 1 1 0 1 5 0 0 8 0 8 10 4 20 0 10	3 2 4 1 2 1 4 3 5 3 3 2 2 1 4 1 2 1 0 0 8 0 8 10 4 20 0 10	În acest caz $p = 2$, deci se va rezolva doar cerința 2. Fișierul conține $T = 2$ poligoane. Vârfurile primului poligon sunt $(0, 0), (1, 0), (1, 1), (0, 1)$. Costul minim asociat unei succesiuni de mutări de cost minim este 3. O succesiune de mutări de cost minim este: Vârfurile celui de-al doilea poligon sunt $(0, 0), (8, 0), (8, 10), (4, 20), (0, 10)$. Costul minim asociat unei succesiuni de mutări de cost minim este 36.770329614269. O succesiune de mutări de cost minim este:

Timp maxim de executare/test: **1.6**

Memorie: total **256 MB** din care pentru stivă **32 MB**

Dimensiune maximă a sursei: **20 KB**

20.2.1 Indicații de rezolvare

Andrei Costin Constantinescu, Oxford University
Bogdan Ciobanu, Universitatea București

60 puncte - $O(N^3)$

Notăm cu $v[1], v[2], \dots, v[N]$ cele N vârfuri ale poligonului, în ordinea în care au fost date.

Se calculează dinamica $dp[i][j] =$ costul minim pentru a comprima subsecvența de vârfuri luate în ordine trigonometrică, începând cu vârful i și terminând cu vârful j , astfel încât vârfurile i și j să fie nemîșcate. Putem scrie recurența $dp[i][j] = \min_{k=i}^{j-1} \{ dp[i][k] + dp[k][j] + \min(\text{dist}(v[i], v[k]), \text{dist}(v[k], v[j])) \}$ pentru $i < k < j$. Mai avem că $dp[i][i+1] = 0$, iar răspunsul este $\min_{i=0}^{N-1} \{ dp[i][j] + dp[j][i] + \text{dist}(v[i], v[j]) \}$.

100 puncte - $O(N^2)$

Construim un *graf neorientat complet* cu N noduri astfel încât fiecare vârf al poligonului să reprezinte un nod în graf, iar costul muchiei de la nodul A la nodul B să fie dat de distanța euclidiană dintre vârful A și vârful B . Analizând cu atenție modul în care poligonul nostru evoluează în poligoane mai mici, deducem următoarele:

- Toate costurile mutărilor efectuate pe parcursul unei secvențe sunt costuri de muchii din graf.
- Nu vom muta niciodată un punct de 2 ori, deci mulțimea muchiilor asociate diagonalelor corespunzătoare mutărilor efectuate într-o secvență fixată descrie un *arbore parțial* al grafului.
- Așadar, cum toate secvențele de mutări descriu un arbore parțial în graf, costul minim nu poate fi mai mic decât costul *Arborelui Parțial de Cost Minim* (APM) al grafului;

Acum, observația cheie este că întotdeauna va exista o succesiune de $N - 1$ mutări ce descrie exact APM-ul grafului dat. Această observație nu este una trivială, așa că o vom demonstra printr-o serie de leme:

Lema 1 Muchiile APM-ului grafului nu se intersectează unele cu celelalte în alte puncte decât, eventual, vârfurile acestora.

Demonstratie Presupunem prin absurd că 2 muchii AB și CD din APM se intersectează. Atunci, în cazul de față, schimbând aceste 2 muchii fie cu AC și BD , fie cu AD și BC , în funcție de restul muchiilor APM-ului, am obține un arbore parțial de cost mai mic, contrazicând minimalitatea arborelui de la care am plecat. Detaliile sunt lăsate ca temă cititorului.

Lema 2 Pentru orice APM al grafului considerat, există cel puțin 2 laturi ale poligonului cu proprietatea că:

1. Muchiile asociate lor fac parte din APM.
2. Pentru fiecare dintre ele, nodul asociat cel puțin unuia dintre vârfuri este o frunză în APM.

Demonstratie Prin inducție, luând în considerare 2 cazuri:

1. APM-ul este un *lanț*, caz în care toate muchiile au asociate laturi ale poligonului, dintre care 2 respectă și condiția de a avea unul din capete *frunză în arbore*.

2. Există cel puțin o muchie a APM-ului care nu are asociată o latură a poligonului (să îi numim diagonala asociată *diagonală specială*). În acest caz, aplicând ipoteza de inducție pe cele 2 poligoane formate prin tăierea poligonului nostru cu diagonala specială, putem deduce că există cel puțin 2 muchii cu proprietățile date în poligonul inițial.

Teorema 3 Există o succesiune de mutări care reduc un poligon la un singur punct cu cost asociat exact costul APM-ului grafului asociat.

Demonstratie Inductiv, se alege una dintre cele 2 muchii generate de Lema 2, se reduce poligonul astfel încât vârful asociat nodului frunză să dispară, și se folosește ipoteza de inducție pe poligonul astfel rezultat.

Nota: Procedeul prezentat, de a construi APM-ul și de a tot muta *frunze* ce au asociate laturi ale poligonului în *tatăl* lor reprezintă exact metoda de construcție a soluției.

Pentru implementare se va folosi *algoritmul lui Prim*, care implementat fără *cozi de priorități* rulează în $O(n^2)$ în *graf dens*.

20.2.2 Cod sursă

Listing 20.2.1: n2_costel_100.cpp

```

1 // Andrei Constantinescu - Oxford University
2 // O(N^2)
3 #include <bits/stdc++.h>
4
5 using namespace std;
6
7 typedef long long int lint;
8 const int NMAX = 2000 + 5;
9
10 int N;
11 struct Point
12 {
13     int x, y;
14     Point(int _x = 0, int _y = 0):
15         x(_x), y(_y) {}
16
17     friend inline long double dist(const Point &A, const Point &B)
18     {
19         return sqrtl(1LL * (A.x - B.x) * (A.x - B.x) +
20                     1LL * (A.y - B.y) * (A.y - B.y));
21     }
22     friend inline lint ccw(const Point &A, const Point &O, const Point &B)
23     {
24         return 1LL*(A.x - O.x)*(B.y - O.y) - 1LL*(A.y - O.y)*(B.x - O.x);
25     }
26 } points[NMAX];
27
28 const long double INF = 1E9L;
29
30 long double costMin[NMAX];
31 bool vis[NMAX];

```

```

32 int father[NMAX];
33
34 vector <int> mst[NMAX];
35
36 ifstream fin("poligon.in");
37 ofstream fout("poligon.out");
38
39 void read()
40 {
41     for (int i = 1; i <= N; ++ i)
42         father[i] = 0, mst[i].clear();
43
44     fin >> N;
45     assert(3 <= N && N <= 2000);
46     for (int i = 1; i <= N; ++ i)
47     {
48         fin >> points[i].x >> points[i].y;
49         assert(-1000000 <= points[i].x && points[i].x <= 1000000);
50         assert(-1000000 <= points[i].y && points[i].y <= 1000000);
51     }
52
53     // Check for convexity
54     for (int i = 1; i + 2 <= N; ++ i)
55         assert(ccw(points[i], points[i + 1], points[i + 2]) <= 0);
56     assert(ccw(points[N - 1], points[N], points[1]) <= 0);
57     assert(ccw(points[N], points[1], points[2]) <= 0);
58     // Not collinear
59     lint area = 0;
60     for (int i = 1; i <= N; ++ i)
61         area += ccw(points[i], 0, points[i % N + 1]);
62     assert(area > 0);
63 }
64
65 long double solvePrim()
66 {
67     for (int i = 1; i <= N; ++ i)
68         costMin[i] = INF, vis[i] = false;
69     costMin[1] = 0;
70
71     // Prim's Algo
72     long double cost = 0;
73     for (int phase = 1; phase <= N; ++ phase)
74     {
75         int node = -1;
76         long double dMin = INF + 5;
77         for (int i = 1; i <= N; ++ i)
78             if (!vis[i] && costMin[i] < dMin)
79                 node = i, dMin = costMin[i];
80
81         vis[node] = true;
82         cost += costMin[node];
83         mst[node].push_back(father[node]);
84         mst[father[node]].push_back(node);
85
86         for (int i = 1; i <= N; ++ i)
87             if (!vis[i])
88             {
89                 const long double dst = dist(points[node], points[i]);
90                 if (dst < costMin[i])
91                 {
92                     costMin[i] = dst;
93                     father[i] = node;
94                 }
95             }
96     }
97
98     return cost;
99 }
100
101 int nxt[NMAX], prc[NMAX], deg[NMAX];
102
103 vector <pair <int, int> > printSolution()
104 {
105     for (int i = 1; i < N; ++ i)
106         nxt[i] = i + 1, prc[i + 1] = i;
107     nxt[N] = 1, prc[1] = N;

```

```

108
109     for (int i = 1; i <= N; ++ i)
110         deg[i] = mst[i].size(), vis[i] = false;
111
112     vector<pair<int, int>> ans;
113
114     for (int phase = 1; phase < N; ++ phase)
115     {
116         for (int i = 1; i <= N; ++ i)
117             if (!vis[i] && deg[i] == 1)
118             {
119                 int node;
120                 for (auto it: mst[i])
121                     if (!vis[it])
122                     {
123                         node = it;
124                         break;
125                     }
126                 if (node == nxt[i] || node == prc[i])
127                 {
128                     ans.push_back({i, node});
129                     vis[i] = true;
130                     -- deg[node];
131                     nxt[prc[i]] = nxt[i], prc[nxt[i]] = prc[i];
132                 }
133             }
134     }
135
136     return ans;
137 }
138
139 void test()
140 {
141     read();
142     fout << fixed << setprecision(12);
143     fout << solvePrim() << '\n';
144     vector<pair<int, int>> sol = printSolution();
145     for (auto it: sol)
146         fout << it.first << ' ' << it.second << '\n';
147 }
148
149 int main()
150 {
151     int T = 0;
152     fin >> T;
153     while (T--)
154         test();
155     return 0;
156 }
```

Listing 20.2.2: n5_bogdan_30.cpp

```

1 // O(N^5)
2 #include <iostream>
3 #include <algorithm>
4 #include <cmath>
5 //#include <tuple>
6
7 using namespace std;
8
9 using cost_t = long double;
10
11 const int kMaxN = 50;
12 const cost_t kInf = 1e9;
13
14 struct Point
15 {
16     int x, y;
17
18     friend cost_t distance(const Point& a, const Point& b)
19     {
20         return sqrtl(cost_t(a.x-b.x)*(a.x-b.x)+cost_t(a.y-b.y)*(a.y-b.y));
21     }
22 };
23
```

```

24  ifstream fin("poligon.in");
25  ofstream fout("poligon.out");
26
27  cost_t cost[kMaxN][kMaxN];
28  cost_t mn_cost[kMaxN][kMaxN][kMaxN];
29  pair<int, int> mn_cost_link[kMaxN][kMaxN][kMaxN];
30  Point p[kMaxN];
31
32  void ComputeTransitions(const int left, const int right, const int endpoint)
33  {
34      cost_t& ans = mn_cost[left][right][endpoint];
35      pair<int, int>& ans_link = mn_cost_link[left][right][endpoint];
36      ans = kInf;
37
38      for (int split = left + 1; split <= endpoint; ++split)
39          for (int oth_endpoint = left; oth_endpoint < split; ++oth_endpoint)
40              if (ans > mn_cost[left][split - 1][oth_endpoint] +
41                  mn_cost[split][right][endpoint] +
42                  cost[endpoint][oth_endpoint])
43              {
44                  ans = mn_cost[left][split - 1][oth_endpoint] +
45                      mn_cost[split][right][endpoint] +
46                      cost[endpoint][oth_endpoint];
47                  ans_link = make_pair(split, oth_endpoint);
48              }
49
50      for (int split = endpoint + 1; split <= right; ++split)
51          for (int oth_endpoint = split; oth_endpoint <= right; ++oth_endpoint)
52              if (ans > mn_cost[left][split - 1][endpoint] +
53                  mn_cost[split][right][oth_endpoint] +
54                  cost[endpoint][oth_endpoint])
55              {
56                  ans = mn_cost[left][split - 1][endpoint] +
57                      mn_cost[split][right][oth_endpoint] +
58                      cost[endpoint][oth_endpoint];
59                  ans_link = make_pair(split, oth_endpoint);
60              }
61  }
62
63  void BuildSolution(const int left, const int right, const int last_endpoint)
64  {
65      if (left == right)
66          return;
67
68      int split, prev_endpoint;
69      tie(split, prev_endpoint) = mn_cost_link[left][right][last_endpoint];
70      const int left_endpoint = min(prev_endpoint, last_endpoint);
71      const int right_endpoint = prev_endpoint ^ last_endpoint ^ left_endpoint;
72
73      BuildSolution(left, split - 1, left_endpoint);
74      BuildSolution(split, right, right_endpoint);
75
76      fout << prev_endpoint + 1 << ' ' << last_endpoint + 1 << '\n';
77  }
78
79  void SolveTest()
80  {
81      int n; fin >> n;
82      for (int i = 0; i < n; ++i)
83          fin >> p[i].x >> p[i].y;
84
85      for (int i = 0; i < n; ++i)
86          for (int j = i + 1; j < n; ++j)
87              cost[i][j] = cost[j][i] = distance(p[i], p[j]);
88
89      for (int i = n - 1; i >= 0; --i)
90      {
91          mn_cost[i][i][i] = 0;
92          for (int j = i + 1; j <= n; ++j)
93              for (int k = i; k <= j; ++k)
94                  ComputeTransitions(i, j, k);
95      }
96
97      int best_endpoint = 0;
98      for (int endpoint = 1; endpoint < n; ++endpoint)
99          if (mn_cost[0][n - 1][best_endpoint] > mn_cost[0][n - 1][endpoint])

```

```

100         best_endpoint = endpoint;
101
102     fout << mn_cost[0][n - 1][best_endpoint] << '\n';
103     BuildSolution(0, n - 1, best_endpoint);
104 }
105
106 int main()
107 {
108     fout.precision(12);
109     fout.setf(ios::fixed, ios::floatfield);
110
111     int num_tests; fin >> num_tests;
112     while (num_tests--> 0)
113         SolveTest();
114
115     fin.close();
116     fout.close();
117 }
```

20.2.3 *Rezolvare detaliată

20.3 tricolor

Problema 3 - tricolor

100 de puncte

Tanaka are un arbore (un *tri*) cu N noduri numerotate de la 1 la N . El vrea să coloreze nodurile arborelui în alb sau negru astfel încât numărul de perechi (neordonate) de noduri înfrățite să fie maxim. Două noduri sunt înfrățite dacă și numai dacă ambele sunt albe și fie sunt legate direct printr-o muchie, fie lanțul elementar unic dintre ele conține doar noduri negre.

Cerințe

Dându-se un arbore cu N noduri, să se afle numărul maxim de perechi de noduri înfrățite ale sale care se poate obține.

Date de intrare

Fisierul de intrare **tricolor.in** va conține pe primul rând un număr natural nenul T ce reprezintă numărul de teste. Urmează T teste, fiecare test va descrie un arbore pentru care trebuie să se rezolve cerința. Pe primul rând al unui test apare un număr natural N ce reprezintă numărul de noduri ale arborelui din testul respectiv. Pe următoarele $N - 1$ rânduri vor apărea câte o pereche de numere întregi x y separate printr-un spațiu, care indică existența unei muchii între nodul x și nodul y .

Date de ieșire

Fisierul de ieșire **tricolor.out** va conține T rânduri. Fiecare rând va conține soluția pentru câte un test, în aceeași ordine ca în fisierul de intrare.

Restricții și precizări

- $1 \leq T \leq 10$
- $1 \leq N \leq 5\,000$
- Într-un test oarecare, $1 \leq x, y \leq N$, $x \neq y$
- Pentru 5 puncte, $T = 1$ și $N \leq 15$
- Pentru alte 10 puncte, $T = 1$ și $N \leq 20$
- Pentru alte 5 puncte, toți arborii descriși au exact 2 frunze și $N \leq 500$
- Pentru alte 10 puncte, pentru toți arborii descriși există exact două noduri ale arborelui de care se leagă toate frunzele, situate la capetele unui *lanț elementar* și $N \leq 500$.
- Pentru alte 50 de puncte, $N \leq 500$
- Pentru alte 20 de puncte, nu există restricții suplimentare.

Exemplu:

tricolor.in	tricolor.out	Explicații
2 8 1 2 2 3 2 4 4 5 5 6 6 7 6 8 2 1 2	7 1	<p>$T = 2$, avem două teste în fișierul de intrare.</p> <p>în primul test arborele are 8 noduri și cu o colorare optimă putem obține un număr maxim de 7 perechi de noduri înfrațite.</p> <p>Perechile înfrațite sunt: (1, 3), (1, 4), (3, 4), (4, 5), (5, 7), (5, 8), (7, 8)</p> <p>în al doilea test arborele are 2 noduri legate cu o singură muchie. Este optim să colorăm ambele noduri în alb obținând o pereche de noduri înfrațite.</p>

Timp maxim de executare/test: **1.6** secunde

Memorie: total **256 MB** din care pentru stivă **32 MB**

Dimensiune maximă a sursei: **20 KB**

20.3.1 Indicații de rezolvare

Autor problema: Tamio-Vesa Nakajima

Solutie in $O(N^2)$ - 100 puncte - Tamio-Vesa Nakajima

Stabilim că 1 este rădăcina arborelui.

Se rezolvă problema cu ajutorul *programării dinamice*. Mai întâi, fie:

$nrfrunze[i]$ = numărul de frunze al subarborelui cu rădăcina la i

Folosim următoarea matrice:

$d[i][j]$ = numărul maxim de perechi de noduri înfrațite, considerând doar subarborele cu rădăcina la i , și fixând că sunt exact j noduri albe x la care se poate ajunge de la i trecând doar prin x și posibil câteva noduri negre.

Pentru a calcula dinamica pentru un nod oarecare i , ai cărui fii sunt $f[1] \dots f[nrfii]$, vom considera o a doua dinamică:

$d2[j][k]$ = numărul maxim de perechi de noduri înfrațite, considerând doar subarborele cu rădăcina la i și primii j fii ai săi, fixând că sunt exact k noduri albe x la care se poate ajunge de la i trecând doar prin x și posibil câteva noduri negre, și fixând că i este colorat în negru.

Recurența acestei dinamici este:

$d2[j+1][k] = \max(d2[j][k-l]+d[f[j+1]][l]+(k-l)*l)$ pentru $0 \leq l \leq nrfrunze[f[j+1]]$

În această recurență, l reprezintă numărul de noduri albe x la care se poate ajunge de la $f[j+1]$ trecând doar prin x și posibil câteva noduri negre ce apar în subarborele lui $f[j+1]$.

Mai calculăm un sir:

$best_daca_tata_alb[i]$ = numărul maxim de perechi înfrațite în subarborele care conține tatăl lui i și subarborele cu rădăcina în i dacă fixăm că tatăl lui i este alb.

Astfel:

$best_daca_tata_alb[i] = \max(d[i][j] + j)$ pentru $1 \leq j \leq nrfrunze[i]$

În această formulă, j reprezintă numărul de noduri albe x la care se poate ajunge de la i trecând doar prin x și posibil câteva noduri negre ce apar în subarborele lui i .

Calculând acestea, cum nodul i este fie alb, fie negru, putem calcula d astfel:

$d[i][1] = \max(d2[nrfii][1], best_daca_tata_alb[f[1]] + \dots + best_daca_tata_alb[f[nrfii]])$
 $d[i][j] = d2[nrfii][j]$ pentru $1 < j \leq nrfunze[i]$

Soluția problemei se află în:

$\max(d[1][1], d[1][2], \dots, d[1][nrfrunze[1]])$

Cu ajutorul inducției se poate demonstra cu această soluție folosește cel mult $O(N^2)$ operații.

Un alt mod de a demonstra se bazează pe faptul că numărul de operații efectuat de această soluție este egal cu numărul de perechi neorientate de noduri, numărate în funcție de locul unde pică cel mai adânc strămoș comun al lor.

20.3.2 Cod sursă

Listing 20.3.1: alex_80.cpp

```
1 #include <stdio.h>
2 #include <bits/stdc++.h>
3
4 using namespace std;
5
6 #define ll long long
7 #define ld long double
8 #define pb push_back
9 #define mp make_pair
10 #define pii pair<int, int>
11 #define pll pair<ll, ll>
12 #define pdd pair<ld, ld>
13 #define all(x) (x).begin(), (x).end()
14 #define fi first
15 #define se second
16
17 const int kMaxN = 5005;
18
19 int n;
20 int sub[kMaxN];
21 vector<int> v[kMaxN];
22 int black[kMaxN][kMaxN];
23 int white[kMaxN];
24
25 void Reset()
26 {
27     for (int i = 1; i <= n; i++)
28     {
29         v[i].clear();
30         sub[i] = 0;
31     }
32
33     memset(black, -1, sizeof(black));
34     memset(white, -1, sizeof(white));
35 }
36
37 void Dfs(int x)
38 {
39     sub[x] = 1;
40     white[x] = 0;
41     black[x][0] = 0;
42
43     for (auto y : v[x])
44     {
45         if (!sub[y])
46         {
47             Dfs(y);
48             sub[x] += sub[y];
49
50             for (int i = sub[x] - 1; i >= 1; i--)
51             {
52                 for (int j = 0; j <= min(i, sub[y]); j++)
53                     if (black[x][i - j] != -1 && black[y][j] != -1)
54                         black[x][i] = max(black[x][i],
55                                           black[x][i - j] + black[y][j] + j * (i - j));
56
57             if (black[x][i - 1] != -1)
```

```

58         black[x][i] = max(black[x][i], black[x][i - 1] + white[y] + i - 1);
59     }
60
61     int best_from_y = white[y] + 1;
62     for (int i = 0; i <= sub[y]; i++)
63     {
64         if (black[y][i] != -1)
65             best_from_y = max(best_from_y, black[y][i] + i);
66
67         white[x] += best_from_y;
68     }
69 }
70
71 int main()
72 {
73     cin.sync_with_stdio(false);
74
75     ifstream cin("tricolor.in");
76     ofstream cout("tricolor.out");
77
78     int t;
79     cin >> t;
80     for (; t; t--)
81     {
82         cin >> n;
83         Reset();
84
85         for (int i = 1; i < n; i++)
86         {
87             int x, y;
88             cin >> x >> y;
89             v[x].pb(y);
90             v[y].pb(x);
91         }
92
93         Dfs(1);
94
95         int ans = white[1];
96         for (int i = 0; i <= n; i++)
97         {
98             ans = max(ans, black[1][i]);
99         }
100
101        cout << ans << '\n';
102    }
103
104    return 0;
105 }
```

Listing 20.3.2: alex_100.cpp

```

1 #include <stdio.h>
2 #include <bits/stdc++.h>
3
4 using namespace std;
5
6 #define ll long long
7 #define ld long double
8 #define pb push_back
9 #define mp make_pair
10 #define pii pair<int, int>
11 #define pll pair<ll, ll>
12 #define pdd pair<ld, ld>
13 #define all(x) (x).begin(), (x).end()
14 #define fi first
15 #define se second
16
17 const int kMaxN = 5005;
18
19 int n;
20 int sub[kMaxN];
21 vector<int> v[kMaxN];
22 int black[kMaxN][kMaxN];
23 int new_black[kMaxN];
24 int white[kMaxN];
```

```

25
26 void Reset()
27 {
28     for (int i = 1; i <= n; i++)
29     {
30         v[i].clear();
31         sub[i] = 0;
32     }
33
34     memset(black, -1, sizeof(black));
35     memset(white, -1, sizeof(white));
36 }
37
38 void Dfs(int x)
39 {
40     sub[x] = 1;
41     white[x] = 0;
42     black[x][0] = 0;
43
44     for (int i = 0; i <= n; i++)
45         new_black[i] = -1;
46
47     for (auto y : v[x])
48         if (!sub[y])
49         {
50             Dfs(y);
51
52             for (int i = 0; i < sub[x]; i++)
53             {
54                 for (int j = 0; j < sub[y]; j++)
55                     if (black[x][i] != -1 && black[y][j] != -1)
56                         new_black[i + j] = max(new_black[i + j],
57                                     black[x][i] + black[y][j] + j * i);
58
59                 if (black[x][i] != -1)
60                     new_black[i + 1] = max(new_black[i + 1], black[x][i] + white[y] + i);
61             }
62
63             int best_from_y = white[y] + 1;
64             for (int i = 0; i <= sub[y]; i++)
65                 if (black[y][i] != -1)
66                     best_from_y = max(best_from_y, black[y][i] + i);
67
68             white[x] += best_from_y;
69
70             sub[x] += sub[y];
71             for (int i = 0; i <= sub[x]; i++)
72             {
73                 black[x][i] = new_black[i];
74                 new_black[i] = -1;
75             }
76         }
77     }
78
79 int main()
80 {
81     cin.sync_with_stdio(false);
82
83     ifstream cin("tricolor.in");
84     ofstream cout("tricolor.out");
85
86     int t;
87     cin >> t;
88     for (; t; t--)
89     {
90         cin >> n;
91         Reset();
92
93         for (int i = 1; i < n; i++)
94         {
95             int x, y;
96             cin >> x >> y;
97             v[x].pb(y);
98             v[y].pb(x);
99         }
100 }
```

```

101     Dfs(1);
102
103     int ans = white[1];
104     for (int i = 0; i <= n; i++)
105         ans = max(ans, black[1][i]);
106
107     cout << ans << '\n';
108 }
109
110 return 0;
111 }
```

Listing 20.3.3: mapa_2_n.cpp

```

1 // O(2^n * n)
2 #include <fstream>
3 #include <iostream>
4 #include <vector>
5
6 using namespace std;
7 using Matrix = vector<vector<int>>;
8
9 int sol, current_config;
10
11 void dfs(int node, Matrix& graph, vector<int>& dp, vector<bool>& used)
12 {
13     int sum = 0;
14     bool color;
15     if (current_config &(1 << node))
16         color = true;
17     else color = false;
18
19     used[node] = true;
20     if (color == false) // white
21         dp[node] = 0;
22     else
23         dp[node] = 1; // black
24
25     for (auto other : graph[node])
26         if (!used[other])
27         {
28             dfs(other, graph, dp, used);
29             if (color == false)
30             {
31                 sum += dp[other];
32                 sol -= dp[other] * dp[other];
33                 dp[node] += dp[other];
34             }
35             else
36                 sol += 2 * dp[other];
37         }
38
39     if (color == false)
40     {
41         sum = sum * sum;
42         sol += sum;
43     }
44 // cerr << node + 1 << " " << dp[node] << "\n";
45 }
46
47 int solve(int config, int n, Matrix& graph)
48 {
49     sol = 0;
50     vector<int> dp(n);
51     vector<bool> used(n);
52
53     current_config = config;
54     dfs(0, graph, dp, used);
55     return sol;
56 }
57
58 int main()
59 {
60     int t, n, a, b;
61     ifstream cin("tricolor.in");
```

```

62     ofstream cout("tricolor.out");
63     cin >> t;
64     while(t--)
65     {
66         cin >> n;
67         Matrix graph(n, vector<int>());
68         for (int i = 0; i < n - 1; ++i)
69         {
70             cin >> a >> b;
71             --a;
72             --b;
73             graph[a].push_back(b);
74             graph[b].push_back(a);
75         }
76
77         int solutions = (1 << n), ans = 0, cnt = 0;
78         for (int config = 0; config < solutions; ++config)
79         {
80             a = solve(config, n, graph);
81             if (a > ans)
82                 ans = a;
83         }
84
85         cout << ans / 2 << "\n";
86     }
87
88     return 0;
89 }

```

Listing 20.3.4: n2_tamio_100.cpp

```

1 #include <iostream>
2 #include <cassert>
3 #include <vector>
4 #include <fstream>
5
6 using namespace std;
7
8 constexpr int maxn = 5e3 + 100, tmp_poz = maxn-1;
9
10 ifstream f("tricolor.in");
11 ofstream g("tricolor.out");
12
13 vector<int> vec[maxn] = {};
14 int n, best[maxn][maxn] = {}, nr[maxn] = {};
15
16 // best[i][j] = # max de inchise pentru subarb i si j deschise
17
18 void join_states(int t, int s){
19     for(int i = 0; i <= nr[s] + nr[t]; ++i)
20         best[tmp_poz][i] = -1e9;
21
22     for(int i = 0; i <= nr[t]; ++i)
23         for(int j = 0; j <= nr[s]; ++j)
24             best[tmp_poz][i+j] = max(best[tmp_poz][i+j],
25                                     best[t][i] + best[s][j] + i * j);
26
27     nr[t] += nr[s];
28     for(int i = 0; i <= nr[t]; ++i)
29     {
30         best[t][i] = best[tmp_poz][i];
31         best[tmp_poz][i] = 0;
32     }
33 }
34
35
36 void dfs(int cur, int prev = -1)
37 {
38     nr[cur] = 0;
39     int for_1 = 0;
40
41     for(auto next : vec[cur])
42         if(next != prev)
43         {

```

```

44         dfs(next, cur);
45
46         int here = 0;
47         for(int i = 1; i <= nr[next]; ++i)
48             here = max(here, best[next][i] + i);
49
50         for_1 += here;
51         join_states(cur, next);
52     }
53
54     best[cur][1] = max(best[cur][1], for_1);
55     if(nr[cur] == 0)
56     {
57         nr[cur] = 1;
58         best[cur][1] = 0;
59     }
60 }
61
62 void do_test()
63 {
64     f >> n;
65     assert(1 <= n && n <= 5000);
66
67     for(int i = 0, x, y; i < n-1; ++i)
68     {
69         f >> x >> y;
70         assert(1 <= x && x <= n);
71         assert(1 <= y && y <= n);
72         vec[x].push_back(y);
73         vec[y].push_back(x);
74     }
75
76     dfs(1, -1);
77
78     int best_val = 0;
79
80     for(int i = 1; i <= nr[1]; ++i)
81         if(best[1][i] > best_val)
82             best_val = best[1][i];
83
84     for(int i = 1; i <= n; ++i)
85         vec[i].clear();
86
87     g << best_val << endl;
88 }
89
90 int main()
91 {
92     int t;
93     f >> t;
94     assert(1 <= t && t <= 10);
95     while(t--) do_test();
96     return 0;
97 }
```

20.3.3 *Rezolvare detaliată

20.4 antivirus

Problema 4 - antivirus

100 de puncte

Se consideră un sir de N numere naturale. O parte dintre pozițiile sirului sunt nevirusate și acest lucru este marcat prin faptul că valoarea de la acele poziții este 0. Restul pozițiilor sunt virusate și valoarea nenulă de la o poziție virusată reprezintă costul cu care ea poate fi devirusată. Devirusăm o parte dintre poziții și dorim ca în final să avem exact K poziții nevirusate, iar costul total al devirusării să fie minim. O poziție poate fi devirusată la un moment dat, dacă și numai dacă are cel puțin o poziție vecină nevirusată. După devirusarea unei poziții costul asociat acesteia se adună la costul total, poziția devine nevirusată și orice altă poziție vecină virusată va putea fi ulterior devirusată.

Cerințe

Cunoscând N , K și sirul de numere naturale să se determine costul minim cu care se pot obține la final exact K poziții nevirusate (inclusiv cele care erau inițial nevirusate).

Date de intrare

Fișierul de intrare **antivirus.in** va conține pe prima linie numărul natural T - numărul de teste. Fiecare test este descris pe două linii. Pe prima linie se găsesc două numere naturale N și K cu semnificația din enunț și pe a doua linie N numere naturale reprezentând elementele sirului.

Date de ieșire

Fișierul de ieșire **antivirus.out** trebuie să conțină T linii. Pe fiecare linie va fi afișat răspunsul pentru un test - un număr natural reprezentând costul minim al unei devirusări astfel încât în final sirul să contină exact K poziții nevirusate - inclusiv cele care erau inițial nevirusate.

Restricții și precizări

- $1 \leq T \leq 4$
- $1 \leq K \leq N \leq 2\,000$
- pentru teste în valoare de 10 puncte se garantează că $N \leq 80$
- pentru alte teste în valoare de 20 puncte se garantează că $N \leq 200$
- pentru alte teste în valoare de 10 puncte în sirul inițial există exact o poziție nevirusată
- pentru alte teste în valoare de 10 puncte în sirul inițial există exact 2 poziții nevirusate

Exemple:

antivirus.in	antivirus.out	Explicații
2	10	Sunt $T = 2$ teste.
8 5	2	Pentru primul test se iau elementele de la indicii 2, 3, 4, 6 și se devirusează. Pentru al doilea test se iau elementele de la indicii 5 și 6 și se devirusează. O alta soluție validă ar fi fost indicii 1 și 5.
9 1 4 4 0 1 3		
9		
6 4		
1 0 2 0 1 1		

Timp maxim de executare/test: **1.3** secunde

Memorie: total **128 MB** din care pentru stivă **32 MB**

Dimensiune maximă a sursei: **20 KB**

20.4.1 Indicații de rezolvare

Autori: Răzvan Sălăjan, Alex Cociorva - Universitatea "Babeș-Bolyai" Cluj-Napoca
index[person]Răzvan Sălăjan

Soluție $O(N^3)$

Putem să ne gândim la următoarea dinamică:

$$dp[i][j] = \text{costul minim să devirusăm } j \text{ elemente considerând doar primele } i$$

Când suntem la pasul i , putem să considerăm o secvență $[k..i]$ cu $k \leq i$ și să devirusăm toate elementele din acea secvență. Evident, putem face asta, doar dacă avem un element nevirusat inițial în acea secvență.

$$dp[i][j] = \min(dp[i-1][j], dp[k-1][j - \text{num_elemente_virusate}] + \text{cost_elemente_virusate});$$

Soluție $O(N^2)$

Folosim aceeași dinamică de la soluția anterioară, numai că modificăm modul de construcție al recurenței. Vom considera și *sumele parțiale* ale secvenței, notate prin vectorul $sum[]$.

Presupunem că suntem la un element i .

1. Putem să considerăm soluția de la pasul anterior și să nu facem nimic cu elementul i .

$$\bullet dp[i][j] = dp[i-1][j];$$

2. Dacă i este un element nevirusat, atunci putem să devirusăm elemente din stânga lui.

Vom considera toți indicii $k < i$ și vom devirusa toată secvența $[k..i-1]$.

- $dp[i][j] = \min(dp[i][j], dp[k][j - (i - k - 1)] + sum[i] - sum[k]);$

Observăm că nu are rost să mergem cu k -ul decât până la primul element nevirusat inițial din stânga lui i . Să notăm această poziție cu $last$. Toate celelalte elemente ar putea fi devirusate de $last$.

3. Dacă i este un element virusat, atunci putem să îl devirusăm cu un element nevirusat inițial din stânga lui, mai exact cel mai din dreapta element nevirusat inițial $< i$. Să notăm această poziție cu $last$. Practic vom devirusa toate elementele de la $last + 1$ la i .

- $dp[i][j] = \min(dp[i][j], dp[last][j - (i - last)] + sum[i] - sum[last]);$

Observăm că e de ajuns să considerăm doar devirusarea cu un element nevirusat inițial din stânga lui i . Cazul în care elementul i va fi devirusat de un element din dreapta lui este acoperit de punctul anterior.

Fiecare element va fi parcurs de maxim 2 ori pentru fiecare j posibil, complexitatea amortizându-se la $O(N^2)$.

20.4.2 Cod sursă

Listing 20.4.1: alex_50.cpp

```

1 #include <stdio.h>
2 #include <bits/stdc++.h>
3
4 using namespace std;
5
6 #define ll long long
7 #define ld long double
8 #define pb push_back
9 #define mp make_pair
10 #define pii pair<int, int>
11 #define pll pair<ll, ll>
12 #define pdd pair<ld, ld>
13 #define all(x) (x).begin(), (x).end()
14 #define fi first
15 #define se second
16
17 const int kMaxN = 2000;
18 const int kInf = (1LL << 31) - 1;
19 const int kMaxVal = 1e6;
20
21 int v[kMaxN + 5];
22 int dp[kMaxN + 5][kMaxN + 5];
23
24 int main()
25 {
26     ifstream cin("antivirus.in");
27     ofstream cout("antivirus.out");
28
29     int t;
30     cin >> t;
31
32     for (; t; t--)
33     {
34         int n, m;
35         cin >> n >> m;
36         assert(n <= kMaxN);
37
38         for (int i = 1; i <= n; i++)
39         {
40             cin >> v[i];
41             assert(v[i] <= kMaxVal);
42             m -= (v[i] == 0);
43         }
44
45         for (int i = 0; i < kMaxN; i++)
46             for (int j = 0; j < kMaxN; j++)
47                 dp[i][j] = kInf;
48
49         for (int i = 0; i <= n; i++)
50             dp[i][0] = 0;
51
52         for (int i = 1; i <= n; i++)

```

```

53     {
54         for (int j = m; j >= 1; j--)
55             dp[i][j] = dp[i - 1][j];
56
57         int speciale = 0;
58         bool have_special = 0;
59         int cost = 0;
60
61         for (int k = i; k >= 1; k--)
62         {
63             speciale += (v[k] != 0);
64             have_special |= (v[k] == 0);
65             cost += v[k];
66             if (!have_special)
67                 continue;
68
69             for (int j = m; j >= 1 && j >= speciale; j--)
70                 if (dp[k - 1][j - speciale] != kInf)
71                     dp[i][j] = min(dp[i][j], dp[k - 1][j - speciale] + cost);
72         }
73     }
74
75     int ans = kInf;
76     for (int i = 1; i <= n; i++)
77         ans = min(ans, dp[i][m]);
78
79     cout << ans << '\n';
80 }
81
82 return 0;
83 }
```

Listing 20.4.2: alex_100.cpp

```

1 #include <stdio.h>
2 #include <bits/stdc++.h>
3
4 using namespace std;
5
6 #define ll long long
7 #define ld long double
8 #define pb push_back
9 #define mp make_pair
10 #define pii pair<int, int>
11 #define pll pair<ll, ll>
12 #define pdd pair<ld, ld>
13 #define all(x) (x).begin(), (x).end()
14 #define fi first
15 #define se second
16
17 const int kMaxN = 2000;
18 const int kMaxVal = 1e6;
19 const int kInf = (1LL << 31) - 1;
20
21 int v[kMaxN + 5];
22 int sum[kMaxN + 5];
23 int dp[kMaxN + 5][kMaxN + 5];
24
25 int main()
26 {
27     cin.sync_with_stdio(false);
28
29     ifstream cin("antivirus.in");
30     ofstream cout("antivirus.out");
31
32     int t;
33     cin >> t;
34
35     for (; t; t--)
36     {
37         int n, m;
38         cin >> n >> m;
39         assert(n >= 1 && n <= kMaxN);
40         assert(m >= 1 && m <= n);
41     }
}
```

```

42     for (int i = 1; i <= n; i++)
43     {
44         cin >> v[i];
45         assert(v[i] >= 0 && v[i] <= kMaxVal);
46         if (v[i] == 0)
47             m--;
48
49         sum[i] = sum[i - 1] + v[i];
50     }
51
52     for (int i = 0; i < kMaxN; i++)
53         for (int j = 0; j < kMaxN; j++)
54             dp[i][j] = kInf;
55
56     for (int i = 0; i <= n; i++)
57         dp[i][0] = 0;
58
59     int last = 0;
60     for (int i = 1; i <= n; i++)
61     {
62         if (v[i] > 0)
63         {
64             if (last > 0)
65                 for (int j = m; j > 0; j--)
66                 {
67                     dp[i][j] = dp[i - 1][j];
68                     if (j - (i - last) >= 0 && dp[last][j - (i - last)] != kInf)
69                         dp[i][j] = min(dp[i][j], dp[last][j - (i - last)] + sum[i] - sum[last]);
70                 }
71         }
72         else
73         {
74             for (int j = 1; j <= m; j++)
75             {
76                 dp[i][j] = dp[i - 1][j];
77                 for (int k = i - 1; k >= last; k--)
78                     if (j - (i - k - 1) >= 0 && dp[k][j - (i - k - 1)] != kInf)
79                         dp[i][j] = min(dp[i][j], dp[k][j - (i - k - 1)] + sum[i] - sum[k]);
80             }
81         }
82         last = i;
83     }
84 }
85
86 ll ans = dp[n][m];
87 cout << ans << '\n';
88 assert(ans != kInf);
89 }
90
91 return 0;
92 }
```

Listing 20.4.3: costel_n2_100.cpp

```

1 // Andrei Costin Constantinescu - Oxford University
2 // O(N^2)
3 #include <bits/stdc++.h>
4
5 using namespace std;
6
7 const int NMAX = 2000;
8 const int VALMAX = 1000000;
9
10 int N, K;
11 vector <int> v;
12 vector <vector <vector <int> >> dp;
13
14 inline void updVal(int &where, const int val)
15 {
16     if (where == -1 || val < where)
17         where = val;
18 }
19
20 int main()
21 {
```

```

22     freopen("antivirus.in", "r", stdin);
23     freopen("antivirus.out", "w", stdout);
24
25     int T;
26     cin >> T;
27
28     for ( ; T; T--)
29     {
30         v.clear();
31         dp.clear();
32
33         cin >> N >> K;
34         assert(1 <= N && N <= NMAX);
35
36         // Read
37         v.push_back(0);
38         int zeroes = 0;
39         for (int i = 1; i <= N; ++ i)
40         {
41             int val;
42             cin >> val;
43             zeroes += (val == 0);
44             assert(0 <= val && val <= VALMAX);
45             v.push_back(val);
46         }
47         assert(zeroes <= K && K <= N);
48
49         dp.push_back(vector <vector <int> >(K + 1, vector <int>(3, -1)));
50         dp[0][0][0] = 0;
51
52         for (int i = 0; i < N; ++ i)
53         {
54             dp.push_back(vector <vector <int> >(K + 1, vector <int>(3, -1)));
55             for (int j = 0; j <= K; ++ j)
56             {
57                 if (v[i + 1] == 0)
58                 {
59                     if (dp[i][j][0] != -1 && j + 1 <= K)
60                         updVal(dp[i + 1][j + 1][2], dp[i][j][0]);
61                     if (dp[i][j][1] != -1 && j + 1 <= K)
62                         updVal(dp[i + 1][j + 1][2], dp[i][j][1]);
63                     if (dp[i][j][2] != -1 && j + 1 <= K)
64                         updVal(dp[i + 1][j + 1][2], dp[i][j][2]);
65                 }
66                 else
67                 {
68                     if (dp[i][j][0] != -1)
69                     {
70                         updVal(dp[i + 1][j][0], dp[i][j][0]);
71                         if (j + 1 <= K)
72                             updVal(dp[i + 1][j + 1][1], dp[i][j][0] + v[i + 1]);
73                     }
74
75                     if (dp[i][j][1] != -1 && j + 1 <= K)
76                         updVal(dp[i + 1][j + 1][1], dp[i][j][1] + v[i + 1]);
77
78                     if (dp[i][j][2] != -1)
79                     {
80                         if (j + 1 <= K)
81                             updVal(dp[i + 1][j + 1][2], dp[i][j][2] + v[i + 1]);
82                         updVal(dp[i + 1][j][0], dp[i][j][2]);
83                     }
84                 }
85             }
86         }
87
88         int ans = -1;
89         for (int i = 0; i < 3; ++ i)
90             if (i != 1 && dp[N][K][i] != -1)
91                 updVal(ans, dp[N][K][i]);
92
93         cout << ans << '\n';
94     }
95
96     return 0;
97 }
```

Listing 20.4.4: costel_n3_50.cpp

```

1 // Andrei Costin Constantinescu - Oxford University
2 // O(N^3)
3 #include <bits/stdc++.h>
4
5 using namespace std;
6
7 const int NMAX = 2000;
8 const int VALMAX = 1000000;
9
10 int N, K;
11 vector <int> v;
12 vector <int> sPart;
13
14 vector <vector <int> > dp;
15
16 inline int getSum(int l, int r)
17 {
18     assert(l > 0), assert(l <= r);
19     assert(sPart[r] - sPart[l - 1] >= 0);
20     return sPart[r] - sPart[l - 1];
21 }
22
23 inline void updVal(int &where, const int val)
24 {
25     if (where == -1 || val < where)
26         where = val;
27 }
28
29 int main()
30 {
31     freopen("antivirus.in", "r", stdin);
32     freopen("antivirus.out", "w", stdout);
33
34     int T;
35     cin >> T;
36
37     for (; T; T--)
38     {
39         v.clear();
40         sPart.clear();
41         dp.clear();
42
43         cin >> N >> K;
44         assert(1 <= N && N <= NMAX);
45
46         // Read
47         v.push_back(0), sPart.push_back(0);
48         int zeroes = 0;
49         for (int i = 1; i <= N; ++ i)
50         {
51             int val;
52             cin >> val;
53             zeroes += (val == 0);
54             assert(0 <= val && val <= VALMAX);
55             v.push_back(val), sPart.push_back(sPart.back() + val);
56         }
57
58         assert(zeroes <= K && K <= N);
59
60         // Add sentinel zero
61         ++ N, ++ K;
62         v.push_back(0);
63         ++ zeroes;
64         sPart.push_back(sPart.back());
65
66         vector <int> zrs = {0};
67         for (int i = 1; i <= N; ++ i)
68             if (!v[i])
69                 zrs.push_back(i);
70
71         dp.push_back(vector <int>(K + 1, -1)),
72         dp.push_back(vector <int>(K + 1, -1));
73
74         const int fstPs = zrs[1];

```

```

75
76     for (int i = fstPs; i >= 1; -- i)
77         if (fstPs - i + 1 <= K)
78             dp[1][fstPs - i + 1] = getSum(i, fstPs);
79
80     for (int i = 2; i < zrs.size(); ++ i)
81     {
82         dp.push_back(vector <int>(K + 1, -1));
83         const int a = zrs[i - 1], b = zrs[i];
84         for (int knap = 0; knap <= K; ++ knap)
85         {
86             for (int j = a; j <= b; ++ j)
87                 for (int k = j + 1; k <= b; ++ k)
88                 {
89                     if (i + 1 == zrs.size() && k < b)
90                         continue;
91                     if (knap - (b - k + 1 + j - a) >= 0 &&
92                         dp[i - 1][knap - (b - k + 1 + j - a)] != -1)
93                         updVal(dp[i][knap],
94                                dp[i - 1][knap - (b - k + 1 + j - a)] +
95                                getSum(a, j) + getSum(k, b));
96                 }
97             }
98         }
99
100        cout << dp.back()[K] << '\n';
101    }
102
103    return 0;
104 }
```

Listing 20.4.5: tamio_n2_100.cpp

```

1 #include <vector>
2 #include <iostream>
3 #include <fstream>
4 #include <numeric>
5 #include <algorithm>
6
7 using namespace std;
8
9 constexpr int maxn = 2000 + 100;
10 int v[maxn], sp[maxn], l[maxn], r[maxn],
11      vals[maxn][maxn], d[maxn][maxn], nr_intervals = 0;
12
13 int get_sum(int x, int y)
14 {
15     return sp[y] - sp[x-1];
16 }
17
18 int main()
19 {
20     int t;
21     cin >> t;
22
23     for(; t; t--)
24     {
25         nr_intervals = 0;
26
27         for(auto& x : vals)
28             for(auto& y : x)
29                 y = 1e9;
30
31         for(auto& x : d)
32             for(auto& y : x)
33                 y = 1e9;
34
35         ifstream f("antivirus.in");
36         ofstream g("antivirus.out");
37
38         int n, k;
39         f >> n >> k;
40         for(int i = 1; i <= n; ++i) f >> v[i];
41
42         partial_sum(v+1, v+n+1, sp+1);
```

```

43
44     for(int i = 1; i <= n; ++i)
45     {
46         if(v[i] != 0 && (i == 1 || v[i-1] == 0))
47             l[nr_intervals] = i;
48         if(v[i] != 0 && (i == n || v[i+1] == 0))
49             r[nr_intervals++] = i;
50     }
51
52     for(int i = 0; i < nr_intervals; ++i)
53         cout << l[i] << ' ' << r[i] << endl;;
54
55     for(int i = 0; i < nr_intervals; ++i)
56     {
57         vals[i][r[i] - l[i] + 1] = get_sum(l[i], r[i]);
58         for(int j = l[i]; j <= r[i]; ++j)
59             for(int k = j; k <= r[i]; ++k)
60             {
61                 if(l[i] == 1 && j != 1) continue;
62                 if(r[i] == n && k != n) continue;
63                 vals[i][r[i] - k + j - l[i]] = min(
64                     vals[i][r[i] - l[i] + j - k],
65                     get_sum(l[i], r[i]) - get_sum(j, k));
66             }
67     }
68
69     d[0][0] = 0;
70
71
72     for(int i = 0; i < nr_intervals; ++i)
73         for(int j = 0; j <= n; ++j)
74             for(int val = 0; j + val <= n &&
75                  /* rem this for O(n^3) */ val <= r[i] - l[i] + 1; ++val)
76                 d[i+1][j+val] = min(d[i+1][j+val], d[i][j] + vals[i][val]);
77
78     g << d[nr_intervals][k - count(v+1, v+n+1, 0)] << endl;
79 }
80
81     return 0;
82 }
```

20.4.3 *Rezolvare detaliată

20.5 dungeon

Problema 5 - dungeon

100 de puncte

Fie G un graf neorientat cu $2 * N$ noduri și $3 * N - 2$ muchii. Fiecare muchie este colorată în alb, negru sau roșu.

Se garantează următoarele:

- Există $N - 1$ muchii albe. Capetele lor sunt noduri din mulțimea $1, 2, \dots, N$. Ele formează un arbore.
- Există $N - 1$ muchii negre. Capetele lor sunt noduri din mulțimea $N + 1, N + 2, \dots, 2 * N$. Ele formează un arbore.
- Există N muchii roșii. Fiecare muchie are un capăt în mulțimea $1, 2, \dots, N$ și celălalt capăt în mulțimea $N + 1, N + 2, \dots, 2 * N$.

Cele $2 * N$ capete ale muchiilor roșii sunt distințe două căte două. Cu alte cuvinte, fiecare nod din graf are exact o muchie roșie incidentă.

Numim *ciclu hamiltonian special* un ciclu care:

- vizitează fiecare nod al grafului exact o dată.
- nu parurge consecutiv două muchii de aceeași culoare.
- începe din nodul 1, iar prima muchie parcursă este de culoare roșie.

Cerințe

Afişaţi un ciclu hamiltonian special al grafului G sau constataţi că nu există niciun astfel de ciclu.

Date de intrare

Fişierul de intrare **dungeon.in** va conţine pe prima linie un număr natural T reprezentând numărul de teste. Pentru fiecare test pe prima linie se află valoarea N . Pe următoarele $N - 1$ linii se găsesc perechi de valori reprezentând capetele muchiilor de culoare albă (valori de la 1 la N). Următoarele $N - 1$ linii conţin perechi de valori ce reprezintă capetele muchiilor de culoare neagră (valori de la $N + 1$ la $2 * N$). Următoarele N perechi de valori reprezintă capetele muchiilor de culoare roşie.

Date de ieşire

Fişierul de ieşire **dungeon.out** va conţine pentru fiecare din cele T teste câte o linie cu $2 * N$ valori reprezentând succesiunea nodurilor care formează ciclul hamiltonian special al fiecărui graf dat, respectiv valoarea -1 dacă nu există un astfel de ciclu.

Restricţii şi precizări

- $N \leq 50\,000$
- $T \leq 5$
- Pentru teste în valoare de 20 puncte se garantează că $N \leq 10$
- Pentru alte teste în valoare de 30 puncte se garantează că ambii arbori au forma de *lanț*.

Exemple:

dungeon.in	dungeon.out	Explicaţii
2	-1	Există două teste.
4	1 7 6 4 3 5 8	În fiecare test graful are $2 * 4$ noduri și $3 * 4 - 2$ muchii (3 muchii de culoare albă, 3 muchii de culoare neagră, respectiv 4 muchii de culoare roșie).
1 2	2	În primul graf nu se găsește un ciclu hamiltonian special.
1 3		
3 4		
5 6		
5 7		
5 8		
1 5		
2 6		
3 7		
4 8		
4		
1 2		
1 3		
3 4		
5 6		
6 7		
5 8		
1 7		
2 8		
3 5		
4 6		

Timp maxim de executare/test: **2.5** secunde

Memorie: total **128 MB** din care pentru stivă **32 MB**

Dimensiune maximă a sursei: **20 KB**

20.5.1 Indicaţii de rezolvare

Se observă că pentru N impar nu există soluție.

Să presupunem următorul algoritm de determinare a unui astfel de ciclu:

- Determinăm un *cuplaj* în primul arbore: o împărțire a arborelui în $N/2$ perechi de noduri astfel încât pentru fiecare pereche (A, B) să existe muchia de la A la B în arbore.
- Determinăm un cuplaj în cel de al doilea arbore.
- Considerăm graful format din cele N muchii roșii, cele $N/2$ muchii albe determinate de cele $N/2$ perechi din primul cuplaj, respectiv cele $N/2$ muchii negre determinate de cele $N/2$ perechi din cel de-al doilea cuplaj.
- Verificăm dacă acest graf chiar este soluție sau nu (dacă formează ciclu sau nu).

Observația 1:

Cum determinăm un cuplaj într-un arbore? Pornim o *parcursere DFS* din rădăcină. Pentru fiecare nod X din DFS avem 3 cazuri:

- Toți fi și acestuia sunt cuplați \Rightarrow transmitem în părinte că X este necuplat
- Un singur fiu e necuplat \Rightarrow îl cuplăm pe acesta cu X și transmitem că X este cuplat
- Mai mulți fi sunt necuplați \Rightarrow nu există soluție

Algoritmul încearcă să cupleze fiecare *frunză* cu părintele acesteia. Astfel, se poate deduce că dacă există soluție, aceasta este unică.

Observația 2:

De ce soluția este unică la această problemă (aceasta fiind ciclul definit mai sus)?

Se observă faptul că fiecare muchie roșie trebuie folosită, iar pentru fiecare nod X din cele $2 * N$, acesta va fi conectat în ciclu prin exact o muchie de arbore (albă sau neagră). Această muchie poate fi să pornească din X într-un vecin Y din același arbore (și din Y merge pe o muchie roșie), fie să vină dintr-un vecin Y din același arbore și din X pornește în celălalt arbore.

Astfel, deducem pentru fiecare arbore că nodurile trebuie împărțite în $N/2$ astfel de perechi, indiferent de sensul în care sunt parcurse. Această împărțire este dată de cele 2 cuplaje menționate mai sus. Din moment ce aceste cuplaje au soluție unică rezultă că și ciclul este unic.

Observația 3:

Deși am determinat muchiile grafului, nu este garantat că acesta formează un ciclu (conținează muchiile roșii). Este posibil ca graful să fie împărțit în mai mulți ciclii. Ca urmare, graful obținut este doar o posibilă soluție ce trebuie verificată la final.

Complexitate: $O(N)$ pentru fiecare test

20.5.2 Cod sursă

Listing 20.5.1: adrian_100.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <algorithm>
5 #include <cassert>
6 #include <queue>
7
8 using namespace std;
9
10 void dfs(int node, const vector<vector<int>> &edges, vector<bool>& used)
11 {
12     if (used[node])
13         return;
14     used[node] = true;
15     for (auto &x : edges[node])
16         dfs(x, edges, used);
17 }
18
19 int main()
20 {
21     ifstream cin("dungeon.in");
22     ofstream cout("dungeon.out");
23
24     int T; cin >> T;
25     assert(1 <= T && T <= 5);
26     while (T--)

```

```

27
28     {
29         int N; assert(cin >> N);
30         assert(1 <= N && N <= 50000);
31         vector<vector<int>> edges(2 * N);
32         for (int i = 0; i < N - 1; ++i)
33         {
34             int x, y; assert(cin >> x >> y);
35             assert(1 <= x && x <= N);
36             assert(1 <= y && y <= N);
37             assert(x != y);
38             edges[x - 1].push_back(y - 1);
39             edges[y - 1].push_back(x - 1);
40         }
41
42         for (int i = 0; i < N - 1; ++i)
43         {
44             int x, y; assert(cin >> x >> y);
45             assert(N + 1 <= x && x <= N * 2);
46             assert(N + 1 <= y && y <= N * 2);
47             assert(x != y);
48             edges[x - 1].push_back(y - 1);
49             edges[y - 1].push_back(x - 1);
50         }
51
52         vector<bool> matched(2 * N, false);
53         vector<vector<int>> matching(2 * N);
54
55         for (int i = 0; i < N; ++i)
56         {
57             int x, y; assert(cin >> x >> y);
58             if (x > y)
59                 swap(x, y);
60             assert(1 <= x && x <= N);
61             assert(N + 1 <= y && y <= 2 * N);
62             --x; --y;
63             matched[x] = true;
64             matched[y] = true;
65             matching[x].push_back(y);
66             matching[y].push_back(x);
67         }
68
69         assert(matched == vector<bool>(2 * N, true));
70
71         vector<bool> used(2 * N, false);
72
73         dfs(0, edges, used);
74         dfs(N, edges, used);
75
76         assert(used == vector<bool>(2 * N, true));
77
78         queue<int> Q;
79         vector<int> degree(2 * N, 0);
80         for (int i = 0; i < 2 * N; ++i)
81         {
82             degree[i] = edges[i].size();
83             if (degree[i] == 1)
84                 Q.push(i);
85         }
86
87         while (!Q.empty())
88         {
89             int node = Q.front();
90             Q.pop();
91             for (auto &x : edges[node])
92             {
93                 if (degree[x] > 0)
94                 {
95                     matching[x].push_back(node);
96                     matching[node].push_back(x);
97                     degree[node] = 0;
98                     degree[x] = 0;
99                     for (auto &y : edges[x])
100                     {
101                         if (--degree[y] == 1)
102                             Q.push(y);
103                     }
104                 }
105             }
106         }
107     }
108 }
```

```

103         }
104     }
105 }
106
107     bool ok = true;
108     for (int i = 0; i < 2 * N; ++i)
109     {
110         if (matching[i].size() != 2)
111         {
112             ok = false;
113         }
114     if (!ok)
115     {
116         cout << "-1\n";
117         continue;
118     }
119
120     vector<int> answer;
121     int node = 0;
122     int last = -1;
123     do
124     {
125         answer.push_back(node);
126         for (int i = 0; i < 2; ++i)
127         {
128             if (matching[node][i] != last)
129             {
130                 int next = matching[node][i];
131                 last = node;
132                 node = next;
133                 break;
134             }
135         } while (node);
136     if (int(answer.size()) != 2 * N)
137     {
138         cout << "-1\n";
139         continue;
140     }
141
142     for (auto &x : answer)
143         cout << x + 1 << " ";
144     cout << "\n";
145 }
146 }
```

Listing 20.5.2: eudanip_100.cpp

```

1 #include<stdio.h>
2 #include<string.h>
3 #include<vector>
4 using namespace std;
5
6 #define NMAX 200005
7
8 vector<int> solution, tree[NMAX];
9 int mark[NMAX], cuplajTree[NMAX], cuplaj[NMAX], t, n;
10 char viz[NMAX], ok = 0;
11
12 void clearAll()
13 {
14     memset(viz, 0, sizeof(viz));
15     memset(mark, 0, sizeof(mark));
16     memset(cuplaj, 0, sizeof(cuplaj));
17     memset(cuplajTree, 0, sizeof(cuplajTree));
18     for(int i = 1; i <= 2 * n; i++)
19         tree[i].clear();
20     solution.clear();
21 }
22
23 inline void dfs(int node)
24 {
25     viz[node] = 1;
26
27     int lim = tree[node].size(), neigh;
28     for(int i = 0; i < lim; i++ && !ok)
```

```

29     {
30         if(viz[neigh = tree[node][i]])
31             continue;
32         dfs(neigh);
33         if(cuplajTree[neigh])
34             continue;
35         if(cuplajTree[node])
36         {
37             ok = 1;
38             return ;
39         }
40         cuplajTree[node] = neigh;
41         cuplajTree[neigh] = node;
42     }
43 }
44
45 inline void getSolution(int node)
46 {
47     for(int timp = 1 ; timp <= 2 * n + 1; timp++)
48     {
49         mark[node]++;
50         if(mark[node] > 1)
51             return ;
52         solution.push_back(node);
53
54         if(timp&1)
55             node = cuplaj[node];
56         else
57             node = cuplajTree[node];
58     }
59 }
60
61 int main ()
62 {
63     int a, b;
64
65     freopen("dungeon.in", "r", stdin);
66     freopen("dungeon.out", "w", stdout);
67
68     scanf("%d", &t);
69     for( ; t; t--)
70     {
71         scanf("%d", &n);
72         for(int i = 1; i < n; i++)
73         {
74             scanf("%d%d", &a, &b);
75             tree[a].push_back(b);
76             tree[b].push_back(a);
77         }
78
79         for(int i = 1; i < n; i++)
80         {
81             scanf("%d%d", &a, &b);
82             tree[a].push_back(b);
83             tree[b].push_back(a);
84         }
85
86         for(int i = 1; i <= n; i++)
87         {
88             scanf("%d%d", &a, &b);
89             cuplaj[a] = b;
90             cuplaj[b] = a;
91         }
92
93         if(n&1)
94         {
95             printf("-1\n");
96             clearAll();
97             continue;
98         }
99
100        ok = 0;
101        dfs(1);
102        dfs(n + 1);
103
104        if(!ok)

```

```

105         getSolution(1);
106
107         int nr = solution.size();
108
109         if(ok || nr != 2 * n || mark[1] != 2)
110             printf("-1\n");
111         else
112         {
113             for(int i = 0; i < 2 * n; i++)
114                 printf("%d ", solution[i]);
115             printf("\n");
116         }
117
118         clearAll();
119     }
120
121     return 0;
122 }
```

Listing 20.5.3: eudanip_back_20.cpp

```

1 #include<stdio.h>
2 #include<vector>
3 using namespace std;
4
5 #define NMAX 100005
6
7 int sol = 0, viz[NMAX], bc[NMAX], cuplaj[NMAX], n, t;
8 vector<int> tree[NMAX];
9
10 void clearAll()
11 {
12     for(int i = 1; i <= 2 * n; i++)
13         tree[i].clear();
14
15     sol = 0;
16 }
17
18 void backtracking(int poz)
19 {
20     if(poz == 2 * n)
21     {
22         int lim = tree[bc[poz]].size();
23         for(int i = 0; i < lim; i++)
24             if(tree[bc[poz]][i] == 1)
25             {
26                 sol = 1;
27                 break;
28             }
29         return ;
30     }
31
32     if(poz&1)
33     {
34         if(viz[coplaj[bc[poz]]])
35             return ;
36
37         viz[coplaj[bc[poz]]] = 1;
38         bc[poz + 1] = cuplaj[bc[poz]];
39         backtracking(poz + 1);
40         viz[coplaj[bc[poz]]] = 0;
41         return ;
42     }
43
44     int lim = tree[bc[poz]].size(), vec;
45     for(int i = 0; i < lim && !sol; i++)
46         if(!viz[vec = tree[bc[poz]][i]])
47         {
48             bc[poz + 1] = vec;
49             viz[vec] = 1;
50             backtracking(poz + 1);
51             viz[vec] = 0;
52         }
53     }
54 }
```

```

55 int main ()
56 {
57     int a, b;
58
59     freopen("dungeon.in", "r", stdin);
60     freopen("dungeon.out", "w", stdout);
61
62     scanf("%d");
63     for( ; t; t--)
64     {
65         scanf("%d", &n);
66         for(int i = 1; i <= 2 * (n - 1); i++)
67         {
68             scanf("%d%d", &a, &b);
69             tree[a].push_back(b);
70             tree[b].push_back(a);
71         }
72
73         for(int i = 1; i <= n; i++)
74         {
75             scanf("%d%d", &a, &b);
76             cuplaj[a] = b;
77             cuplaj[b] = a;
78         }
79
80         if(n & 1)
81         {
82             printf("-1\n");
83             clearAll();
84             continue;
85         }
86
87         bc[1] = 1;
88         viz[1] = 1;
89         backtracking(1);
90
91         if(!sol)
92         {
93             printf("-1\n");
94         }
95         else
96         {
97             for(int i = 1; i <= 2 * n; i++)
98                 printf("%d ", bc[i]);
99             printf("\n");
100        }
101
102        clearAll();
103    }
104
105    return 0;
106 }
```

Listing 20.5.4: mapa_lant_20.cpp

```

1 #include <iostream>
2 #include <assert>
3 #include <iostream>
4 #include <queue>
5 #include <vector>
6
7 using namespace std;
8 using Graph = vector<vector<int>>;
9
10 int n;
11
12 int find_root(vector<int>& degree, int l, int r)
13 {
14     for (int i = l; i < r; ++i)
15         if (degree[i] == 1)
16             return i;
17
18     return -1; // !!!
19 }
20
```

```

21 void dfs(int node, Graph& graph, vector<bool>& used, int col, Graph& newGraph)
22 {
23     used[node] = true;
24     for (auto other : graph[node])
25         if (!used[other])
26         {
27             if (col == 0)
28             {
29                 newGraph[node].push_back(other);
30                 newGraph[other].push_back(node);
31             }
32             dfs(other, graph, used, !col, newGraph);
33         }
34     }
35 }
36
37 bool check(int node, Graph& graph, vector<bool>& used,
38             vector<int>& ans, int last_color)
39 {
40     int color;
41     used[node] = true;
42     ans.push_back(node);
43     for (auto other : graph[node])
44     {
45         if (other < n && node < n)
46             color = 0;
47         else
48             if (other >= n && node >= n)
49                 color = 1;
50             else
51                 color = 2;
52
53         if (!used[other])
54         {
55             if (color == last_color)
56                 return false;
57
58             return check(other, graph, used, ans, color);
59         }
60     }
61
62     return true;
63 }
64
65 int main()
66 {
67     ifstream cin("12-dungeon.in");
68     ofstream cout("dungeon.out");
69
70     int t, a, b;
71     cin >> t;
72     while(t--)
73     {
74         cin >> n;
75         Graph graph(2 * n, vector<int>());
76         vector<int> degree(2 * n);
77
78         // Read white edges
79         for (int i = 0; i < n - 1; ++i)
80         {
81             cin >> a >> b;
82             --a;
83             --b;
84             graph[a].push_back(b);
85             graph[b].push_back(a);
86             ++degree[a];
87             ++degree[b];
88         }
89
90         // Read black edges
91         for (int i = 0; i < n - 1; ++i)
92         {
93             cin >> a >> b;
94             --a; --b;
95             graph[a].push_back(b);
96             graph[b].push_back(a);

```

```

97         ++degree[a];
98         ++degree[b];
99     }
100
101    if (n % 2)
102    {
103        cout << "-1";
104        continue;
105    }
106
107    vector<bool> used(2 * n, false);
108    int root_white = find_root(degree, 0, n);
109    int root_black = find_root(degree, n, 2 * n);
110    Graph hton(2 * n);
111
112    dfs(root_white, graph, used, 0, hton);
113    dfs(root_black, graph, used, 0, hton);
114
115    fill(used.begin(), used.end(), false);
116
117 // Read red edges
118 for (int i = 0; i < n; ++i)
119 {
120     cin >> a >> b;
121     --a; --b;
122     hton[a].push_back(b);
123     hton[b].push_back(a);
124 }
125
126 bool ok = true;
127 for (int i = 0; i < 2 * n && ok; ++i)
128 {
129     if (hton[i].size() != 2)
130     {
131         ok = false;
132     }
133
134     if (ok == false)
135     {
136         cout << "-1\n";
137         continue;
138     }
139
140     int other = -1;
141
142     if (hton[0][0] >= n)
143         other = hton[0][0];
144
145     if (hton[0][1] >= n)
146         other = hton[0][1];
147
148     if (other == -1)
149     {
150         cout << "-1\n";
151         continue;
152     }
153
154     used[0] = true;
155     vector<int> ans;
156     ans.push_back(0);
157     ok = check(other, hton, used, ans, 2);
158
159     if (ok == false || ans.size() != 2 * n)
160         cout << "-1\n";
161     else
162     {
163         for (auto x : ans)
164             cout << x + 1 << " ";
165         cout << "\n";
166     }
167
168     return 0;
169 }
```

20.5.3 *Rezolvare detaliată

20.6 zuma

Problema 6 - zuma

100 de puncte

Se dă un sir de caractere de lungime N format din litere mari ale alfabetului englez și un număr întreg K .

Asupra sirului se poate aplica în mod repetat următoarea operație: se alege o subsecvență de lungime cel puțin K având toate elementele egale și se elimină din sir. Evident că prima dată operația se aplică asupra sirului inițial și ulterior asupra sirului obținut din aplicarea operației anterioare. Operația se aplică până când sirul devine sirul vid (de lungime 0) sau sirul nu mai conține subsecvențe de lungime cel puțin K cu toate elemente egale.

Cerințe

Cunoscând N , K și sirul de caractere, să se determine care este lungimea minimă la care poate fi redus sirul după aplicarea operațiilor într-un mod convenabil.

Date de intrare

Fișierul de intrare **zuma.in** conține pe prima linie numerele N și K separate prin spațiu și pe a doua linie sirul de caractere format din N litere mari.

Date de ieșire

În fișierul de ieșire **zuma.out** trebuie să afișați un singur număr natural reprezentând lungimea minimă pe care o poate avea sirul după aplicarea în mod convenabil a operațiilor.

Restricții și precizări

- $1 \leq K \leq N \leq 500$
- Pentru teste în valoare de 10 puncte $25 \leq K \leq N \leq 100$
- Pentru alte teste în valoare de 10 puncte $K = 2$
- Pentru alte teste în valoare de 10 puncte numărul de caractere distințe din sir este 2
- Pentru alte teste în valoare de 15 puncte $1 \leq N \leq 50$
- Pentru alte teste în valoare de 35 puncte $1 \leq N \leq 200$

Exemplu:

zuma.in	zuma.out	Explicații
10 3 AAABBBCCCA	0	<p>Avem $N=10$ și $K=3$. Se poate elimina orice subsecvență cu litere identice de lungime cel puțin 3.</p> <p>Considerând sirul indexat de la 1, la prima operație putem elibera subsecvența $[4,6]=BBB$ pentru că are lungimea 3.</p> <p>Acum sirul este AAACCCA, deci putem elibera subsecvența $[4,6]=CCC$, obținând sirul AAAA.</p> <p>Acesta este format din 4 caractere egale, deci îl putem sterge și obținem sirul vid ce are lungimea 0.</p> <p>Dacă am fi ales la pasul anterior să eliberau subsecvența $[1,3]=AAA$ și apoi subsecvența CCC, am fi obținut un sir final de lungime 1, format din ultimul A.</p>

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

Memorie: total **128 MB** din care pentru stivă **32 MB**

Dimensiune maximă a sursei: **20 KB**

20.6.1 Indicații de rezolvare

Bogdan Ciobanu - Universitatea din București

Dacă am știi pentru o subsecvență dacă putem să o reducem la sirul vid sau nu, am putea construi următoarea recurență:

- $lg_min(i) = \text{lungimea minimă a sirului considerând primele } i \text{ caractere, după oricătre operații}$

Aceasta se poate calcula, spre exemplu, după următoarea recurență:

$$lg_min(i) = \min(lg_min(i-1) + 1, \min(lg_min(j-1) \mid [j, i] \text{ se poate comprima}))$$

Pentru a stabili dacă o subsecvență se poate comprima sau nu, avem mai multe abordări posibile.

În complexitate $O(SIGMA * N^3)$ putem construi următoarea recurență:

• $d(i, j, ch) = \text{numărul maxim de caractere egale cu } ch, \text{ dacă ar fi să reducem subsecvența } [i, j] \text{ la un sir format doar din elemente egale cu } ch; \text{ În cazul în care subsecvența nu se poate reduce doar la elemente egale cu } ch, \text{ atunci starea este invalidă.}$

- În primul rând, putem fixa un punct de spargere $i \leq p \leq j-1$, atunci

$$dp(i, j, ch) = \max(dp(i, p, ch) + dp(p+1, j, ch)).$$

Mai trebuie să tratăm cazul în care $[i, p]$ sau $[p+1, j]$ se reduce la sirul vid și considerăm doar partea rămasă.

Această soluție este punctată cu aproximativ 70 de puncte.

Pentru a reduce un factor de $SIGMA$ observăm că are sens să calculăm dinamica mai sus menționată doar pentru (i, j, ch) , unde ch este caracterul de pe poziția i . Această soluție se va încadra în 100 de puncte.

20.6.2 Cod sursă

Listing 20.6.1: bc_zuma_n3_100.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 const int kMaxN = 500, kInf = 0x3f3f3f3f;
6
7 char s[kMaxN + 1];
8 bool compress[kMaxN][kMaxN];
9 int mx_rem[kMaxN][kMaxN];
10 int mn_size[kMaxN];
11
12 int main()
13 {
14     ifstream cin("zuma.in");
15     ofstream cout("zuma.out");
16
17     int n, k;
18     cin >> n >> k >> s;
19     if (k == 1)
20     {
21         cout << 0 << endl;
22         return 0;
23     }
24
25     for (int l = n - 1; l >= 0; --l)
26     {
27         mx_rem[l][l] = 1;
28         for (int r = l + 1; r < n; ++r)
29         {
30             if (s[l] == s[r])
31                 mx_rem[l][r] = mx_rem[l][r - 1] + 1;
32             else
33                 mx_rem[l][r] = -kInf;
34
35             for (int split = l; split < r; ++split)
36                 if (compress[split + 1][r])

```

```

37             mx_rem[l][r] = max(mx_rem[l][r], mx_rem[l][split]);
38
39             compress[l][r] = (mx_rem[l][r] >= k);
40             for (int split = l; split < r; ++split)
41                 compress[l][r] |= compress[l][split] and
42                 compress[split + 1][r];
43             }
44         }
45
46         for (int i = 0; i < n; ++i)
47     {
48             mn_size[i] = 1 + (i > 0 ? mn_size[i - 1] : 0);
49             for (int j = 0; j < i; ++j)
50                 if (compress[j][i])
51                     mn_size[i] = min(mn_size[i], (j > 0 ? mn_size[j - 1] : 0));
52         }
53
54         cout << mn_size[n - 1] << endl;
55         return 0;
56     }

```

Listing 20.6.2: bc_zuma_n3sigma_70.cpp

```

1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4
5 using namespace std;
6
7 const int kMaxN = 500, kInf = (1 << 30);
8
9 int compress_size;
10 int dp[kMaxN][kMaxN][26];
11 int mn_length[kMaxN];
12
13 bool CanCompress(int li, int lf)
14 {
15     for (int i = 0; i < 26; ++i)
16         if (dp[li][lf][i] >= compress_size)
17             return true;
18
19     return false;
20 }
21
22 void UpdateMax(int& x, int y)
23 {
24     if (x < y)
25         x = y;
26 }
27
28 void ComputeTransitions(int li, int ki, int kf, int lf)
29 {
30     for (int ch = 0; ch < 26; ++ch)
31         if (dp[li][ki][ch] != -kInf and dp[kf][lf][ch] != -kInf)
32             UpdateMax(dp[li][lf][ch],
33                         dp[li][ki][ch] + dp[kf][lf][ch]);
34
35     if (CanCompress(li, ki))
36         for (int ch = 0; ch < 26; ++ch)
37             UpdateMax(dp[li][lf][ch], dp[kf][lf][ch]);
38
39     if (CanCompress(kf, lf))
40         for (int ch = 0; ch < 26; ++ch)
41             UpdateMax(dp[li][lf][ch], dp[li][ki][ch]);
42 }
43
44 int main()
45 {
46     ifstream cin("zuma.in");
47     ofstream cout("zuma.out");
48
49     int n;
50     string s;
51
52     cin >> n >> compress_size >> s;

```

```

53
54     for (int i = 0; i < n; ++i)
55         for (int j = i; j < n; ++j)
56             for (int ch = 0; ch < 26; ++ch)
57                 dp[i][j][ch] = -kInf;
58
59     for (int li = n - 1; li >= 0; --li)
60     {
61         dp[li][li][s[li] - 'A'] = 1;
62
63         for (int lf = li + 1; lf < n; ++lf)
64             for (int k = li; k < lf; ++k)
65                 ComputeTransitions(li, k, k + 1, lf);
66     }
67
68     for (int i = 0; i < n; ++i)
69     {
70         if (CanCompress(0, i))
71         {
72             mn_length[i] = 0;
73         }
74         else
75         {
76             mn_length[i] = 1 + (i > 0 ? mn_length[i - 1] : 0);
77             for (int j = i - 1; j >= 0; --j)
78                 if (mn_length[j] < mn_length[i] and CanCompress(j + 1, i))
79                     mn_length[i] = mn_length[j];
80         }
81     }
82
83     cout << mn_length[n - 1] << endl;
84 }
```

Listing 20.6.3: eric_back.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int NITER = 1000;
6
7 using it_t = string::iterator;
8
9 int random_int(int low, int high)
10 {
11     uniform_int_distribution<> d(low, high);
12     static default_random_engine gen;
13     return d(gen);
14 }
15
16 template<typename T>
17 T random_choice(const vector<T>& v)
18 {
19     assert (!v.empty());
20
21     return v[random_int(0, v.size() - 1)];
22 }
23
24 struct Solver
25 {
26     int n, k;
27
28     Solver(int n, int k):
29         n(n),
30         k(k)
31     { }
32
33     vector<pair<it_t, it_t>> splits(string& s) const
34     {
35         vector<pair<it_t, it_t>> candids;
36
37         for (auto i = s.begin(); i != s.end();)
38         {
39             auto j = i;
40             j++;
```

```

41             while (j != s.end() && *i == *j)
42                 j++;
43
44             if (j - i >= k)
45                 candids.push_back({i, j});
46
47             i = j;
48         }
49
50     return candids;
51 }
52
53
54 int back(string s) const
55 {
56     auto candids = splits(s);
57     if (candids.empty())
58         return s.size();
59
60     int best = s.size();
61     for (auto c : candids) {
62         string now = s;
63         tie(i, j) = c;
64
65         now.erase(now.begin() + (i - s.begin()), now.begin() + (j - s.begin()));
66
67         int posib = back(now);
68         if (posib < best)
69             best = posib;
70     }
71     return best;
72 }
73
74
75 int solve(string s)
76 {
77     return back(s);
78 }
79 };
80
81 int main()
82 {
83     ifstream fin("zuma.in");
84     ofstream fout("zuma.out");
85
86     int n, k;
87     fin >> n >> k;
88
89     string s;
90     fin >> s;
91
92     assert (s.length() == static_cast<size_t>(n));
93
94     Solver solver(n, k);
95     fout << solver.solve(s) << "\n";
96
97     return 0;
98 }
```

20.6.3 *Rezolvare detaliată

Capitolul 21

ONI 2017

21.1 incurcatura

Problema 1 - incurcatura

100 de puncte

Cu ocazia Olimpiadei Naționale de Informatică, Ninel (fratele mai mic al lui Gigel), a primit un graf neorientat conex G cu N noduri, numerotate de la 1 la N . Acesta a scris pe o foaie lista de vecini corespunzătoare fiecărui nod. Poznaș din fire, Gigel a schimbat lista de vecini pentru unul sau două noduri, schimbând astfel nodurile adiacente acestora. Mai exact, dacă un nod are X vecini, Gigel va scrie tot X vecini, dintre care unii nu vor coincide cu cei scriși inițial de Ninel.

Cerințe

Cunoscând atât numărul de modificări făcute de Gigel, cât și listele de adiacență corespunzătoare fiecărui nod după modificările făcute de Gigel, se cere să se afle nodul sau cele 2 noduri cărora Gigel le-a schimbat listele de adiacență.

Date de intrare

Fisierul **incurcatura.in** conține pe prima linie un singur număr natural P , reprezentând numărul de noduri care au suferit modificări.

Pe a doua linie se va afla un singur număr natural N , reprezentând numărul de noduri ale grafului, iar pe fiecare dintre următoarele N linii, separate printr-un spațiu se află:

- un număr natural K_i , reprezentând numărul de vecini ai nodului i , atât înainte cât și după efectuarea operațiilor;
- K_i numere naturale distințe din multimea $\{1, 2, \dots, n\}$, reprezentând vecinii nodului i , după efectuarea operațiilor.

Date de ieșire

Dacă $P = 1$ fișierul **incurcatura.out** va conține pe prima linie nodul care a fost modificat.

Dacă $P = 2$ fișierul **incurcatura.out** va conține pe prima linie nodurile care au fost modificate, în ordine crescătoare, separate printr-un spațiu.

Restricții și precizări

- pentru datele de intrare problema întotdeauna are soluție
- se garantează că pentru datele de intrare, **solutia este unică**
- $3 \leq N \leq 10^5$
- $1 \leq K_i \leq N - 1$, pentru orice i de la 1 la N
- $K_1 + K_2 + \dots + K_N \leq 4 * 10^5$
- $P \in \{1, 2\}$
- Pentru 40% din teste, $P = 1$

Exemple:

incircatura.in	incircatura.out	Explicații
2 7 4 7 3 2 4 2 6 1 4 7 6 4 1 4 1 3 5 6 2 4 1 3 7 5 1 1 3	1 6	Gigel a modificat listele de adiacență corespunzătoare nodurilor 1 și 6.

Timp maxim de executare/test: **1.2** secunde

Memorie: total **64 MB**

Dimensiune maximă a sursei: **20 KB**

21.1.1 Indicații de rezolvare

Autori: Răzvan Dan Sălăjan, Eugenie-Daniel Posdărăscu, Lucian Bicsi

Soluție: Lucian Bicsi

O primă observație este că, pentru fiecare muchie (a, b) din graf pentru care nu există inversă acesteia (muchia (b, a)), deducem că unul din nodurile a sau b sunt "greșite".

Să tragem o muchie (neorientată) de la a la b pentru fiecare relație de tipul " a sau b este greșit", obținând astfel un graf G' .

Pe noul graf creat, este suficient și necesar să găsim unul (respectiv două) noduri a căror eliminare, împreună cu muchiile adiacente, vor determina ca graful rezultat să nu mai conțină muchii. În literatura de specialitate, o mulțime de noduri cu această proprietate se numește *vertex cover*.

Cu toate acestea, nu avem nevoie de noțiuni de teorie a grafului pentru a rezolva problema.

Se observă că, pentru ca eliminarea unui nod să producă un graf fără muchii, este suficient și necesar ca gradul maxim al unui nod din graf să fie egal cu numărul de muchii al grafului. Pentru două noduri, putem să considerăm pe rând fiecare nod ca fiind unul din cele greșite, să îl eliminăm și, mai apoi, să verificăm dacă gradul maxim în graful rezultat este egal cu numărul de noduri.

Simularea se poate face într-o complexitate bună folosind structuri care mențin dinamic gradul maxim în graf (e.g. *set*, *max-heap*).

Complexitatea finală este $O((n + m)\log(n + m))$ sau $O(n + m)$, în funcție de implementare, ambele soluții obținând 100 de puncte.

Există inclusiv soluții analitice (bazate pe tratarea unor cazuri particulare, în funcție de gradele nodurilor în graf G'), care, analizate cu atenție, obțin tot punctaj maxim, în complexități egale cu cea a citirii grafului.

21.1.2 Cod sursă

Listing 21.1.1: incurcatura_70p.cpp

```

1 #include <bits/stdc++.h>
2
3 #define pii pair<int,int>
4
5 using namespace std;
6
7 ifstream f("incurcatura.in");
8 ofstream g("incurcatura.out");
9
10 const int N = 100010;
11 int n, x, y, i, lies[N];
12
13 set<pii> M;
14 int Pp;
15
16 int main()
17 {

```

```

18     f >> Pp;
19     f >> n;
20     for (x = 1; x <= n; x++)
21     {
22         f >> i;
23         for (; i; i--)
24         {
25             f >> y;
26             pii P = {min(x, y), max(x, y)};
27             if (M.find(P) == M.end())
28                 M.insert(P);
29             else
30                 M.erase(P);
31         }
32     }
33
34     x = y = 0;
35     for (auto it : M)
36     {
37         tie(x, y) = it;
38         lies[x]++;
39         lies[y]++;
40         //cout << x << " " << y << "\n";
41     }
42
43     x = y = 0;
44     for (i = 1; i <= n; i++)
45     {
46         if (lies[i] >= lies[y])
47         {
48             x = y;
49             y = i;
50         }
51         else
52             if (lies[i] >= lies[x])
53                 x = i;
54     }
55
56     if (Pp == 1)
57     {
58         g << y << "\n";
59         return 0;
60     }
61
62     if (x > y) swap(x, y);
63     g << x << ' ' << y << '\n';
64
65     return 0;
66 }
```

Listing 21.1.2: incurcatura_100p_1.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int kMaxN = 100005;
6
7 set<int> G[kMaxN], Bad[kMaxN];
8
9 int sumK;
10
11 int main()
12 {
13     freopen("incurcatura.in", "r", stdin);
14     freopen("incurcatura.out", "w", stdout);
15
16     int n;
17     int p;
18
19     cin >> p;
20     assert(1 <= p && p <= 2);
21
22     cin >> n;
23     assert(1 <= n && n <= 1e5);
```

```

24
25     for (int node = 1; node <= n; ++node)
26     {
27         int k;
28         cin >> k;
29         sumK += k;
30         while (k--)
31         {
32             int x;
33             cin >> x;
34             G[node].insert(x);
35         }
36     }
37
38     assert(1 <= sumK && sumK <= 4 * 100000);
39
40     int bad_edges = 0;
41
42     for (int i = 1; i <= n; ++i)
43         for (auto j : G[i])
44             if (G[j].count(i) == 0)
45             {
46                 Bad[i].insert(j);
47                 Bad[j].insert(i);
48                 ++bad_edges;
49             }
50
51     int sol = -1;
52
53     // One is wrong
54     for (int node = 1; node <= n; ++node)
55     {
56         if (Bad[node].size() == bad_edges)
57         {
58             if (sol != -1)
59             {
60                 cerr << "ANOTHER SOL: " << node << endl;
61                 assert(false);
62             }
63
64             cout << node << endl;
65             sol = 1;
66         }
67     }
68
69     if (sol != -1) return 0;
70
71     auto cmp = [](int a, int b)
72     {
73         return make_pair(Bad[a].size(), a) > make_pair(Bad[b].size(), b);
74     };
75
76     set<int, decltype(cmp)> OrderSet(cmp);
77
78     for (int i = 1; i <= n; ++i)
79     {
80         OrderSet.insert(i);
81     }
82
83     assert(bad_edges > 0);
84
85     // Two are wrong
86     for (int node = 1; node <= n; ++node)
87     {
88         OrderSet.erase(node);
89         for (auto vec : Bad[node])
90         {
91             OrderSet.erase(vec);
92             Bad[vec].erase(node);
93             OrderSet.insert(vec);
94         }
95
96         int oth = *OrderSet.begin();
97
98         if (oth > node && Bad[oth].size() + Bad[node].size() == bad_edges)
99         {

```

```

100         if (sol != -1)
101     {
102         cerr << "ANOTHER SOL: " << node << " " << oth << endl;
103         assert(false);
104     }
105
106     cout << node << " " << oth << endl;
107     sol = 2;
108 }
109
110 for (auto vec : Bad[node])
111 {
112     OrderSet.erase(vec);
113     Bad[vec].insert(node);
114     OrderSet.insert(vec);
115 }
116
117 OrderSet.insert(node);
118 }
119
120 assert(sol == 2);
121
122 return 0;
123 }
```

Listing 21.1.3: incurcatura_100p_2.cpp

```

1 #include<stdio.h>
2 #include<map>
3 #include<vector>
4
5 using namespace std;
6
7 #define NMAX 100005
8
9 map< pair<int,int>, int> mymap;
10
11 int f[NMAX], n, m, possible[NMAX], cnt;
12 vector<int> graph[NMAX], graph2[NMAX];
13
14 void solveOne()
15 {
16     for(int i = 1; i <= n; i++)
17         if(f[i] > 1) { printf("%d\n", i); break; }
18 }
19
20 int main ()
21 {
22     int k, val, degeaba;
23
24     freopen("incurcatura.in","r",stdin);
25     freopen("incurcatura.out","w",stdout);
26
27     scanf("%d",&degeaba);    scanf("%d",&n);
28     for(int i = 1; i <= n; i++)
29     {
30         scanf("%d",&k);
31         for(int j = 1; j <= k; j++)
32         {
33             scanf("%d",&val);
34             mymap[make_pair(i, val)] = 1;
35             graph[i].push_back(val);
36         }
37     }
38
39     for(int i = 1; i <= n; i++)
40     {
41         int k = graph[i].size();
42         for(int j = 0; j < k; j++)
43         {
44             if(!mymap[make_pair(graph[i][j], i)])
45             {
46                 f[i]++;
47                 f[graph[i][j]]++;
48                 graph2[i].push_back(graph[i][j]);
49                 graph2[graph[i][j]].push_back(i);
50             }
51         }
52     }
53
54     if(degeaba == 1) { solveOne(); return 0; }
55 }
```

```

49     for(int i = 1; i <= n; i++)
50         if(f[i] > 2)
51             {   int lim = graph2[i].size();
52                 for(int j = 0; j < lim; j++)
53                     {   int neigh = graph2[i][j]; f[neigh]--;
54                         printf("%d ", i);
55                         f[i] = 0;           solveOne();      return 0;
56                     }
57             else if(f[i] == 2) {   possible[++cnt] = i;}
58
59         if(cnt < 3)
60             {   for(int i = 1; i <= cnt; i++) printf("%d ", possible[i]);
61                 printf("\n");
62             }
63
64         for(int i = 1; i <= 3; i++)
65             for(int j = i + 1; j <= 3; j++)
66                 {   int a = possible[i];
67                     int b = possible[j];
68                     if(mymap[make_pair(a,b)] + mymap[make_pair(b,a)] == 1)
69                     {
70                         f[a]--; f[b]--;
71                     }
72             }
73
74         for(int i = 1; i <= 3; i++)
75             if(f[possible[i]] == 1) printf("%d ", possible[i]);
76
77         printf("\n");
78
79     return 0;
80 }
```

21.1.3 *Rezolvare detaliată

21.2 startrek

Problema 2 - startrek

"Spațiul: ultima frontieră. Acestea sunt călătoriile navei stelare Enterprise. Misiunea ei neîncetată: să exploreze lumi noi și stranii, să caute noi forme de viață și noi civilizații, să meargă acolo unde nimenei nu a mai fost vreodată."



100 de puncte

Jean-Luc Picard - căpitanul navei Enterprise stabilește noua destinație: Imperiul Stelar Romulan. Gaura de vierme nou descoperită de echipaj asigură calea de acces spre Quadrantul Beta - aria galactică unde se află Imperiul Stelar Romulan.

Ipotetic, putem privi gaura de vierme ca un segment de dreaptă partilionat în N sectoare de lungimi egale numerotate de la 1 la N pe care nava le va parcurge exact în această ordine. În funcție de distorsiunile spațiutimp întâlnite, nava poate parcurge într-un an sideral (UTC - timpul universal coordonat) cel puțin p sectoare și cel mult q sectoare. Pentru fiecare sector căpitanul Picard trebuie să informeze Federația despre anul în care a fost parcurs. Datorită interferențelor transmisiei datelor este deficitară. Astfel, Federația nu primește informații din toate cele N sectoare.

Cerințe

Cunoscând descrierea a M transmisiile primite de Federație de forma: $s \ t$, unde s reprezintă indicele unui sector, iar t reprezintă anul în care este parcurs acesta, să se determine numărul maxim de ani în care este străbătută gaura de vierme formată din N sectoare numerotate de la 1 la N .

Date de intrare

Fișierul de intrare **startrek.in** conține pe prima linie patru numere naturale N , p , q și M , separate prin câte un spațiu, cu semnificația din enunț. Pe următoarele M linii se află descrierea celor M transmisiile primite de Federație, în ordinea crescătoare a anilor și a sectoarelor.

Date de ieșire

Fișierul de ieșire **startrek.out** va conține două linii. Pe prima linie se va afla un număr natural nenul A ce reprezintă numărul maxim de ani siderali necesar parcurgerii celor N sectoare. Pe cea de a doua linie se vor găsi N valori despărțite prin câte un spațiu ce reprezintă descrierea temporală minimală lexicografică a sectoarelor parcuse, pentru fiecare sector fiind specificat anul.

Restricții și precizări

- $2 \leq N \leq 100\,000$
- $2 \leq p < q \leq N$
- $1 \leq M \leq N$
- fiecare sector trebuie parcurs în totalitate în cadrul aceluiași an sideral;
- întreaga călătorie trebuie parcursă într-un număr întreg de ani siderali (cu alte cuvinte: parcurgerea a cel puțin p sectoare și cel mult q sectoare este valabilă și pentru primul și ultimul an);
 - pentru datele de intrare există întotdeauna soluție;
 - pentru 30 de puncte $2 \leq N \leq 100$, $2 \leq p < q \leq 50$
 - pentru 70 de puncte $2 \leq N \leq 30000$, $2 \leq p < q \leq 100$
 - pentru 100 de puncte $2 \leq N \leq 100000$, $2 \leq p < q \leq N$

Exemple:

startrek.in	startrek.out	Explicații
5 2 3 1 2 1	2 1 1 1 2 2	știm că al doilea sector este parcurs în primul an. 1 1 2 2 3 nu poate fi soluție deoarece în anul 3 nu sunt parcurse minim 2 sectoare. Sunt necesari 2 ani pentru parcurgerea celor 5 sectoare. Primele 3 sectoare vor fi parcuse în primul an, următoarele 2 sectoare în cel de-al doilea an.
7 2 5 2 2 1 6 3	3 1 1 1 2 2 3 3	Sunt necesari 3 ani pentru parcurgerea celor 7 sectoare. Primele 3 sectoare vor fi parcuse în primul an, următoarele 2 sectoare în cel de-al doilea an, iar ultimele 2 sectoare în cel de-al treilea an.
16 3 4 2 5 2 15 5	5 1 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5	Sunt necesari 5 ani pentru parcurgerea celor 16 sectoare.

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

Memorie: total **64 MB**

Dimensiune maximă a sursei: **20 KB**

21.2.1 Indicații de rezolvare

Autor: Eugen Nodea - prof. Colegiul Național "Tudor Vladimirescu" Tg Jiu |index[person]Eugen Nodea

Soluție: Lucian Bicsi

Una din soluțiile problemei este folosind *metoda programării dinamice*. Soluția va consta în două parcurgeri: una de la 1 la n , în care aflăm numărul maxim și a doua de la n la 1, în care determinăm soluția minimă lexicografică. Vom prezenta ideea doar în cadrul parcurgerii "stângă-dreptă", cu considerențele că cealaltă parcurgere este foarte asemănătoare.

Fie $OK[i][j] = \text{true}$ dacă pe poziția i se poate termina un bloc continuu cu valori egale cu j .

Pentru a calcula această dinamică, avem cazul de bază $OK[0][0] = \text{true}$ și, pentru $1 \leq i, j \leq n$ vom căuta $i' < i$ care respectă $p \leq i - i' + 1 \leq q$ și $OK[i'][j - 1] = \text{true}$. Important este ca, în

plus, pentru orice poziție între $i' + 1$ și i să nu existe restricții de valori diferite de j . Soluția are complexitate $O(n * \text{maxx} * q)$ și ar trebui să obțină între 30 și 70 de puncte, unde maxx este numărul maxim de ani.

Să optimizăm acum soluția de la pasul anterior. Observația-cheie pentru aceasta este că, pentru un i fixat, $\text{OK}[i][j] = \text{true}$ pe un interval compact de valori. Cu alte cuvinte, este suficient să calculăm capetele acestui interval, pe care le vom numi de acum $\text{Min}[i]$, respectiv $\text{Max}[i]$.

Pentru a calcula aceste valori, observăm că putem fixa $i' < i$ care respectă proprietatea $p \leq i - i' + 1 \leq q$ și obținem trei cazuri:

1. Nu există restricții în intervalul $[i' + 1, i]$: "reunim" intervalul $[\text{Min}[i], \text{Max}[i]]$ cu $[\text{Min}[i'] + 1, \text{Max}[i'] + 1]$

2. Există restricții de o singură valoare (x) în intervalul $[i' + 1, i]$: "reunim" intervalul $[\text{Min}[i], \text{Max}[i]]$ cu x , doar dacă $x - 1$ aparține intervalului $[\text{Min}[i'], \text{Max}[i']]$

3. Există restricții de cel puțin două valori distincte în intervalul $[i' + 1, i]$: nu facem nimic

Soluția are complexitate $O(n * q)$ și, implementată cu grijă, ar trebui să obțină cel puțin 70 de puncte.

Să optimizăm din nou soluția anterioară. Se observă că cele 3 cazuri reprezintă intervale compacte de valori i' . Astfel, putem folosi un arbore de intervale / aib care să de furnizeze răspunsuri de tip minim / maxim pe interval în timp logaritmic și, astfel, să actualizăm o poziție a dinamicii în timp logaritmic.

Complexitate finală: $O(n \log(n))$

Pentru a calcula minimul lexicografic, pornim cu $\text{OK}[n + 1][\text{maxx} + 1] = \text{true}$, unde, în noile considerente, $\text{OK}[i][j] = \text{true}$ dacă pe poziția i poate începe un bloc continuu cu valori egale cu j (respectiv noile definiții în acest caz pentru $\text{Max}[i]$ și $\text{Min}[i]$). Dinamica se construiește de la dreapta la stânga, într-o manieră similară descrierii de mai sus.

O dată calculată această dinamică, construirea sirului se poate realiza în timp liniar, necesitând totuși câteva detalii de implementare.

Notăm faptul că există rezolvări care folosesc euristică (greedy) foarte bune pentru a estima valoarea maximă și pentru a construi sirul, care pot lua punctaje variate.

Solutie 2 (Eugen Nodea):

- 1) pentru fiecare an fixat reținem cel mai din stânga sector $\text{st}[i]$, respectiv cel mai din dreapta sector fixat $\text{dr}[i]$

- 2) vom construi doi vectorii auxiliari: $\text{start_st}[i]$, dacă se ajunge în sectorul i cu număr minim de ani (p) și $\text{start_dr}[i]$, cu număr maxim de ani (q). Pentru updatarea următorului sector $i+1$ avem cazurile:

```

start_dr[i + 1] =
min(start_dr[i + 1], st[i + 1] + MAX - 1) start_st[i + 1] =
max(start_st[i + 1], dr[i + 1] - MAX + 1)

```

- 3) pentru determinarea numărului maxim de ani vom pleca de la ultimul sector fixat, căutând un ultim an/sector ce permite respectarea condiției necesare (în ultimul an trebuie parcurs cel puțin p sectoare)

- 4) Pentru construirea soluției minim lexicografic vom pleca de la sfârșit

21.2.2 Cod sursă

Listing 21.2.1: startrek_30p.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int kMaxN = 500005;
6 int V[kMaxN];
7 int n, lo, hi;
8
9 pair<int, int> DP[kMaxN];
10
11 pair<int, int> unite(pair<int, int> a, pair<int, int> b)
12 {
13     if(a.first == -1) return b;
14     if(b.first == -1) return a;
15 }
```

```

16     return { min(a.first, b.first), max(a.second, b.second) };
17 }
18
19 pair<int, int> intersect(pair<int, int> a, pair<int, int> b)
20 {
21     if(a.first == -1) return a;
22     if(b.first == -1) return b;
23
24     auto ret = make_pair(max(a.first, b.first), min(a.second, b.second));
25     if(ret.first > ret.second) return {-1, -1};
26     return ret;
27 }
28
29 struct ArbInt
30 {
31     vector<pair<int, int>> T;
32
33     ArbInt(int n) : T(4 * n, make_pair(-1, -1)) {}
34
35     void Update(int node, int b, int e, int pos, pair<int, int> val)
36     {
37         if(b == e)
38         {
39             assert(b == pos);
40             T[node] = val;
41         }
42         else
43         {
44             int m = (b + e) / 2;
45             if(pos <= m)
46                 Update(node * 2, b, m, pos, val);
47             else
48                 Update(node * 2 + 1, m + 1, e, pos, val);
49
50             T[node] = unite(T[node * 2], T[node * 2 + 1]);
51         }
52     }
53
54     pair<int, int> Query(int node, int b, int e, int l, int r)
55     {
56         if(b >= l && e <= r) return T[node];
57         if(b > r || e < l) return make_pair(-1, -1);
58
59         int m = (b + e) / 2;
60         return unite(Query(node * 2, b, m, l, r),
61                     Query(node * 2 + 1, m + 1, e, l, r));
62     }
63 };
64
65 template<typename cmp>
66 struct Poset
67 {
68     vector<int> Pos;
69     set<int, cmp> Set;
70
71     Poset(int n) : Pos(n, -1) {}
72
73     void Update(int pos, int val)
74     {
75         if(Pos[val] != -1)
76         {
77             Set.erase(Pos[val]);
78         }
79         Pos[val] = pos;
80         Set.insert(pos);
81     }
82
83     pair<int, int> GetLastTwo()
84     {
85         auto it = Set.begin();
86         pair<int, int> ret(-1, -1);
87
88         if(it != Set.end())
89         {
90             ret.first = *it;
91             ++it;
92         }
93     }

```

```

92         if(it != Set.end())
93             ret.second = *it;
94     }
95
96     return ret;
97 }
98 };
99
100 pair<int, int> DPX[kMaxN];
101
102 int ForwardPropX()
103 {
104     DPX[0] = {0, 0};
105
106     for(int i = 1; i <= n; ++i)
107     {
108         int last = -1;
109         DPX[i] = {-1, -1};
110
111         for(int k = 0; k <= hi; ++k)
112         {
113             if(i - k < 0) break;
114
115             if(k >= lo && DP[i - k].first != -1)
116             {
117                 if(last == -1)
118                 {
119                     DPX[i] = unite(DPX[i], make_pair(DPX[i - k].first + 1,
120                                         DPX[i - k].second + 1));
121                 }
122                 else
123                 {
124                     if(DPX[i - k].first + 1 <= last &&
125                         last <= DPX[i - k].second + 1)
126                         DPX[i] = unite(DPX[i], make_pair(last, last));
127                 }
128             }
129
130             if(V[i - k] != -1)
131             {
132                 if(last == -1) last = V[i - k];
133                 if(last != V[i - k]) break;
134             }
135         }
136     }
137
138     return DPX[n].second;
139 }
140
141 int ForwardProp()
142 {
143     DP[0] = {0, 0};
144
145     ArbInt ai(n + 2);
146     Poset<greater<int>> pos(n + 2);
147
148     ai.Update(1, 0, n, 0, DP[0]);
149     for(int i = 1; i <= n; ++i)
150     {
151         DP[i] = {-1, -1};
152
153         if(V[i] != -1) pos.Update(i, V[i]);
154         auto lasts = pos.GetLastTwo();
155
156         pair<int, int> inter, range;
157
158         inter = intersect(make_pair(max(0, i - hi), i - lo),
159                           make_pair(lasts.first == -1 ? 0 : lasts.first, i));
160         inter = intersect(inter, make_pair(0, n + 1));
161         range = ai.Query(1, 0, n, inter.first, inter.second);
162
163         if(range.first != -1)
164         {
165             DP[i] = unite(DP[i], make_pair(range.first+1, range.second+1));
166         }
167

```

```

168     if(lasts.first != -1)
169     {
170         inter = intersect(make_pair(max(0, i - hi), i - lo),
171                            make_pair(lasts.second == -1 ?
172                                         0 : lasts.second, lasts.first-1));
173         inter = intersect(inter, make_pair(0, n + 1));
174         range = ai.Query(1, 0, n, inter.first, inter.second);
175
176         if(range.first <= V[lasts.first] - 1 &&
177             V[lasts.first] - 1 <= range.second)
178         {
179             DP[i]=unite(DP[i], make_pair(V[lasts.first], V[lasts.first]));
180         }
181     }
182
183     ai.Update(1, 0, n, i, DP[i]);
184 }
185
186 return DP[n].second;
187 }
188
189 void BackPropX(int maxx)
190 {
191     DPX[n + 1] = {maxx + 1, maxx + 1};
192
193     for(int i = n; i >= 1; --i)
194     {
195         int last = -1;
196         DPX[i] = {-1, -1};
197
198         for(int k = 0; k <= hi; ++k)
199         {
200             if(i + k > n + 1) break;
201
202             if(k >= lo && DPX[i + k].first != -1)
203             {
204                 if(last == -1)
205                 {
206                     DPX[i] = unite(DPX[i],
207                                     make_pair(max(0,
208                                               DPX[i + k].first - 1),
209                                               max(0, DPX[i + k].second - 1)));
210                 }
211                 else
212                 {
213                     if(DPX[i + k].first - 1 <= last &&
214                         last <= DPX[i + k].second - 1)
215                         DPX[i] = unite(DPX[i], make_pair(last, last));
216                 }
217             }
218
219             if(V[i + k] != -1)
220             {
221                 if(last == -1) last = V[i + k];
222                 if(last != V[i + k]) break;
223             }
224         }
225     }
226 }
227
228 void BackProp(int maxx)
229 {
230     DP[n + 1] = {maxx + 1, maxx + 1};
231
232     ArbInt ai(n + 2);
233     Poset<less<int>> pos(n + 2);
234
235     ai.Update(1, 1, n + 1, n + 1, DP[n + 1]);
236     for(int i = n; i >= 1; --i)
237     {
238         DP[i] = {-1, -1};
239
240         if(V[i] != -1) pos.Update(i, V[i]);
241         auto lasts = pos.GetLastTwo();
242
243         pair<int, int> inter, range;

```

```

244     inter = intersect(make_pair(i + lo, i + hi),
245         make_pair(i, lasts.first == -1 ? n + 1 : lasts.first));
246     inter = intersect(inter, make_pair(0, n + 1));
247     range = ai.Query(1, 1, n + 1, inter.first, inter.second);
248
249     if(range.first != -1)
250     {
251         DP[i] = unite(DP[i],
252             make_pair(max(0, range.first-1), max(0, range.second-1)));
253     }
254
255     if(lasts.first != -1)
256     {
257         inter = intersect(make_pair(i + lo, i + hi),
258             make_pair(i, lasts.second == -1 ? n + 1 : lasts.second));
259         inter = intersect(inter, make_pair(0, n + 1));
260         range = ai.Query(1, 1, n + 1, inter.first, inter.second);
261
262         if(range.first <= V[lasts.first] + 1 &&
263             V[lasts.first] + 1 <= range.second)
264         {
265             DP[i]=unite(DP[i], make_pair(V[lasts.first], V[lasts.first]));
266         }
267     }
268
269     ai.Update(1, 1, n + 1, i, DP[i]);
270 }
271 }
272
273 int main()
274 {
275     ifstream cin("startrek.in");
276     ofstream cout("startrek.out");
277
278     int m;
279     cin >> n >> lo >> hi >> m;
280
281     assert(n <= 100 && hi <= 50);
282
283     memset(V, -1, sizeof(V));
284
285     while(m--)
286     {
287         int pos, val;
288         cin >> pos >> val;
289         V[pos] = val;
290     }
291
292     int maxx = ForwardProp();
293     BackProp(maxx);
294
295     cout << maxx << endl;
296     int pos = 1;
297     for(int val = 1; val <= maxx; ++val)
298     {
299         int go = -1;
300         for(int nxt = pos + lo; nxt <= pos + hi; ++nxt)
301         {
302             if(nxt > n + 1) break;
303
304             if(DP[nxt].first != -1 && DP[nxt].first <= val + 1) go = nxt;
305
306             if(V[nxt] != -1 && V[nxt] != val)
307                 break;
308         }
309
310         assert(go != -1);
311         while(pos != go)
312         {
313             cout << val << " ";
314             ++pos;
315         }
316     }
317
318     assert(pos == n + 1);

```

```

320     cout << endl;
321
322     return 0;
323 }
```

Listing 21.2.2: startrek_70p.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int kMaxN = 500005;
6 int V[kMaxN];
7 int n, lo, hi;
8
9 pair<int, int> DP[kMaxN];
10
11 pair<int, int> unite(pair<int, int> a, pair<int, int> b)
12 {
13     if(a.first == -1) return b;
14     if(b.first == -1) return a;
15     return { min(a.first, b.first), max(a.second, b.second) };
16 }
17
18 pair<int, int> intersect(pair<int, int> a, pair<int, int> b)
19 {
20     if(a.first == -1) return a;
21     if(b.first == -1) return b;
22
23     auto ret = make_pair(max(a.first, b.first), min(a.second, b.second));
24     if(ret.first > ret.second) return {-1, -1};
25     return ret;
26 }
27
28 int ForwardProp()
29 {
30     DP[0] = {0, 0};
31
32     for(int i = 1; i <= n; ++i)
33     {
34         int last = -1;
35         DP[i] = {-1, -1};
36
37         for(int k = 0; k <= hi; ++k)
38         {
39             if(i - k < 0) break;
40
41             if(k >= lo && DP[i - k].first != -1)
42             {
43                 if(last == -1)
44                 {
45                     DP[i] = unite(DP[i],
46                                   make_pair(DP[i - k].first + 1,
47                                             DP[i - k].second + 1));
48                 }
49                 else
50                 {
51                     if(DP[i - k].first + 1 <= last &&
52                         last <= DP[i - k].second + 1)
53                         DP[i] = unite(DP[i], make_pair(last, last));
54                 }
55             }
56
57             if(V[i - k] != -1)
58             {
59                 if(last == -1) last = V[i - k];
60                 if(last != V[i - k]) break;
61             }
62         }
63     }
64     return DP[n].second;
65 }
66
67 void BackProp(int maxx)
68 {
```

```

69     DP[n + 1] = {maxx + 1, maxx + 1};
70
71     for(int i = n; i >= 1; --i)
72     {
73         int last = -1;
74         DP[i] = {-1, -1};
75
76         for(int k = 0; k <= hi; ++k)
77         {
78             if(i + k > n + 1) break;
79
80             if(k >= lo && DP[i + k].first != -1)
81             {
82                 if(last == -1)
83                 {
84                     DP[i] = unite(DP[i],
85                                 make_pair(DP[i + k].first - 1,
86                                           DP[i + k].second - 1));
87                 }
88                 else
89                 {
90                     if(DP[i + k].first - 1 <= last && last <= DP[i + k].second - 1)
91                         DP[i] = unite(DP[i], make_pair(last, last));
92                 }
93             }
94
95             if(V[i + k] != -1)
96             {
97                 if(last == -1) last = V[i + k];
98                 if(last != V[i + k]) break;
99             }
100        }
101    }
102 }
103 }
104
105 int main()
106 {
107     freopen("startrek.in", "r", stdin);
108     freopen("startrek.out", "w", stdout);
109
110     int m;
111     assert(cin >> n >> lo >> hi >> m);
112
113     assert(n <= 30000 && hi <= 300);
114
115     memset(V, -1, sizeof(V));
116
117     while(m--)
118     {
119         int pos, val;
120         assert(cin >> pos >> val);
121         V[pos] = val;
122     }
123
124     int maxx = ForwardProp();
125     BackProp(maxx);
126
127     cout << maxx << endl;
128     int pos = 1;
129     for(int val = 1; val <= maxx; ++val)
130     {
131         int go = -1;
132         for(int nxt = pos + lo; nxt <= pos + hi; ++nxt)
133         {
134             if(nxt > n + 1) break;
135
136             if(DP[nxt].first != -1 && DP[nxt].first <= val + 1) go = nxt;
137
138             if(V[nxt] != -1 && V[nxt] != val)
139                 break;
140         }
141
142         assert(go != -1);
143         while(pos != go)
144         {

```

```

145         cout << val << " ";
146         ++pos;
147     }
148 }
149
150 assert(pos == n + 1);
151 cout << endl;
152
153 return 0;
154 }
```

Listing 21.2.3: startrek_80p_1.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int kMaxN = 500005;
6 int V[kMaxN];
7 int n, lo, hi;
8
9 pair<int, int> DP[kMaxN];
10
11 pair<int, int> unite(pair<int, int> a, pair<int, int> b)
12 {
13     if(a.first == -1) return b;
14     if(b.first == -1) return a;
15     return { min(a.first, b.first), max(a.second, b.second) };
16 }
17
18 pair<int, int> intersect(pair<int, int> a, pair<int, int> b)
19 {
20     if(a.first == -1) return a;
21     if(b.first == -1) return b;
22
23     auto ret = make_pair(max(a.first, b.first), min(a.second, b.second));
24     if(ret.first > ret.second) return {-1, -1};
25     return ret;
26 }
27
28 int ForwardProp()
29 {
30     DP[0] = {0, 0};
31
32     for(int i = 1; i <= n; ++i)
33     {
34         int last = -1;
35         DP[i] = {-1, -1};
36
37         for(int k = 0; k <= hi; ++k)
38         {
39             if(i - k < 0) break;
40
41             if(k >= lo && DP[i - k].first != -1)
42             {
43                 if(last == -1)
44                 {
45                     DP[i] = unite(DP[i],
46                                   make_pair(DP[i - k].first + 1,
47                                             DP[i - k].second + 1));
48                 }
49                 else
50                 {
51                     if(DP[i - k].first + 1 <= last &&
52                         last <= DP[i - k].second + 1)
53                         DP[i] = unite(DP[i], make_pair(last, last));
54                 }
55             }
56
57             if(V[i - k] != -1)
58             {
59                 if(last == -1) last = V[i - k];
60                 if(last != V[i - k]) break;
61             }
62         }
63     }
64 }
```

```

63     }
64     return DP[n].second;
65 }
66
67 void BackProp(int maxx)
68 {
69     DP[n + 1] = {maxx + 1, maxx + 1};
70
71     for(int i = n; i >= 1; --i)
72     {
73         int last = -1;
74         DP[i] = {-1, -1};
75
76         for(int k = 0; k <= hi; ++k)
77         {
78             if(i + k > n + 1) break;
79
80             if(k >= lo && DP[i + k].first != -1)
81             {
82                 if(last == -1)
83                 {
84                     DP[i] = unite(DP[i],
85                                   make_pair(DP[i + k].first - 1,
86                                             DP[i + k].second - 1));
87                 }
88                 else
89                 {
90                     if(DP[i + k].first - 1 <= last &&
91                         last <= DP[i + k].second - 1)
92                         DP[i] = unite(DP[i], make_pair(last, last));
93                 }
94             }
95
96             if(V[i + k] != -1)
97             {
98                 if(last == -1) last = V[i + k];
99                 if(last != V[i + k]) break;
100            }
101        }
102    }
103 }
104
105 int main()
106 {
107     freopen("startrek.in", "r", stdin);
108     freopen("startrek.out", "w", stdout);
109
110     int m;
111     cin >> n >> lo >> hi >> m;
112
113     memset(V, -1, sizeof(V));
114
115     while(m--)
116     {
117         int pos, val;
118         cin >> pos >> val;
119         V[pos] = val;
120     }
121
122
123     int maxx = ForwardProp();
124     BackProp(maxx);
125
126     cout << maxx << endl;
127     int pos = 1;
128     for(int val = 1; val <= maxx; ++val)
129     {
130         int go = -1;
131         for(int nxt = pos + lo; nxt <= pos + hi; ++nxt)
132         {
133             if(nxt > n + 1) break;
134
135             if(DP[nxt].first != -1 && DP[nxt].first <= val + 1) go = nxt;
136
137             if(V[nxt] != -1 && V[nxt] != val)
138                 break;

```

```

139         }
140
141         assert(go != -1);
142         while(pos != go)
143         {
144             cout << val << " ";
145             ++pos;
146         }
147     }
148
149     assert(pos == n + 1);
150     cout << endl;
151
152     return 0;
153 }
```

Listing 21.2.4: startrek_80p_2.cpp

```

1  /*      Radu      O(n) ? */
2  #include <iostream>
3  #include <cstring>
4  #include <algorithm>
5
6  using namespace std;
7
8  const int Q=1000000;
9
10 int n,p,q,m;
11
12 struct tipe
13 {
14     int poz;
15     int val;
16 } v[Q];
17
18 bool operator <(const tipe &a, const tipe &b)
19 {
20     if(a.val!=b.val)
21         return a.val<b.val;
22     return a.poz<b.poz;
23 }
24
25 int puse[Q];
26 int max_b1[Q];
27 int disp[Q];
28 //int capat_dr[Q];
29 int pnt[Q];
30
31 int aib[Q];
32
33 int ask(int poz)
34 {
35     int rez=0;
36     while(poz)
37     {
38         rez+=aib[poz];
39         poz-=poz&(-poz);
40     }
41     return rez;
42 }
43
44 void update(int val,int poz)
45 {
46     while(poz<Q)
47     {
48         aib[poz]+=val;
49         poz+=poz&(-poz);
50     }
51 }
52
53 int elibereaza(int no,int cursor, int pozitie)
54 {
55     int loc=no;
56
57     while(cursor>=pozitie)
```

```

58     {
59         while(disp[pnt[loc]]==0)
60         {
61             pnt[loc]=pnt[pnt[loc]];
62         }
63         loc=pnt[loc];
64
65         if(cursor-disp[loc]>=pozitie-1)
66         {
67             cursor-=disp[loc];
68             puse[loc]-=disp[loc];
69             update(-disp[loc],loc);
70             // capat_dr[loc]-=disp[loc];
71             disp[loc]=0;
72         }
73         else
74         {
75             puse[loc]==cursor-pozitie+1;
76             disp[loc]==cursor-pozitie+1;
77             update(-(cursor-pozitie+1),loc);
78             // capat_dr[loc]==cursor-pozitie+1;
79             cursor=pozitie-1;
80             return cursor;
81         }
82     }
83
84     return cursor;
85 }
86
87 void get_from(int need,int ultimul)
88 {
89     for(int no=ultimul; no>0; no=pnt[no])
90     {
91         if(disp[no] <=need)
92         {
93             update(-disp[no],no);
94             //capat_dr[no]-=disp[no];
95
96             puse[no]-=disp[no];
97             need-=disp[no];
98             disp[no]=0;
99         }
100        else
101        {
102            puse[no]-=need;
103            disp[no]-=need;
104            update(-need,no);
105            need=0;
106            return;
107        }
108    }
109 }
110
111 int can_get(int ultimul)
112 {
113     int rez=0;
114
115     int max=2000000000;
116
117     for(int no=ultimul; no>0; no=pnt[no])
118     {
119         if(disp[no]<=max)
120         {
121             max-=disp[no];
122             rez+=disp[no];
123         }
124         else
125         {
126             rez+=max;
127             max=0;
128             break;
129         }
130
131         if(ask(no)-max_b1[no] < max)
132             max=ask(no)-max_b1[no];
133     }

```

```

134     return rez;
135 }
136 }
137
138 void insereaza(int who,int cv)
139 {
140     //who++;
141     while(cv)
142     {
143         if(cv >= q-puse[who])
144         {
145             cv-=q-puse[who];
146             puse[who]=q;
147         }
148         else
149         {
150             puse[who]+=cv;
151             break;
152         }
153         who++;
154     }
155 }
156
157 int main()
158 {
159     freopen("startrek.in","r",stdin);
160     freopen("startrek.out","w",stdout);
161
162     scanf("%d%d%d",&n,&p,&q,&m);
163
164     for(int i=1; i<=m; i++)
165         scanf("%d%d",&v[i].poz, &v[i].val);
166
167     std::sort(v+1,v+m+1);
168
169     int last_val=0;
170     int last_poz=0;
171
172     int th=0;
173
174     int cursor=0;
175
176     int ultimul;
177
178     for(int no=1; cursor<n || th<m; no++)
179     {
180         ultimul=no;
181         if(disp[no-1])
182             pnt[no]=no-1;
183         else
184             pnt[no]=pnt[no-1];
185
186         if(v[th+1].val!=no)
187         {
188
189             update(q,no);
190             // capat_dr[no]=cursor+q;
191             cursor+=q;
192             puse[no]=q;
193             disp[no]=q-p;
194
195             continue;
196         }
197
198         while(v[th+1].val==no)
199         {
200             th++;
201
202             max_b1[no]=v[th].poz;
203
204             if(v[th].val==v[th-1].val)
205             {
206
207                 //disp[no]=min(capat_dr[no]-v[th].poz,q-p);
208                 int my=ask(no);
209                 disp[no]=min(my-v[th].poz,q-p);

```

```

210         }
211     else
212     {
213         cursor=elibereaza(no,cursor,v[th].poz);
214
215         update(q,no);
216         //capat_dr[no]=cursor+q;
217         cursor+=q;
218         puse[no]=q;
219         //disp[no]=min(q+capat_dr[no-1]-max_b1[no], q-p);
220
221         int my=ask(no-1);
222         disp[no]=min(q+my-max_b1[no], q-p);
223     }
224 }
225
226 if(th==m)
227     break;
228 }
229
230 int surplus;
231
232 update(-disp[ultimul],ultimul);
233 //capat_dr[ultimul]==disp[ultimul];
234 puse[ultimul]=disp[ultimul];
235 disp[ultimul]=0;
236
237 int my=ask(ultimul);
238
239 int lul=ultimul;
240
241 surplus=n-my;
242
243 if(surplus<0)
244 {
245     get_from(-surplus,ultimul-1);
246     surplus=0;
247 }
248
249 surplus+=can_get(ultimul);
250 surplus=(surplus/p)*p;
251 my=ask(ultimul);
252
253 int need=surplus-(n-my);
254 int cv=0;
255
256 if(need<0)
257 {
258     cv=-need;
259     need=0;
260 }
261
262 get_from(need, ultimul);
263
264 while(surplus>=p)
265 {
266     ultimul++;
267     if(disp[ultimul-1])
268         pnt[ultimul]=ultimul-1;
269     else
270         pnt[ultimul]=pnt[ultimul-1];
271
272     surplus-=p;
273     puse[ultimul]=p;
274     disp[ultimul]=0;
275     //    capat_dr[ultimul]=capat_dr[ultimul-1]+p;
276     int my=ask(ultimul-1);
277
278     update(p,ultimul);
279 }
280
281 //afisare();
282 printf("%d\n",ultimul);
283
284 insereaza(lul,cv);

```

```

286
287     for(int i=1; i<=ultimul; i++)
288         for(int j=1; j<=puse[i]; j++)
289             printf("%d ",i);
290
291     return 0;
292 }
```

Listing 21.2.5: startrek_90p.cpp

```

1 #include<bits/stdc++.h>
2
3 using namespace std;
4
5 ifstream in("startrek.in");
6 ofstream out("startrek.out");
7
8 const int Nmax = 100001;
9 int a[Nmax],oc[Nmax],lef[Nmax],ret[Nmax];
10 int N,M,p,q;
11
12 vector<int> poz;
13
14 int main()
15 {
16     in>>N>>p>>q>>M;
17     for(int i=1;i<=M;i++)
18     {
19         int pos,val;
20         in>>pos>>val;
21         a[pos]=val;
22     }
23
24     for(int j=1;j<=N;j++) if(a[j]!=0) poz.push_back(j);
25
26     int nxl=0,extlef=1,sum=0,j=0;
27     int stp=0,crp=0;
28
29     if(M==0)
30         stp=1;
31     else
32         crp=poz[nxl];
33
34     while(sum<N)
35     {
36         j++;
37         oc[j]=p;
38         lef[j]=q-p;
39         sum+=p;
40         while(sum>=crp)
41         {
42             if(a[crp]!=j)
43             {
44                 // suma de OC
45                 int miz=0;
46                 for(int k=1;k<=a[crp];k++) miz+=oc[k];
47
48                 int need = crp-miz;
49                 while(lef[extlef]<need)
50                 {
51                     oc[extlef]+=lef[extlef];
52                     sum+=lef[extlef];
53                     need-=lef[extlef];
54                     lef[extlef]=0;
55                     extlef++;
56                 }
57
58                 oc[extlef]+=need;
59                 sum+=need;
60                 lef[extlef]-=need;
61                 need=0;
62             }
63             /// setare restrictie automata
64
65             /// suma de OC
```

```

66         int miz=0;
67         for(int k=1;k<a[crp];k++)
68             miz+=oc[k];
69
70         miz++;
71
72         int exag = crp-miz;
73         int k=extlef;
74
75         while(k<a[crp] && lef[k]<exag)
76             exag-=lef[k++];
77
78         if(k<a[crp])
79             lef[k++]=exag;
80
81         while(k<a[crp]) lef[k++]=0;
82
83         /// avansare
84         nxl++;
85         if(nxl<poz.size())
86             crp=poz[nxl];
87         else
88         {
89             while(sum>N)
90             {
91                 if(oc[j]<=sum-N)
92                 {
93                     sum-=oc[j];
94                     lef[j]+=oc[j];
95                     oc[j]=0;
96                     j--;
97                 }
98                 else
99                 {
100                     while(sum>N)
101                     {
102                         sum--;
103                         lef[j]++;
104                         oc[j]--;
105                     }
106                 }
107             }
108
109             if(oc[j]<p)
110             {
111                 sum-=oc[j];
112                 oc[j]=0;
113                 j--;
114             }
115
116             stp=1;
117             break;
118         }
119     }
120
121     if(stp)
122     {
123         if(N-sum>0)
124         {
125             while(N-sum>0)
126             {
127                 j++;
128                 oc[j]=p;
129                 lef[j]=q-p;
130                 sum+=p;
131             }
132
133             while(sum>N)
134                 sum--,oc[j]--,lef[j]++;
135
136             int need=0;
137             if(oc[j]<p)
138             {
139                 sum-=oc[j];
140                 need=oc[j];
141                 oc[j]=0;

```

```

142             j--;
143         }
144
145         if (need)
146         {
147             while (lef[extlef] < need)
148             {
149                 oc[extlef] += lef[extlef];
150                 sum += lef[extlef];
151                 need -= lef[extlef];
152                 lef[extlef] = 0;
153                 extlef++;
154             }
155
156             oc[extlef] += need;
157             sum += need;
158             lef[extlef] -= need;
159             need = 0;
160         }
161     }
162
163     int sz = 0;
164     for (int i = 1; i <= N; i++)
165     {
166         for (int j = 1; j <= oc[i]; j++)
167             if (sz < N) ret[++sz] = i;
168
169         break;
170     }
171
172     out << ret[N] << '\n';
173     for (int i = 1; i <= N; i++)
174         out << ret[i] << ' '; out << '\n';
175
176     return 0;
177 }
```

Listing 21.2.6: startrek_100p_1.cpp

```

1 //100 p - prof. Lucian Bicsi
2
3 #include <bits/stdc++.h>
4
5 using namespace std;
6
7 const int kMaxN = 500005;
8 int V[kMaxN];
9 int n, lo, hi;
10
11 pair<int, int> DP[kMaxN];
12
13 pair<int, int> unite(pair<int, int> a, pair<int, int> b)
14 {
15     if (a.first == -1) return b;
16     if (b.first == -1) return a;
17     return { min(a.first, b.first), max(a.second, b.second) };
18 }
19
20 pair<int, int> intersect(pair<int, int> a, pair<int, int> b)
21 {
22     if (a.first == -1) return a;
23     if (b.first == -1) return b;
24
25     auto ret = make_pair(max(a.first, b.first), min(a.second, b.second));
26     if (ret.first > ret.second) return {-1, -1};
27     return ret;
28 }
29
30 struct ArbInt
31 {
32     vector<pair<int, int>> T;
33
34     ArbInt(int n) : T(4 * n, make_pair(-1, -1)) {}
35
36     void Update(int node, int b, int e, int pos, pair<int, int> val)
```

```

37     {
38         if(b == e)
39         {
40             assert(b == pos);
41             T[node] = val;
42         }
43         else
44         {
45             int m = (b + e) / 2;
46             if(pos <= m)
47             {
48                 Update(node * 2, b, m, pos, val);
49             }
50             else
51             {
52                 Update(node * 2 + 1, m + 1, e, pos, val);
53             }
54
55             T[node] = unite(T[node * 2], T[node * 2 + 1]);
56         }
57     }
58
59     pair<int, int> Query(int node, int b, int e, int l, int r)
60     {
61         if(b >= l && e <= r) return T[node];
62         if(b > r || e < l) return make_pair(-1, -1);
63
64         int m = (b + e) / 2;
65         return unite(Query(node * 2, b, m, l, r),
66                     Query(node * 2 + 1, m + 1, e, l, r));
67     }
68 };
69
70 template<typename cmp>
71
72 struct Poset
73 {
74     vector<int> Pos;
75
76     set<int, cmp> Set;
77
78     Poset(int n) : Pos(n, -1) {}
79
80     void Update(int pos, int val)
81     {
82         if(Pos[val] != -1)
83         {
84             Set.erase(Pos[val]);
85         }
86         Pos[val] = pos;
87         Set.insert(pos);
88     }
89
90     pair<int, int> GetLastTwo()
91     {
92         auto it = Set.begin();
93         pair<int, int> ret(-1, -1);
94
95         if(it != Set.end())
96         {
97             ret.first = *it;
98             ++it;
99             if(it != Set.end())
100                 ret.second = *it;
101         }
102
103         return ret;
104     }
105 };
106
107 int ForwardProp()
108 {
109     DP[0] = {0, 0};
110
111     ArbInt ai(n + 2);
112     Poset<greater<int>> pos(n + 2);

```

```

113
114     ai.Update(1, 0, n, 0, DP[0]);
115     for(int i = 1; i <= n; ++i)
116     {
117         DP[i] = {-1, -1};
118
119         if(V[i] != -1) pos.Update(i, V[i]);
120         auto lasts = pos.GetLastTwo();
121
122         pair<int, int> inter, range;
123
124         inter = intersect(make_pair(max(0, i - hi), i - lo),
125                           make_pair(lasts.first == -1 ? 0 : lasts.first, i));
126         inter = intersect(inter, make_pair(0, n + 1));
127         range = ai.Query(1, 0, n, inter.first, inter.second);
128
129         if(range.first != -1)
130         {
131             DP[i] = unite(DP[i], make_pair(range.first+1, range.second+1));
132         }
133
134         if(lasts.first != -1)
135         {
136             inter = intersect(make_pair(max(0, i - hi), i - lo),
137                               make_pair(lasts.second == -1 ?
138                                         0 : lasts.second, lasts.first - 1));
139             inter = intersect(inter, make_pair(0, n + 1));
140             range = ai.Query(1, 0, n, inter.first, inter.second);
141
142             if(range.first <= V[lasts.first] - 1 &&
143                 V[lasts.first] - 1 <= range.second)
144             {
145                 DP[i]=unite(DP[i], make_pair(V[lasts.first], V[lasts.first]));
146             }
147         }
148
149         ai.Update(1, 0, n, i, DP[i]);
150     }
151
152     return DP[n].second;
153 }
154
155 void BackProp(int maxx)
156 {
157     DP[n + 1] = {maxx + 1, maxx + 1};
158
159     ArbInt ai(n + 2);
160     Poset<less<int>> pos(n + 2);
161
162     ai.Update(1, 1, n + 1, n + 1, DP[n + 1]);
163     for(int i = n; i >= 1; --i)
164     {
165         DP[i] = {-1, -1};
166
167         if(V[i] != -1) pos.Update(i, V[i]);
168         auto lasts = pos.GetLastTwo();
169
170         pair<int, int> inter, range;
171
172         inter = intersect(make_pair(i + lo, i + hi),
173                           make_pair(i, lasts.first == -1 ? n + 1 : lasts.first));
174         inter = intersect(inter, make_pair(0, n + 1));
175         range = ai.Query(1, 1, n + 1, inter.first, inter.second);
176
177         if(range.first != -1)
178         {
179             DP[i] = unite(DP[i],
180                           make_pair(max(0, range.first - 1),
181                                     max(0, range.second - 1)));
182         }
183
184         if(lasts.first != -1)
185         {
186             inter = intersect(make_pair(i + lo, i + hi),
187                               make_pair(i, lasts.second == -1 ? n + 1 : lasts.second));
188             inter = intersect(inter, make_pair(0, n + 1));

```

```

189         range = ai.Query(1, 1, n + 1, inter.first, inter.second);
190
191         if(range.first <= V[lasts.first] + 1 &&
192             V[lasts.first] + 1 <= range.second)
193         {
194             DP[i]=unite(DP[i], make_pair(V[lasts.first], V[lasts.first]));
195         }
196     }
197
198     ai.Update(1, 1, n + 1, i, DP[i]);
199 }
200
201
202 int main()
203 {
204     freopen("startrek.in", "r", stdin);
205     freopen("startrek.out", "w", stdout);
206
207     int m;
208     assert(cin >> n >> lo >> hi >> m);
209     assert(1 <= n && n <= 100000);
210     assert(1 <= lo && lo < hi && hi <= n);
211     assert(1 <= m && m <= n);
212
213     memset(V, -1, sizeof(V));
214
215     int old = -1;
216     while(m--)
217     {
218         int pos, val;
219         assert(cin >> pos >> val);
220         assert(1 <= pos && pos <= n);
221         assert(1 <= val && val <= n);
222         assert(pos > old);
223         old = pos;
224         V[pos] = val;
225     }
226
227     int maxx = ForwardProp();
228     BackProp(maxx);
229
230     cout << maxx << endl;
231     int pos = 1;
232     for(int val = 1; val <= maxx; ++val)
233     {
234         int go = -1;
235         for(int nxt = pos + lo; nxt <= pos + hi; ++nxt)
236         {
237             if(nxt > n + 1) break;
238
239             if(DP[nxt].first != -1 && DP[nxt].first <= val + 1) go = nxt;
240
241             if(V[nxt] != -1 && V[nxt] != val)
242                 break;
243         }
244
245         assert(go != -1);
246         while(pos != go)
247         {
248             cout << val << " ";
249             ++pos;
250         }
251     }
252
253     assert(pos == n + 1);
254     cout << endl;
255
256     return 0;
257 }
```

Listing 21.2.7: startrek_100p_2.cpp

```

1 //100 p - prof. Eugen Nodea
2 # include <bits/stdc++.h>
3
```

```

4 # define NMax 2000001
5
6 using namespace std;
7
8 const int INF = 2e9;
9
10 int st[NMax], dr[NMax];
11 int start_st[NMax], start_dr[NMax];
12 int n, m, p,q;
13
14 void construieste( int n )
15 {
16     int N = n;
17     for(int i=0; i<N; ++i)
18     {
19         start_st[i] = INF;
20         start_dr[i] = -INF;
21     }
22
23     start_st[0] = start_dr[0] = 0;
24     for(int i=0; i<N-1; ++i)
25     {
26         start_dr[i] = min(start_dr[i], st[i]);
27         if(start_st[i] + p >= n)      continue;
28         if(start_st[i] > start_dr[i]) return;
29         if(start_dr[i] + q - 1 < dr[i]) return;
30         start_dr[i + 1] = min(n - 1, start_dr[i] + q);
31         start_dr[i + 1] = min(start_dr[i + 1], st[i + 1] + q - 1);
32         start_st[i + 1] = max(dr[i] + 1, start_st[i] + p);
33         start_st[i + 1] = max(start_st[i + 1], dr[i + 1] - q + 1);
34     }
35 }
36
37
38 int main()
39 {
40     freopen("startrek.in", "r", stdin);
41     freopen("startrek.out", "w", stdout);
42
43     scanf("%d %d %d %d", &n, &p, &q, &m);
44     vector <int> a(n);
45     for(int i=1; i<=m; ++i)
46     {
47         int s, t;
48         scanf("%d %d", &s, &t);
49         a[s-1] = t;
50     }
51
52     for(int i=0; i<n; ++i)
53     {
54         st[i] = INF; // indicele sectorul de inceput(stanga) al unui an
55         dr[i] = -INF; // indicele sectorul de sfarsit al anului respectiv
56     }
57
58     int ultima_trs = 0; //determinam ultima transmisie
59     for(int i=0; i<n; ++i)
60     {
61         if(a[i])
62         {
63             a[i]--;
64             ultima_trs = max(ultima_trs, a[i]);
65             st[a[i]] = min(st[a[i]], i);
66             dr[a[i]] = max(dr[a[i]], i);
67         }
68     }
69
70     //incercam sa umplem
71     construieste( n );
72
73     // determin timpul maxim
74     int an = -1;
75     for(int j = ultima_trs; j < n; ++j)
76     {
77         if(start_st[j] > start_dr[j]) continue;
78         if( n - start_st[j] >= p && n - start_dr[j] <= q ) an = j;
79     }

```

```

80
81     printf("%d\n", an + 1);
82
83     int poz = n - 1;
84     for(; an >= 0; --an)
85     {
86         int j = min(poz, start_dr[an]);
87         for(; j >= 0 ; --j)
88         {
89             if(poz - j + 1 >= p && poz - j + 1 <= q) break;
90         }
91         /// umplu
92         for(int i = j; i <= poz; ++i) a[i] = an + 1;
93         poz = j - 1;
94     }
95
96     for(int i=0; i<n; ++i) printf("%d ", a[i]);
97
98     return 0;
99 }
```

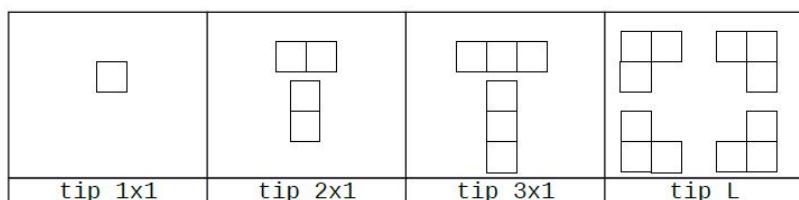
21.2.3 *Rezolvare detaliată

21.3 tris

Problema 3 - tris

100 de puncte

Ninel, fratele mai mic al lui Gigel, a primit de ziua lui un joc tetris în care toate piesele sunt formate din maxim 3 pătrățele. Există 4 tipuri de astfel de piese care, luând în considerare rotirile pieselor, se pot plasa pe un grid în 9 moduri distințe:



Jocul conține din fiecare tip de piesă cel puțin 2 și cel mult 100 de bucăți. El dorește să plaseze toate piesele astfel încât acestea să formeze un ciclu, adică orice pătrățel să aibă exact doi vecini pe cele patru direcții (sus, jos, dreapta, stânga) și zona interioară ciclului să fie conexă pe cele patru direcții.

O mulțime de pătrățele se consideră zonă conexă dacă din oricare pătrățel din mulțime se poate ajunge în oricare alt pătrățel trecând doar prin pătrățele din mulțime pe cele patru direcții.

Cerințe

Cunoscând numărul de piese din fiecare tip, ajutați-l pe Ninel să rezolve problema.

Date de intrare

Fișierul **tris.in** conține pe o singură linie 4 numere naturale $a \ b \ c \ d$, separate prin câte un spațiu, reprezentând numărul de piese de tipul 1×1 , 2×1 , 3×1 respectiv L în această ordine.

Date de ieșire

Fișierul **tris.out** va conține pe prima linie două numere n și m , reprezentând dimensiunile matricei-soluție.

Pe următoarele n linii se vor afla câte m numere naturale din mulțimea $\{0, 1, \dots, a + b + c + d\}$, fiecare element semnificând:

- 0 - dacă pe poziția respectivă nu se găsește niciun element;
- i - dacă pe poziția respectivă este plasată una din cele $a + b + c + d$ piese, identificată cu numărul i .

Pieselete pot fi numerotate în orice ordine cu numere de la 1 la $a + b + c + d$, cu condiția ca acestea să aibă numere distincte.

Evaluare

O soluție se consideră validă dacă și numai dacă se respectă următoarele condiții:

- dimensiunile matricei sunt cel mult egale cu 800x800;
- fiecare celulă ocupată de o piesă are exact 2 vecini;
- zona ocupată de piese formează un ciclu;
- zona interioară ciclului este conexă pe cele patru direcții.

Restricții și precizări

- pentru datele de intrare problema întotdeauna are soluție;
- pentru 30 de puncte $10 \leq a, b, c, d \leq 100$;
- pentru 50 de puncte $5 \leq a, b, c, d \leq 100$;
- pentru 80 de puncte $3 \leq a, b, c, d \leq 100$;
- pentru 100 de puncte $2 \leq a, b, c, d \leq 100$;

Exemple:

tris.in	tris.out	Explicații																																																																		
3 4 3 4	11 6 0 1 2 4 4 4 1 1 0 0 0 3 8 0 0 0 3 3 8 0 0 0 9 0 8 0 0 0 9 9 10 0 0 0 0 13 10 0 0 0 0 11 12 0 0 0 0 11 12 0 0 0 0 14 6 0 0 0 0 7 6 5 5 5 7 7	Avem 3 piese de tip 1x1 Avem 4 piese de tip 2x1 Avem 3 piese de tip 3x1 Avem 4 piese de tip L Matricea-soluție este formată din 11 linii și 6 coloane: <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>0</td><td>1</td><td>2</td><td>4</td><td>4</td><td>4</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>3</td></tr> <tr><td>8</td><td>0</td><td>0</td><td>0</td><td>3</td><td>3</td></tr> <tr><td>8</td><td>0</td><td>0</td><td>0</td><td>9</td><td>0</td></tr> <tr><td>8</td><td>0</td><td>0</td><td>0</td><td>9</td><td>9</td></tr> <tr><td>10</td><td>0</td><td>0</td><td>0</td><td>0</td><td>13</td></tr> <tr><td>10</td><td>0</td><td>0</td><td>0</td><td>0</td><td>11</td></tr> <tr><td>12</td><td>0</td><td>0</td><td>0</td><td>0</td><td>11</td></tr> <tr><td>12</td><td>0</td><td>0</td><td>0</td><td>0</td><td>14</td></tr> <tr><td>6</td><td>0</td><td>0</td><td>0</td><td>0</td><td>7</td></tr> <tr><td>6</td><td>5</td><td>5</td><td>5</td><td>7</td><td>7</td></tr> </table>	0	1	2	4	4	4	1	1	0	0	0	3	8	0	0	0	3	3	8	0	0	0	9	0	8	0	0	0	9	9	10	0	0	0	0	13	10	0	0	0	0	11	12	0	0	0	0	11	12	0	0	0	0	14	6	0	0	0	0	7	6	5	5	5	7	7
0	1	2	4	4	4																																																															
1	1	0	0	0	3																																																															
8	0	0	0	3	3																																																															
8	0	0	0	9	0																																																															
8	0	0	0	9	9																																																															
10	0	0	0	0	13																																																															
10	0	0	0	0	11																																																															
12	0	0	0	0	11																																																															
12	0	0	0	0	14																																																															
6	0	0	0	0	7																																																															
6	5	5	5	7	7																																																															

Observație:

Următoarea matrice nu formează soluție din multiple motive:

- există pătrățele care nu au exact doi vecini (vezi piesele 6, 9, 13 și respectiv 15);
- zona interioară ciclului nu este conexă pe cele patru direcții. Există două zone interioare conexe cu 3 pătrățele, respectiv 13 pătrățele.

0	1	2	4	4	4
1	1	0	0	0	3
8	0	5	5	3	3
8	0	5	0	0	0
8	0	6	6	6	7
10	0	15	0	0	7
10	0	0	0	0	9
11	0	0	0	9	9
11	11	14	12	13	13

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

Memorie: total **64 MB**

Dimensiune maximă a sursei: **20 KB**

21.3.1 Indicații de rezolvare

Autor: Lucian Bicsi, Eugenie-Daniel Posdărăscu

Soluție: Eugenie-Daniel Posdărăscu

Un criteriu foarte important pentru rezolvarea problemei este sa observam ca problema admite solutie doar daca numarul total de patratele este par (formal, $a + c + d$ trebuie sa fie par).

Din moment ce ni se garanteaza ca exista mereu solutie, aceasta proprietate va fi mereu satisfacuta.

Fiind o problema de constructie, solutia poate sa apara in diferite forme care sa functioneze.

O posibila abordare este aceea de a construi un "tunel" foarte lung, marginile tunelului afandu-se la distanta fix 2 (o casuta intre). Tunelul se imparte in 2 categorii:

- Sectiunea formata din pisele de tip L .
- Sectiunea formata din piesele 1×1 , 1×2 si 1×3 .

Sectiunea pieselor de tip L va fi completate in felul urmator:

...

77088

70008 // 0urile reprezinta casute goale iar zonele cu 1, 2, 3, ... reprezinta piesele

55066

05060

33044

30004

10002

11022

Mai exact, formam 2 siruri serpuite pe stanga si pe dreapta, cu o singura coloana nula intre. Capatul de sus va ramane astfel liber, acesta avand 4 stari posibile in functie de restul numarului de piese de tip L la 4.

In continuarea tunelului, completam cu eventualele piese suplimentare de tip 1×1 , 1×2 si 1×3 de care nu o sa avem nevoie pentru completarea capetelor. Acestea vor arata in felul urmator:

01010

02020

02020

03030

03030

03030

...

Raman de tratat inchiderea celor 2 capete. Din acest punct, solutia admite atat o solutie analitica (bazata pe tratarea mai multor cazuri in capete), cat si o solutie bazata pe brute-forsarea capetelor.

Cu cat avem mai putine piese, cu atat avem mai putina libertate de constructie, ca urmare, atat brute-forsarea cat si tratarea analitica a capetelor se complica (foarte important sa aveti in vedere proprietatea $a + c + d$ par).

Atentie: Tunelul se completeaza dupa construirea capetelor, intrucat este necesar sa ne rezervam suficiente piese pentru capete, restul fiind folosite in cantitarea oricat de mare pentru tunel.

21.3.2 Cod sursă

Listing 21.3.1: tris_94p.cpp

```

1 // Solutie TRIS 94 puncte
2 /// Autor: Lucian Bicsi
3
4 #include <bits/stdc++.h>
5
6 using namespace std;
7
8 // [1, 2, 3, 3*]
9 vector<vector<int>> caps = {
10     {1, 0, 1, 0},

```

```

11     {0, 2, 0, 0},
12     // {2, 1, 0, 0},
13     // {0, 0, 0, 2},
14     // {1, 1, 0, 1},
15     // {1, 1, 1, 0},
16     // {0, 3, 0, 0}
17 };
18
19 const int kSize = 800;
20
21 struct Solver
22 {
23     vector<int> split;
24     vector<vector<int>> ret;
25
26     Solver() : split(4, 0), ret(kSize, vector<int>(kSize, 0)) {}
27
28     void BuildSolution(vector<int> pieces, int c1, int c2)
29     {
30         int up = 3, dw = 0;
31         int col = 0;
32
33         int piece = 1;
34         auto alloc = [&]()
35         {
36             return piece++;
37         };
38
39         auto putCap = [&](int cid)
40         {
41             int p1 = alloc(), p2 = alloc();
42             if(cid == 1)
43             {
44                 ret[dw][col] = ret[dw + 1][col] = p1;
45                 ret[dw + 2][col] = ret[dw + 3][col] = p2;
46             }
47             else
48             {
49                 ret[dw][col] = p1;
50                 ret[dw + 1][col] = ret[dw + 2][col] = ret[dw + 3][col] = p2;
51             }
52             ++col;
53         };
54
55         auto putPiece = [&](int pid)
56         {
57             #define small_macro() ret[dw][col] = p1, ret[up][col] = p2, ++col
58
59             int p1 = alloc(), p2 = alloc();
60             pieces[pid] -= 2;
61
62             small_macro();
63             if(split[pid]) split[pid] = 0, p1 = alloc();
64             if(pid == 0) return;
65             small_macro();
66             if(pid == 1) return;
67             if(pid == 3) ++up, ++dw, --col;
68             small_macro();
69
70             #undef small_macro
71         };
72
73         putCap(c1);
74         for(int i = 3; i >= 0; --i)
75             while(pieces[i])
76                 putPiece(i);
77         putCap(c2);
78     }
79
80     bool TrySolve(vector<int> pieces, int c1, int c2)
81     {
82         for(int i = 0; i < 4; ++i)
83         {
84             pieces[i] -= caps[c1][i] + caps[c2][i];
85             if(pieces[i] < 0)
86                 return false;

```

```

87         }
88
89         if(pieces[3] % 2) ++pieces[3], --pieces[1], --pieces[0], ++split[3];
90         if(pieces[2] % 2) ++pieces[2], --pieces[1], --pieces[0], ++split[2];
91         if(pieces[1] % 2) ++pieces[1], --pieces[0], --pieces[0], ++split[1];
92
93         if(pieces[0] % 2 || pieces[0]==0 && pieces[1]==0 && pieces[2]==0)
94             return false;
95         for(int i = 0; i < 4; ++i)
96         {
97             if(pieces[i] < 0)
98                 return false;
99         }
100
101     // Found solution
102     BuildSolution(pieces, c1, c2);
103     return true;
104 }
105
106 static vector<vector<int>> Solve(vector<int> pieces)
107 {
108     int cs = caps.size();
109     for(int i = 0; i < cs; ++i)
110         for(int j = i; j < cs; ++j)
111         {
112             Solver solver;
113             if(solver.TrySolve(pieces, i, j))
114                 return solver.ret;
115         }
116     return {};
117 }
118 };
119
120 void Output(ostream &out, vector<vector<int>> mat)
121 {
122     assert(!mat.empty());
123     out << mat.size() << " " << mat[0].size() << endl;
124     for(auto row : mat)
125     {
126         for(auto cell : row)
127             out << cell << " ";
128         out << endl;
129     }
130 }
131
132 int main()
133 {
134     ifstream fin("tris.in");
135     ofstream fout("tris.out");
136
137     vector<int> freq(4);
138     for(auto &x : freq)
139         fin >> x;
140
141     Output(fout, Solver::Solve(freq));
142
143     return 0;
144 }
```

Listing 21.3.2: tris_100p.cpp

```

1 #include<stdio.h>
2 #include<algorithm>
3 using namespace std;
4
5 #define NMAX 805
6
7 int tris1, tris2, tris3, trisL;
8 int rest1, rest2, rest3,n,m,nr;
9 int answer[NMAX][NMAX];
10
11 void puneL(int px, int py, int dx, int dy)
12 {
13     answer[px][py] = ++nr;
14     answer[px + dx][py] = nr;
```

```

15     answer[px][py + dy] = nr;
16     n = max(n, px + dx);
17     n = max(n, px);
18 }
19
20 void puneS1(int px, int py)
21 {
22     answer[px][py] = ++nr;
23     n = max(n, px);
24 }
25
26 void puneS2(int px, int py, int type)
27 {
28     answer[px][py] = ++nr;
29     if(type)
30     {
31         answer[px + 1][py] = nr;
32         n = max(n, px + 1);
33     }
34     else
35     {
36         answer[px][py + 1] = nr;
37         n = max(n, px);
38     }
39 }
40
41 void puneS3(int px, int py, int type)
42 {
43     answer[px][py] = ++nr;
44     if(type)
45     {
46         answer[px + 1][py] = answer[px + 2][py] = nr;
47         n = max(n, px + 2);
48     }
49     else
50     {
51         answer[px][py + 1] = answer[px][py + 2] = nr;
52         n = max(n, px);
53     }
54 }
55
56 int putLs(int line)
57 {
58     int cLine = line;
59     for(int i = 2; i <= trisL; i += 4)
60     { // *** *
61         puneL(line, 2, 1, -1); // *
62         puneL(line, 4, 1, 1); // *
63         line += 4;
64     }
65     line = cLine + 2;
66     for(int i = 4; i <= trisL; i += 4)
67     { // ** **
68         puneL(line, 1, 1, 1); // *
69         puneL(line, 5, 1, -1); // *
70         line += 4;
71     }
72     return trisL + cLine;
73 }
74
75 int puts3s(int line, int tris3)
76 {
77     for(int i = 2; i <= tris3; i += 2)
78     { // **
79         puneS3(line, 2, 1); // **
80         puneS3(line, 4, 1); // **
81         line += 3;
82     }
83     return line;
84 }
85
86 int puts2s(int line, int tris2)
87 {
88     for(int i = 2; i <= tris2; i += 2)
89     { // **
90         puneS2(line, 2, 1); // **

```

```

91         punoS2(line, 4, 1);
92         line += 2;
93     }
94     return line;
95 }
96
97 int puts1s(int line, int tris1)
98 {
99     for(int i = 2; i <= tris1; i += 2)
100    {
101        // *
102        punoS1(line, 2);
103        punoS1(line, 4);
104        line++;
105    }
106    return line;
107 }
108 void transformCase()
109 {
110     rest1 = (tris1 & 1);
111     rest2 = (tris2 & 1);
112     rest3 = (tris3 & 1);
113 }
114
115 int main ()
116 {
117     freopen("tris.in","r",stdin);
118     freopen("tris.out","w",stdout);
119
120     scanf("%d%d%d%d",&tris1,&tris2,&tris3,&trisL);
121     m = 5;
122
123     if(trisL % 4 == 3)
124    {
125         punoS3(1, 1, 0); tris3--; // 0 orizontal 1 vertical
126         punoL(1,5,+1,-1); trisL--; //      11122
127         punoS1(2,1); tris1--; //      30002
128         int line = putLs(3);
129         transformCase();
130
131         line = puts3s(line, tris3);
132         line = puts2s(line, tris2);
133         line = puts1s(line, tris1);
134
135         if(rest1 && !rest2 && !rest3)
136         { // 1 0 0
137             punoS1(line - 1, 3);
138         }
139         else if(!rest1 && !rest2 && rest3)
140         { // 0 0 1
141             punoS3(line, 2, 0);
142         }
143         else if(rest1 && rest2 && !rest3)
144         { // 1 1 0
145             punoS2(line,2,0);
146             punoS1(line,4);
147         }
148         else
149         { // 0 1 1
150             nr--;
151             punoS3(line - 1, 4, 1);
152             punoS2(line, 2, 1);
153             punoS1(line + 1, 3);
154         }
155     }
156     else if(trisL % 4 == 2)
157    {
158         int line = 1;
159         transformCase();
160         if(!rest1 && !rest2 && !rest3)
161         {
162             punoS3(1,1,0); tris3--;
163             punoS1(1,4); tris1 -= 2;
164             punoS1(1,5);
165             punoS2(2,1,1); tris2 -= 2;
166             punoS2(2,5,1);

```

```

167         line += 3;
168     }
169     else if(!rest1 && rest2 && !rest3)
170     {
171         puneS3(1,1,0); tris3--;
172         puneS2(1,4,0); tris2--;
173         puneS1(2,1); tris1 -= 2;
174         puneS1(2,5);
175         line += 2;
176     }
177     else if(rest1 && !rest2 && rest3)
178     {
179         puneS2(1,1,0);
180         puneS1(1,3); tris1--;
181         puneS2(1,4,0); tris2 -= 2;
182         puneS3(2,1,1); tris3 -= 2;
183         puneS3(2,5,1);
184         line += 4;
185     }
186     else
187     {
188         puneS2(1,1,0); tris2 -= 3;
189         puneS1(1,3);
190         puneS2(1,4,0);
191         puneS2(2,1,1);
192         puneS1(2,5); tris1 -= 3;
193         puneS1(3,5);
194         line += 3;
195     }
196
197 //printf("a intrat %d", line);
198 line = putLs(line);
199 //printf("a iesit %d\n", line);
200 line = putS3s(line, tris3);
201 line = putS2s(line, tris2);
202 line = putS1s(line, tris1);
203 puneS3(line, 2, 0);
204 }
205 else if(trisL % 4 == 1)
206 {
207     transformCase();
208     if(!rest1 && !rest2 && rest3)
209     {
210         puneS1(1,5);
211         puneS1(2,5); tris1 -= 2;
212     }
213     else
214     {
215         puneS2(1,5,1);
216         tris2--;
217     }
218
219     puneS3(1,2,0); tris3 -= 2;
220     puneS3(2,2,1);
221     puneL(3,4,1,1); trisL -= 3;
222     puneL(5,1,1,1);
223     puneL(5,5,1,-1);
224     int line = putLs(7);
225     line = putS3s(line, tris3);
226     line = putS2s(line, tris2);
227     line = putS1s(line, tris1);
228
229     if(rest1 && !rest2 && !rest3)
230     {
231         puneS2(line,2,0);
232         puneS1(line,4);
233     }
234     else if(rest1 && rest2 && !rest3)
235     {
236         puneS1(line - 1, 3);
237     }
238     else
239     {
240         puneS3(line, 2, 0);
241     }
242 }
```

```

243     else if(trisL % 4 == 0)
244     {
245         puneL(1, 1, 1, 1);
246         puneL(1, 5, 1,-1);
247         trisL -= 2;
248         int line = putLs(3);
249         transformCase();
250
251         line = putS2s(line, tris2);
252
253         if(!rest1 && !rest2 && !rest3)
254         {
255             line = putS3s(line, tris3);
256             line = putS1s(line, tris1 - 2);
257
258             puneS1(1,3);
259             puneS1(line - 1, 3);
260         }
261         else if(!rest1 && rest2 && !rest3)
262         {
263             line = putS3s(line, tris3 - 2);
264             line = putS1s(line, tris1 - 2);
265
266             puneS2(line,2,1);
267             puneS1(line + 2, 2);
268             puneS3(line,4,1);
269             puneS1(1,3);
270             puneS3(line + 3, 2, 0);
271         }
272         else if(rest1 && !rest2 && rest3)
273         {
274             line = putS3s(line, tris3);
275             line = putS1s(line, tris1);
276
277             puneS1(1,3);
278             puneS3(line,2,0);
279         }
280         else
281         {
282             line = putS3s(line, tris3);
283             line = putS1s(line, tris1 - 2);
284
285             puneS2(line,2,1);
286             puneS1(line + 2, 2);
287             puneS3(line,4,1);
288             puneS1(1,3);
289             puneS1(line + 2, 3);
290         }
291     }
292
293     printf("%d %d\n",n,m);
294     for(int i = 1; i <= n; i++, printf("\n"))
295         for(int j = 1; j <= m; j++)
296             printf("%d ", answer[i][j]);
297
298     return 0;
299 }
```

21.3.3 *Rezolvare detaliată

21.4 bvarcolaci

Problema 4 - bvarcolaci

100 de puncte

Se pare că toate ideile mari care sau înscrîptionat pe vecie în inima Umanității au trebuit să se înfățișeze lumii întâi cu măști monstruoase și terifinante.

Friedrich Nietzsche

Se dă un vector cu N elemente. Să se precizeze câte subsecvențe ale acestuia admit element majoritar.

Cerințe

Date de intrare

Fisierul **bvarcolaci.in** conține pe prima linie numărul natural N , iar pe a doua linie cele N elemente ale vectorului, separate prin câte un spațiu.

Date de ieșire

Fisierul **bvarcolaci.out** va conține pe prima linie numărul subsecvențelor care admit element majoritar.

Restricții și precizări

- $1 \leq N \leq 250\,000$
- Toate valorile din vector sunt cuprinse între 1 și N
- O subsecvență este un subșir de elemente care apar pe poziții consecutive în sirul inițial. Subsecvențele sunt definite unic de indicii primului și ultimului element din subsecvență în sirul initial.
- O valoare este element majoritar al unui vector cu K elemente dacă apare de cel puțin $[K/2] + 1$ ori în vector, unde prin $[x]$ am notat partea întreagă a lui x .

Exemple:

bvarcolaci.in	bvarcolaci.out	Explicații
6 1 2 1 2 3 2	10	<p>Fiecare subsecvență de câte un element are element majoritar.</p> <p>Celelalte subsecvențe sunt:</p> <p style="text-align: center;"> $\underline{1} \ 2 \ 1 \ 2 \ 3 \ 2$ $\underline{1} \ 2 \ 1 \ 2 \ 3 \ 2$ $1 \ \underline{2} \ 1 \ 2 \ 3 \ 2$ $1 \ 2 \ 1 \ \underline{2} \ 3 \ 2$ $1 \ 2 \ 1 \ 2 \ \underline{3} \ 2$ </p>

Timp maxim de executare/test: **3.0** secunde

Memorie: total **256 MB**

Dimensiune maximă a sursei: **20 KB**

21.4.1 Indicații de rezolvare

Eugenie-Daniel Posdărăscu

Soluție $O(N * N)$:

Fixăm Left și Right, capătul stânga respectiv dreapta al subsecvenței. Este suficient să verificăm doar cel mai frecvent element al subsecvenței dacă are sau nu size / 2 + 1 elemente. Pe măsură ce variem capătul dreapta, actualizăm frecvența elementului nou inserat într-un vector de frecvență.

Cel mai frecvent element va fi ori precedentul cel mai frecvent element, ori elementul nou inserat (ne decidem în funcție de valorile din vectorul de frecvențe).

Acestă soluție obține între 20 și 30 de puncte.

Soluție $O(N * \sigma)$:

Sigma reprezintă numărul de elemente distincte din vector. Fixăm X, un element distinct din vectorul initial (vectorul V). Acum dorim să aflăm în $O(N)$ câte subsecvențe există care îl au pe X ca element majoritar.

Primul pas este să construim un vector auxiliar în felul următor: la poziția P, dacă avem valoarea X în V, punem +1, altfel -1. Facem *sume parțiale* pe acest vector și fixăm capătul dreapta Right al subsecvenței. Rămâne să selectăm un capăt stânga Left astfel încât elementul X să fie majoritar în intervalul [Left, Right].

Acest lucru este echivalent cu a selecta un capăt stânga Left astfel încât suma elementelor în vectorul auxiliar pe intervalul [Left, Right] să fie strict pozitivă.

Cu ajutorul *sumelor parțiale*, se pot număra ușor toate capetele Left care respectă această proprietate. O implementare ușoară poate să fie realizată cu *AIB*, dar se poate scăpa și de log dacă aveți în vedere că sumele cresc/scad cu 1.

Soluție $O(N * \text{sqrt}(N))$:

Impărțim numerele în 2 categorii:

- Elementele care au frecvență mai mică de sqrt . Pentru aceste elemente observăm că nu pot să fie elemente majoritare într-o subsecvență de lungime mai mare ca $2 * \text{sqrt}$. Aceste elemente pot fi tratate *brute-force*.
- Elemente care au frecvență mai mare de sqrt . Pentru aceste elemente observăm că sunt maxim $\text{sqrt}(N)$ astfel de valori distințe. Ca urmare, putem pentru fiecare din ele să aplicăm algoritmul prezentat mai sus în soluția de $O(N * \sigma)$.

Există multe soluții alternative în $O(N * \text{sqrt}(N))$. O altă variantă se bazează pe *algoritmul lui Mo* în care facem *brute-force* secvențele mici, iar pentru un bloc de bucăți de radical observăm că este suficient să ținem doar cele mai frecvente 2-3 elemente. Această soluție este în schimb mai tehnică și mai neintuitivă.

Soluțiile în această complexitate obțin între 70-90 de puncte, dar nu este exclus să se obțina și 100 de puncte.

Soluție $O(N * \log(N) * \log(N))$:

Aplicăm *Divide-Et-Impera*. Vrem să aflăm numărul de subsecvențe bune pe un interval [Left, Right].

Impărțim intervalul în 2 și rezolvăm recursiv partea stângă și partea dreaptă.

Rămâne de văzut cum combinăm cele 2 zone.

Observația de bază este că sunt maxim $\log(N)$ candidați pentru un posibil element majoritar. Pentru fiecare candidat, determinăm câte subsecvențe îl au ca și element majoritar în $O(N)$ folosind algoritmul prezentat în soluțiile anterioare. Întrebarea rămâne de ce sunt maxim $\log(N)$ candidați și cum îi determinăm.

O secvență [Left, Right] care admite un element majoritar X are următoarea proprietate: X este element majoritar ori în secvența [Left, Mid], ori în secvența [Mid + 1, Right]. Deci putem să analizăm independent fiecare jumătate și să extragem posibilele elemente majoritare.

Să zicem că ne uităm la jumătatea stângă (capătul dreaptă va fi mereu Mid).

Dacă ar fi să completăm element cu element pornind din capătul Mid, am pune 1 element ca să obținem primul candidat, 2 elemente ca să obținem al doilea candidat, 4 elemente ca să obținem al 3-lea candidat, etc. În final avem $\log(N)$ candidați.

Această soluție obține 100 de puncte.

Soluție $O(N * \log(N))$:

Pornind de la soluția în $O(N * \sigma)$, ideea acestei soluții se bazează pe următorul principiu.

Pentru un element X fixat, am dori să aflăm numărul de subsecvențe care îl conțin ca element majoritar în $O(\text{numărul de apariții a lui } X \text{ în vector}) * \log(N)$ în loc de $O(N)$.

Complexitatea astfel se amortizează la $O(N * \log(N))$ pentru toate elementele.

Algoritmul cu $+1,-1$ poate să fie adaptat cu ajutorul unor structuri de date (*arbori de intervale* spre exemplu), dar trebuie să aveți grijă la foarte multe detalii (în principiu, trebuie avut în vedere faptul că o secvență nu incepe/termină într-un capăt cu $+1$). Dezvoltarea soluției rămâne temă pentru acasă întrucât detalii sunt mai complicate.

Această soluție obține 100 de puncte.

21.4.2 Cod sursă

Listing 21.4.1: bvarcolaci_80p.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int NMAX = 500005;
6
7 int n, sq, a[NMAX], fr[NMAX];
8 int AIB[NMAX], sum[NMAX];
9 vector<int> v[NMAX];

```

```

10 long long sol;
11
12 int zeros(int x)
13 {
14     return x^(x-(x-1));
15 }
16
17 void Update(int poz, int val)
18 {
19     for (int i = poz; i < NMAX; i += zeros(i)) AIB[i] += val;
20 }
21
22 int Query(int poz)
23 {
24     int i, rez = 0;
25     for (int i = poz; i >= 1; i -= zeros(i)) rez += AIB[i];
26
27     return rez;
28 }
29
30 int main()
31 {
32     freopen("bvarcolaci.in", "r", stdin);
33     freopen("bvarcolaci.out", "w", stdout);
34
35     scanf("%d", &n); sq = sqrt(n);
36     for (int i = 1; i <= n; i++)
37     {
38         scanf("%d", &a[i]);
39         v[a[i]].push_back(i);
40     }
41
42     //brute small
43     for (int i = 1; i <= n; i++)
44     {
45         int mx = 0;
46         for (int j = i; j >= 1 && (i - j + 1) / 2 + 1 <= sq; j--)
47         {
48             fr[a[j]]++;
49             mx = max(mx, fr[a[j]]);
50             if (mx >= (i - j + 1) / 2 + 1) sol++;
51         }
52
53         //clean
54         for (int j = i; j >= 1 && (i - j + 1) / 2 + 1 <= sq; j--)
55             fr[a[j]] = 0;
56     }
57
58     int ADD = n;
59     //AIB for big
60     for (int i = 1; i < NMAX; i++)
61     {
62         if (v[i].size() > sq)
63         {
64             for (auto it : v[i])
65                 sum[it]++;
66             for (int j = 1; j <= n; j++)
67             {
68                 if (sum[j] == 0)
69                     sum[j] = -1;
70                 sum[j] += sum[j - 1];
71             }
72
73             int last = 0;
74             for (int j = 1; j <= n; j++)
75             {
76                 while ((j - last) / 2 + 1 > sq) //good for Update
77                 {
78                     Update(sum[last] + ADD, 1);
79                     last++;
80                 }
81             }
82             sol += Query(sum[j] - 1 + ADD);
83         }
84         //clean
85         for (int j = 0; j < last; j++)

```

```

86             Update(sum[j] + ADD, -1);
87         for (int j = 1; j <= n; j++)
88             sum[j] = 0;
89     }
90
91     printf("%lld\n", sol);
92     return 0;
93 }
```

Listing 21.4.2: bvarcolaci_100p_1.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int NMAX = 250005;
6
7 int n, a[NMAX];
8 long long sol;
9
10 void Go(int st, int dr)
11 {
12     if (st == dr) {sol++; return;}
13
14     int mij = (st + dr) / 2;
15
16     Go(st, mij);
17     Go(mij + 1, dr);
18
19     //make left suffixes
20     map<int, int> M, fr;
21     for (int i = mij; i >= st; i--)
22     {
23         fr[a[i]]++;
24         if (fr[a[i]] >= (mij - i + 1) / 2 + 1) M[a[i]] = 1;
25     }
26
27     fr.clear();
28     for (int i = mij + 1; i <= dr; i++)
29     {
30         fr[a[i]]++;
31         if (fr[a[i]] >= (i - mij) / 2 + 1) M[a[i]] = 1;
32     }
33
34     //for every maj make sums
35     int ADD = dr - st + 1, lim = 2 * (dr - st + 5);
36     int* freq = (int*)malloc(lim * sizeof(int)); //frequencies for left sums
37     for (auto it : M)
38     {
39         memset(freq, 0, lim * sizeof(int));
40         int act = 0;
41         freq[0 + ADD]++;
42         for (int i = st; i <= mij; i++)
43         {
44             if (a[i] == it.first) act++;
45             else act--;
46             if (i < mij) freq[act + ADD]++;
47         }
48
49         for (int i = 1; i < lim; i++) freq[i] += freq[i - 1];
50
51         //go through right sums, act remains
52         for (int i = mij + 1; i <= dr; i++)
53         {
54             if (a[i] == it.first) act++;
55             else act--;
56
57             sol += freq[act + ADD - 1];
58         }
59     }
60     free(freq);
61 }
62
63 int main()
```

```

65  {
66      freopen("bvarcolaci.in", "rb", stdin);
67      freopen("bvarcolaci.out", "wb", stdout);
68      //cin.sync_with_stdio(false);
69
70      cin >> n;
71      for (int i = 1; i <= n; i++) cin >> a[i];
72
73      Go(1, n);
74
75      cout << sol << "\n";
76      return 0;
77 }

```

Listing 21.4.3: bvarcolaci_100p_2.cpp

```

1 #include <cassert>
2 #include <fstream>
3 #include <cstring>
4 #include <queue>
5 #include <algorithm>
6 #include <bitset>
7 #include <ctime>
8 #include <set>
9 #include <cmath>
10 #include <random>
11 #include <iomanip>
12 #include <map>
13 #include <stack>
14 #include <vector>
15 #include <bitset>
16 #include <iostream>
17
18 using namespace std;
19
20 #define FOR(i, a, n) for (int i = a; i <= n; ++i)
21 #define ROF(i, n, a) for (int i = n; i >= a; i--)
22 #define FIT(i, v) for (auto &i : v)
23 #define pb push_back
24 #define mp make_pair
25 #define mt make_tuple
26 #define all(x) (x).begin(), (x).end()
27 #define fi first
28 #define se second
29 #define sz(x) ((int)(x).size())
30 #define log log2
31
32 typedef long long ll;
33 typedef pair<int,int> pii;
34
35 const int mod = 1000000007;
36
37 ll powmod(ll a, ll b)
38 {
39     ll res=1;
40     a %= mod;
41     assert(b >= 0);
42     for(; b; b >= 1)
43     {
44         if (b & 1) res = res * a % mod;
45         a = a * a % mod;
46     }
47
48     return res;
49 }
50
51 ifstream f("bvarcolaci.in");
52 ofstream g("bvarcolaci.out");
53
54 const int DIM = 8000100;
55 const int maxN = 1000000;
56 const int N = maxN + 100;
57
58 ll sol, prog[N << 2];
59 vector<int> poz[N];

```

```

60  int sum[N << 2], lef[N << 2], rig[N << 2], nods, n, x, buf;
61  char b[DIM];
62
63  void gInt(int &x)
64  {
65      x = 0;
66      while (b[buf] > '9' || b[buf] < '0')
67          ++buf;
68
69      while (b[buf] >= '0' && b[buf] <= '9')
70          x = x * 10 + b[buf++]- '0';
71  }
72
73  void push(int nod, int st, int dr)
74  {
75      int mij = (st + dr) >> 1;
76
77      if (!lef[nod])
78          lef[nod] = ++nods;
79
80      if (!rig[nod])
81          rig[nod] = ++nods;
82
83      if (sum[nod] != sum[lef[nod]] + sum[rig[nod]])
84      {
85          int dif = (sum[nod] - (sum[lef[nod]] + sum[rig[nod]])) / (dr-st+1);
86          sum[lef[nod]] += (mij - st + 1) * dif;
87          sum[rig[nod]] += (dr - mij) * dif;
88          prog[lef[nod]] += 1LL * (mij - st + 1) * (mij - st + 2) / 2 * dif;
89          prog[rig[nod]] += 1LL * (dr - mij) * (dr - mij + 1) / 2 * dif;
90      }
91  }
92
93  void upd(int nod, int st, int dr, int l, int r)
94  {
95      if (st >= l && dr <= r)
96      {
97          sum[nod] += (dr - st + 1);
98          prog[nod] += 1LL * (dr - st + 1) * (dr - st + 2) / 2;
99          return;
100     }
101
102     push(nod, st, dr);
103     int mij = (st + dr) >> 1;
104     if (l <= mij)
105         upd(lef[nod], st, mij, l, r);
106
107     if (r > mij)
108         upd(rig[nod], mij + 1, dr, l, r);
109
110     sum[nod] = sum[lef[nod]] + sum[rig[nod]];
111     prog[nod] = prog[lef[nod]] + prog[rig[nod]] + 1LL*sum[lef[nod]]*(dr-mij);
112 }
113
114  void qryp(int nod, int st, int dr, int l, int r)
115  {
116      if (st >= l && dr <= r)
117      {
118          sol += prog[nod] + 1LL * sum[nod] * (r - dr);
119          return;
120      }
121
122     push(nod, st, dr);
123     int mij = (st + dr) >> 1;
124     if (l <= mij)
125         qryp(lef[nod], st, mij, l, r);
126
127     if (r > mij)
128         qryp(rig[nod], mij + 1, dr, l, r);
129 }
130
131  void qrys(int nod, int st, int dr, int l, int r, int m)
132  {
133      if (st >= l && dr <= r)
134      {
135          sol += 1LL * sum[nod] * m;

```

```

136         return;
137     }
138
139     push(nod, st, dr);
140     int mij = (st + dr) >> 1;
141     if (l <= mij)
142         qrys(lef[nod], st, mij, l, r, m);
143
144     if (r > mij)
145         qrys(rig[nod], mij + 1, dr, l, r, m);
146 }
147
148 int main()
149 {
150     f.get(b, DIM, EOF);
151     gInt(n);
152     // assert(1 <= n && n <= maxN);
153
154     FOR(i,1,n)
155         poz[i].pb(0);
156
157     FOR(i,1,n)
158     {
159         gInt(x);
160         // assert(1 <= x && x <= n);
161         poz[x].pb(i);
162     }
163
164     FOR(i,1,n)
165         poz[i].pb(n + 1);
166
167     FOR(i,1,n)
168     {
169         while(nods)
170         {
171             lef[nods] = rig[nods] = sum[nods] = prog[nods] = 0;
172             --nods;
173         }
174
175         ++nods;
176         for (int j = 0; j < poz[i].size() - 1; ++j)
177         {
178             int cs = j * 2 - poz[i][j];
179             int dist = poz[i][j + 1] - poz[i][j];
180             if (j)
181             {
182                 qryp(1, -n, n, cs - 1 - dist + 1, cs - 1);
183                 if (cs - 1 - dist - 1 >= -n)
184                     qrys(1, -n, n, -n, cs - 1 - dist, dist);
185             }
186
187             if (!j)
188                 upd(1, -n, n, -dist + 1, 0);
189             else
190                 upd(1, -n, n, cs - dist + 1, cs);
191         }
192     }
193
194     g << sol;
195     return 0;
196 }
```

21.4.3 *Rezolvare detaliată

21.5 minarea

Problema 5 - minarea

100 de puncte

O furnică se deplasează în sistemul de coordinate xOy . Furnica pleacă din origine. Dacă furnica se găsește la coordonatele (x, y) atunci ea se va deplasa în linie dreaptă în unul din următoarele sase puncte:

1. $(x+1, y+1)$
2. $(x+2, y+2)$
3. $(x+3, y+3)$
4. $(x+1, y-1)$
5. $(x+2, y-2)$
6. $(x+3, y-3)$

Furnica se va deplasa astfel încât în niciun moment al deplasării să nu se găsească într-un punct de coordonată y negativă. **La final furnica va ajunge pe axa Ox .**

Cerințe

Cunoscând numărul de deplasări de fiecare din cele şase tipuri să se aleagă o ordine în care acestea pot fi efectuate astfel încât suprafața delimitată inferior de axa Ox și superior de traseul furnicii să aibă aria minimă.

Date de intrare

Fișierul de intrare **minarea.in** va conține 6 numere naturale a, b, c, d, e, f separate prin câte un spațiu, reprezentând numărul de deplasări de tip 1, 2, 3, 4, 5 respectiv 6.

Date de ieșire

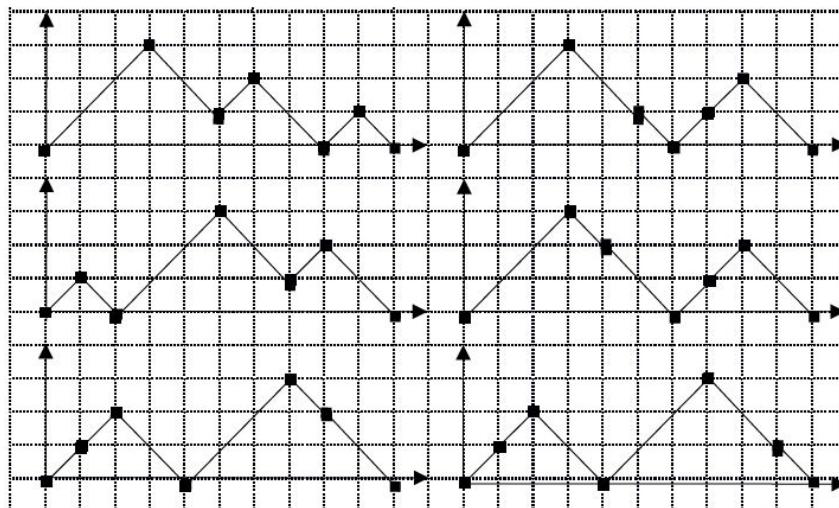
Fișierul de ieșire **minarea.out** va conține un singur număr reprezentând aria minimă.

Restricții și precizări

- $1 \leq a, b, c, d, e, f \leq 1\,000\,000\,000$
- $a + 2b + 3c = d + 2e + 3f$
- pentru teste în valoare de 10 puncte $c = f = 0$

Exemple:

minarea.in	minarea.out	Explicații
2 0 1 1 2 0	13	Aria minimă este 13. Această arie poate fi obținută în mai multe moduri. În figura de mai jos sunt descrise toate cele şase moduri în care traseul furnicii și axa Ox ar putea delimita o suprafață de arie 13.
219 221 5 108 47 158	1760	Aria minimă este 1760.



Timp maxim de executare/test: **0.3** secunde

Memorie: total **256 MB**

Dimensiune maximă a sursei: **20 KB**

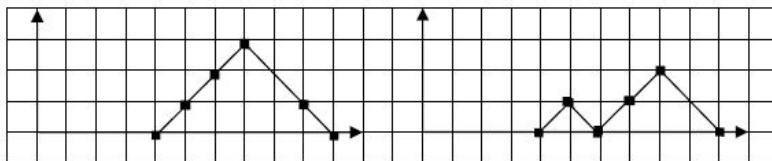
21.5.1 Indicații de rezolvare

prof. Adrian Panaete - C.N. "A. T. Laurian" Botosani

Mihai Ciucu - CS Academy

Sunt două soluții posibile: matematic sau greedy.

Observația de baza a problemei este că într-o soluție nu putem să avem un drum format din segmente care pleacă de la un nivel $X > 0$ și care la final ajunge tot la nivelul X . Orice drum de genul acesta poate fi tăiat dintr-o soluție validă și adăugat la final, scăzând aria totală cu $X^*lungimea_drumului$. O simplificare de genul se poate observa în figura de mai jos, unde $X=1$ (drumul format din segmente 2, 3 și 4):



Se poate arăta atunci că orice soluție validă este compusă dintr-un număr finit de figuri care pleacă de pe Ox și ajung tot pe Ox . Figurile atunci pot fi luate într-o ordine a preferințelor:

- $\{+3, -3\}$
- $\{+3, -2, -1\}$
- $\{+3, -1, -1, -1\}$
- $\{+3, -2, +1, -2\}$ (aceleasi segmente ca $\{+3, +1, -2, -2\}$, doar cu arie mai mică)
- $\{+3, -2, +3, -2, -2\}$
- $\{+2, -2\}$
- $\{+2, -1, -1\}$
- $\{+1, -1\}$

Prin $+x$ sau $-x$ am notat un segment de lungime x care scade sau crește.

Pentru simplificare, am considerat că tot timpul sunt mai multe segmente pozitive de felul cel mai lung, ca să nu ne complicăm cu dublarea cazurilor.

O implementare e recomandată să folosească aceeași metodă pentru a rezolva fiecare figură în parte, pentru a evita cod duplicat în care se pot strecura greșeli.

Soluția matematică se bazează pe un sistem de ecuații având drept necunoscute numărul de figuri de fiecare fel din cele enumerate mai sus.

21.5.2 Cod sursă

Listing 21.5.1: minarea.cpp

```

1 #include <iostream>
2 #include <cstdio>
3
4 using namespace std;
5
6 constexpr auto fin = "minarea.in";
7 constexpr auto fout = "minarea.out";
8
9 int poz[4], neg[4];
10 int sp=0; // suma parțială
11 int xc=0, yc=0, aria1=0, dim=0;
12
13 int minNEG()
14 {
15     int valMIN=0;
16     if(neg[1]>0) valMIN=1;
17     else
18         if(neg[2]>0) valMIN=2;
19     else
20         if(neg[3]>0) valMIN=3;
21 }
22

```

```

23 void adaugPOZ()
24 {
25     if(poz[1]+poz[2]+poz[3]==0) return;
26
27     cout<<"\n----- adaugPOZ ... \n";
28     if(poz[3]>0)
29     {
30         sp+=3;
31         --poz[3];
32         xc+=3;
33     }
34     else
35     if(poz[2]>0)
36     {
37         sp+=2;
38         --poz[2];
39         xc+=2;
40     }
41     else
42 //     if(poz[1]>0)
43     {
44         sp+=1;
45         --poz[1];
46         xc+=1;
47     }
48     cout<<poz[1]<<" " <<poz[2]<<" " <<poz[3]<<" " <<
49     neg[1]<<" " <<neg[2]<<" " <<neg[3]<<" " <<
50     "sp = "<<sp<<" " <<"xc = "<<xc<<" " yc = "<<yc<<" \n";
51 }
52
53 void adaugNEG()
54 {
55     cout<<"\n----- adaugNEG ... \n";
56
57     if((neg[1]>0) && (sp>=1))
58     {
59         sp-=1;
60         --neg[1];
61         yc+=1;
62         arial+=1*(dim-xc);
63     }
64     else
65     if((neg[2]>0) && (sp>=2))
66     {
67         sp-=2;
68         --neg[2];
69         yc+=2;
70         arial+=2*(dim-xc);
71     }
72     else
73     if((neg[3]>0) && (sp>=3))
74     {
75         sp-=3;
76         --neg[3];
77         yc+=3;
78         arial+=3*(dim-xc);
79     }
80     cout<<poz[1]<<" " <<poz[2]<<" " <<poz[3]<<" " <<
81     neg[1]<<" " <<neg[2]<<" " <<neg[3]<<" " <<
82     "sp = "<<sp<<" " <<"xc = "<<xc<<" " yc = "<<yc<<" \n";
83 }
84
85 void bruteforce()
86 {
87     dim=1*poz[1]+2*poz[2]+3*poz[3];
88
89     cout<<"dim = "<<dim<<" \n";
90
91     cout<<poz[1]<<" " <<poz[2]<<" " <<poz[3]<<" " <<
92     neg[1]<<" " <<neg[2]<<" " <<neg[3]<<" " <<
93     "sp = "<<sp<<" " <<"xc = "<<xc<<" " yc = "<<yc<<" \n";
94
95     while((poz[1]+poz[2]+poz[3]>0) || (neg[1]+neg[2]+neg[3]>0))
96     {
97
98

```

```

99         if( (sp==0) || (sp<minNEG()) )
100     {
101         adaugPOZ();
102     }
103     else
104     {
105         adaugNEG();
106         adaugPOZ();
107     }
108 }
109 // while
110 cout<<poz[1]<<" "<<poz[2]<<" "<<poz[3]<<"   "<<
111     neg[1]<<" "<<neg[2]<<" "<<neg[3]<<"   "<<
112     "sp = "<<sp<<"    "<<"xc = "<<xc<<"    yc = "<<yc<<"\\n";
113
114     cout<<"arial = "<<arial<<"\\n";
115 }
116
117
118 int main()
119 {
120     int k, n;
121     freopen(fin, "r", stdin);
122     freopen(fout, "w", stdout);
123
124     cin >> poz[1]; cin >> poz[2]; cin >> poz[3];
125     cin >> neg[1]; cin >> neg[2]; cin >> neg[3];
126
127     bruteforce();
128
129     int aria=(dim*dim-aria1)*2;
130
131     cout<<aria;
132     cout<<'\\n';
133 }
```

Listing 21.5.2: minarea_100p_1.cpp

```

1 #include <cstdio>
2 #include <vector>
3 #include <cmath>
4
5 using namespace std;
6
7 const int MAX_VAL = 3;
8 int buf[2 * MAX_VAL + 1];
9 int *countAvail = buf + MAX_VAL;
10
11 long long sol = 0;
12
13 void solve(const vector<int> types)
14 {
15     int buf[2 * MAX_VAL + 1] = {0};
16     int *countUsages = buf + MAX_VAL;
17     int lastHeight = 0, area = 0;
18     for (int i = 0; i < (int)types.size(); i++)
19     {
20         int nextHeight = lastHeight + types[i];
21         area += abs(types[i]) * (lastHeight + nextHeight); //divide in half later
22         lastHeight = nextHeight;
23         countUsages[types[i]]++;
24     }
25     area /= 2;
26     int numMoves = 1 << 30;
27     for (int i = -MAX_VAL; i <= MAX_VAL; i++)
28     {
29         if (countUsages[i])
30         {
31             numMoves = min(numMoves, countAvail[i] / countUsages[i]);
32         }
33     }
34
35     for (int i = -MAX_VAL; i <= MAX_VAL; i++)
36     {
37         countAvail[i] -= countUsages[i] * numMoves;
```

```

38     }
39     sol += (long long)numMoves * area;
40 }
41
42 int main()
43 {
44     freopen("minarea.in", "r", stdin);
45     freopen("minarea.out", "w", stdout);
46     for (int i = 1; i <= MAX_VAL; i++)
47     {
48         scanf("%d", countAvail + i);
49     }
50
51     for (int i = 1; i <= MAX_VAL; i++)
52     {
53         scanf("%d", countAvail - i);
54     }
55
56     solve({+3, -3});
57
58     if (!countAvail[MAX_VAL])
59     {
60         for (int i = 1; i <= MAX_VAL; i++)
61         {
62             swap(countAvail[i], countAvail[-i]);
63         }
64     }
65
66     solve({+3, -2, -1});
67     solve({+3, -1, -1, -1});
68     solve({+3, -2, +1, -2});
69     solve({+3, -2, +3, -2, -2});
70     solve({+2, -2});
71     solve({+2, -1, -1});
72     solve({+1, +1, -2});
73
74     solve({+1, -1});
75
76     printf("%lld\n", sol);
77
78     return 0;
79 }
```

Listing 21.5.3: minarea_100p_2.cpp

```

1 #include <iostream>
2 #include <algorithm>
3
4 using namespace std;
5
6 ifstream fin("minarea.in");
7 ofstream fout("minarea.out");
8
9 long long a,b,c,d,e,f,sol;
10
11 int main()
12 {
13     fin>>a>>b>>c>>d>>e>>f;
14
15     if(c<f)
16     {
17         swap(a,d);
18         swap(b,e);
19         swap(c,f);
20     }
21
22     sol+=9*f;
23     c-=f;
24     if(d<=c-a)
25         sol+=4*b+12*c-3*d;
26     else
27         if(d<=c+a)
28             sol+=2*a+4*b+10*c-d;
29         else
30             sol+=a+4*b+9*c;
```

```

31     fout<<sol<<' \n' ;
32     return 0;
33 }

```

21.5.3 *Rezolvare detaliată

21.6 order

Problema 6 - order

100 de puncte

Se consideră toate șirurile finite de numere naturale nenule ordonate astfel:

[1]; [1,1]; [2]; [1,1,1]; [1,2]; [2,1]; [3]; [1,1,1,1]; [1,1,2]; [1,2,1]; [1,3]; ...

Ordonarea se face după următoarea regulă: dacă avem două șiruri cu sumele termenilor diferite, atunci șirul cu suma termenilor mai mică se găsește pe o poziție mai mică. Dacă avem două șiruri cu sumele termenilor egale atunci se compară termen cu termen șirurile până când se găsește un termen diferit. șirul care are termenul mai mic se găsește pe poziție mai mică. Cu alte cuvinte, primul criteriu de ordonare este suma termenilor, iar în caz de egalitate, al doilea criteriu de sortare este ordinea lexicografică.

Oricărui șir i se asociază o poziție (număr natural nenul) și invers, oricarei poziții i se asociază un șir.

De exemplu:

- șirului [1,1,2] i se asociază poziția 9.
- Poziției 14 i se asociază șirul [3,1]

Cerințe

Să se răspundă la un număr de interogări de tipul:

1. Cunoscând un șir de numere naturale nenule să se determine poziția asociată șirului.
2. Cunoscând un număr natural reprezentând o poziție asociată unui șir să se determine șirul corespunzător.

Date de intrare

Fisierul de intrare **order.in** conține pe prima linie un număr natural Q reprezentând numărul de interogări.

Pe următoarele Q linii vor fi descrise interogările.

Dacă interogarea este de tip 1 linia va conține numărul 1, apoi un număr natural N reprezentând numărul de termeni ai șirului urmat de N numere naturale separate prin căsuțe un spațiu reprezentând termenii șirului.

Dacă interogarea este de tip 2 linia va conține numărul 2 urmat de un număr natural nenul P reprezentând poziția șirului solicitat.

Date de ieșire

Fisierul de ieșire **order.out** va conține Q linii. Pe fiecare linie este descris răspunsul la interogarea corespunzătoare din fisierul de intrare.

Dacă interogarea este de tip 1, pe linia corespunzătoare se va afișa un singur număr P reprezentând poziția șirului descris în interogare.

Dacă interogarea este de tip 2, linia corespunzătoare va conține un număr natural N reprezentând numărul de termeni pentru șirul solicitat, urmat de N numere naturale nenule reprezentând termenii șirului. Numerele de pe aceste linii trebuie să fie separate prin căsuțe un spațiu.

Restricții și precizări

- $1 \leq P \leq 10^{15}$ (mai precis se asigură ca pentru ambele tipuri de interogări poziția asociată șirului considerat nu depășește 10^{15})
- $1 \leq Q \leq 10^5$
- Pentru 40 de puncte testele vor conține doar interogări de tip 1
- Pentru 40 de puncte testele vor conține doar interogări de tip 2
- Pentru 20 de puncte testele vor conține interogări de ambele tipuri

Exemplu:

order.in	order.out	Explicații
2	9	Fisierul de intrare conține două interogări. Prima este de tip 1 și cere determinarea poziției sirului [1,1,2] care are lungimea 3.
1 3 1 1 2	2 3 1	Acest sir este pe poziția a 9-a conform ordinii descrise în enunț.
2 14		A doua interogare cere determinarea sirului aflat pe poziția 14. Acest sir este sirul [3,1] de lungime 2.

Timp maxim de executare/test: **1.2** secunde

Memorie: total **64 MB**

Dimensiune maximă a sursei: **20 KB**

21.6.1 Indicații de rezolvare

prof. Adrian Panaete - Colegiul National "A. T. Laurian" Botoșani

Se observă ca sirurile cu suma termenilor $S+1$ se pot obține în ordine din sirurile cu suma termenilor S astfel:

Prima dată se trec sirurile de suma S la care se atașează valoarea 1 la ineput. Apoi se trec sirurile de suma S la care se adună 1 la primul termen.

De exemplu, să considerăm cele 4 siruri suma $S=3$ în ordine lexicografică. [1,1,1]

[1,2]

[2,1]

[3]

și cele de suma $S+1 = 4$ [1,1,1,1]

[1,1,2]

[1,2,1]

[1,3]

[2,1,1]

[2,2]

[3,1]

[4]

Se observă că primele 4 sunt exact cele de la $S = 3$ cu un 1 adăugat la început iar ultimele 4 sunt obținute din cele de la $S = 3$ adunând o unitate la primul termen.

Se deduce foarte ușor că avem un sir pentru $S=1$, două siruri pentru $S=2$, 4 siruri pentru $S=3$, și în general 2^{S-1} siruri de suma S .

De aici se observă că pentru un sir de suma S poziția va fi un număr care se reprezintă pe S biti în baza 2. Se poate deduce de aici o legătură bijectivă între sir și reprezentarea binară a poziției acestuia.

Cel mai simplu se observă această bijectie pe exemplu:

Fie sirul [3, 4, 1, 5, 2, 1, 3]. Suma termenilor sirului este 20.

Scrierea binară a poziției p asociată acestui sir este un număr pe 20 de biti mai precis 11101110011111010011.

Consider descompunerea în secvențe cu lungimi exact termenii sirului. [111] [0111] [0] [01111] [01] [0] [011].

Schimb primul bit 1 în 0 și obțin: [011] [0111] [0] [01111] [01] [0] [011]

Se remarcă faptul că fiecare secvență are o formă caracteristică valorii corespunzătoare din sir.

Mai precis dacă în sir avem o valoare x în reprezentarea binară vom avea o secvență formată dintr-un 0 și x-1 de 1.

1 -> 0

2 -> 01

3 -> 011

...

Astfel conversia din poziție în sir și invers este realizată astfel:

Dacă avem sirul concatenam codificările binare ale termenilor și formăm scrierea binară a poziției (bineînteleș nu uităm să adăugăm unbit suplimentar la început).

Invers dacă avem poziția descompunem aceasta poziție în secvențe binare formate dintr-un 0 și orice valoare de 1 care urmează și astfel determinăm exact elementele din sir.

Bijectia este evidenta: Atat sirurile de lungime S cat si numerele pe S biti sunt in numar de 2^S .

Ordinea intre multimi si secventele binare care le codifica se pastreaza. Motivul e ca daca se compara doua multimi acestea vor genera secvente binare cu atat mai mari cu cat pozitia in sortarea pe criteriile indicate in enunt este mai mare.

21.6.2 Cod sursă

Listing 21.6.1: order_100p_1.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 ifstream f("order.in");
6 ofstream g("order.out");
7
8 int q,c,n,s,x,st[100],top,cnt;
9 long long sol;
10
11 int main()
12 {
13     f>>q;
14     for(;q;q--)
15     {
16         f>>c;
17         if(c==1)
18         {
19             for(f>>n,sol=0,s=0;n;n--)
20             {
21                 f>>x;
22                 s+=x;
23                 sol<<=x;
24                 x--;
25                 sol|=((1LL<<x)-1);
26             }
27
28             sol|=(1LL<<(--s));
29             g<<sol<<' \n';
30         }
31         else
32         {
33             f>>sol;
34             top=0;
35             while(sol)
36             {
37                 top++;
38                 while(sol%2)
39                 {
40                     st[top]++;
41                     sol/=2;
42                 }
43                 st[top]++;
44                 sol/=2;
45             }
46
47             st[top]--;
48             g<<top<<' ';
49             for(;top;top--)
50             {
51                 g<<st[top]<<' ';
52                 st[top]=0;
53             }
54             g<<' \n';
55         }
56     }
57
58     return 0;
59 }
```

Listing 21.6.2: order_100p_2.cpp

```

1 #include<bits/stdc++.h>
2
3 using namespace std;
4
5 ifstream in("order.in");
6 ofstream out("order.out");
7
8 const long long MAX_VAL = (long long)(1e15);
9 int v[100],t,y,k;
10 long long s;
11
12 int main()
13 {
14     in>>t;
15     while(t--)
16     {
17         in>>y;
18         assert((y==1 || y==2));
19         if(y==1)
20         {
21             in>>k;
22             s=0;
23             for(int i=1;i<=k;i++)
24                 in>>v[i],s+=v[i];
25             long long ans=(1LL<<(-s));
26             assert(0<=ans && ans<=MAX_VAL);
27
28             for(int i=1;i<=k;i++)
29             {
30                 for(int w=1;w<v[i];w++) ans+=(1LL<<(--s));
31                 assert(0<=ans && ans<=MAX_VAL);
32                 s--;
33             }
34             assert(1<=ans && ans<=MAX_VAL);
35             out<<ans<<'\\n';
36         }
37         else if(y==2)
38         {
39             in>>s;
40             assert(1<=s && s<=MAX_VAL);
41             int p=0;
42             k=0;
43
44             while(s>0)
45             {
46                 if(s%2==1) p++;
47                 else
48                 {
49                     v[++k]=p+1;
50                     p=0;
51                 }
52                 s>>=1;
53             }
54             v[++k]=p;
55             out<<k;
56             for(int i=k;i>=1;i--) out<<' '<<v[i];
57             out<<'\\n';
58         }
59     }
60
61     return 0;
62 }
```

21.6.3 *Rezolvare detaliată

Capitolul 22

ONI 2016

22.1 euro

Problema 1 - euro

100 de puncte

După calificarea la campionatul european de fotbal din Franța, având în vizor N jucători din care trebuie să convoace câțiva în echipa națională, selecționerul României a apelat la niște metode mai puțin ortodoxe. Acesta a mers la vrăjitoarele renumite din Craiova pentru a-l ajuta să gasească formula câștigătoare pentru meciul de deschidere cu Franța. Vrăjitoarele, după descântece îndelungate, au ajuns la concluzia că lotul de jucători trebuie să aibă valoarea exact X și coeficientul de aroganță cât mai mic. Valoarea unui lot de jucători e definită ca suma valorilor jucătorilor ce intră în componența lotului. Coeficientul de aroganță al unui lot de jucători e definit ca diferența dintre valoarea maximă a unui jucător din lot și valoarea minimă a unui jucător din lot. Se mai știe că valoarea lotului nu poate depăși o valoare cunoscută V_{max} . Un lot de jucători e definit ca o submulțime nevidă de jucători aleși dintre cei N . Atenție, un lot poate fi format și dintr-un singur jucător.

Cerințe

Se dă numărul N de jucători, numărul V_{max} definit mai sus și valoarea fiecărui jucător. Selecționerul României a găsit formula câștigătoare și e curios dacă puteți și voi. Fiindcă nu are încredere totală în vrăjitoare, acesta vă cere să aflați pentru fiecare valoare X din intervalul $[1, V_{max}]$ coeficientul de aroganță minim posibil pentru care există cel puțin un lot dintre cei N jucători cu valoare exact X . Dacă nu se poate obține nici un lot de valoare exact X , se consideră ca răspuns -1.

Date de intrare

Fișierul de intrare **euro.in** conține pe prima linie T , reprezentând numărul de teste. În continuare vor urma T teste, fiecare având următoarea structură: pe prima linie dintr-un test se află N și V_{max} , reprezentând numărul total de jucători, respectiv valoarea maximă pe care o poate avea un lot de jucători. A doua linie a testului conține N numere naturale despărțite prin câte un spațiu. Al i -lea număr de pe această linie reprezintă valoarea pe care o are al i -lea jucător.

Date de ieșire

În fișierul de ieșire **euro.out** se vor afișa T linii, câte una pentru fiecare test din fișierul de intrare. O linie corespunzătoare unui test conține V_{max} numere (V_{max} -ul testului curent), unde cel de-al i -lea număr reprezintă coeficientul de aroganță minim posibil pentru o submulțime de jucători de valoare exact i . În cazul în care nu există o submulțime de jucători de valoare exact i se afișează -1.

Restricții și precizări

- $1 \leq T \leq 2$
- $1 \leq N \leq 4000$
- $1 \leq V_{max} \leq 8000$
- $1 \leq valoare[i] \leq V_{max}$
- Pentru 20% din teste $N \leq 20$
- Pentru 40% din teste $N \leq 100$ și $V_{max} \leq 100$
- Pentru 50% din teste $N \leq 300$ și $V_{max} \leq 300$

Exemplu:

euro.in	euro.out	Explicații
2	-1 0 0 0 0 2 1	Pentru primul test:
47	0 0 0 2 1 0 5 0 3 5 4 5 6 2 7	<ul style="list-style-type: none"> Nu se poate gasi un lot de valoarea 1, deci raspunsul pentru 1 este 1. Se pot obtine loturi de valoare 2, 3, 4, 5 dintr-un singur jucator. Lotul de valoare 6 se poate obtine din jucatorii cu valorile 2 si 4. Pentru valoarea 7 există două loturi posibile formate din jucatorii cu valorile 5 2 respectiv 3 4. Cel din urmă lot are coeficientul de aroganță mai mic (adică $\max(3,4) - \min(3,4) = 1$).
5 2 3 4		
5 15		
1 8 2 3 6		

Timp maxim de executare/test: **1.5** secunde pe Windows, **0.7** secunde pe Linux

Memorie: total **16 MB**

Dimensiune maximă a sursei: **10 KB**

22.1.1 Indicații de rezolvare

Răzvan Dan Sălăjan, Universitatea "Babeș-Bolyai" Cluj-Napoca

Solutia 1. Complexitate $O(2^N * N)$

Se fixeaza fiecare submultime de jucatori si se actualizeaza raspunsul pentru valoarea respectiva.

Solutia 2. Complexitate $O(N^3 * V_{max})$

Fie $dp[i][j] = 1/0$. 1 - daca se poate forma un lot de jucatori de valoare exact j din primii i jucatori, 0 altfel.

Recurenta este:

$$dp[0][0] = 1;$$

$$dp[i][j] = dp[i-1][j];$$

$dp[i][j] = dp[i][j] \text{ OR (logic)} dp[i-1][j-a[i]]$, unde $a[i]$ reprezinta valoarea celui de-al i-lea jucator si $j >= a[i]$.

Se sorteaza crescator jucatorii in functie de valoarea lor. Pentru fiecare subsecventa de jucatori din sirul sortat se verifica pentru fiecare valoare intre $[1, V_{max}]$ daca se poate obtine. In cazul in care o putem obtine se actualizeaza coeficientul cu diferentea maxima din subsecventa curenta.

Solutia 3. Complexitate $O(N^2 * V_{max})$

Solutia se bazeaza pe solutia precedenta. Se observa ca putem construi dinamica de mai sus pentru un capat stang fixat in timp ce iteram prin posibilele pozitii ale capatului drept.

Solutie 4 Complexitate $O(N * V_{max} * V_{max})$ (Denis-Gabriel Mita, Universitatea Bucuresti).

Sortam valorile crescator. Ne fixam suma pentru care vrem sa aflam solutia in $O(V_{max})$.

Acum, daca avem minimul fixat, cat timp nu reusim sa formam suma dorita avansam cu maximul pana ce o obtinem. Daca am folosit la un moment de timp ultimul element al sirului si inca nu am obtinut suma dorita este clar ca nu mai avem solutie de aici incolo. Minimul il fixam incremental de la cel mai mic la cel mai mare. Pe parcurs, pentru a sti daca putem obtine suma dorita "adaugam" un element nou la rucsac si "scoatem" un element din rucsac.

Astfel ne mentionem $D[i] =$ in cate feluri putem forma suma i cu actuala subsecventa de elemente.

Atunci cand adaugam un element iteram prin sumele posibile in ordine descrescatoare si vom updata noua suma din actuala (daca $D[i] != 0$, $D[i+x] = D[i+x] + D[i]$), iar atunci cand scoatem iteram in ordine crescatoare sumele, iar daca o suma poate fi obtinuta, inseamna ca am obtinut si suma + valoarea curenta, asa ca din noua suma scadem actuala ($D[i+x] = D[i+x] - D[i]$).

Deoarece numarul de posibilitati poate fi foarte mare, ii putem retine restul la impartirea cu un numar prim mare. In practica este probabilitate foarte mica sa se intampla ca numarul de

posibilitati sa fie un multiplu de modulo selectat (in cazul nostru putem folosi lejer $10^9 + 7$ sau $10^9 + 9$).

Fixarea sumei este $O(Vmax)$, parcurgerea subsecventelor este $O(N)$, iar actualizarea rucsacului este $O(Vmax)$. Complexitate finala $O(N * Vmax * Vmax)$.

Solutia 5. Complexitate $O(N * Vmax)$

Se sorteaza crescator jucatorii in functie de valoarea lor.

Se construiese urmatoarea matrice $dp[i][j] =$ cea mai mare valoarea minima posibila a unui lot de jucatori de valoarea exact j din primii i jucatori.

Recurenta este:

$dp[i][j] = dp[i-1][j];$
 $dp[i][j] = \max(dp[i][j], dp[i-1][j-a[i]]),$ unde $a[i]$ reprezinta valoarea celui de-al i -lea jucator si $j \geq a[i].$
 $dp[i][a[i]] = a[i];$

Fiind la jucatorul i si avand fixata valoarea X incercam sa actualizam coeficientul de aroganta cu coeficientul curent(care este $a[i] - dp[i][X]$).

22.1.2 Cod sursă

Listing 22.1.1: euro_alex_100.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define ll long long
6 #define ld long double
7 #define pb push_back
8 #define mp make_pair
9 #define pii pair<int, int>
10 #define pll pair<ll, ll>
11 #define pdd pair<ld, ld>
12 #define all(x) (x).begin(), (x).end()
13 #define fi first
14 #define se second
15
16 const int NMAX = 10005;
17 const int INF = 1 << 29;
18
19 int t, n, p, a[NMAX], dp[NMAX], sol[NMAX];
20
21 int main()
22 {
23     cin.sync_with_stdio(false);
24
25     freopen("euro.in", "r", stdin);
26     freopen("euro.out", "w", stdout);
27
28     scanf("%d", &t);
29
30     for ( ; t; t--)
31     {
32         scanf("%d%d", &n, &p);
33         for (int i = 1; i <= n; i++)
34             scanf("%d", &a[i]);
35
36         sort(a + 1, a + n + 1);
37
38         for (int i = 1; i <= p; i++)
39         {
40             dp[i] = -1;
41             sol[i] = INF;
42         }
43
44         for (int i = 1; i <= n; i++)
45         {
46             for (int j = p; j - a[i] > 0; j--)
47             {
48                 dp[j] = max(dp[j], dp[j - a[i]]);
49             }
50         }
51
52         for (int i = 1; i <= p; i++)
53             printf("%d ", sol[i]);
54     }
55 }
```

```

49             if (dp[j] != -1)
50                 sol[j] = min(sol[j], a[i] - dp[j]);
51         }
52
53         dp[a[i]] = a[i];
54         sol[a[i]] = 0;
55     }
56
57     for (int i = 1; i <= p; i++)
58     {
59         if (sol[i] == INF)
60             sol[i] = -1;
61         printf("%d ", sol[i]);
62     }
63
64     printf("\n");
65 }
66
67     return 0;
68 }
```

Listing 22.1.2: euro_ciucu_100.cpp

```

1 #include <cstdio>
2 #include <cstdlib>
3 #include <algorithm>
4
5 using namespace std;
6
7 const int MaxN = 20000;
8 const int MaxS = 50000;
9 const int INF = 0x3f3f3f3f;
10
11 int N, S;
12 int a[MaxN];
13 int largestMin[MaxS], sol[MaxS];
14
15 void ReadInput()
16 {
17     scanf("%d %d", &N, &S);
18     for (int i = 0; i < N; i += 1) {
19         scanf("%d", a + i);
20     }
21 }
22
23 void SolveOk()
24 {
25     sort(a, a + N);
26     int sum = 0;
27     for (int i = 1; i <= S; i += 1)
28     {
29         sol[i] = INF;
30         largestMin[i] = 0;
31     }
32
33     for (int index = 0; index < N; index += 1)
34     {
35         int curVal = a[index];
36         sum += curVal;
37         largestMin[0] = curVal;
38         for (int k = min(sum, S); k >= curVal; k--)
39         {
40             if (largestMin[k - curVal] > largestMin[k])
41             {
42                 largestMin[k] = largestMin[k - curVal];
43                 if (curVal - largestMin[k] < sol[k])
44                     sol[k] = curVal - largestMin[k];
45             }
46         }
47     }
48
49     for (int i = 1; i <= S; i += 1)
50     {
51         if (sol[i] == INF)
52             sol[i] = -1;
```

```

53         printf("%d ", sol[i]);
54     }
55 }
56
57     printf("\n");
58 }
59
60 int main()
61 {
62     int tst;
63
64     freopen("euro.in", "r", stdin);
65     freopen("euro.out", "w", stdout);
66
67     scanf("%d", &tst);
68
69     while (tst--)
70     {
71         ReadInput();
72         SolveOk();
73     }
74
75     return 0;
76 }
```

Listing 22.1.3: euro_denis_20.cpp

```

1 #include <fstream>
2 #include <cstring>
3 #include <cassert>
4 #include <algorithm>
5
6 using namespace std;
7
8 ifstream f("euro.in");
9 ofstream g("euro.out");
10
11 const int maxP = 10000;
12 const int maxN = 5000;
13 const int mod = 1000000009;
14
15 const int P = maxP + 10;
16 const int N = maxN + 10;
17
18 int n, p, a[N], mi[P], sol[P], D[P];
19
20 void print()
21 {
22     for (int i = 1; i <= p; ++i)
23         if (sol[i] == 1 << 30)
24             sol[i] = -1;
25
26     for (int i = 1; i <= p; ++i)
27         g << sol[i] << (i == p ? "\n" : " ");
28 }
29
30 // Optim, N * P
31 void solve_100()
32 {
33     memset(mi, 0, sizeof(mi));
34     for (int i = 1; i <= n; ++i)
35     {
36         for (int j = p - a[i]; j >= 1; --j)
37             if (mi[j])
38             {
39                 mi[j + a[i]] = max(mi[j + a[i]], mi[j]);
40                 sol[j + a[i]] = min(sol[j + a[i]], a[i] - mi[j]);
41             }
42
43         mi[a[i]] = a[i];
44         sol[a[i]] = 0;
45     }
46
47     print();
48 }
```

```

49
50 // P^2 * N
51 void solve_brut()
52 {
53     for (int i = 1; i <= p; ++i)
54     {
55         memset(D, 0, sizeof(D));
56         D[0] = 1;
57         int k = 0;
58         for (int j = 1; j <= n; ++j)
59         {
60             while (!D[i] && k <= n)
61             {
62                 ++k;
63                 for (int l = i - a[k]; l >= 0; --l)
64                     if (D[l])
65                     {
66                         D[l + a[k]] += D[l];
67                         if (D[l + a[k]] >= mod)
68                             D[l + a[k]] -= mod;
69                     }
70             }
71             if (k > n)
72                 break;
73
74             sol[i] = min(sol[i], a[k] - a[j]);
75             for (int l = 0; l <= i - a[j]; ++l)
76                 if (D[l])
77                 {
78                     D[l + a[j]] -= D[l];
79                     if (D[l + a[j]] < 0)
80                         D[l + a[j]] += mod;
81                 }
82             }
83         }
84     }
85
86     print();
87 }
88
89 // 2 ^ N
90 void solve_back()
91 {
92     for (int cnf = 1; cnf < 1 << n; ++cnf)
93     {
94         int poz = 1;
95         int minim = 0;
96         int maxim;
97         int sum = 0;
98         int x = cnf;
99         while (x)
100        {
101            if (x & 1)
102            {
103                if (!minim)
104                {
105                    minim = a[poz];
106                }
107                maxim = a[poz];
108                sum += a[poz];
109            }
110            x >= 1;
111            poz++;
112        }
113
114        if (sum <= p)
115            sol[sum] = min(sol[sum], maxim - minim);
116    }
117
118    print();
119 }
120
121 int main ()
122 {
123     int T;
124     f >> T;

```

```

125     while (T --)
126     {
127         f >> n >> p;
128         assert (n >= 1 && n <= maxN);
129         assert (p >= 1 && p <= maxP);
130
131         for (int i = 1; i <= n; ++i)
132         {
133             f >> a[i];
134             assert (a[i] >= 1 && a[i] <= p);
135         }
136
137         for (int i = 1; i <= p; ++i)
138             sol[i] = 1 << 30;
139
140         sort (a + 1, a + n + 1);
141         solve_back();
142     }
143
144     return 0;
145 }
```

Listing 22.1.4: euro_denis_50.cpp

```

1 #include <iostream>
2 #include <cstring>
3 #include <cassert>
4 #include <algorithm>
5
6 using namespace std;
7
8 ifstream f("euro.in");
9 ofstream g("euro.out");
10
11 const int maxP = 10000;
12 const int maxN = 5000;
13 const int mod = 1000000009;
14
15 const int P = maxP + 10;
16 const int N = maxN + 10;
17
18 int n, p, a[N], mi[P], sol[P], D[P];
19
20 void print()
21 {
22     for (int i = 1; i <= p; ++i)
23         if (sol[i] == 1 << 30)
24             sol[i] = -1;
25
26     for (int i = 1; i <= p; ++i)
27         g << sol[i] << (i == p ? "\n" : " ");
28 }
29
30 // Optim, N * P
31 void solve_100()
32 {
33     memset(mi, 0, sizeof(mi));
34     for (int i = 1; i <= n; ++i)
35     {
36         for (int j = p - a[i]; j >= 1; --j)
37             if (mi[j])
38             {
39                 mi[j + a[i]] = max(mi[j + a[i]], mi[j]);
40                 sol[j + a[i]] = min(sol[j + a[i]], a[i] - mi[j]);
41             }
42
43         mi[a[i]] = a[i];
44         sol[a[i]] = 0;
45     }
46
47     print();
48 }
49
50 // P^2 * N
51 void solve_brut()
```

```

52  {
53      for (int i = 1; i <= p; ++i)
54      {
55          memset(D, 0, sizeof(D));
56          D[0] = 1;
57          int k = 0;
58          for (int j = 1; j <= n; ++j)
59          {
60              while (!D[i] && k <= n)
61              {
62                  ++k;
63                  for (int l = i - a[k]; l >= 0; --l)
64                      if (D[l])
65                      {
66                          D[l + a[k]] += D[l];
67                          if (D[l + a[k]] >= mod)
68                              D[l + a[k]] -= mod;
69                      }
70              }
71              if (k > n)
72                  break;
73          }
74          sol[i] = min(sol[i], a[k] - a[j]);
75          for (int l = 0; l <= i - a[j]; ++l)
76              if (D[l])
77              {
78                  D[l + a[j]] -= D[l];
79                  if (D[l + a[j]] < 0)
80                      D[l + a[j]] += mod;
81              }
82          }
83      }
84  }
85
86  print();
87 }
88
89 //  $2^N$ 
90 void solve_back()
91 {
92     for (int cnf = 1; cnf < 1 << n; ++cnf)
93     {
94         int poz = 1;
95         int minim = 0;
96         int maxim;
97         int sum = 0;
98         int x = cnf;
99         while (x)
100        {
101            if (x & 1)
102            {
103                if (!minim)
104                    minim = a[poz];
105
106                maxim = a[poz];
107                sum += a[poz];
108            }
109
110            x >= 1;
111            poz++;
112        }
113
114        if (sum <= p)
115            sol[sum] = min(sol[sum], maxim - minim);
116    }
117
118    print();
119 }
120
121 int main ()
122 {
123     int T;
124     f >> T;
125     while (T --)
126     {
127         f >> n >> p;

```

```

128     assert (n >= 1 && n <= maxN);
129     assert (p >= 1 && p <= maxP);
130
131     for (int i = 1; i <= n; ++i)
132     {
133         f >> a[i];
134         assert (a[i] >= 1 && a[i] <= p);
135     }
136
137     for (int i = 1; i <= p; ++i)
138         sol[i] = 1 << 30;
139
140     sort (a + 1, a + n + 1);
141     solve_brut();
142 }
143
144     return 0;
145 }
```

Listing 22.1.5: euro_denis_100.cpp

```

1 #include <fstream>
2 #include <cstring>
3 #include <cassert>
4 #include <algorithm>
5
6 using namespace std;
7
8 ifstream f("euro.in");
9 ofstream g("euro.out");
10
11 const int maxP = 10000;
12 const int maxN = 5000;
13 const int mod = 1000000009;
14
15 const int P = maxP + 10;
16 const int N = maxN + 10;
17
18 int n, p, a[N], mi[P], sol[P], D[P];
19
20 void print()
21 {
22     for (int i = 1; i <= p; ++i)
23         if (sol[i] == 1 << 30)
24             sol[i] = -1;
25
26     for (int i = 1; i <= p; ++i)
27         g << sol[i] << (i == p ? "\n" : " ");
28 }
29
30 // Optim, N * P
31 void solve_100()
32 {
33     memset(mi, 0, sizeof(mi));
34     for (int i = 1; i <= n; ++i)
35     {
36         for (int j = p - a[i]; j >= 1; --j)
37             if (mi[j])
38             {
39                 mi[j + a[i]] = max(mi[j + a[i]], mi[j]);
40                 sol[j + a[i]] = min(sol[j + a[i]], a[i] - mi[j]);
41             }
42
43         mi[a[i]] = a[i];
44         sol[a[i]] = 0;
45     }
46
47     print();
48 }
49
50 // P^2 * N
51 void solve_brut()
52 {
53     for (int i = 1; i <= p; ++i)
54     {
```

```

55         memset(D, 0, sizeof(D));
56         D[0] = 1;
57         int k = 0;
58         for (int j = 1; j <= n; ++j)
59         {
60             while (!D[i] && k <= n)
61             {
62                 ++k;
63                 for (int l = i - a[k]; l >= 0; --l)
64                 if (D[l])
65                 {
66                     D[l + a[k]] += D[l];
67                     if (D[l + a[k]] >= mod)
68                         D[l + a[k]] -= mod;
69                 }
70             }
71             if (k > n)
72                 break;
73         }
74         sol[i] = min(sol[i], a[k] - a[j]);
75         for (int l = 0; l <= i - a[j]; ++l)
76             if (D[l])
77             {
78                 D[l + a[j]] -= D[l];
79                 if (D[l + a[j]] < 0)
80                     D[l + a[j]] += mod;
81             }
82         }
83     }
84 }
85
86     print();
87 }
88
89 // 2 ^ N
90 void solve_back()
91 {
92     for (int cnf = 1; cnf < 1 << n; ++cnf)
93     {
94         int poz = 1;
95         int minim = 0;
96         int maxim;
97         int sum = 0;
98         int x = cnf;
99         while (x)
100        {
101            if (x & 1)
102            {
103                if (!minim)
104                    minim = a[poz];
105
106                maxim = a[poz];
107                sum += a[poz];
108            }
109            x >>= 1;
110            poz++;
111        }
112
113        if (sum <= p)
114            sol[sum] = min(sol[sum], maxim - minim);
115    }
116    print();
117 }
118
119 int main ()
120 {
121     int T;
122     f >> T;
123     while (T --)
124     {
125         f >> n >> p;
126         assert (n >= 1 && n <= maxN);
127         assert (p >= 1 && p <= maxP);
128
129         for (int i = 1; i <= n; ++i)
130         {

```

```

131         f >> a[i];
132         assert (a[i] >= 1 && a[i] <= p);
133     }
134
135     for (int i = 1; i <= p; ++i)
136         sol[i] = 1 << 30;
137
138     sort (a + 1, a + n + 1);
139     solve_100();
140 }
141
142 return 0;
143 }
```

Listing 22.1.6: euro_radu_50.cpp

```

1 #include <iostream>
2 #include <algorithm>
3
4 using namespace std;
5
6 const int nmax = 10010, pmax = 5010, inf = 0x3f3f3f3f;
7
8 int t, n, p, ans[pmax], dp[pmax], v[nmax];
9
10 int main()
11 {
12     freopen("euro.in", "r", stdin);
13     freopen("euro.out", "w", stdout);
14
15     scanf("%i", &t);
16     for (; t; -- t)
17     {
18         scanf("%i %i", &n, &p);
19         for (int i = 1; i <= n; ++ i)
20             scanf("%i", &v[i]);
21
22         sort(v + 1, v + n + 1);
23
24         for (int i = 1; i <= p; ++ i)
25             ans[i] = inf;
26
27         for (int i = 1; i <= n; ++ i)
28         {
29             for (int j = 0; j < pmax; ++ j)
30                 dp[j] = inf;
31             dp[0] = 1;
32
33             for (int j = i; j <= n; ++ j)
34                 for (int k = pmax - 1; k >= v[j]; -- k)
35                     if (dp[k - v[j]] != inf)
36                     {
37                         if (dp[k] == inf)
38                             ans[k] = min(ans[k], v[j] - v[i]);
39                         dp[k] = 1;
40                     }
41         }
42
43         for (int i = 1; i <= p; ++ i)
44         {
45             if (ans[i] == inf)
46                 ans[i] = -1;
47             printf("%i ", ans[i]);
48         }
49         printf("\n");
50     }
51 }
52
53 return 0;
54 }
```

Listing 22.1.7: euro_razvan_40.cpp

```
1 #include <bits/stdc++.h>
```

```

2
3  using namespace std;
4
5  ifstream f("euro.in");
6  ofstream g("euro.out");
7
8  #define pb push_back
9  #define ll long long
10 #define mp make_pair
11
12 const int NMAX = 1005;
13 const int PMAX = 1005;
14 const int INF = (1 << 29);
15
16 int n, p, a[NMAX];
17 int dp[NMAX][PMAX];
18 int ans[PMAX];
19
20 void bagaDin(vector< int> &v, int minVal, int maxVal)
21 {
22     for (int i = 0; i <= v.size(); ++i)
23         for (int j = 0; j <= p; ++j)
24             dp[i][j] = 0;
25
26     dp[0][0] = 1;
27     for (int i = 1; i <= v.size(); ++i)
28     {
29         int currVal = v[i - 1];
30         for (int j = 0; j <= p; ++j)
31         {
32             dp[i][j] = dp[i - 1][j];
33             if (j - currVal >= 0)
34                 dp[i][j] |= dp[i - 1][j - currVal];
35         }
36     }
37
38     for (int i = 1; i <= p; ++i)
39         if (dp[v.size()][i] != 0)
40             ans[i] = min(ans[i], maxVal - minVal);
41 }
42
43 int main()
44 {
45     int t;
46     f >> t;
47     for (; t; --t)
48     {
49         f >> n >> p;
50         for (int i = 1; i <= n; ++i)
51             f >> a[i];
52
53         sort(a + 1, a + n + 1);
54
55         for (int i = 1; i <= p; ++i)
56             ans[i] = INF;
57
58         for (int i = 1; i <= n; ++i)
59         {
60             vector< int> v;
61             for (int j = i; j <= n; ++j)
62             {
63                 v.pb(a[j]);
64                 bagaDin(v, a[i], a[j]);
65             }
66         }
67
68         for (int i = 1; i <= p; ++i)
69         {
70             if (ans[i] == INF)
71                 ans[i] = -1;
72
73             g << ans[i] << " ";
74         }
75
76         g << "\n";
77     }
}

```

```

78     return 0;
79 }

```

Listing 22.1.8: euro_razvan_100.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 ifstream f("euro.in");
6 ofstream g("euro.out");
7
8 #define pb push_back
9 #define ll long long
10 #define mp make_pair
11
12 const int NMAX = 5005;
13 const int PMAX = 10005;
14 const int INF = (1 << 29);
15 const int nmax = 4000;
16 const int pmax = 8000;
17
18 int n, p, dp[2][PMAX];
19 int a[NMAX];
20 int ans[PMAX];
21
22 int main()
23 {
24     int t;
25     f >> t;
26     assert(t >= 1 && t <= 2);
27     for (; t; --t)
28     {
29         f >> n >> p;
30         assert(n >= 1 && n <= nmax);
31         assert(p >= 1 && p <= pmax);
32         for (int i = 1; i <= n; ++i)
33         {
34             f >> a[i];
35
36             assert(a[i] <= p);
37             assert(a[i] > 0);
38             //assert(a[i] != 1);
39         }
40
41         sort(a + 1, a + n + 1);
42
43         for (int i = 1; i <= p; ++i) {
44             ans[i] = INF;
45         }
46
47         int prevLine = 0;
48         for (int i = 0; i <= p; ++i)
49             dp[prevLine][i] = 0;
50
51         for (int i = 1; i <= n; ++i)
52         {
53             int currentLine = prevLine ^ 1;
54             for (int j = 0; j <= p; ++j)
55             {
56                 dp[currentLine][j] = dp[prevLine][j];
57                 if (j - a[i] >= 0)
58                     dp[currentLine][j] = max(dp[currentLine][j],
59                                         dp[prevLine][j - a[i]]);
60             }
61
62             dp[currentLine][a[i]] = a[i];
63             for (int j = 1; j <= p; ++j)
64                 if (dp[currentLine][j] != 0)
65                     ans[j] = min(ans[j], a[i] - dp[currentLine][j]);
66
67             //cerr << i << " " << ans << " " << a[i] << " " << dp[i][p] << "\n";
68             prevLine = currentLine;
69         }

```

```

70
71     for (int i = 1; i <= p; ++i)
72     {
73         if (ans[i] == INF)
74             ans[i] = -1;
75
76         g << ans[i] << " ";
77     }
78
79     g << "\n";
80 }
81
82     return 0;
83 }
```

Listing 22.1.9: euro_stefan_100.cpp

```

1 #include <cstdio>
2 #include <algorithm>
3
4 #define MAXN 4005
5 #define MAXP 8005
6
7 using namespace std;
8
9 int nrt;
10 int n, p, v[MAXN];
11 int mnCan[MAXP];
12 int sol[MAXP];
13
14 int main()
15 {
16     freopen("euro.in", "r", stdin);
17     freopen("euro.out", "w", stdout);
18
19     scanf("%d", &nrt);
20
21     while(nrt--)
22     {
23         scanf("%d %d", &n, &p);
24
25         for(int i = 1; i <= n; ++i)
26             scanf("%d", &v[i]);
27
28         for(int i = 1; i <= p; ++i)
29         {
30             sol[i] = p + 1;
31             mnCan[i] = -1;
32         }
33
34         sort(v + 1, v + n + 1);
35
36         for(int i = 1; i <= n; ++i)
37         {
38             for(int j = p; j > v[i]; --j)
39                 mnCan[j] = max(mnCan[j], mnCan[j - v[i]]);
40             mnCan[v[i]] = v[i];
41
42             for(int j = 0; j <= p; ++j)
43                 if(mnCan[j] > -1)
44                     sol[j] = min(sol[j], v[i] - mnCan[j]);
45         }
46
47         for(int i = 1; i <= p; ++i)
48             printf("%d ", (sol[i] == p + 1)?(-1):sol[i]);
49             printf("\n");
50     }
51
52     return 0;
53 }
```

22.1.3 *Rezolvare detaliată

22.2 sushi

Problema 2 - sushi

100 de puncte

După o zi productivă de făcut curățenie, Henry și Hetty au ieșit în oraș la un restaurant de sushi. În acest restaurant există N mese unite între ele prin $N - 1$ benzi rulante cu dublu sens, astfel încât oricare două mese sunt conectate direct sau indirect prin benzi rulante. Pentru fiecare masă i , $1 \leq i \leq N$, cunoaștem atât numărul K_i de mese cu care este conectată direct, cât și lista ordonată de mese vecine acesteia: $V_{i,1}, V_{i,2}, \dots, V_{i,K_i}$.

Benzile rulante au rolul de a transporta preparatele la clienți. Acestea urmează un traseu unic, definit după următoarea regulă: pentru orice masă i , un preparat aflat la masa i care tocmai a venit dinspre masa $V_{i,j}$, va pleca de la masa i spre masa:

- $V_{i,j+1}$, dacă $1 \leq j < K_i$
- $V_{i,1}$, dacă $j = K_i$.

În plus, dacă un preparat nou este trimis de la masa 1 spre masa $V_{1,1}$, știm că acesta va ajunge la masa i pentru prima oară venind dinspre masa $V + i$, 1, pentru orice i , $1 \leq i \leq N$.

Cerințe

Henry și Hetty au intrat în restaurant la momentul de timp 0.

Ei știu că pe parcursul vizitei lor pe benzile rulante vor fi așezate M preparate.

Pentru fiecare din cele M preparate ei cunosc tripletul (x, y, t) , semnificând faptul că la momentul de timp t preparatul va fi așezat pe bandă în dreptul mesei x pentru a pleca spre spă masa $V_{x,y}$.

Ei mai știu și că timpul necesar unui preparat de a parcurge distanța dintre două mese vecine este de o unitate.

Cei doi se vor aseza la o masă și vor lua de pe bandă toate preparatele care trec prin dreptul mesei respective.

Henry și Hetty se întreabă: pentru fiecare masă i , care este timpul minim după care culeg toate cele M preparate ce vor fi puse pe bandă?

Date de intrare

Pe prima linie a fișierului de intrare **sushi.in** se vor afla două numere naturale N și M , reprezentând numărul de mese, respectiv numărul de preparate aflate în restaurant.

Pe următoarele N linii se vor afla descrierile listelor de vecini ale fiecărei mese. Aftfel, pe linia $i + 1$, se va afla numărul natural K_i , urmat de K_i numere naturale: $V_{i,1}, V_{i,2}, \dots, V_{i,K_i}$, cu semnificația din enunț.

Pe fiecare din următoarele M linii se va afla cate un triplet de numere naturale (x, y, t) , semnificând faptul că la momentul de timp t un preparat va fi așezat pe bandă în dreptul mesei x pentru a pleca spre masa $V_{x,y}$.

Date de ieșire

Pe prima linie a fișierului de ieșire **sushi.out** se vor afisa N numere naturale, al i -ulea dintre acestea reprezentând timpul necesar pentru culegerea tuturor preparatelor de pe bandă dacă Henry și Hetty s-ar aseza la masa cu indice i .

Restricții și precizări

- $1 \leq N \leq 100\,000$ • $1 \leq M \leq 100\,000$ • Pentru fiecare triplet (x, y, t) avem $1 \leq x \leq N$, $1 \leq y \leq K_x$ și $0 \leq t \leq 100\,000$

Exemplu:

sushi.in	sushi.out	Explicații
-----------------	------------------	-------------------

5 1 3 2 3 4 1 1 2 1 5 1 1 1 3 3 1 0	1 4 0 2 7	<p>Avem $N = 5$ mese și $M = 1$ preparate.</p> <p>Masa 1 se învecinează cu 3 mese: (2, 3, 4)</p> <p>Masa 2 se învecinează cu 1 masă: (1)</p> <p>Masa 3 se învecinează cu 2 mese: (1, 5)</p> <p>Masa 4 se învecinează cu 1 masă: (1)</p> <p>Masa 5 se învecinează cu 1 masă: (3)</p> <p>Singurul preparat va fi pus la momentul 0 la masa 3 pentru a pleca spre prima masă din lista de vecini a lui 3: masa cu indicele 1.</p> <p>Preparatul va avea următorul traseu: 3, 1, 4, 1, 2, 1, 3, 5, 3 ...</p> <p>El poate fi ridicat de la:</p> <ul style="list-style-type: none"> - masa 1 la momentul 1 - masa 2 la momentul 4 - masa 3 la momentul 0 - masa 4 la momentul 2 - masa 5 la momentul 7
3 2 2 2 3 1 1 1 1 2 1 0 3 1 1	2 3 2	<p>Avem $N = 3$ mese și $M = 2$ preparate.</p> <p>Masa 1 se învecinează cu 2 mese: (2, 3)</p> <p>Masa 2 se învecinează cu 1 masă: (1)</p> <p>Masa 3 se învecinează cu 1 masă: (1)</p> <p>Un preparat este pus la momentul 0 la masa 2 plecând spre prima masă din lista de vecini a lui 2: masa cu indicele 1. El poate fi ridicat de la:</p> <ul style="list-style-type: none"> - masa 1 la momentul 1 - masa 2 la momentul 0 - masa 3 la momentul 2 <p>Celălalt preparat este pus la momentul 1 la masa 3 plecând spre prima masă din lista de vecini a lui 3: masa cu indicele 1. El poate fi ridicat de la:</p> <ul style="list-style-type: none"> - masa 1 la momentul 2 - masa 2 la momentul 3 - masa 3 la momentul 1

Timp maxim de executare/test: **2.5** secunde pe Windows, **0.8** secunde pe Linux

Memorie: total **128 MB** din care pentru stivă **128 MB**

Dimensiune maximă a sursei: **20 KB**

22.2.1 Indicații de rezolvare

Vlad Gavrilă - Universitatea Cambridge - Anglia
Eugenie-Daniel Posdărăscu - Imperial College - Anglia

Observam ca traseul parcurs de sushi este o *parcuregere Euleriana* a arborelui unde benzile sunt muchii, iar mesele noduri.

Solutie $O(N * M)$ 40p

Luam fiecare produs și simulăm parcurgerea sa, actualizând timpul maxim în care ajunge un produs la o masă. Avem M produse și $2 * N$ benzi în parcurgere, deci complexitatea este $O(N * M)$

Solutie $O(N \log N)$ 100p

Dacă ne fixăm un nod, trebuie să calculăm pentru subsecvențele dintre 2 apariții consecutive ale nodului în parcurgerea Euleriana timpul maxim la care produsul ajunge în urmatoarea apariție a nodului.

Presupunând că pozițiile subsecvenței le considerăm în ordine crescătoare, produsul care ajunge ultimul va fi cel ce are timpul de apariție + distanța până la urmatoarea apariție a nodului maximă.

Distanța poate fi calculată ca poziția nodului - poziția produsului. Întrucât poziția nodului este fixată pentru o subsecvență, ne interesează doar timpul de apariție - poziția produsului maximă.

Pentru a calcula aceste maxime pe subsecvențe putem folosi fie un *RMQ*, fie un *arbore de intervale* sau o structură de date asemanătoare.

In ambele situații complexitatea este $O(N \log N)$. Implementarea cu grijă, aceasta soluție ia 100 de puncte.

Solutie $O(N)$ 100p

Calculăm pentru fiecare nod timpul maxim la care un produs va ajunge acolo.

Fixăm arbitrar radacina arborelui iar apoi calculăm

$D[i]$ = timpul maxim la care un preparat va ajunge în nodul i din subarbore și

$E[i]$ = timpul maxim la care un preparat va ieși din subarbore.

Așteptăm, $D[nod] = \max(E[i] + 1)$ sau produsul care apare cel mai tarziu în nodul nod.

Pentru a calcula $E[nod]$, luăm în considerare produsele ce apar în nodul nod și se duc în subarbore, cat și $E[i] +$ numărul de muchii ce trebuie parcurs după subarborele acestui fiu.

In primul caz putem *precalculate* pe masura ce ni se dau produsele folosindu-ne de timpii de intrare și ieșire ai nodurilor, iar în al doilea caz numărul de muchii ce trebuie parcurs le putem calcula ușor în funcție de numărul de noduri din subarborei la dreapta subarborelui fixat folosind *sume parțiale*.

Mai ramane acum pentru un nod să calculăm timpul maxim la care ajunge un produs din afara subarborelui său.

Așteptăm, dacă stim poziția de intrare și cea de ieșire a unui nod în *parcurgerea Euleriana*, produsul ce va ajunge ultimul va fi cel care are timpul de apariție + distanța în parcurgere fata de nodul curent maximă. Aceste valori pot fi *precalculate* cu sume parțiale de prefixe și sufixe pe sirul produselor în ordinea parcurgerii Euleriene în $O(N)$.

Așteptăm, pentru fiecare nod soluția va fi maximul dintre cele 2 valori calculate independent. Complexitate finală: $O(N)$.

22.2.2 Cod sursă

Listing 22.2.1: daniBrute.cpp

```

1 // Daniel Posdarascu
2 // 40 de puncte
3 #include<stdio.h>
4 #include<string.h>
5 #include<unordered_map>
6 #include<vector>
7
8 using namespace std;
9
10#define ll long long
11#define NMAX 100005
12#define pb push_back
13#define INF 1000000005
14
15unordered_map< ll, int > myHash;
16
17int euler[2 * NMAX], sol[NMAX], n, m;
18char viz[NMAX];
19vector<int> v[NMAX];
20int left[2 * NMAX], right[2 * NMAX], nrEuler;
21int dfirst[NMAX], dlast[NMAX], entry[NMAX], last[NMAX];
22
23inline void dfs(int nod, int lst)
24{
25    euler[++nrEuler] = nod;
26    viz[nod] = 1;
27
28    int i, lim = v[nod].size(), start = 0;
29    if(lst)
30    {
31        for(i = 0; i < lim; i++)
32            if(v[nod][i] == lst)
33                break;
34        if(i < lim - 1)

```

```

35         start = i + 1;
36     else
37         start = 0;
38 }
39
40     for(i = start; i < lim; i++)
41         if(!viz[v[nod][i]])
42         {
43             dfs(v[nod][i], nod);
44             euler[++nrEuler] = nod;
45         }
46
47     for(int i = 0; i < start; i++)
48         if(!viz[v[nod][i]])
49         {
50             dfs(v[nod][i], nod);
51             euler[++nrEuler] = nod;
52         }
53 }
54
55 inline void read()
56 {
57     int val, nr;
58
59     scanf("%d%d", &n, &m);
60     for(int i = 1; i <= n; i++)
61     {
62         scanf("%d", &nr);
63         for(int j = 1; j <= nr; j++)
64         {
65             scanf("%d", &val);
66             v[i].pb(val);
67         }
68     }
69 }
70
71 inline void prep()
72 {
73     int nod1, nod2, timp;
74
75     dfs(1, 0);
76
77     for(int i = nrEuler + 1; i <= 2 * nrEuler - 1; i++)
78         euler[i] = euler[i - nrEuler + 1];
79
80     for(int i = 1; i < nrEuler; i++)
81         myHash[(ll)euler[i] * NMAX + euler[i + 1]] = i;
82
83     for(int i = 1; i <= m; i++)
84     {
85         memset(viz, 0, sizeof(viz));
86         scanf("%d%d%d", &nod1, &nod2, &timp);
87         int index = myHash[(ll)nod1 * NMAX + v[nod1][nod2 - 1]];
88         for(int j = index; j <= index + nrEuler; j++)
89             if(!viz[euler[j]])
90             {
91                 sol[euler[j]] = max(sol[euler[j]], timp + j - index);
92                 viz[euler[j]] = 1;
93             }
94     }
95 }
96
97 inline void write()
98 {
99     for(int i = 1; i <= n; i++)
100        printf("%d ", (sol[i] < 0 ? 0 : sol[i]));
101        printf("\n");
102 }
103
104 int main ()
105 {
106     freopen("sushi.in", "r", stdin);
107     freopen("sushi.out", "w", stdout);
108
109     read();
110     prep();

```

```

111     write();
112
113     return 0;
114 }
```

Listing 22.2.2: daniN.cpp

```

1 //100 de puncte O(N)
2 #include<stdio.h>
3 #include<string.h>
4 #include<unordered_map>
5 #include<vector>
6
7 using namespace std;
8
9 #define ll long long
10#define NMAX 100005
11#define pb push_back
12#define INF 1000000005
13
14 unordered_map< ll, int > myHash;
15
16 int euler[2 * NMAX], sol[NMAX], n, m;
17 char viz[NMAX];
18 vector<int> v[NMAX];
19 int left[2 * NMAX], right[2 * NMAX], nrEuler;
20 int dfirst[NMAX], dlast[NMAX], entry[NMAX], last[NMAX];
21
22 inline void dfs(int nod, int lst)
23 {
24     euler[++nrEuler] = nod;
25     entry[nod] = last[nod] = nrEuler;
26     viz[nod] = 1;
27
28     int i, lim = v[nod].size(), start = 0;
29     if(lst)
30     {
31         for(i = 0; i < lim; i++)
32             if(v[nod][i] == lst)
33                 break;
34         if(i < lim - 1)
35             start = i + 1;
36         else
37             start = 0;
38     }
39
40     for(i = start; i < lim; i++)
41         if(!viz[v[nod][i]])
42         {
43             dfs(v[nod][i], nod);
44             euler[++nrEuler] = nod;
45             last[nod] = nrEuler;
46         }
47
48     for(int i = 0; i < start; i++)
49         if(!viz[v[nod][i]])
50         {
51             dfs(v[nod][i], nod);
52             euler[++nrEuler] = nod;
53             last[nod] = nrEuler;
54         }
55 }
56
57 inline void dfsDinamica(int nod)
58 {
59     viz[nod] = 1;
60     int lim = v[nod].size();
61     for(int i = 0; i < lim; i++)
62         if(!viz[v[nod][i]])
63         {
64             dfsDinamica(v[nod][i]);
65             dlast[nod] = max(dlast[nod],
66                               dlast[v[nod][i]] + last[nod] - last[v[nod][i]]);
67             dfirst[nod] = max(dfirst[nod], dlast[v[nod][i]] + 1);
68         }
}
```

```

69     /* printf("%d %d %d %d %d\n",
70      nod, entry[nod], last[nod], dfirst[nod], dlast[nod]);*/
71 }
72
73 inline void read()
74 {
75     int val, nr;
76
77     scanf("%d%d", &n, &m);
78     for(int i = 1; i <= n; i++)
79     {
80         scanf("%d", &nr);
81         for(int j = 1; j <= nr; j++)
82         {
83             scanf("%d", &val);
84             v[i].pb(val);
85         }
86     }
87 }
88
89 inline void prep()
90 {
91     int nod1, nod2, timp;
92
93     dfs(1, 0);
94
95     for(int i = 1; i < nrEuler; i++)
96         myHash[(ll)euler[i] * NMAX + euler[i + 1]] = i;
97
98     memset(viz, 0, sizeof(viz));
99     memset(dfirst, -INF, sizeof(dfirst));
100    memset(dlast, -INF, sizeof(dlast));
101    memset(left, -INF, sizeof(left));
102    memset(right, -INF, sizeof(right));
103
104    for(int i = 1; i <= m; i++)
105    {
106        scanf("%d%d%d", &nod1, &nod2, &timp);
107        int index = myHash[(ll)nod1 * NMAX + v[nod1][nod2 - 1]];
108        dlast[nod1] = max(dlast[nod1], timp + last[nod1] - index);
109        dfirst[nod1] = max(dfirst[nod1], timp);
110        left[index] = right[index] = max(left[index], timp);
111    }
112
113    dfsDinamica(1);
114    for(int i = 2; i <= nrEuler; i++)
115        left[i] = max(left[i], left[i - 1] + 1);
116    for(int i = nrEuler - 1; i >= 1; i--)
117        right[i] = max(right[i + 1], right[i] + nrEuler - i);
118 }
119
120 inline void solve()
121 {
122     for(int i = 1; i <= n; i++)
123     {
124         sol[i] = dfirst[i];
125         sol[i] = max(sol[i], left[entry[i] - 1] + 1);
126         sol[i] = max(sol[i], right[last[i] + 1] + entry[i] - 1);
127     }
128 }
129
130 inline void write()
131 {
132     for(int i = 1; i <= n; i++)
133         printf("%d ", (sol[i] < 0 ? 0 : sol[i]));
134         printf("\n");
135 }
136
137 int main ()
138 {
139     freopen("sushi.in", "r", stdin);
140     freopen("sushi.out", "w", stdout);
141
142     read();
143     prep();
144     solve();

```

```

145     write();
146
147     return 0;
148 }
```

Listing 22.2.3: daniNlogN.cpp

```

1 //100 de puncte O(N log N)
2 #include<stdio.h>
3 #include<string.h>
4 #include<unordered_map>
5 #include<vector>
6
7 using namespace std;
8
9 #define ll long long
10 #define NMAX 100005
11 #define pb push_back
12 #define INF 1000000005
13
14 unordered_map< ll, int > myHash;
15
16 int euler[4 * NMAX], sol[NMAX], n, m;
17 char viz[NMAX];
18 vector<int> v[NMAX];
19 vector<int> aparitii[NMAX];
20 int aint[16 * NMAX], nrEuler;
21
22 inline void dfs(int nod, int last)
23 {
24     euler[++nrEuler] = nod;
25     viz[nod] = 1;
26     int i, lim = v[nod].size(), start = 0;
27     if(last)
28     {
29         for(i = 0; i < lim; i++)
30             if(v[nod][i] == last)
31                 break;
32         if(i < lim - 1)
33             start = i + 1;
34         else
35             start = 0;
36     }
37
38     for(i = start; i < lim; i++)
39         if(!viz[v[nod][i]])
40         {
41             dfs(v[nod][i], nod);
42             euler[++nrEuler] = nod;
43         }
44
45     for(int i = 0; i < start; i++)
46         if(!viz[v[nod][i]])
47         {
48             dfs(v[nod][i], nod);
49             euler[++nrEuler] = nod;
50         }
51 }
52
53 inline void read()
54 {
55     int val, nr;
56
57     scanf("%d%d", &n, &m);
58     for(int i = 1; i <= n; i++)
59     {
60         scanf("%d", &nr);
61         for(int j = 1; j <= nr; j++)
62         {
63             scanf("%d", &val);
64             v[i].pb(val);
65         }
66     }
67 }
```

```

69  inline void update(int n, int st, int dr, int poz, int val)
70  {
71      if(st == dr)
72      {
73          aint[n] = max(aint[n], val);
74          return ;
75      }
76      int mij = (st + dr) / 2;
77      if(poz <= mij)
78          update(2 * n, st, mij, poz, val);
79      else
80          update(2 * n + 1, mij + 1, dr, poz, val);
81      aint[n] = max(aint[2 * n] + dr - mij, aint[2 * n + 1]);
82  }
83
84  inline int query(int n, int st, int dr, int a, int b)
85  {
86      if(a > b)
87          return 0;
88
89      if(a <= st && dr <= b)
90          return aint[n] + b - dr;
91
92      int mij = (st + dr) / 2, ans = -INF;
93      if(a <= mij)
94          ans = max(ans, query(2 * n, st, mij, a, b));
95      if(b > mij)
96          ans = max(ans, query(2 * n + 1, mij + 1, dr, a, b));
97      return ans;
98  }
99
100 inline void prep()
101 {
102     int nod1, nod2, timp;
103
104     dfs(1, 0);
105
106     for(int i = nrEuler + 1; i <= 2 * nrEuler - 1; i++)
107         euler[i] = euler[i - nrEuler + 1];
108
109     for(int i = 1; i < nrEuler; i++)
110         myHash[(ll)euler[i] * NMAX + euler[i + 1]] = i;
111
112     for(int i = 1; i <= nrEuler; i++)
113         aparitii[euler[i]].pb(i);
114
115     memset(aint, -INF, sizeof(aint));
116     for(int i = 1; i <= m; i++)
117     {
118         scanf("%d%d%d", &nod1, &nod2, &timp);
119         int index = myHash[(ll)nod1 * NMAX + v[nod1][nod2 - 1]];
120         update(1, 1, 2 * nrEuler - 1, index, timp);
121         update(1, 1, 2 * nrEuler - 1, index + nrEuler - 1, timp);
122     }
123 }
124
125 inline void solve()
126 {
127     for(int i = 1; i <= n; i++)
128     {
129         aparitii[i].pb(aparitii[i][0] + nrEuler - 1);
130         int lim = aparitii[i].size();
131         for(int j = 0; j < lim - 1; j++)
132             sol[i] = max(sol[i],
133                           query(1, 1, 2 * nrEuler - 1,
134                                 aparitii[i][j] + 1,
135                                 aparitii[i][j + 1]));
136     }
137 }
138
139 inline void write()
140 {
141     for(int i = 1; i <= n; i++)
142         printf("%d ", sol[i]);
143     printf("\n");
144 }
```

```

145
146 int main ()
147 {
148     freopen("sushi.in", "r", stdin);
149     freopen("sushi.out", "w", stdout);
150
151     read();
152     prep();
153     solve();
154     write();
155
156     return 0;
157 }
```

Listing 22.2.4: Stefan_Popa_NlogN.cpp

```

1 //100 de puncte
2 #include <cstdio>
3 #include <cassert>
4 #include <vector>
5 #include <unordered_map>
6 #include <map>
7 #define MAXN 100005
8 #define INF 0x3f3f3f3f
9
10 using namespace std;
11
12 int n, m, last[MAXN], prvAp[8 * MAXN], ai[16 * MAXN], sol[MAXN], mxt;
13 vector<int> trOrder[MAXN];
14
15 unordered_map<int, int> nextNode[MAXN];
16 map< pair<int, int>, int > posNode;
17
18 vector<int> traversal, tStart;
19
20 void build_ai(int node, int l, int r)
21 {
22     if(l == r)
23     {
24         ai[node] = tStart[l];
25         return;
26     }
27
28     int mid = (l + r) >> 1;
29     build_ai(2 * node, l, mid);
30     build_ai(2 * node + 1, mid + 1, r);
31
32     ai[node] = max(ai[2 * node], ai[2 * node + 1]);
33 }
34
35 void query_ai(int node, int l, int r, int lq, int rq, int &ans)
36 {
37     if(lq <= l && r <= rq)
38     {
39         ans = max(ans, ai[node]);
40         return;
41     }
42
43     int mid = (l + r) >> 1;
44     if(lq <= mid) query_ai(2 * node, l, mid, lq, rq, ans);
45     if(mid + 1 <= rq) query_ai(2 * node + 1, mid + 1, r, lq, rq, ans);
46 }
47
48 int main()
49 {
50     freopen("sushi.in", "r", stdin);
51     freopen("sushi.out", "w", stdout);
52
53     scanf("%d %d", &n, &m);
54
55     if(n == 1)
56     {
57         printf("0\n");
58         return 0;
59     }
```

```

60
61     assert(1 <= n && n <= 100000);
62     assert(1 <= m && m <= 100000);
63
64     for(int i = 1; i <= n; ++i)
65     {
66         int k, x;
67
68         scanf("%d", &k);
69
70         while(k--)
71         {
72             scanf("%d", &x);
73             trOrder[i].push_back(x);
74
75             assert(1 <= x && x <= n);
76         }
77
78         trOrder[i].push_back(trOrder[i][0]);
79
80         for(int j = 0; j < (int) trOrder[i].size() - 1; ++j)
81             nextNode[i][trOrder[i][j]] = trOrder[i][j + 1];
82
83         trOrder[i].pop_back();
84     }
85
86     int u, rootU;
87     int p, rootP;
88     int cntRoot = 0;
89
90     u = rootU = 1;
91     p = rootP = trOrder[1].back();
92
93     traversal.push_back(0);
94
95     while(cntRoot <= 2)
96     {
97         if(u == rootU && p == rootP) ++cntRoot;
98
99         traversal.push_back(u);
100        int nxt = nextNode[u][p];
101
102        p = u; u = nxt;
103    }
104
105    assert((int) traversal.size() == 4 * n - 2);
106
107    int nt = 2 * (n - 1);
108
109    for(int i = 1; i <= nt; ++i)
110        posNode[make_pair(traversal[i], traversal[i + 1])] = i;
111
112    tStart.resize((int) traversal.size(), -INF);
113
114    for(int i = 1; i <= m; ++i)
115    {
116        int x, y, t;
117
118        scanf("%d %d %d", &x, &y, &t);
119        mxt = max(mxt, t);
120
121        assert(1 <= x && x <= n);
122        assert(1 <= y && y <= (int) trOrder[x].size());
123        assert(0 <= t && t <= 100000);
124
125        int pos = posNode[make_pair(x, trOrder[x][y - 1])];
126
127        tStart[pos] = tStart[pos + nt] = max(tStart[pos], t);
128    }
129
130    for(int i = 1; i <= n; ++i) last[i] = -1;
131
132    for(int i = 1; i <= 2 * nt; ++i)
133    {
134        prvAp[i] = last[traversal[i]];
135        last[traversal[i]] = i;

```

```

136     }
137
138     for(int i = 1; i <= 2 * nt; ++i)
139         tStart[i] -= i;
140
141     build_ai(1, 1, 2 * nt);
142
143     for(int i = 1; i <= 2 * nt; ++i)
144         if(prvAp[i] > 0)
145             if(prvAp[i] + 1 <= i - 1)
146             {
147                 int x = -INF;
148                 query_ai(1, 1, 2 * nt, prvAp[i] + 1, i - 1, x);
149                 x += i;
150                 sol[traversal[i]] = max(sol[traversal[i]], x);
151             }
152
153     for(int i = 1; i <= n; ++i)
154         printf("%d ", max(sol[i], mxt));
155
156     printf("\n");
157
158     return 0;
159 }
```

Listing 22.2.5: sushi_bulaneala.cpp

```

1 // 40 de puncte O(N*M) cu stop
2 #include<stdio.h>
3 #include<string.h>
4 #include<unordered_map>
5 #include<vector>
6 #include <ctime>
7 #include <algorithm>
8
9 using namespace std;
10
11 #define ll long long
12 #define NMAX 1000005
13 #define pb push_back
14 #define INF 1000000005
15
16 unordered_map< ll, int > myHash;
17
18 int euler[2 * NMAX], sol[NMAX], n, m;
19 char viz[NMAX];
20 vector<int> v[NMAX];
21 int left[2 * NMAX], right[2 * NMAX], nrEuler;
22 int dfirst[NMAX], dlast[NMAX], entry[NMAX], last[NMAX];
23
24 double programStartTime = clock();
25
26 inline void dfs(int nod, int lst)
27 {
28     euler[++nrEuler] = nod;
29     viz[nod] = 1;
30
31     int i, lim = v[nod].size(), start = 0;
32     if(lst)
33     {
34         for(i = 0; i < lim; i++)
35             if(v[nod][i] == lst)
36                 break;
37             if(i < lim - 1)
38                 start = i + 1;
39             else
40                 start = 0;
41     }
42
43     for(i = start; i < lim; i++)
44         if(!viz[v[nod][i]])
45         {
46             dfs(v[nod][i], nod);
47             euler[++nrEuler] = nod;
48         }
}
```

```

49
50     for(int i = 0; i < start; i++)
51         if(!viz[v[nod][i]])
52         {
53             dfs(v[nod][i], nod);
54             euler[++nrEuler] = nod;
55         }
56     }
57
58 inline void read()
59 {
60     int val, nr;
61
62     scanf("%d%d", &n, &m);
63     for(int i = 1; i <= n; i++)
64     {
65         scanf("%d", &nr);
66         for(int j = 1; j <= nr; j++)
67         {
68             scanf("%d", &val);
69             v[i].pb(val);
70         }
71     }
72 }
73
74 struct Platter
75 {
76     int startNode, nextNode, startTime;
77
78     void read()
79     {
80         scanf("%d %d %d", &startNode, &nextNode, &startTime);
81     }
82
83     bool operator<(const Platter &o) const
84     {
85         return startTime > o.startTime;
86     }
87
88     void updateSolutions()
89     {
90         memset(viz, 0, sizeof(viz));
91         int index = myHash[(ll)startNode * NMAX + v[startNode][nextNode - 1]];
92         for (int j = index; j <= index + nrEuler; j++)
93         {
94             int currentNode = euler[j];
95             if (!viz[currentNode])
96             {
97                 if (startTime + j - index > sol[currentNode])
98                     sol[currentNode] = startTime + j - index;
99
100                viz[currentNode] = 1;
101            }
102        }
103    }
104 };
105
106 inline void prep()
107 {
108     dfs(1, 0);
109
110    for(int i = nrEuler + 1; i <= 2 * nrEuler - 1; i++)
111        euler[i] = euler[i - nrEuler + 1];
112
113    for(int i = 1; i < nrEuler; i++)
114        myHash[(ll)euler[i] * NMAX + euler[i + 1]] = i;
115
116    vector<Platter> platters(m);
117
118    for(int i = 0; i < m; i++)
119        platters[i].read();
120
121    sort(platters.begin(), platters.end());
122
123    int currentPlatter = 0;
124

```

```

125     while (currentPlatter < m &&
126            (clock() - programStartTime) / CLOCKS_PER_SEC < 2.0)
127     {
128         platters[currentPlatter++].updateSolutions();
129     }
130
131     fprintf(stderr, "Did %d iterations\n", currentPlatter);
132 }
133
134 inline void write()
135 {
136     for(int i = 1; i <= n; i++)
137         printf("%d ", (sol[i] < 0 ? 0 : sol[i]));
138     printf("\n");
139 }
140
141 int main ()
142 {
143     freopen("sushi.in", "r", stdin);
144     freopen("sushi.out", "w", stdout);
145
146     read();
147     prep();
148     write();
149
150     return 0;
151 }
```

Listing 22.2.6: sushi-HH-100.cpp

```

1 /**
2 * Author: Andrei Heidelbacher
3 * Time complexity: O(N)
4 * Space complexity: O(N)
5 */
6
7 #include <iostream>
8 #include <vector>
9 #include <algorithm>
10
11 using namespace std;
12
13 const char IN_FILE[] = "sushi.in";
14 const char OUT_FILE[] = "sushi.out";
15 const int NIL = -1;
16 const int oo = 0x3f3f3f3f;
17
18 struct Edge
19 {
20     int vertex, index;
21 };
22
23 struct Query
24 {
25     int index, t;
26 };
27
28 vector<vector<Edge>> Tree;
29 int V, Root;
30 vector<int> Euler, First, Last, Position;
31 int Q;
32 vector<Query> Queries;
33 vector<int> Values, Mint;
34
35 void eulerDfs(const int x, const int father = NIL)
36 {
37     First[x] = Last[x] = int(Euler.size());
38     Euler.push_back(x);
39     for (int e = (father != NIL ? 1 : 0); e < int(Tree[x].size()); ++e)
40     {
41         const int y = Tree[x][e].vertex;
42         if (y != father)
43         {
44             Position[Tree[x][e].index] = int(Euler.size()) - 1;
45             eulerDfs(y, x);
46         }
47     }
48 }
```

```

46         Last[x] = int(Euler.size());
47         Euler.push_back(x);
48     }
49 }
50
51 if (father != NIL)
52     Position[Tree[x][0].index] = int(Euler.size()) - 1;
53 }
54
55 void buildEuler()
56 {
57     First = Last = vector<int>(V, -1);
58     eulerDfs(0);
59 }
60
61 void buildValues()
62 {
63     const int eulerSize = int(Euler.size());
64     Values = vector<int>(eulerSize, -oo);
65     for (int q = 0; q < Q; ++q)
66     {
67         const int p = Position[Queries[q].index];
68         Values[p] = max(Values[p], Queries[q].t - p);
69     }
70 }
71
72 void solveLeft()
73 {
74     int prefixMaxValue = -oo;
75     for (int i = 0; i < int(Euler.size()); ++i)
76     {
77         const int x = Euler[i];
78         prefixMaxValue = max(prefixMaxValue, Values[i]);
79         if (i == First[x])
80             MinT[x] = max(MinT[x], prefixMaxValue + i);
81     }
82 }
83
84 void solveRight()
85 {
86     int suffixMaxValue = -oo;
87     for (int i = int(Euler.size()) - 1; i >= 0; --i)
88     {
89         const int x = Euler[i];
90         if (i == Last[x])
91             MinT[x] = max(MinT[x], suffixMaxValue+First[x]+int(Euler.size())-1);
92         suffixMaxValue = max(suffixMaxValue, Values[i]);
93     }
94 }
95
96 void solveMiddle()
97 {
98     vector<pair<int, int>> stack;
99     for (int i = 0; i < int(Euler.size()); ++i)
100    {
101        const int x = Euler[i];
102        if (i == First[x])
103        {
104            stack.push_back(make_pair(x, Values[i]));
105        }
106        else
107        {
108            int maxValue = Values[i];
109            while (!stack.empty() && stack.back().first != x)
110            {
111                maxValue = max(maxValue, stack.back().second);
112                stack.pop_back();
113            }
114
115            MinT[x] = max(MinT[x], i + maxValue);
116            maxValue = max(maxValue, stack.back().second);
117            stack.pop_back();
118            stack.push_back(make_pair(x, maxValue));
119        }
120    }
121 }

```

```

122
123 void solve()
124 {
125     buildEuler();
126     buildValues();
127     solveLeft();
128     solveRight();
129     solveMiddle();
130 }
131
132 void read()
133 {
134     ifstream in(IN_FILE);
135     in >> V >> Q;
136
137     Tree = vector<vector<Edge>>(V);
138
139     Root = 0;
140     int edgeIndex = 0;
141     for (int x = 0; x < V; ++x)
142     {
143         int degree;
144         in >> degree;
145         Tree[x] = vector<Edge>(degree);
146         for (int i = 0; i < degree; ++i)
147         {
148             in >> Tree[x][i].vertex;
149             --Tree[x][i].vertex;
150             Tree[x][i].index = edgeIndex++;
151         }
152     }
153
154     Position = vector<int>(edgeIndex, -1);
155     Queries = vector<Query>(Q);
156     for (int q = 0; q < Q; ++q)
157     {
158         int x, i;
159         in >> x >> i >> Queries[q].t;
160         --x;
161         --i;
162         Queries[q].index = Tree[x][i].index;
163     }
164
165     MinT = vector<int>(V, 0);
166     in.close();
167 }
168
169 void write()
170 {
171     ofstream out(OUT_FILE);
172     for (int x = 0; x < V; ++x)
173         out << MinT[x] << ((x == V - 1) ? "\n" : " ");
174     out.close();
175 }
176
177 int main()
178 {
179     read();
180     solve();
181     write();
182     return 0;
183 }
```

Listing 22.2.7: sushi-HH-100-hash.cpp

```

1 /**
2 * Author: Andrei Heidelbacher
3 * Time complexity: O(N)
4 * Space complexity: O(N)
5 * 100 points
6 */
7
8 #include <iostream>
9 #include <vector>
10 #include <algorithm>
```

```

11 #include <unordered_map>
12
13 using namespace std;
14
15 const char IN_FILE[] = "sushi.in";
16 const char OUT_FILE[] = "sushi.out";
17 const int NIL = -1;
18 const int oo = 0x3f3f3f3f;
19
20 typedef vector<int>::const_iterator Edge;
21
22 struct Query
23 {
24     int x, y, t;
25 };
26
27 vector<vector<int>> Tree;
28 int V, Root;
29 vector<int> Euler;
30 unordered_map<int64_t, int> Position;
31 int Q;
32
33 vector<Query> Queries;
34 vector<int> Values;
35 vector<int> MinT;
36
37 inline int edgeIndex(const int x, const int y)
38 {
39     return 1LL * x * V + y;
40 }
41
42 void eulerDfs(const int x, const int father = NIL)
43 {
44     Euler.push_back(x);
45     for (Edge y = Tree[x].begin(); y != Tree[x].end(); ++y)
46     {
47         if (*y != father)
48         {
49             Position[edgeIndex(x, *y)] = int(Euler.size()) - 1;
50             eulerDfs(*y, x);
51             Position[edgeIndex(*y, x)] = int(Euler.size()) - 1;
52             Euler.push_back(x);
53         }
54     }
55 }
56
57 void buildValues()
58 {
59     eulerDfs(0);
60     const int eulerSize = int(Euler.size());
61     Values = vector<int>(eulerSize, -oo);
62     for (int q = 0; q < Q; ++q)
63     {
64         const int p = Position[edgeIndex(Qualities[q].x, Qualities[q].y)];
65         Values[p] = max(Values[p], Qualities[q].t - p);
66     }
67 }
68
69 void solveLeft()
70 {
71     vector<bool> visited = vector<bool>(V, false);
72     int prefixMaxValue = -oo;
73     for (int i = 0; i < int(Euler.size()); ++i)
74     {
75         const int x = Euler[i];
76         prefixMaxValue = max(prefixMaxValue, Values[i]);
77         if (!visited[x])
78         {
79             MinT[x] = max(MinT[x], prefixMaxValue + i);
80             visited[x] = true;
81         }
82     }
83 }
84
85 void solveRight()
86 {

```

```

87     vector<bool> visited = vector<bool>(V, false);
88     int suffixMaxValue = -oo;
89     for (int i = int(Euler.size()) - 1; i >= 0; --i)
90     {
91         const int x = Euler[i];
92         if (!visited[x])
93         {
94             MinT[x] = max(MinT[x], suffixMaxValue + i + int(Euler.size()) - 1);
95             visited[x] = true;
96         }
97         suffixMaxValue = max(suffixMaxValue, Values[i]);
98     }
99 }
100
101 void solveMiddle()
102 {
103     vector<pair<int, int>> stack;
104     vector<bool> visited = vector<bool>(V, false);
105
106     for (int i = 0; i < int(Euler.size()); ++i)
107     {
108         const int x = Euler[i];
109         if (!visited[x])
110         {
111             visited[x] = true;
112             stack.push_back(make_pair(x, Values[i]));
113         }
114     else
115     {
116         int maxValue = Values[i];
117         while (!stack.empty() && stack.back().first != x)
118         {
119             maxValue = max(maxValue, stack.back().second);
120             stack.pop_back();
121         }
122
123         MinT[x] = max(MinT[x], i + maxValue);
124         maxValue = max(maxValue, stack.back().second);
125         stack.pop_back();
126         stack.push_back(make_pair(x, maxValue));
127     }
128 }
129 }
130
131 void solve()
132 {
133     buildValues();
134     solveLeft();
135     solveRight();
136     solveMiddle();
137 }
138
139 void read()
140 {
141     ifstream in(IN_FILE);
142     in >> V >> Q;
143     Tree = vector<vector<int>>(V);
144     for (int x = 0; x < V; ++x)
145     {
146         int degree;
147         in >> degree;
148         Tree[x] = vector<int>(degree);
149         for (int i = 0; i < degree; ++i)
150         {
151             in >> Tree[x][i];
152             --Tree[x][i];
153         }
154     }
155
156     Root = 0;
157     Queries = vector<Query>(Q);
158     for (int q = 0; q < Q; ++q)
159     {
160         in >> Queries[q].x >> Queries[q].y >> Queries[q].t;
161         --Queries[q].x;
162         --Queries[q].y;

```

```

163     Queries[q].y = Tree[Queries[q].x][Queries[q].y];
164 }
165
166 MinT = vector<int>(V, 0);
167 in.close();
168 }
169
170 void write()
171 {
172     ofstream out(OUT_FILE);
173     for (int x = 0; x < V; ++x)
174         out << MinT[x] << ((x == V - 1) ? "\n" : " ");
175     out.close();
176 }
177
178 int main()
179 {
180     read();
181     solve();
182     write();
183     return 0;
184 }
```

Listing 22.2.8: sushi-HH-brute.cpp

```

1 /**
2 * Author: Andrei Heidelbacher
3 * Time complexity: O(N^2)
4 * Space complexity: O(N)
5 * 40 points
6 */
7
8 #include <fstream>
9 #include <vector>
10 #include <algorithm>
11 #include <unordered_map>
12
13 using namespace std;
14
15 const char IN_FILE[] = "sushi.in";
16 const char OUT_FILE[] = "sushi.out";
17 const int NIL = -1;
18 const int oo = 0x3f3f3f3f;
19
20 typedef vector<int>::const_iterator Edge;
21
22 struct Query
23 {
24     int x, y, t;
25 };
26
27 vector<vector<int>> Tree;
28 int V, Root;
29 vector<int> Euler;
30 unordered_map<int64_t, int> Position;
31 int Q;
32
33 vector<Query> Queries;
34 vector<int> Values;
35
36 inline int edgeIndex(const int x, const int y)
37 {
38     return 1LL * x * V + y;
39 }
40
41 void eulerDfs(const int x, const int father = NIL)
42 {
43     Euler.push_back(x);
44     for (Edge y = Tree[x].begin(); y != Tree[x].end(); ++y)
45     {
46         if (*y != father)
47         {
48             Position[edgeIndex(x, *y)] = int(Euler.size()) - 1;
49             eulerDfs(*y, x);
50             Position[edgeIndex(*y, x)] = int(Euler.size()) - 1;
51         }
52     }
53 }
```

```

51         Euler.push_back(x);
52     }
53 }
54 }
55
56 void buildValues()
57 {
58     eulerDfs(0);
59     const int eulerSize = int(Euler.size());
60     Values = vector<int>(eulerSize, -oo);
61     for (int q = 0; q < Q; ++q)
62     {
63         const int p = Position[edgeIndex(Questions[q].x, Questions[q].y)];
64         Values[p] = max(Values[p], Questions[q].t - p);
65     }
66
67     for (int i = 1; i < eulerSize; ++i)
68     {
69         Euler.push_back(Euler[i]);
70         Values.push_back(Values[i] - eulerSize + 1);
71     }
72 }
73
74 void solve()
75 {
76     buildValues();
77 }
78
79 void read()
80 {
81     ifstream in(IN_FILE);
82     in >> V >> Q;
83     Tree = vector<vector<int>>(V);
84     for (int x = 0; x < V; ++x)
85     {
86         int degree;
87         in >> degree;
88         Tree[x] = vector<int>(degree);
89         for (int i = 0; i < degree; ++i)
90         {
91             in >> Tree[x][i];
92             --Tree[x][i];
93         }
94     }
95
96     Root = 0;
97     Questions = vector<Query>(Q);
98     for (int q = 0; q < Q; ++q)
99     {
100         in >> Questions[q].x >> Questions[q].y >> Questions[q].t;
101         --Questions[q].x;
102         --Questions[q].y;
103         Questions[q].y = Tree[Questions[q].x][Questions[q].y];
104     }
105     in.close();
106 }
107
108 void write()
109 {
110     ofstream out(OUT_FILE);
111     for (int x = 0; x < V; ++x)
112     {
113         int minT = 0;
114         for (int i = 0, maxValue = -oo; i < int(Euler.size()); ++i)
115         {
116             maxValue = max(maxValue, Values[i]);
117             if (Euler[i] == x)
118             {
119                 minT = max(minT, i + maxValue);
120                 maxValue = -oo;
121             }
122         }
123         out << minT << ((x == V - 1) ? "\n" : " ");
124     }
125     out.close();
126 }

```

```

127
128 int main()
129 {
130     read();
131     solve();
132     write();
133     return 0;
134 }
```

Listing 22.2.9: sushi-Nlog-vladgavrilă.cpp

```

1 //100 de puncte O(N log N)
2 #include <iostream>
3 #include <vector>
4
5 using namespace std;
6
7 #define maxn 100010
8 #define inf (1000000000)
9
10 int n, m, k, poz;
11 vector<int> v[maxn], w[maxn];
12 int f[maxn];
13 int euler[4*maxn], hasPrep[4*maxn],
14     whenPrep[4*maxn], prevPos[4*maxn], lastOcc[maxn];
15 int sol[maxn];
16 int aint[4*4*maxn];
17
18 void addNode(int nod, int prep)
19 {
20     euler[nod]=nod;
21     if(prep!=-1)
22     {
23         hasPrep[nod]=1;
24         whenPrep[nod]=prep;
25     }
26     ++poz;
27 }
28
29 void df(int nod)
30 {
31     f[nod]=1;
32
33     addNode(nod, w[nod][0]);
34
35     int fiu;
36
37     for(int i=1; i<v[nod].size(); ++i)
38     {
39         fiu=v[nod][i];
40         df(fiu);
41         addNode(nod, w[nod][i]);
42     }
43 }
44
45 void build(int nod, int left, int right)
46 {
47     if(left==right)
48     {
49         if(hasPrep[left]==0)
50             aint[nod]=-inf;
51         else
52             aint[nod]=whenPrep[left]-left;
53         return;
54     }
55
56     int med=(left+right)/2;
57
58     build(nod*2, left, med);
59     build(nod*2+1, med+1, right);
60
61     aint[nod]=max(aint[nod*2], aint[nod*2+1]);
62 }
63
64 int query(int nod, int left, int right, int qleft, int qright)
```

```

65  {
66      if(qleft<=left && right<=qright)
67          return aint[nod];
68
69      int med=(left+right)/2, sol=-inf;
70
71      if(qleft<=med)
72          sol=max(sol, query(nod*2, left, med, qleft, qright));
73      if(med<qright)
74          sol=max(sol, query(nod*2+1, med+1, right, qleft, qright));
75
76      return sol;
77  }
78
79  int main()
80  {
81      freopen("sushi.in", "r", stdin);
82      freopen("sushi.out", "w", stdout);
83
84      scanf("%d%d", &n, &m);
85
86      v[1].push_back(0);
87      w[1].push_back(-1);
88
89      for(int i=1; i<=n; ++i)
90      {
91          scanf("%d", &k);
92          for(int j=1; j<=k; ++j)
93          {
94              int x;
95              scanf("%d", &x);
96              v[i].push_back(x);
97              w[i].push_back(-1);
98          }
99      }
100
101     for(int i=1; i<=m; ++i)
102     {
103         int x, y, t;
104         scanf("%d%d%d", &x, &y, &t);
105         --y; //Daca un preparat merge spre masa y atunci
106         // el vine dinspre masa y-1
107
108         if(y==0)
109         {
110             if(x==1)
111                 y=v[x].size()-1;
112             else
113                 y=v[x].size();
114         }
115
116         if(x!=1)
117             w[x][y-1]=max(w[x][y-1], t);
118         else
119             w[x][y]=max(w[x][y], t);
120     }
121
122     df(1);
123
124     --poz;
125
126     for(int i=1; i<=poz; ++i)
127     {
128         euler[i+poz]=euler[i];
129         hasPrep[i+poz]=hasPrep[i];
130         whenPrep[i+poz]=whenPrep[i];
131     }
132
133     poz *= 2;
134
135     for(int i=1; i<=poz; ++i)
136     {
137         int nod = euler[i];
138         prevPos[i]=lastOc[nod];
139         lastOc[nod]=i;
140     }

```

```

141     build(1, 1, poz);
142
143     for(int i=poz/2+1; i<=poz; ++i)
144     {
145         int nod = euler[i];
146         int prevOc = prevPos[i];
147         int prep = query(1, 1, poz, prevOc+1, i);
148
149         sol[nod]=max(sol[nod], i+prep);
150     }
151
152     for(int i=1; i<=n; ++i)
153         printf("%d ", sol[i]);
154     printf("\n");
155
156     return 0;
157 }

```

22.2.3 *Rezolvare detaliată

22.3 xor

Problema 3 - xor

100 de puncte

Se consideră o matrice cu un număr infinit de linii și coloane indexate începând cu 0.

Pe prima linie matricea conține sirul numerelor naturale (0, 1, 2, 3, ...).

Pe fiecare linie începând cu linia a doua pe poziția j matricea conține suma xor a elementelor situate pe linia anterioară de la poziția 0 până la poziția j .

Cerințe

Se cere să se răspundă la q întrebări de forma "Pentru i și j date, să se determine numărul situat pe linia i coloana j a matricei". Pentru a genera cele q întrebări vor fi cunoscute următoarele valori: i_1, j_1, a, b, m .

i_1, j_1 reprezintă valorile pentru prima întrebare. Următoarele întrebări i_k, j_k vor fi generate una din alta folosind următoarea regulă:

$$i_k = (a * i_{k-1} + b) \mod m$$

$$j_k = (a * j_{k-1} + b) \mod m$$

Date de intrare

Fișierul de intrare **xor.in** conține pe prima linie numerele naturale q, i_1, j_1, a, b, m , separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire **xor.out** va conține q linii. Pe linia k se va afișa elementul situat pe linia i_k coloana j_k a matricei.

Restricții și precizări

- Pentru 10% dintre teste $1 \leq q \leq 100, 1 \leq m \leq 100$
- Pentru alte 10% dintre teste $1 \leq q \leq 100\ 000, 1 \leq m \leq 1\ 000$
- Pentru alte 30% dintre teste $1 \leq q \leq 50, 1 \leq m \leq 30\ 000$
- Pentru alte 50% dintre teste $1 \leq q \leq 100\ 000, 1 \leq m \leq 100\ 000\ 000$
- $0 \leq i_1, j_1 \leq m$
- $1 \leq a, b \leq 9$

Exemplu:

xor.in	xor.out	Explicații
4 2 3 1 1 5	2	Avem $q = 4$ întrebări.
	7	Pentru $i_1 = 2, j_1 = 3, a = 1, b = 1, m = 5$ se obțin întrebările: (2,3), (3,4), (4,0), (0,1)
	0	Matricea este:
	1	$\begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & \dots \\ 0 & 1 & 3 & 0 & 4 & 1 & 7 & \dots \\ 0 & 1 & 2 & 2 & 6 & 7 & 0 & \dots \\ 0 & 1 & 3 & \underline{1} & 7 & 0 & 0 & \dots \\ 0 & 1 & 2 & 3 & \underline{4} & 4 & 4 & \dots \end{matrix}$ <p>...</p> <p>Se observă că pe pozițiile corespunzătoare întrebărilor avem valorile 2,7,0 și 1</p>

Timp maxim de executare/test: **1.5** secunde pe Windows, **0.5** secunde pe Linux

Memorie: total **16 MB**

Dimensiune maximă a sursei: **10 KB**

22.3.1 Indicații de rezolvare

Adrian Panaete, Colegiul Național "A. T. Laurian" Botoșani |index[person]Adrian Panaete

Toate soluțiile se referă la rezolvarea unui singur query.

Solutia 1. Complexitate $O(m^2 + q)$

Se precalculează matricea până la linia m , coloana m . Apoi se răspunde la fiecare întrebare în timp constant.

Solutia 2 de 60 p. Complexitate $O(q * m)$.

Urmărim de câte ori apare fiecare număr de la 1 la j în rezultat. Să luăm de exemplu $i = 3, j = 5$. Pentru 1 avem 0 apariții

0 1 0 0 0
0 1 1 0 0
0 1 2 1 0
0 1 3 3 1 0.

Pentru 2 avem o apariție deci 2 apare o dată în rezultat

0 0 1 0 0
0 0 1 1 0
0 0 1 2 1
0 0 1 3 3 1.

Pentru 3 avem 3 apariții deci 3 apare de 3 ori în rezultat

0 0 0 1 0
0 0 0 1 1
0 0 0 1 2
0 0 0 1 3 3.

Pentru 4 avem 3 apariții deci 4 apare de 3 ori în rezultat

0 0 0 0 1
0 0 0 0 1
0 0 0 0 2
0 0 0 0 3 3.

Pentru 5 avem o apariție deci 2 apare o dată în rezultat

0 0 0 0 0 1
0 0 0 0 0 1
0 0 0 0 0 1
0 0 0 0 0 1 1.

Folosind proprietatea $a \text{ xor } a = 0$ ne interesează doar restul modulo 2 al *numărului de apariții*.

În cazul nostru rezultatul este $2 \text{ xor } 3 \text{ xor } 4 \text{ xor } 5 = 0$

În general *numărul de apariții* a unei valori k pe linia i coloana j este C_i^{j-k} care se ia 0 dacă $j - k > i$.

Ne interesează doar *paritatea* acestei *combinări* deci este suficient să parcurgem valorile de la j la 1 și să actualizăm numărul de factori 2 care apar la numitor și la numărător pentru trecerea

de la o valoare la valoarea imediat mai mică. Se obține un algoritm de complexitate $O(j)$ pentru fiecare întrebare.

Solutia de 100 p. Complexitate $O(q * \log m)$ sau $O(q * \log^2 m)$

Folosim ideea de la soluția de 60 de puncte dar facem observația suplimentară că rezultatul poate fi calculat la nivelul fiecărui bit (ceea ce adaugă un factor $\log m$ la fiecare întrebare) .

Se observă că prezența sau nu în rezultatul final a fiecărui bit respectă o regulă foarte clară - mai precis numărul de apariții se dovedește a fi tot o combinare modulo 2.

Calculul combinărilor modulo 2 poate fi realizat cu diverse complexități (patratice, liniar, logarithmic, constant). Cele mai rapide metode de calcul vor fi prezentate în continuare.

Detalii:

Fie matricea infinită $s[i, j] = C_{i+j}^j$ care poate fi construită cu formula de recurență:

$$s[i, j] = s[i - 1, j] \wedge s[i, j - 1]$$

$$s[i, 0] = s[0, j] = 1$$

Matricea este foarte des întâlnită în probleme și aplicații de informatică sub denumirea de "triunghiul Serpinski".

Dacă vom construi parțial matricea vom obține:

```

1111111111111111 ... L0
1010101010101010 ... L1
1100110011001100 ... L2
1000100010001000
1111000011110000 ... L4
1010000010100000
1100000011000000
1000000010000000
1111111100000000 ... L8
1010101000000000
1100110000000000
1000100000000000
1111000000000000
1010000000000000
1100000000000000
1000000000000000
1111111111111111 ... L16

```

Proprietăți ale "triunghiului Serpinski"

1. $s[i, j] = C_{i+j}^i \bmod 2$
2. $s[i, j] = s[j, i]$ pentru $i, j \geq 0$
3. $s[i, j] = s[i - 1, j] + s[i, j - 1]$ pentru $i, j \geq 1$
4. $s[i, j] = s[i - 2^p, j]$ pentru $i \in [2^p, 2^{p+1})$ și $j \in [0, 2^p)$
5. $s[i, j] = 0$ pentru $i \& j \neq 0$
6. $s[i, j] = 1$ pentru $i \& j = 0$
7. $s[i, j] = 1$ dacă există p astfel încât $i + j = 2^p - 1$
8. $s[2^p + j, 2^p] = s[2^p, 2^p + j] = 0$ pentru $j \in [0, 2^p)$
9. $s[i, j] = 0$ pentru $i, j \in [2^p, 2^{p+1})$

Toate aceste proprietăți pot fi deduse fie din proprietățile generale ale combinărilor fie direct analizând puterea la care apare factorul prim 2 în combinări.

Proprietatea 3. ne permite să calculăm valorile matricei în timp patratic.

Dacă folosim formula combinărilor un element oarecare se poate calcula liniar folosind *inversul modular*.

Proprietățile 3 și 8 ne permit să calculăm un element în timp logarithmic.

Proprietățile 4 și 5 ne permit să calculăm un element în timp constant.

Să notăm cu $bp[]$ tabloul definit astfel.

$bp[i][j] = 1$ dacă soluția problemei pentru poziția (i, j) conține bitul p (mai precis conține valoarea 2^p în descompunerea unică în sumă de puteri ale lui 2).

$bp[i][j] = 0$ dacă soluția problemei pentru poziția (i, j) nu conține bitul p

în particular $bp[0][j]$ este 1 sau 0 după cum j conține sau nu conține bitul p .

Din modul cum e definit tabloul din enunț se deduce că:

$$bp[i][j] = bp[i - 1][j] \wedge bp[i][j - 1]$$

adică respectă aceeași relație de recurență ca matricea s .

Este foarte ușor de observat că un număr j conține bitul p dacă și numai dacă $j \pm 2^p$ nu conține bitul p .

De aceea avem $bp[0][j] = 1 \Leftrightarrow j$ conține bitul $p \Leftrightarrow j \pm 2^p$ nu conține bitul $p \Leftrightarrow 2^p \& (j \pm 2^p) = 0 \Leftrightarrow s[2^p][j \pm 2^p] = 1$.

Analog $bp[0][j] = 0 \Leftrightarrow s[2^p][j \pm 2^p] = 0$

Concluzia este că $bp[0][j] = s[2^p][j \pm 2^p] = 0$.

Pentru $j < 2^p$ avem evident $bp[i][j] = 0$.

Pentru $j \geq 2^p$ folosind recurențele se va deduce în final $bp[i][j] = s[i + 2^p][j - 2^p]$.

Acum avem o metodă prin care putem calcula bit cu bit soluția pentru orice pozitie folosind "triunghiul Serpinski".

Soluția va fi sau pe biți din valorile $2^p * s[i + 2^p][j - 2^p]$ pentru toate valorile lui p cu $2^p \leq j$.

22.3.2 Cod sursă

Listing 22.3.1: denis_brut.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 ifstream f("xor.in");
6 ofstream g("xor.out");
7
8 using namespace std;
9
10 int q, i, j, a, b, m, p;
11
12 int solve(int i, int j, int p)
13 {
14     if (!i)
15         return j;
16
17     if (!j)
18         return 0;
19
20     while (p > i || p > j)
21         p >>= 1;
22
23     return solve(i - p, j, p) ^ solve(i, j - p, p);
24 }
25
26 int main ()
27 {
28     f >> q >> i >> j >> a >> b >> m;
29     while (q -- )
30     {
31         p = 1;
32         while (p <= i && p <= j)
33             p <<= 1;
34
35         p >>= 1;
36         g << solve(i ,j, p) << "\n";
37         i = (a * i + b) % m;
38         j = (a * j + b) % m;
39     }
40
41     return 0;
42 }
```

Listing 22.3.2: denis_brut_precalculare.cpp

```

1 #include <iostream>
2 #include <string.h>
3
```

```

4  using namespace std;
5
6  ifstream f("xor.in");
7  ofstream g("xor.out");
8
9  int q, i, j, a, b, m;
10
11 int main ()
12 {
13     f >> q >> i >> j >> a >> b >> m;
14     int r[m][m];
15     memset(r, 0, sizeof(r));
16     for (int i = 1; i < m; ++i)
17         r[0][i] = i;
18
19     for (int i = 1; i < m; ++i)
20         for (int j = 1; j < m; ++j)
21             r[i][j] = r[i - 1][j] ^ r[i][j - 1];
22
23     while (q -- )
24     {
25         g << r[i][j] << "\n";
26         i = (a * i + b) % m;
27         j = (a * j + b) % m;
28     }
29
30     return 0;
31 }
```

Listing 22.3.3: oficiala.cpp

```

1 // 100 de puncte O(Q * log N)
2 #include <iostream>
3
4 using namespace std;
5
6 ifstream f("xor.in");
7 ofstream g("xor.out");
8
9 int q,i,j,p,P,s,a,b,m;
10
11 int main()
12 {
13     for(f>>q>>i>>j>>a>>b>>m;
14         q;
15         g<<s<<'\n', q--, i=(a*i+b)%m, j=(a*j+b)%m
16         )
17
18     for(p=0,P=1,s=0; P<=j; p++,P<<=1)
19         s|=((i+P)&(j-P))==0<<p;
20
21     return 0;
22 }
```

Listing 22.3.4: optim.cpp

```

1 //100 de puncte, O(Q * log^2 N)
2 #include <iostream>
3
4 #define SOL 1
5 #define BRUT 0
6 #define OPTIM 1
7 #define SUB_OPTIM 3
8 #define COMB 2
9 #define SUB_COMB 4
10 #define C(u,v) (((u)&(v))==0)
11
12 using namespace std;
13
14 ifstream f("xor.in");
15 ofstream g("xor.out");
16
17 void brut();
18 void optim();
```

```

19 void comb();
20 void sub_optim_log();
21 void sub_comb_log();
22 int n,m,i,j,k,q,b,sol,A,B,mod,aux,x[1020][1020],Clog(int,int);
23
24 int main()
25 {
26     if(SOL==BRUT)brut();
27     if(SOL==SUB_OPTIM)sub_optim_log();
28     if(SOL==OPTIM)optim();
29     if(SOL==SUB_COMB)sub_comb_log();
30     if(SOL==COMB)comb();
31     return 0;
32 }
33
34 void brut()
35 {
36     f>>q>>n>>m>>A>>B>>mod;
37     for(j=1;j<mod;j++)
38         x[0][j]=j;
39     for(i=1;i<mod;i++)
40         for(j=1;j<mod;j++)
41             x[i][j]=x[i-1][j]^x[i][j-1];
42     for(;q;q--)
43     {
44         g<<x[n][m]<<'\\n';
45         m=(1LL*A*m+B)%mod;
46         n=(1LL*A*n+B)%mod;
47     }
48 }
49
50 void optim()
51 {
52     f>>q>>n>>m>>A>>B>>mod;
53     for(;q;q--)
54     {
55         for(b=1,sol=0;b<=(1<<30)&&m>=b;b<<=1)
56             if(C(m-b,n+b))
57                 sol|=b;
58         g<<sol<<'\\n';
59         m=(1LL*A*m+B)%mod;
60         n=(1LL*A*n+B)%mod;
61     }
62 }
63
64 void sub_optim_log()
65 {
66     f>>q>>n>>m>>A>>B>>mod;
67     for(;q;q--)
68     {
69         for(b=1,sol=0;b<=(1<<30)&&m>=b;b<<=1)
70             if(Clog(m-b,n+b))
71                 sol|=b;
72         if(sol)g<<sol;
73         m=(1LL*A*m+B)%mod;
74         n=(1LL*A*n+B)%mod;
75     }
76 }
77
78 void comb()
79 {
80     f>>q>>n>>m>>A>>B>>mod;
81     for(;q;q--)
82     {
83         for(j=0,k=m,sol=0;k;j++,k--)
84             if(C(n-1,j))
85                 sol^=k;
86         if(sol)g<<sol;
87         m=(1LL*A*m+B)%mod;
88         n=(1LL*A*n+B)%mod;
89     }
90 }
91
92 void sub_comb_log()
93 {
94     f>>q>>n>>m>>A>>B>>mod;

```

```

95     for(;q;q--)
96     {
97         for(j=0,k=m,sol=0;k;j++,k--)
98             if(Clog(n-1,j))
99                 sol^=k;
100            if(sol)g<<sol;
101            m=(1LL*A*m+B)%mod;
102            n=(1LL*A*n+B)%mod;
103        }
104    }
105
106 int Clog(int u,int v)
107 {
108     if(u==0||v==0) return 1;
109     int bu=u&(-u),bv=v&(-v);
110     if(bu==bv) return 0;
111     if(bu<bv) return Clog(u-bu,v);
112     return Clog(u,v-bv);
113 }
```

Listing 22.3.5: radu_oficial.cpp

```

1 // 100 de puncte O(Q * log^2 N)
2 #include <iostream>
3 #include <cmath>
4 #include <algorithm>
5 #include <cassert>
6
7 using namespace std;
8
9 int q, line, col, a, b, mod;
10
11 int getPow(int n)
12 {
13     int pw = 0;
14     while (n) pw += n, n /= 2;
15     return pw;
16 }
17
18 int getCombParity(int n, int k)
19 {
20     if (k < 0 || n < 0 || n < k) return 0;
21     int pwUp = getPow(n), pwDown = getPow(k) + getPow(n - k);
22     if (pwUp > pwDown) return 0;
23     return 1;
24 }
25
26 int main(int argc, char **argv)
27 {
28     freopen(argv[1], "r", stdin);
29     freopen(argv[2], "w", stdout);
30
31     scanf("%i %i %i %i %i %i", &q, &line, &col, &a, &b, &mod);
32
33     for (int i = 1; i <= q; ++ i)
34     {
35         int ans = 0;
36         for (int j = 0; (1 << j) <= col; ++ j)
37         {
38             int n = line + (1 << j);
39             int k = col - (1 << j);
40             ans ^= ((1 << j) * getCombParity(n + k, k));
41         }
42         printf("%d\n", ans);
43         line = (a * line + b) % mod;
44         col = (a * col + b) % mod;
45     }
46
47     return 0;
48 }
```

Listing 22.3.6: radu_task3.cpp

```
1 //âži50 de puncte
```

```

2 #include <cstdio>
3 #include <cstdlib>
4 #include <algorithm>
5 #include <cassert>
6
7 using namespace std;
8
9 int q, line, col, a, b, mod;
10
11 int getPow(int n)
12 {
13     int pw = 0;
14     while (n) pw += n, n /= 2;
15     return pw;
16 }
17
18 int getCombParity(int n, int k)
19 {
20     if (k < 0 || n < 0 || n < k) return 0;
21     int pwUp = getPow(n), pwDown = getPow(k) + getPow(n - k);
22     if (pwUp > pwDown) return 0;
23     return 1;
24 }
25
26 int main(int argc, char **argv)
27 {
28     freopen(argv[1], "r", stdin);
29     freopen(argv[2], "w", stdout);
30
31     scanf("%i %i %i %i %i %i", &q, &line, &col, &a, &b, &mod);
32
33     for (int i = 1; i <= q; ++ i)
34     {
35         int ans = 0;
36         for (int j = 0; j <= col; ++ j)
37             ans ^= j * getCombParity(line + col - 1 - j, col - j);
38         printf("%d\n", ans);
39         line = (a * line + b) % mod;
40         col = (a * col + b) % mod;
41     }
42
43     return 0;
44 }
```

Listing 22.3.7: radu_task12.cpp

```

1 //20 de puncte
2 #include <cstdio>
3 #include <cstdlib>
4 #include <algorithm>
5 #include <cassert>
6
7 using namespace std;
8
9 const int nmax = 1010;
10
11 int q, line, col, a, b, mod;
12 int mat[nmax][nmax];
13
14 int main(int argc, char **argv)
15 {
16     freopen(argv[1], "r", stdin);
17     freopen(argv[2], "w", stdout);
18
19     scanf("%i %i %i %i %i %i", &q, &line, &col, &a, &b, &mod);
20
21     assert(1 <= q && q <= 100000);
22     assert(1 <= mod && mod <= 1000);
23     assert(0 <= line && line < mod);
24     assert(0 <= col && col < mod);
25     assert(1 <= a && a <= 9);
26     assert(1 <= b && b <= 9);
27
28     for (int i = 0; i < mod; ++ i)
29         mat[0][i] = i;
```

```

30
31     for (int i = 1; i < mod; ++ i)
32         for (int j = 0; j < mod; ++ j)
33     {
34         if (j > 0) mat[i][j] = mat[i][j - 1];
35         mat[i][j] ^= mat[i - 1][j];
36     }
37
38     for (int i = 1; i <= q; ++ i)
39     {
40         printf("%d\n", mat[line][col]);
41         line = (a * line + b) % mod;
42         col = (a * col + b) % mod;
43     }
44
45     return 0;
46 }
```

22.3.3 *Rezolvare detaliată

22.4 arbore

Problema 4 - arbore

100 de puncte

Se dă un arbore (graf conex aciclic) cu N noduri. Vrem să eliminăm noduri (împreună cu muchiile adiacente) din arborele dat, astfel încât numărul de componente conexe ale grafului rămas să fie maxim. Aflați care este numărul maxim de componente conexe pe care le putem obține și câte submulțimi distințe de noduri se pot elimina din arbore astfel încât să rămână la final acest număr maxim de componente conexe.

Cerințe

Date de intrare

Pe prima linie a fișierului de intrare **arbore.in** se va afla numărul natural N , reprezentând numărul de noduri ale arborelui. Pe următoarele $N - 1$ linii se vor afla câte două numere X și Y , cu semnificația că există o muchie între nodurile X și Y .

Date de ieșire

Pe prima linie a fișierului de ieșire **arbore.out** se vor afișa două numere naturale reprezentând numărul maxim de componente conexe pe care îl putem obține, respectiv numărul de moduri în care putem obține acest număr de componente conexe modulo $10^9 + 7$ (adică restul împărțirii acestui număr la 1 000 000 007)

Restricții și precizări

- $1 \leq N \leq 100\ 000$
- Se acordă 40% din punctajul unui test dacă numărul maxim de componente conexe este corect.
 - Se acordă 60% din punctajul unui test dacă numărul de moduri este corect.
 - Pentru 20% din teste $N \leq 20$
 - Pentru alte 30% din teste $N \leq 1000$

Exemple:

arbore.in	arbore.out	Explicații
6 1 2 1 3 1 4 4 5 4 6	4 1	Se șterg nodurile 1 și 4. Nicio altă submulțime de noduri șterse nu produce 4 sau mai multe componente conexe.
4 1 2 2 3 3 4	2 5	Se pot șterge următoarele submulțimi de noduri pentru a obține 2 componente conexe: {2}, {3}, {2, 3}, {2, 4}, {1, 3}

Timp maxim de executare/test: **1.0** secunde pe Windows, **0.5** secunde pe Linux

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

Dimensiune maximă a sursei: **20 KB**

22.4.1 Indicații de rezolvare

Denis-Gabriel Mită, Universitatea din București

Soluție $O(2^N * N)$ - 20p

Ne setăm fiecare submulțime posibilă de noduri ce vrem să le eliminăm și verificăm numărul de componente conexe cu o parcurgere în adâncime / lățime. Complexitatea finală este $O(2^N * N)$.

Soluție $O(N)$ prima cerință - 40p

Pentru a afla numarul maxim de componente conexe putem face observația că la orice pas nu are rost să ștergem o frunză, deoarece numărul de componente conexe nu va crește.

În schimb, dacă am elibera vecinul lor, toate frunzele legate de acesta vor forma componente conexe independente. Dacă nu l-am elibera, în cazul în care acesta ar avea legături cu cel puțin 2 frunze, eliberaându-l ne-ar crește numărul de componente conexe, iar neeliberaându-l numărul de componente conexe nu ar putea crește față de cazul în care l-am elibera.

Astfel, introducem nodurile cu gradul 1 într-o coadă, iar la fiecare pas scoatem din coadă un nod cu grad 1, dacă nu este șters incrementăm numărul maxim de componente conexe, iar apoi ștergem vecinul său și reintroducem în coadă nodurile nou create cu grad 1. Complexitatea finală este $O(N)$.

Soluție $O(N)$ - 100p

Ne fixăm arborele într-o rădăcină (spre exemplu nodul 1) și apoi ne construim următoarea dinamică printr-o parcurgere în adâncime:

$nr[0][nod]$ = numărul maxim de componente conexe rezultate din subarborele nodului nod dacă NU eliminăm nodul nod

$nr[1][nod]$ = numărul maxim de componente conexe rezultate din subarborele nodului nod dacă eliminăm nodul nod

$dp[0][nod]$ = numărul de moduri în care obținem $nr[0][nod]$ componente conexe din subarborele nodului nod dacă NU eliminăm nodul nod

$dp[1][nod]$ = numărul de moduri în care obținem $nr[1][nod]$ componente conexe din subarborele nodului nod dacă eliminăm nodul nod

Recurrența este următoarea:

$$nr[0][nod] = 1 + \sum \text{din } \max(nr[1][fiu], nr[0][fiu]) - 1$$

$$nr[1][nod] = \sum \text{din } \max(nr[1][fiu], nr[0][fiu])$$

$$dp[0][nod] = 1 * \prod \text{din : } dp[1][fiu] + dp[0][fiu] \text{ dacă } nr[1][fiu] = nr[0][fiu] - 1 \\ dp[1][fiu] \text{ dacă } nr[1][fiu] > nr[0][fiu] - 1$$

$$dp[0][fiu] \text{ dacă } nr[1][fiu] < nr[0][fiu] - 1$$

$$dp[1][nod] = 1 * \prod \text{din : } dp[1][fiu] + dp[0][fiu] \text{ dacă } nr[1][fiu] = nr[0][fiu] \\ dp[1][fiu] \text{ dacă } nr[1][fiu] > nr[0][fiu]$$

$$dp[0][fiu] \text{ dacă } nr[1][fiu] < nr[0][fiu]$$

Soluția pentru prima cerință se va fi $\max(nr[0][1], nr[1][1])$, iar pentru a doua cerință va fi

$dp[0][1]$ dacă $nr[0][1] > nr[1][1]$, în $dp[1][1]$ dacă $nr[1][1] > nr[0][1]$, sau $dp[0][1] + dp[1][1]$ dacă $nr[0][1] = nr[1][1]$. Complexitatea finală este $O(N)$.

22.4.2 Cod sursă

Listing 22.4.1: arboreAH.cpp

```

1 /**
2  * Author: Andrei Heidelbacher
3  * Time complexity: O(N)
4  * Space complexity: O(N)
5 */
6
7 #include <fstream>
8 #include <vector>
9 #include <algorithm>
10
11 using namespace std;
12
13 typedef vector<int>::const_iterator edge;
14
15 const char IN_FILE[] = "arbore.in";
16 const char OUT_FILE[] = "arbore.out";
17 const int NIL = -1;
18 const int MOD = 1000000007;
19
20 vector<vector<int>> Tree;
21 int V, Root;
22 vector<vector<int>> ValueDP, CountDP;
23
24 inline void multiply(int &a, const int b)
25 {
26     a = (1LL * a * b) % MOD;
27 }
28
29 void dfs(const int x, const int father = NIL)
30 {
31     ValueDP[x][0] = 1;
32     ValueDP[x][1] = 0;
33     CountDP[x][0] = CountDP[x][1] = 1;
34
35     for (edge y = Tree[x].begin(); y != Tree[x].end(); ++y)
36     {
37         if (*y != father)
38         {
39             dfs(*y, x);
40
41             ValueDP[x][0] += max(ValueDP[*y][0] - 1, ValueDP[*y][1]);
42             if (ValueDP[*y][0] - 1 > ValueDP[*y][1])
43                 multiply(CountDP[x][0], CountDP[*y][0]);
44             else if (ValueDP[*y][1] > ValueDP[*y][0] - 1)
45                 multiply(CountDP[x][0], CountDP[*y][1]);
46             else
47                 multiply(CountDP[x][0], CountDP[*y][0] + CountDP[*y][1]);
48
49             ValueDP[x][1] += max(ValueDP[*y][0], ValueDP[*y][1]);
50             if (ValueDP[*y][0] > ValueDP[*y][1])
51                 multiply(CountDP[x][1], CountDP[*y][0]);
52             else if (ValueDP[*y][1] > ValueDP[*y][0])
53                 multiply(CountDP[x][1], CountDP[*y][1]);
54             else
55                 multiply(CountDP[x][1], CountDP[*y][0] + CountDP[*y][1]);
56         }
57     }
58 }
59
60 void solve()
61 {
62     ValueDP = CountDP = vector<vector<int>>(V, vector<int>(2));
63     dfs(Root);
64 }
65
66 inline void addEdge(const int x, const int y)
67 {
68     Tree[x].push_back(y);
69     Tree[y].push_back(x);
70 }
```

```

71 void read()
72 {
73     ifstream in(IN_FILE);
74     in >> V;
75     Tree = vector<vector<int>>(V, vector<int>());
76     for (int i = 0; i < V - 1; ++i)
77     {
78         int x, y;
79         in >> x >> y;
80         addEdge(x - 1, y - 1);
81     }
82 }
83
84 Root = 0;
85 in.close();
86 }
87
88 void write()
89 {
90     ofstream out(OUT_FILE);
91     int value = max(ValueDP[Root][0], ValueDP[Root][1]), count = 0;
92
93     if (ValueDP[Root][0] > ValueDP[Root][1])
94         count = CountDP[Root][0];
95     else if (ValueDP[Root][1] > ValueDP[Root][0])
96         count = CountDP[Root][1];
97     else
98         count = (CountDP[Root][0] + CountDP[Root][1]) % MOD;
99
100    out << value << " " << count << "\n";
101    out.close();
102 }
103
104 int main()
105 {
106     read();
107     solve();
108     write();
109     return 0;
110 }
```

Listing 22.4.2: arboreDM1.cpp

```

1 #include <iostream>
2 #include <vector>
3 #define N 100100
4 #define mod 1000000007
5
6 using namespace std;
7
8 ifstream cin("arbore.in");
9 ofstream cout("arbore.out");
10
11 vector<int> v[N];
12 int D[3][N], x, y, n;
13 long long W[3][N];
14
15 void dfs(int x, int p)
16 {
17     D[1][x] = 1;
18     D[0][x] = 0;
19     D[2][x] = 1;
20     W[1][x] = 1;
21     W[0][x] = 1;
22     W[2][x] = 1;
23
24     for (auto &it : v[x])
25         if (it != p)
26             dfs(it, x);
27
28     // Cazul 1. Stergem nodul
29     for (auto &it : v[x])
30         if (it != p)
31         {
32             D[0][x] += D[2][it];
33         }
34 }
```

```

33         W[0][x] *= W[2][it];
34         W[0][x] %= mod;
35     }
36
37 // Cazul 2. Nu stergem nodul
38 for (auto &it : v[x])
39 {
40     if (it != p)
41     {
42         if (D[0][it] + 1 == D[1][it])
43         {
44             W[1][x] *= (W[0][it] + W[1][it]) % mod;
45             W[1][x] %= mod;
46             D[1][x] += D[0][it];
47         } else if (D[0][it] + 1 < D[1][it])
48         {
49             W[1][x] *= W[1][it];
50             W[1][x] %= mod;
51             D[1][x] += D[1][it] - 1;
52         } else
53         {
54             W[1][x] *= W[0][it];
55             W[1][x] %= mod;
56             D[1][x] += D[0][it];
57         }
58     }
59 }
60
61 // Calculam solutia pentru acest subarbore stiind cele 2 cazuri
62 D[2][x] = max(D[1][x], D[0][x]);
63 if (D[1][x] == D[0][x])
64 {
65     W[2][x] = (W[1][x] + W[0][x]) % mod;
66 } else if (D[1][x] > D[0][x])
67 {
68     W[2][x] = W[1][x];
69 } else
70 {
71     W[2][x] = W[0][x];
72 }
73 }
74
75 int main ()
76 {
77     cin >> n;
78     for (int i = 1; i < n; ++i)
79     {
80         cin >> x >> y;
81         v[x].push_back(y);
82         v[y].push_back(x);
83     }
84     dfs(1, 0);
85     cout << D[2][1] << " " << W[2][1];
86     return 0;
87 }
```

Listing 22.4.3: arboreRS.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 ifstream f("arbore.in");
6 ofstream g("arbore.out");
7
8 #define pb push_back
9 #define ll long long
10 #define mp make_pair
11
12 const int nmax = 100005;
13 const int mod = 1e9 + 7;
14 const int RAMANE = 0;
15 const int ILSCOT = 1;
16
17 int viz[nmax], dp[nmax][3], cnt[nmax][3];
```

```

18 vector<int> gf[nmax];
19 int t[nmax];
20 int n;
21
22 void update(int &x)
23 {
24     if (x >= mod)
25         x -= mod;
26 }
27
28 void dfs(int node)
29 {
30     viz[node] = 1;
31     int nrFii = 0;
32     for (int i = 0; i < gf[node].size(); ++i)
33     {
34         int vc = gf[node][i];
35         if (viz[vc] == 0)
36         {
37             ++nrFii;
38             t[vc] = node;
39             dfs(vc);
40         }
41     }
42
43     if (nrFii == 0)
44     {
45         // e frunza;
46         dp[node][RAMANE] = 1;
47         cnt[node][RAMANE] = 1;
48         dp[node][ILSCOT] = 0;
49         cnt[node][ILSCOT] = 1;
50         return;
51     }
52
53     int maxComp = 1;
54     cnt[node][ILSCOT] = 1;
55     cnt[node][RAMANE] = 1;
56     for (int i = 0; i < gf[node].size(); ++i)
57     {
58         int vc = gf[node][i];
59         if (t[node] != vc)
60         {
61             maxComp += max(dp[vc][RAMANE] - 1, dp[vc][ILSCOT]);
62             dp[node][ILSCOT] += max(dp[vc][RAMANE], dp[vc][ILSCOT]);
63             update(maxComp);
64
65             int ce[2];
66
67             if (dp[vc][RAMANE] == dp[vc][ILSCOT])
68                 ce[ILSCOT] = (cnt[vc][RAMANE] + cnt[vc][ILSCOT]) % mod;
69             else
70                 if (dp[vc][RAMANE] > dp[vc][ILSCOT])
71                     ce[ILSCOT] = cnt[vc][RAMANE];
72                 else
73                     ce[ILSCOT] = cnt[vc][ILSCOT];
74
75             if (dp[vc][RAMANE] - 1 == dp[vc][ILSCOT])
76                 ce[RAMANE] = (cnt[vc][RAMANE] + cnt[vc][ILSCOT]) % mod;
77             else
78                 if (dp[vc][RAMANE] - 1 > dp[vc][ILSCOT])
79                     ce[RAMANE] = cnt[vc][RAMANE];
80                 else
81                     ce[RAMANE] = cnt[vc][ILSCOT];
82
83             cnt[node][ILSCOT] = (1LL * cnt[node][ILSCOT] * ce[ILSCOT]) % mod;
84             cnt[node][RAMANE] = (1LL * cnt[node][RAMANE] * ce[RAMANE]) % mod;
85         }
86     }
87
88     dp[node][RAMANE] = maxComp;
89 }
90
91 int main()
92 {
93     f >> n;

```

```

94     for (int i = 2; i <= n; ++i)
95     {
96         int x, y;
97         f >> x >> y;
98         gf[x].pb(y);
99         gf[y].pb(x);
100    }
101
102    dfs(1);
103    int node = 1;
104    g << max(dp[node][RAMANE], dp[node][ILSCOT]) << "\n";
105
106    if (dp[node][RAMANE] == dp[node][ILSCOT])
107        g << (cnt[node][RAMANE] + cnt[node][ILSCOT]) % mod << "\n";
108    else
109        if (dp[node][RAMANE] > dp[node][ILSCOT])
110            g << cnt[node][RAMANE] << "\n";
111        else
112            g << cnt[node][ILSCOT] << "\n";
113
114    return 0;
115 }
```

22.4.3 *Rezolvare detaliată

22.5 calafat

Problema 5 - calafat

100 de puncte

Această problemă se numește Calafat pentru că a fost compusă în timpul excursiei la Calafat de mâine.

Cerințe

Se dă un sir format din N numere naturale. Pentru fiecare valoare distinctă dintr-o subsecvență cuprinsă între 2 indici st și dr considerăm distanța dintre indicii primei și ultimei apariții ale acesteia în cadrul subsecvenței.

Dându-se M subsecvențe de forma $[st, dr]$, se cere să se calculeze suma distanțelor corespunzătoare tuturor lor distincte din subsecvență.

Date de intrare

Pe prima linie a fișierului de intrare **calafat.in** se vor afla două numere naturale N și M . Pe a doua linie se vor afla cele N numere din sirul dat. Pe următoarele M linii se vor afla câte două numere st și dr , cu semnificația că vrem să calculăm suma menționată mai sus pentru subsecvența $[st, dr]$.

Date de ieșire

În fișierul de ieșire **calafat.out** se vor afișa M numere naturale, câte unul pe fiecare linie, reprezentând cele M sume cerute.

Restricții și precizări

- $1 \leq N, M \leq 200\ 000$
- $1 \leq st \leq dr \leq N$
- Valorile din sir se vor afla în intervalul $[1, N]$
- Pentru 20% din teste se garantează că $N, M \leq 1000$
- Pentru alte 25% din teste se garantează că $N, M \leq 35\ 000$ iar numărul de elemente distincte din sir este maxim 100.
- Pentru alte 25% din teste se garantează că $N, M \leq 70\ 000$

Exemplu:

calafat.in	calafat.out	Explicații
7 3	0	În prima subsecvență fiecare valoare apare o singură dată, deci suma diferențelor este 0.
1 3 1 2 2 1 3	9	În a doua subsecvență:
2 4	4	Valoarea 3 apare la indicii 2 și 7 rezultând diferența $7-2=5$ Valoarea 1 apare la indicii 3 și 6 \Rightarrow diferența $6-3=3$ Valoarea 2 apare la indicii 4 și 5 \Rightarrow diferența $5-4=1$ Suma diferențelor este 9.
2 7		În a treia subsecvență:
3 6		Valoarea 1 apare la indicii 3 și 6 \Rightarrow diferența $6-3=3$ Valoarea 2 apare la indicii 4 și 5 \Rightarrow diferența $5-4=1$ Suma diferențelor este 4.

Timp maxim de executare/test: **2.5** secunde pe Windows, **1.5** secunde pe Linux

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

Dimensiune maximă a sursei: **20 KB**

22.5.1 Indicații de rezolvare

Denis-Gabriel Mită, Universitatea din București

Soluție $O(N^2 + M)$ / $O(N * M)$ - **20-30p**

Putem precalcula răspunsurile pentru fiecare subsecvență posibilă și răspunde la queryuri în $O(1)$.

Ne fixăm capătul stânga și incrementăm capătul dreapta al intervalului, menținând prima și ultima apariție a fiecărei valori, actualizând răspunsul la incrementarea capătului dreapta în $O(1)$.

Complexitatea acestei soluții este $O(N^2 + M)$.

Analog putem calcula soluția pentru un query în $O(N)$ cu o parcurgere a elementelor din subsecvență.

Complexitatea acestei soluții este $O(N * M)$.

Soluție $O(N + K * M * \log N)$ - **45p**

Definim K ca fiind numărul de valori distincte din sir.

Astfel, la un query, pentru fiecare valoare putem căuta binar prima și ultima apariție a acesteia în cadrul subsecvenței, ceea ce este suficient pentru a calcula suma dorită.

Complexitatea finală este $O(N + K * M * \log N)$.

Soluție $O((N + M) * K)$ - **45p**

Definim K ca fiind numărul de valori distincte din sir.

Precalcułăm în complexitate $O(N * K)$ pentru fiecare poziție din sir ultima apariție a fiecărei valori de la stânga la dreapta și ultima apariție a fiecărei valori de la dreapta la stânga. Pentru un query, folosindu-ne de cele 2 precalculări putem afla prima și ultima apariție a fiecărei valori în cadrul subsecvenței, ceea ce este suficient pentru a calcula suma dorită.

Complexitatea finală este $O((N + M) * K)$.

Soluție $O((N + M) * \sqrt{N})$ - **70p**

Împărțim sirul în bucăți de lungime \sqrt{N} și la un pas rezolvăm queryurile în complexitate $O(\sqrt{N})$, făcând și o parcurgere a sirului în complexitate $O(N)$. Dacă ne fixăm o bucată $[st, dr]$ de lungime \sqrt{N} , calculăm soluția pentru un query inclus în acest interval în complexitate $O(\sqrt{N})$.

Incrementăm un indice de la $dr + 1$ până la N și pe parcurs calculăm soluția pentru subsecvența $[dr + 1, i]$ și la fel ca în soluția precedentă menținem prima și ultima apariție a unei valori.

Dacă găsim un query cu capătul dreapta în i și capătul stânga în intervalul $[st, dr]$, putem decrementa capătul stânga ce momentan este $dr + 1$ și să calculăm soluția până când acesta va fi identic cu capătul stânga al queryului.

Practic, vom actualiza prima apariție a valorilor, iar după ce terminăm de calculat asta trebuie să avem grija să refacem primele apariții cum erau înainte de query și să resetăm actualul capăt stânga la poziția $dr + 1$. Deoarece pentru fiecare query tot ce facem este să decrementăm un indice de maxim \sqrt{N} ori, vom da răspunsul la query în complexitate $O(\sqrt{N})$.

Astfel, pentru fiecare query calculăm răspunsul în $O(\sqrt{N})$, iar pentru fiecare bucătă de lungime \sqrt{N} parcurgem $O(N)$ valori.

Complexitatea finală este $O((N + M) * \sqrt{N})$.

Soluție $O(N \log N + M)$ - 100p

O primă observație este că pentru o valoare fixată distanța maximă între prima și ultima sa apariție este suma distanțelor dintre aparițiile consecutive.

Astfel, parcurgem sirul de la stânga la dreapta, iar când întâlnim o valoare actualizăm

$A[\text{indicele_ultimei_aparii}]$

cu diferența dintre indicele curent și indicele ultimei apariții, și apoi actualizăm indicele ultimei apariții a acestei valori.

Când întâlnim un query cu capătul dreapta în indicele curent, soluția va fi dată de $A[st] + A[st + 1] + \dots + A[dr]$, adică suma pe subsecvența $[st, dr]$ pe sirul A .

Atât un update cât și un query se pot face în complexitate $O(\log(N))$ folosind un *arbore indexat binar / arbore de intervale* sau o altă structură asemănătoare.

Complexitatea finală este $O(N \log N + M)$.

22.5.2 Cod sursă

Listing 22.5.1: calafatDEM.cpp

```

1 #include<stdio.h>
2 #include<vector>
3
4 using namespace std;
5
6 #define pb push_back
7 #define pi pair<int, int>
8 #define x first
9 #define y second
10 #define NMAX 200005
11 #define ll long long
12
13 ll AIB[NMAX], sp[NMAX], sol[NMAX];
14 vector<int> queries[NMAX];
15 pi q[NMAX];
16 int last[NMAX], left, v[NMAX];
17 int n, m;
18
19 inline void update(int poz, int val)
20 {
21     if(!poz)
22         return ;
23     for(; poz <= n; poz += ((poz ^ (poz - 1)) & poz))
24         AIB[poz] += val;
25 }
26
27 inline ll query(int poz)
28 {
29     ll s = 0;
30     for(; poz; poz -= ((poz ^ (poz - 1)) & poz))
31         s += AIB[poz];
32     return s;
33 }
34
35
36 inline void read()
37 {
38     scanf("%d%d", &n, &m);
39     for(int i = 1; i <= n; i++)
40         scanf("%d", &v[i]);
41     for(int i = 1; i <= m; i++)
42     {
43         scanf("%d%d", &q[i].x, &q[i].y);
44         queries[q[i].y].pb(i);
45     }
46 }
47
48 inline int cmp(const pi& a, const pi& b)

```

```

49  {
50      return a.y < b.y;
51  }
52
53  inline void solve()
54  {
55      for(int i = 1; i <= n; i++)
56      {
57          left = last[v[i]];
58          if(left)
59              update(left, i - left);
60          int lim = queries[i].size();
61          for(int j = 0; j < lim; j++)
62              sol[queries[i][j]] = query(i - 1) - query(q[queries[i][j]].x - 1);
63          last[v[i]] = i;
64      }
65  }
66
67  inline void write()
68  {
69      for(int i = 1; i <= m; i++)
70          printf("%lld\n", sol[i]);
71  }
72
73  int main ()
74  {
75      freopen("calafat.in", "r", stdin);
76      freopen("calafat.out", "w", stdout);
77
78      read();
79      solve();
80      write();
81
82      return 0;
83 }
```

Listing 22.5.2: calafatDM_aib.cpp

```

1 #include <iostream>
2 #include <assert>
3 #include <vector>
4
5 using namespace std;
6
7 ifstream f("calafat.in");
8 ofstream g("calafat.out");
9
10 const int Nmax = 200000;
11 const int Qmax = 200000;
12
13 const int Q = Qmax + 10;
14 const int N = Nmax + 10;
15
16 vector<pair<int, int> > qr[N];
17 pair<int, int> upd[N];
18 long long sol[Q], A[N];
19 int last[N], n, q, l, r, x;
20
21 void update(int p, int x)
22 {
23     for(; p <= n; p += p & -p)
24         A[p] += x;
25 }
26
27 long long find(int p)
28 {
29     long long sum = 0;
30     for (; p; p -= p & -p)
31         sum += A[p];
32
33     return sum;
34 }
35
36 int main ()
37 {
```

```

38     f >> n >> q;
39     assert(n >= 1 && n <= Nmax);
40     assert(q >= 1 && q <= Qmax);
41     for (int i = 1; i <= n; ++i)
42     {
43         f >> x;
44         assert(x >= 1 && x <= Nmax);
45         if (!last[x])
46             last[x] = i;
47         else
48         {
49             upd[i] = make_pair(last[x], i - last[x]);
50             last[x] = i;
51         }
52     }
53
54     for (int i = 1; i <= q; ++i)
55     {
56         f >> l >> r;
57         assert(l <= r && l <= 1 && l <= n && l <= r && r <= n);
58         qr[r].push_back(make_pair(l, i));
59     }
60
61     for (int i = 1; i <= n; ++i)
62     {
63         if (upd[i].first)
64             update(upd[i].first, upd[i].second);
65
66         for (auto &it : qr[i])
67             sol[it.second] = find(i) - find(it.first - 1);
68     }
69
70     for (int i = 1; i <= q; ++i)
71         g << sol[i] << "\n";
72
73     return 0;
74 }
```

Listing 22.5.3: calafatDM_aint.cpp

```

1 #include <iostream>
2 #include <cassert>
3 #include <vector>
4
5 using namespace std;
6
7 ifstream f("calafat.in");
8 ofstream g("calafat.out");
9
10 const int Nmax = 200000;
11 const int Qmax = 200000;
12
13 const int Q = Qmax + 10;
14 const int N = Nmax + 10;
15
16 vector<pair<int, int> > qr[N];
17 pair<int, int> upd[N];
18 long long sol[Q], A[N * 4];
19 int last[N], n, q, l, r, x;
20
21 void update(int st, int dr, int po, int p, int x)
22 {
23     if (st == dr)
24     {
25         A[po] = x;
26         return;
27     }
28
29     int mij = (st + dr) >> 1;
30     if (p <= mij)
31         update(st, mij, po << 1, p, x);
32     else
33         update(mij + 1, dr, (po << 1) + 1, p, x);
34
35     A[po] = A[po << 1] + A[(po << 1) + 1];

```

```

36 }
37
38 void query(int st, int dr, int po, int l, int r, long long &sol)
39 {
40     if (st >= l && dr <= r)
41     {
42         sol += A[po];
43         return;
44     }
45
46     int mij = (st + dr) >> 1;
47     if (l <= mij)
48         query(st, mij, po << 1, l, r, sol);
49
50     if (r > mij)
51         query(mij + 1, dr, (po << 1) + 1, l, r, sol);
52 }
53
54 int main ()
55 {
56     f >> n >> q;
57     assert(n >= 1 && n <= Nmax);
58     assert(q >= 1 && q <= Qmax);
59
60     for (int i = 1; i <= n; ++i)
61     {
62         f >> x;
63         assert(x >= 1 && x <= Nmax);
64         if (!last[x])
65             last[x] = i;
66         else
67         {
68             upd[i] = make_pair(last[x], i - last[x]);
69             last[x] = i;
70         }
71     }
72
73     for (int i = 1; i <= q; ++i)
74     {
75         f >> l >> r;
76         assert(l <= r && l <= 1 && l <= n && l <= r && r <= n);
77         qr[r].push_back(make_pair(l, i));
78     }
79
80     for (int i = 1; i <= n; ++i)
81     {
82         if (upd[i].first)
83             update(1, n, 1, upd[i].first, upd[i].second);
84
85         for (auto &it : qr[i])
86             query(1, n, 1, it.first, i, sol[it.second]);
87     }
88
89     for (int i = 1; i <= q; ++i)
90         g << sol[i] << "\n";
91
92     return 0;
93 }
```

22.5.3 *Rezolvare detaliată

22.6 transform

Problema 6 - transform

100 de puncte

O matrice pătratică de dimensiuni $N \times N$ cu liniile și coloanele indexate de la 1 la N se numește *matrice șmecheră de Calafat* dacă pe fiecare linie și fiecare coloană există exact două valori de 1, restul elementelor fiind 0.

Cerințe

Având două matrice řmechere de Calafat notate cu A și B , se cere ca prin interschimbări de linii și coloane să se transforme matricea B în matricea A .

Date de intrare

Fișierul **transform.in** conține pe prima linie numărul N , reprezentând dimensiunea matricei.

Pe următoarele $2N$ linii avem câte o pereche de numere naturale separate prin spațiu, reprezentând linia și coloana unui element de valoare 1 din matricea A .

În continuare pe următoarele $2N$ linii avem câte o pereche de numere naturale separate prin spațiu, reprezentând linia și coloana unui element de valoare 1 din matricea B .

Date de ieșire

Fișierul **transform.out** va conține pe fiecare linie câte o operație de interschimbare a două linii sau două coloane, codificată printr-un triplet $ch \ x \ y$ format dintr-un caracter ch și două numere naturale x și y separate prin câte un spațiu.

Valoarea lui ch de fiecare dată poate fi doar 'L', 'C' sau '0'.

Dacă valoarea lui ch este egală cu 'L', atunci se vor interschimba liniiile x și y în matricea B .

Dacă valoarea lui ch este egală cu 'C', atunci se vor interschimba coloanele x și y în matricea B .

Ultimul triplet de valori introdus în fișierul de ieșire va fi '0 0 0' reprezentând terminarea acțiunii.

Restricții și precizări

- $1 \leq N \leq 80\,000$
- Pentru un program care se încadrează în timpul de execuție, punctajul acordat depinde de numărul de operații tipărite în fișierul de ieșire. Să notăm cu op numărul de operații efectuate. Astfel pentru fiecare fișier de ieșire corect, punctajul se va acorda astfel:
 - dacă $1 \leq op \leq 2N$, se acordă 100% din punctaj;
 - dacă $2N + 1 \leq op \leq 4N$, se acordă 75% din punctaj;
 - dacă $op > 4N$, se acordă 50% din punctaj.
- Pentru toate testele de intrare există soluție.

Exemple:

transform.in	transform.out	Explicații
4	L 3 4	$N=4$
1 1	C 3 2	Prima dată se citește matricea A: 1 1 0 0
2 2	0 0 0	0 1 1 0
3 3		0 0 1 1
4 1		1 0 0 1
3 4		apoi citim matricea B: 1 0 1 0
4 4		0 1 1 0
2 3		1 0 0 1
1 2		0 1 0 1
1 3		Aplicăm operația L 3 4 interschimbând liniile 3 și 4 în matricea B: 1 0 1 0
2 3		0 1 1 0
1 1		1 0 0 1
2 2		0 1 0 1
4 2		Aplicăm operația C 3 2 interschimbând coloanele 3 și 2 în matricea B: 1 0 1 0
4 4		0 1 1 0
3 4		0 1 0 1
3 1		1 0 0 1
		Citind linia 0 0 0 înțelegem că s-au terminat operatiile și s-a obținut matricea A.

Timp maxim de executare/test: **3.5** secunde pe Windows, **1.5** secunde pe Linux

Memorie: total **128 MB** din care pentru stivă **128 MB**

Dimensiune maximă a sursei: **20 KB**

22.6.1 Indicații de rezolvare

Szabo Zoltan -Liceul Tehnologic "Petru Maior" Reghin

Soluția 1 - 75 de puncte

Vom încerca să transformăm fiecare matrice într-o *formă canonică*. O matrice care conține un singur *ciclu* poate fi adusă la urmatoarea *formă scără*:

```
11000...000
01100...000
00110...000
...
00000...011
10000...001
```

Vom aduce fiecare ciclu al matricii B , respectiv A , la forma sa canonică, iar acestea vor fi poziționate în matrice crescător după lungimea ciclurilor (ciclul de lungime minimă $L1$ va ocupa primele $L1$ linii și $L1$ coloane, următorul ciclu de lungime minimă $L2$ va ocupa următoarele $L2$ linii și $L2$ coloane etc.).

Pentru a transforma matricea B în matricea A , vom efectua mutările necesare pentru a aduce matricea B în forma canonică, apoi mutările necesare pentru a aduce matricea A în forma canonică, dar în ordine inversă.

Această soluție folosește cel mult $4N$ mutări și are o complexitate de timp și spațiu de $O(N)$.

Soluția 2 - 100 de puncte

Vom folosi aceeași idee de la soluția anterioră, cu următoarea observație: după efectuarea tuturor mutărilor asupra matricii B , vom obține o anumită permutare a liniilor, respectiv coloanelor.

Fiecare din aceste permutări poate fi obținută cu numai N mutări.

Astfel, scădem numărul total de mutări utilizate de la cel mult $4N$ la cel mult $2N$. Complexitatea de timp și de spațiu este $O(N)$.

Soluția 3 - 100 de puncte

Observăm, că o interschimbare de linie și o interschimbare de coloană poate poziționa orice element al matricei pe orice altă coordonată. Iar apoi, dacă linia și coloana elementului fixat nu se va mai antrena în alte interschimbări, atunci elementul corespunzător nu se va mai mișca din loc.

Vom încerca să repozitionăm elementele matricei B în funcție de elementele matricei A .

Elementele de pe linii și coloane formează cicluri distințe cu lungimea totală N .

Calculăm lungimea ciclurilor pentru fiecare linie din A respectiv B , apoi vom repozitiona elementele din B , folosindu-ne de doar de linii și coloane care aparțin unor cicluri de aceeași lungime.

Astfel fiecărui element aplicăm cel mult două interschimbări: una pe linie și una pe coloană, dar elementele fiind câte două pe linie și coloană, iar interschimbările le vom aplica înălțuit, numărul total al operațiilor nu va depăși $2N$.

Complexitatea algoritmului, în funcție de implementare variază între $O(N)$ și $O(N\sqrt{N})$.

22.6.2 Cod sursă

Listing 22.6.1: AH_75p.cpp

```

1 /**
2 * Author: Andrei Heidelbacher
3 * Time complexity: O(NlogN)
4 * Memory complexity: O(N)
5 */
6
7 #include <fstream>
```

```

8 #include <vector>
9 #include <algorithm>
10
11 using namespace std;
12
13 const char IN_FILE[] = "transform.in";
14 const char OUT_FILE[] = "transform.out";
15
16 class Point
17 {
18 public:
19     int x, y;
20
21     Point(const int _x, const int _y):
22         x(_x),
23         y(_y) {}
24 };
25
26 class Move
27 {
28 public:
29     bool type;
30     int v0, v1;
31
32     Move(const bool _type, const int _v0, const int _v1):
33         type(_type),
34         v0(_v0),
35         v1(_v1) {}
36 };
37
38 class Matrix
39 {
40 public:
41     Matrix(const int _n, const vector<Point> &ones):
42         n(_n),
43         points(ones),
44         rows(vector<vector<int>>(n)),
45         columns(vector<vector<int>>(n))
46     {
47         for (int i = 0; i < 2 * n; ++i)
48         {
49             rows[points[i].x].push_back(i);
50             columns[points[i].y].push_back(i);
51         }
52     }
53
54     int pairH(const int index) const
55     {
56         if (rows[points[index].x][0] == index)
57             return rows[points[index].x][1];
58         else
59             return rows[points[index].x][0];
60     }
61
62     int pairV(const int index) const
63     {
64         if (columns[points[index].y][0] == index)
65             return columns[points[index].y][1];
66         else
67             return columns[points[index].y][0];
68     }
69
70     vector<Move> getMoves() const
71     {
72         return moves;
73     }
74
75     void normalize()
76     {
77         vector<int> chosen = normalizedOrder();
78         for (int i = 0, offset = 0; i < int(chosen.size()); ++i)
79         {
80             int begin = chosen[i], end = pairV(chosen[i]);
81             swapRows(points[begin].x, offset);
82             swapColumns(points[begin].y, offset);
83             int now = begin, next = pairH(begin);

```

```

84
85     while (next != end)
86     {
87         swapColumns(points[now].y + 1, points[next].y);
88         now = next;
89         next = pairV(now);
90         swapRows(points[now].x + 1, points[next].x);
91         now = next;
92         next = pairH(now);
93         ++offset;
94     }
95
96     ++offset;
97 }
98
99
100 private:
101     int n;
102     vector<Point> points;
103     vector<vector<int>> rows, columns;
104     vector<Move> moves;
105
106     void swapRows(const int r0, const int r1)
107     {
108         for (int i = 0; i < 2; ++i)
109         {
110             points[rows[r0][i]].x = r1;
111             points[rows[r1][i]].x = r0;
112         }
113         swap(rows[r0], rows[r1]);
114         if (r0 != r1)
115             moves.push_back(Move(false, r0, r1));
116     }
117
118     void swapColumns(const int c0, const int c1)
119     {
120         for (int i = 0; i < 2; ++i)
121         {
122             points[columns[c0][i]].y = c1;
123             points[columns[c1][i]].y = c0;
124         }
125
126         swap(columns[c0], columns[c1]);
127         if (c0 != c1)
128             moves.push_back(Move(true, c0, c1));
129     }
130
131     vector<int> normalizedOrder()
132     {
133         vector<pair<int, int>> cycles;
134         vector<bool> visited = vector<bool>(2 * n, false);
135         for (int i = 0; i < 2 * n; ++i)
136         {
137             if (!visited[i])
138             {
139                 int length = 0;
140                 int j = i;
141                 do
142                 {
143                     visited[j] = true;
144                     j = pairH(j);
145                     visited[j] = true;
146                     j = pairV(j);
147                     length += 2;
148                 } while (j != i);
149
150                 cycles.push_back(make_pair(length, i));
151             }
152         }
153
154         sort(cycles.begin(), cycles.end());
155         vector<int> chosen;
156         for (int i = 0; i < int(cycles.size()); ++i)
157             chosen.push_back(cycles[i].second);
158         return chosen;
159     }

```

```

160 };
161
162 vector<Move> solve(Matrix from, Matrix to)
163 {
164     from.normalize();
165     vector<Move> first = from.getMoves();
166     to.normalize();
167     vector<Move> second = to.getMoves();
168     reverse(second.begin(), second.end());
169     vector<Move> moves = first;
170     moves.insert(moves.end(), second.begin(), second.end());
171     return moves;
172 }
173
174 pair<Matrix, Matrix> read()
175 {
176     ifstream in(IN_FILE);
177     int n;
178     in >> n;
179     vector<Matrix> matrices;
180     for (int i = 0; i < 2; ++i)
181     {
182         vector<Point> points;
183         for (int i = 0; i < 2 * n; ++i)
184         {
185             int x, y;
186             in >> x >> y;
187             points.push_back(Point(x - 1, y - 1));
188         }
189         matrices.push_back(Matrix(n, points));
190     }
191     in.close();
192     return make_pair(matrices[1], matrices[0]);
193 }
194
195 void write(const vector<Move> &moves)
196 {
197     ofstream out(OUT_FILE);
198     for (int i = 0; i < int(moves.size()); ++i)
199         out << (moves[i].type ? 'C' : 'L') << " " <<
200         moves[i].v0 + 1 << " " <<
201         moves[i].v1 + 1 << "\n";
202     out << "0 0 0\n";
203     out.close();
204 }
205
206
207 int main()
208 {
209     pair<Matrix, Matrix> input = read();
210     vector<Move> moves = solve(input.first, input.second);
211     write(moves);
212     return 0;
213 }

```

Listing 22.6.2: transformAH1.cpp

```

1 /**
2 * Author: Andrei Heidelbacher
3 * Time complexity: O(N)
4 * Memory complexity: O(N)
5 */
6
7 #include <iostream>
8 #include <vector>
9 #include <algorithm>
10
11 using namespace std;
12
13 const char IN_FILE[] = "transform.in";
14 const char OUT_FILE[] = "transform.out";
15
16 class Point
17 {
18     public:

```

```

19     int x, y;
20
21     Point(const int _x, const int _y):
22         x(_x),
23         y(_y) {}
24     };
25
26 class Move
27 {
28 public:
29     bool type;
30     int v0, v1;
31
32     Move(const bool _type, const int _v0, const int _v1):
33         type(_type),
34         v0(_v0),
35         v1(_v1) {}
36     };
37
38 class Matrix
39 {
40 public:
41     Matrix(const int _n, const vector<Point> &ones):
42         n(_n),
43         points(ones),
44         rows(vector<vector<int>>(n)),
45         columns(vector<vector<int>>(n))
46     {
47         for (int i = 0; i < 2 * n; ++i)
48         {
49             rows[points[i].x].push_back(i);
50             columns[points[i].y].push_back(i);
51         }
52     }
53
54     int size() const
55     {
56         return n;
57     }
58
59     int pairH(const int index) const
60     {
61         if (rows[points[index].x][0] == index)
62             return rows[points[index].x][1];
63         else
64             return rows[points[index].x][0];
65     }
66
67     int pairV(const int index) const
68     {
69         if (columns[points[index].y][0] == index)
70             return columns[points[index].y][1];
71         else
72             return columns[points[index].y][0];
73     }
74
75     vector<Move> getMoves() const
76     {
77         return moves;
78     }
79
80     void normalize()
81     {
82         vector<int> chosen = normalizedOrder();
83         for (int i = 0, offset = 0; i < int(chosen.size()); ++i)
84         {
85             int begin = chosen[i], end = pairV(chosen[i]);
86             swapRows(points[begin].x, offset);
87             swapColumns(points[begin].y, offset);
88             int now = begin, next = pairH(begin);
89             while (next != end)
90             {
91                 swapColumns(points[now].y + 1, points[next].y);
92                 now = next;
93                 next = pairV(now);
94                 swapRows(points[now].x + 1, points[next].x);

```

```

95         now = next;
96         next = pairH(now);
97         ++offset;
98     }
99     ++offset;
100 }
101 }
102
103 private:
104     int n;
105     vector<Point> points;
106     vector<vector<int>> rows, columns;
107     vector<Move> moves;
108
109     void swapRows(const int r0, const int r1)
110     {
111         for (int i = 0; i < 2; ++i)
112         {
113             points[rows[r0][i]].x = r1;
114             points[rows[r1][i]].x = r0;
115         }
116         swap(rows[r0], rows[r1]);
117         moves.push_back(Move(false, r0, r1));
118     }
119
120     void swapColumns(const int c0, const int c1)
121     {
122         for (int i = 0; i < 2; ++i)
123         {
124             points[columns[c0][i]].y = c1;
125             points[columns[c1][i]].y = c0;
126         }
127         swap(columns[c0], columns[c1]);
128         moves.push_back(Move(true, c0, c1));
129     }
130
131     vector<int> normalizedOrder()
132     {
133         vector<pair<int, int>> cycles;
134         vector<bool> visited = vector<bool>(2 * n, false);
135         for (int i = 0; i < 2 * n; ++i)
136         {
137             if (!visited[i])
138             {
139                 int length = 0;
140                 int j = i;
141
142                 do
143                 {
144                     visited[j] = true;
145                     j = pairH(j);
146                     visited[j] = true;
147                     j = pairV(j);
148                     length += 2;
149                 } while (j != i);
150
151                 cycles.push_back(make_pair(length, i));
152             }
153         }
154
155         vector<vector<int>> buckets = vector<vector<int>>(2 * n + 1, vector<int>());
156         for (int i = 0; i < int(cycles.size()); ++i)
157             buckets[cycles[i].first].push_back(i);
158         vector<int> chosen;
159         for (int length = 1; length <= 2 * n; ++length)
160             for (int i = 0; i < int(buckets[length].size()); ++i)
161                 chosen.push_back(cycles[buckets[length][i]].second);
162         return chosen;
163     }
164 };
165
166     vector<Move> getMoves(Matrix from, Matrix to)
167     {
168         from.normalize();
169         vector<Move> moves = from.getMoves();
170         to.normalize();

```

```

171     vector<Move> second = to.getMoves();
172     reverse(second.begin(), second.end());
173     moves.insert(moves.end(), second.begin(), second.end());
174     return moves;
175 }
176
177 vector<Move> solvePermutation(const vector<int> &permutation, const bool type)
178 {
179     const int n = int(permutation.size());
180     vector<int> values = vector<int>(n), position = vector<int>(n);
181     for (int i = 0; i < n; ++i)
182         values[i] = position[i] = i;
183     vector<Move> moves;
184     for (int i = 0; i < n; ++i)
185     {
186         const int x = values[i];
187         const int y = permutation[i];
188         moves.push_back(Move(type, position[x], position[y]));
189         swap(values[position[x]], values[position[y]]);
190         swap(position[x], position[y]);
191     }
192     return moves;
193 }
194
195 vector<Move> reduceMoves(const int n, const vector<Move> &moves)
196 {
197     vector<int> rows = vector<int>(n), columns = vector<int>(n);
198     for (int i = 0; i < n; ++i)
199         rows[i] = columns[i] = i;
200     for (int i = 0; i < int(moves.size()); ++i)
201     {
202         if (moves[i].type)
203             swap(columns[moves[i].v0], columns[moves[i].v1]);
204         else
205             swap(rows[moves[i].v0], rows[moves[i].v1]);
206     }
207     vector<Move> reduced = solvePermutation(rows, false);
208     vector<Move> columnMoves = solvePermutation(columns, true);
209     reduced.insert(reduced.end(), columnMoves.begin(), columnMoves.end());
210     return reduced;
211 }
212
213 vector<Move> solve(const Matrix &from, const Matrix &to)
214 {
215     const int n = int(from.size());
216     return reduceMoves(n, getMoves(from, to));
217 }
218
219 pair<Matrix, Matrix> read()
220 {
221     ifstream in(IN_FILE);
222     int n;
223     in >> n;
224     vector<Matrix> matrices;
225     for (int i = 0; i < 2; ++i)
226     {
227         vector<Point> points;
228         for (int i = 0; i < 2 * n; ++i)
229         {
230             int x, y;
231             in >> x >> y;
232             points.push_back(Point(x - 1, y - 1));
233         }
234         matrices.push_back(Matrix(n, points));
235     }
236     in.close();
237     return make_pair(matrices[1], matrices[0]);
238 }
239
240 void write(const vector<Move> &moves)
241 {
242     ofstream out(OUT_FILE);
243     for (int i = 0; i < int(moves.size()); ++i)
244         out << (moves[i].type ? 'C' : 'L') << " " <<
245             moves[i].v0 + 1 << " " <<
246             moves[i].v1 + 1 << "\n";

```

```

247     out << "0 0 0\n";
248     out.close();
249 }
250
251 int main()
252 {
253     pair<Matrix, Matrix> input = read();
254     vector<Move> moves = solve(input.first, input.second);
255     write(moves);
256     return 0;
257 }
```

Listing 22.6.3: transformAH2.cpp

```

1 /**
2  * Author: Andrei Heidelbacher
3  * Time complexity: O(NlogN)
4  * Memory complexity: O(N)
5 */
6
7 #include <fstream>
8 #include <vector>
9 #include <algorithm>
10
11 using namespace std;
12
13 const char IN_FILE[] = "transform.in";
14 const char OUT_FILE[] = "transform.out";
15
16 class Point
17 {
18 public:
19     int x, y;
20
21     Point(const int _x, const int _y):
22         x(_x),
23         y(_y) {}
24 };
25
26 class Move
27 {
28 public:
29     bool type;
30     int v0, v1;
31
32     Move(const bool _type, const int _v0, const int _v1):
33         type(_type),
34         v0(_v0),
35         v1(_v1) {}
36 };
37
38 class Matrix
39 {
40 public:
41     Matrix(const int _n, const vector<Point> &ones):
42         n(_n),
43         points(ones),
44         rows(vector<vector<int>>(n)),
45         columns(vector<vector<int>>(n))
46     {
47         for (int i = 0; i < 2 * n; ++i)
48         {
49             rows[points[i].x].push_back(i);
50             columns[points[i].y].push_back(i);
51         }
52     }
53
54     int size() const
55     {
56         return n;
57     }
58
59     int pairH(const int index) const
60     {
61         if (rows[points[index].x][0] == index)
```

```

62         return rows[points[index].x][1];
63     else
64         return rows[points[index].x][0];
65 }
66
67 int pairV(const int index) const
68 {
69     if (columns[points[index].y][0] == index)
70         return columns[points[index].y][1];
71     else
72         return columns[points[index].y][0];
73 }
74
75 vector<Move> getMoves() const
76 {
77     return moves;
78 }
79
80 void normalize()
81 {
82     vector<int> chosen = normalizedOrder();
83     for (int i = 0, offset = 0; i < int(chosen.size()); ++i)
84     {
85         int begin = chosen[i], end = pairV(chosen[i]);
86         swapRows(points[begin].x, offset);
87         swapColumns(points[begin].y, offset);
88         int now = begin, next = pairH(begin);
89         while (next != end)
90         {
91             swapColumns(points[now].y + 1, points[next].y);
92             now = next;
93             next = pairV(now);
94             swapRows(points[now].x + 1, points[next].x);
95             now = next;
96             next = pairH(now);
97             ++offset;
98         }
99         ++offset;
100    }
101 }
102
103 private:
104     int n;
105     vector<Point> points;
106     vector<vector<int>> rows, columns;
107     vector<Move> moves;
108
109 void swapRows(const int r0, const int r1)
110 {
111     for (int i = 0; i < 2; ++i)
112     {
113         points[rows[r0][i]].x = r1;
114         points[rows[r1][i]].x = r0;
115     }
116     swap(rows[r0], rows[r1]);
117     moves.push_back(Move(false, r0, r1));
118 }
119
120 void swapColumns(const int c0, const int c1)
121 {
122     for (int i = 0; i < 2; ++i)
123     {
124         points[columns[c0][i]].y = c1;
125         points[columns[c1][i]].y = c0;
126     }
127     swap(columns[c0], columns[c1]);
128     moves.push_back(Move(true, c0, c1));
129 }
130
131 vector<int> normalizedOrder()
132 {
133     vector<pair<int, int>> cycles;
134     vector<bool> visited = vector<bool>(2 * n, false);
135     for (int i = 0; i < 2 * n; ++i)
136     {
137         if (!visited[i])

```

```

138         {
139             int length = 0;
140             int j = i;
141
142             do
143             {
144                 visited[j] = true;
145                 j = pairH(j);
146                 visited[j] = true;
147                 j = pairV(j);
148                 length += 2;
149             } while (j != i);
150
151             cycles.push_back(make_pair(length, i));
152         }
153     }
154
155     sort(cycles.begin(), cycles.end());
156     vector<int> chosen;
157     for (int i = 0; i < int(cycles.size()); ++i)
158         chosen.push_back(cycles[i].second);
159     return chosen;
160 }
161 };
162
163 vector<Move> getMoves(Matrix from, Matrix to)
164 {
165     from.normalize();
166     vector<Move> moves = from.getMoves();
167     to.normalize();
168     vector<Move> second = to.getMoves();
169     reverse(second.begin(), second.end());
170     moves.insert(moves.end(), second.begin(), second.end());
171     return moves;
172 }
173
174 vector<Move> solvePermutation(const vector<int> &permutation, const bool type)
175 {
176     const int n = int(permutation.size());
177     vector<int> values = vector<int>(n), position = vector<int>(n);
178     for (int i = 0; i < n; ++i)
179         values[i] = position[i] = i;
180     vector<Move> moves;
181
182     for (int i = 0; i < n; ++i)
183     {
184         const int x = values[i];
185         const int y = permutation[i];
186         moves.push_back(Move(type, position[x], position[y]));
187         swap(values[position[x]], values[position[y]]);
188         swap(position[x], position[y]);
189     }
190
191     return moves;
192 }
193
194 vector<Move> reduceMoves(const int n, const vector<Move> &moves)
195 {
196     vector<int> rows = vector<int>(n), columns = vector<int>(n);
197     for (int i = 0; i < n; ++i)
198         rows[i] = columns[i] = i;
199     for (int i = 0; i < int(moves.size()); ++i)
200     {
201         if (moves[i].type)
202             swap(columns[moves[i].v0], columns[moves[i].v1]);
203         else
204             swap(rows[moves[i].v0], rows[moves[i].v1]);
205     }
206
207     vector<Move> reduced = solvePermutation(rows, false);
208     vector<Move> columnMoves = solvePermutation(columns, true);
209     reduced.insert(reduced.end(), columnMoves.begin(), columnMoves.end());
210     return reduced;
211 }
212
213 vector<Move> solve(const Matrix &from, const Matrix &to)

```

```

214  {
215      const int n = int(from.size());
216      return reduceMoves(n, getMoves(from, to));
217  }
218
219 pair<Matrix, Matrix> read()
220 {
221     ifstream in(IN_FILE);
222     int n;
223     in >> n;
224     vector<Matrix> matrices;
225     for (int i = 0; i < 2; ++i)
226     {
227         vector<Point> points;
228         for (int i = 0; i < 2 * n; ++i)
229         {
230             int x, y;
231             in >> x >> y;
232             points.push_back(Point(x - 1, y - 1));
233         }
234         matrices.push_back(Matrix(n, points));
235     }
236
237     in.close();
238     return make_pair(matrices[1], matrices[0]);
239 }
240
241 void write(const vector<Move> &moves)
242 {
243     ofstream out(OUT_FILE);
244     for (int i = 0; i < int(moves.size()); ++i)
245         out << (moves[i].type ? 'C' : 'L') << " " <<
246         moves[i].v0 + 1 << " " <<
247         moves[i].v1 + 1 << "\n";
248     out << "0 0 0\n";
249     out.close();
250 }
251
252 int main()
253 {
254     pair<Matrix, Matrix> input = read();
255     vector<Move> moves = solve(input.first, input.second);
256     write(moves);
257     return 0;
258 }
```

Listing 22.6.4: transformZS.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 ifstream fin("transform.in");
6 ofstream fout("transform.out");
7
8 int n,lin1[80001][2],col1[80001][2],lin2[80001][2],col2[80001][2],
9     clin1[80001],clin2[80001],ccol1[80001],
10    ccol2[80001],oklin1[80001],oklin2[80001],okcol1[80001],okcol2[80001];
11
12 void citire()
13 {   int i,x,y;
14     fin>>n;
15     for (i=2*n;i--i)
16     {
17         fin>>x>>y;
18         if(lin1[x][0])
19             lin1[x][1]=y;
20         else
21             lin1[x][0]=y;
22         if(col1[y][0])
23             col1[y][1]=x;
24         else
25             col1[y][0]=x;
26     }
27 }
```

```

28     for (i=2*n;i;--i)
29     {
30         fin>>x>>y;
31         if(lin2[x][0])
32             lin2[x][1]=y;
33         else
34             lin2[x][0]=y;
35         if(col2[y][0])
36             col2[y][1]=x;
37         else
38             col2[y][0]=x;
39     }
40 }
41
42 void rezolvacicluri (int lin[][2],int col[][2],
43                      int clin[],int ccol[],int oklin[], int okcol[])
44 {
45     int p=1,l_curent,c_curent,lung,l_urmator,c_urmator;
46     while (1)
47     {
48         while (p<=n and oklin[p]) p++; // cautam un ciclu inca nedepistat
49         if (p>n) return;
50         l_curent=p;
51         c_curent=lin[l_curent][0];
52         lung=0;
53         do
54         {
55             if (col[c_curent][0]==l_curent)
56                 l_urmator=col[c_curent][1];
57             else
58                 l_urmator=col[c_curent][0];
59
60             if (lin[l_urmator][0]==c_curent)
61                 c_urmator=lin[l_urmator][1];
62             else
63                 c_urmator=lin[l_urmator][0];
64             l_curent=l_urmator;
65             c_curent=c_urmator;
66             lung++; // calculam lungimea ciclului
67         } while (l_urmator!=p);
68
69         l_curent=p;
70         c_curent=lin[l_curent][0]; // vom memora pentru
71                                // fiecare nod al ciclului lungimea
72         do
73         {
74             if (col[c_curent][0]==l_curent)
75                 l_urmator=col[c_curent][1];
76             else
77                 l_urmator=col[c_curent][0];
78
79             if (lin[l_urmator][0]==c_curent)
80                 c_urmator=lin[l_urmator][1];
81             else
82                 c_urmator=lin[l_urmator][0];
83             l_curent=l_urmator;
84             c_curent=c_urmator;
85             oklin[l_curent]=lung;
86         } while (l_urmator!=p);
87     }
88 }
89
90 void rezolvamatrice()
91 {
92     int p1=1,p2,l_curent1,c_curent1,l_urmator1,
93          c_urmator1,l_curent2,c_curent2,l_urmator2,c_urmator2;
94     while (1)
95     {
96         while (p1<=n and !oklin1[p1])
97             p1++; // cautam un ciclu in matricea 1 inca nerezolvat
98         if (p1>n) return;
99
100        p2=1;
101        while (p2<=n and oklin2[p2]!=oklin1[p1])
102            p2++; // cautam un ciclu in matricea 2 cu lungime identica
103        if (p2>n)

```

```

104
105     {
106         fout<<"date eronate!!!!!!!!!!!!!!\n";
107         return;
108     }
109
110     l_curent1=p1;
111     c_curent1=lin1[l_curent1][0]; // matricea 2 o vom transforma in
112                               // matricea 1 prin interschimbari
113                               // de linii si coloane
114     l_curent2=p2;
115     c_curent2=lin2[l_curent2][0];
116     do
117     {
118
119         oklin1[l_curent1]=0;
120         oklin2[l_curent2]=oklin2[l_curent1]; // dupa ce rezolvam o linie,
121                                         // transformam starea in 0
122         oklin2[l_curent1]=0;                // dupa ce rezolvam o linie,
123                                         // transformam starea in 0
124         if (l_curent1!=l_curent2)
125         {
126             fout<<"L "<<l_curent1<<" "<<l_curent2<<"\n";
127
128             swap(lin2[l_curent1][0],lin2[l_curent2][0]);
129             swap(lin2[l_curent1][1],lin2[l_curent2][1]);
130
131             if (col2[lin2[l_curent1][0]][0]==l_curent2)
132                 col2[lin2[l_curent1][0]][0]=-l_curent1;
133
134             if (col2[lin2[l_curent1][0]][1]==l_curent2)
135                 col2[lin2[l_curent1][0]][1]=-l_curent1;
136
137             if (col2[lin2[l_curent1][1]][0]==l_curent2)
138                 col2[lin2[l_curent1][1]][0]=-l_curent1;
139
140             if (col2[lin2[l_curent1][1]][1]==l_curent2)
141                 col2[lin2[l_curent1][1]][1]=-l_curent1;
142
143             if (col2[lin2[l_curent2][0]][0]==l_curent1)
144                 col2[lin2[l_curent2][0]][0]=-l_curent2;
145
146             if (col2[lin2[l_curent2][0]][1]==l_curent1)
147                 col2[lin2[l_curent2][0]][1]=-l_curent2;
148
149             if (col2[lin2[l_curent2][1]][0]==l_curent1)
150                 col2[lin2[l_curent2][1]][0]=-l_curent2;
151
152             if (col2[lin2[l_curent2][1]][1]==l_curent1)
153                 col2[lin2[l_curent2][1]][1]=-l_curent2;
154
155             // retransformare in numere pozitive
156
157             if (col2[lin2[l_curent1][0]][0]<0)
158                 col2[lin2[l_curent1][0]][0]*=-1;
159
160             if (col2[lin2[l_curent1][0]][1]<0)
161                 col2[lin2[l_curent1][0]][1]*=-1;
162
163             if (col2[lin2[l_curent1][1]][0]<0)
164                 col2[lin2[l_curent1][1]][0]*=-1;
165
166             if (col2[lin2[l_curent1][1]][1]<0)
167                 col2[lin2[l_curent1][1]][1]*=-1;
168
169             if (col2[lin2[l_curent2][0]][0]<0)
170                 col2[lin2[l_curent2][0]][0]*=-1;
171
172             if (col2[lin2[l_curent2][0]][1]<0)
173                 col2[lin2[l_curent2][0]][1]*=-1;
174
175             if (col2[lin2[l_curent2][1]][0]<0)
176                 col2[lin2[l_curent2][1]][0]*=-1;
177
178             if (col2[lin2[l_curent2][1]][1]<0)
179                 col2[lin2[l_curent2][1]][1]*=-1;

```

```

180         l_curent2=l_curent1;
181     }
182
183     if (c_curent1!=c_curent2)
184     {
185         fout<<"C "<<c_curent1<<" "<<c_curent2<<"\n";
186
187         swap(col2[c_curent1][0],col2[c_curent2][0]);
188         swap(col2[c_curent1][1],col2[c_curent2][1]);
189
190         if (lin2[col2[c_curent1][0]][0]==c_curent2)
191             lin2[col2[c_curent1][0]][0]=-c_curent1;
192
193         if (lin2[col2[c_curent1][0]][1]==c_curent2)
194             lin2[col2[c_curent1][0]][1]=-c_curent1;
195
196         if (lin2[col2[c_curent1][1]][0]==c_curent2)
197             lin2[col2[c_curent1][1]][0]=-c_curent1;
198
199         if (lin2[col2[c_curent1][1]][1]==c_curent2)
200             lin2[col2[c_curent1][1]][1]=-c_curent1;
201
202         if (lin2[col2[c_curent2][0]][0]==c_curent1)
203             lin2[col2[c_curent2][0]][0]=-c_curent2;
204
205         if (lin2[col2[c_curent2][0]][1]==c_curent1)
206             lin2[col2[c_curent2][0]][1]=-c_curent2;
207
208         if (lin2[col2[c_curent2][1]][0]==c_curent1)
209             lin2[col2[c_curent2][1]][0]=-c_curent2;
210
211         if (lin2[col2[c_curent2][1]][1]==c_curent1)
212             lin2[col2[c_curent2][1]][1]=-c_curent2;
213
214         // retransformare in pozitive
215         if (lin2[col2[c_curent1][0]][0]<0)
216             lin2[col2[c_curent1][0]][0]*=-1;
217
218         if (lin2[col2[c_curent1][0]][1]<0)
219             lin2[col2[c_curent1][0]][1]*=-1;
220
221         if (lin2[col2[c_curent1][1]][0]<0)
222             lin2[col2[c_curent1][1]][0]*=-1;
223
224         if (lin2[col2[c_curent1][1]][1]<0)
225             lin2[col2[c_curent1][1]][1]*=-1;
226
227         if (lin2[col2[c_curent2][0]][0]<0)
228             lin2[col2[c_curent2][0]][0]*=-1;
229
230         if (lin2[col2[c_curent2][0]][1]<0)
231             lin2[col2[c_curent2][0]][1]*=-1;
232
233         if (lin2[col2[c_curent2][1]][0]<0)
234             lin2[col2[c_curent2][1]][0]*=-1;
235
236         if (lin2[col2[c_curent2][1]][1]<0)
237             lin2[col2[c_curent2][1]][1]*=-1;
238
239         c_curent2=c_curent1;
240     }
241
242
243     if (col1[c_curent1][0]==l_curent1)
244         l_urmator1=col1[c_curent1][1];
245     else
246         l_urmator1=col1[c_curent1][0];
247
248     if (lin1[l_urmator1][0]==c_curent1)
249         c_urmator1=lin1[l_urmator1][1];
250     else
251         c_urmator1=lin1[l_urmator1][0];
252
253     if (col2[c_curent2][0]==l_curent2)
254         l_urmator2=col2[c_curent2][1];
255     else

```

```
256         l_urmator2=col2[c_curent2][0];
257
258         if (lin2[l_urmator2][0]==c_curent2)
259             c_urmator2=lin2[l_urmator2][1];
260         else
261             c_urmator2=lin2[l_urmator2][0];
262
263         l_curent1=l_urmator1;
264         c_curent1=c_urmator1;
265         l_curent2=l_urmator2;
266         c_curent2=c_urmator2;
267     }
268     while (l_urmator1!=p1);
269 }
270 }
271
272 int main()
273 {
274     citire();
275     rezolvacicluri(lin1,col1,clin1,ccoll,oklin1,okcoll);
276     rezolvacicluri(lin2,col2,clin2,ccol2,oklin2,okcol2);
277     rezolvamatrice();
278     fout<<"0 0 0\n";
279     return 0;
280 }
```

22.6.3 *Rezolvare detaliată

Capitolul 23

ONI 2015

23.1 metrou

Problema 1 - metrou

100 de puncte

Această problemă este dedicată celor care aşteaptă metroul cu cea mai mare ardoare: locuitorii din Drumul Taberei.

Se dă planul unei rețele de metrou cu N stații și M tuneluri bidirectionale între stații. Două stații de metrou se numesc vecine dacă există un tunel între ele în acest plan. Fiecare stație i are asociat un profit p_i dat.

Henry a fost recent promovat dintr-un post de angajat al departamentului de curătenie pe postul de project manager al firmei. Deoarece nu există fonduri pentru construirea întregii rețele de metrou, Henry trebuie să aleagă o submulțime de stații care vor fi construite, astfel încât oricare două stații alese să nu fie vecine în planul inițial. Pentru a-și păstra poziția în companie, suma profiturilor stațiilor alese în această submulțime trebuie să fie maximă.

Cerințe

Dându-se N, M , profiturile aduse de fiecare din cele N stații și planul inițial al rețelei, să se determine suma maximă a profiturilor stațiilor pe care le poate alege Henry astfel încât oricare două stații alese să nu fie vecine în planul inițial.

Date de intrare

Pe prima linie a fișierului de intrare **metrou.in** se vor afla două numere naturale N și M separate printr-un spațiu, reprezentând numărul de stații, respectiv numărul de tuneluri din planul inițial.

Pe a doua linie se vor afla N numere naturale separate prin câte un spațiu, al i -lea dintre acestea reprezentând profitul p_i adus dacă stația i ar fi construită.

Pe următoarele M linii se vor afla câte două numere naturale x și y separate printr-un spațiu, semnificând faptul că un tunel unește stațiile x și y în planul inițial.

Date de ieșire

În fișierul de ieșire **metrou.out** se va afișa o singură linie conținând un singur număr natural, reprezentând suma maximă a profiturilor stațiilor pe care le poate alege Henry astfel încât oricare două stații alese să nu fie vecine în planul inițial.

Restricții și precizări

- $1 \leq N \leq 100\,000$
- $1 \leq M \leq 150\,000$
- $1 \leq x, y \leq N$
- $1 \leq p_i \leq 10\,000$, pentru orice i , $1 \leq i \leq N$.
- Există maximum 15 stații care se învecinează cu 3 sau mai multe stații în planul dat.
- Există maximum 20 de stații care se învecinează cu exact o stație în planul dat.
- Pentru 20% din teste, $N \leq 20$.
- Pentru alte 10% din teste, planul rețelei de metrou este de forma unui lanț simplu într-un graf neorientat.

- Pentru alte 10% din teste, planul rețelei de metrou este de forma unui ciclu simplu într-un graf neorientat.
- Putem ajunge din orice stație în oricare altă stație urmând tunelurile existente în planul inițial.

Exemplu:

metrou.in	metrou.out	Explicații
8 10 1 3 2 5 4 1 2 1 1 2 2 3 3 4 4 5 5 3 3 6 2 6 2 7 7 8 8 3	9	<p>Avem $N = 8$ stații de metrou și $M = 10$ tuneluri în plan.</p> <p>Submulțimea de stații $\{2, 4, 8\}$ asigură profitul maxim de $3 + 5 + 1 = 9$.</p> <p>Observăm că submulțimea respectă regula descrisă în enunț, încărcă nu există niciun tunel care să unească stațiile 2-4, 2-8 sau 4-8.</p>

Timp maxim de executare/test: **1.5** secunde pe Windows, **0.4** secunde pe Linux

Memorie: total **64 MB** din care pentru stivă **16 MB**

Dimensiune maximă a sursei: **20 KB**

23.1.1 Indicații de rezolvare

Vlad Alexandru Gavrilă

20 de puncte

Pentru $N \leq 20$, problema admite o *soluție exponențială* ce presupune selectarea fiecărei submulțimi posibile de stații (având grija să nu includem două stații vecine) și calcularea profitului adus de respectiva submulțime. Complexitatea acestei soluții este $O(2^N)$ ca timp.

100 de puncte

Să considerăm întâi cum rezolvăm problema în situația în care rețeaua de metrou are forma unui *lanț simplu*.

Această soluție presupune *programare dinamică*: ținem tabloul

dpi = profitul maxim adus de o submulțime a primelor i noduri din lanț.

Recurența este

$dp[i] = \max(dp[i-1], dp[i-2]+p[nod]),$

unde $nod =$ al i -lea nod din lanț.

Observăm că rezolvarea acestui caz are complexitate liniară $O(N)$, și aducea 10 puncte.

În situația în care rețeaua avea forma unui *ciclu simplu*, putem avea aceeași abordare ca la un lanț, rupând ciclul (eliminând o muchie) și transformându-l într-un lanț, apoi calculând dinamica precedentă.

Această dinamică trebuie calculată de două ori, întâi exclusând primul nod din lanț, iar apoi exclusând ultimul nod, evitând astfel situația în care submulțimea de profit maxim includea atât primul cât și ultimul nod din lanț, care sunt vecine în ciclu $O(N)$. Observăm că rezolvarea acestui caz are aceeași complexitate liniară, și aducea încă 10 puncte.

Pentru rezolvarea problemei de 100 de puncte, trebuie să facem următoarea observație: dacă excludem din rețea toate stațiile cu mai mult de 2 vecini, vom obține o rețea formată din mai multe lanțuri care nu se intersectează. Pentru fiecare din aceste lanțuri vom reține de care nod sau noduri speciale erau legate, și vom aplica programarea dinamica anterioară de patru ori, pentru următoarele cazuri:

1. Excludem atât primul cât și ultimul nod din lanț.
2. Excludem doar primul nod din lanț.
3. Excludem doar ultimul nod din lanț.

4. Nu excludem niciun nod din lanț.

Având $K \leq 15$ noduri care au mai mult de 3 vecini (să le numim speciale), putem precalcula pentru fiecare pereche de noduri speciale care este profitul maxim adus de lanțurile ce unesc nodurile din pereche dacă ar fi să selectăm în soluția finală zero, pe primul, pe al doilea, sau ambele noduri din pereche. Această *precalculare* se face folosindu-ne de valorile obținute prin programarea dinamică pentru fiecare lanț. De asemenea se va trata similar cazul în care un lanț este unit de un singur nod special, celălalt capăt fiind o frunză.

Apoi putem selecta în timp exponential submulțimi de noduri speciale, folosindu-ne de precalcularea anterioară pentru a determina soluția finală. Complexitatea acestei abordări este $O(N + 2^K)$.

Menționăm că o abordare greedy care selectează la fiecare pas nodul cel mai profitabil care nu este învecinat cu un alt nod, obține 0 puncte deoarece nu produce răspunsul corect. Un caz ar fi lanțul (8 1 2 3 2 1 8) - abordarea greedy ar selecta (8 1 2 3 2 1 8) care ar aduce profitul 13, în timp ce programarea dinamică ar selecta (8 1 2 3 2 1 8) care produce răspunsul 14.

23.1.2 Cod sursă

Listing 23.1.1: .cpp

```

1 #include <cstdio>
2 #include <set>
3 #include <vector>
4
5 using namespace std;
6
7 #define maxn 200010
8 #define maxk 31
9 #define maxl 201000
10
11 int n, m;
12 int p[maxn];
13 int sp[maxn], nsp, who[maxn];
14 int lant[maxn], d[maxn][2];
15 int f[maxn], g[maxn], g2[maxn];
16 vector<int> v[maxn], w[maxn];
17 int conflict[maxk][maxk];
18
19 int nl;
20 struct lant
21 {
22     int d[2][2];
23     int c1, c2;
24 } l[maxl];
25
26 pair<pair<int, int>, pair<int, int> > getChainResult(int a[])
27 {
28     // Dinamica pe lant - d[i][j] = suma maxima din primele i noduri de pe lant
29     // j=0 daca am luat primul nod, j=1 daca nu
30
31     d[0][0]=d[0][1]=0;
32     d[1][0]=p[a[1]];
33     d[1][1]=0;
34
35     int lg=a[0];
36
37     for(int i=2; i<=lg; ++i)
38     {
39         // printf("%d ", a[i]);
40         for(int j=0; j<2; ++j)
41             d[i][j]=max(d[i-2][j]+p[a[i]], d[i-1][j]);
42     }
43
44     return make_pair(make_pair(d[lg][0], d[lg-1][0]),
45                      make_pair(d[lg][1], d[lg-1][1]));
46 }
47
48 void newChain(int i)
49 {
50     ++nl;
51     if(g2[i]==0) //Nodul i este singur pe lant

```

```

52     {
53         l[nl].c1=who[v[i][0]];
54
55         if(g[i]>1) l[nl].c2=who[v[i][1]];
56         else        l[nl].c2=nsp+1;
57
58         l[nl].d[0][0]=p[i];
59
60         return;
61     }
62
63 //Capatul 1
64
65 l[nl].c1=nsp+1;
66 for(int j=0; j<v[i].size(); ++j)
67     if(g[v[i][j]]>2) l[nl].c1=who[v[i][j]];
68
69 //Construieste lantul
70
71 lant[0]=0;
72 lant[++lant[0]]=i;
73
74 int x=w[i][0];
75
76 lant[++lant[0]]=x;
77
78 f[i]=f[x]=1;
79
80 while(g2[x]==2)
81 {
82     if(f[w[x][0]]==1) x=w[x][1];
83     else                  x=w[x][0];
84     f[x]=1;
85     lant[++lant[0]]=x;
86 }
87
88 //Capatul 2
89
90 l[nl].c2=nsp+1;
91 for(int j=0; j<v[x].size(); ++j)
92     if(g[v[x][j]]>2) l[nl].c2=who[v[x][j]];
93
94 //valorile lantunului - l[nl].d[0/1][0/1] valoarea lantului daca
95 //am luat (1) sau n-am luat (0) fiecare din capete
96
97 pair<pair<int, int>, pair<int, int> > rez = getChainResult(lant);
98
99 l[nl].d[0][0]=rez.first.first;
100 l[nl].d[0][1]=rez.first.second;
101 l[nl].d[1][0]=rez.second.first;
102 l[nl].d[1][1]=rez.second.second;
103 }
104
105 int solveCycle()
106 {
107     //Luam nodul 1 si adaugam toate nodurile din ciclu
108     int x=1;
109
110     while(f[x]==0)
111     {
112         f[x]=1;
113         lant[++lant[0]]=x;
114
115         if(f[v[x][0]]==1) x=v[x][1];
116         else                  x=v[x][0];
117     }
118
119     pair<pair<int, int>, pair<int, int> > rez = getChainResult(lant);
120
121     // Consideram toate variantele mai putin cea in care
122     // luam atat primul cat si ultimul nod din lant
123     return max(rez.first.second, max(rez.second.second, rez.second.first));
124 }
125
126 int solveChain()
127 {

```

```

128     //Construim lantul de la primul nod cu grad 1
129     int x=1;
130
131     for(int i=1; i<=n; ++i)
132         if(g[i]==1) x=i;
133
134     while(f[x]==0)
135     {
136         f[x]=1;
137         lant[++lant[0]]=x;
138
139         for(int i=0; i<v[x].size(); ++i)
140             if(f[v[x][i]]==0)
141                 {
142                     x=v[x][i];
143                     break;
144                 }
145     }
146
147     pair<pair<int, int>, pair<int, int> > rez = getChainResult(lant);
148
149     //Oricare varianta e buna
150     return max(max(rez.first.first, rez.first.second),
151                max(rez.second.first, rez.second.second));
152 }
153
154 int solve()
155 {
156     int has1=0; //Graful are noduri cu grad 1 - Da = 1, Nu = 0
157     int has3=0; //Graful are noduri cu grad mai mare sau egal cu 3 - Da=1, Nu=0
158
159     for(int i=1; i<=n; ++i)
160     {
161         if(g[i]>2) //Nodul i este nod "special"
162         {
163             has3=1;
164             sp[nsp++]=i;
165             who[i]=nsp;
166         }
167         if(g[i]<2) has1=1;
168     }
169
170     if(has1==0 && has3==0) // Nu are nici noduri de grad 1
171         // nici mai mari ca 3 -> e ciclu
172     return solveCycle();
173     if(has3==0) //Nu are noduri de grad mai mare sau egal cu 3 -> e lant
174     return solveChain();
175
176     //Construim graful format doar din nodurile cu grad <= 2
177     for(int i=1; i<=n; ++i)
178     {
179         if(g[i]>2) continue;
180
181         for(int j=0; j<v[i].size(); ++j)
182             if(g[v[i][j]]<=2)
183             {
184                 w[i].push_back(v[i][j]);
185                 ++g2[i];
186             }
187     }
188
189     //Procesam fiecare lant dintr-un nod care in nouul graf are gradul <= 2
190     for(int i=1; i<=n; ++i)
191     {
192         if(g[i]>2 || f[i]==1 || g2[i]>1) continue;
193         newChain(i);
194     }
195
196     // Conflict[i][j] = nu putem pune intr-o solutie si
197     // nodul special i si nodul special j
198     for(int i=0; i<nsp; ++i)
199     {
200         int nod=sp[i];
201         for(int j=0; j<v[nod].size(); ++j)
202             conflict[i+1][who[v[nod][j]]]=1;
203     }

```

```

204     int sol=0;
205
206     // Pentru configuratia i, bitul i = 1 daca luam
207     // nodul special i, 0 daca nu
208     for(int i=0; i<(1<<nsp); ++i)
209     {
210         int ok=1;
211         for(int j=0; j<nsp; ++j)
212             for(int k=0; k<nsp; ++k)
213                 if(conflict[j+1][k+1] && ((i>>j)&1)==1 && ((i>>k)&1)==1)
214                     ok=0;
215
216         if(!ok) continue;
217
218         int rez=0;
219
220         //Adaugam la solutia curenta nodurile speciale selectate
221         for(int j=0; j<nsp; ++j)
222             if((i>>j)&1) rez+=p[sp[j]];
223
224         //Pentru fiecare lant selectam configuratia potrivita
225         for(int j=1; j<=nl; ++j)
226         {
227             int c1=l[j].c1-1;
228             int c2=l[j].c2-1;
229             rez+=l[j].d[(i>>c1)&1][(i>>c2)&1];
230         }
231
232         if(rez>sol) sol=rez;
233     }
234
235     return sol;
236 }
237
238 int main()
239 {
240     freopen("9-metrou.in", "r", stdin);
241     freopen("metrou.out", "w", stdout);
242
243     scanf("%d%d", &n, &m);
244     for(int i=1; i<=n; ++i) scanf("%d", &p[i]);
245     for(int i=1; i<=m; ++i)
246     {
247         int a, b;
248         scanf("%d%d", &a, &b);
249         v[a].push_back(b);
250         v[b].push_back(a);
251         ++g[a];
252         ++g[b];
253     }
254
255     printf("%d\n", solve());
256
257     return 0;
258 }
```

23.1.3 *Rezolvare detaliată

23.2 spiridusi

Problema 2 - spiridusi

100 de puncte

Mei și Satsuki s-au întors de curând în casa de vacanță a familiei lor. Această casă este formată din N camere, unite între ele prin $N - 1$ culoare, astfel încât să se poată ajunge din orice cameră în orice altă cameră. Intrarea în casă se face prin camera 1. Deoarece casa n-a fost locuită timp de mai multe luni, în fiecare cameră i s-au stabilit și spiriduși de praf.

Cele două fete doresc să-și amenajeze un spațiu de joacă întins pe mai multe camere. Ele vor să stabilească două camere a și b (nu neapărat distințe), astfel încât drumul cel mai scurt de la

intrarea în casă până în camera b trece prin camera a . Fetele vor merge apoi din camera a în camera b pe drumul cel mai scurt (fără a trece de două ori prin aceeași cameră), gonind spiridușii de praf aflați în fiecare cameră prin care trec, inclusiv pe cei din camerele a și b . După ce fetele ajung în camera b , ele consideră că toate camerele din care au gonit spiridușii de praf au fost alese pentru spațiul de joacă.

Fetele au stabilit pentru fiecare cameră i un coeficient p_i care reprezintă cât de plăcută ar fi camera i pentru spațiul lor de joacă. În plus, ele au convenit că nu vor goni în total mai mult de C spiriduși ai prafului din camerele prin care trec.

Cerințe

Cunoscând valorile lui N și C , numărul de spiriduși ai prafului s_i , coeficienții p_i pentru fiecare cameră i , cât și modul în care sunt unite camerele prin culoare, să se determine suma maximă a coeficienților p ai camerelor alese pentru un spațiu de joacă ce respectă condițiile impuse de Mei și Satsuki.

Date de intrare

Pe prima linie a fișierului de intrare **spiridusi.in** se vor afla două numere naturale N și C cu semnificația din enunț.

Pe a doua linie se vor afla N numere naturale separate prin câte un spațiu, al i -lea dintre acestea reprezentând numărul de spiriduși s_i aflați în camera i .

Pe a treia linie se vor afla N numere întregi separate prin câte un spațiu, al i -lea dintre acestea reprezentând coeficientul p_i al camerei i .

Pe următoarele $N - 1$ linii se vor afla câte două numere întregi x și y separate printr-un spațiu, semnificând existența unei culori ce unește camerele x și y .

Date de ieșire

În fișierul de ieșire **spiridusi.out** se va afișa o singură linie conținând un singur număr natural, reprezentând suma maximă a coeficienților p ai camerelor alese pentru un spațiu de joacă ce respectă condițiile impuse de Mei și Satsuki.

Restricții și precizări

- $1 \leq N \leq 100\,000$
- $1 \leq C \leq 20\,000\,000$
- $1 \leq s_i \leq 20\,000\,000$, pentru orice i , $1 \leq i \leq N$.
- $-10\,000 \leq p_i \leq 10\,000$, pentru orice i , $1 \leq i \leq N$.
- $1 \leq x, y \leq N$
- Pentru 20% din teste, fiecare cameră are maximum 2 vecini.
- Pentru 30% din teste, $N \leq 1\,000$.
- Se garantează că pentru orice cameră x , numărul total de spiriduși aflați în camerele de pe drumul cel mai scurt de la camera 1 la camera x nu depășește 1 000 000 000.

Exemple:

spiridusi.in	spiridusi.out	Explicații
6 8		Dacă alegem camerele $a = 2$ și $b = 6$, obținem un spațiu de joacă format din camerele 2, 4 și 6.
2 4 6 2 4 1		Numărul total de spiriduși goniți din aceste camere este $4 + 2 + 1 = 7$, care este mai mic sau egal decât $C = 8$.
3 10 11 -2 4 5		Suma coeficienților p ai acestor camere este $10 + (-2) + 5 = 13$, maximul posibil ce se poate obține.
1 2		
2 3		
2 4		
4 5		
4 6		

Timp maxim de executare/test: **1.0** secunde pe Windows, **0.4** secunde pe Linux

Memorie: total **64 MB** din care pentru stivă **16 MB**

Dimensiune maximă a sursei: **20 KB**

23.2.1 Indicații de rezolvare

Vlad Alexandru Gavrilă

30 de puncte

Pentru $N \leq 1\,000$ putem avea următoarea soluție: fixăm camera b într-unul dintre cele N noduri. Apoi, parcurgem în ordine toate nodurile de pe drumul de la b la camera 1, fixând camera a pe rând în fiecare dintre ele. Calculând pe parcurs suma coeficienților p și numărul de spiriduși goniți până la un moment dat pe drumul de la b la a , putem obține soluția în $O(N^2)$, obținând 30 de puncte.

100 de puncte

Pentru 100 de puncte, facem o *parcurgere DF* a arborelui începând din nodul 1, ținând o *stivă Nodes* cu nodurile întâlnite pe drumul direct de la rădăcină la nodul curent x , o altă *stivă Costs* cu suma numerelor de spiriduși ale nodurilor de la rădăcina 1 până la fiecare nod din stiva *Nodes*, și o altă *stivă Profits* cu suma coeficienților p a nodurilor de la rădăcina 1 până la fiecare nod din stiva *Nodes*. Pentru fiecare nod mai ținem următoarele dinamici:

$st[i][nod]$ = nodul aflat cu 2^i pozitii mai aproape de radacina decat nodul *nod*.

$dp[i][nod]$ = valoare minimă dintre elementele stivei *Profits* aferente unui nod din stiva *Nodes* aflat cu intre 1 si 2^i pozitii mai sus decât nodul *nod*.

Recurențele acestor dinamici sunt:

$st[i][nod] = st[i - 1][st[i - 1][nod]]$.

$dp[i][nod] = \min(dp[i - 1][nod], dp[i - 1][st[i - 1][nod]])$.

Acum, va trebui să facem următoarele observații.

1. Pentru două noduri a și b aflate pe pozițiile x , respectiv y în stiva *Nodes*, suma numerelor de spiriduși aflați pe drumul de la camera a la camera b este $Costs[y] - Costs[x - 1]$.

2. Deoarece numărul de spiriduși dintr-o cameră este pozitiv, pentru o cameră b fixată, numărul de spiriduși aflați pe *lanțul* de la o cameră a la camera b crește pe măsură ce fixăm camera a mai departe de b .

3. Pentru două noduri a și b aflate pe pozițiile x , respectiv y în stiva *Nodes*, suma coeficienților p ai camerelor aflate pe drumul de la camera a la camera b este $Profits[y] - Profits[x - 1]$.

Acum fixăm camera b în nodul curent *nod*, care să presupunem că se află în stiva *Nodes* la poziția *pozB*.

Din cele două observații ne vine ideea să căutăm binar cea mai de mică poziție *pozMinA* pentru care putem fixa camera a în *Nodes[pozMinA]* încât să nu avem mai mult de C spiriduși pe lanțul de la camera a la camera b , calculând costul aşa cum am dedus în observația 1.

Acum ne rămâne doar să aflăm care este poziția *pozA* în care trebuie fixată camera a pentru a obține profitul maxim. Această poziție este, conform observațiilor 2 și 3, cea pentru care *Profits[pozA - 1]* este minimă.

Putem afla această valoare cu ajutorul dinamicii *dp*, ca fiind

$$\min(d[l][b], dp[l][Nodes[pozMinA + 2^l - 1]])$$

unde l este cel mai mare număr pentru care 2^l este mai mic decât *pozB - pozMinA + 1*.

Aceasta soluție are complexitate $O(N \log N)$ și obține 100 de puncte.

23.2.2 Cod sursă

Listing 23.2.1: spiridusi.cpp

```

1 #include <cstdio>
2 #include <cassert>
3 #include <vector>
4
5 using namespace std;
6
7 #define mlog 20
8 #define maxn 100010
9
10 int n, b, sol;
11 int s[maxn], p[maxn], t[maxn];

```

```

12 int stk0, stk[maxn], ps[maxn], pp[maxn];
13 int st[mlog][maxn], dp[mlog][maxn];
14 vector<int> v[maxn];
15
16 int cbin() // cautare binara
17 {
18     int left=0, right=stk0, sol=stk0;
19
20     while(left<=right)
21     {
22         int med=(left+right)/2;
23         if(ps[stk0]-ps[med]<=b)
24         {
25             sol=med;
26             right=med-1;
27         }
28         else    left=med+1;
29     }
30
31     return sol;
32 }
33
34 void df(int nod)
35 {
36     stk[++stk0]=nod;
37     ps[stk0]=s[nod]+ps[stk0-1];
38     pp[stk0]=p[nod]+pp[stk0-1];
39
40     int up = cbin();
41
42     st[0][nod]=stk[stk0-1];
43     dp[0][nod]=pp[stk0-1];
44
45     for(int i=1; st[i-1][st[i-1][nod]]!=0; ++i)
46     {
47         st[i][nod]=st[i-1][st[i-1][nod]];
48         dp[i][nod]=min(dp[i-1][nod], dp[i-1][st[i-1][nod]]);
49     }
50
51     if(up!=stk0)
52     {
53         int p2=0;
54         while((1<<(p2+1))<=stk0-up) ++p2;
55
56         sol=max(sol, pp[stk0]-min(dp[p2][nod], dp[p2][stk[up+(1<<p2)]]));
57     }
58
59     int fiu;
60
61     for(int i=0; i<v[nod].size(); ++i)
62     {
63         fiu=v[nod][i];
64         if(fiu==t[nod]) continue;
65
66         t[fiu]=nod;
67         df(fiu);
68     }
69
70     --stk0;
71 }
72
73 int main()
74 {
75     freopen("spiridusi.in", "r", stdin);
76     freopen("spiridusi.out", "w", stdout);
77
78     scanf("%d%d", &n, &b);
79     for(int i=1; i<=n; ++i) scanf("%d", &s[i]);
80     for(int i=1; i<=n; ++i) scanf("%d", &p[i]);
81
82     for(int i=1; i<n; ++i)
83     {
84         int a, b;
85
86         scanf("%d%d", &a, &b);
87         v[a].push_back(b);

```

```

88         v[b].push_back(a);
89     }
90
91     df(1);
92
93     printf("%d\n", sol);
94
95     return 0;
96 }
```

23.2.3 *Rezolvare detaliată

23.3 text

Problema 3 - text

100 de puncte

Un sir format din cifre trebuie să fie tastat în una sau mai multe sesiuni.

Există două tastaturi:

- tastatura *A* care conține taste cu toate combinațiile de exact două cifre: tasta 00, tasta 01, 02, ..., 98, 99 și
- tastatura *B* care conține taste cu toate combinațiile de exact trei cifre: tasta 000, tasta 001, ..., 998, 999.

Cifrele se vor introduce în una sau mai multe sesiuni, pentru o sesiune putându-se folosi o singură tastatură.

Datorită unei ordonanțe de urgență, dacă o combinație de taste a fost introdusă cu una din tastaturi în sesiunea curentă și, continuând sesiunea, această combinație poate fi introdusă din nou, este necesar să continuăm sesiunea cel puțin până când o vom introduce din nou.

în cazul în care introducem până atunci și alte taste, trebuie să continuăm sesiunea până când vom introduce ultima apariție a lor.

Astfel, dacă sirul 255222255257 este început folosind tastatura *A*, se va scrie obligatoriu într-o sesiune **25** **52** **22** **25** **52**. Suntem obligați să tastăm până la ultima apariție a tastei **25** în sesiunea curentă, și când folosim tasta **52** suntem obligați să continuăm până la ultima apariție a acesteia.

A se observa că cifrele de pe pozițiile subliniate sunt tot 2 și 5, însă nu formează o tastă care se poate apăsa în sesiunea curentă. Deoarece se dorește un număr cât mai mare de sesiuni, se va începe o nouă sesiune în care se va scrie doar **57**.

Cerințe

Cunoscându-se numărul total de cifre și secvența de cifre ce formează sirul, să se determine o modalitate de a despărți textul astfel încât el să poată fi scris într-un număr maxim de sesiuni.

Date de intrare

Din fișierul **text.in** se citesc:

- de pe prima linie un număr natural *N* reprezentând numărul de cifre
- de pe următoarea linie *N* cifre, scrise fără spații, reprezentând sirul de tastat

Date de ieșire

În fișierul **text.out** se afișează:

- pe prima linie *S*, reprezentând numărul maxim de sesiuni
- pe fiecare dintre următoarele *S* linii, câte două numere *p*, *k*, scrise cu spațiu între ele, fiecare astfel de pereche descriind, în ordinea în care apar în text, secvențele tipărite în sesiuni:

p_i - poziția din sirul de cifre dat unde începe sesiunea *i* și

k_i - tipul de tastatură folosită în sesiunea *i* (2 pentru tastatura *A*, 3 pentru tastatura *B*)

Restricții și precizări

- $3 \leq N \leq 1\ 000\ 000$
- cifrele din secvență sunt între 0 și 9
- testele propuse asigură existența unei soluții pentru cerința dată
- dacă există mai multe soluții, se va furniza oricare dintre ele.
- pentru numărul corect de sesiuni, fără liniile care descriu soluția completă și corectă se acordă 50% din punctaj.

Exemple:

text.in	text.out	Explicații
15 23332333333322	5 1 3 4 2 6 3 12 2 14 2	Șirul poate fi scris în maximum 5 sesiuni astfel: 233 32 333 333 33 22 - prima sesiune, începe cu prima cifră și folosește tastatura B (cu taste de 3 cifre) - următoarea sesiune începe la cifra a 3-a și folosește tastatura A - a treia sesiune începe la cifra a 6-a și folosește tastatura B - a patra sesiune începe la cifra a 12-a și folosește tastatura A - ultima sesiune începe la cifra a 14-a și folosește tastatura A
8 46234623	3 1 3 4 2 6 3	Soluția corespunde secvențelor 462 34 623

Timp maxim de executare/test: **0.4** secunde pe Windows, **0.2** secunde pe Linux

Memorie: total **64 MB**

Dimensiune maximă a sursei: **20 KB**

23.3.1 Indicații de rezolvare

Se observă că două taste nu pot fi introduse de pe aceeași tastatură decât dacă sunt pe poziții la distanță de multiplu de 2 (pentru tastatura A) sau de multiplu de 3 (pentru tastatura B).

Pentru fiecare tastă și fiecare rest posibil calculăm în tabela U ultima apariție a sa în sir.

Pentru fiecare poziție și fiecare tip de tastatură, vom calcula cu ajutorul tabelei U o altă tabelă de salt care memorează până unde vom fi obligați să ne continuăm sesiunea, dacă începem în poziția curentă.

Calculul acesteia se poate face în complexitate amortizată $O(N)$ cu memoizare, dacă parcuregem indicii descrescător.

Tot ce ne mai rămâne de făcut este o recurență simplă de programare dinamică pentru a determina cel mai mare număr de sesiuni pe care le putem avea dacă începem să tastăm cu fiecare tastatură, și ne ducem înainte la indicele obținut în tabela de salt în caz că tastăm următorul caracter cu tastatura respectivă.

23.3.2 Cod sursă

Listing 23.3.1: .cpp

```

1 //Mihai Ciucu
2 //Complexitate O(N)
3
4 #include <iostream>
5 #include <cmath>
6 #include <algorithm>
7
8 using namespace std;
9
10 #define MaxN (1 << 20)
11 #define STEP(x) ((x) + 2)
12
13 int N;
14 char str[MaxN];
15
16 int lastPosition[2][3000];
17 int memJump[MaxN][2];
18 int best[MaxN][2];
19

```

```

20 int getNum(int st, int type)
21 {
22     if (st + STEP(type) > N)
23         fprintf(stderr, "Eroare de overflow!\n");
24
25     if (type == 0)
26         return str[st] * 10 + str[st + 1];
27     else
28         return str[st] * 100 + str[st + 1] * 10 + str[st + 2];
29 }
30
31 int GetLastPoz(int poz, int type)
32 {
33     const int modulo = poz % STEP(type);
34     return lastPosition[type][modulo * 1000 + getNum(poz, type)];
35 }
36
37 void UpdateLastPoz(int poz, int type)
38 {
39     if (poz + STEP(type) > N)
40         return;
41
42     const int modulo = poz % STEP(type);
43     lastPosition[type][modulo * 1000 + getNum(poz, type)] = poz;
44 }
45
46 int CalcJump(int poz, int type)
47 {
48     if (memJump[poz][type])
49         return memJump[poz][type];
50
51     if (poz + STEP(type) >= N)
52         return poz + STEP(type);
53
54     int last = GetLastPoz(poz, type);
55     if (last + STEP(type) >= N || last == poz)
56     {
57         memJump[poz][type] = last + STEP(type);
58         return memJump[poz][type];
59     }
60
61     int curPoz = poz + STEP(type);
62     while (curPoz <= last)
63         curPoz = CalcJump(curPoz, type);
64
65     memJump[poz][type] = curPoz;
66     return memJump[poz][type];
67 }
68
69 int main()
70 {
71     freopen("text.in", "rb", stdin);
72     freopen("text.out", "wb", stdout);
73
74     scanf("%d\n", &N);
75     fgets(str, MaxN, stdin);
76     for (int i = 0; i < N; i++)
77         str[i] -= '0';
78
79     for (int i = 0; i < N; i++)
80     {
81         UpdateLastPoz(i, 0);
82         UpdateLastPoz(i, 1);
83     }
84
85     for (int i = 0; i <= N + 3; i++)
86         best[i][0] = best[i][1] = -999999999;
87
88     best[N][0] = best[N][1] = 0;
89
90     for (int i = N - 2; i >= 0; i--)
91         for (int type = 0; type < 2; type++)
92             if (i + STEP(type) <= N)
93             {
94                 int jumpPoz = CalcJump(i, type);
95                 best[i][type] = max(best[jumpPoz][0], best[jumpPoz][1]) + 1;

```

```

96         }
97
98     printf("%d\n", max(best[0][0], best[0][1]));
99
100    int poz = 0;
101   while (poz < N)
102   {
103       const int bestType = (best[poz][0] > best[poz][1]) ? 0 : 1;
104       printf("%d %d\n", poz + 1, bestType + 2);
105
106       poz = CalcJump(poz, bestType);
107   }
108
109   return 0;
110 }
```

23.3.3 *Rezolvare detaliată

23.4 arbvalmax

Problema 4 - arbvalmax

100 de puncte

Se dă un arbore cu N noduri numerotate de la 1 la N cu rădăcina în nodul 1. Fiecare nod din arborele dat are o valoare întreagă atașată. Se dau M întrebări de forma (x, y) , unde x este un strămoș al nodului y : dacă s-ar elimina toate nodurile de pe lanțul care unește x cu y (inclusiv nodurile x și y), care ar fi valoarea maximă din nodurile neeliminate?

Cerințe

Cunoscând numărul de noduri N , configurația arborelui, valorile atașate celor N noduri, și cele M întrebări, să se răspundă la fiecare întrebare dată.

Date de intrare

Fișierul de intrare **arbvalmax.in** conține pe prima linie două numere naturale N și M separate printr-un spațiu, reprezentând numărul de noduri ale arborelui, respectiv numărul de întrebări.

A doua linie a fișierului conține $N - 1$ numere naturale despărțite prin câte un spațiu. Al i -lea număr de pe această linie reprezintă părintele nodului cu indicele $i + 1$.

A treia linie a fișierului conține N numere întregi separate prin câte un spațiu. Al i -lea număr de pe această linie reprezintă valoarea atașată nodului cu indicele i .

Pe următoarele M linii se află câte două numere x, y separate prin câte un spațiu, reprezentând câte o întrebare de forma descrisă în enunț.

Date de ieșire

În fișierul de ieșire **arbvalmax.out** se vor afișa, câte unul pe linie, M numere reprezentând răspunsurile pentru cele M întrebări, în ordinea primită în fișierul de intrare.

Restricții și precizări

- $1 \leq N, M \leq 300\,000$
- $1 \leq valoare_i \leq 2\,000\,000\,000$, pentru orice i , $1 \leq i \leq N$.
- $1 \leq x, y \leq N$ Atenție! Nodul x este unul dintre nodurile de pe lanțul $1 - y$!
- Pentru 40% din teste, $N \leq 1000$ și $M \leq 10\,000$.
- Adâncimea maximă a arborelui nu va depăși valoarea de 100 000.

Exemple:

arbvalmax.in	arbvalmax.out	Explicații
8 3	6	Arborele conține următoarele muchii: 1-2, 2-3, 2-4, 1-5, 5-6, 4-7, 5-8.
1 2 2 1 5 4 5	10	Pentru prima întrebare, dacă s-ar elimina nodurile de pe lanțul 1-7 (1, 2, 4, 7), nodurile rămase ar fi: 3, 5, 6, 8 și ar avea valorile: 6, 3, 5, 4.
7 10 6 1 3 5 2 4	7	Dintre acestea valoarea maximă este 6.
1 7		
5 6		
2 3		

Timp maxim de executare/test: **2.5** secunde pe Windows, **0.8** secunde pe Linux

Memorie: total **64 MB** din care pentru stivă **16 MB**

Dimensiune maximă a sursei: **20 KB**

23.4.1 Indicații de rezolvare

Răzvan Dan Sălăjan

40 de puncte - $O(N * M)$

Pentru $N \approx 1\ 000$, pentru fiecare întrebare marcăm nodurile de pe lanț ca să delimităm. Răspunsul reprezintă valoarea maximă din nodurile nemarcate.

100 de puncte - $O(N + M)$

Pentru 100 de puncte ne vom folosi de următoarele dinamici:

- $dp1[i]$ = valoarea maximă din arbore daca nu luăm în considerare valorile de pe lanțul rădăcină - i și subarborele lui i (inclusiv i).
- $dp2[i]$ = valoarea maximă din arbore daca nu luăm în considerare valorile din subarborele lui i (inclusiv i).
- $bestValue[i][1]$ = valoarea maximă din subarborii filor lui i (excluzând pe i).
- $bestValue[i][2]$ = a doua valoare maximă din subarborii filor lui i (excluzând pe i).
- $bestSon[i]$ = fiul nodului i în care găsim valoarea maximă din subarborele lui i .

Vom face 2 *parcurgeri DF* pe arborele dat.

În prima parcurgere calculăm matricea $bestValue[i][j]$.

$bestValue[i][1] = \max(bestValue[fiu][1], valoare[fiu])$.

Pentru $bestValue[i][2]$ trebuie să fim atenți să nu alegem fiul pe care l-am ales deja pentru $bestValue[i][1]$ ($bestSon[i]$).

În a doua parcurgere vom calcula cele două dinamici:

Suntem în nodul i și calculăm dinamicile pentru fiile lui i :

- $dp1[fiu] = \max(dp1[i], bestSon[i][1])$. Dacă cel mai bun fiu al lui i e chiar fiu atunci o să ne uităm în $bestSon[i][2]$.
- $dp2[fiu] = \max(dp2[i], \max(valoare[i], bestSon[i][1]))$. La fel trebuie să fim atenți la cazul în care cel mai bun fiu al lui i e chiar fiul curent, atunci ne uităm în $bestSon[i][2]$.

Pentru o întrebare (x, y) (cu x stramoș al lui y) răspunsul este:

$\max(bestSon[y][1], \max(dp2[x], dp1[y]))$.

23.4.2 Cod sursă

Listing 23.4.1: arborevalmax.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <algorithm>
5 #include <set>
6 #include <queue>
7 #include <deque>
8 #include <cassert>
9
10 using namespace std;
11
12 ifstream f("arbvalmax.in");
13 ofstream g("arbvalmax.ok");
14
15 #define ll long long
16 #define pb push_back

```

```

17 #define mp make_pair
18 #define sz size
19 #define x first
20 #define y second
21
22 #define nmax 1000005
23 #define inf 2000000002
24
25 const int NMAX = 1e6;
26 const int MMAX = 1e6;
27 const int VALMAX = 2000000000;
28
29 int n, m, val[nmax], with[nmax], withOut[nmax];
30 vector<int> gf[nmax];
31 pair<int,int> bestSon[nmax][2];
32
33 void citeste()
34 {
35     f >> n >> m;
36
37     assert(n >= 1 && n <= NMAX);
38     assert(m >= 1 && m <= MMAX);
39
40     for(int i=2; i<=n; ++i)
41     {
42         int x; f >> x;
43         assert(x >= 1 && x <= n);
44         gf[x].pb(i);
45     }
46
47     for(int i=1; i<=n; ++i)
48     {
49         f >> val[i];
50         assert(val[i] >= 1 && val[i] <= VALMAX);
51     }
52 }
53
54 void inJos(int nod)
55 {
56     for(int i=0; i<(int)gf[nod].sz(); ++i)
57     {
58         int vc = gf[nod][i];
59         inJos(vc);
60     }
61     pair<int,int> best[2];
62     for(int i=0; i<2; ++i) best[i] = mp(-inf, -1);
63
64     for(int i=0; i<(int)gf[nod].sz(); ++i)
65     {
66         int vc = gf[nod][i];
67         if ( best[0].x < max(val[vc], bestSon[vc][0].x) )
68         {
69             best[1] = best[0];
70             best[0] = mp( max(val[vc], bestSon[vc][0].x), vc );
71         }
72         else
73             if ( best[1].x < max( val[vc], bestSon[vc][0].x ) )
74                 best[1] = mp( max(val[vc], bestSon[vc][0].x), vc );
75     }
76
77     for(int i=0; i<2; ++i)
78         bestSon[nod][i] = best[i];
79 }
80
81 void inSus(int nod)
82 {
83     for(int i=0; i<(int)gf[nod].sz(); ++i)
84     {
85         int vc = gf[nod][i];
86         if ( bestSon[nod][0].y == vc )
87             withOut[vc] = max( withOut[nod], bestSon[nod][1].x );
88         else
89             withOut[vc] = max( withOut[nod], bestSon[nod][0].x );
90
91         if ( bestSon[nod][0].y == vc )
92             with[vc] = max( with[nod], max( val[nod], bestSon[nod][1].x ) );

```

```

93         else
94             with[vc] = max( with[nod], max( val[nod], bestSon[nod][0].x ) );
95
96         inSus(vc);
97     }
98 }
99
100 void rezolva()
101 {
102     inJos(1);
103     withOut[1] = -inf; with[1] = -inf;
104     inSus(1);
105
106     for(int i=1; i<=m; ++i)
107     {
108         int nod, tNod;
109         f >> tNod >> nod;
110         assert(tNod >= 1 && tNod <= n);
111         assert(nod >= 1 && nod <= n);
112         int ans = max( bestSon[nod][0].x, max( withOut[nod], with[tNod] ) );
113         assert(ans >= 1 && ans <= VALMAX);
114         g << ans << "\n";
115     }
116 }
117
118 int main()
119 {
120     citeste();
121     rezolva();
122
123     f.close();
124     g.close();
125
126     return 0;
127 }
```

Listing 23.4.2: arbvalmax-cartita100.cpp

```

1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 #define maxn 1000010
7
8 int n, m, st0;
9 int d[maxn], fSon[maxn], sSon[maxn], val[maxn], lvl[maxn];
10 int a[maxn], b[maxn], sol[maxn], st[maxn], mp[maxn];
11 vector<int> v[maxn], q[maxn];
12
13 void df(int nod)
14 {
15     ++lvl[nod];
16
17     int fiu, b1=0, b2=0;
18
19     d[nod]=val[nod];
20
21     for(int i=0; i<v[nod].size(); ++i)
22     {
23         fiu=v[nod][i];
24         lvl[fiu]=lvl[nod];
25
26         df(fiu);
27
28         d[nod]=max(d[nod], d[fiu]);
29
30         if(d[b1]<=d[fiu])
31         {
32             b2=b1;
33             b1=fiu;
34         }
35         else
36             if(d[b2]<=d[fiu])
37                 b2=fiu;
```

```

38      }
39
40      fSon[nod]=b1;
41      sSon[nod]=b2;
42  }
43
44 void df2(int nod)
45 {
46     int fiu, l, nq;
47
48     ++st0;
49     mp[st0]=max(mp[st0-1], val[nod]);
50
51     for(int i=0; i<q[nod].size(); ++i)
52     {
53         nq=q[nod][i];
54         l=lvl[a[nq]];
55
56         sol[nq]=max(mp[l-1], max(st[st0-1], d[fSon[nod]]));
57     }
58
59     for(int i=0; i<v[nod].size(); ++i)
60     {
61         fiu=v[nod][i];
62
63         if(fiu==fSon[nod])
64             st[st0]=max(st[st0-1], d[sSon[nod]]);
65         else
66             st[st0]=max(st[st0-1], d[fSon[nod]]);
67
68         df2(fiu);
69     }
70
71     --st0;
72 }
73
74 int main()
75 {
76     freopen("arbvalmax.in", "r", stdin);
77     freopen("arbvalmax.out", "w", stdout);
78
79     scanf("%d%d", &n, &m);
80
81     for(int i=2; i<=n; ++i)
82     {
83         int x;
84         scanf("%d", &x);
85         v[x].push_back(i);
86     }
87
88     for(int i=1; i<=n; ++i)
89         scanf("%d", &val[i]);
90
91     for(int i=1; i<=m; ++i)
92     {
93         scanf("%d%d", &a[i], &b[i]);
94         q[b[i]].push_back(i);
95     }
96
97     df(1);
98     df2(1);
99
100    for(int i=1; i<=m; ++i)
101        printf("%d\n", sol[i]);
102
103    return 0;
104 }
```

Listing 23.4.3: arbvalmax-vladii.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <algorithm>
4 #include <cstdio>
5 #include <cstring>
```

```

6  #include <vector>
7
8  using namespace std;
9
10 #define maxn 1000100
11 #define INF 99999999
12
13 int N, Q;
14 int globalMaxVal;
15 int level[maxn], value[maxn], F[maxn];
16 int maxValWithoutPath[maxn];
17 int maxPath[maxn];
18 vector<int> G[maxn];
19
20 void DFS1(int node, int father)
21 {
22     level[node] = level[father] + 1;
23     F[node] = father;
24     maxPath[node] = max(maxPath[father], value[node]);
25
26     for (int i = 0; i < G[node].size(); i++)
27         if (G[node][i] != father)
28             DFS1(G[node][i], node);
29
30     maxValWithoutPath[node] = max(maxValWithoutPath[node], globalMaxVal);
31     globalMaxVal = max(globalMaxVal, value[node]);
32 }
33
34 void DFS2(int node, int father)
35 {
36     for (int i = G[node].size() - 1; i >= 0; i--)
37         if (G[node][i] != father)
38             DFS2(G[node][i], node);
39
40     maxValWithoutPath[node] = max(maxValWithoutPath[node], globalMaxVal);
41     globalMaxVal = max(globalMaxVal, value[node]);
42 }
43
44 int main()
45 {
46     fstream f1, f2;
47     f1.open("arbvalmax.in", ios::in);
48     f2.open("arbvalmax.out", ios::out);
49
50     f1 >> N >> Q;
51
52     maxValWithoutPath[0] = -INF;
53     maxPath[0] = -INF;
54
55     for (int i = 2; i <= N; i++)
56     {
57         int x, y;
58         //f1 >> x >> y;
59         f1 >> x;
60
61         G[x].push_back(i);
62         G[i].push_back(x);
63     }
64
65     for (int i = 1; i <= N; i++)
66     {
67         f1 >> value[i];
68
69         maxValWithoutPath[i] = -INF;
70         maxPath[i] = -INF;
71     }
72
73     globalMaxVal = -INF;
74     DFS1(1, 0);
75
76     globalMaxVal = -INF;
77     DFS2(1, 0);
78
79     while (Q--)
80     {
81         int x, y;

```

```

82         f1 >> x >> y;
83
84     if (level[x] > level[y])
85         swap(x, y);
86
87     f2 << max(maxPath[F[x]], maxValWithoutPath[y]) << '\n';
88 }
89
90 f1.close();
91 f2.close();
92
93 return 0;
94 }
```

23.4.3 *Rezolvare detaliată

23.5 ksecv

Problema 5 - ksecv

100 de puncte

Fie un vector V cu N elemente și un număr K . Vectorul V trebuie împărțit în exact K subsecvențe nevide, astfel încât fiecare element din vector să aparțină exact unei subsecvențe.

Această împărțire trebuie făcută astfel încât maximul șmecheriei fiecărei subsecvențe să fie cât mai mic. (Această problemă concepe greșit sistemul de șmecherie și valoare). șmecheria fiecărei subsecvențe se definește ca fiind parte întreagă din $((V_{max} - V_{min} + 1)/2)$, unde V_{max} este valoarea maximă din subsecvență, iar V_{min} este valoarea minimă.

Vectorul V de N elemente va fi generat în felul următor: se dă un număr M și 2 vectori A și B de lungime M (indexați de la 0 la $M - 1$). Fiecare element i , $0 \leq i < N$, din vectorul V va fi calculat cu următoarea formulă: $V[i] = (A[i \% M] ^ B[i / M])$, unde $x \% y$ reprezintă restul lui x la împărțirea cu y , x/y reprezintă câtul împărțirii lui x la y , și $x ^ y$ reprezintă rezultatul operației xor (sau exclusiv pe biți) dintre x și y .

Cerințe

Date de intrare

Pe prima linie a fișierului **ksecv.in** se află 3 numere naturale N , K și M , cu semnificația din enunț, separate prin câte un spațiu. Pe a doua linie a fișierului se află M numere naturale separate prin câte un spațiu, reprezentând vectorul A . Pe a treia linie se află M numere naturale separate prin câte un spațiu, reprezentând vectorul B .

Date de ieșire

Pe prima linie a fișierului de ieșire **ksecv.out** se va afișa cel mai mic număr natural S pentru care vectorul V poate fi împărțit în exact K subsecvențe nevide, fiecare având șmecherie mai mică sau egală cu S .

Restricții și precizări

- $1 \leq N \leq 1\,000\,000$
- $1 \leq K \leq 1\,000$
- $1 \leq M \leq 2\,048$
- $N < M * M$
- $1 \leq K < N$
- Valorile vectorilor A și B vor fi din intervalul $[0, 2^{60} - 1]$
- Fiecare din cele K subsecvențe trebuie să aibă cel puțin un element.
- Pentru 20% din teste $N \leq 100$, $K \leq 50$.
- Pentru alte 20% din teste $N \leq 100\,000$, $K \leq 1\,000$.
- Pentru alte 20% din teste $N \leq 1\,000\,000$, $K \leq 50$.
- Vectorul V are indicări indexați de la 0 la $N - 1$.
- Vectorii A și B au indicări indexați de la 0 la $M - 1$.

Exemplu:

ksecv.in	ksecv.out	Explicații
6 3 6 13 4 6 19 4 10 0 0 0 0 0 0	5	Valorile vectorului V sunt 13, 4, 6, 19, 4, 10. Dacă împărțim sirul în subsecvențele [0,2] [3,3] și [4,5] obținem şmecherile 5, 0 și 3. şmecheria maxima este 5. Nu putem împărți vectorul V astfel încât şmecheria maximă a unei subsecvențe să fie mai mică decât 5.
6 4 6 13 4 6 19 4 10 0 0 0 0 0 0	3	Valorile vectorului V sunt 13, 4, 6, 19, 4, 10. O posibilă împărțire este: [0,0] [1,2], [3,3], [4,5]. şmecherile fiecărei subsecvențe sunt 0, 1, 0 și 3.
6 3 3 3 4 2 4 5 3	3	Valorile vectorului V sunt 7, 0, 6, 6, 1, 7. Dacă împărțim în subsecvențele [0,0], [1,4] și [5,5], obținem şmecherile 0, 3 și 0.

Timp maxim de executare/test: **0.4** secunde pe Windows, **0.2** secunde pe Linux

Memorie: total **128 MB** din care pentru stivă **16 MB**

Dimensiune maximă a sursei: **20 KB**

23.5.1 Indicații de rezolvare

Eugenie Daniel Posdarascu, Mihai Ciucu, Radu Voroneanu

$O(N * K * K)$ - **20 de puncte**

Programare dinamică

$Best[i][j]$ = Maximul dacă se împart primele i numere în j grupe.

$O(N \log ValMax)$ - **40 de puncte**

Se cauta binar rezultatul, facand o functie de verificare *greedy* in felul urmator:

Pentru fiecare secvență nouă, incercam să ne intindem cat mai mult la drepta, cat timp condiția $(MaxCurrent - MinCurrent + 1)/2 \leq valoare$. Dacă putem împărți astfel sirul în K sau mai puține subsecvențe, updatăm soluția curentă și scădem valoarea căutată. Altfel, creștem valoarea căutată.

$O(N + \log ValMax * K * \log^2 N)$ - **40 de puncte**

Rafinăm soluția de mai sus:

Construim în $O(N)$ doi *arbori de intervale*, unul ținând minimul valorilor pe interval, altul ținând maximul valorilor. Cu ajutorul acestora putem să facem fiecare din cele K testări de extindere a intervalului curent la dreapta căutând binar capătul subsecvenței cu care extinedem, și testând maximul și minimul pe subsecvența cu care am dori să extindem.

$O(N + \log ValMax * K * \log N)$ - **100 de puncte**

îmbunătățim și mai mult soluția de mai sus dacă în loc de căutare binară, căutăm direct pe *arborii de intervale* secvența cu care extindem.

$O(N + \log ValMax * (N/B + \min(K * B, N)))$, cu B ales optim (ciucuială) - **100 de puncte**

Consideram vectorul împărțit în blocuri de câte B elemente. Reținem minimul și maximul pentru fiecare bloc, și facem o îmbunătățire a soluției în $O(N \log ValMax)$.

Dacă ne aflăm cu elementul curent la începutul unui bloc, încercăm să adăugăm câte un bloc întreg la secvența curentă. Deoarece împărțim în maxim K subsecvențe, va trebui să parcurgem element cu element maxim K blocuri, deci pe total $K * B$ elemente, iar pe restul le vom sări. B trebuie ales încât $N/B = K * B$, deci $B = \sqrt{N/K}$.

23.5.2 Cod sursă

Listing 23.5.1: ksecv.cpp

```

1 #include <cstdio>
2 #include <algorithm>
3

```

```

4  using namespace std;
5
6 #define maxn 1000010
7 #define inf (1LL*1000000000*1000000000)
8 #define bucketSize (32)
9
10 int n, k, m;
11 long long a[maxn], b[maxn], v[maxn];
12 long long bucketMn[maxn], bucketMx[maxn];
13
14 intincearca(long long med)
15 {
16     int kk=1;
17
18     long long mn=inf;
19     long long mx=-inf;
20
21     for(int i=0; i<n; ++i)
22     {
23         if(i%bucketSize==0 && i+bucketSize<n)
24         {
25             long long mnt = min(bucketMn[i/bucketSize], mn);
26             long long mxt = max(bucketMx[i/bucketSize], mx);
27
28             if((mxt-mnt+1)/2 <= med)
29             {
30                 mn=mnt;
31                 mx=mxt;
32                 i+=bucketSize-1;
33             }
34         }
35
36         mn = min(v[i], mn);
37         mx = max(v[i], mx);
38
39         if((mx-mn+1)/2 > med)
40         {
41             ++kk;
42             mx=mn=v[i];
43             if(kk>k)
44                 return 0;
45         }
46     }
47
48     return 1;
49 }
50
51 int main()
52 {
53     freopen("ksecv.in", "r", stdin);
54     freopen("ksecv.out", "w", stdout);
55
56     scanf("%d%d%d", &n, &k, &m);
57
58     for(int i=0; i<m; ++i)
59         scanf("%lld", &a[i]);
60
61     for(int i=0; i<m; ++i)
62         scanf("%lld", &b[i]);
63
64     for(int i=0; i<n; ++i)
65         v[i] = (a[i%m] ^ b[i/m]);
66
67     for(int i=0; i<n; i+=bucketSize)
68     {
69         long long mn=inf;
70         long long mx=-inf;
71
72         for(int j=i; j<n && j<i+bucketSize; ++j)
73         {
74             mn=min(mn, v[j]);
75             mx=max(mx, v[j]);
76         }
77
78         bucketMn[i/bucketSize]=mn;
79         bucketMx[i/bucketSize]=mx;

```

```

80     }
81
82     long long left=0, right=inf, sol=inf;
83
84     while(left<=right)
85     {
86         long long med=(left+right)/2;
87
88         if(incearca(med))
89         {
90             sol=med;
91             right=med-1;
92         }
93         else
94             left=med+1;
95     }
96
97     printf("%lld\n", sol);
98
99     return 0;
100 }
```

Listing 23.5.2: ksecv-mihai.cpp

```

1 #include <cstdio>
2 #include <ctime>
3 #include <cstdlib>
4 #include <algorithm>
5
6 using namespace std;
7
8 #define MaxN 1000100
9
10 #define INF (1ll << 60)
11 #define Number long long
12 #ifdef WIN32
13 #define FORMAT_LONG "%I64d"
14 #else
15 #define FORMAT_LONG "%lld"
16 #endif //WIN32
17
18 #define BLOCK_SIZE 64
19
20 int N, K, M;
21 Number val[MaxN];
22 Number minBlock[MaxN / BLOCK_SIZE], maxBlock[MaxN / BLOCK_SIZE];
23
24 void ReadInput(const char *inputFile)
25 {
26     double start = clock();
27     Number a[5000], b[5000];
28     FILE *input = fopen(inputFile, "rb");
29
30     fscanf(input, "%d %d %d", &N, &K, &M);
31     for (int i = 0; i < M; i++)
32         fscanf(input, FORMAT_LONG, a + i);
33
34     for (int i = 0; i < M; i++)
35         fscanf(input, FORMAT_LONG, b + i);
36
37     for (int i = 0; i <= N / BLOCK_SIZE + 1; i++)
38     {
39         minBlock[i] = INF;
40         maxBlock[i] = -INF;
41     }
42
43     for (int i = 0; i < N; i++)
44     {
45         val[i] = a[i % M] ^ b[i / M];
46         if (val[i] < minBlock[i / BLOCK_SIZE])
47             minBlock[i / BLOCK_SIZE] = val[i];
48
49         if (val[i] > maxBlock[i / BLOCK_SIZE])
50             maxBlock[i / BLOCK_SIZE] = val[i];
51     }
}
```

```

52     fclose(input);
53     /* fprintf(stderr, "Durata citire + generare: %.0f ms\n",
54                1000.0 * (clock() - start) / CLOCKS_PER_SEC); */
55 }
56 }
57 bool CanSolve(Number maxDif)
58 {
59     maxDif = 2 * maxDif;
60     int poz = 0, nrGroups = 0;
61     while (nrGroups < K && poz < N)
62     {
63         Number curMin = val[poz], curMax = val[poz];
64         while (poz < N)
65         {
66             poz++;
67             if (val[poz] < curMin)
68             {
69                 curMin = val[poz];
70                 if (curMax - curMin > maxDif) break;
71             }
72             if (val[poz] > curMax)
73             {
74                 curMax = val[poz];
75                 if (curMax - curMin > maxDif) break;
76             }
77         }
78         nrGroups++;
79     }
80     return poz == N;
81 }
82 }
83 void FindSolution(Number st = 0, Number fn = INF)
84 {
85     while (st < fn)
86     {
87         Number mid = (st + fn) / 2;
88         if (CanSolve(mid))
89             fn = mid;
90         else
91             st = mid + 1;
92     }
93     printf(FORMAT_LONG"\n", st);
94 }
95 }
96 int main()
97 {
98     double start = clock();
99     freopen("ksecv.out", "wb", stdout);
100    ReadInput("ksecv.in");
101    FindSolution();
102    fprintf(stderr, "Durata: %.0f ms\n", (clock() - start));
103    return 0;
104 }

```

23.5.3 *Rezolvare detaliată

23.6 trenuri

Problema 6 - trenuri

100 de puncte

Gara de Nord este cea mai vestită gară din lume. Japonezii, invidioși pe sistemul performant de întârziere al trenurilor din Gara de Nord, s-au hotărât să analizeze motivul realizării unei astfel de performanțe.

În Gara de Nord (considerată stația 0) există N trenuri. Pentru fiecare tren i știm că va pleca din Gara noastră protagonistă (stația 0) și o să meargă până la stația $statie_i$. Stațiile x și $x + 1$ sunt legate în mod direct pentru orice x , astfel că trenul i va opri în toate stațiile din intervalul $[0, statie_i]$. De asemenea, știm că trenul i are o capacitate egală cu numărul maxim de oameni pe care îl poate transporta. Această capacitate este notată cu $capacitate_i$.

Avem M pasageri dornici să folosească magnificul traseu. Pentru fiecare pasager i știm intervalul de stații $[a_i, b_i]$ pe care vrea să îl parcurgă. Mai exact, acesta vrea să se urce într-un tren în stația a_i și să coboare în stația b_i .

Cerințe

Din cauza capacitatei limitate a trenurilor, este posibil ca nu toți pasagerii să poată obțină un loc și să ajungă în destinația dorită. Să se determine numărul maxim de pasageri care pot ajunge din stația de plecare în stația de sosire, precum și o configurație în care aceștia se pot urca în trenuri.

Date de intrare

Pe prima linie a fișierului **trenuri.in** se află 2 numere naturale N și M , separate printr-un spațiu, cu semnificația din enunț.

Următoarele N linii vor descrie cele N trenuri. Pe linia $i + 1$ se vor afla două valori întregi separate prin câte un spațiu: $stati_{ei}$ și $capacitate_{ei}$ care descriu trenul cu numărul i .

Următoarele M linii vor descrie itinerariile celor M pasageri. Astfel, pe linia $N + i + 1$ se vor afla două valori întregi a_i și b_i , separate printr-un spațiu reprezentând stațiile între care dorește să circule pasagerul cu numărul i .

Date de ieșire

Pe prima linie a fișierului **trenuri.out** se va afișa un număr natural P , reprezentând numărul maxim de pasageri care pot să își realizeze traseul propus. Pe următoarele M linii se vor afișa M numere naturale. Astfel, pe linia $i + 1$ se va afișa trenul în care va urca pasagerul i . Dacă pasagerul i nu poate să se urce în niciun tren, se va afișa valoarea 0.

Restricții și precizări

- $1 \leq N, M \leq 100\,000$
- $1 \leq stati_{ei}, capacitate_{ei} \leq 1\,000\,000\,000$ pentru orice i , $1 \leq i \leq N$.
- $1 \leq a_i, b_i \leq 1\,000\,000\,000$ pentru orice i , $1 \leq i \leq M$.
- Un pasager nu poate să coboare dintr-un tren și să ia alt tren. Pasagerul i poate să urce doar în stația a_i și să coboare doar la stația b_i .
- Pot exista mai multe soluții pentru repartizarea pasagerilor în trenuri. Orice soluție cu număr maxim de pasageri posibil va obține punctaj maxim.
 - Pentru afișarea numărului corect de pasageri se va acorda 30% din punctajul pe un test.
 - Pentru 20% din teste $N = 1$.
 - Pentru 60% din teste $N, M \leq 2\,000$.
 - Trenurile sunt indexate de la 1 la N .

Exemple:

trenuri.in	trenuri.out	Explicații
2 3	3	Primul și ultimul pasager vor urca în trenul 2, iar pasagerul 2 în trenul 1.
10 1	2	
15 1	1	Dacă pasagerul 1 s-ar fi urcat în trenul 1, ar fi trebuit să alegem care dintre pasagerul 2 și 3 să se urce în trenul 2 deoarece cele 2 itinerarii se intersectează, iar cei doi pasageri nu ar avea loc în același tren.
2 8	2	
7 10		
8 13		
1 3	2	Orice combinație în care selectăm 2 din cei 3 pasageri se consideră validă.
10 2	1	
1 5	0	
3 7	1	
4 9		

Timp maxim de executare/test: **2.0** secunde pe Windows, **1.0** secunde pe Linux

Memorie: total **64 MB** din care pentru stivă **16 MB**

Dimensiune maximă a sursei: **20 KB**

23.6.1 Indicații de rezolvare

Eugenie Daniel Posdarascu

20 de puncte

Pentru $N = 1$. Fie 3 tipuri de evenimente:

1. Urcă un pasager;
2. Coboară un pasager;
3. Se oprește trenul.

Parcurgem evenimentele *sortate* după stație de la stânga la dreapta.

Când urcă un pasager avem 2 variante:

- 1) Avem încă loc în tren și îl urcăm.
- 2) Nu mai avem loc în tren, caz în care putem să îl urcăm doar dacă dăm alt pasager jos.

Aplicând o *strategie greedy*, este clar că trebuie să păstrăm pasagerii care coboară cât mai repede.

Concluzia în acest caz este că o să dăm jos pasagerul care coboară ultimul.

Pentru asta putem să ne întreținem o *structură de tip heap* care să păstreze mereu pasagerul care coboară ultimul.

Evenimentul 3 este relativ irelevant, trebuie avut grija doar ca un pasager să urce în tren doar dacă urcă și coboară înainte ca trenul să ajungă în stația finală.

100 de puncte

Din păcate acest greedy nu merge adaptat și pentru $N > 1$ (un exemplu se află în enunț), deoarece contează în momentul în care vrei să urci un pasager în ce tren îl atribui. Pentru a obține 100 de puncte vom aplica aceeași *strategie greedy* doar că o să luăm evenimentele *de la dreapta la stânga*.

Evenimentele acum vor fi în felul următor:

1. Urcă un pasager (în realitate coboară);
2. Coboară un pasager (în realitate urcă);
3. Apare un tren (avantajul acum este că trenul nu se mai oprește, merge până în capăt (stația 0)).

În momentul în care un pasager vrea să urce nu mai contează în ce tren îl punem deoarece toate trenurile acum duc până în capăt, deci putem să îl punem în primul liber (atribuirea trenurilor se poate face cu un set).

Strategia Greedy rămâne în schimb aceeași: dacă urcă un pasager și toate trenurile disponibile sunt ocupate, o să dăm jos pasagerul care coboară primul (în realitate primul care urcă).

Este irrelevant în ce tren se află pasagerul pe care îl dăm jos, deoarece în locul lui oricum urcăm un pasager nou.

23.6.2 Cod sursă

Listing 23.6.1: trenuri100.cpp

```

1 #include<stdio.h>
2 #include<set>
3 #include<algorithm>
4
5 using namespace std;
6
7 #define pi pair<int,int>
8 #define pii pair<pi, pi >
9 #define x first
10 #define y second
11 #define dist x.x
12 #define extra x.y
13 #define type y.x
14 #define index y.y
15 #define mp make_pair
16 #define NMAX 100005
17
18 pi v[NMAX];

```

```

19 pii event[3 * NMAX];
20 int n,m,capacitate,mysize,nr;
21 int sol, tren[NMAX];
22
23 class cmpset
24 {
25 public:
26     /bool operator() (const int& a,const int& b)
27     bool operator() (const int& a,const int& b) const
28     {
29         if(v[a].x == v[b].x)
30             return a < b;
31         return v[a].x < v[b].x;
32     }
33 };
34
35 set<int,cmpset> myset;
36 multiset<int> trainset;
37
38 inline int cmp(const pii& a, const pii& b)
39 {
40     if(a.dist == b.dist)
41     {
42         if(a.type == b.type)
43             return a.extra > b.extra;
44         return a.type < b.type;
45     }
46     return a.dist > b.dist;
47 }
48
49 int main ()
50 {
51     int i,a,b,aux,val;
52
53     freopen("trenuri.in","r",stdin);
54     freopen("trenuri.ok","w",stdout);
55
56     scanf("%d%d",&n,&m);
57     for(i = 1; i <= n; i++)
58     {
59         scanf("%d%d",&a,&b);
60         event[++nr] = mp(mp(a,b),mp(0,i));
61     }
62     for(i = 1; i <= m; i++)
63     {
64         scanf("%d%d",&a,&b);
65         event[++nr] = mp(mp(b,0),mp(1,i));
66         event[++nr] = mp(mp(a,1),mp(1,i));
67         v[i] = mp(a,b);
68     }
69
70     sort(event + 1, event + nr + 1, cmp);
71
72     mysize = 0;
73
74     for(i = 1; i <= nr; i++)
75     {
76         if(event[i].type == 0)
77         {
78             if(capacitate + event[i].extra < m)
79             {
80                 capacitate += event[i].extra;
81                 aux = event[i].extra;
82             }
83             else
84             {
85                 aux = m - capacitate;
86                 capacitate = m;
87             }
88             while(aux)
89             {
90                 trainset.insert(event[i].index);
91                 aux--;
92             }
93             continue;
94         }

```

```

95         if(event[i].extra == 0)
96     {
97         if(mysize < capacitate)
98         {
99             mysize++;
100            tren[event[i].index] = *trainset.begin();
101            trainset.erase(trainset.begin());
102            myset.insert(event[i].index);
103        }
104    else
105    {
106        val = *myset.begin();
107        if(v[event[i].index].x > v[val].x)
108        {
109            tren[event[i].index] = tren[val];
110            tren[val] = 0;
111            myset.erase(val);
112            myset.insert(event[i].index);
113        }
114    }
115 }
116 else
117 {
118     if(!tren[event[i].index])
119         continue;
120     sol++;
121     mysize--;
122     myset.erase(event[i].index);
123     trainset.insert(tren[event[i].index]);
124 }
125 }
126
127 printf("%d\n",sol);
128 for(i = 1; i <= m; i++)
129     printf("%d\n", tren[i]);
130 printf("\n");
131
132 return 0;
133 }
```

Listing 23.6.2: trenuri-andrei-ciocan.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <set>
5 #include <algorithm>
6
7 using namespace std;
8
9 #define MAX_N 100005
10 #define MAX_M 100005
11 #define PASAGER_COBOARA 2
12 #define PASAGER_URCA 1
13 #define FINAL_TREN 0
14
15 struct pasager_t
16 {
17     int left;
18     int right;
19     int tren;
20 };
21
22 struct event
23 {
24     int state;
25     int pozition;
26     int indice;
27     event (int s, int p, int i)
28     {
29         state= s; pozition = p; indice = i;
30     }
31 };
32
33 vector<pair<int, int> > sol;
```

```

34  pasager_t p[MAX_N];
35  vector<event *> ev;
36  int n, m, tren[MAX_N], cap[MAX_N], used[MAX_N], using_train[MAX_N];
37
38  bool sort_ev(const event *ev1, const event *ev2)
39  {
40      return ev1->pozition > ev2->pozition
41          || ev1->pozition == ev2->pozition && ev1->state < ev2->state;
42  }
43
44  void prepare()
45  {
46      for (int i = 1; i <= n; i++)
47      {
48          ev.push_back(new event(PASAGER_URCA, p[i].left, i));
49          ev.push_back(new event(PASAGER_COBOARA, p[i].right, i));
50      }
51
52      for (int i = 1; i <= m; i++)
53          ev.push_back(new event(FINAL_TREN, tren[i], i));
54
55      sort(ev.begin(), ev.end(), sort_ev);
56  }
57
58  void solve()
59  {
60      int tren_ind;
61      // indicele trenurilor
62
63      set<int> trenuri_free;
64      set<int> trenuri_full;
65      // pair< left_interval, indice>
66      set<pair<int, int> > pasageri;
67
68      for (int i = 0 ; i < ev.size() ; i++)
69      {
70          int pasager, left_pasager;
71          switch (ev[i]->state)
72          {
73              case FINAL_TREN:
74                  trenuri_free.insert(ev[i]->indice);
75                  break;
76
77              case PASAGER_COBOARA:
78                  pasager = ev[i]->indice;
79                  using_train[pasager] = -1;
80                  if (trenuri_free.size() > 0)
81                  {
82                      int tren = *(trenuri_free.begin());
83                      used[tren]++;
84                      using_train[pasager] = tren;
85                      pasageri.insert(make_pair(p[ev[i]->indice].left,
86                                              ev[i]->indice));
87                      if (used[tren] == cap[tren])
88                      {
89                          set<int> :: iterator it = trenuri_free.begin();
90                          trenuri_free.erase(it);
91                          trenuri_full.insert(tren);
92                      }
93                  }
94                  else
95                  {
96                      if (pasageri.size() > 0)
97                      set<pair<int, int> :: iterator p_begin=pasageri.begin();
98                      left_pasager = (*p_begin).second;
99                      if (p[left_pasager].left < p[pasager].left)
100                     {
101                         set<pair<int, int> :: iterator ir=pasageri.begin();
102                         pasageri.erase(ir);
103                         pasageri.insert(make_pair(p[pasager].left,
104                                                 pasager));
105                         int x = using_train[left_pasager];
106                         using_train[left_pasager] = -1;
107                         using_train[pasager] = x;
108                     }
109                 }
110             }
111         }
112     }

```

```

110         break;
111
112     case PASAGER_URCA:
113         pasager = ev[i]->indice;
114         int used_train = using_train[pasager];
115         if (used_train == -1)
116             break;
117         sol.push_back(make_pair(pasager, used_train));
118         used[used_train] --;
119         if (used[used_train] == cap[used_train] - 1)
120         {
121             set<int > :: iterator it2 = trenuri_full.find(used_train);
122             trenuri_full.erase(it2);
123             trenuri_free.insert(used_train);
124         }
125         set<pair<int , int> > :: iterator it3 = pasageri.find(
126             make_pair(ev[i]->pozition, ev[i]->indice));
127         pasageri.erase(it3);
128         break;
129     }
130 }
131
132 void citire()
133 {
134     ifstream f("trenuri.in");
135     f >> m >> n;
136
137     for (int i = 1; i <= m; i++)
138         f >> tren[i] >> cap[i];
139
140     for (int i = 1; i <= n; i++)
141         f >> p[i].left >> p[i].right;
142
143     f.close();
144 }
145
146 void afisare()
147 {
148     ofstream g("trenuri.out");
149     g << sol.size() << '\n';
150
151     sort(sol.begin(), sol.end());
152
153     for (int i = 1; i <= n; i++)
154         using_train[i] = 0;
155
156     for (int i = 0; i < sol.size(); i++)
157         using_train[sol[i].first] = sol[i].second;
158
159     for (int i = 1 ; i <= n; i++)
160         g << using_train[i] << '\n';
161
162     g.close();
163 }
164
165 int main()
166 {
167     citire();
168     prepare();
169     solve();
170     afisare();
171     return 0;
172 }
173 }
```

Listing 23.6.3: trenuri-cartita100.cpp

```

1 #include <iostream>
2 #include <map>
3 #include <vector>
4 #include <algorithm>
5 #include <set>
6
7 using namespace std;
8
```

```

9 #define maxn 400010
10
11 int n, m, nr;
12 int sol[maxn], st[maxn], cap[maxn], a[maxn], b[maxn];
13
14 map<int, int> mp;
15 vector<pair<int, int> > load[maxn];
16 vector<int> capAdd[maxn], trains[maxn];
17 set<pair<int, int> > hsf, hlf;
18
19 int willRide[maxn];
20 int where[maxn], seat[maxn];
21
22 //Reconst
23 int seats[maxn];
24 set<int> av, oc;
25
26 int main()
27 {
28     freopen("trenuri.in", "r", stdin);
29     freopen("trenuri.out", "w", stdout);
30
31     scanf("%d%d", &n, &m);
32     mp[0]=1;
33     for(int i=1; i<=n; ++i)
34     {
35         scanf("%d%d", &st[i], &cap[i]);
36         mp[st[i]]=1;
37     }
38
39     for(int i=1; i<=m; ++i)
40     {
41         scanf("%d%d", &a[i], &b[i]);
42         mp[a[i]]=1;
43         mp[b[i]]=1;
44     }
45
46     nr=-1;
47     for(map<int, int> :: iterator it = mp.begin(); it != mp.end(); ++it)
48         it->second = ++nr;
49
50     for(int i=1; i<=n; ++i)
51     {
52         st[i]=mp[st[i]];
53         capAdd[st[i]].push_back(cap[i]);
54         trains[st[i]].push_back(i);
55     }
56
57     for(int i=1; i<=m; ++i)
58     {
59         a[i]=mp[a[i]];
60         b[i]=mp[b[i]];
61         load[b[i]].push_back(make_pair(a[i], i));
62     }
63
64     int avSeats=0;
65
66     for(int i=nr; i>=0; --i)
67     {
68         //descarcam pasagerii
69
70         while(hsf.size()>0)
71         {
72             pair<int, int> current = *hlf.begin();
73             current.first *= -1;
74
75             if(current.first<i)
76                 break;
77
78             willRide[current.second] = 1;
79             hsf.erase(current);
80
81             current.first *= -1;
82             hlf.erase(current);
83         }
84

```

```

85      //adaugam trenurile noi
86      for(int j=0; j<trains[i].size(); ++j)
87          avSeats = min(m, avSeats + capAdd[i][j]);
88
89      //incarcam pasagerii
90
91      for(int j=0; j<load[i].size(); ++j)
92      {
93          pair<int, int> current = load[i][j];
94
95          hsf.insert(current);
96          current.first *= -1;
97          hlf.insert(current);
98
99          if(hsf.size() > avSeats)
100         {
101             current = *hsf.begin();
102             hsf.erase(current);
103
104             current.first *= -1;
105             hlf.erase(current);
106         }
107     }
108 }
109
110 int sol=0;
111
112 for(int i=1; i<=m; ++i)
113     sol+=willRide[i];
114
115 printf("%d\n", sol);
116
117 hsf.clear();
118 hlf.clear();
119
120 avSeats=0;
121
122 for(int i=nr; i>=0; --i)
123 {
124     //descarcam pasagerii
125
126     while(hsf.size()>0)
127     {
128         pair<int, int> current = *hlf.begin();
129         current.first *= -1;
130
131         if(current.first<i)
132             break;
133
134         av.insert(seat[current.second]);
135         oc.erase(seat[current.second]);
136         hsf.erase(current);
137
138         current.first *= -1;
139         hlf.erase(current);
140     }
141
142     //adaugam trenurile noi
143     for(int j=0; j<trains[i].size(); ++j)
144     {
145         for(int k=0; k<capAdd[i][j] && avSeats<m; ++k)
146         {
147             where[++avSeats]=trains[i][j];
148             av.insert(avSeats);
149         }
150     }
151
152     //incarcam pasagerii
153
154     for(int j=0; j<load[i].size(); ++j)
155     {
156         pair<int, int> current = load[i][j];
157         if(willRide[current.second]==0)
158             continue;
159
160         hsf.insert(current);

```

```

161         current.first *= -1;
162         hlf.insert(current);
163
164         seat[current.second]=*av.begin();
165         oc.insert( *av.begin());
166         av.erase(av.begin());
167     }
168 }
169
170 for(int i=1; i<=m; ++i)
171     printf("%d\n", where[seat[i]]);
172
173 return 0;
174 }
```

Listing 23.6.4: trenuri-HH-100.cpp

```

1 /**
2 * Author: Andrei Heidelbacher
3 * Expected score: 100 points
4 */
5 #include <iostream>
6 #include <vector>
7 #include <algorithm>
8 #include <set>
9
10 using namespace std;
11
12 const char IN_FILE[] = "trenuri.in";
13 const char OUT_FILE[] = "trenuri.out";
14 const int NIL = -1;
15
16 class Event
17 {
18     public:
19         int time, type, index;
20
21     Event(const int _time, const int _type, const int _index):
22         time(_time),
23         type(_type),
24         index(_index) {}
25
26     bool operator<(const Event &other) const
27     {
28         if (time != other.time)
29             return time < other.time;
30         if (type != other.type)
31             return type < other.type;
32         return index < other.index;
33     }
34 };
35
36 vector< pair<int, int> > Passengers, Trains;
37 vector<int> AssignedTrains;
38 int N, T;
39
40 void Solve()
41 {
42     set<Event> events;
43     for (int i = 0; i < T; ++i)
44         events.insert(Event(Trains[i].first, 0, i));
45     for (int i = 0; i < N; ++i)
46     {
47         events.insert(Event(Passengers[i].second, 1, i));
48         events.insert(Event(Passengers[i].first, 2, i));
49     }
50
51     vector<int> activeTrains;
52     set< pair<int, int> > travellingPassengers;
53
54     int capacity = 0;
55     for (; !events.empty(); events.erase(events.begin()))
56     {
57         Event e = *events.begin();
58         if (e.type == 0)
```

```

59         {
60             activeTrains.push_back(e.index);
61             capacity += Trains[e.index].second;
62         }
63     else if (e.type == 1 && AssignedTrains[e.index] != NIL)
64     {
65         if (Trains[AssignedTrains[e.index]].second == 0)
66             activeTrains.push_back(AssignedTrains[e.index]);
67             ++Trains[AssignedTrains[e.index]].second;
68             travellingPassengers.erase(make_pair(e.time, e.index));
69     }
70     else if (e.type == 2 && capacity > 0)
71     {
72         if (int(travellingPassengers.size()) == capacity)
73         {
74             int last = travellingPassengers.rbegin() ->second;
75             if (Passengers[e.index].second < Passengers[last].second)
76             {
77                 AssignedTrains[e.index] = AssignedTrains[last];
78                 AssignedTrains[last] = NIL;
79                 travellingPassengers.erase(
80                     *travellingPassengers.rbegin());
81                 travellingPassengers.insert(
82                     make_pair(Passengers[e.index].second, e.index));
83             }
84         }
85     else if (!activeTrains.empty())
86     {
87         AssignedTrains[e.index] = activeTrains.back();
88         --Trains[activeTrains.back()].second;
89         if (Trains[activeTrains.back()].second == 0)
90             activeTrains.pop_back();
91         travellingPassengers.insert(
92             make_pair(Passengers[e.index].second, e.index));
93     }
94 }
95 }
96 }
97
98 void Read()
99 {
100    ifstream cin(IN_FILE);
101    cin >> T >> N;
102
103    Trains = vector< pair<int, int> >(T);
104    for (int i = 0; i < T; ++i)
105    {
106        cin >> Trains[i].first >> Trains[i].second;
107        Trains[i].first = -Trains[i].first;
108    }
109
110    Passengers = vector< pair<int, int> >(N);
111    for (int i = 0; i < N; ++i)
112    {
113        cin >> Passengers[i].second >> Passengers[i].first;
114        Passengers[i].first = -Passengers[i].first;
115        Passengers[i].second = -Passengers[i].second;
116    }
117
118    AssignedTrains = vector<int>(N, NIL);
119
120    cin.close();
121 }
122
123 void Print()
124 {
125    ofstream cout(OUT_FILE);
126    int count = 0;
127    for (int i = 0; i < N; ++i)
128        if (AssignedTrains[i] != NIL)
129            ++count;
130    cout << count << "\n";
131    for (int i = 0; i < N; ++i)
132        cout << AssignedTrains[i] + 1 << "\n";
133    cout.close();
134 }

```

```

135
136 int main()
137 {
138     Read();
139     Solve();
140     Print();
141     return 0;
142 }
```

Listing 23.6.5: trenuri-vladii.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <algorithm>
4 #include <vector>
5 #include <set>
6 #include <cstdio>
7 #include <cstring>
8
9 using namespace std;
10
11 #define maxn 100100
12 #define mkp make_pair
13 #define pii pair<int, int>
14 #define x first
15 #define y second
16
17 int N, M;
18 int pass[maxn], trainCap[maxn];
19 pii passengerInfo[maxn];
20
21 vector<pair<int, pii> > events;
22 vector<int> trains;
23 set<pair<int, pii> > assoc;
24
25 int cmp(pair<int, pii> a, pair<int, pii> b)
26 {
27     if (a.first != b.first)
28         return a.first > b.first;
29     return a.second.second > b.second.second;
30 }
31
32
33 int main()
34 {
35     fstream f1, f2;
36     f1.open("trenuri.in", ios::in);
37     f2.open("trenuri.out", ios::out);
38
39     f1 >> M >> N;
40
41     for (int i = 1; i <= M; i++)
42     {
43         int s, c;
44         f1 >> s >> c;
45
46         events.push_back(mkp(s, mkp(c, i))); // New train.
47     }
48
49     for (int i = 1; i <= N; i++)
50     {
51         int a, b;
52         f1 >> a >> b;
53
54         passengerInfo[i].x = a;
55         passengerInfo[i].y = b;
56
57         events.push_back(mkp(b, mkp(i, -1))); // Passenger goes up.
58         events.push_back(mkp(a, mkp(i, +0))); // Passenger goes down.
59     }
60
61     sort(events.begin(), events.end(), cmp);
62
63     for (int i = 0; i < events.size(); i++)
64     {
```

```

65     if (events[i].second.second > 0)
66     {
67         // New train.
68         trains.push_back(events[i].second.second);
69         trainCap[events[i].second.second] = events[i].second.first;
70     }
71     else if (events[i].second.second == 0)
72     {
73         // Passenger goes down.
74         int associated_train = pass[events[i].second.first];
75         if (associated_train > 0)
76         {
77             trainCap[associated_train]++;
78
79             if (trainCap[associated_train] == 1)
80                 trains.push_back(associated_train);
81
82             set<pair<int, pii>>::iterator it =
83                 assoc.find(mkp(events[i].first,
84                                 mkp(events[i].second.first, associated_train)));
85             if (it != assoc.end())
86                 assoc.erase(it);
87         }
88     }
89     else
90     {
91         // Passenger goes up.
92         if (trains.size() > 0)
93         {
94             pass[events[i].second.first] = trains.back();
95             assoc.insert(mkp(passengerInfo[events[i].second.first].x,
96                               mkp(events[i].second.first, trains.back())));
97
98             trainCap[trains.back()]--;
99             if (trainCap[trains.back()] == 0)
100                trains.pop_back();
101         }
102     }
103     else
104     {
105         // Maybe we can remove another passenger? :)
106         int passengerId = events[i].second.first;
107
108         if (assoc.size())
109         {
110             set<pair<int, pii>>::iterator it = assoc.begin();
111             if ((*it).first < passengerInfo[passengerId].x)
112             {
113                 trainCap[( *it).second.second]++;
114                 if (trainCap[( *it).second.second] == 1)
115                     trains.push_back(( *it).second.second);
116
117                 pass[( *it).second.first] = 0;
118
119                 assoc.erase(it);
120                 i--;
121
122                 continue;
123             }
124         }
125     }
126 }
127 }
128
129 int sol = 0;
130 for (int i = 1; i <= N; i++)
131     if (pass[i])
132         sol++;
133
134 cout << sol << endl;
135
136 f2 << sol << '\n';
137 for (int i = 1; i <= N; i++)
138     f2 << pass[i] << '\n';
139
140 f1.close();

```

```
141     f2.close();  
142     return 0;  
143 }  
_____
```

23.6.3 *Rezolvare detaliată

Capitolul 24

ONI 2014

24.1 avârcolaci

Problema 1 - avârcolaci

100 de puncte

Un vârcolac bântuie ulițele satului Bosston, semănând panică printre săteni. Satul Bosston este compus din $2 * N$ săteni, fiecare dintre aceștia fiind rudă cu exact un vârcolac. Vârcolacii sunt codificați cu numere naturale. Pentru a afla care este vârcolacul care le cauzează probleme, aceștia s-au dus la vraciu local. Acesta a spus că, dacă există un vârcolac V astfel încât oricum să împărți cei $2 * N$ săteni în două grupuri de N săteni, există cel puțin un sătean în primul grup și cel puțin un sătean în al doilea care să fie rude cu V , atunci vârcolacul V sigur este cel care bântuie satul. Dacă nu există un astfel de vârcolac, atunci sătenii nu își pot da seama cine le bântuie satul.

Cerințe

Cunoscând N și indicii vârcolacilor cu care se înrudează fiecare dintre cei $2 * N$ săteni, să se determine vârcolacul care bântuie satul, în cazul în care acesta există.

Date de intrare

Fișierul de intrare **avarcolaci.in** conține pe prima linie numărul T de teste. Următoarele $2 * T$ linii conțin cele T teste. Prima linie dintr-un test conține numărul natural N , indicând faptul că avem $2 * N$ săteni în Bosston. Următoarea linie conține $2 * N$ elemente, al i -lea dintre aceste elemente reprezentând indicele vârcolacului cu care este rudă al i -lea sătean.

Date de ieșire

Fișierul de ieșire **avarcolaci.out** va conține T linii, câte una pentru fiecare test din fișierul de intrare. Dacă vârcolacul care bântuie satul nu poate fi determinat, atunci se va afișa textul "Mozart" (fără ghilimele). Dacă vârcolacul poate fi determinat, atunci se va afișa indicele acestuia.

Restricții și precizări

- $1 \leq T \leq 15$
- $1 \leq N \leq 500.000$
- Pentru 10% din teste se garantează că $N = 1$
- Pentru 20% din teste se garantează că $N < 10$
- Pentru 40% din teste se garantează că $N \leq 500$
- Pentru 80% din teste se garantează că $N \leq 50.000$
- Indicii vârcolacilor sunt numere întregi pozitive mai mici ca 10^9
- Pentru 50% din teste indicii vârcolacilor sunt mai mici strict ca 32.762
- ATENȚIE! Satul conține $2 * N$ săteni!
- ATENȚIE la limita de memorie!
- ATENȚIE! În cazul în care sunt mai multe soluții pentru vârcolacul care bântuie satul, se acceptă oricare dintre ele.

Exemple:

aavarcolaci.in	aavarcolaci.out	Explicații
3	Mozart	1. Dacă împățim în grupuri vârcolacii asociați celor 4 săteni obținem (1, 3) (4, 4) și nu avem un vârcolac atât în primul grup, cât și în al doilea.
2	1	2. Vârcolacul 1 bântuie satul.
1 4 3 4	4	3. Vârcolacul 4 bântuie satul - nu putem împărți sătenii în două grupuri astfel încât 4 să nu fie în niciunul dintre ele.
1		
1 1		
3		
4 1 4 4 4 4		

Timp maxim de executare/test: **5.0** secunde

Memorie: total **3 MB** din care pentru stivă **1 MB**

Dimensiune maximă a sursei: **25 KB**

24.1.1 Indicații de rezolvare

stud. Andrei Pârvu, Universitatea "Politehnica" București

10 puncte

Pentru N egal cu 1, adică 2 săteni, tot ce trebuie să facem este să verificăm dacă vârcolacii cu care sunt înruditi cei doi sunt egali. Dacă da, atunci vârcolacul bântuie satul, altfel nu.

20 puncte

Pentru fiecare vârcolac cu care este rudă vreun sătean, putem simula prin *backtracking* împărțirea celor 2^N săteni în două grupuri de căte N, și să vedem dacă există vreo împărțire în care vârcolacul nu este prezent într-unul din cele două grupuri. Dacă există, atunci vârcolacul respectiv sigur nu bântuie satul. Complexitate $O(N^2 * 2^N)$.

40 puncte

Observăm că, dacă există un vârcolac care bântuie satul, atunci acesta trebuie să apară de cel puțin $N+1$ ori în sirul de 2^N vârcolaci cu care sunt rude sătenii. Putem lua fiecare vârcolac și să vedem de căte ori apare în sir. Dacă există vreunul care apare de cel puțin $N+1$ ori, atunci acesta bântuie satul. Complexitate $O(N^2)$.

80 puncte

Putem sorta sirul de 2^N vârcolaci și observăm că dacă există un vârcolac care bântuie satul acesta trebuie să se afle pe poziția N în sirul sortat. După acesta trebuie doar verificat dacă apare de cel puțin $N+1$ ori. Complexitate $O(N \log N)$ sau $O(N)$, dacă folosim *statistici de ordine* pentru aflarea celui de-al N-lea element în sirul sortat.

100 puncte

Datorită limitei de memorie, nu putem ține tot sirul de 2^N vârcolaci în memorie. Putem parcurge sirul pe măsura ce îl citim și să reținem un contor cu elementul probabil de a fi majoritar. Dacă citim un element care este diferit de cel probabil, decrementăm contorul, iar dacă citim un element egal cu cel probabil, îl incrementăm. Când contorul ajunge la 0, atunci înlocuim elementul probabil cu cel curent.

La final, trebuie să mai parcurgem o dată sirul ca să verificăm dacă elementul probabil chiar apare de cel puțin $N+1$ ori. Deoarece nu am reținut sirul în memorie, trebuie să *închidem fișierul* de intrare, să îl *deschidem din nou* pentru citire și să *citim din nou* sirul.

Complexitate temporală O(N), complexitate spațială O(1).

24.1.2 Cod sursă

Listing 24.1.1: aavarcolaci-10-andrei.c

```

1 /* Andrei Parvu
2  ONI 2014
3 */
4 #include <stdio.h>
5 #include <assert.h>
6
7 #define NMAX 50000
8 const int TMAX = 15;
```

```

9  const int EL_MAX = 1e9;
10
11 int v[2 * NMAX];
12
13 int main()
14 {
15     freopen("avarcolaci.in", "rt", stdin);
16     freopen("avarcolaci.out", "wt", stdout);
17
18     int nrTests;
19     scanf("%d", &nrTests);
20     assert(1 <= nrTests && nrTests <= TMAX);
21
22     int test;
23     for (test = 1; test <= nrTests; test++)
24     {
25         int n;
26         scanf("%d", &n);
27         assert(1 <= n && n <= NMAX);
28
29         int i;
30         for (i = 0; i < 2 * n; i++)
31         {
32             scanf("%d", v + i);
33             assert(0 <= v[i] && v[i] <= EL_MAX);
34         }
35
36         if (v[0] == v[1])
37             printf("%d\n", v[0]);
38         else
39             printf("Mozart\n");
40     }
41
42     return 0;
43 }
```

Listing 24.1.2: avarcolaci-20-andrei.c

```

1 /* Andrei Parvu
2  ONI 2014
3 */
4 #include <stdio.h>
5 #include <assert.h>
6
7 #define NMAX 50000
8
9 const int TMAX = 15;
10 const int EL_MAX = 1e9;
11
12 int v[2 * NMAX];
13
14 int main()
15 {
16     freopen("avarcolaci.in", "rt", stdin);
17     freopen("avarcolaci.out", "wt", stdout);
18
19     int nrTests;
20     scanf("%d", &nrTests);
21     assert(1 <= nrTests && nrTests <= TMAX);
22
23     int test;
24     for (test = 1; test <= nrTests; test++)
25     {
26         int n;
27
28         scanf("%d", &n);
29         assert(1 <= n && n <= NMAX);
30
31         int i;
32         for (i = 0; i < 2 * n; i++)
33         {
34             scanf("%d", v + i);
35             assert(0 <= v[i] && v[i] <= EL_MAX);
36         }
37 }
```

```

38     for (i = 0; i < 2 * n; i++)
39     {
40         int noGood = 0;
41         int j;
42         for (j = 0; j < (1 << (2 * n)); j++)
43         {
44             int done1 = 0, done2 = 0;
45             int nr1 = 0;
46
47             int k;
48             for (k = 0; k < 2 * n; k++)
49             {
50                 if ((j & (1 << k)) > 0)
51                 {
52                     nr1++;
53                     if (v[k] == v[i])
54                         done1 = 1;
55                 }
56                 else
57                     if (v[k] == v[i])
58                         done2 = 1;
59             }
60
61             if (nr1 == n && (done1 == 0 || done2 == 0))
62             {
63                 noGood = 1;
64                 break;
65             }
66         }
67
68         if (!noGood)
69         {
70             printf("%d\n", v[i]);
71             break;
72         }
73     }
74
75     if (i == 2 * n)
76         printf("Mozart\n");
77 }
78
79 return 0;
80 }
```

Listing 24.1.3: avarcolaci-40-andrei.c

```

1 /* Andrei Parvu
2  ONI 2014
3 */
4 #include <stdio.h>
5 #include <assert.h>
6
7 #define NMAX 50000
8
9 const int TMAX = 15;
10 const int EL_MAX = 1e9;
11
12 int v[2 * NMAX];
13
14 int main()
15 {
16     freopen("avarcolaci.in", "rt", stdin);
17     freopen("avarcolaci.out", "wt", stdout);
18
19     int nrTests;
20     scanf("%d", &nrTests);
21     assert(1 <= nrTests && nrTests <= TMAX);
22
23     int test;
24     for (test = 1; test <= nrTests; test++)
25     {
26         int n;
27
28         scanf("%d", &n);
29         assert(1 <= n && n <= NMAX);
```

```

30
31     int i;
32     for (i = 0; i < 2 * n; i++)
33     {
34         scanf("%d", v + i);
35         assert(0 <= v[i] && v[i] <= EL_MAX);
36     }
37
38     for (i = 0; i < 2 * n; i++)
39     {
40         int nrEl = 0;
41
42         int j;
43         for (j = 0; j < 2 * n; j++)
44             if (v[i] == v[j])
45                 nrEl++;
46
47         if (nrEl > n)
48         {
49             printf("%d\n", v[i]);
50             break;
51         }
52     }
53
54     if (i == 2 * n)
55         printf("Mozart\n");
56 }
57
58 return 0;
59 }
```

Listing 24.1.4: avarcolaci-20-mihai-random.cpp

```

1 //Mihai Popa
2 //bulan
3 #include<stdio.h>
4 #include<time.h>
5 #include<algorithm>
6 #include<cstring>
7 #include<cassert>
8 #include<vector>
9 #include<cstdlib>
10
11 #define maxn 50005
12 #define mod 100003
13
14 using namespace std;
15
16 int n, ch;
17 int a[maxn<<1];
18
19 vector<int>toErase;
20 vector< pair<int, pair<int,int> > >H[mod+5];
21
22 inline void insert ( const int &x , const int &step )
23 {
24     int h = x % mod;
25
26     for ( vector< pair<int,pair<int,int> > ::iterator itt = H[h].begin() ;
27          itt != H[h].end() ; ++itt )
28     {
29         if ( itt->first == x )
30         {
31             if ( itt->second.second == step )    return ;
32             ++itt->second.first;
33             itt->second.second = step;
34             return ;
35         }
36     }
37
38     if ( H[h].empty() ) toErase.push_back(h);
39     H[h].push_back(make_pair(x,make_pair(1,step)));
40     return ;
41 }
42 }
```

```

43 int main ()
44 {
45     freopen("avarcolaci.in","r",stdin);
46     freopen("avarcolaci.out","w",stdout);
47
48     int tests;
49     scanf("%d",&tests);
50
51     srand(time(NULL));
52     while ( tests-- )
53     {
54         scanf("%d",&n); int sol = -1;
55         for ( int i = 1 ; i <= (n<<1) ; ++i )
56         {
57             scanf("%d",&a[i]);
58             assert(a[i] >= 0 && a[i] <= 1000000000);
59         }
60
61         int steps = min(100,10000000/n);
62
63         for ( int step = 1 ; step <= steps ; ++step )
64         {
65             random_shuffle(a+1,a+n+n+1);
66
67             for ( int i = 1 ; i <= n ; ++i )
68                 insert(a[i],step);
69
70             for ( int i = n+1 ; i <= (n<<1) ; ++i )
71                 insert(a[i],step+steps);
72         }
73
74         for ( int i = 0 ; i < (int)toErase.size() ; ++i )
75         {
76             int h = toErase[i];
77             for(vector< pair<int,pair<int,int>>>::iterator itt = H[h].begin();
78                  itt != H[h].end() ; ++itt)
79             {
80                 if ( itt->second.first == (steps<<1) )
81                     sol = itt->first;
82             }
83         }
84
85         if ( sol != -1 )
86             printf("%d\n",sol);
87         else
88             printf("Mozart\n");
89
90         for ( int i = 0 ; i < (int)toErase.size() ; ++i )
91             H[toErase[i]].clear();
92
93         toErase.clear();
94     }
95
96     return 0;
97 }
```

Listing 24.1.5: avarcolaci-30-adrian-8bit.cpp

```

1 //Solutie Panaete Adrian
2 //Rupere in sechete de 8 biti
3 #include <cstdio>
4 #include <cstring>
5 #define N 2000005
6
7 using namespace std;
8
9 const int STEPS = 4;
10 const int CHUNK = 8;
11 const int POW = 1 << CHUNK;
12 int t,n,maj,ans[15],cnt[STEPS][POW],sol_p1(int),sol_p2(int);
13
14 int main()
15 {
16     freopen("avarcolaci.in","r",stdin);
17     freopen("avarcolaci.out","w",stdout);

```

```

18
19     int T;
20     scanf("%d", &T);
21     for(int t = 0; t < T; t++)
22     {
23         sol_p1(t);
24         if (ans[t] == 0)
25             printf("Mozart\n");
26         else
27             printf("%d\n", ans[t]);
28     }
29 // Aici trebuia a doua parcurgere
30     return 0;
31 }
32
33 int sol_p1(int t)
34 {
35     int i,j,a;
36     memset(cnt,0,sizeof(cnt));
37     scanf("%d",&n);maj=n+1;n<=1;
38     for(i=1;i<=n;i++)
39     {
40         scanf("%d",&a);
41         for(j=0;j<STEPS;j++)
42         {
43             cnt[j][a&(POW - 1)]++;
44             a>>=CHUNK;
45         }
46     }
47
48     for(i=STEPS-1;i>=0;i--)
49     {
50         for(j=0;j<POW;j++)
51             if(cnt[i][j]>=maj)
52                 break;
53         if(j==POW) return -1;
54         a=(a<<CHUNK) | j;
55     }
56
57     ans[t] = a;
58     return a;
59 }
60
61 int sol_p2(int t)
62 {
63     int i,a,Cnt;
64     scanf("%d", &n);maj=n+1;n<=1;
65     for(Cnt=0,i=1;i<=n;i++)
66     {
67         scanf("%d", &a);
68         if(ans[t]==a) Cnt++;
69     }
70
71     return Cnt>=maj?ans[t]:-1;
72 }

```

Listing 24.1.6: avarcolaci-40-adrian-16bit.cpp

```

1 //Solutie Panaete Adrian
2 //Buchete de 16 biti
3 #include <cstdio>
4 #include <cstring>
5 #include <vector>
6
7 #define Mask 65535
8
9 using namespace std;
10
11 vector<int> P[65535],S[65535];
12 int t,i,n,maj,v;
13
14 int main()
15 {
16     freopen("avarcolaci.in","r",stdin);
17     freopen("avarcolaci.out","w",stdout);

```

```

18     scanf("%d", &t);
19     for(;t;t--)
20     {
21         for(i=0;i<=Mask;i++) {P[i].resize(0);S[i].resize(0);}
22         scanf("%d", &n);maj=n+1;n<<=1;
23         for(;n;n--)
24         {
25             scanf("%d", &v);
26             P[v&Mask].push_back(v);
27         }
28     }
29
30     for(i=0;i<=Mask;i++)
31         if(P[i].size()>=maj)
32             break;
33
34     if(i==Mask+1)
35     {
36         printf("Mozart\n");
37         continue;
38     }
39
40     for(vector<int>::iterator it=P[i].begin();it!=P[i].end();it++)
41         S[*it>>16].push_back(*it );
42
43     for(i=0;i<=Mask;i++)
44         if(S[i].size()>=maj)
45         {
46             printf("%d\n",S[i].front());
47             break;
48         }
49
50     if(i==Mask+1)
51         printf("Mozart\n");
52 }
53
54     return 0;
55 }
```

Listing 24.1.7: avarcolaci-50-mihai-parsare.cpp

```

1 //Mihai Popa
2 //hashuri + parsare
3 #include<stdio.h>
4 #include<algorithm>
5 #include<cstring>
6 #include<cassert>
7
8 #include <vector>
9
10#define maxn 500005
11#define buff_size 500000
12#define mod 100003
13
14 using namespace std;
15
16 int n,ch;
17 char buff[buff_size+5];
18 vector<int>toErase;
19
20 inline int nextInt()
21 {
22
23     while ( !(buff[ch] >= '0' && buff[ch] <= '9') )
24     {
25         ++ch;
26         if ( ch == buff_size )
27         {
28             fread(buff,buff_size,1,stdin);
29             ch = 0;
30         }
31     }
32
33     int r = 0;
34     while ( buff[ch] >= '0' && buff[ch] <= '9' )
```

```

35     {
36         r = r * 10 + buff[ch]-'0';
37         ++ch;
38         if ( ch == buff_size )
39         {
40             fread(buff,buff_size,1,stdin);
41             ch = 0;
42         }
43     }
44
45     return r;
46 }
47
48 vector< pair<int,int> >H[mod+5];
49
50 inline int insert ( const int &x )
51 {
52     int h = x % mod;
53
54     for ( vector< pair<int,int> >::iterator itt = H[h].begin() ;
55                                     itt != H[h].end() ; ++itt )
56     {
57         if ( itt->first == x )
58         {
59             ++itt->second;
60             if ( itt->second > n )
61                 return 1;
62             return 0;
63         }
64     }
65
66     if ( H[h].empty() ) toErase.push_back(h);
67     H[h].push_back(make_pair(x,1));
68     return 0;
69 }
70
71 int main ()
72 {
73
74     freopen("avarcolaci.in","r",stdin);
75     freopen("avarcolaci.out","w",stdout);
76
77     fread(buff,buff_size,1,stdin);
78
79     int tests;
80     tests = nextInt();
81
82     while ( tests-- )
83     {
84         n = nextInt(); int sol = -1;
85         for ( int i = 1 ; i <= (n<<1) ; ++i )
86         {
87             int x = nextInt();
88             assert(x >= 0 && x <= 1000000000);
89             if ( sol == -1 )
90                 if ( insert(x) )
91                     sol = x;
92         }
93
94         if ( sol != -1 )
95             printf("%d\n",sol);
96         else
97             printf("Mozart\n");
98
99         for ( int i = 0 ; i < (int)toErase.size() ; ++i )
100            H[toErase[i]].clear();
101
102        toErase.clear();
103    }
104
105    return 0;
106 }
```

Listing 24.1.8: avarcolaci-70-mihai.cpp

```

1 //Mihai Popa
2 //hashuri
3 #include<stdio.h>
4 #include<algorithm>
5 #include<cstring>
6 #include<cassert>
7 #include<vector>
8
9 #define maxn 500005
10 #define mod 100003
11
12 using namespace std;
13
14 int n,ch;
15 vector<int>toErase;
16 vector< pair<int,int> >H[mod+5];
17
18 inline int insert ( const int &x )
19 {
20     int h = x % mod;
21
22     for ( vector< pair<int,int> >::iterator itt = H[h].begin() ; itt != H[h].end() ; ++
23         itt )
24     {
25         if ( itt->first == x )
26         {
27             ++itt->second;
28             if ( itt->second > n )
29                 return 1;
30             return 0;
31         }
32     }
33
34     if ( H[h].empty() ) toErase.push_back(h);
35     H[h].push_back(make_pair(x,1));
36     return 0;
37 }
38
39 int main ()
40 {
41     freopen("avarcolaci.in","r",stdin);
42     freopen("avarcolaci.out","w",stdout);
43
44     int tests;
45     scanf("%d",&tests);
46
47     while ( tests-- )
48     {
49         scanf("%d",&n); int sol = -1;
50         for ( int i = 1 ; i <= (n<<1) ; ++i )
51         {
52             int x; scanf("%d",&x);
53             assert(x >= 0 && x <= 1000000000);
54             if ( sol == -1 )
55                 if ( insert(x) )
56                     sol = x;
57
58             if ( sol != -1 )
59                 printf("%d\n",sol);
60             else
61                 printf("Mozart\n");
62
63             for ( int i = 0 ; i < (int)toErase.size() ; ++i )
64                 H[toErase[i]].clear();
65
66             toErase.clear();
67     }
68
69     return 0;
70 }
```

Listing 24.1.9: avarcolaci-80-adrian-8bit.cpp

```
1 //Rupere in sechete de 8 biti
```

```

2 #include <cstdio>
3 #include <cstring>
4
5 #define N 100005
6
7 using namespace std;
8
9 int t,n,maj,ans,cnt[4][256],x[N],sol();
10
11 int main()
12 {
13     freopen("avarcolaci.in","r",stdin);
14     freopen("avarcolaci.out","w",stdout);
15
16     scanf("%d", &t);
17     for(;t;t--)
18     {
19         ans=sol();
20         ans<0?printf("Mozart\n"):printf("%d\n",ans);
21     }
22
23     return 0;
24 }
25
26 int sol()
27 {
28     int i,j,a,Cnt;
29     memset(cnt,0,sizeof(cnt));
30     scanf("%d",&n);maj=n+1;n<=1;
31     for(i=1;i<=n;i++)
32     {
33         scanf("%d",&a);
34         x[i]=a;
35         for(j=0;j<4;j++)
36         {
37             cnt[j][a&255]++;
38             a>>=8;
39         }
40     }
41
42     for(i=3;i>=0;i--)
43     {
44         for(j=0;j<256;j++)
45             if(cnt[i][j]>=maj)
46                 break;
47         if(j==256) return -1;
48         a=(a<<8)|j;
49     }
50
51     for(Cnt=0,i=1;i<=n;i++)
52         if(x[i]==a) Cnt++;
53     return Cnt>=maj?a:-1;
54 }
```

Listing 24.1.10: avarcolaci-80-andrei.cpp

```

1 /* Andrei Parvu
2  341C3
3 */
4 #include <cstdio>
5 #include <cassert>
6 #include <algorithm>
7
8 using namespace std;
9
10 const int NMAX = 50000;
11 const int TMAX = 15;
12 const int EL_MAX = 1e9;
13
14 int v[2 * NMAX];
15
16 int main()
17 {
18     freopen("avarcolaci.in", "rt", stdin);
19     freopen("avarcolaci.out", "wt", stdout);
```

```

20
21     int nrTests;
22     scanf("%d", &nrTests);
23     assert(1 <= nrTests && nrTests <= TMAX);
24
25     for (int test = 1; test <= nrTests; test++)
26     {
27         int n;
28
29         scanf("%d", &n);
30         assert(1 <= n && n <= NMAX);
31
32         for (int i = 0; i < 2 * n; i++)
33         {
34             scanf("%d", &v[i]);
35             assert(0 <= v[i] && v[i] <= EL_MAX);
36         }
37
38         nth_element(v + 0, v + n, v + 2 * n);
39         int maj = v[n];
40         int nrMaj = 0;
41
42         for (int i = 0; i < 2 * n; i++)
43             if (v[i] == maj)
44                 nrMaj++;
45
46         if (nrMaj > n)
47             printf("%d\n", maj);
48         else
49             printf("Mozart\n");
50     }
51
52     return 0;
53 }
```

Listing 24.1.11: avarcolaci-80-andrei-sort.cpp

```

1  /* Andrei Parvu
2   ONI 2014
3 */
4 #include <cstdio>
5 #include <cassert>
6 #include <algorithm>
7
8 using namespace std;
9
10 const int NMAX = 50000;
11 const int TMAX = 15;
12 const int EL_MAX = 1e9;
13
14 int v[2 * NMAX];
15
16 int main()
17 {
18     freopen("avarcolaci.in", "rt", stdin);
19     freopen("avarcolaci.out", "wt", stdout);
20
21     int nrTests;
22     scanf("%d", &nrTests);
23     assert(1 <= nrTests && nrTests <= TMAX);
24
25     for (int test = 1; test <= nrTests; test++)
26     {
27         int n;
28         scanf("%d", &n);
29         assert(1 <= n && n <= NMAX);
30
31         for (int i = 0; i < 2 * n; i++)
32         {
33             scanf("%d", v + i);
34             assert(0 <= v[i] && v[i] <= EL_MAX);
35         }
36
37         sort(v + 0, v + 2 * n);
38 }
```

```

39     int elMaj = v[n];
40     int nrCnt = 0;
41
42     for (int i = 0; i < 2 * n; i++)
43         if (v[i] == elMaj)
44             nrCnt++;
45
46     if (nrCnt > n)
47         printf("%d\n", elMaj);
48     else
49         printf("Mozart\n");
50 }
51
52     return 0;
53 }
```

Listing 24.1.12: avarcolaci-80-bogdan.cpp

```

1 /*
2  * Olimpiada NaÈzionalÄC de InformaticÄC, PiteÈzti 2014
3  * Ål' 2014 Bogdan Cristian TÄtÄcroiu
4  */
5 #include <iostream>
6 #include <cassert>
7 #include <vector>
8
9 using namespace std;
10
11 const int MAXT = 15;
12 const int MAXN = 500000;
13 const int MAXV = 999999999;
14
15 int main()
16 {
17     assert(freopen("avarcolaci.in", "rt", stdin));
18 #ifndef DEBUG
19     assert(freopen("avarcolaci.out", "wt", stdout));
20 #endif
21
22     int T, N;
23     vector<int> v;
24
25     // We read the count of tests
26     assert(scanf("%d", &T) == 1);
27     assert(1 <= T && T <= MAXT);
28     for (; T; T--)
29     {
30         // And then we do our bests
31         // We read the count of peasants
32         assert(scanf("%d", &N) == 1);
33         assert(1 <= N && N <= MAXN);
34
35         // And allocate memory like adolescents
36         v.clear();
37         v.reserve(2 * N);
38
39         // I'm not a god damn poet
40         for (int i = 0; i < 2 * N; i++)
41         {
42             int val;
43             assert(scanf("%d", &val) == 1);
44             assert(0 <= val && val <= MAXV);
45             v.push_back(val);
46         }
47
48         // And I'm sure that by now you know it
49         int MAJ = v[0], MAJCNT = 1;
50         for (int i = 1; i < 2 * N; i++)
51         {
52             if (v[i] == MAJ)
53             {
54                 MAJCNT += 1;
55             }
56             else
57             {
```

```

58             MAJCNT -= 1;
59             if (MAJCNT <= 0)
60             {
61                 MAJ = v[i];
62                 MAJCNT = 1;
63             }
64         }
65     }
66
67     // But this is a pretty good start
68     MAJCNT = 0;
69     for (int i = 0; i < 2 * N; i++)
70     {
71         if (v[i] == MAJ)
72             MAJCNT += 1;
73
74         // For this task is more classic than Mozart
75         // ( Pentru cĂC e mai clasicĂ ca MoÈžart )
76         if (MAJCNT >= N + 1)
77             printf("%d\n", MAJ);
78         else
79             printf("Mozart\n");
80     }
81
82     return 0;
83 }
```

Listing 24.1.13: avarcolaci-80-mihai-sortare.cpp

```

1 //Mihai Popa
2 //radix Sort
3 #include<stdio.h>
4 #include<algorithm>
5 #include<cstring>
6
7 #define maxn 120005
8 #define buff_size 80000
9
10 using namespace std;
11
12 int n,ch;
13 const int base = 1023,pbase = 10;
14 int a[maxn<<1],AUX[maxn<<1],Count[base];
15 char buff[buff_size+5];
16
17 inline int nextInt ()
18 {
19     while ( !(buff[ch] >= '0' && buff[ch] <= '9') )
20     {
21         ++ch;
22         if ( ch == buff_size )
23         {
24             fread(buff,buff_size,1,stdin);
25             ch = 0;
26         }
27     }
28
29     int r = 0;
30     while ( buff[ch] >= '0' && buff[ch] <= '9' )
31     {
32         r = r * 10 + buff[ch]-'0';
33         ++ch;
34         if ( ch == buff_size )
35         {
36             fread(buff,buff_size,1,stdin);
37             ch = 0;
38         }
39     }
40
41     return r;
42 }
43
44 inline void radix ( int *A , const int &n )
45 {
46     memset(AUX,0,sizeof(AUX));
47     memset(Count,0,sizeof(Count));
```

```

48
49     int step = 0;
50     for ( int pasi = 1 ; pasi <= 4; ++pasi )
51     {
52         for ( int i = 1 ; i <= n ; ++i )
53             ++Count[ (A[i]>>step)&base ];
54
55         for ( int i = 0 ; i <= base ; ++i )
56             Count[i] += Count[i-1];
57
58         for ( int i = n ; i >= 1 ; --i )
59         {
60             int last = ((A[i]>>step)&base);
61             AUX[ Count[last]-- ] = A[i];
62         }
63
64         step += pbase;
65         for ( int i = 1 ; i <= n ; ++i )
66             A[i] = AUX[i];
67
68         for ( int i = 0 ; i <= base ; ++i )
69             Count[i] = 0;
70     }
71 }
72
73 int main () {
74
75     freopen("avarcolaci.in","r",stdin);
76     freopen("avarcolaci.out","w",stdout);
77
78     fread(buff,buff_size,1,stdin);
79
80     int tests;
81     tests = nextInt ();
82
83     while ( tests-- )
84     {
85         n = nextInt ();
86         for ( int i = 1 ; i <= (n<<1) ; ++i )
87             a[i] = nextInt ();
88
89         radix(a,n+n);
90
91         int cons = 0,found = -1;
92         for ( int i = 1 ; i <= (n<<1) ; ++i )
93             if ( a[i] != a[i-1] )
94             {
95                 if ( cons > n )
96                     found = a[i-1]; break ;
97                 cons = 1;
98             }
99             else
100                 ++cons;
101
102         if ( found == -1 && cons > n )
103             found = a[n<<1];
104
105         if ( found != -1 )
106             printf("%d\n",found);
107         else
108             printf("Mozart\n");
109     }
110
111     return 0;
112 }
```

Listing 24.1.14: avarcolaci-90-andrei.cpp

```

1 #include <cstdio>
2 #include <cassert>
3 #include <cstring>
4 #include <algorithm>
5
6 using namespace std;
7
```

```

8  const int NMAX = 50000;
9  const int TMAX = 15;
10 const int EL_MAX = 1e9;
11 const int EL_JUM = 32762;
12
13 int v[2 * NMAX];
14 int cnt[EL_JUM + 1];
15
16 int main()
17 {
18     freopen("avarcolaci.in", "rt", stdin);
19     freopen("avarcolaci.out", "wt", stdout);
20
21     int nrTests;
22
23     scanf("%d", &nrTests);
24
25     for (int test = 1; test <= nrTests; test++)
26     {
27         int n;
28
29         int elMaj;
30         int nrCnt = 0;
31
32         scanf("%d", &n);
33
34         if (n <= 50000)
35         {
36             for (int i = 0; i < 2 * n; i++)
37             {
38                 scanf("%d", v + i);
39                 assert(0 <= v[i] && v[i] <= EL_MAX);
40             }
41
42             sort(v + 0, v + 2 * n);
43
44             elMaj = v[n];
45             for (int i = 0; i < 2 * n; i++)
46                 if (v[i] == elMaj)
47                     nrCnt++;
48
49         }
50         else
51         {
52             memset(cnt, 0, sizeof(cnt));
53
54             for (int i = 0; i < 2 * n; i++)
55             {
56                 int el;
57                 scanf("%d", &el);
58                 assert(0 <= el && el <= EL_JUM);
59                 cnt[el]++;
60             }
61
62             elMaj = 0;
63             int nrEl = 0;
64             for (int i = 0; i <= EL_JUM; i++)
65                 for (int j = 1; j <= cnt[i]; j++)
66                 {
67                     nrEl++;
68                     if (nrEl == n)
69                         elMaj = i;
70                 }
71
72             nrCnt = cnt[elMaj];
73         }
74
75         if (nrCnt > n)
76             printf("%d\n", elMaj);
77         else
78             printf("Mozart\n");
79     }
80
81     return 0;
82 }
```

Listing 24.1.15: avarcolaci-100-adrian-8bit.cpp

```

1 //Solutie Panaete Adrian
2 //Rupere in sechete de 8 biti
3 #include <cstdio>
4 #include <cstring>
5
6 #define N 2000005
7
8 using namespace std;
9
10 int t,n,maj,ans[15],cnt[4][256],sol_p1(int),sol_p2(int);
11
12 int main()
13 {
14     freopen("avarcolaci.in","r",stdin);
15     freopen("avarcolaci.out","w",stdout);
16
17     int T;
18     scanf("%d", &T);
19     for(int t = 0; t < T; t++)
20         sol_p1(t);
21
22     freopen("avarcolaci.in","r",stdin);
23     scanf("%d", &T);
24     for (int t = 0; t < T; t++)
25     {
26         int a = sol_p2(t);
27         a<0?printf("Mozart\n"):printf("%d\n",a);
28     }
29
30     return 0;
31 }
32
33 int sol_p1(int t)
34 {
35     int i,j,a;
36     memset(cnt,0,sizeof(cnt));
37     scanf("%d",&n);maj=n+1;n<=1;
38     for(i=1;i<=n;i++)
39     {
40         scanf("%d",&a);
41         for(j=0;j<4;j++)
42         {
43             cnt[j][a&255]++;
44             a>>=8;
45         }
46     }
47
48     for(i=3;i>=0;i--)
49     {
50         for(j=0;j<256;j++)
51             if(cnt[i][j]>=maj)
52                 break;
53             if(j==256) return -1;
54             a=(a<<8)|j;
55     }
56
57     ans[t] = a;
58     return a;
59 }
60
61 int sol_p2(int t)
62 {
63     int i,a,Cnt;
64     scanf("%d", &n);
65     maj=n+1;
66     n<=1;
67     for(Cnt=0,i=1;i<=n;i++)
68     {
69         scanf("%d", &a);
70         if(ans[t]==a) Cnt++;
71     }
72
73     return Cnt>=maj?ans[t]:-1;
74 }
```

Listing 24.1.16: avarcolaci-100-andrei.cpp

```

1  /* Andrei Parvu
2   ONI 2014
3 */
4 #include <cstdio>
5 #include <cassert>
6 #include <algorithm>
7
8 using namespace std;
9
10 const int NMAX = 500000;
11 const int TMAX = 15;
12 const int EL_MAX = 1e9;
13
14 int potential[TMAX];
15
16 int main()
17 {
18     freopen("avarcolaci.in", "rt", stdin);
19     freopen("avarcolaci.out", "wt", stdout);
20
21     int nrTests;
22     scanf("%d", &nrTests);
23     assert(1 <= nrTests && nrTests <= TMAX);
24
25     for (int test = 1; test <= nrTests; test++)
26     {
27         int n;
28
29         scanf("%d", &n);
30         assert(1 <= n && n <= NMAX);
31         int maj = -1, cnt = 0;
32
33         for (int i = 0; i < 2 * n; i++)
34         {
35             int el;
36             scanf("%d", &el);
37
38             assert(0 <= el && el <= EL_MAX);
39
40             if (maj == -1)
41             {
42                 maj = el;
43                 cnt = 1;
44             }
45             else
46             {
47                 if (el == maj)
48                     cnt++;
49                 else
50                 {
51                     cnt--;
52                     if (cnt == 0)
53                     {
54                         maj = el;
55                         cnt = 1;
56                     }
57                 }
58             }
59         }
60
61         potential[test] = maj;
62     }
63
64     fclose(stdin);
65     freopen("avarcolaci.in", "rt", stdin);
66
67     scanf("%d", &nrTests);
68
69     for (int test = 1; test <= nrTests; test++)
70     {
71         int n;
72         scanf("%d", &n);
73
74         int cnt = 0;

```

```

75     for (int i = 0; i < 2 * n; i++)
76     {
77         int x;
78         scanf("%d", &x);
79
80         if (x == potential[test])
81             cnt++;
82     }
83
84     if (cnt > n)
85     {
86         printf("%d\n", potential[test]);
87     }
88     else
89     {
90         // Problema e mai clasica ca Mozart
91         printf("Mozart\n");
92     }
93 }
94
95 return 0;
96 }
```

Listing 24.1.17: avarcolaci-100-bogdan.cpp

```

1 /*
2  * Olimpiada NaÈzionalÄC de InformaticÄC, PiteÈzti 2014
3  * Ål' 2014 Bogdan Cristian TÄtÄcroiu
4 */
5
6 #include <cstdio>
7 #include <cassert>
8
9 using namespace std;
10
11 const int MAXT = 15;
12 const int MAXN = 500000;
13 const int MAXV = 999999999;
14
15 int main()
16 {
17     assert(freopen("avarcolaci.in", "rt", stdin));
18 #ifndef DEBUG
19     assert(freopen("avarcolaci.out", "wt", stdout));
20 #endif
21
22     int T, N;
23     int maj[MAXT];
24
25     // We read the count of tests
26     assert(scanf("%d", &T) == 1);
27     assert(1 <= T && T <= MAXT);
28     for (int t = 0; t < T; t++)
29     {
30         // And then we do our bests
31         // We read the count of peasants
32         assert(scanf("%d", &N) == 1);
33         assert(1 <= N && N <= MAXN);
34
35         // *Don't* allocate memory like adolescents
36         int MAJ = -1, MAJCNT = 0;
37         for (int i = 0; i < 2 * N; i++)
38         {
39             int val;
40             assert(scanf("%d", &val) == 1);
41             assert(0 <= val && val <= MAXV);
42
43             if (val == MAJ)
44             {
45                 MAJCNT += 1;
46             }
47             else
48             {
49                 MAJCNT -= 1;
50                 if (MAJCNT <= 0)
```

```

51             {
52                 MAJ = val;
53                 MAJCNT = 1;
54             }
55         }
56     }
57
58     // I'm not a god damn poet
59     maj[t] = MAJ;
60 }
61
62
63     // And I'm sure that by now you know it
64     freopen("avarcolaci.in", "rt", stdin);
65     assert(scanf("%d", &T) == 1);
66     for (int t = 0; t < T; t++)
67     {
68         assert(scanf("%d", &N) == 1);
69
70         // But this is a pretty good start
71         int MAJCNT = 1;
72         MAJCNT = 0;
73         for (int i = 0; i < 2 * N; i++)
74         {
75             int val;
76             assert(scanf("%d", &val) == 1);
77             if (val == maj[t])
78                 MAJCNT += 1;
79         }
80
81         // For this task is more classic than Mozart
82         // ( Pentru că e mai clasică ca Moș Mozart )
83         if (MAJCNT >= N + 1)
84             printf("%d\n", maj[t]);
85         else
86             printf("Mozart\n");
87     }
88
89     return 0;
90 }
```

Listing 24.1.18: avarcolaci-100-bogdan-parsare.cpp

```

1 /*
2  * Olimpiada Națională de Informatică, Pitești 2014
3  * Al' 2014 Bogdan Cristian Tătăroiu
4  */
5
6 #include <iostream>
7 #include <assert>
8
9 using namespace std;
10
11 const int MAXT = 15;
12 const int MAXN = 500000;
13 const int MAXV = 999999999;
14
15 const int buff_size = 500000;
16 char buff[buff_size + 5]; int ch;
17
18 inline int read_int()
19 {
20     /* Al' 2014 Mihai Popa */
21     while ( !(buff[ch] >= '0' && buff[ch] <= '9') )
22     {
23         ++ch;
24         if ( ch == buff_size )
25         {
26             fread(buff,buff_size,1,stdin);
27             ch = 0;
28         }
29     }
30
31     int r = 0;
32     while ( buff[ch] >= '0' && buff[ch] <= '9' )
```

```

33     {
34         r = r * 10 + buff[ch]-'0';
35         ++ch;
36         if ( ch == buff_size )
37         {
38             fread(buff,buff_size,1,stdin);
39             ch = 0;
40         }
41     }
42
43     return r;
44 }
45
46 int main()
47 {
48     assert(freopen("avarcolaci.in", "rt", stdin));
49 #ifndef DEBUG
50     assert(freopen("avarcolaci.out", "wt", stdout));
51 #endif
52
53     int T, N;
54     int maj[MAXT];
55
56     // We read the count of tests
57     fread(buff, buff_size, 1, stdin);
58     ch = 0;
59
60     T = read_int();
61     assert(1 <= T && T <= MAXT);
62     for (int t = 0; t < T; t++)
63     {
64         // And then we do our bests
65         // We read the count of peasants
66         N = read_int();
67         assert(1 <= N && N <= MAXN);
68
69         // *Don't* allocate memory like adolescents
70         int MAJ = -1, MAJCNT = 0;
71         for (int i = 0; i < 2 * N; i++)
72         {
73             int val;
74             val = read_int();
75             assert(0 <= val && val <= MAXV);
76
77             if (val == MAJ)
78                 MAJCNT += 1;
79             else
80             {
81                 MAJCNT -= 1;
82                 if (MAJCNT <= 0)
83                 {
84                     MAJ = val;
85                     MAJCNT = 1;
86                 }
87             }
88         }
89
90         // I'm not a god damn poet
91         maj[t] = MAJ;
92     }
93
94
95     // And I'm sure that by now you know it
96     freopen("avarcolaci.in", "rt", stdin);
97     fread(buff, buff_size, 1, stdin);
98     ch = 0;
99
100    T = read_int();
101    for (int t = 0; t < T; t++)
102    {
103        N = read_int();
104
105        // But this is a pretty good start
106        int MAJCNT = 1;
107        MAJCNT = 0;
108        for (int i = 0; i < 2 * N; i++)

```

```

109      {
110          int val;
111          val = read_int();
112          if (val == maj[t])
113              MAJCNT += 1;
114      }
115
116      // For this task is more classic than Mozart
117      // ( Pentru că e mai clasică ca Moș Mozart )
118      if (MAJCNT >= N + 1)
119          printf("%d\n", maj[t]);
120      else
121          printf("Mozart\n");
122  }
123
124  return 0;
125 }
```

24.1.3 Rezolvare detaliată

Ideea de a închide și a redeschide fișierul de intrare ... era "ascunsă" și "arătată" în cele 5 secunde (timp mare!) din ... **"Timp maxim de executare/test: 5.0 secunde"**

24.2 karb

Problema 2 - karb

100 de puncte

În perioada Campionatului Mondial din Brazilia se preconizează o creștere a traficului de cafea. Se știe că sunt N orașe, conectate prin $N - 1$ străzi bidirectionale, astfel încât se poate ajunge din orice oraș în altul. În prezent există K carteluri de cafea aflate în orașe distințe, care își exercită influența în propriul oraș. Se știe că fiecare din aceste carteluri dorește să-și extindă influența în orașele vecine. Astfel, la un moment de timp, un cartel poate să-și extindă influența într-un oraș vecin doar dacă acesta nu se află sub influență altui cartel. O dată ce un cartel își extinde influența asupra unui nou oraș, cartelul își poate extinde influența și în orașele vecine acestuia. Se știe că până la începerea campionatului mondial, fiecare oraș va fi sub influența unui cartel.

ABIN (Agência Brasileira de Inteligência) dorește să afle în câte moduri poate fi dominată țara de influențele celor K carteluri la data începerii campionatului mondial, modulo 666013.

Cerințe

Cunoscând numărul de orașe N , modul în care acestea sunt conectate, numărul de carteluri inițiale K și cele K orașe în care se află cartelurile, să se determine numărul de moduri în care țara poate fi împărțită între cartelurile de cafea, modulo 666013.

Date de intrare

Fișierul de intrare **karb.in** conține pe prima linie 2 numere naturale N și K , reprezentând numărul de orașe, respectiv numărul cartelurilor existente inițial. Pe a doua linie din fișier se vor afla K numere, reprezentând orașele în care se află cele K carteluri. Pe următoarele $N - 1$ linii se vor afla câte două numere naturale, reprezentând o legătură între cele două orașe.

Date de ieșire

Pe prima linie a fișierului de ieșire **karb.out** se va scrie un singur număr natural reprezentând numărul de moduri modulo 666013.

Restricții și precizări

- $1 \leq K \leq N \leq 100\ 000$
- Pentru teste în valoare de 10% din punctaj se garantează că $k \leq n \leq 7$, iar pentru alte 20% din teste se garantează că $k = 2$.
- Două orașe sunt vecine dacă există o stradă bidirectională între ele.

Exemplu:

karb.in	karb.out	Explicații
6 3	5	Cele 5 moduri posibile:
3 4 5		1. (3) (1, 2, 5) (4, 6)
1 2		2. (3, 1) (2, 5) (4, 6)
1 3		3. (3, 1, 2) (5) (4, 6)
2 4		4. (3, 1) (5) (2, 4, 6)
2 5		5. (3) (5) (1, 2, 4, 6)
4 6		

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

Memorie: total **32 MB** din care pentru stivă **4 MB**

Dimensiune maximă a sursei: **25 KB**

24.2.1 Indicații de rezolvare

Andrei Ciocan, Student Universitatea Politehnica, București

Problema se poate rezolva prin intermediul *programării dinamice*.

Vom încerca să pornim de jos, începând cu subarborei cât mai mici și apoi să urcăm în arbore. În încercarea colorării unui subarbore, observăm că apar 2 cazuri:

- rădăcina subarborelui este influențată de un nod care se află în subarbore, sau

- influențat de un nod care nu aparține subarborelui (apare în sus).

Astfel vom obține o dinamică:

- din[nod][0] = numărul de noduri astfel încât nod este influențat de un cartel din subarbore

- din[nod][1] = numărul de noduri astfel încât nod este influențat de un cartel care nu aparține subarborelui

Recurența se poate obține în felul următor:

- $\text{din[nod][1]} = \prod (\text{din[fiu][0]} + \text{din[fiu][1]})$

- $\text{din[nod][0]} = \sum_i \text{din[i][0]} * \text{prod}(\text{din[fiu][0]} + \text{din[fiu][1]}),$

unde i este fiu al lui nod; fiu este de asemenea copil al lui nod, dar diferit de i .

Ideea este că alegem ca nod sa fie influențat de același cartel care influențează și fiul i (ceilași copii ai nodului nod putând fi influențați în orice mod).

Se pot diferenția și cazuri speciale, atunci când un nod este cartel, $\text{din[nod][1]} = 0$ (orice nod din subarbore nu poate fi influențat de un nod din afara subarborelui, întrucât nu au acces).

În acest caz, când nodul este cartel, subarborei fii se pot colora în orice fel, deci

$\text{din[nod][0]} = \prod (\text{din[fiu][0]} + \text{din[fiu][1]}).$

În final, rezultatul se va afla în din[1][0] .

Pentru această abordare, se obțin în jur de 70 de puncte.

Pentru a obține punctajul maxim, trebuie optimizată formula de calcul al numărului de configurații când nodul rădăcina este influențat de un nod din subarbore.

Una din soluții ar fi să precalculez produsul tuturor fililor unui nod,

$\text{produsPrecalculat} = \prod (\text{din[fiu][0]} + \text{din[fiu][1]}),$

iar la calcularea sumei înmulțim cu inversul modular al factorului respectiv lui i :

$(\text{din}[i][0] + \text{din}[i][1]),$ unde i este în formula de mai sus.

Vom avea astfel:

$\text{din[nod][0]} = \sum \text{produsPrecalculat} * \text{inversModular}(\text{din}[i][0] + \text{din}[i][1]).$

Trebuie avut grijă când $\text{din}[i][0] + \text{din}[i][1]$ este 0, întrucât nu putem împărții la el.

Altă abordare ar fi prin programare dinamică:

$\text{left}[i] = \prod (\text{din[fiu][0]} + \text{din[fiu][1]}),$

unde fiu este un fiu de la stânga lui i , și

$\text{right}[i] = \prod (\text{din[fiu][0]} + \text{din[fiu][1]}),$

cu fiu la dreapta lui i .

Nu vom avea decât să adăugăm la suma $\text{left}[i] * \text{din}[i][0] * \text{right}[i].$

24.2.2 Cod sursă

Listing 24.2.1: karb_.brut-mihai.cpp

```

1 //Mihai Popa
2 //backtracking, 10p
3 #include<stdio.h>
4 #include<vector>
5 #include<algorithm>
6 #include<queue>
7 #include<cassert>
8
9 #define maxdim 200005
10#define mod 666013
11
12 using namespace std;
13
14 int n,k,sol;
15 int special[maxdim],colorat[maxdim],T[maxdim];
16 vector<int>G[maxdim],speciale,gives[maxdim];
17
18 inline int check ()
19 {
20
21     for ( int i = 1 ; i <= n ; ++i )
22         gives[i].clear();
23
24     for ( int i = 1 ; i <= n ; ++i )
25         gives[T[i]].push_back(i);
26
27     queue<int>Q; vector<int>c(n+1,0);
28
29     for ( int i = 0 ; i < (int)speciale.size() ; ++i )
30     {
31         Q.push(speciale[i]);
32         c[speciale[i]] = speciale[i];
33     }
34
35     while ( !Q.empty() )
36     {
37         int nod = Q.front(); Q.pop();
38         for ( vector<int>::iterator itt = gives[nod].begin() ;
39               itt != gives[nod].end() ; ++itt )
40         {
41             int vecin = ( *itt );
42             if ( c[vecin] ) return 0;
43             c[vecin] = c[nod];
44             Q.push(vecin);
45         }
46     }
47
48     for ( int i = 1 ; i <= n ; ++i )
49     if ( !c[i] )
50         return 0;
51
52     return 1;
53 }
54
55 void back ( int nod )
56 {
57     if ( nod == n+1 )
58     {
59         sol += check();
60         return ;
61     }
62
63     if ( special[nod] )
64     {
65         back(nod+1);
66         return ;
67     }
68
69     for ( vector<int>::iterator itt = G[nod].begin() ;
70           itt != G[nod].end() ; ++itt ){
71         int vecin = ( *itt );
72
73         if ( T[vecin] != nod )
74         {

```

```

75         T[nod] = vecin;
76         back(nod+1);
77         T[nod] = 0;
78     }
79 }
80 }
81
82 int main ()
83 {
84     freopen("karb.in", "r", stdin);
85     freopen("karb.out", "w", stdout);
86
87     scanf("%d %d", &n, &k);
88     assert(n >= 1 && n <= 200000);
89     assert(k >= 1 && k <= n);
90     int x,y;
91     for ( int i = 1 ; i <= k ; ++i )
92     {
93         scanf("%d", &x);
94         assert(!special[x]);
95         assert(x >= 1 && x <= n);
96         special[x] = x;
97         speciale.push_back(x);
98     }
99
100    for ( int i = 1 ; i < n ; ++i )
101    {
102        scanf("%d %d", &x, &y);
103        assert(x >= 1 && x <= n);
104        assert(y >= 1 && y <= n);
105        assert(x != y);
106        G[x].push_back(y);
107        G[y].push_back(x);
108    }
109
110    back(1);
111
112    printf("%d\n", sol);
113
114    return 0;
115 }
```

Listing 24.2.2: karb_cartita.cpp

```

1 #include <cstdio>
2 #include <vector>
3
4 using namespace std;
5
6 #define maxn 100010
7 #define mod 666013
8
9 int n, k, a, b, m;
10 int d[maxn][2], f[maxn], sp[maxn], left[maxn], right[maxn];
11 char isPar[maxn];
12 vector<int> v[maxn];
13
14 void df(int nod)
15 {
16     if(f[nod])
17         return;
18
19     f[nod]=1;
20
21     int tata=0, nFii=v[nod].size(), fiu;
22
23     for(int i=0; i<nFii; ++i)
24     {
25         fiu=v[nod][i];
26         if(f[fiu]==1)
27             tata=fiu;
28         else
29             df(fiu);
30     }
31 }
```

```

32     for(int i=0; i<nFii; ++i)
33     {
34         if(v[nod][i]==tata)
35             isPar[i]=1;
36         else
37             isPar[i]=0;
38     }
39
40     left[0]=right[nFii+1]=1;
41
42     for(int i=0; i<nFii; ++i)
43     {
44         if(isPar[i])
45         {
46             left[i+1]=right[i+1]=1;
47             left[i+1]=(1LL*left[i+1]*left[i])%mod;
48             continue;
49         }
50
51         fiu=v[nod][i];
52
53         left[i+1]=right[i+1]=(d[fiu][0]+d[fiu][1]);
54         left[i+1]=(1LL*left[i+1]*left[i])%mod;
55     }
56
57
58     if(sp[nod])
59     {
60         d[nod][0]=left[nFii];
61         return;
62     }
63
64     d[nod][1]=left[nFii];
65
66     for(int i=nFii; i>0; --i)
67         right[i]=(1LL*right[i+1]*right[i])%mod;
68
69     for(int i=1; i<=nFii; ++i)
70     {
71         fiu=v[nod][i-1];
72
73         if(isPar[i-1])
74             continue;
75
76         d[nod][0]=(d[nod][0]+((1LL*d[fiu][0]*left[i-1])%mod)*right[i+1]))%mod;
77     }
78 }
79
80 int main()
81 {
82     freopen("karb.in", "r", stdin);
83     freopen("karb.out", "w", stdout);
84
85     scanf("%d%d", &n, &k);
86     for(int i=1; i<=k; ++i)
87     {
88         scanf("%d", &a);
89         sp[a]=1;
90     }
91
92     for(int i=1; i<n; ++i)
93     {
94         scanf("%d%d", &a, &b);
95         v[a].push_back(b);
96         v[b].push_back(a);
97     }
98
99     df(1);
100
101    printf("%d\n", d[1][0]);
102
103    return 0;
104 }
```

Listing 24.2.3: karb_npatrat-mihai.cpp

```

1 //Mihai Popa
2 //n^2, 60p
3 #include<stdio.h>
4 #include<vector>
5 #include<algorithm>
6
7 #define maxdim 5005
8 #define mod 666013
9
10 using namespace std;
11
12 int n,k;
13 int special[maxdim],viz[maxdim],T[maxdim],
14     D[maxdim][maxdim],localD[maxdim][maxdim],toate[maxdim];
15 vector<int>G[maxdim];
16
17 void dfs ( int nod )
18 {
19     viz[nod] = 1;
20
21     int sons = 0;
22     for ( vector<int>::iterator itt = G[nod].begin() ;
23             itt != G[nod].end() ; ++itt )
24     {
25         int fiu = *itt;
26         if ( viz[fiu] ) continue ;
27
28         ++sons; T[fiu] = nod;
29         dfs(fiu);
30     }
31
32     if ( !sons )
33     {
34         if ( special[nod] )
35         {
36             D[nod][nod] = 1;
37             toate[nod] = 1;
38         }
39         else
40             D[nod][0] = 1; toate[nod] = 1;
41
42         return ;
43     }
44
45     localD[0][0] = 1; int ind = 0;
46     for ( vector<int>::iterator itt = G[nod].begin() ;
47             itt != G[nod].end() ; ++itt )
48     {
49         int fiu = *itt;
50         if ( T[fiu] != nod ) continue ;
51
52         ++ind;
53         for ( int j = 1 ; j <= n ; ++j )
54         {
55             localD[ind][j] = (1LL*localD[ind-1][j]*toate[fiu] +
56                                 1LL*localD[ind-1][0]*D[fiu][j])%mod;
57         }
58
59         localD[ind][0] = (1LL*localD[ind-1][0]*toate[fiu])%mod;
60     }
61
62     if ( !special[nod] )
63     {
64         for ( int j = 0 ; j <= n ; ++j )
65         {
66             D[nod][j] = localD[ind][j];
67             toate[nod] += D[nod][j];
68             if ( toate[nod] >= mod )
69                 toate[nod] -= mod;
70         }
71     }
72     else
73     {
74         D[nod][nod] = localD[ind][0];
75         toate[nod] = D[nod][nod];

```

```

76      }
77  }
78
79  int main ()
80  {
81      freopen("karb.in", "r", stdin);
82      freopen("karb.out", "w", stdout);
83
84      scanf("%d %d", &n, &k);
85      int x,y;
86      for ( int i = 1 ; i <= k ; ++i )
87      {
88          scanf("%d", &x);
89          special[x] = 1;
90      }
91
92      for ( int i = 1 ; i < n ; ++i )
93      {
94          scanf("%d %d", &x, &y);
95          G[x].push_back(y);
96          G[y].push_back(x);
97      }
98
99      dfs(1);
100
101     int sol = toate[1]-D[1][0];
102     if ( sol < 0 )
103         sol += mod;
104
105     printf("%d\n",sol);
106
107     return 0;
108 }
```

Listing 24.2.4: karb-100p-mihai.cpp

```

1 //Mihai Popa
2 //O(n) - 100p
3 #include<stdio.h>
4 #include<vector>
5 #include<algorithm>
6 #include <cassert>
7
8 #define maxdim 200005
9 #define mod 666013
10
11 using namespace std;
12
13 int n,k;
14 int special[maxdim],viz[maxdim],T[maxdim],D[maxdim][2];
15 vector<int>G[maxdim];
16
17 void dfs ( int nod )
18 {
19     viz[nod] = 1;
20
21     int sons = 0;
22     for ( vector<int>::iterator itt = G[nod].begin() ;
23             itt != G[nod].end() ; ++itt )
24     {
25         int fiu = *itt;
26         if ( viz[fiu] ) continue ;
27
28         ++sons; T[fiu] = nod;
29         dfs(fiu);
30     }
31
32     if ( !sons )
33     {
34         if ( special[nod] )
35             D[nod][0] = 0,D[nod][1] = 1;
36         else
37             D[nod][0] = 1,D[nod][1] = 0;
38
39     return ;

```

```

40     }
41
42     vector<int>localD[2];
43     localD[0].resize(sons+1);
44     localD[1].resize(sons+1);
45
46     localD[0][0] = 1;
47     int ind = 0;
48     for ( vector<int>::iterator itt = G[nod].begin() ;
49           itt != G[nod].end() ; ++itt )
50     {
51         int fiu = *itt;
52         if ( T[fiu] != nod )    continue ;
53
54         ++ind;
55         localD[0][ind] = (1LL*localD[0][ind-1]*
56                             (D[fiu][0]+D[fiu][1]))%mod;
57         localD[1][ind] = (1LL*localD[0][ind-1]*D[fiu][1] +
58                             1LL*localD[1][ind-1]*(D[fiu][0]+D[fiu][1]))%mod;
59     }
60
61     if ( !special[nod] )
62     {
63         D[nod][0] = localD[0][ind];
64         D[nod][1] = localD[1][ind];
65     }
66     else
67     {
68         D[nod][0] = 0;
69         D[nod][1] = localD[0][ind];
70     }
71 }
72
73 int main () {
74
75     freopen("karb.in", "r", stdin);
76     freopen("karb.out", "w", stdout);
77
78     scanf("%d %d", &n, &k);
79     assert(n >= 1 && n <= 200000);
80     assert(k >= 1 && k <= n);
81     int x,y;
82     for ( int i = 1 ; i <= k ; ++i )
83     {
84         scanf("%d", &x);
85         assert(!special[x]);
86         assert(x >= 1 && x <= n);
87         special[x] = x;
88     }
89
90     for ( int i = 1 ; i < n ; ++i )
91     {
92         scanf("%d %d", &x, &y);
93         assert(x >= 1 && x <= n);
94         assert(y >= 1 && y <= n);
95         assert(x != y);
96         G[x].push_back(y);
97         G[y].push_back(x);
98     }
99
100    dfs(1);
101
102    printf("%d\n", D[1][1]);
103
104    return 0;
105 }
```

Listing 24.2.5: karb-ciocan.cpp

```

1 #include<iostream>
2 #include<fstream>
3 #include <vector>
4
5 using namespace std;
6
```

```

7 #define MOD 666013
8
9 vector<int> v[100005];
10
11 int n, k, viz[100005], father[100005], dint[100005],
12      dinf[100005], cartel[100005];
13
14 void afisare()
15 {
16     ofstream g("karb.out");
17     g << dinf[1] << endl;
18     g.close();
19 }
20
21 void citire()
22 {
23     ifstream f("karb.in");
24     f >> n >> k ;
25     for ( int i = 0 ;i < k ;i++)
26     {
27         int x;
28         f >> x;
29         cartel[x] = 1;
30     }
31
32     for (int i = 1 ; i< n ;i++)
33     {
34         int x, y;
35         f >> x >> y ;
36         v[x].push_back(y);
37         v[y].push_back(x);
38     }
39
40     f.close();
41 }
42
43 void solve(int nod)
44 {
45     vector<int> :: iterator it;
46     viz[nod] = 1;
47
48     for( it = v[nod].begin(); it < v[nod].end() ;it++)
49     {
50         if (viz[*it] == 0)
51         {
52             father[*it] = nod;
53             solve( *it);
54         }
55     }
56
57     if (v[nod].size() == 1 && nod != 1)
58     {
59         if (cartel[nod])
60         {
61             dint[nod] = 1;
62             dint[nod] = 0;
63         }
64         else
65         {
66             dint[nod] = 0;
67             dint[nod] = 1;
68         }
69         return ;
70     }
71
72     if (cartel[nod] == 1)
73     {
74         dint[nod] = 0;
75         long long p = 1;
76         for (it = v[nod].begin(); it < v[nod].end(); it++)
77             if ( *it != father[nod])
78             {
79                 p = (p * (dinf[*it] + dint[*it])) % MOD ;
80             }
81
82     dint[nod] = p ;

```

```

83     }
84     else
85     {
86         vector<int> :: iterator it2;
87         long long p = 1;
88
89         for (it = v[nod].begin(); it < v[nod].end(); it++)
90             if (*it != father[nod])
91                 p = (p * (dinf[*it] + dint[*it])) % MOD ;
92
93         dint[nod] = p ;
94
95         for (it = v[nod].begin(); it < v[nod].end() ;it++)
96         {
97             if (*it != father[nod])
98             {
99                 p = dint[*it];
100                for (it2= v[nod].begin(); it2 < v[nod].end() ;it2++)
101                    if (*it != *it2 && *it2 != father[nod])
102                    {
103                        p = (p * (dinf[*it2] + dint[*it2])) % MOD ;
104                    }
105
106                dint[nod] = (dinf[nod] + p) % MOD ;
107            }
108        }
109    }
110 }
111
112 int main()
113 {
114     citire();
115     solve(1);
116     afisare();
117 }
```

24.2.3 *Rezolvare detaliată

24.3 volum

Problema 3 - volum

100 de puncte

K.L. 2.0 și-a dorit o piscină pe un grid A cu N linii și M coloane. Cum K.L. 2.0 nu a fost foarte inspirat, el a uitat să își niveleze terenul înainte de a construi piscina, astfel încât fiecare celulă de coordonate (i, j) a gridului are o înălțime $A_{i,j}$ ($1 \leq i \leq N$ și $1 \leq j \leq M$). La un moment dat începe o ploaie puternică, care umple piscina cu apă. După terminarea ploii, K.L. 2.0 se întreabă câtă apă are în piscină.

Dintr-o celulă apă se varsă în celulele vecine cu care are o latură comună și care au înălțimea strict mai mică decât celula curentă. Apa de pe marginea piscinei se scurge în exterior.

Cerințe

Pentru N , M și gridul A date, să se determine volumul de apă care a rămas în piscină.

Date de intrare

Fișierul de intrare **volum.in** conține pe prima linie două numere naturale N și M , reprezentând dimensiunile gridului, iar pe fiecare dintre următoarele N linii se află câte M numere, reprezentând înălțimile terenului, separate prin câte un spatiu.

Date de ieșire

Fișierul de ieșire **volum.out** conține un singur număr, reprezentând volumul de apă care a rămas în piscină.

Restricții și precizări

- $3 \leq N, M \leq 752$
- $0 \leq A_{i,j} \leq 10^9 + 2$
- Pentru 30% din punctaj, $3 \leq N, M \leq 82$.
- Pentru 40% din punctaj, $0 \leq A_{i,j} \leq 10^3 + 2$.
- Volumul apei este suma unităților de apă care rămâne în celulele piscinei.

Exemple:

volum.in	volum.out	Explicații
3 3 2 2 2 2 0 2 2 2 2	2	După ploaie rămân două unități de apă în celula cu înălțimea 0. Nu pot rămâne 3 unități, deoarece o unitate s-ar scurge prin una din cele 4 celule vecine în exteriorul piscinei.
3 3 3 3 3 3 0 2 3 3 3	2	După ploaie rămân două unități de apă în celula cu înălțimea 0. Nu pot rămâne 3 unități, deoarece o unitate s-ar scurge prin celula vecină cu valoarea 2 în exteriorul piscinei.
5 5 2 2 3 3 3 2 2 3 1 3 2 3 1 3 3 2 2 3 2 2 2 2 2 2 2	4	După ploaie rămân câte două unități de apă în celulele cu înălțimea 1. Nu pot rămâne câte 3 unități. De exemplu, din celula (2,4) apa se poate scurge în celula (2,5) și apoi în exterior, respectiv din celula (3,3) în sirul de celule (4,3) - (5,3) și apoi în exterior.

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

Memorie: total **32 MB** din care pentru stivă **8 MB**

Dimensiune maximă a sursei: **25 KB**

24.3.1 Indicații de rezolvare

stud. Radu Voroneanu - University of Cambridge

Soluția 1: Radu Voroneanu - $O(N * M * \log(N * M))$ - 100 p

Vom porni de la ideea că singurele celule prin care poate să se scurgă apa sunt cele de pe marginea matricei.

Intuitiv, apa se va scurge mai repede prin marginile mai mici.

Vom ține o *listă de priorități* (simulată cu ajutorul unei structuri de date la alegere - *heap, priority_queue, set*) în care vom insera inițial toate celulele din marginea matricei.

La fiecare pas extragem minimul din această listă și ne extindem cu ajutorul unei *cozi* în toate căsuțele accesibile cu valori mai mici sau egale decât celula inițială.

Toate valorile atinse se vor umple cu apă doar până la valoarea celulei inițiale, orice unitate suplimentară scurgându-se spre exterior prin acea celulă. Eliminăm acea secțiune din matrice. Prin această eliminare, se vor crea margini noi care trebuie să ele *inserate* în lista de priorități. Procesul se repetă până ce lista de priorități devine goală (cu alte cuvinte vor fi parcuse toate elementele matricei).

Soluția 2: Vlad Gavrilă - 80-100 p

Vom începe prin a *sorta* celulele crescător după înălțime. Pentru a continua, ne vom folosi de structura de *păduri de mulțimi disjuncte*, la care vom adăuga trei parametri:

size - dimensiunea componentei,

maxHeight - înălțimea maximă a unei celule din componentă și un boolean

ext - True dacă componentă noastră este accesibilă din exteriorul piscinei, False în caz contrar.

Pe rând, vom lua fiecare celulă în ordinea sortării și îi vom crea o componentă de dimensiune 1, înălțime maximă crHeight a celulei, setând și parametrul ext în mod corespunzător (True pentru interior, False pentru margine).

Vom uni această componentă cu toate componentele vecine care au fost deja create (cele care conțin înălțimi mai mici decât cea curentă).

Observăm că, atunci când componentă pe care o unim la cea curentă nu este accesibila din exterior și are înălțime maximă maxH, putem adăuga un nivel de apă de înălțime crHeight - maxH peste toate celulele acelei componente, fără ca apa să se verse.

Acest lucru este posibil întrucât componenta, nefind accesibilă din exterior, este mărginită doar de celule cu înălțime mai mare decât crHeight a celulei curente. În acest caz, adăugăm la soluție volumul $(crHeight - maxH) * sizeAdd$, unde sizeAdd este dimensiunea componentei pe care o adăugăm.

În cazul în care componenta este accesibilă din exterior, toată apa pe care încercam să o adăugăm se va scurge în exterior, deci nu putem adăuga nimic la soluția noastră.

Complexitatea acestei soluții este $O(N * M * \log(N * M))$ ca timp, din cauza sortării, și $O(N * M)$ ca memorie.

Solutia 3: Adrian Panaete - 60 p

La citire se calculeaza înălțimea maximă a unei celule.

Se "umple" fiecare celulă cu apă până la înălțimea maximă ținând înălțimea apei adăugate pe o altă matrice.

Acum se simuleaza vărsarea apei din celulele care au un surplus de apă.

Pentru aceasta - se parcurg în mod repetat cele $n * m$ poziții și se verifică doar cele în care mai există apă.

Se alege (dacă există) celula vecina cu cea mai mare înălțime totală (înălțime apă + înălțime celula), dar mai mică decât celula curentă.

Se elimină surplusul de apă astfel încât fie nu mai rămâne apă în celulă, fie s-a coborât nivelul apei la înălțimea totală a celulei vecine aleasă.

Dacă la o parcurgere nu se elimină apă înseamnă că s-a vărsat tot surplusul de apă și se calculează soluția ca suma valorilor din matricea care reține înălțimea apei.

24.3.2 Cod sursă

Listing 24.3.1: volum1.cpp

```

1 //Vlad Gavrila - O(N*M*log(N))
2 #include <iostream>
3 #include <algorithm>
4 #include <set>
5 #include <vector>
6
7 using namespace std;
8
9 #define maxn 755
10
11 int n, m;
12 int v[maxn][maxn];
13 char f[maxn][maxn];
14 long long sol;
15 vector<pair<int, pair<int, int>> hp;
16 const int d1[]={0, 0, -1, 1};
17 const int d2[]={1, -1, 0, 0};
18
19 void addElement(int x, int y, int h)
20 {
21     if(x<=0 || y<=0 || x>n || y>m)
22         return;
23     if(f[x][y]==1)
24         return;
25
26     //calcul solutie
27     if(h<v[x][y])
28         h=v[x][y];
29     sol=sol+h-v[x][y];
30
31     //add la heap
32     hp.push_back(make_pair(-h, make_pair(x, y)));
33     push_heap(hp.begin(), hp.end());
34     f[x][y]=1;
35 }
36
37 int main()
38 {
39     freopen("volum.in", "r", stdin);
40     freopen("volum.out", "w", stdout);
41 }
```

```

42     scanf("%d%d", &n, &m);
43     for(int i=1; i<=n; ++i)
44         for(int j=1; j<=m; ++j)
45             scanf("%d", &v[i][j]);
46
47     for(int i=1; i<=n; ++i)
48     {
49         addElement(i, 1, v[i][1]);
50         addElement(i, m, v[i][m]);
51     }
52
53     for(int j=1; j<=m; ++j)
54     {
55         addElement(1, j, v[1][j]);
56         addElement(n, j, v[n][j]);
57     }
58
59     int xc, yc, hc;
60
61     while(hp.size()>0)
62     {
63         hc=-hp[0].first;
64         xc=hp[0].second.first;
65         yc=hp[0].second.second;
66
67         pop_heap(hp.begin(), hp.end());
68         hp.pop_back();
69
70         for(int i=0; i<4; ++i)
71             addElement(xc+d1[i], yc+d2[i], hc);
72     }
73
74     printf("%lld\n", sol);
75
76     return 0;
77 }
```

Listing 24.3.2: volum2.cpp

```

1 #include <stdio.h>
2 #include <algorithm>
3 #include <queue>
4
5 using namespace std;
6
7 #define MAXN 755
8
9 int A[MAXN][MAXN];
10 bool Sel[MAXN][MAXN], In_q[MAXN][MAXN];
11
12 priority_queue<pair<int, pair<int, int>> Q;
13 pair<int, int> Q2[MAXN * MAXN];
14
15 int N, M, H, Nr;
16 long long Ans;
17 int x, y, xn, yn;
18
19 int dirx[4] = {-1, 1, 0, 0};
20 int diry[4] = {0, 0, -1, 1};
21
22 int main()
23 {
24     freopen("volum.in", "r", stdin);
25     freopen("volum.out", "w", stdout);
26
27     scanf("%d %d", &N, &M);
28     for (int i = 1; i <= N; ++i)
29         for (int j = 1; j <= M; ++j)
30             scanf("%d", &A[i][j]);
31
32     for (int i = 0; i <= M+1; ++i)
33         Sel[0][i] = Sel[N+1][i] = true;
34
35     for (int i = 0; i <= N+1; ++i)
36         Sel[i][0] = Sel[i][M+1] = true;
```

```

37
38     for (int i = 1; i <= N; ++i)
39     {
40         Q.push(make_pair(-A[i][1], make_pair(i, 1)));
41         Q.push(make_pair(-A[i][M], make_pair(i, M)));
42         In_q[i][1] = In_q[i][M] = true;
43     }
44
45     for (int i = 2; i < M; ++i)
46     {
47         Q.push(make_pair(-A[1][i], make_pair(1, i)));
48         Q.push(make_pair(-A[N][i], make_pair(N, i)));
49         In_q[1][i] = In_q[N][i] = true;
50     }
51
52     while (Q.size())
53     {
54         Nr = 1;
55         Q2[Nr] = (Q.top()).second;
56         H = -(Q.top()).first;
57         Sel[Q2[1].first][Q2[1].second] = true;
58
59         for (int i = 1; i <= Nr; ++i)
60         {
61             x = Q2[i].first;
62             y = Q2[i].second;
63             for (int j = 0; j < 4; ++j)
64             {
65                 xn = x + dirx[j];
66                 yn = y + diry[j];
67                 if (Sel[xn][yn]) continue;
68                 if (A[xn][yn] <= H)
69                 {
70                     Ans += (long long) H - A[xn][yn];
71                     Q2[++Nr] = make_pair(xn, yn);
72                     Sel[xn][yn] = true;
73                 }
74                 else
75                 {
76                     if (In_q[xn][yn]) continue;
77                     Q.push(make_pair(-A[xn][yn], make_pair(xn, yn)));
78                     In_q[xn][yn] = true;
79                 }
80             }
81         }
82         Q.pop();
83     }
84
85     printf("%lld\n", Ans);
86     return 0;
87 }
```

Listing 24.3.3: volum3.cpp

```

1 //Vlad Gavrila - O(N*M*log(N)) - paduri de multimi disjuncte
2 #include <cstdio>
3 #include <algorithm>
4 #include <set>
5 #include <vector>
6
7 using namespace std;
8
9 #define maxn 755
10
11 int n, m, nod;
12 int v[maxn][maxn], t[maxn*maxn], ext[maxn*maxn], h[maxn*maxn], sz[maxn*maxn];
13 long long sol;
14 vector<pair<int, pair<int, int>> hp;
15 const int d1[]={0, 0, -1, 1};
16 const int d2[]={1, -1, 0, 0};
17
18 int tata(int nod)
19 {
20     if(t[nod]==nod)
21         return nod;
```

```

22
23     int aux=tata(t[nod]);
24     t[nod]=aux;
25     return aux;
26 }
27
28 void uniune(int nod1, int nod2)
29 {
30     int t1=tata(nod1);
31     int t2=tata(nod2);
32
33     if(t1==t2 || h[t1]<h[t2])
34         return;
35
36     t[t2]=t1;
37     sol=sol+1LL*sz[t2]*(h[t1]-h[t2])*ext[t2];
38
39     sz[t1]+=sz[t2];
40     ext[t1]*=ext[t2];
41 }
42
43 int main()
44 {
45     freopen("volum.in", "r", stdin);
46     freopen("volum.out", "w", stdout);
47
48     scanf("%d%d", &n, &m);
49     for(int i=1; i<=n; ++i)
50         for(int j=1; j<=m; ++j)
51         {
52             scanf("%d", &v[i][j]);
53
54             nod=(i-1)*m+j;
55
56             t[nod]=nod;
57             h[nod]=v[i][j];
58             sz[nod]=1;
59             if(i>1 && i<n && j>1 && j<m)
60                 ext[nod]=1;
61
62             hp.push_back(make_pair(v[i][j], make_pair(i, j)));
63         }
64
65     sort(hp.begin(), hp.end());
66
67     int xc=0, yc=0, xn, yn, n1, n2;
68
69     for(int i=0; i<n*m; ++i)
70     {
71         xc=hp[i].second.first;
72         yc=hp[i].second.second;
73
74         n1=(xc-1)*m+yc;
75
76         for(int j=0; j<4; ++j)
77         {
78             xn=xc+d1[j];
79             yn=yc+d2[j];
80
81             if(xn<=0 || yn<=0 || xn>n || yn>m)
82                 continue;
83
84             n2=(xn-1)*m+yn;
85
86             uniune(n1, n2);
87         }
88     }
89
90     printf("%lld\n", sol);
91
92     return 0;
93 }
```

Listing 24.3.4: volum4.cpp

```

1 //Panaete Adrian
2 //Neoptim -- golire de la Hmax
3 #include <cstdio>
4 #include <algorithm>
5
6 #define N 800
7
8 using namespace std;
9
10 int n,m,i,j,k,rep,x,y,h[N][N],H[N][N],
11     Hmax,hact,hmax,dif,di[4]={0,0,1,-1},dj[4]={1,-1,0,0};
12 long long sol;
13
14 int main()
15 {
16     freopen("volum.in","r",stdin);
17     freopen("volum.out","w",stdout);
18
19     scanf("%d%d",&n,&m);
20     for(i=1;i<=n;i++)
21         for(j=1;j<=m;j++)
22         {
23             scanf("%lld",&h[i][j]);
24             Hmax=max(Hmax,h[i][j]);
25         }
26
27     for(i=1;i<=n;i++)
28         for(j=1;j<=m;j++)
29             H[i][j]=Hmax-h[i][j];
30
31     for(rep=1;rep;)
32     {
33         rep=0;
34         for(i=1;i<=n;i++)
35             for(j=1;j<=m;j++)
36                 if(H[i][j])
37                 {
38                     int hvec,hmax=-1;
39                     for(k=0;k<4;k++)
40                     {
41                         hvec=h[i+di[k]][j+dj[k]]+H[i+di[k]][j+dj[k]];
42                         if(hvec<h[i][j]+H[i][j]&&hvec>hmax)hmax=hvec;
43                     }
44                     if(hmax== -1) continue;
45                     hmax=max(hmax,h[i][j]);
46                     H[i][j]=hmax-h[i][j];
47                     rep=1;
48                 }
49     }
50
51     for(i=1;i<=n;i++)
52         for(j=1;j<=m;j++)
53             sol+=H[i][j];
54
55     printf("%lld\n",sol);
56     return 0;
57 }

```

24.3.3 *Rezolvare detaliată

24.4 clepsidra

Problema 4 - clepsidra

100 de puncte

Un graf conex cu N noduri și M muchii poate fi privit ca o clepsidră cu centrul în nodul X , $1 \leq X \leq N$, dacă putem împărți toate nodurile, mai puțin nodul X , în două submulțimi nevide astfel încât orice drum de la un nod dintr-o mulțime la un nod din cealaltă mulțime trece prin nodul X . Voi trebui să determinați numărul de moduri distințe în care putem privi graful ca o clepsidră pentru fiecare din cele N noduri alese drept centru, modulo 666013. Două moduri

se consideră distințe dacă cele două submulțimi aferente sunt diferite. Ordinea submulțimilor într-un mod este relevantă, dar ordinea nodurilor în cadrul unei mulțimi nu este. Spre exemplu, soluțiile $(\{1,2,3\}, \{4,5\})$ și $(\{4,5\}, \{1,2,3\})$ sunt distințe, dar soluțiile $(\{4,5\}, \{1,2,3\})$ și $(\{4,5\}, \{1,3,2\})$ nu sunt distințe.

Cerințe

Date de intrare

Fișierului de intrare **clepsidra.in** conține pe prima linie două numere naturale, N și M , reprezentând numărul de noduri, respectiv numărul de muchii din graf. Pe următoarele M linii se vor afla câte două numere naturale separate prin căte un spațiu, reprezentând căte o muchie.

Date de ieșire

În fișierul de ieșire **clepsidra.out** se vor afișa N linii. A i -a linie, $1 \leq i \leq N$, va conține numărul de moduri în care putem privi graful ca o clepsidră cu centrul în nodul i , modulo 666013.

Restricții și precizări

- $2 \leq N \leq 200\ 002$
- $1 \leq M \leq 250\ 002$
- Pentru 40% din teste avem restricțiile $2 \leq N \leq 1002$ și $1 \leq M \leq 1502$.
- Atenție! Graful este conex.

Exemplu:

clepsidra.in	clepsidra.out	Explicații
6 7	0	
4 3	0	
1 3	2	
5 4	0	
4 1	2	
3 2	0	
1 5		Pentru nodul cu indicele 3, soluțiile sunt: $(\{2\}, \{1,4,5,6\})$ și $(\{1,4,5,6\}, \{2\})$
5 6		Pentru nodul cu indicele 5, soluțiile sunt: $(\{6\}, \{1,2,3,4\})$ și $(\{1,2,3,4\}, \{6\})$

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

Memorie: total **32** MB din care pentru stivă **8** MB

Dimensiune maximă a sursei: **25** KB

24.4.1 Indicații de rezolvare

Radu Voroneanu

În primul rând se observă că nodurile X sunt noduri critice ale grafului.

În același timp, numărul de submulțimi depinde de numărul de componente conexe rămase prin eliminarea nodului X.

Să presupunem că acest număr este egal cu K.

Răspunsul la problema este dat de $2^k - 2$, deoarece orice componentă conexă poate fi introdusă fie în prima mulțime, fie în a doua.

Trebuie să eliminate cazurile în care toate componentele conexe sunt introduse în aceeași submulțime.

Mai trebuie să determinăm, pentru fiecare nod X, numărul de componente conexe rămase prin eliminarea acestuia din graf.

Vom folosi dinamica clasică de noduri critice prin care determinăm

$Din[nod] =$ adâncimea minimă la care se poate ajunge din subarborele nod

(a unui arbore obținut printr-o parcurgere DF) parcurgând muchiile de întoarcere.

Pentru orice nod, dacă unul din fi și poate atinge un nod superior lui, atunci poate fi considerat ca făcând parte din componenta conexă superioară.

De aceea, $K[X] =$ numărul de fi a lui X care pot urca peste $X + 1$.

Atenție specială trebuie data nodului radăcină care nu are o componentă superioară.

24.4.2 Cod sursă

Listing 24.4.1: clepsidra.cpp

```

1 #include <stdio.h>
2 #include <algorithm>
3 #include <vector>
4
5 using namespace std;
6
7 #define MAXN 300010
8 #define MOD 666013
9
10 vector<int> G[MAXN];
11 int D[MAXN], Lv[MAXN], Ans[MAXN], Power[MAXN];
12 bool Ok[MAXN];
13 int N, M;
14
15 void df(int nod, int tata)
16 {
17     Ok[nod] = true;
18     Lv[nod] = Lv[tata] + 1;
19     int nr = 1;
20     if (nod == 1) nr = 0;
21     D[nod] = Lv[nod];
22     for (vector<int>::iterator it = G[nod].begin(); it != G[nod].end(); ++it)
23     {
24         int nod2 = *it;
25         if (nod2 == tata) continue;
26         if (Ok[nod2])
27         {
28             D[nod] = min(D[nod], Lv[nod2]);
29             continue;
30         }
31         df(nod2, nod);
32         if (D[nod2] >= Lv[nod])
33             ++nr;
34         D[nod] = min(D[nod2], D[nod]);
35     }
36     Ans[nod] = Power[nr];
37 }
38
39 int main()
40 {
41     freopen("clepsidra.in", "r", stdin);
42     freopen("clepsidra.out", "w", stdout);
43
44     scanf("%d %d", &N, &M);
45     for (int i = 1; i <= M; ++i)
46     {
47         int x, y;
48         scanf("%d %d", &x, &y);
49         G[x].push_back(y);
50         G[y].push_back(x);
51     }
52
53     Power[0] = 1;
54     for (int i = 1; i <= N; ++i)
55         Power[i] = (Power[i - 1] * 2) % MOD;
56     for (int i = 1; i <= N; ++i)
57         Power[i] = (Power[i] + MOD - 2) % MOD;
58
59     df(1, 0);
60
61     for (int i = 1; i <= N; ++i)
62         printf("%d\n", Ans[i]);
63     return 0;
64 }
```

Listing 24.4.2: clepsidra_bulaneala.cpp

```

1 #include <stdio.h>
2 #include <algorithm>
```

```

3 #include <vector>
4
5 using namespace std;
6
7 #define MAXN 300000
8 #define MOD 666013
9
10 int Q[MAXN], Lv[MAXN], Power[MAXN];
11 bool Ok[MAXN];
12 vector<int> G[MAXN];
13 int i, N, M, K, nr;
14
15 void df(int nod, int tata)
16 {
17     Q[++nr] = nod;
18     Ok[nod] = true;
19     Lv[nod] = Lv[tata] + 1;
20     vector<int>::iterator it;
21     if (Lv[nod] == 20) return;
22     for (it = G[nod].begin(); it != G[nod].end(); ++it)
23     {
24         if (Ok[*it]) continue;
25         df(*it, nod);
26         if (nod == i)
27             ++K;
28     }
29 }
30
31 int main()
32 {
33     freopen("clepsidra.in", "r", stdin);
34     freopen("clepsidra.out", "w", stdout);
35
36     scanf("%d %d", &N, &M);
37     for (i = 1; i <= M; ++i)
38     {
39         int x, y;
40         scanf("%d %d", &x, &y);
41         G[x].push_back(y);
42         G[y].push_back(x);
43     }
44     Power[0] = 1;
45     for (i = 1; i <= N; ++i)
46         Power[i] = (Power[i - 1] * 2) % MOD;
47
48     for (i = 1; i <= N; ++i)
49     {
50         K = 0; nr = 0; Lv[i] = 0;
51         Ok[i] = true;
52         df(i, 0);
53         for (int j = 1; j <= nr; ++j)
54             Ok[Q[j]] = false;
55         printf("%d\n", (Power[K] + MOD - 2) % MOD);
56     }
57
58     return 0;
59 }
```

Listing 24.4.3: clepsidra_gavrila_log.cpp

```

1 //Vlad Gavrila - O(N+M)
2 #include <cstdio>
3 #include <vector>
4 #include <cstring>
5
6 using namespace std;
7
8 #define maxn 200010
9 #define mod 666013
10
11 int n, m, a, b, cc;
12 int f[maxn], tata[maxn], sol[maxn], niv[maxn], d[maxn];
13 vector<int> v[maxn];
14
15 void df(int nod)
```

```

16  {
17      if(f[nod])
18          return;
19
20      int fiu, cc=0;
21
22      f[nod]=1;
23      d[nod]=niv[nod];
24
25      for(int i=0; i<v[nod].size(); ++i)
26      {
27          fiu=v[nod][i];
28          if(fiu==tata[nod])
29              continue;
30          if(f[fiu])
31          {
32              d[nod]=min(d[nod], niv[fiu]);
33              continue;
34          }
35
36          niv[fiu]=niv[nod]+1;
37          tata[fiu]=nod;
38          df(fiu);
39
40          if(d[fiu]>=niv[nod])
41              ++cc;
42          d[nod]=min(d[nod], d[fiu]);
43      }
44
45      if(tata[nod]!=0)
46          ++cc;
47
48      sol[nod]=cc;
49  }
50
51 inline int p2(int N)
52 {
53     int ret = 1, P = N;
54     N = 2;
55     while (P)
56     {
57         if (P & 1)
58             ret = (1LL * ret * N) % mod;
59         N = (1LL * N * N) % mod;
60         P >= 1;
61     }
62     return ret;
63 }
64
65 int main()
66 {
67     freopen("clepsidra.in", "r", stdin);
68     freopen("clepsidra.out", "w", stdout);
69
70     scanf("%d%d", &n, &m);
71     for(int i=1; i<=m; ++i)
72     {
73         scanf("%d%d", &a, &b);
74         v[a].push_back(b);
75         v[b].push_back(a);
76     }
77
78     niv[1]=1;
79     df(1);
80
81     for(int i=1; i<=n; ++i)
82         printf("%d\n", (p2(sol[i])-2+mod)%mod);
83
84     return 0;
85 }

```

Listing 24.4.4: clepsidra-100cartita.cpp

```

1 //Vlad Gavrila - O(N+M)
2 #include <cstdio>

```

```

3 #include <vector>
4 #include <cstring>
5
6 using namespace std;
7
8 #define maxn 500010
9 #define mod 666013
10
11 int n, m, a, b, cc;
12 int p2[maxn], f[maxn], tata[maxn], sol[maxn], niv[maxn], d[maxn];
13 vector<int> v[maxn];
14
15 void df(int nod)
16 {
17     if(f[nod])
18         return;
19
20     int fiu, cc=0;
21
22     f[nod]=1;
23     d[nod]=niv[nod];
24
25     for(int i=0; i<v[nod].size(); ++i)
26     {
27         fiu=v[nod][i];
28
29         if(fiu==tata[nod])
30             continue;
31
32         if(f[fiu])
33         {
34             d[nod]=min(d[nod], niv[fiu]);
35             continue;
36         }
37
38         niv[fiu]=niv[nod]+1;
39         tata[fiu]=nod;
40         df(fiu);
41
42         if(d[fiu]>=niv[nod])
43             ++cc;
44         d[nod]=min(d[nod], d[fiu]);
45     }
46
47     if(tata[nod]!=0)
48         ++cc;
49
50     sol[nod]=cc;
51 }
52
53 int main()
54 {
55     freopen("clepsidra.in", "r", stdin);
56     freopen("clepsidra.out", "w", stdout);
57
58     scanf("%d%d", &n, &m);
59     for(int i=1; i<=m; ++i)
60     {
61         scanf("%d%d", &a, &b);
62         v[a].push_back(b);
63         v[b].push_back(a);
64     }
65
66     p2[0]=1;
67     for(int i=1; i<=n; ++i)
68         p2[i]=(p2[i-1]*2)%mod;
69
70     niv[1]=1;
71     df(1);
72
73     for(int i=1; i<=n; ++i)
74         printf("%d\n", (p2[sol[i]]-2+mod)%mod);
75
76     return 0;
77 }
```

Listing 24.4.5: clepsidra-brutcartita.cpp

```

1 //Vlad Gavrila - O(N*M)
2 #include <iostream>
3 #include <vector>
4 #include <cstring>
5
6 using namespace std;
7
8 #define maxn 100010
9 #define mod 666013
10
11 int n, m, a, b, cc;
12 int p2[maxn], f[maxn];
13 vector<int> v[maxn];
14
15 void df(int nod)
16 {
17     if(f[nod])
18         return;
19
20     f[nod]=1;
21
22     for(int i=0; i<v[nod].size(); ++i)
23         df(v[nod][i]);
24 }
25
26 int main()
27 {
28     freopen("clepsidra.in", "r", stdin);
29     freopen("clepsidra.out", "w", stdout);
30
31     scanf("%d%d", &n, &m);
32     for(int i=1; i<=m; ++i)
33     {
34         scanf("%d%d", &a, &b);
35         v[a].push_back(b);
36         v[b].push_back(a);
37     }
38
39     p2[0]=1;
40     for(int i=1; i<=n; ++i)
41         p2[i]=(p2[i-1]*2)%mod;
42
43
44     for(int i=1; i<=n; ++i)
45     {
46         memset(f, 0, sizeof(f));
47         f[i]=1;
48         cc=0;
49
50         for(int j=1; j<=n; ++j)
51         {
52             if(f[j]==0)
53             {
54                 ++cc;
55                 df(j);
56             }
57         }
58         printf("%d\n", (p2[cc]-2+mod)%mod);
59     }
60 }
61
62     return 0;
63 }
```

24.4.3 *Rezolvare detaliată

24.5 permutare

Problema 5 - permutare

100 de puncte

Se dă o matrice cu m linii și n coloane, fiecare linie reprezentând o permutare. Se știe că liniile de la 2 la m sunt permutări circulare ale primei liniilor. Unei liniilor x ($1 \leq x \leq m$) i se pot aplica următoarele operații:

- o permutare circulară la stânga: elementul de pe poziția i ($1 < i \leq n$) se mută pe poziția $i - 1$, mai puțin primul primul element, care devine ultimul;
 - o permutare circulară la dreapta: elementul de pe pozitia i ($1 \leq i < n$) se mută pe pozitia $i + 1$, mai puțin ultimul element care devine primul.
- Scopul este să permuteam circular liniile, la stânga sau la dreapta, astfel încât în final toate liniile să fie egale, folosind un număr minim de operații.

Cerințe

Dându-se o matrice cu proprietatea din enunț se cere să se determine numărul minim de operații necesare pentru a ajunge la o matrice în care toate liniile sunt egale.

Date de intrare

Fișierul de intrare **permuteare.in** conține pe prima linie două numere naturale n și m , reprezentând numărul de coloane și numărul de liniile ale matricei.

Pe a doua linie a fișierului de intrare se află n numere naturale, reprezentând permutarea de pe prima linie a matricei.

Pe următoarele $m - 1$ linii, se află câte un număr natural cuprins între 0 și $n - 1$. Al i -lea ($0 < i < m$) dintre aceste numere reprezintă numărul de poziții cu care este permuatată circular la dreapta a $(i + 1)$ -a linie față de linia 1.

Date de ieșire

Pe prima linie a fișierului de ieșire **permuteare.out** se va scrie un singur număr natural reprezentând numărul minim de operații necesare.

Restricții și precizări

- $1 \leq n, m \leq 100\,000$
- Două liniile dintr-un tablou sunt egale dacă elementele aflate pe aceeași coloană sunt egale.

Exemple:

permuteare.in	permuteare.out	Explicații
6 5		Matricea va fi:
3 1 4 2 6 5		3 1 4 2 6 5
1		5 3 1 4 2 6
1		5 3 1 4 2 6
3		2 6 5 3 1 4
3		2 6 5 3 1 4
		Dacă permuteam circular la dreapta prima linie cu o poziție, iar liniile 4 și 5 le permuteam la stânga cu două poziții, vom obține din 5 operații o matrice cu toate liniile egale între ele. Liniile vor fi determinate de permutarea: 5 3 1 4 2 6

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

Memorie: total **32 MB** din care pentru stivă **4 MB**

Dimensiune maximă a sursei: **25 KB**

24.5.1 Indicații de rezolvare

Se poate observă că nu este nevoie să ne construim matricea linie cu linie.

Vom încerca să reținem fiecare linie a matricei prin numărul de permutări circulare față de prima linie.

Apoi vom încerca să indexăm aceste linii în felul urmator: avem initial un vector L de dimensiune n doar cu zerouri.

O linie o salvăm prin incrementarea valorii L[p], unde p reprezintă de câte ori a fost permuată linia respectivă față de prima.

Acum nu vom avea decât să găsim o poziție în acest vector astfel încât dacă deplasăm toate valorile la această poziție, să o facem cu număr minim de operații.

O operație se definește în felul următor: se scade o valoare de la o poziție și se va incrementa unul din vecinii poziției respective.

Inseamnă că linia a cărei valori am mutat-o a fost permuată circular la stânga/dreapta.

O abordare brută, n^2 ar obține 40% din punctaj.

Pentru 100 de puncte, se încearcă optimizarea pasului anterior.

Se observă că atunci când fixăm o pozitie poz, vom deplasa la stânga elementele între pozițiile poz și poz + n/2, și la dreapta elementele între poz - n/2 și poz.

Pornim de la dreapta la stânga, și inițial calculăm pentru poziția 1 prin o parcurgere a întregului sir, numărul de operații pentru a deplasa elementele de la dreapta lui 1 (elementele de pe pozițiile de la 1 la n/2), și numărul de operații pentru a deplasa elementele de la stânga lui 1 (elementele de pe pozițiile de la n/2+1 la n).

De altfel va trebui reținut și numărul de elemente aflate la stânga, respectiv la dreapta lui 1.

Acum vom încerca să calculăm soluția pentru poziția 2, iar operația de update să o facem în timp constant, O(1).

Aceasta se poate face usor, prin observații simple. Suma se va updata în felul următor: din sumă se scade numărul de elemente de la dreapta lui, iar în suma se adună numărul de elemente de la stânga lui.

Va trebui pe urmă să updatăm numărul de elemente aflate la dreapta lui 2 și numărul de elemente la stânga lui 2.

Vom face pe urma acestei operații succesiv, pentru fiecare i de la 3 la n.

În final, afișăm suma minima care am obținut-o în timpul acestei parcurgeri.

24.5.2 Cod sursă

Listing 24.5.1: brut_40.cpp

```

1 #include <iostream>
2 #include <fstream>
3
4 using namespace std;
5
6 #define MAX 1000000000
7
8 int n , m , a[100000];
9
10 int optim()
11 {
12     int sol = MAX, s =0 ;
13     int right_num_el = n/2;
14     int left_num_el = n/2 - (n % 2 == 0) ;
15     int right_sum = 0, left_sum = 0, left_nr = 0, right_nr = 0;
16     int pointer = 1 + right_num_el;
17     int i;
18
19     for (i = pointer ; i>=1 ;i--)
20     {
21         right_sum = right_sum + right_nr;
22         right_nr += a[i];
23     }
24     right_nr -= a[1];
25
26     for (i = pointer+1; i<=n; i++)
27     {
28         left_sum = left_sum + left_nr;
29         left_nr += a[i];
30     }

```

```

31     left_sum += left_nr;
32
33     s = left_sum + right_sum;
34     sol = s;
35
36     for (i = 2; i<=n ;i++)
37     {
38         pointer++;
39         if(pointer > n)
40             pointer = 1;
41         right_sum -= right_nr;
42         right_nr -= a[i];
43         right_nr += a[pointer];
44         right_sum += right_num_el * a[pointer];
45
46         left_nr -= a[pointer];
47         left_nr += a[i-1];
48         left_sum += left_nr;
49         left_sum -= a[pointer] * left_num_el;
50
51         s = left_sum + right_sum;
52         sol = min(sol, s);
53     }
54
55     return sol;
56 }
57
58 int brut()
59 {
60     int i, j;
61     int sol = MAX,s;
62
63     for ( i = 1; i<= n; i++)
64     {
65         s = 0;
66         for (j = 1;j <= n; j++)
67             if (j < i)
68                 s += a[j] * min(i -j, j + n-i);
69             else
70                 s += a[j] * min(j -i, n - j + i);
71         sol = min (sol, s);
72     }
73
74     return sol;
75 }
76
77 void citire()
78 {
79     int x;
80     ifstream f("permutare.in");
81     f >> n >> m ;
82     for (int i = 1 ;i <= n ; i++)
83         f>> x;
84
85     a[1] = 1;
86     for ( int i =1; i< m ;i++)
87     {
88         f >> x;
89         x ++ ;
90         if (x > n) x = 1;
91         a[x] ++ ;
92     }
93     f.close();
94 }
95
96 int main()
97 {
98     citire();
99     ofstream g("permutare.out");
100    g << brut();
101    g.close();
102 //    cout << optim();
103    return 0;
104 }
```

Listing 24.5.2: permutare-100cartita.cpp

```

1 #include <cstdio>
2 #include <algorithm>
3
4 using namespace std;
5
6 #define maxn 100010
7
8 int n, m, x;
9 int v[maxn];
10 long long sc, sol;
11 int pp, mm, ps[maxn], mn[maxn];
12
13 int main()
14 {
15     freopen("permutare.in", "r", stdin);
16     freopen("permutare.out", "w", stdout);
17
18     scanf("%d%d", &n, &m);
19     for(int i=1; i<=n; ++i)
20         scanf("%d", &x);
21     for(int i=1; i<m; ++i)
22     {
23         scanf("%d", &v[i]);
24
25         --mn[v[i]];
26         ++ps[v[i]];
27
28         --ps[(v[i]+n/2)%n];
29         ++mn[(v[i]+n/2+n%2)%n];
30
31         if(v[i]<n/2)
32             ++mm;
33         if(v[i]>=n/2+n%2)
34             ++pp;
35
36         sol+=min(v[i], n-v[i]);
37     }
38
39     sc=sol;
40
41     for(int i=0; i<n; ++i)
42     {
43         sol=min(sol, sc+min(i, n-i));
44
45         pp+=ps[i];
46         mm+=mn[i];
47
48         sc+=pp-mm;
49     }
50
51     printf("%lld\n", sol);
52
53     return 0;
54 }
```

Listing 24.5.3: permutare-ciocan-100.cpp

```

1 #include <iostream>
2 #include <fstream>
3
4 using namespace std;
5
6 #define MAX 1000000000
7
8 long long n, m, a[100000];
9
10 long long optim()
11 {
12     long long sol = MAX, s = 0 ;
13     long long right_num_el = n/2;
14     long long left_num_el = n/2 - (n % 2 == 0) ;
15     long long right_sum = 0, left_sum = 0, left_nr = 0, right_nr = 0;
16     long long pointer = 1 + right_num_el;
17     long long i;
```

```

18     for (i = pointer ; i>=1 ;i--)
19     {
20         right_sum = right_sum + right_nr;
21         right_nr += a[i];
22     }
23     right_nr -= a[1];
24
25     for (i = pointer+1; i<=n; i++)
26     {
27         left_sum = left_sum + left_nr;
28         left_nr += a[i];
29     }
30     left_sum += left_nr;
31
32     s = left_sum + right_sum;
33     sol = s;
34
35     for (i = 2; i<=n ;i++)
36     {
37
38         pointer++;
39         if(pointer > n)
40             pointer = 1;
41         right_sum -= right_nr;
42         right_nr -= a[i];
43         right_nr += a[pointer];
44         right_sum += right_num_el * a[pointer];
45
46         left_nr -= a[pointer];
47         left_nr += a[i-1];
48         left_sum += left_nr;
49         left_sum -= a[pointer] * left_num_el;
50
51         s = left_sum + right_sum;
52         sol = min(sol, s);
53     }
54
55     return sol;
56 }
57
58
59 void citire()
60 {
61     long long x;
62     ifstream f("permutare.in");
63     f >> n >> m;
64     for (int i = 1 ;i <= n ; i++)
65         f>>x;
66     a[1] = 1;
67     for ( int i =1; i< m ;i++)
68     {
69         f >> x;
70         x ++ ;
71         if (x > n) x = 1;
72         a[x] ++ ;
73     }
74     f.close();
75 }
76
77 int main()
78 {
79     citire();
80     ofstream g("permutare.out");
81 //    g << brut();
82     g << optim();
83     g.close();
84 //    cout << optim();
85     return 0;
86 }
```

Listing 24.5.4: permutarePA.cpp

```

1 //Panaete Adrian O(n)
2 #include <cstdio>
3
```

```

4 #define N 200010
5
6 using namespace std;
7
8 long long n,m,i,ceva,now,bst,r,a[N],s[N],M,L,R,t,add,Add;
9
10 int main()
11 {
12     freopen("permutare.in", "r", stdin);
13     freopen("permutare.out", "w", stdout);
14
15     scanf("%lld%lld", &n, &m);
16     for(i=1;i<=n;i++) scanf("%lld", &ceva);
17
18     for(m--;m;m--)
19     {
20         scanf("%d", &r);
21         r%=n;
22         a[r+1]++;
23         a[n+r+1]++;
24     }
25
26     for(i=1;i<=2*n;i++)
27         s[i]=s[i-1]+a[i];
28     M=(n+1)/2;
29     for(i=M-1;i>=1;i--)
30     {
31         bst+=(M-i)*a[i];
32         L++;
33     }
34     for(i=M+1;i<=n;i++)
35     {
36         bst+=(i-M)*a[i];
37         R++;
38     }
39     now=bst;
40     add=M-1;
41     for(t=n-1;t;t--)
42     {
43         now=now+(R-L)*a[M-L]+2*s[M]-s[M-L]-s[M+R];
44         M++;
45         Add=M<=n?n+1-M:M-n-1;
46         if(bst+add>now+Add)
47         {
48             bst=now;
49             add=Add;
50         }
51     }
52
53     printf("%lld\n", bst+add);
54     return 0;
55 }
```

Listing 24.5.5: permutarePAn2.cpp

```

1 //Panaete Adrian O(N^2)
2 #include <cstdio>
3 #include <algorithm>
4
5 #define N 200010
6 #define tip long long
7
8 using namespace std;
9
10 tip n,m,i,ceva,r,a[N],bst,st,dr;
11 void upd(tip);
12
13 int main()
14 {
15     freopen("permutare.in", "r", stdin);
16     freopen("permutare.out", "w", stdout);
17
18     scanf("%lld%lld", &n, &m);
19     for(i=1;i<=n;i++) scanf("%lld", &ceva);
20
```

```

21     for (m--; m; m--)
22     {
23         scanf("%lld", &r);
24         r%=n; a[r+1]++;
25     }
26
27     bst=1LL*N*N;
28     st=n/2;
29     dr=n-n/2-1;
30     for(i=1;i<=n;i++)
31         upd(i);
32     printf("%lld\n",bst);
33     return 0;
34 }
35
36 void upd(tip poz)
37 {
38     long long ret, j, k;
39     ret=min(poz-1,n+1-poz);
40     for(j=1;j<=st;j++)
41     {
42         k=poz-j>0?poz-j:poz-j+n;
43         ret+=j*a[k];
44     }
45
46     for(j=1;j<=dr;j++)
47     {
48         k=poz+j<=n?poz+j:poz+j-n;
49         ret+=j*a[k];
50     }
51     bst=min(bst,ret);
52 }
```

24.5.3 *Rezolvare detaliată

24.6 xcmmdc

Problema 6 - xcmmdc

100 de puncte

Se dă o matrice cu m linii și n coloane, cu elementele numere naturale nenule și un număr natural nenul fixat k .

Cerințe

Pentru matricea dată și numărul k dat să se răspundă la q întrebări de forma: "Câte submatrici pătratice de latură L cu cel mai mare divizor comun al elementelor egal cu k există în matricea dată?"

Date de intrare

Fișierul de intrare **xcmmdc.in** conține pe prima linie patru numere naturale nenule separate prin câte un spațiu: m și n - numărul de linii și numărul de coloane ale matricei, k - numărul natural dat și q - numărul de întrebări.

Pe următoarele m linii se găsesc liniile matricei. Fiecare dintre aceste linii conține câte n numere naturale separate prin câte un spațiu - elementele liniei corespunzătoare din matrice.

Următoarele q linii descriu întrebările. Fiecare dintre aceste linii conține câte un număr natural L - latura submatricei din întrebarea corespunzătoare.

Date de ieșire

Fișierul de ieșire **xcmmdc.out** va conține q linii. Pe fiecare dintre aceste linii se va scrie un singur număr natural reprezentând răspunsul la întrebarea corespunzătoare din fișierul de intrare.

Restricții și precizări

- $1 \leq n, m \leq 1002$
- Pentru 50% din teste $1 \leq n, m \leq 502$
- $1 \leq q \leq 50\ 002$
- $1 \leq 10^9 + 2$
- Elementele matricei sunt numere naturale nenule mai mici sau egale decât $10^9 + 2$.
- $1 \leq L \leq \min(m, n)$ pentru fiecare întrebare
- Prin submatrice pătratică de latură L se înțelege o matrice obținută prin intersecția a L linii consecutive cu L coloane consecutive din matrice.

Exemple:

xcmmdc.in	xcmmdc.out	Explicații
3 3 3 4	2	Pentru prima și ultima întrebare avem două submatrice:
3 6 2	3	3 6
9 12 3	0	9 12
2 6 3	2	—
2		12 3
1		6 3
3		Prima submatrice se obține prin intersecția primelor două linii cu primele două coloane, iar a doua prin intersecția ultimelor două linii cu ultimele două coloane.
2		

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

Memorie: total **128 MB** din care pentru stivă **16 MB**

Dimensiune maximă a sursei: **25 KB**

24.6.1 Indicații de rezolvare

Soluția 1 - 100 de puncte, Radu Voroneanu

Observăm întai că numărul de query-uri este mult mai mare decât dimensiunea maximă a unui query, aşa că este necesar să precalculăm răspunsul pentru fiecare latură posibilă.

Vom construi întai o dinamică de forma:

$DP[l][i][j] = cmmdc-ul unei submatrice pătratice de latură 2^l care are colțul stânga sus pe poziția (i, j) .$

Recurența acestei dinamici este:

$$\begin{aligned} DP[l+1][i][j] &= cmmdc(DP[l][i][j], DP[l][i+2^l][j], DP[l][i][j+2^l]), \\ DP[l][i+2^l][j+2^l] \end{aligned}$$

Pentru a continua, trebuie să facem următoarea observație: dacă A este cmmdc-ul unei submatrice de latură l cu colțul stânga sus în (x, y) și B este cmmdc-ul unei submatrice de latură $l+1$ cu colțul stânga sus în (x, y) , atunci $A \geq B$.

Acest lucru este ușor de demonstrat întrucât adăugând elemente la o submatrice nu îi putem crește cmmdc-ul ($cmmdc(a, b) \leq a, b$).

Astfel, pentru fiecare poziție (x, y) putem căuta binar care este latura minimă și latura maximă a unei submatrice care are cmmdc-ul K. Ne vom folosi de dinamica DP pentru a determina în $O(1)$ răspunsul la întrebări de forma: care este cmmdc-ul C al unei submatrice de latură L cu colțul stânga-sus în poziția (x, y) .

Fie P cel mai mare număr natural astfel încât $2^P \leq X$. Atunci,

$$C = cmmdc(DP[P][x][y], DP[P][x+L-2^P][y], DP[P][x][y+L-2^P], DP[P][x+L-2^P][y+L-2^P])$$

Știind latura minimă Lmin și latura maximă Lmax a unei submatrice de cmmdc K pentru fiecare poziție (x, y) , putem să ne folosim de un arbore de intervale pentru a incrementa numărul de soluții pentru toate laturile din intervalul $[Lmin, Lmax]$, folosindu-ne apoi de același arbore pentru a reține răspunsul final pentru toate laturile din intervalul $[1, \min(N, M)]$.

Alternativ, ne putem folosi de *tehnica lui Mars* în locul arborelui de intervale pentru a implementa această parte din soluție.

Complexitatea finală a acestei soluții este $O(N*M*\log(N*M)*\log(MAX))$ ca timp și ca spațiu.

<http://www.infoarena.ro/multe-smenuri-de-programare-in-cc-si-nu-numai>

"Smenul lui Mars" (Marius Andrei)

Consideram urmatoarea problema: se da un vector A de N elemente pe care se fac M astfel de operații:

ADUNA(st, dr, x) - toate elementele cu indicii intre st si dr ($0 \leq st \leq dr < N$) isi cresc valoarea cu x .

La sfarsit trebuie sa se afiseze vectorul rezultat.

In continuarea vom descrie o metoda care ne da un timp de rulare de $O(1)$ pentru operatia ADUNA si $O(N)$ pentru a determina toate elementele din vector. Vom construi un al doilea vector B de $N+1$ elemente, cu proprietatea ca $A_i = B_0 + B_1 + \dots + B_i$.

Astfel, o operatie ADUNA(st, dr, x) devine:

$B[st] += x;$

$B[dr + 1] -= x;$

Da, este chiar asa de simplu!

Pentru a determina un element A_i vom aduna pur si simplu $B_0 + B_1 + \dots + B_i$.

Incercați pe foaie să vedeti cum funcționează.

Aceasta ideea poate fi extinsă și în două dimensiuni, construind B astfel încât

$A_{i,j} = \text{suma subtabloului din } B \text{ cu colțul în } (0, 0) \text{ și } (i, j)$, astfel (pt. ADUNA(x1,y1,x2,y2,v)):

$B[x1][y1] += v;$

$B[x1][y2 + 1] -= v;$

$B[x2 + 1][y1] -= v;$

$B[x2 + 1][y2 + 1] += v;$

Pe cazul general, dacă vrem să facem operații în d dimensiuni vom avea o complexitate $O(2^d)$.

Reamintesc că aceasta metoda este eficientă doar când se vrea afisata vectorul/matricea/etc. doar la sfârșitul operațiilor sau sunt foarte puține interogări ale valorilor elementelor, deoarece afătarea unui element este o operație foarte ineficientă: $O(i)$ pentru a afla valorile elementelor până la poziția i.

Soluția 2 - 50 puncte, Vlad Gavrilă

Vom construi o dinamică

$D[l][i][j] = \text{cmmdc-ul unei submatrice de latură } l \text{ cu colțul stânga sus în poziția } (i, j)$.

Recurența acestei dinamici va fi:

$D[i+1][i][j] = \text{cmmdc}(D[i][i][j], D[i][i+1][j], D[i][i][j+1], D[i][i+1][j+1])$

Pentru fiecare latură l între 1 și $\min(N, M)$ vom număra cu ajutorul acestei dinamici câte submatrice de latură l au cmmdc-ul K.

Observăm că este necesar să ținem doar ultimele 2 linii ale acestei dinamici pentru a ne încadra în limita de memorie.

Complexitatea acestei soluții este $O(N*M*\min(N, M)*\log(\text{MAX}))$ ca timp și $O(N*M)$ ca memorie.

24.6.2 *Cod sursă

Listing 24.6.1: xcmmdc.cpp

```

1 #include <stdio.h>
2 #include <algorithm>
3
4 using namespace std;
5
6 #define MAXN 1024
7 #define MAXLOG 11
8
9 int RMQ[MAXLOG][MAXN][MAXN];
10 int Mars[MAXN], Log[MAXN];
11 int N, M, K, Q;
12 int minim, maxim, st, dr, mij, aux, sh, lg;
13 int i, j;
14
15 inline int gcd(int a, int b)
16 {
17     int r;
18     while (b)
19     {
20         r = a % b;
21         a = b;
22         b = r;
23     }
24     return a;

```

```

25 }
26
27 int main()
28 {
29     freopen("cmmdc.in", "r", stdin);
30     freopen("cmmdc.out", "w", stdout);
31
32     scanf("%d %d %d %d", &N, &M, &K, &Q);
33     for (i = 1; i <= N; ++i)
34         for (j = 1; j <= M; ++j)
35             scanf("%d", &RMQ[0][i][j]);
36
37     for (lg = 1; (1 << lg) <= N; ++lg)
38         for (i = 1; i <= N - (1 << lg) + 1; ++i)
39             for (j = 1; j <= M - (1 << lg) + 1; ++j)
40             {
41                 sh = 1 << (lg - 1);
42                 aux = gcd(RMQ[lg-1][i][j], RMQ[lg-1][i + sh][j]);
43                 aux = gcd(aux, RMQ[lg-1][i][j + sh]);
44                 aux = gcd(aux, RMQ[lg-1][i + sh][j + sh]);
45                 RMQ[lg][i][j] = aux;
46             }
47
48     Log[1] = 0;
49     for (int i = 2; i <= N || i <= M; ++i)
50         Log[i] = Log[i >> 1] + 1;
51
52     for (i = 1; i <= N; ++i)
53         for (j = 1; j <= M; ++j)
54         {
55             minim = 0; st = 1; dr = min(N - i + 1, M - j + 1);
56             while (st <= dr)
57             {
58                 mij = (st + dr) >> 1;
59                 lg = Log[mij];
60                 sh = 1 << lg;
61                 aux = gcd(RMQ[lg][i][j], RMQ[lg][i + mij - sh][j]);
62                 aux = gcd(aux, RMQ[lg][i][j + mij - sh]);
63                 aux = gcd(aux, RMQ[lg][i + mij - sh][j + mij - sh]);
64                 if (aux > K)
65                 {
66                     minim = mij;
67                     st = mij + 1;
68                 }
69                 else
70                     dr = mij - 1;
71             }
72
73             ++minim;
74             maxim = 0;
75             st = 1;
76             dr = min(N - i + 1, M - j + 1);
77             while (st <= dr)
78             {
79                 mij = (st + dr) >> 1;
80                 lg = Log[mij];
81                 sh = 1 << lg;
82                 aux = gcd(RMQ[lg][i][j], RMQ[lg][i + mij - sh][j]);
83                 aux = gcd(aux, RMQ[lg][i][j + mij - sh]);
84                 aux = gcd(aux, RMQ[lg][i + mij - sh][j + mij - sh]);
85                 if (aux >= K)
86                 {
87                     maxim = mij;
88                     st = mij + 1;
89                 }
90                 else
91                     dr = mij - 1;
92             }
93
94             if (minim <= maxim)
95             {
96                 Mars[minim]++;
97                 Mars[maxim+1]--;
98             }
99         }
100 }
```

```

101     for (i = 1; i <= N || i <= M; ++i)
102         Mars[i] += Mars[i-1];
103
104     for (i = 1; i <= Q; ++i)
105     {
106         scanf("%d", &aux);
107         printf("%d\n", Mars[aux]);
108     }
109
110     return 0;
111 }
```

Listing 24.6.2: xcmmdc_brut.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 ifstream fin("xcmmdc.in");
6 ofstream fout("xcmmdc.out");
7
8 const int NMAX = 1005;
9 const int LOGMAX = 12;
10
11 int RMQ[LOGMAX][NMAX][NMAX];
12 int res[NMAX], lg[NMAX];
13 int N, M, K, Q;
14
15 void read ()
16 {
17     fin >> N >> M >> K >> Q;
18     for(int i = 1; i <= N; ++i)
19         for(int j = 1; j <= M; ++j)
20             fin >> RMQ[0][i][j];
21 }
22
23 inline int gcd(int A, int B)
24 {
25     if (!B)
26         return A;
27     return gcd(B, A % B);
28 }
29
30 inline int get_cmmdc(int x, int y, int L)
31 {
32     int lgL = lg[L];
33     return gcd(gcd(RMQ[lgL][x][y],
34                     RMQ[lgL][x + L - (1 << lgL)][y]),
35                     gcd(RMQ[lgL][x][y + L - (1 << lgL)],
36                         RMQ[lgL][x + L - (1 << lgL)][y + L - (1 << lgL)]));
37 }
38
39 void solve ()
40 {
41     int NM = min (N, M);
42     for(int i = 1; i <= NM; ++i)
43     {
44         if (i > 1)
45             lg[i] = lg[i >> 1] + 1;
46         res[i] = -1;
47     }
48
49     for(int step = 1; (1 << step) <= NM; ++step)
50         for(int i = 1; i <= N - (1 << step) + 1; ++i)
51             for(int j = 1; j <= M - (1 << step) + 1; ++j)
52                 RMQ[step][i][j] = gcd(gcd(RMQ[step - 1][i][j],
53                     RMQ[step - 1][i + (1 << (step - 1))][j]),
54                     gcd(RMQ[step - 1][i][j + (1 << (step - 1))],
55                         RMQ[step - 1][i + (1 << (step - 1))][j + (1 << (step - 1))]));
56
57     for (int i = 1; i <= Q; ++i)
58     {
59         int L; fin >> L;
60
61         if (res[L] == -1)
```

```

62         {
63             int cnt = 0;
64             for (int x = 1; x <= N - L + 1; ++x)
65                 for (int y = 1; y <= M - L + 1; ++y)
66                 {
67                     if (get_cmmdc(x, y, L) == K)
68                         ++ cnt;
69                 }
70             res[L] = cnt;
71         }
72     }
73     fout << res[L] << "\n";
74 }
75
76 int main ()
77 {
78     read();
79     solve();
80
81     return 0;
82 }
```

Listing 24.6.3: xcmmdc-cartital.cpp

```

1 #include <iostream>
2 #include <algorithm>
3
4 using namespace std;
5
6 #define maxn 1010
7
8 int n, m, k, x, q, g, hasBeen;
9 int v[maxn][maxn];
10 int d[maxn][maxn];
11 int sol[maxn];
12
13 int cmmdc(int a, int b)
14 {
15     int r=a%b;
16
17     while(r)
18     {
19         a=b;
20         b=r;
21         r=a%b;
22     }
23
24     return b;
25 }
26
27 int main()
28 {
29     freopen("xcmmdc.in", "r", stdin);
30     freopen("xcmmdc.out", "w", stdout);
31
32     scanf("%d%d%d%d", &n, &m, &k, &q);
33
34     for(int i=1; i<=n; ++i)
35         for(int j=1; j<=m; ++j)
36         {
37             scanf("%d", &v[i][j]);
38             if(v[i][j]==k)
39             {
40                 ++sol[1];
41                 hasBeen=1;
42             }
43         }
44
45     for(int l=2; l<=min(n, m) && (sol[l-1]>0 || hasBeen==0); ++l)
46     {
47         int mn=n-l+1, mm=m-l+1;
48         for(int i=1; i<=mn; ++i)
49             for(int j=1; j<=mm; ++j)
```

```
50
51         {
52             v[i][j]=cmmdc(cmmdc(v[i][j], v[i][j+1]),
53                             cmmdc(v[i+1][j], v[i+1][j+1]));
54             if(v[i][j]==k)
55             {
56                 ++sol[l];
57                 hasBeen=1;
58             }
59         }
60
61     while(q--)
62     {
63         scanf("%d", &x);
64         printf("%d\n", sol[x]);
65     }
66
67     return 0;
68 }
```

Listing 24.6.4: xcmmdc-cartita2.cpp

```

54     while(q--)
55     {
56         scanf("%d", &x);
57         printf("%d\n", sol[x]);
58     }
59 }
60
61     return 0;
62 }
```

Listing 24.6.5: xcmmdc-cartita3.cpp

```

1 #include <cstdio>
2 #include <algorithm>
3 #include <cstring>
4
5 using namespace std;
6
7 #define maxn 1010
8
9 int n, m, k, p, x, q, g, hasBeen;
10 int v[maxn][maxn];
11 int d[maxn][maxn];
12 int sol[maxn];
13 char s[maxn*10];
14
15 int cmmdc(int a, int b)
16 {
17     int r=a%b;
18
19     while(r)
20     {
21         a=b;
22         b=r;
23         r=a%b;
24     }
25
26     return b;
27 }
28
29 int getNr()
30 {
31     int nr=0;
32
33     while(s[p]<'0' || s[p]>'9')
34         ++p;
35
36     while(s[p]>='0' && s[p]<='9')
37     {
38         nr=nr*10+s[p]-'0';
39         ++p;
40     }
41
42     ++p;
43
44     return nr;
45 }
46
47 int main()
48 {
49     freopen("xcmmdc.in", "r", stdin);
50     freopen("xcmmdc.out", "w", stdout);
51
52     scanf("%d%d%d%d\n", &n, &m, &k, &q);
53
54     for(int i=1; i<=n; ++i)
55     {
56         memset(s, 0, sizeof(s));
57         fgets(s, maxn*10, stdin);
58         p=0;
59         for(int j=1; j<=m; ++j)
60         {
61             v[i][j]=getNr();
62             if(v[i][j]==k)
63             {
```

```

64             hasBeen=1;
65         }
66     }
67 }
68
69 for(int l=2; l<=min(n, m) && (sol[l-1]>0 || hasBeen==0); ++l)
70 {
71     int mn=n-l+1, mm=m-l+1;
72     for(int i=1; i<=mn; ++i)
73         for(int j=1; j<=mm; ++j)
74         {
75             v[i][j]=cmmdc(cmmdc(v[i][j], v[i][j+1]),
76                             cmmdc(v[i+1][j], v[i+1][j+1]));
77             if(v[i][j]==k)
78             {
79                 ++sol[l];
80                 hasBeen=1;
81             }
82         }
83     }
84 }
85
86 while(q--)
87 {
88     scanf("%d", &x);
89     printf("%d\n", sol[x]);
90 }
91
92 return 0;
93 }
```

Listing 24.6.6: xcmmdc-cartita4.cpp

```

1 #include <cstdio>
2 #include <algorithm>
3 #include <cstring>
4
5 using namespace std;
6
7 #define maxn 1010
8
9 int n, m, p, k, x, q, g, hasBeen;
10 int v[maxn][maxn];
11 int d[maxn][maxn];
12 int sol[maxn];
13 char s[maxn*10];
14
15 int cmmdc(int a, int b)
16 {
17     int r=a%b;
18
19     while(r)
20     {
21         a=b;
22         b=r;
23         r=a%b;
24     }
25
26     return b;
27 }
28
29 int getNr()
30 {
31     int nr=0;
32
33     while(s[p]<'0' || s[p]>'9')
34         ++p;
35
36     while(s[p]>='0' && s[p]<='9')
37     {
38         nr=nr*10+s[p]-'0';
39         ++p;
40     }
41
42     ++p;
```

```

43
44     return nr;
45 }
46
47 int main()
48 {
49     freopen("xcmmdc.in", "r", stdin);
50     freopen("xcmmdc.out", "w", stdout);
51
52     scanf("%d%d%d%d\n", &n, &m, &k, &q);
53
54     for(int i=1; i<=n; ++i)
55     {
56         memset(s, 0, sizeof(s));
57         fgets(s, maxn*10, stdin);
58         p=0;
59         for(int j=1; j<=m; ++j)
60         {
61             v[i][j]=getNr();
62             if(v[i][j]==k)
63                 ++sol[1];
64         }
65     }
66
67     for(int l=2; l<=min(n, m); ++l)
68     {
69         int mn=n-l+1, mm=m-l+1;
70         for(int i=1; i<=mn; ++i)
71             for(int j=1; j<=mm; ++j)
72             {
73                 v[i][j]=cmmdc(cmmdc(v[i][j], v[i][j+1]),
74                                 cmmdc(v[i+1][j], v[i+1][j+1]));
75                 if(v[i][j]==k)
76                     ++sol[l];
77             }
78     }
79
80     while(q--)
81     {
82         scanf("%d", &x);
83         printf("%d\n", sol[x]);
84     }
85
86     return 0;
87 }
```

24.6.3 *Rezolvare detaliată

Capitolul 25

ONI 2013

25.1 amici

Problema 1 - amici

100 de puncte

În cadrul Comisiei de la clasele 11-12 a apărut, în mod natural, o rețea de socializare. Inițial între cei N membri ai Comisiei există M relații de prietenie. În fiecare zi, se formează noi asemenea relații după următoarea regulă: dacă membrul A nu este încă prieten cu membrul B , dar ei au cel puțin un prieten comun, atunci A și B vor deveni prieteni în ziua imediat următoare.

Această socializare intensă va naște, bineînteles, multe povești și anecdotă care le vor înveseli în mod cert viitoarele întâlniri. Din păcate, autorul este insensibil la această latură umanistă a activității comisiei și insistă că situația prezintă, este de fapt doar o oportunitate pentru o provocare algoritmică. El se întreabă câte zile va dura până când orice membru al comisiei va deveni prieten cu orice alt membru. Deoarece comisia are multi membri anul acesta, iar autorul nu are, de fel, standarde foarte ridicate, acesta se multumeste cu o aproximare a rezultatului. Mai exact, dacă răspunsul adevărat este X , atunci răspunsurile $X + 1$ sau $X - 1$ sunt considerate și ele acceptabile.

Cerințe

Dându-se numerele N și M , cât și cele M relații de prietenie dintre membrii comisiei, să se estimeze câte zile trebuie să treacă până când există relație de prietenie între oricare doi membri ai comisiei.

Date de intrare

Pe prima linie a fisierului **amici.in** se va afla numărul T , reprezentând numărul de teste din fișier, fiecare test respectând următorul format:

Prima linie conține două numere naturale N și M , cu semnificația din enunț.

Următoarele M linii vor conține câte două numere X și Y , semnificând faptul că X și Y sunt prieteni inițial. O anumită pereche X Y se poate repeta în cadrul fisierului de intrare.

Date de ieșire

În fișierul **amici.out** se vor afișa T linii, fiecare conținând răspunsul pentru câte un test în ordinea în care sunt date testele.

Restricții și precizări

- $1 \leq N \leq 21.000$
- $0 \leq M \leq 41.000$
- $1 \leq T \leq 5$
- Se garantează că răspunsul este finit.
- Reamintim că răspunsul este considerat corect chiar dacă diferă cu o unitate față de răspunsul adevărat.

Exemple:

amici.in	amici.out	Explicații
2	1	În primul caz membrii 2 și 3 devin prieteni după o zi, avându-l prieten comun pe membrul 1. Astfel, toate relațiile posibile au apărut după o singură zi.
3 2	2	În cel de al doilea caz au apărut următoarele relații în prima zi: (1, 5), (3, 2), (1, 4), (4, 5).
1 2		În cea de a doua zi, apar și relațiile (3, 5) și (3, 4), rețeaua devenind astfel completă.
1 3		
5 4		
3 1		
1 2		
2 4		
2 5		

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

Memorie: total **16 MB** din care pentru stivă **8 MB**

Dimensiune maximă a sursei: **10 KB**

25.1.1 Indicații de rezolvare

stud. Mihai Calancea, Universitatea Bucharest

Se observă că rețeaua menționată în enunțul problemei poate fi modelată sub forma unui *graf neorientat conex*.

Astfel, la fiecare pas, apar muchii noi între nodurile aflate la distanța 2 în graf iar răspunsul este dat de momentul în care *graful devine complet*.

Este clar că ultimele noduri legate prin muchie directă vor fi perechile de noduri cu distanță minimă maximă în graful initial. Distanța maximă dintre două noduri ale unui graf se numește *diametrul grafului*, motiv pentru care va fi notată în continuare cu D .

Se observă că diametrul se injumătățește la fiecare pas, iar cu puțina analiză putem trage concluzia că răspunsul problemei este dat de cel mai mic K pentru care $2^K \geq D$.

Problema se reduce astfel la calcularea valorii D .

Algoritmul brut implică aplicarea unei *parcurgeri în lățime* pentru fiecare nod al grafului. Acest algoritm are complexitatea $O(N * M)$ și nu este fezabil având în vedere limitele problemei.

Din păcate, aceasta este și cea mai bună complexitate pe care o poate lua un *algoritm deterministic* de găsire a diametrului unui graf, găsirea unei soluții mai eficiente ramanând o *problema deschisă*.

Total, problema de fata se poate approxima cu eroare de o unitate în timp liniar.

Fie X prima zi în care un nod A are muchie directă cu toate celelalte noduri. Este evident că după acest punct, procesul mai poate dura o singură zi în plus, deoarece orice pereche de noduri neadiacente va folosi nodul A pentru a adăuga muchiile necesare.

Răspunsul real al problemei este deci fie X , fie $X + 1$.

Aceasta soluție implică aplicarea unei *parcurgeri în lățime* pentru un singur nod al grafului (oricare dintre ele) și se traduce prin faptul că pentru orice nod X din graf, distanța maximă a unui nod Y fată de X este cel puțin jumătate din diametrul real al grafului.

Să demonstrăm acest lucru.

Fie Z și T două noduri care determină diametrul real al grafului, D și fie d distanța dintre X și Y . Stim sigur că distanțele de la X la Z respectiv de la X la T sunt $\leq d$ (altfel, unul din aceste două noduri ar fi înlocuit nodul Y). Astfel, există sigur un drum între Z și T de lungime $\leq 2 * d$, deci D este la rândul său mai mic sau egal cu $2 * d$.

Menționăm că fișierele de ieșire ale comisiei conțin, evident, răspunsul exact.

25.1.2 Cod sursă

Listing 25.1.1: amici.cpp

```

1 #include <iostream>
2 #include <queue>
3 #include <vector>
4 #include <cassert>
5
6 using namespace std;
7

```

```

8 const int maxn = 200 * 1000 + 5;
9 const int INF = 2 * 1000 * 1000;
10
11 int n, m, i, j, a, b, diametru, d[maxn], ans, t;
12 vector<int> G[maxn];
13 queue <int> Q;
14
15 void BFS(int sursa)
16 {
17
18     Q.push(sursa);
19
20     for(int i = 1; i <= n; ++i)
21         d[i] = INF;
22
23     d[sursa] = 0;
24
25     for(;!Q.empty();)
26     {
27         int act = Q.front();
28         Q.pop();
29
30         for(int i = 0; i < G[act].size(); ++i)
31         {
32             int vecin = G[act][i];
33             if(d[vecin] == INF)
34             {
35                 d[vecin] = d[act] + 1;
36                 Q.push(vecin);
37                 diametru = max(diametru, d[vecin]);
38             }
39         }
40     }
41 }
42
43 int main()
44 {
45     freopen("amici.in","r",stdin);
46     freopen("amici.out","w",stdout);
47
48     scanf("%d",&t);
49
50 //assert(n >= 1 && n <= 21 * 1000);
51 //assert(m >= 1 && m <= 41 * 1000);
52
53     for(;t--;)
54     {
55
56         scanf("%d %d",&n,&m);
57         diametru = 0;
58         for(i = 1; i <= n; ++i)
59             G[i].clear();
60
61         for(i = 1; i <= m; ++i)
62         {
63             scanf("%d %d",&a,&b);
64             G[a].push_back(b);
65             G[b].push_back(a);
66         }
67
68         BFS(1);
69         for(ans = 0; (1 << ans) < diametru; ++ans);
70         printf("%d\n",ans);
71     }
72
73     return 0;
74 }
```

Listing 25.1.2: amici_bogdan.cpp

```

1 // (C) 2013 Bogdan Cristian Tataroiu
2 #include <cstdio>
3 #include <cassert>
4 #include <vector>
5 #include <queue>
```

```

6
7  using namespace std;
8
9  const int MAXN = 200000;
10 const int MAXM = 200000;
11
12 int N, M;
13 vector<int> con[MAXN];
14
15 inline int bfs(int S)
16 {
17     vector<int> dist(N, -1);
18     queue<int> Q;
19     Q.push(S);
20     dist[S] = 0;
21     int MAX = 0;
22     for (; !Q.empty(); Q.pop())
23     {
24         int nod = Q.front();
25         if (dist[nod] > MAX)
26         {
27             MAX = dist[nod];
28         }
29
30         vector<int> :: iterator it;
31         for (it = con[nod].begin(); it != con[nod].end(); it++)
32         {
33             if (dist[*it] != -1)
34                 continue;
35
36             dist[*it] = dist[nod] + 1;
37             Q.push(*it);
38         }
39     }
40
41     return MAX;
42 }
43
44 int main()
45 {
46     assert(freopen("amici.in", "rt", stdin));
47     assert(freopen("amici.out", "wt", stdout));
48
49     int T;
50     assert(scanf("%d", &T) == 1);
51     for (T; T--)
52     {
53         assert(scanf("%d %d", &N, &M) == 2);
54         assert(1 <= N && N <= MAXN);
55         assert(1 <= M && M <= MAXM);
56         for (int i = 0; i < N; i++)
57             con[i].clear();
58
59         for (int i = 0; i < M; i++)
60         {
61             int x, y;
62             assert(scanf("%d %d", &x, &y) == 2);
63             assert(1 <= x && x <= N);
64             assert(1 <= y && y <= N);
65             assert(x != y);
66             x -= 1; y -= 1;
67             con[x].push_back(y);
68             con[y].push_back(x);
69         }
70
71         int len = bfs(N - 1);
72         int cnt = 0;
73         for (; (1 << cnt) < len; cnt++);
74
75         printf("%d\n", cnt);
76     }
77
78     return 0;
79 }
```

Listing 25.1.3: amici_brute.cpp

```

1 #include <cstdio>
2 #include <queue>
3 #include <vector>
4 #include <cassert>
5
6 using namespace std;
7
8 const int maxn = 200 * 1000 + 5;
9 const int INF = 2 * 1000 * 1000;
10
11 int n, m, i, j, a, b, diametru, d[maxn], t, ans;
12
13 vector<int> G[maxn];
14 queue <int> Q;
15
16 void BFS(int sursa)
17 {
18     Q.push(sursa);
19
20     for(int i = 1; i <= n; ++i)
21         d[i] = INF;
22
23     d[sursa] = 0;
24
25     for(;!Q.empty();)
26     {
27         int act = Q.front();
28         Q.pop();
29
30         for(int i = 0; i < G[act].size(); ++i)
31         {
32             int vecin = G[act][i];
33             if(d[vecin] == INF)
34             {
35                 d[vecin] = d[act] + 1;
36                 Q.push(vecin);
37             }
38         }
39     }
40 }
41
42 int main()
43 {
44
45     freopen("amici.in", "r", stdin);
46     freopen("amici.out", "w", stdout);
47
48     scanf("%d", &t);
49
50     for(;t--;}
51     {
52         scanf("%d %d", &n, &m);
53
54         assert(n >= 1 && n <= 21 * 1000);
55         assert(m >= 1 && m <= 41 * 1000);
56         for(i = 1; i <= n; ++i)
57             G[i].clear();
58
59         for(i = 1; i <= m; ++i)
60         {
61             scanf("%d %d", &a, &b);
62             assert(a != b);
63             G[a].push_back(b);
64             G[b].push_back(a);
65         }
66
67         diametru = 0;
68
69         for(i = 1; i <= n; ++i)
70         {
71             BFS(i);
72             for(j = 1; j <= n; ++j)
73                 diametru = max(diametru, d[j]);
74         }
75 }
```

```

76         for(ans = 0; (1 << ans) < diametru; ans++);
77         //fprintf(stderr,"%d\n",diametru);
78         printf("%d\n", ans);
79     }
80
81     return 0;
82 }
83 }
```

Listing 25.1.4: amici_simulare.cpp

```

1 #include <cstdio>
2 #include <queue>
3 #include <vector>
4 #include <cassert>
5
6 using namespace std;
7
8 const int maxn = 200 * 1000 + 5;
9 const int INF = 2 * 1000 * 1000;
10
11 int n, m, i, j, a, b, diametru, steps, d[maxn], t;
12
13 vector<int> G[maxn];
14 queue <int> Q;
15
16 void BFS(int sursa) {
17
18     Q.push(sursa);
19
20     for(int i = 1; i <= n; ++i)
21         d[i] = INF;
22
23     d[sursa] = 0;
24
25     for(;!Q.empty();)
26     {
27         int act = Q.front();
28         Q.pop();
29
30         for(int i = 0; i < G[act].size(); ++i)
31         {
32             int vecin = G[act][i];
33             if(d[vecin] == INF)
34             {
35                 d[vecin] = d[act] + 1;
36                 Q.push(vecin);
37             }
38         }
39     }
40 }
41
42 int main()
43 {
44
45     freopen("amici.in","r",stdin);
46     freopen("amici.out","w",stdout);
47
48     scanf("%d",&t);
49
50     for(;t--;}
51     {
52         scanf("%d %d",&n,&m);
53
54         for(i = 1; i <= n; ++i)
55             G[i].clear();
56         //assert(n >= 1 && n <= 200 * 1000);
57         //assert(m >= 1 && m <= 200 * 1000);
58
59         for(i = 1; i <= m; ++i)
60         {
61             scanf("%d %d",&a,&b);
62             //assert(a != b);
63             G[a].push_back(b);
64 }
```

```

64         G[b].push_back(a);
65     }
66
67     steps = 0;
68     bool ok = true;
69
70     while(ok)
71     {
72         ok = false;
73         for(i = 1; i <= n; ++i)
74         {
75             BFS(i);
76             for(j = 1; j <= n; ++j)
77                 if(d[j] == 2)
78                 {
79                     ok = true;
80                     G[i].push_back(j);
81                     G[j].push_back(i);
82                 }
83         }
84
85         if(ok) steps++;
86     }
87
88     printf("%d\n", steps);
89     t = 0;
90 }
91
92 return 0;
93 }
```

25.1.3 *Rezolvare detaliată

25.2 bemo

Problema 2 - bemo

100 de puncte

Se dă o matrice cu R linii și C coloane de numere distincte de la 1 la $R * C$. Bemo, personajul emoțional, dorește să urmărească cel mai bun drum din colțul superior stânga, de coordonate $(1, 1)$, în colțul inferior dreapta, de coordonate (R, C) .

Un drum este o secvență de numere din matrice în care fiecare număr se găsește în jos sau la dreapta numărului anterior, i.e. dacă (i, j) este poziția unui număr de pe un drum, atunci următorul număr poate fi cel de pe poziția $(i + 1, j)$ sau cel de pe poziția $(i, j + 1)$.

Pentru a determina dacă un drum A este mai bun decât un drum B , numerele fiecărui drum se vor sorta și se va alege cel mai mic lexicografic, e.g. $[1, 3, 5, 6, 8] < [1, 4, 5, 6, 7]$.

Cerințe

Date de intrare

Fișierul de intrare **bemo.in** conține pe prima linie două numere naturale R și C , unde R este numărul liniilor, iar C numărul coloanelor matricei lui Bemo.

Pe următoarele R linii se vor găsi câte C numere separate printr-un spațiu. Fiecare număr va fi unic și va fi cuprins în intervalul $[1, R * C]$.

Date de ieșire

Fișierul de ieșire **bemo.out** va conține $R + C - 1$ numere reprezentând cel mai bun drum pe care Bemo îl poate alege. Numerele vor fi scrise separate printr-un spațiu.

Restricții și precizări

- $0 < R, C < 1501$;
- Pentru 40% din teste: $0 < R, C < 751$;
- Pentru 70% din teste: $0 < R, C < 1301$;
- Spunem că un drum $A = (a_1, a_2, \dots, a_{R+C-1})$ este mai mic lexicografic decât un drum $B = (b_1, b_2, \dots, b_{R+C-1})$ dacă există o poziție p astfel încât $x_p < y_p$ și $x_1 = y_1, x_2 = y_2, \dots, x_{p-1} = y_{p-1}$.

Exemplu:

bemo.in	bemo.out	Explicații
4 4	7 4 11 9 1 5 6	Cel mai bun drum este secvența 7, 4, 11, 9, 1, 5, 6.
7 4 13 3		
8 11 12 2		
10 9 1 5		
16 14 15 6		

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

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

Dimensiune maximă a sursei: **10 KB**

25.2.1 Indicații de rezolvare

Marius Laurențiu Stroe, Universitatea București

Soluție 40p

Inițial, orice drum poate include orice număr din matrice. Dacă nu alegem primul drum care să îl conțină pe 1, atunci va exista un alt drum care să îl conțină. Considerăm astfel că 1 se găsește pe poziția (i_1, j_1) .

Atunci, următorul element trebuie să se găsească într-o poziție (i, j) ce respectă condițiile:

- $1 \leq i \leq i_1$ și $1 \leq j \leq j_1$;
- $i_1 \leq i \leq R$ și $j_1 \leq j \leq C$;

Această soluție folosește *metoda Divide et Impera*, iar fiecare pas constă în a găsi minimul într-un dreptunghi care să descompună problema inițială. Soluțiile vor obține 40p sau 50p, depinzând de cât de bine sunt implementate.

Complexitate: $O(R^2C^2)$

Soluție 70p

Folosind ideea anterioară, de a căuta minimul în fiecare subproblemă, observăm că drumul, după ce am găsit K astfel de numere, îndeplinește proprietatea:

- $i_1 \leq i_2 \leq \dots \leq i_K$;
- $j_1 \leq j_2 \leq \dots \leq j_K$;

Unde (i_1, j_1) este poziția primului număr etc.

Acum va trebui să introducem al K+1-lea număr dintre cele R*C.

Al K+1-lea număr trebuie să fie cel mai mic număr posibil și să se încadreze în sirul anterior. Adică, va trebui să existe un indice p astfel încât

$i_p \leq i_{K+1} \leq i_{p+1}$ și $j_p \leq j_{K+1} \leq j_{p+1}$.

Astfel, sirul anterior se va extinde.

Inițial, numerele de pe pozițiile $(1, 1)$ și (R, C) trebuie să facă parte din cel mai bun drum, conform definiției unui drum bun. Numerele vor fi considerate în ordine crescătoare, iar la fiecare pas se va căuta binar poziția sa în sirul anterior.

Complexitate: $O(RC \log(R+C))$.

Soluție 100p

La fel ca în soluția de 40p, vom împărți problema inițială în două subprobleme în funcție de minimul din dreptunghiul acelei subprobleme.

Presupunem că o subproblemă are dreptunghiul (i_l, j_l, i_r, j_r) iar minimul din acest dreptunghi se găsește pe poziția (i_m, j_m) .

Atunci, în loc să căutăm minimul în valorile de care avem nevoie, marcăm elementele de care nu avem nevoie. Astfel, vom marca toate numerele din subdreptunghiurile $(i_l, j_{m+1}, i_{m-1}, j_r)$, respectiv $(i_{m+1}, j_l, i_r, j_{m-1})$ ca fiind inutile.

Dacă vom considera toate numerele în ordine crescătoare, atunci la fiecare pas e posibil ca unul din dreptunghiuri să fie împărțit în alte două subdreptunghiuri, iar numerele nefolosite să fie marcate.

Observăm că fiecare număr va fi marcat ca nefolosit cel mult o dată. Dacă nu este marcat ca nefolosit, atunci va fi un număr din drumul bun.

Complexitate: $O(RC)$.

Soluție 100p

Ca și în soluția anterioară vom încerca să diferențiem pe parcursul soluției elementele utile și cele inutile.

Se observă ușor că la orice moment de timp, pe fiecare coloană, elementele utile reprezintă un interval compact. Menținem acest interval pentru fiecare coloană, iar pentru a verifica dacă un element este util verificăm doar dacă se încadrează în intervalul coloanei sale. După ce l-am fixat, pe toate coloanele anterioare, intervalele cu elemente utile se termină cel mult pe același rând cu elementul fixat. Iteram prin ele și marcam asta. Pe coloanele următoare intervalele cu elemente utile încep cel mai devreme pe același rând cu elementul fixat și facem la fel.

Întrucât fixăm $R + C - 1$ elemente în total complexitatea este $O(R(R + C))$.

25.2.2 Cod sursă

Listing 25.2.1: bemo_adi.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4
5 using namespace std;
6
7 int main()
8 {
9     ifstream cin("bemo.in");
10    ofstream cout("bemo.out");
11
12    int N, M; cin >> N >> M;
13
14    vector< vector<int> > A(N, vector<int>(M)), viz(N, vector<int>(M));
15    vector<int> X(N * M + 1), Y(N * M + 1);
16
17    for (int i = 0; i < N; ++i)
18        for (int j = 0; j < M; ++j)
19        {
20            int x; cin >> x;
21            A[i][j] = x;
22            X[x] = i;
23            Y[x] = j;
24        }
25
26    vector<int> from(M, 0);
27    vector<int> to(M, N - 1);
28
29    for (int i = 1; i <= N * M; ++i)
30    {
31        int row = X[i];
32        int col = Y[i];
33        if (row < from[col] || row > to[col])
34            continue;
35
36        for (int j = 0; j < col; ++j)
37            to[j] = min(to[j], row);
38        for (int j = col + 1; j < M; ++j)
39            from[j] = max(from[j], row);
40
41        viz[X[i]][Y[i]] = 1;
42    }
43
44    for (int i = 0; i < N; ++i)
45        for (int j = 0; j < M; ++j)
46            if (viz[i][j])
47                cout << A[i][j] << " ";
48
49    return 0;
50 }
```

Listing 25.2.2: bemo_mugurel_log.cpp

```
1 // Mugurel Ionut Andreica
```

```

2 // O(M * N * log(M + N))
3 // FARA citire parsata.
4
5 #include <stdio.h>
6 #include <string.h>
7 #include <stdlib.h>
8
9 #define NMAX 2011
10
11 int A[NMAX][NMAX], row[NMAX * NMAX], col[NMAX * NMAX];
12 int M, N;
13
14 void ReadInput()
15 {
16     int i, j;
17
18     freopen("bemo.in", "r", stdin);
19     scanf("%d %d", &M, &N);
20     for (i = 1; i <= M; i++)
21         for (j = 1; j <= N; j++)
22         {
23             scanf("%d", &A[i][j]);
24             row[A[i][j]] = i;
25             col[A[i][j]] = j;
26         }
27 }
28
29 char selected[NMAX * NMAX];
30 int elems[2 * NMAX], nelems;
31
32 void Solve()
33 {
34     int i, j, li, ls, mid, before;
35
36     memset(selected, 0, sizeof(selected));
37     selected[A[1][1]] = 1;
38     nelems = 1;
39     elems[1] = A[1][1];
40
41     if (M == 1 && N == 1)
42         return;
43
44     selected[A[M][N]] = 1;
45     nelems++;
46     elems[nelems] = A[M][N];
47
48     for (i = 1; i <= M * N && nelems < M + N - 1; i++)
49     {
50         if (selected[i])
51             continue;
52
53         // Check if we can add i to the path.
54         li = 1; ls = nelems; before = 0;
55
56         while (li <= ls)
57         {
58             mid = (li + ls) / 2;
59             if (row[elems[mid]] <= row[i] && col[elems[mid]] <= col[i])
60             {
61                 before = mid;
62                 li = mid + 1;
63             }
64             else
65                 ls = mid - 1;
66         }
67
68         if (before == 0)
69             continue;
70
71         if (row[i] <= row[elems[before+1]] && col[i] <= col[elems[before+1]])
72         {
73             // Insert i on position before + 1.
74             for (j = nelems; j > before; j--)
75                 elems[j + 1] = elems[j];
76             elems[before + 1] = i;
77             nelems++;
78         }
79     }
80 }

```

```

78         selected[i] = 1;
79     }
80 }
81
82 if (nelems != M + N - 1)
83 {
84     fprintf(stderr, "Not enough elements selected!\n");
85     exit(1);
86 }
87 }
88
89 void WriteOutput()
90 {
91     int i;
92
93     freopen("bemo.out", "w", stdout);
94     for (i = 1; i <= nelems; i++)
95     {
96         printf("%d", elems[i]);
97         if (i < nelems)
98             printf(" ");
99     }
100    printf("\n");
101 }
102
103 int main()
104 {
105     ReadInput();
106     Solve();
107     WriteOutput();
108     return 0;
109 }
```

Listing 25.2.3: bemo_mugurel_log_parsed.cpp

```

1 // Mugurel Ionut Andreica
2 // O(M * N * log(M + N))
3 // Citire parsata.
4 #include <stdio.h>
5 #include <string.h>
6 #include <stdlib.h>
7
8 #define NMAX 2511
9
10 int A[NMAX][NMAX], row[NMAX * NMAX], col[NMAX * NMAX];
11 int M, N;
12 char tmp[65536];
13
14 void ReadInput()
15 {
16     int i, j;
17
18     freopen("bemo.in", "r", stdin);
19     scanf("%d %d ", &M, &N);
20     for (int i = 1; i <= M; i++)
21     {
22         fgets(tmp, 65536, stdin);
23         char *p = tmp;
24         for (int j = 1; j <= N; j++)
25         {
26             int val = 0;
27             for (; '0' <= *p && *p <= '9'; p++)
28             {
29                 val = val * 10 + *p - '0';
30             }
31             for (; '0' > *p || *p > '9'; p++);
32             A[i][j] = val;
33             row[A[i][j]] = i;
34             col[A[i][j]] = j;
35         }
36     }
37 }
38
39 char selected[NMAX * NMAX];
40 int elems[2 * NMAX], nelems;
```

```

41
42 void Solve()
43 {
44     int i, j, li, ls, mid, before;
45
46     memset(selected, 0, sizeof(selected));
47     selected[A[1][1]] = 1;
48     nelems = 1;
49     elems[1] = A[1][1];
50
51     if (M == 1 && N == 1)
52         return;
53
54     selected[A[M][N]] = 1;
55     nelems++;
56     elems[nelems] = A[M][N];
57
58     for (i = 1; i <= M * N && nelems < M + N - 1; i++)
59     {
60         if (selected[i])
61             continue;
62
63         // Check if we can add i to the path.
64         li = 1; ls = nelems; before = 0;
65
66         while (li <= ls) {
67             mid = (li + ls) / 2;
68             if (row[elems[mid]] <= row[i] && col[elems[mid]] <= col[i])
69             {
70                 before = mid;
71                 li = mid + 1;
72             }
73             else
74                 ls = mid - 1;
75         }
76
77         if (before == 0)
78             continue;
79
80         if (row[i] <= row[elems[before+1]] && col[i] <= col[elems[before+1]])
81         {
82             // Insert i on position before + 1.
83             for (j = nelems; j > before; j--)
84                 elems[j + 1] = elems[j];
85             elems[before + 1] = i;
86             nelems++;
87             selected[i] = 1;
88         }
89     }
90
91     if (nelems != M + N - 1)
92     {
93         fprintf(stderr, "Not enough elements selected!\n");
94         exit(1);
95     }
96 }
97
98 void WriteOutput()
99 {
100     int i;
101
102     freopen("bemo.out", "w", stdout);
103     for (i = 1; i <= nelems; i++)
104     {
105         printf("%d", elems[i]);
106         if (i < nelems)
107             printf(" ");
108     }
109     printf("\n");
110 }
111
112 int main()
113 {
114     ReadInput();
115     Solve();
116     WriteOutput();

```

```
117     return 0;
118 }
```

Listing 25.2.4: bemo_n^2 logn.cpp

```

1 #include <cstdio>
2 #include <set>
3
4 using namespace std;
5
6 const int NMAX=2005,CHMAX=10005;
7
8 int pozX[NMAX*NMAX],pozY[NMAX*NMAX];
9 set<pair<int,int>> P;
10 int A[NMAX][NMAX];
11 int N,M;
12
13 char buff[CHMAX];
14 int poz=CHMAX-1;
15
16 inline void read (int &X)
17 {
18     while (!('0'<=buff[poz] && buff[poz]<='9'))
19         if (++poz==CHMAX)
20             {
21                 poz=0;
22                 fread (buff,1,CHMAX,stdin);
23             }
24
25     X=0;
26     while ('0'<=buff[poz] && buff[poz]<='9')
27     {
28         X=X*10+buff[poz]-'0';
29         if (++poz==CHMAX)
30             {
31                 poz=0;
32                 fread (buff,1,CHMAX,stdin);
33             }
34     }
35 }
36
37 int main ()
38 {
39     freopen ("bemo.in","r",stdin);
40     freopen ("bemo.out","w",stdout);
41
42     read (N); read (M);
43     for (int i=1; i<=N; ++i)
44         for (int j=1; j<=M; ++j)
45         {
46             read (A[i][j]);
47
48             pozX[A[i][j]]=i;
49             pozY[A[i][j]]=j;
50         }
51
52     P.insert (make_pair (1,1));
53     P.insert (make_pair (N,M));
54     for (int i=1; i<=N*M; ++i)
55         if (i!=A[1][1] && i!=A[1][1])
56         {
57             set<pair<int,int>> :: iterator it1,it2;
58             it2=P.lower_bound (make_pair (pozX[i],pozY[i]));
59             it1=--it2; it2++;
60
61             if (it1->first<=pozX[i] &&
62                 pozX[i]<it2->first &&
63                 it1->second<=pozY[i] &&
64                 pozY[i]<=it2->second)
65                 P.insert (make_pair (pozX[i],pozY[i]));
66         }
67
68     set<pair<int,int>> :: iterator it;
69     for (it=P.begin (); it!=P.end (); ++it)
70         printf ("%d ",A[it->first][it->second]);

```

```

71     return 0;
72 }

```

Listing 25.2.5: bemo-bogdan.cpp

```

1 // (C) 2013 Bogdan Cristian Tataroiu
2 #include <cstdio>
3 #include <cassert>
4
5 using namespace std;
6
7 const int MAXN = 2048;
8
9 int N, M;
10 int m[MAXN][MAXN];
11 char ok[MAXN][MAXN];
12 int px[MAXN * MAXN], py[MAXN * MAXN];
13 char tmp[65536];
14
15 int main()
16 {
17     assert(freopen("bemo.in", "rt", stdin));
18     assert(freopen("bemo.out", "wt", stdout));
19
20     assert(scanf("%d %d ", &N, &M) == 2);
21     for (int i = 0; i < N; i++)
22     {
23         fgets(tmp, 65536, stdin);
24         char *p = tmp;
25         for (int j = 0; j < M; j++)
26         {
27             int val = 0;
28             for (; '0' <= *p && *p <= '9'; p++)
29             {
30                 val = val * 10 + *p - '0';
31             }
32             for (; '0' > *p || *p > '9'; p++);
33             assert(1 <= val && val <= N * M);
34             m[i][j] = val - 1;
35
36             px[m[i][j]] = i;
37             py[m[i][j]] = j;
38             ok[i][j] = 1;
39         }
40     }
41
42     for (int i = 0; i < N * M; i++)
43     {
44         if (!ok[px[i]][py[i]])
45             continue;
46
47         for (int j = px[i] - 1; j >= 0; j--)
48         {
49             if (py[i] + 1 < M && !ok[j][py[i] + 1])
50                 break;
51
52             for (int k = py[i] + 1; k < M; k++)
53             {
54                 if (!ok[j][k])
55                     break;
56
57                 ok[j][k] = 0;
58             }
59         }
60
61         for (int j = px[i] + 1; j < N; j++)
62         {
63             if (py[i] - 1 >= 0 && !ok[j][py[i] - 1])
64                 break;
65
66             for (int k = py[i] - 1; k >= 0; k--)
67             {
68                 if (!ok[j][k])
69                     break;

```

```

70
71         ok[j][k] = 0;
72     }
73 }
74
75 for (int i = 0; i < N; i++)
76     for (int j = 0; j < M; j++)
77         if (ok[i][j])
78             printf("%d ", m[i][j] + 1);
79
80 printf("\n");
81
82 return 0;
83
84 }
```

Listing 25.2.6: bemo-N3-cristi.cpp

```

1 #include<stdio.h>
2 #include<vector>
3 #include<ctype.h>
4 #include<algorithm>
5
6 using namespace std;
7
8 FILE *f = fopen("bemo.in", "r");
9 FILE *g = fopen("bemo.out", "w");
10
11 typedef struct poz
12 {
13     int x,y;
14 } ;
15
16 typedef struct sol
17 {
18     int x,y, val;
19 } ;
20
21 typedef struct Compare
22 {
23     bool operator() (const sol a,const sol b)
24     {
25         if(a.x == b.x)
26             return a.y < b.y;
27         return a.x < b.x;
28     }
29 } ICompare;
30
31 #define MaxN 2010
32 #define MaxS (10*MaxN)
33 #define INF (1<<29)
34
35 int N,M;
36 int A[MaxN][MaxN];
37 vector<sol> SolV;
38 char S[MaxS];
39
40 inline int min(int a,int b)
41 {
42     return a < b ? a : b;
43 }
44
45 inline poz take_min(int x,int y,int z,int w)
46 {
47     poz myPoz = {0,0};
48     int myMin = INF;
49
50     for(int i=x;i<=z;i++)
51         for(int j=y;j<=w;j++)
52             if(myMin > A[i][j])
53             {
54                 myPoz.x = i;
55                 myPoz.y = j;
56                 myMin = A[i][j];
57             }
58 }
```

```

58
59     return myPoz;
60 }
61
62 inline void bemo(int x,int y,int z,int w)
63 {
64     if(x == z && w == y)
65         return ;
66
67     poz myPoz = take_min(x,y,z,w);
68
69     if(A[myPoz.x][myPoz.y] == INF)
70         return ;
71
72     sol Sol;
73     Sol.x = myPoz.x;
74     Sol.y = myPoz.y;
75     Sol.val = A[myPoz.x][myPoz.y];
76
77     SolV.push_back(Sol);
78
79     A[myPoz.x][myPoz.y] = INF;
80
81     bemo (x , y , myPoz.x , myPoz.y);
82     bemo (myPoz.x , myPoz.y , z , w);
83 }
84
85 void citire(void)
86 {
87     int poz = 0;
88     fscanf(f,"%d %d\n",&N,&M);
89     for(int i=1;i<=N;i++)
90     {
91         fgets(S,sizeof(S),f);
92         int k = 0;
93
94         for(int j=1;j<=M;j++)
95         {
96             for(;!isdigit(S[k]);k++);
97             for(;isdigit(S[k]);A[i][j] = A[i][j] * 10 + S[k++]- '0');
98         }
99     }
100 }
101
102 int main()
103 {
104     citire();
105
106     A[0][0] = INF;
107
108     bemo(1,1,N,M);
109
110     sort(Solv.begin(),Solv.end(),ICompare());
111
112     for(int i=0;i<Solv.size();i++)
113         fprintf(g,"%d ",Solv[i].val);
114 }
```

25.2.3 *Rezolvare detaliată

25.3 confuzie

Problema 3 - confuzie

100 de puncte

"Been [...] confused for so long itâŽs not true"

ZLed a devenit un pic confuz în ultima vreme, aşa că a început să se joace cu arbori (în scop terapeutic). Fiecare nod din arbore poate fi colorat fie cu alb, fie cu negru. Inițial toate nodurile sunt colorate în alb.

Pe parcursul jocului, ZLed poate alege un nod din arbore căruia să îi schimbe culoarea (din alb în negru sau din negru în alb). De asemenea, el poate selecta două noduri x și y din arbore, cu

x strămoș al lui y , și se poate întreba: "Dintre toate nodurile de pe drumul de la x la y (inclusiv x și y) care este cel mai apropiat nod față de x care este colorat în negru?".

Cerințe

Deoarece vreți să disipați starea de confuzie a lui ZLed, trebuie să îl ajutați și să-i rezolvați operațiile de colorare a unui nod, respectiv de interogare a celui mai apropiat nod de culoare neagră de pe drumul de la x la y , pentru un strămoș x al lui y .

Date de intrare

Fișierul **confuzie.in** va conține pe prima linie două numere N și M , unde N este numărul de noduri din arbore, iar M numărul de operații care se efectuează, atât modificări de culoare, cât și interogări.

Următoarele $N - 1$ linii conțin descrierea arborelui, pe fiecare linie aflându-se două numere a și b , semnificând faptul că există o muchie între a și b în arbore.

Pe următoarele M linii sunt descrise operațiile efectuate. Primul număr de pe fiecare din aceste linii reprezintă tipul operației: 0 dacă este vorba de o modificare de culoare, respectiv 1 dacă este vorba de o interogare. În primul caz, după 0 va urma un număr x , semnificând că se va schimba culoarea lui x , din negru în alb sau din alb în negru. În al doilea caz, după 1 vor urma două numere x și y , cu x strămoș al lui y , semnificând că se dorește aflarea, dintre toate nodurile de pe drumul de la x la y , a celui mai apropiat nod față de x care este colorat în negru.

Date de ieșire

Fișierul **confuzie.out** va conține câte o linie pentru fiecare operație de interogare prezentă în fișierul de intrare. Această linie poate conține fie nodul cerut (cel mai apropiat nod negru de x) de pe drumul dintre cele două noduri date la interogare, fie -1 dacă drumul dintre nodurile date la interogare nu conține niciun nod colorat cu negru.

Restricții și precizări

- Rădăcina arborelui se consideră nodul cu indice 1.
- $1 \leq N \leq 200.000$
- $1 \leq M \leq 450.000$
- $1 \leq x, y, a, b \leq N$
- Un arbore este un *graf neorientat, conex și aciclic*.
- Un nod x se numește *strămoș* al lui y dacă el se află pe drumul de la y la rădăcina arborelui.

Exemple:

confuzie.in	confuzie.out	Explicații
7 10	2-	Pe rand operațiile:
1 2	1	Setam 2 pe negru
2 4	15-	Din drumul [1, 2], 2 e negru
1 3	1	Din drumul [1, 1], nu avem nod negru
3 5	5	Setam 1 pe negru
4 6		Din drumul [1, 2], 1 și 2 negre, primul este 1
3 7		Setam 5 pe negru
0 2		Din drumul [3, 5], 5 e negru
1 1 2		Din drumul [3, 7] nu avem nod negru
1 1 1		Setam 1 pe alb
0 1		Din drumul [1, 3, 5], 5 e negru
1 1 2		
0 5		
1 3 5		
1 3 7		
0 1		
1 1 5		

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

Memorie: total **64 MB** din care pentru stivă **16 MB**

25.3.1 Indicații de rezolvare

stud. Andrei Pârvu - Universitatea Politehnica București

1. 40p - $O(N^2)$

Se face o *parcursere DFS* a arborelui, începând din rădăcină, și pentru fiecare nod se reține părintele acestuia.

Când avem de procesat un query, urcăm pe arbore din tată în tată, reținând ultimul nod colorat în negru găsit.

2. 50p - $O(N^2)$ optimizat

Soluția este asemănătoare cu cea de 40 de puncte, numai ca se face pacurgerea drumului începând cu nodul x, coborând în arbore.

Pentru a ști pe care din fii trebuie coborât, pentru fiecare nod vom reține, când facem parcurgerea DFS, timpul de intrare, respectiv timpul de ieșire dintr-un nod.

Astfel, când dorim să vedem daca un nod anume este strămoș al lui y, este suficient să verificăm daca timpul de intrare în y este cuprins între timpul de intrare și cel de ieșire al nodului pentru care facem verificarea.

Cu această tehnică, putem parcurge drumul de sus în jos, de fiecare dată ducându-ne în fiul care este strămoș al lui y, oprindu-ne atunci când gasim un nod colorat în negru (astfel nefiind nevoie să parcurgem tot drumul între x și y).

O altă soluție ce poate obține acest punctaj presupune folosirea unui *arbore echilibrat* (containerul *set* din STL), în care vom reține doar înălțimile nodurilor colorate în negru. Atunci când avem o interogare pe drumul dintre x și y, selectăm din set, folosind metodele *lower_bound* și *upper_bound*, doar intervalul cuprins între înălțimile celor două noduri. Primul nod găsit ce este strămos al lui y, va fi răspunsul la query.

3. 70p - $O(N \log^2 N)$

Fiecare nod din arbore colorat cu negru la un moment îi putem asocia valoarea 1, iar fiecare nod colorat în alb valoarea 0.

Vom nota cu $\text{suma}[x]$ suma tuturor nodurilor de pe drumul de la rădăcină la nodul x.

Pentru un drum de la x la y, răspunsul căutat va fi cel mai de sus nod pentru care $\text{suma}[\text{tata}[x]] + 1 = \text{suma}[\text{nod}]$.

Se poate observa că acest nod poate fi *căutat binar* pe intervalul de noduri dintre x și y.

Pentru a executa eficient operațiile de sumă (cu tot cu eventualele update-uri) trebuie folosit un *arbore indexat binar*, iar pentru a afla al i-lea strămoș al nodului y, trebuie menținut, pentru fiecare nod din arbore, al 2^k -lea strămoș al său.

4. 100p - $O(N \log N)$

Pentru soluția optimă, trebuie realizată o împărțire a arborelui pe drumuri, numită *heavy-path decomposition*.

Dacă ne aflăm într-un nod x, atunci path-ul în care este el prezent va continua în fiul cu subarborele cel mai greu (cu cele mai multe noduri), rămânând ca din ceilalți fi să înceapă un path nou. Astfel, se garantează că drumul de la y la x, dintr-un query, va trece prin maxim $\log N$ drumuri.

Pentru fiecare drum vom reține un *arbore echilibrat* (se poate folosi containerul *set* din STL) pentru a reține înălțimile nodurilor colorate cu negru.

Atunci când parcurgem path-urile de la y la x, pentru fiecare path, mai puțin ultimul, este suficient să ne uităm la primul element din set (cel cu înălțimea cea mai mică) și să verificăm dacă acesta se află mai sus decât nodul din care s-a intrat în path-ul curent.

Pentru ultimul path, nu mai putem să luăm elementul cu înălțimea cea mai mică, deoarece s-ar putea să fie mai sus în arbore decât x. De aceea, vom folosi metoda *lower_bound* a containerului *set* pentru a afla nodul cu înălțimea minimă mai mare sau egală decât înălțimea lui x (operația se implementează într-un *arbore echilibrat de căutare* folosind o parcursere din rădăcină).

Din toate path-urile parcurse, răspunsul va fi nodul selectat din cel mai de sus path (bineînteles, se poate ca unele pathuri să nu aibă niciun astfel de nod, fie că în path nu există niciun nod colorat cu negru, fie că cel mai de sus nod este mai jos decât punctul de intrare în path).

Pentru fiecare path, mai puțin ultimul, inspectarea minimului are complexitatea $O(1)$, iar pentru ultimul path complexitatea este $O(\log N)$.

5. 100p - ($N \log N$)

O soluție alternativă sa bazează pe o *liniarizare a arborelui*.

Dacă facem un *dfs* din rădăcina și scriem fiecare nod o dată când intrăm în el (înainte să parcurgem fiile) și o dată când ieșim din el vom obține un sir cu exact $2 * N$ elemente. Dacă un nod este selectat atunci pe prima poziție a acestui nod (să o notăm *begin[nod]*) în sub acest sir vom avea valoarea -1, iar pe a doua poziție (să o notăm *end[nod]*) vom avea valoarea -1, altfel ambele valori vor fi 0.

Acum dacă avem două noduri x și y , x stramoș al lui y , avem $\text{end}[y] < \text{end}[x]$ și totodata că suma numerelor scrise sub liniarizare pe intervalul ($\text{end}[y]$, $\text{end}[x]$), inclusive capetele, reprezintă cele noduri sunt selectate pe drumul de la x la y . Nodul pe care îl cautăm este chiar nodul scris în liniarizare pe prima poziție t ($\text{end}[y] \leq t \leq \text{end}[x]$) astfel încât suma pe intervalul ($\text{end}[y]$, t) este egală cu suma pe intervalul ($\text{end}[y], \text{end}[x]$).

Ca să găsim acest t vom ține un *arbore de intervale* pe valorile de sub liniarizare (-1, 0 sau 1), iar în fiecare nod vom tine suma numerelor din interval și *prefixul de suma maxima* din acel interval.

Update-ul doar adaugă sau scade unu într-o poziție.

Pentru query vom parcurge cele $\log N$ intervale în care se imparte intervalul de query de la stanga la dreapta și vom cauta primul interval cu proprietatea că

suma pe intervalele de dinainte + bestul din intervalul curent
da exact valoarea cautată.

Apoi vom coborî în acest interval pe aceeași idee.

Complexitatea este $O(\log N)$ pe query și $O(N)$ în memorie.

25.3.2 Cod sursă

Listing 25.3.1: confuzie-40-adrian.cpp

```

1 #include <iostream>
2 #include<vector>
3
4 using namespace std;
5
6 vector<int> v[200001];
7 int n,m,x,y,z,i,t[200001],a[100001];
8 void dfs(int);
9
10 int main()
11 {
12     freopen("confuzie.in", "r", stdin);
13     freopen("confuzie.out", "w", stdout);
14
15     scanf("%d%d", &n, &m);
16     for(i=1;i<n;i++)
17     {
18         scanf("%d%d", &x, &y);
19         v[x].push_back(y);
20         v[y].push_back(x);
21     }
22
23     t[1]=-1;
24     dfs(1);
25
26     for(i=1;i<=m;i++)
27     {
28         scanf("%d", &x);
29         if(x)
30         {
31             scanf("%d%d", &x, &y);
32             z=-1;
33             if(a[x])
34             {
35                 z=x;
36             }
37             else
38             {
39                 for(;y!=x;y=t[y])
40                     if(a[y])

```

```

41             z=y;
42         }
43         printf("%d\n",z);
44     continue;
45   }
46   scanf("%d",&x);a[x]=1-a[x];
47 }
48
49 return 0;
50 }
51
52 void dfs(int nod)
53 {
54     for(vector<int>::iterator it=v[nod].begin();it!=v[nod].end();it++)
55     if(!t[*it])
56     {
57         t[*it]=nod;
58         dfs(*it);
59     }
60 }
```

Listing 25.3.2: confuzie-40-cristi.cpp

```

1 //Brut - O(1) flip
2 //      - O(N) query
3 #include <cstdio>
4 #include <cassert>
5 #include <vector>
6
7 using namespace std;
8
9 #define MaxN 250100
10
11 int N,Q;
12 vector<int> A[MaxN];
13 vector<int> B[MaxN];
14 int T[MaxN],Val[MaxN],viz[MaxN];
15
16 void citire(void)
17 {
18     int a,b;
19
20     scanf("%d %d",&N,&Q);
21     for(int i=1;i<N;i++)
22     {
23         assert(scanf("%d %d",&a,&b) == 2);
24         assert(1 <= a && a <= N);
25         assert(1 <= b && b <= N);
26         assert(a != b);
27         A[a].push_back(b);
28         A[b].push_back(a);
29     }
30 }
31
32 void DF(int nod)
33 {
34     viz[nod] = 1;
35
36     for(int i=0;i<A[nod].size();i++)
37     if(!viz[A[nod][i]])
38     {
39         T[A[nod][i]] = nod;
40         DF(A[nod][i]);
41     }
42 }
43
44 inline void flip(int a)
45 {
46     Val[a] = 1-Val[a];
47 }
48
49 void creareArbore(void)
50 {
51     for(int i=2;i<=N;i++)
52         B[T[i]].push_back(i);
```

```

53 }
54
55 inline int first_try(int a)
56 {
57     int i;
58
59     if(Val[a] == 1)
60         return -a;
61
62     for(i=a;B[i].size() == 1;i=B[i][0])
63         if(Val[i] == 1)
64             return -i;
65
66     return i;
67 }
68
69 inline int query(int a,int b)
70 {
71     int i,answer = -1;
72
73     a = first_try(a);
74     if(a < 0)
75         return -a;
76
77     for(i = b;i != a && i != 0;i = T[i])
78         if(Val[i] == 1)
79             answer = i;
80     if(Val[a] == 1)
81         answer = a;
82
83     assert(i != 0);
84
85     return answer;
86 }
87
88 int main()
89 {
90     int op,a,b;
91
92     assert(freopen("confuzie.in", "rt", stdin));
93 #ifndef DEBUG
94     assert(freopen("confuzie.out", "wt", stdout));
95 #endif
96
97     citire();
98
99     assert(1 <= N <= 250000);
100
101    viz[1] = 1;
102    creareArbore();
103    DF(1);
104
105    for(int i=1;i<=Q;i++)
106    {
107        scanf("%d ",&op);
108        switch(op)
109        {
110            case 0 : scanf("%d ",&a);
111                assert(1 <= a && a <= N);
112                flip(a);
113                break;
114            case 1 : scanf("%d %d",&a,&b);
115                assert(1 <= a && a <= N);
116                assert(1 <= b && b <= N);
117                printf("%d\n",query(a,b));
118                break;
119        }
120    }
121
122    return 0;
123 }

```

Listing 25.3.3: confuzie-50-andrei.cpp

```
1 #include <cstdio>
```

```

2 #include <vector>
3 #include <set>
4
5 using namespace std;
6
7 #define FLIP 0
8 #define QUERY 1
9
10 #define ii pair<int, int>
11
12 const int lg = 250005;
13
14 bool fst[lg];
15 int inaltime[lg], timp[lg][2];
16 vector<int> v[lg];
17 set<ii> bigSet;
18 int ind;
19
20 void df(int nod, int adn)
21 {
22     fst[nod] = true;
23     inaltime[nod] = adn;
24
25     ind++;
26     timp[nod][0] = ind;
27
28     for (int i = 0; i < (int)v[nod].size(); i++)
29         if (!fst[v[nod][i]])
30             df(v[nod][i], adn + 1);
31
32     timp[nod][1] = ind;
33 }
34
35
36 void update(int nod)
37 {
38     int h = inaltime[nod];
39
40     if (bigSet.find(make_pair(h, nod)) != bigSet.end())
41         bigSet.erase(make_pair(h, nod));
42     else
43         bigSet.insert(make_pair(h, nod));
44 }
45
46 int query(int x, int y)
47 {
48     set<ii> :: iterator start = bigSet.lower_bound(make_pair(inaltime[x], x)),
49         end = bigSet.upper_bound(make_pair(inaltime[y], y)), it;
50
51     for (it = start; it != end; it++)
52     {
53         int nodCur = it->second;
54
55         if (timp[x][0] <= timp[nodCur][0] && timp[nodCur][1] <= timp[x][1] &&
56             timp[nodCur][0] <= timp[y][0] && timp[y][1] <= timp[nodCur][1])
57         {
58             return nodCur;
59         }
60     }
61
62     return -1;
63 }
64
65 int main()
66 {
67     freopen("confuzie.in", "rt", stdin);
68     freopen("confuzie.out", "wt", stdout);
69
70     int n, m, x, y;
71
72     scanf("%d%d", &n, &m);
73     for (int i = 1; i < n; i++)
74     {
75         scanf("%d%d", &x, &y);
76         v[x].push_back(y);
77         v[y].push_back(x);

```

```

78     }
79
80     df(1, 1);
81
82     for (int i = 0; i < m; i++)
83     {
84         int tip, nod;
85
86         scanf("%d", &tip);
87
88         if (tip == FLIP)
89         {
90             scanf("%d", &nod);
91
92             update(nod);
93         }
94         else
95         {
96             scanf("%d%d", &x, &y);
97             printf("%d\n", query(x, y));
98         }
99     }
100
101    return 0;
102 }
```

Listing 25.3.4: confuzie-50-mugurel.cpp

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <time.h>
4 #include <vector>
5 #include <stdlib.h>
6
7 using namespace std;
8
9 #define NMAX 200111
10#define DEBUG 0
11#define FILEIN "confuzie.in"
12#define FILEOUT "confuzie.out"
13
14 vector<int> vec[NMAX];
15 int N, M;
16
17 void ReadInput()
18 {
19     int i, j, k;
20
21     freopen(FILEIN, "r", stdin);
22     scanf("%d %d", &N, &M);
23
24     for (k = 1; k < N; k++)
25     {
26         scanf("%d %d", &i, &j);
27         vec[i].push_back(j);
28         vec[j].push_back(i);
29     }
30 }
31
32 char visited[NMAX];
33 int stk[NMAX], stkidx[NMAX], parent[NMAX], idxmin[NMAX],
34     idxmax[NMAX], q[NMAX], level[NMAX], idx, niv;
35 vector<int> sons[NMAX];
36
37 void DFS()
38 {
39     int i, j;
40
41     memset(visited, 0, sizeof(visited));
42     visited[1] = 1;
43     parent[1] = 1;
44
45     stk[niv = 1] = 1;
46     stkidx[1] = -1;
47     idx = 1;
```

```

48     idxmin[1] = 1;
49     level[1] = 1;
50     q[1] = 1;
51
52     while (niv >= 1)
53     {
54         stkidx[niv]++;
55         i = stk[niv];
56
57         if (stkidx[niv] >= vec[i].size())
58         {
59             idxmax[i] = idx;
60             niv--;
61         }
62         else
63         {
64             j = vec[i][stkidx[niv]];
65             if (!visited[j])
66             {
67                 visited[j] = 1;
68                 parent[j] = i;
69                 sons[i].push_back(j);
70                 level[j] = level[i] + 1;
71                 idx++;
72                 idxmin[j] = idx;
73                 q[idx] = j;
74                 niv++;
75                 stk[niv] = j;
76                 stkidx[niv] = -1;
77             }
78         }
79     }
80
81     fprintf(stderr, "idx=%d N=%d\n", idx, N);
82     if (idx != N)
83     {
84         fprintf(stderr, "Bad test! The nodes do not form a tree!\n");
85         exit(1);
86     }
87 }
88
89 int color[NMAX];
90
91 void ProcessQueries()
92 {
93     int op, x, y, i, z, answer, qcnt = 0;
94
95     freopen(FILEOUT, "w", stdout);
96
97     if (DEBUG)
98         freopen("log.txt", "w", stderr);
99
100    memset(color, 0, sizeof(color));
101
102    while (M--)
103    {
104        scanf("%d %d", &op, &x);
105        if (op == 0)
106        {
107            color[x] = 1 - color[x];
108        }
109        else
110        {
111            scanf("%d", &y);
112            answer = -1;
113
114            if (DEBUG)
115            {
116                qcnt++;
117                fprintf(stderr, "QUERY %d: x=%d (%d) y=%d (%d)\n",
118                        qcnt, x, level[x], y, level[y]);
119            }
120
121            if (color[x])
122                answer = x;
123            else

```

```

124         {
125             while (x != y)
126             {
127                 if (DEBUG)
128                     fprintf(stderr, " x=%d (%d)\n", x, level[x]);
129
130                 for (i = 0; i < sons[x].size(); i++)
131                 {
132                     z = sons[x][i];
133                     if (idxmin[z] <= idxmin[y] && idxmax[y] <= idxmax[z])
134                     {
135                         x = z;
136                         break;
137                     }
138                 }
139
140                 if (color[x])
141                 {
142                     answer = x;
143                     break;
144                 }
145             }
146
147             if (DEBUG)
148                 fprintf(stderr, " final x=%d (%d %d)\n",
149                         x, level[x], color[x]);
150         }
151
152         printf("%d\n", answer);
153     }
154 }
155 }
156
157 int main()
158 {
159     int tstart = clock();
160
161     ReadInput();
162     DFS();
163     ProcessQueries();
164
165     fprintf(stderr, "Duration=%.3lf sec\n",
166             (double) (clock() - tstart) / CLOCKS_PER_SEC);
167
168     return 0;
169 }
```

Listing 25.3.5: confuzie-70-andrei.cpp

```

1 #include <cstdio>
2 #include <vector>
3 #include <set>
4 #include <assert.h>
5
6 using namespace std;
7
8 #define FLIP 0
9 #define QUERY 1
10
11 #define ii pair<int, int>
12 #define NMAX 250000
13 #define MMAX 450001
14
15 const int lg = 250005;
16
17 bool fst[lg];
18 int inaltime[lg];
19 int tata[20][lg];
20
21 int aib[2 * lg];
22
23 vector<int> v[lg];
24 int ind, timp[lg][2];
25
26 bool isSet[lg];
27
```

```

28 void df(int nod, int adn)
29 {
30     fst[nod] = true;
31     inaltime[nod] = adn;
32
33     timp[nod][0] = ++ind;
34
35     for (int i = 0; i < (int)v[nod].size(); i++)
36         if (!fst[v[nod][i]])
37         {
38             tata[0][v[nod][i]] = nod;
39             df(v[nod][i], adn + 1);
40         }
41
42     timp[nod][1] = ++ind;
43 }
44
45 void aibUpdate(int poz, int val)
46 {
47     for (int i = poz; i <= ind; i += (i ^ (i - 1)) & i)
48         aib[i] += val;
49 }
50
51 int aibQuery(int poz)
52 {
53     int rez = 0;
54
55     for (int i = poz; i > 0; i -= (i ^ (i - 1)) & i)
56         rez += aib[i];
57
58     return rez;
59 }
60
61 void update(int nod)
62 {
63     int s;
64
65     if (isSet[nod] == false)
66     {
67         isSet[nod] = true;
68         s = 1;
69     }
70     else
71     {
72         isSet[nod] = false;
73         s = -1;
74     }
75
76     aibUpdate(timp[nod][0], s);
77     aibUpdate(timp[nod][1], -s);
78 }
79
80 int find(int nod, int k)
81 {
82     if (k == 0)
83         return nod;
84
85     for (int i = 18; i >= 0; i--)
86         if ((k & (1 << i)) > 0)
87             nod = tata[i][nod];
88
89     return nod;
90 }
91
92 int query(int x, int y)
93 {
94     int suma, rsp = -1;
95
96     if (x == 1)
97         suma = 0;
98     else
99         suma = aibQuery(timp[tata[0][x]][0]);
100
101    int li = 0, ls = inaltime[y] - inaltime[x];
102
103    while (li <= ls)

```

```

104     {
105         int cur = (li + ls) / 2;
106         stramos = find(y, cur);
107
108         if (aibQuery(timp[stramos][0]) - suma == 0)
109             ls = cur - 1;
110         else
111         {
112             li = cur + 1;
113             rsp = stramos;
114         }
115     }
116
117     return rsp;
118 }
119
120 int main()
121 {
122     freopen("confuzie.in", "rt", stdin);
123     freopen("confuzie.out", "wt", stdout);
124
125     int n, m, x, Y;
126
127     assert(scanf("%d%d", &n, &m) == 2);
128
129     assert(1 <= n && n <= NMAX);
130     assert(1 <= m && m <= MMAX);
131
132     for (int i = 1; i < n; i++)
133     {
134         assert(scanf("%d%d", &x, &y) == 2);
135         assert(1 <= x && x <= n);
136         assert(1 <= y && y <= n);
137
138         v[x].push_back(y);
139         v[y].push_back(x);
140     }
141
142     df(1, 1);
143
144     for (int i = 1; i <= 19; i++)
145         for (int j = 1; j <= n; j++)
146             tata[i][j] = tata[i - 1][tata[i - 1][j]];
147
148     for (int i = 0; i < m; i++)
149     {
150         int tip, nod;
151
152         assert(scanf("%d", &tip) == 1);
153         assert(tip == FLIP || tip == QUERY);
154
155         if (tip == FLIP)
156         {
157             assert(scanf("%d", &nod) == 1);
158             assert(1 <= nod && nod <= n);
159
160             update(nod);
161         }
162         else
163         {
164             assert(scanf("%d%d", &x, &y) == 2);
165
166             assert(1 <= x && x <= n);
167             assert(1 <= y && y <= n);
168
169             assert(timp[x][0] <= timp[y][0] && timp[y][1] <= timp[x][1]);
170
171             printf("%d\n", query(x, y));
172         }
173     }
174
175     return 0;
176 }

```

Listing 25.3.6: confuzie-70-dragos.cpp

```

1 #include <cstdio>
2 #include <vector>
3 #include <set>
4 #include <assert.h>
5
6 using namespace std;
7
8 #define FLIP 0
9 #define QUERY 1
10
11 #define ii pair<int, int>
12 #define NMAX 250000
13 #define MMAX 450001
14
15 const int lg = 250005;
16
17 bool fst[lg];
18 int inaltime[lg];
19 int tata[20][lg];
20
21 int aib[2 * lg];
22
23 vector<int> v[lg];
24 int ind, timp[lg][2];
25
26 bool isSet[lg];
27
28 void df(int nod, int adn)
29 {
30     fst[nod] = true;
31     inaltime[nod] = adn;
32
33     timp[nod][0] = ++ind;
34
35     for (int i = 0; i < (int)v[nod].size(); i++)
36         if (!fst[v[nod][i]])
37         {
38             tata[0][v[nod][i]] = nod;
39             df(v[nod][i], adn + 1);
40         }
41
42     timp[nod][1] = ++ind;
43 }
44
45 void aibUpdate(int poz, int val)
46 {
47     for (int i = poz; i <= ind; i += (i ^ (i - 1)) & i)
48         aib[i] += val;
49 }
50
51 int aibQuery(int poz)
52 {
53     int rez = 0;
54
55     for (int i = poz; i > 0; i -= (i ^ (i - 1)) & i)
56         rez += aib[i];
57
58     return rez;
59 }
60
61 void update(int nod)
62 {
63     int s;
64
65     if (isSet[nod] == false)
66     {
67         isSet[nod] = true;
68         s = 1;
69     }
70     else
71     {
72         isSet[nod] = false;
73         s = -1;
74     }
75 }
```

```

76     aibUpdate(timp[nod][0], s);
77     aibUpdate(timp[nod][1], -s);
78 }
79
80 int find(int nod, int k)
81 {
82     if (k == 0)
83         return nod;
84
85     for (int i = 18; i >= 0; i--)
86         if ((k & (1 << i)) > 0)
87             nod = tata[i][nod];
88
89     return nod;
90 }
91
92 int query(int x, int y)
93 {
94     int suma, rsp = -1;
95
96     if (x == 1)
97         suma = 0;
98     else
99         suma = aibQuery(timp[x][0] - 1);
100
101    if( aibQuery( timp[y][0] ) - aibQuery(timp[x][0] - 1) == 0 )
102        return -1;
103
104    int li = 0, ls = inaltime[y] - inaltime[x];
105
106    int stramos = y;
107
108    for (int i = 18; i >= 0; --i)
109        if(aibQuery( timp[ tata[i][stramos] ][0] ) - suma > 0 )
110            stramos = tata[i][stramos];
111
112    return stramos;
113 }
114
115 int main()
116 {
117     freopen("confuzie.in", "rt", stdin);
118     freopen("confuzie.out", "wt", stdout);
119
120     int n, m, x, y;
121
122     assert(scanf("%d%d", &n, &m) == 2);
123
124     assert(1 <= n && n <= NMAX);
125     assert(1 <= m && m <= MMAX);
126
127     for (int i = 1; i < n; i++)
128     {
129         assert(scanf("%d%d", &x, &y) == 2);
130         assert(1 <= x && x <= n);
131         assert(1 <= y && y <= n);
132
133         v[x].push_back(y);
134         v[y].push_back(x);
135     }
136
137     df(1, 1);
138
139     for (int i = 1; i <= 19; i++)
140         for (int j = 1; j <= n; j++)
141             tata[i][j] = tata[i - 1][tata[i - 1][j]];
142
143     for (int i = 0; i < m; i++)
144     {
145         int tip, nod;
146
147         assert(scanf("%d", &tip) == 1);
148         assert(tip == FLIP || tip == QUERY);
149
150         if (tip == FLIP)
151     {

```

```

152     assert(scanf("%d", &nod) == 1);
153     assert(1 <= nod && nod <= n);
154
155     update(nod);
156 }
157 else
158 {
159     assert(scanf("%d%d", &x, &y) == 2);
160     assert(1 <= x && x <= n);
161     assert(1 <= y && y <= n);
162     assert(timp[x][0] <= timp[y][0] && timp[y][1] <= timp[x][1]);
163     printf("%d\n", query(x, y));
164 }
165 }
166
167 return 0;
168 }
```

Listing 25.3.7: confuzie-100-adi.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4
5 using namespace std;
6
7 const int inf = 0x3f3f3f3f;
8
9 class Graph
10 {
11 public:
12     Graph(const int &size = 1): edges_(size) { }
13
14     void addEdge(const int &firstNode, const int &secondNode)
15     {
16         edges_[firstNode].push_back(secondNode);
17         edges_[secondNode].push_back(firstNode);
18     }
19
20     void liniarize()
21     {
22         begin_ = end_ = vector<int>(edges_.size());
23         liniarize_ = vector<int>(edges_.size() * 2);
24         time_ = 0;
25
26         dfs(0);
27     }
28
29     const int & start(const int &node) const
30     {
31         return begin_[node];
32     }
33
34     const int & end(const int &node) const
35     {
36         return end_[node];
37     }
38
39     int get(const int &position) const
40     {
41         if (position == -inf)
42             return -2;
43         return liniarize_[position];
44     }
45
46 private:
47     void dfs(const int &node, const int &father = -1)
48     {
49         begin_[node] = time_;
50         liniarize_[time_++] = node;
51
52         for (vector<int>::iterator it = edges_[node].begin();
53              it != edges_[node].end(); ++it)
54             if (*it != father)
55                 dfs(*it, node);
```

```

56         end_[node] = time_;
57         liniarize_[time_++] = node;
58     }
59 }
60
61     vector< vector<int> > edges_;
62
63     int time_;
64     vector<int> liniarize_, begin_, end_;
65 };
66
67 class SegmentTree
68 {
69     public:
70     SegmentTree(const int &size_)
71     {
72         for (size = 1; size < size_; size *= 2);
73
74         sum_ = best_ = vector<int>(size * 2, 0);
75     }
76
77     void add(int position, const int &value)
78     {
79         position += size;
80         best_[position] = (sum_[position] += value);
81
82         for (position /= 2; position; position /= 2)
83         {
84             sum_[position] += value;
85             best_[position] = max(best_[position * 2],
86                                   sum_[position*2] + best_[position*2+1]);
87         }
88
89         //add(1, 0, size, position, value);
90     }
91
92     int query(const int &start, const int &finish)
93     {
94         int value = sum(start, finish);
95         if (value == 0)
96             return -inf;
97         tempsum = 0;
98         return query(start, finish, value);
99         return query(1, 0, size, start, finish, value);
100    }
101
102    private:
103    void add(const int &node, const int &begin, const int &end,
104             const int &position, const int &value)
105    {
106        if (end - begin == 1)
107        {
108            sum_[node] += value;
109            best_[node] = sum_[node];
110            return;
111        }
112
113        int mid = (begin + end) / 2;
114        if (position < mid)
115            add(node * 2, begin, mid, position, value);
116        else
117            add(node * 2 + 1, mid, end, position, value);
118
119        sum_[node] += value;
120        best_[node] = max(best_[node*2], sum_[node*2] + best_[node*2+1]);
121    }
122
123    int sum(int from, int to)
124    {
125        int answer = 0;
126        from += size;
127        to += size;
128        while (from <= to)
129        {
130            if (from % 2)
131                answer += sum_[from];

```

```

132         if (to % 2 == 0)
133             answer += sum_[to];
134
135         from = (from + 1) / 2;
136         to = (to - 1) / 2;
137     }
138
139     return answer;
140 }
141
142 int find(int node, int value)
143 {
144     for (; node < size;)
145     {
146         if (best_[node * 2] >= value)
147             node = node * 2;
148         else
149         {
150             value -= sum_[node * 2];
151             node = node * 2 + 1;
152         }
153     }
154
155     return node - size;
156 }
157
158 int query(int from, int to, int value)
159 {
160     int newFrom = from + size;
161     int newTo = to + size;
162
163     int dir = 0;
164     while (newFrom <= newTo)
165     {
166         if (newFrom % 2)
167         {
168             if (best_[newFrom] >= value)
169                 return find(newFrom, value);
170             value -= sum_[newFrom];
171         }
172
173         dir = dir * 2 + newTo % 2;
174         newFrom = (newFrom + 1) / 2;
175         newTo = (newTo - 1) / 2;
176     }
177
178     do
179     {
180         newTo = newTo * 2 + 2 - dir % 2;
181
182         if (dir % 2 == 0)
183         {
184             if (best_[newTo] >= value)
185                 return find(newTo, value);
186             value -= sum_[newTo];
187         }
188
189         dir /= 2;
190     } while (newTo != to);
191
192     return -inf;
193 }
194
195 int query(const int &node, const int &begin, const int &end,
196           const int &start, const int &finish, const int &value)
197 {
198     if (start <= begin && end - 1 <= finish)
199     {
200         if (best_[node] + tempsum < value)
201         {
202             tempsum += sum_[node];
203             return -inf;
204         }
205
206         if (end - begin == 1)
207         {

```

```

208         tempsum += sum_[node];
209     }
210 }
211 }
212
213 int mid = (begin + end) / 2;
214
215 if (start >= mid)
216     return query(node * 2 + 1, mid, end, start, finish, value);
217 if (finish < mid)
218     return query(node * 2, begin, mid, start, finish, value);
219
220 // if it's on the left size
221 int answer = query(node * 2, begin, mid, start, finish, value);
222 if (answer != -inf)
223     return answer;
224 // else
225 answer = query(node * 2 + 1, mid, end, start, finish, value);
226
227 if (start <= begin && end - 1 <= finish)
228     tempsum += sum_[node];
229
230 return answer;
231 }
232
233 int size;
234 int tempsum;
235 vector<int> sum_, best_;
236 };
237
238 int main()
239 {
240     ifstream cin("confuzie.in");
241     ofstream cout("confuzie.out");
242
243     int N, M; cin >> N >> M;
244
245     Graph G(N);
246     for (int i = 1; i < N; ++i)
247     {
248         int x, y; cin >> x >> y;
249         G.addEdge(x - 1, y - 1);
250     }
251
252     G.liniarize();
253
254     SegmentTree S(2 * N);
255     vector<bool> value(N, false);
256
257     for (int i = 0; i < M; ++i)
258     {
259         int type, node;
260         cin >> type >> node;
261
262         if (type == 0)
263         {
264             --node;
265             value[node] = !value[node];
266             if (value[node])
267             {
268                 S.add(G.start(node), -1);
269                 S.add(G.end(node), 1);
270             }
271             else
272             {
273                 S.add(G.start(node), 1);
274                 S.add(G.end(node), -1);
275             }
276         }
277         else
278         {
279             int second; cin >> second;
280             cout << G.get(S.query(G.end(second - 1), G.end(node - 1))) + 1 << endl;
281         }
282     }

```

Listing 25.3.8: confuzie-100-alex.cpp

```

1 #include<cstdio>
2 #include<cstring>
3 #include<vector>
4 #include<algorithm>
5 #include<set>
6
7 using namespace std;
8
9 #define NMAX 250010
10
11 int N, Q;
12 int value[NMAX];
13 int used[NMAX];
14 int father[NMAX];
15
16 int num_paths;
17 int heavy[NMAX];
18 int depth[NMAX];
19 int how_many[NMAX];
20
21 int where[NMAX];
22 int position[NMAX];
23 int first_node[NMAX];
24 int prevv[NMAX];
25 int is_free[NMAX];
26
27 set< pair<int, int> > Set[NMAX];
28
29 vector<int> tree[NMAX];
30
31 void dfs(int node, int d)
32 {
33
34     used[node] = 1; heavy[node] = 1; depth[node] = d;
35     for(vector<int>::iterator it = tree[node].begin();
36         it != tree[node].end(); it++)
37     {
38         if(!used[*it])
39         {
40             dfs( *it, d + 1);
41             heavy[node]+= heavy[*it];
42             father[*it] = node;
43         }
44     }
45     void heavy_path(int node, int path)
46     {
47
48         used[node] = 1;
49         where[node] = path;
50         position[node] = ++how_many[path];
51
52         if(position[node] == 1)
53             first_node[path] = node;
54
55         for(vector<int>::iterator it = tree[node].begin();
56             it != tree[node].end(); it++)
57         {
58             if(!used[*it])
59             {
60                 if(is_free[node] == 0)
61                 {
62                     is_free[node] = 1;
63                     heavy_path( *it, path);
64                 }
65                 else
66                 {
67                     ++num_paths;
68                     prevv[num_paths] = node;
69                     heavy_path( *it, num_paths);
70                 }
71             }
72         }
73     void update(int node)
74     {

```

```

75     value[node] = (!value[node]);
76     if(value[node])
77         Set[ where[node] ].insert( make_pair(depth[node], node) );
78     else
79     {
80         set< pair<int, int> >::iterator
81         it = Set[ where[node] ].find( make_pair( depth[node], node ) );
82
83         Set[ where[node] ].erase(it);
84     }
85 }
86
87 int query(int x, int y)
88 {
89
90     int current_path = where[y];
91 //    printf ("!%d\n", current_path);
92
93     int result = -1;
94     if (current_path != where[x])
95         result = query (x, prevv[current_path]);
96
97     if (result != -1)
98         return result;
99
100    set < pair <int, int> >::iterator it, it2;
101    if (current_path == where[x])
102    {
103        it = Set[current_path].lower_bound( make_pair( depth[x], x ) );
104        if (it == Set[current_path].end())
105            return -1;
106
107        int node = it->second;
108
109        if( depth[node] > depth[y] )
110            return -1;
111
112        return node;
113    }
114
115    it = Set[current_path].begin();
116    if (it == Set[current_path].end())
117        return -1;
118    if (depth[it->second] > depth[y])
119        return -1;
120
121    return it->second;
122 }
123
124 bool cmp(int x, int y)
125 {
126     return heavy[x] > heavy[y];
127 }
128
129 void read_tree()
130 {
131     int x, y;
132
133     scanf("%d %d", &N, &Q);
134     for(int i = 1; i < N; i++)
135     {
136         scanf("%d %d", &x, &y);
137         tree[x].push_back(y);
138         tree[y].push_back(x);
139     }
140
141     dfs(1, 1);
142     for(int i = 1; i <= N; i++)
143         sort(tree[i].begin(), tree[i].end(), cmp);
144
145     memset(used, 0, sizeof(used));
146
147     num_paths = 1;
148     heavy_path(1, 1);
149 //    printf("paths : %d\n", num_paths);
150 }
```

```

151
152 int main()
153 {
154
155     freopen("confuzie.in", "r", stdin);
156     freopen("confuzie.out", "w", stdout);
157
158     read_tree();
159
160     int type, x, y;
161     for(; Q; Q--)
162     {
163         scanf("%d", &type);
164         if (type == 0)
165         {
166             scanf("%d", &x);
167             update(x);
168         }
169         else
170         {
171             scanf("%d %d", &x, &y);
172             // printf("%d %d : ", x, y);
173             printf("%d\n", query(x, y));
174             // printf("END\n");
175         }
176     }
177
178     return 0;
179 }
```

Listing 25.3.9: confuzie-100-andrei.cpp

```

1 /* Andrei Parvu
2      O(logN) / query, smen heavy path
3 */
4 #include <cstdio>
5 #include <vector>
6 #include <set>
7 #include <assert.h>
8
9 using namespace std;
10
11 #define FLIP 0
12 #define QUERY 1
13
14 #define ii pair<int, int>
15 #define NMAX 250000
16 #define MMAX 450001
17
18 const int lg = 250005;
19
20 bool fst[lg];
21 int inaltime[lg], whichPath[lg], father[lg], D[lg], Max[lg];
22 int nr_paths;
23 vector<int> v[lg];
24 set<ii> sets[lg];
25 ii minElements[lg];
26 int ind, timp[lg][2];
27
28 void df(int nod, int adn)
29 {
30     fst[nod] = true;
31     inaltime[nod] = adn;
32
33     ind++;
34     timp[nod][0] = ind;
35
36     for (int i = 0; i < (int)v[nod].size(); i++)
37         if (!fst[v[nod][i]])
38         {
39             df(v[nod][i], adn + 1);
40             D[nod] += D[v[nod][i]];
41
42             if (D[v[nod][i]] > D[Max[nod]])
43                 Max[nod] = v[nod][i];
44
45         }
46 }
```

```

44      }
45
46      D[nod]++;
47
48      timp[nod][1] = ind;
49  }
50
51 void setPaths(int nod, int path)
52 {
53     fst[nod] = false;
54
55     whichPath[nod] = path;
56
57     if (Max[nod] != 0)
58         setPaths(Max[nod], path);
59
60     for (int i = 0; i < (int)v[nod].size(); i++)
61         if (fst[v[nod][i]] == true && v[nod][i] != Max[nod])
62         {
63             nr_paths++;
64             father[nr_paths] = nod;
65             setPaths(v[nod][i], nr_paths);
66         }
67     }
68
69 void update(int nod)
70 {
71     int h = inaltime[nod], path = whichPath[nod];
72
73     if (sets[path].find(make_pair(h, nod)) != sets[path].end())
74         sets[path].erase(make_pair(h, nod));
75     else
76         sets[path].insert(make_pair(h, nod));
77
78     if (!sets[path].empty())
79         minElements[path] = *sets[path].begin();
80     else
81         minElements[path] = make_pair(-1, 0);
82 }
83
84 int query(int x, int y)
85 {
86     int startPath = whichPath[y], endPath = whichPath[x], rsp = -1;
87
88     for (; startPath != endPath; )
89     {
90         ii cur = minElements[startPath];
91         if (cur.first != -1 && cur.first <= inaltime[y])
92             rsp = cur.second;
93
94         y = father[startPath];
95         startPath = whichPath[y];
96     }
97
98     set<ii> :: iterator it=sets[endPath].lower_bound(make_pair(inaltime[x],x));
99
100    if (it != sets[endPath].end())
101        if (it->first <= inaltime[y])
102            rsp = it->second;
103
104    return rsp;
105 }
106
107 int main()
108 {
109     freopen("confuzie.in", "rt", stdin);
110     freopen("confuzie.out", "wt", stdout);
111
112     int n, m, x, y;
113
114     assert(scanf("%d%d", &n, &m) == 2);
115
116     assert(1 <= n && n <= NMAX);
117     assert(1 <= m && m <= MMAX);
118
119     for (int i = 1; i < n; i++)

```

```

120    {
121        assert(scanf("%d%d", &x, &y) == 2);
122        assert(1 <= x && x <= n);
123        assert(1 <= y && y <= n);
124
125        v[x].push_back(y);
126        v[y].push_back(x);
127    }
128
129    nr_paths = 1;
130
131    df(1, 1);
132
133    setPaths(1, 1);
134
135    for (int i = 1; i <= nr_paths; i++)
136        minElements[i] = make_pair(-1, 0);
137
138    for (int i = 0; i < m; i++)
139    {
140        int tip, nod;
141
142        assert(scanf("%d", &tip) == 1);
143        assert(tip == FLIP || tip == QUERY);
144
145        if (tip == FLIP)
146        {
147            assert(scanf("%d", &nod) == 1);
148            assert(1 <= nod && nod <= n);
149
150            update(nod);
151        }
152        else
153        {
154            assert(scanf("%d%d", &x, &y) == 2);
155            assert(1 <= x && x <= n);
156            assert(1 <= y && y <= n);
157            assert(timp[x][0] <= timp[y][0] && timp[y][1] <= timp[x][1]);
158            printf("%d\n", query(x, y));
159        }
160    }
161
162    return 0;
163}

```

Listing 25.3.10: confuzie-100-bogdan.cpp

```

1 // (C) 2013 Bogdan Cristian Tataroiu
2 #include <cstdio>
3 #include <cassert>
4 #include <vector>
5 #include <set>
6
7 using namespace std;
8
9 const int MAXN = 200000;
10 const int MAXM = 450001;
11
12 int N, M;
13
14 vector<int> con[MAXN];
15 int cnt[MAXN], child[MAXN], lvl[MAXN], p[MAXN];
16 int path[MAXN], flipped[MAXN];
17
18 int Pid = -1;
19 vector<vector<int> > paths;
20 vector<set<pair<int, int> > > sets;
21
22 void dfs(int k)
23 {
24     vector<int> :: iterator it;
25     cnt[k] = 1; child[k] = -1;
26     for (it = con[k].begin(); it != con[k].end(); it++)
27     {
28         if (*it == p[k])

```

```

29         continue;
30         lvl[*it] = lvl[k] + 1;
31         p[*it] = k;
32         dfs( *it);
33         cnt[k] += cnt[*it];
34         if (child[k] == -1 || cnt[*it] > cnt[child[k]])
35             child[k] = *it;
36     }
37
38     if (child[k] == -1)
39     {
40         Pid += 1;
41         path[k] = Pid;
42         paths.push_back(vector<int>(1, k));
43         sets.push_back(set<pair<int, int>>());
44     }
45     else
46     {
47         path[k] = path[child[k]];
48         paths[path[k]].push_back(k);
49     }
50 }
51
52 int main()
53 {
54     assert(freopen("confuzie.in", "rt", stdin));
55     assert(freopen("confuzie.out", "wt", stdout));
56
57     assert(scanf("%d %d", &N, &M) == 2);
58     for (int i = 1; i < N; i++)
59     {
60         int x, y;
61         assert(scanf("%d %d", &x, &y) == 2);
62         assert(1 <= x && x <= N);
63         assert(1 <= y && y <= N);
64         assert(x != y);
65         x -= 1; y -= 1;
66         con[x].push_back(y);
67         con[y].push_back(x);
68     }
69
70     p[0] = -1;
71     dfs(0);
72
73     for (int i = 0; i < M; i++)
74     {
75         int type;
76         assert(scanf("%d", &type) == 1);
77         assert(type == 0 || type == 1);
78         if (type == 0)
79         {
80             int nod;
81             assert(scanf("%d", &nod) == 1);
82             assert(1 <= nod && nod <= N);
83             nod -= 1;
84             if (flipped[nod])
85             {
86                 flipped[nod] = 0;
87                 sets[path[nod]].erase(make_pair(lvl[nod], nod));
88             }
89             else
90             {
91                 flipped[nod] = 1;
92                 sets[path[nod]].insert(make_pair(lvl[nod], nod));
93             }
94         }
95         else
96             if (type == 1)
97             {
98                 int x, y;
99                 assert(scanf("%d %d", &x, &y) == 2);
100                assert(1 <= x && x <= N);
101                assert(1 <= y && y <= N);
102                x -= 1;
103                y -= 1;
104                assert(lvl[x] <= lvl[y]);

```

```

105     int rez = -2;
106     for (; path[y] != path[x]; y = p[paths[path[y]].back()])
107     {
108         if (!sets[path[y]].empty() &&
109             lvl[sets[path[y]].begin() ->second] <= lvl[y])
110         {
111             rez = sets[path[y]].begin() ->second;
112         }
113     }
114
115     assert(path[y] == path[x]);
116     set<pair<int, int> :: iterator it;
117     it = sets[path[y]].upper_bound(make_pair(lvl[x] - 1, 0x3f3f3f3f));
118     if (it != sets[path[y]].end() && lvl[it->second] <= lvl[y])
119     {
120         rez = it->second;
121     }
122
123     printf("%d\n", rez + 1);
124 }
125
126 return 0;
127 }
```

25.3.3 *Rezolvare detaliată

25.4 ausoara

Problema 4 - ausoara

100 de puncte

Dorind să se angajeze, Arius M. a fost nevoie să dea un interviu în care a primit următoarea problemă simplă: dându-se N siruri crescătoare de numere întregi, să se determine cel mai lung subșir comun al acestora.

Cerințe

Rezolvați această problemă pe care Arius M. a considerat-o destul de ușoară.

Date de intrare

Pe prima linie a fișierului **ausoara.in** se află N , numărul sirurilor. Următoarele N linii descriu cele N siruri. Linia i este formată din M_i , numărul elementelor sirului curent, urmat de M_i numere, reprezentând elementele sirului i .

Date de ieșire

Fișierul de ieșire **ausoara.out** va conține pe prima linie T , numărul elementelor celui mai lung subșir comun al celor N siruri. Urmează T numere întregi ce descriu elementele subșirului comun de lungime maximă.

Restricții și precizări

- $0 < N < 101$
- $0 < M_i < 1001$
- Dacă avem un sir de numere a_1, a_2, \dots, a_n atunci numim subșir un sir de forma $a_{i_1}, a_{i_2}, \dots, a_{i_k}$ cu i_1, i_2, \dots, i_k aparținând mulțimii $\{1, 2, \dots, n\}$ și $i_1 < i_2 < \dots < i_k$.
- Elementele sirurilor sunt numere întregi în intervalul $[1, 1\ 000\ 000]$.
- Elementele fiecărui sir sunt date în ordine crescătoare.
- Pentru 60% din teste, elementele fiecărui sir sunt distințe.
- Pentru 90% din teste, elementele sirurilor sunt în intervalul $[1, 10\ 000]$.

Exemplu:

ausoara.in	ausoara.out	Explicații
1 3 1 2 3	3 1 2 3	
2 2 1 2 2 2 3	1 2	
3 6 1 2 2 3 5 5 9 2 2 2 2 2 5 5 5 7 9 2 2 2 4 5 7 7 7 7	3 2 2 5	
3 3 1 2 3 3 4 5 6 3 7 8 9	0	
3 3 1 1 1 1 1 2 1 1	1 1	

Timp maxim de executare/test: **0.3** secunde

Memorie: total **16 MB** din care pentru stivă **8 MB**

25.4.1 Indicații de rezolvare

mrd. Marius Laurențiu Stroe, Sl. Dr. Ing. Mugurel Ionuț Andreica, stud. Andrei Pârvu

Soluție 60p

Numerele fiecărui sir sunt distințe. Cum valorile nu depășesc 10^4 putem folosi un *vector de frecvențe*.

Astfel, dacă incrementăm pentru fiecare număr de câte ori apare în fiecare sir, atunci rezultatul sunt toate numerele care apar de N ori.

Soluție 90p

Valorile sunt mai mici decât 10^4 . Astfel, considerăm fiecare număr și iterăm prin toate mulțimile să vedem de câte ori apare fiecare număr. Luăm numărul minim de apariții pentru fiecare număr, obținem o complexitate de $O(N * MAXV)$. Pentru a afla în $O(1)$ de câte ori apare un număr într-un sir, va trebui să preprocesăm aceste valori.

Soluție 100p

Cum sirurile sunt sortate, se poate efectua o *interclasare* a tuturor. Interclasarea se poate face fie două căte două mulțimi sau ținând căte un indice pentru fiecare sir. Complexitate $O(N)$.

25.4.2 Cod sursă

Listing 25.4.1: ausoara-90-bogdan.cpp

```

1  /* (C) 2013 Bogdan Cristian Tataroiu */
2  #include <stdio.h>
3  #include <string.h>
4  #include <assert.h>
5
6  #define MAXN 100
7  #define MAXM 1000
8  #define MAXC 1000000
9
10 int N, cnt[MAXC], cntMult[MAXC];
11
12 inline int min(int a, int b)

```

```

13  {
14      if (a < b)
15          return a;
16      return b;
17  }
18
19 int main()
20 {
21     assert(freopen("ausoara.in", "rt", stdin));
22     assert(freopen("ausoara.out", "wt", stdout));
23
24     int i, j;
25     assert(scanf("%d", &N) == 1);
26     assert(1 <= N && N <= MAXN);
27     int maxVal = 0;
28     for (i = 0; i < N; i++)
29     {
30         int M;
31         assert(scanf("%d", &M) == 1);
32         assert(1 <= M && M <= MAXM);
33
34         if (maxVal > 0)
35             memset(cntMult, 0, sizeof(int) * maxVal);
36
37         for (j = 0; j < M; j++)
38         {
39             int val;
40             assert(scanf("%d", &val) == 1);
41             for (; maxVal <= val - 1; maxVal++)
42             {
43                 if (i == 0)
44                     cnt[maxVal] = 0x3f3f3f3f;
45                 else
46                     cnt[maxVal] = 0;
47
48                 cntMult[maxVal] = 0;
49             }
50
51             assert(1 <= val && val <= MAXC);
52             cntMult[val - 1] += 1;
53         }
54
55         for (j = 0; j < maxVal; j++)
56             cnt[j] = min(cnt[j], cntMult[j]);
57     }
58
59     int CNT = 0;
60     for (i = 0; i < maxVal; i++)
61         CNT += cnt[i];
62
63     printf("%d", CNT);
64     for (i = 0; i < maxVal; i++)
65         for (; cnt[i]; cnt[i]--)
66             printf(" %d", i + 1);
67
68     printf("\n");
69
70     return 0;
71 }
```

Listing 25.4.2: ausoara-100-bogdan.c

```

1  /* (C) 2013 Bogdan Cristian Tataroiu */
2  #include <stdio.h>
3  #include <string.h>
4  #include <assert.h>
5
6  #define MAXN 100
7  #define MAXM 1000
8  #define MAXC 1000000
9
10 int N, cnt[MAXC], lastSeen[MAXC], cntMult[MAXC];
11 int v[MAXM];
12
13 inline int min(int a, int b)
```

```

14  {
15      if (a < b)
16          return a;
17
18      return b;
19 }
20
21 int main()
22 {
23     assert(freopen("ausoara.in", "rt", stdin));
24     assert(freopen("ausoara.out", "wt", stdout));
25
26     memset(cnt, 0x3f, sizeof(cnt));
27     memset(lastSeen, -1, sizeof(lastSeen));
28     memset(cntMult, 0, sizeof(cntMult));
29
30     int i, j;
31     assert(scanf("%d", &N) == 1);
32     assert(1 <= N && N <= MAXN);
33
34     for (i = 0; i < N; i++)
35     {
36         int M;
37         assert(scanf("%d", &M) == 1);
38         assert(1 <= M && M <= MAXM);
39
40         for (j = 0; j < M; j++)
41         {
42             assert(scanf("%d", v + j) == 1);
43             assert(1 <= v[j] && v[j] <= MAXC);
44             cntMult[v[j] - 1] += 1;
45         }
46
47         for (j = 0; j < M; j++)
48         {
49             if (lastSeen[v[j] - 1] < i - 1)
50                 cnt[v[j] - 1] = 0;
51             else
52             {
53                 cnt[v[j] - 1] = min(cnt[v[j] - 1], cntMult[v[j] - 1]);
54                 lastSeen[v[j] - 1] = i;
55             }
56         }
57
58         for (j = 0; j < M; j++)
59             cntMult[v[j] - 1] -= 1;
60     }
61
62     int CNT = 0;
63     for (i = 0; i < MAXC; i++)
64         if (lastSeen[i] == N - 1)
65             CNT += cnt[i];
66
67     printf("%d", CNT);
68     for (i = 0; i < MAXC; i++)
69         for (; lastSeen[i] == N - 1 && cnt[i]; cnt[i]--)
70             printf(" %d", i + 1);
71
72     printf("\n");
73
74     return 0;
75 }
```

Listing 25.4.3: ausoara_60_marius.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <cassert>
5
6 using namespace std;
7
8 const char iname[] = "ausoara.in";
9 const char oname[] = "ausoara.out";
10 const int kMaxValue = 1000005;
```

```

11 vector <int> counts(k.MaxValue);
12
13
14 int main(void)
15 {
16     int num_sets;
17
18     ifstream in(iname);
19     assert(in >> num_sets);
20     assert(1 <= num_sets && num_sets <= 100);
21     for (int i = 0; i < num_sets; ++ i)
22     {
23         int set_size;
24         assert(in >> set_size);
25         assert(1 <= set_size && set_size <= 1000);
26         for (int j = 0; j < set_size; ++ j)
27         {
28             int number;
29             assert(in >> number);
30             assert(1 <= number && number <= 1000000);
31             counts[number]++;
32         }
33     }
34
35     ofstream out(ename);
36     int answer = 0;
37     for (int i = 1; i < k.MaxValue; ++ i)
38         if (counts[i] == num_sets)
39             answer++;
40     out << answer << " ";
41     for (int i = 1; i < k.MaxValue; ++ i)
42         if (counts[i] == num_sets)
43             out << i << " ";
44     out.close();
45     return 0;
46 }
```

Listing 25.4.4: ausoara_dragos.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 ifstream fin ("ausoara.in");
6 ofstream fout ("ausoara.out");
7
8 const int MAXVAL=1000005;
9
10 int used[MAXVAL],cnt[MAXVAL];
11 int N,M,ret;
12
13 int main ()
14 {
15     int N,M; fin>>N>>M;
16     for (int j=1; j<=M; ++j)
17     {
18         int x;
19         fin>>x;
20         used[x]=1;
21         ++cnt[x];
22     }
23
24     for (int i=2; i<=N; ++i)
25     {
26         int prev,curr,count;
27
28         fin>>M>>prev; count=1;
29         for (int j=2; j<=M; ++j)
30         {
31             fin>>curr;
32
33             if (prev==curr)
34                 ++count;
35             else
36             {
```

```

37         if (used[prev]==i-1)
38         {
39             cnt[prev]=min (cnt[prev],count);
40             used[prev]=i;
41         }
42         else
43             cnt[prev]=0;
44
45         count=1;
46         prev=curr;
47     }
48 }
49
50     if (used[prev]==i-1)
51     {
52         cnt[prev]=min (cnt[prev],count);
53         used[prev]=i;
54     }
55     else
56         cnt[prev]=0;
57 }
58
59     for (int i=1; i<=MAXVAL; ++i)
60         if (used[i]==N)
61             ret+=cnt[i];
62
63     fout<<ret<<" ";
64     for (int i=1; i<=MAXVAL; ++i)
65         if (used[i]==N)
66             for (int j=1; j<=cnt[i]; ++j)
67                 fout<<i<<" ";
68
69     return 0;
70 }
```

Listing 25.4.5: ausoara_PA.cpp

```

1 #include <cstdio>
2 #include<deque>
3 #include<vector>
4 #include<algorithm>
5
6 using namespace std;
7
8 deque<int> q[10005];
9 vector<int> v;
10
11 int n,m,i,j,x,Cnt;
12
13 int main()
14 {
15     freopen("ausoara.in","r",stdin);
16     freopen("ausoara.out","w",stdout);
17
18     scanf("%d",&n);
19
20     for(i=1;i<=n;i++)
21     {
22         scanf("%d",&m);
23         for(j=1;j<=m;j++)
24         {
25             scanf("%d",&x);
26             q[x].push_back(i);
27         }
28     }
29
30     for(i=1;i<=10000;i++)
31     if(q[i].size())
32     {
33         Cnt=1000100;
34         for(j=1;j<=n;j++)
35         {
36             if(q[i].front()!=j)break;
37             for(cnt=0;q[i].size()&&q[i].front()==j;cnt++)
38                 q[i].pop_front();
```

```

39         Cnt=min(cnt,Cnt);
40     }
41
42     if(j<=n) continue;
43     for(;Cnt;Cnt--) v.push_back(i);
44   }
45
46     printf("%d ",v.size());
47   for(vector<int>::iterator it=v.begin();it!=v.end();it++)
48     printf("%d ",*it);
49   return 0;
50 }
```

Listing 25.4.6: ausoara-1-adrianP.cpp

```

1 #include <cstdio>
2 #include<utility>
3 #include<vector>
4 #include<algorithm>
5
6 using namespace std;
7
8 vector<pair<int,int> > v;
9 vector<int> V;
10
11 int n,m,i,j,x,cnt,Cnt,cnt1,val;
12
13 int main()
14 {
15   freopen("ausoara.in","r",stdin);
16   freopen("ausoara.out","w",stdout);
17
18   scanf("%d",&n);
19   for(i=1;i<=n;i++)
20   {
21     scanf("%d",&m);
22     for(j=1;j<=m;j++)
23     {
24       scanf("%d",&x);
25       v.push_back(make_pair(x,i));
26     }
27   }
28
29   sort(v.begin(),v.end());
30   v.push_back(make_pair(10000000,0));
31   vector<pair<int,int> ::iterator it;
32
33   for(it=v.begin();)
34   {
35     if(it->first>1000000)break;
36     val=it->first;cnt=1000000000;
37     for(i=1;i<=n;i++)
38     {
39       cnt1=0;
40       while(it->first == val && it->second==i){it++;cnt1++;}
41       cnt=min(cnt,cnt1);
42     }
43
44     Cnt+=cnt;
45     for(;cnt;cnt--)V.push_back(val);
46   }
47
48   printf("%d ",Cnt);
49   for(vector<int>::iterator iv=V.begin();iv!=V.end();iv++)
50     printf("%d ",*iv);
51   return 0;
52 }
```

Listing 25.4.7: ausoara-100-mugurel.cpp

```

1 #include <stdio.h>
2
3 #define VMAX 1000001
4 #define ANSMAX 1001
```

```

5
6 int cnt[VMAX], lasti[VMAX];
7 int N, M, i, j, x, lastx, cntx;
8
9 int main()
10 {
11     for (i = 1; i < VMAX; i++)
12     {
13         lasti[i] = 0;
14         cnt[i] = ANSMAX;
15     }
16
17 freopen("ausoara.in", "r", stdin);
18 scanf("%d", &N);
19
20 for (i = 1; i <= N; i++)
21 {
22     scanf("%d", &M);
23     for (lastx = cntx = 0, j = 1; j <= M; j++)
24     {
25         scanf("%d", &x);
26         if (x == lastx)
27             cntx++;
28         else
29         {
30             if (lastx > 0)
31                 if (lasti[lastx] == i - 1)
32                 {
33                     lasti[lastx] = i;
34                     if (cntx < cnt[lastx])
35                         cnt[lastx] = cntx;
36                 }
37
38             lastx = x;
39             cntx = 1;
40         }
41     }
42
43     if (lastx > 0)
44         if (lasti[lastx] == i - 1)
45         {
46             lasti[lastx] = i;
47             if (cntx < cnt[lastx])
48                 cnt[lastx] = cntx;
49         }
50     }
51
52 M = 0;
53 for (i = 1; i < VMAX; i++)
54     if (lasti[i] == N)
55         M += cnt[i];
56
57 freopen("ausoara.out", "w", stdout);
58 printf("%d ", M);
59 for (i = 1; i < VMAX; i++)
60     if (lasti[i] == N)
61         for (j = 1; j <= cnt[i]; j++)
62             printf("%d ", i);
63 printf("\n");
64 return 0;
65 }
```

Listing 25.4.8: ausoara-100-stl.cpp

```

1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4 #include <fstream>
5
6 using namespace std;
7
8 int main()
9 {
10     ifstream cin("ausoara.in");
11     ofstream cout("ausoara.out");

```

```

12
13     int N; cin >> N;
14     vector<int> intersect;
15
16     for (int i = 0; i < N; ++i)
17     {
18         int K; cin >> K;
19         vector<int> V(K);
20         for (int j = 0; j < K; ++j)
21             cin >> V[j];
22
23         if (i == 0)
24         {
25             intersect = V;
26             continue;
27         }
28
29         vector<int> aux(K);
30         aux.erase(set_intersection(intersect.begin(),
31                                     intersect.end(),
32                                     V.begin(), V.end(),
33                                     aux.begin()), aux.end());
34         swap(intersect, aux);
35     }
36
37     cout << intersect.size() << " ";
38     for (int i = 0; i < int(intersect.size()); ++i)
39         cout << intersect[i] << " ";
40
41 }
```

25.4.3 *Rezolvare detaliată

25.5 drumuri

Problema 5 - drumuri

100 de puncte

Fie G un graf orientat cu N noduri și M arce.

Spunem că nodul Y este accesibil din nodul X dacă se poate ajunge de la X la Y mergând pe arce în sensul corespunzător al acestora.

Spunem că nodul X este "popular" dacă pentru fiecare nod Y al grafului G se îndeplinește cel puțin una din condițiile:

1. X este accesibil din Y ;
2. Y este accesibil din X .

Cerințe

Dându-se cele două numere N și M cât și arcele grafului, să se afle care sunt nodurile populare din graf.

Date de intrare

Prima linie a fișierului **drumuri.in** conține numerele N și M , cu semnificația din enunț. Următoarele M linii conțin câte două numere X și Y , semnificând faptul că există arc orientat de la X la Y .

Date de ieșire

Prima linie a fișierului **drumuri.out** conține numărul NR , reprezentând numărul de noduri populare ale grafului. Următoarea linie va conține cele NR noduri populare afișate în ordine crescătoare.

Restricții și precizări

- $1 \leq N \leq 150\ 000$
- $1 \leq M \leq 300\ 000$
- Pentru 50% din punctaj $N \leq 700, M \leq 1100$
- Pentru 65% din teste, G este aciclic

Exemple:

drumuri.in	drumuri.out	Explicații
5 4	3	Nodurile 2, 4 și 5 sunt singurele noduri populare.
1 2	2 4 5	Nodul 1, spre exemplu, nu este popular deoarece nu este accesibil din 3, iar nici nodul 3 nu este accesibil din 1.
3 2		
2 4		
4 5		

Timp maxim de executare/test: **0.3** secunde

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

25.5.1 Indicații de rezolvare

stud. Alexandru Cazacu, Universitatea București

Soluție 50 puncte - complexitate $O(N * (N + M))$

Pentru 50% din teste, putem să calculăm pentru fiecare nod în parte, nodurile în care se poate ajunge plecând din el. Deasemnea, făcând încă o parcurgere pe graful transpus, putem vedea nodurile din care se ajunge în nodul curent.

Solutia 1 (100 de puncte) Alex Cazacu - complexitate $O(N)$

Se observă că dacă un nod este în mulțimea soluțiilor, atunci toate nodurile din *componenta sa tare conexă* respectă această proprietate.

Putem lucra pe *graful aciclic al componentelor tare conexe*.

În continuare, începem să parcurgem nodurile ca la *sortarea topologică*, pornind din cele exterioare (care au gradul interior 0).

Dacă la un moment dat avem un singur nod exterior, atunci acest nod este soluție.

În momentul în care avem mai multe astfel de noduri, este evident că nu se poate ajunge de la unul la altul.

Următorul nod din soluție, este cel în care se unesc aceste "ramuri" care pleacă din nodurile exterioare.

Este necesară o tăiere simultană a lor.

Procedam astfel : calculăm pentru fiecare nod distanța celui mai lung drum care pleacă din el. La un pas eliminăm nodurile cu gradul de intrare 0, care au distanță maximă. Când ajungem să avem la un pas, un singur nod exterior, il adăugam la soluție.

Solutia 2 (100 de puncte) Adrian Panaete - complexitate $O(N)$

Lucrăm din nou pe *graful aciclic al componentelor tare conexe*.

Fie T o *sortare topologică* a *grafului*.

Pentru ca un nod X situat pe poziția P în sortarea topologică să fie popular este nevoie ca X să fie accesibil din fiecare nod situat pe o poziție $< P$ în sortare.

Analog, este nevoie ca fiecare nod situat pe o poziție $> P$ în sortare să fie accesibil din X .

Astfel, ne vine ideea să ținem pentru fiecare nod X valoarea

$nr_pred[X]$ = numarul de predecesori ai lui X în sortare cu proprietatea că X este accesibil din predecesorul respectiv.

Această valoare nu poate fi calculată în timp liniar pentru fiecare nod, deoarece un algoritm simplu de tip programare dinamică va număra anumiți predecesori de mai multe ori. Putem însă realiza un algoritm simplu care calculează valorile $nr_pred[]$ în mod corect doar pentru nodurile candidate la soluție.

Mai exact, să spunem că avem deja calculată valoarea $nr_pred[X]$. Vom aduna aceasta valoare la primul vecin al lui X care îl urmează în sortarea topologică. Astfel știm sigur că valoarea $nr_pred[]$ va fi corectă pentru fiecare nod care este accesibil din toate nodurile precedente.

Argumentăm prin faptul că singurul caz în care un anumit nod poate "rata" un update de la un predecesor este cel în care nodul nu este accesibil din predecesorul respectiv. Astfel, el nu poate face parte din soluție.

Pentru a completa soluția, vom aplica același algoritm și pe *graful transpus* pentru a calcula numărul de succesorii pentru fiecare nod.

Nodurile care au atât numarul necesar de predecesori cat și numarul necesar de succesorii vor fi nodurile din solutie.

25.5.2 Cod sursă

Listing 25.5.1: drumuri-50-mugurel.cpp

```

1 // Mugurel Ionut Andreica
2 // Brute force: O(N * (N + M))
3
4 #include <stdio.h>
5 #include <vector>
6
7 using namespace std;
8
9 #define NMAX 150111
10
11 vector<int> vec[NMAX][2];
12 int N, M;
13
14 void ReadInput()
15 {
16     int i, j, k;
17     freopen("drumuri.in", "r", stdin);
18     scanf("%d %d", &N, &M);
19     for (k = 1; k <= M; k++)
20     {
21         scanf("%d %d", &i, &j);
22         vec[i][0].push_back(j);
23         vec[j][1].push_back(i);
24     }
25 }
26
27 int visited[NMAX][2];
28 int qnode[2 * NMAX], qdir[2 * NMAX], li, ls;
29
30 int IsPopular(int start)
31 {
32     int i, dir, j, k, nviz = 1;
33
34     qnode[1] = start; qdir[1] = 0;
35     qnode[2] = start; qdir[2] = 1;
36     li = 1; ls = 2;
37     visited[start][0] = visited[start][1] = start;
38
39     while (li <= ls && nviz < N)
40     {
41         i = qnode[li];
42         dir = qdir[li];
43         li++;
44
45         for (k = 0; k < vec[i][dir].size(); k++)
46         {
47             j = vec[i][dir][k];
48             if (visited[j][dir] != start)
49             {
50                 visited[j][dir] = start;
51                 ls++;
52                 qnode[ls] = j;
53                 qdir[ls] = dir;
54                 if (visited[j][1 - dir] != start)
55                     nviz++;
56             }
57         }
58     }
59
60 //fprintf(stderr, "start=%d nviz=%d ls=%d\n", start, nviz, ls);
61 return (nviz == N);
62 }
63

```

```

64 char popular[NMAX];
65
66 void Solve()
67 {
68     M = 0;
69     for (int start = 1; start <= N; start++)
70         if (IsPopular(start))
71         {
72             popular[start] = 1;
73             M++;
74         }
75     else
76         popular[start] = 0;
77 }
78
79 void WriteOutput()
80 {
81     int i;
82     freopen("drumuri.out", "w", stdout);
83     printf("%d\n", M);
84     for (i = 1; i <= N; i++)
85         if (popular[i])
86             printf("%d ", i);
87     printf("\n");
88 }
89
90 int main()
91 {
92     ReadInput();
93     Solve();
94     WriteOutput();
95     return 0;
96 }
```

Listing 25.5.2: drumuri-100-adi.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <stack>
5 #include <algorithm>
6
7 using namespace std;
8
9 ostream& operator<<(ostream& cout, const vector<int> &V)
10 {
11     cout << V.size() << "\n";
12
13     for (vector<int>::const_iterator it = V.begin(); it != V.end(); ++it)
14         cout << *it + 1 << " ";
15     cout << "\n";
16     return cout;
17 }
18
19
20 class Graph
21 {
22     public:
23     Graph(const int &size):
24         edges_(size) {
25     }
26
27     void addArc(const int &from, const int &to)
28     {
29         edges_[from].push_back(to);
30     }
31
32     vector<int> kernels()
33     {
34         lowlink = index = scc = vector<int>(edges_.size(), 0);
35         indecs = 0;
36
37         for (int i = 0; i < int(edges_.size()); ++i)
38         {
39             tarjan(i);

```

```

40         }
41
42     vector<int> bestLeft(sccs.size(), 1),
43             bestRight(sccs.size(), 1),
44             aux(sccs.size(), 1);
45
46     for (int i = 0; i < int(sccs.size()); ++i)
47     {
48         for (vector<int>::iterator it = sccs[i].begin();
49              it != sccs[i].end(); ++it)
50             for (vector<int>::iterator jt = edges_[*it].begin();
51                  jt != edges_[*it].end(); ++jt)
52                 if (i != scc[*jt])
53                 {
54                     aux[i] += aux[scc[*jt]];
55                     aux[scc[*jt]] = 0;
56                 }
57
58         bestLeft[i] = aux[i];
59     }
60
61     vector<bool> ok(edges_.size(), false);
62     int total = 0;
63     for (int i = sccs.size() - 1; i >= 0; --i)
64     {
65         if (bestLeft[i] + bestRight[i] == int(sccs.size()) + 1)
66         {
67             total += sccs[i].size();
68             for (vector<int>::iterator it = sccs[i].begin();
69                  it != sccs[i].end(); ++it)
70                 ok[*it] = true;
71         }
72
73         int minplace = -1;
74         for (vector<int>::iterator it = sccs[i].begin();
75              it != sccs[i].end(); ++it)
76             for (vector<int>::iterator jt = edges_[*it].begin();
77                  jt != edges_[*it].end(); ++jt)
78                 if (i > scc[*jt] && scc[*jt] > minplace)
79                     minplace = scc[*jt];
80
81         if (minplace != -1)
82             bestRight[minplace] += bestRight[i];
83     }
84
85     vector<int> answer;
86     answer.reserve(total);
87     for (int i = 0; i < int(edges_.size()); ++i)
88         if (ok[i])
89             answer.push_back(i);
90     return answer;
91 }
92
93 private:
94     void tarjan(const int &node)
95     {
96         if (lowlink[node] != 0)
97             return;
98
99         index[node] = lowlink[node] = ++indecs;
100        S.push(node);
101
102        for (vector<int>::iterator it = edges_[node].begin();
103              it != edges_[node].end(); ++it)
104            if (index[*it] == 0)
105            {
106                tarjan(*it);
107                lowlink[node] = min(lowlink[node], lowlink[*it]);
108            }
109            else
110                if (index[*it] != -1)
111                {
112                    lowlink[node] = min(lowlink[node], lowlink[*it]);
113                }
114
115        if (lowlink[node] == index[node])

```

```

116         {
117             vector<int> now;
118             int nod;
119             do
120             {
121                 nod = S.top();
122                 S.pop();
123                 now.push_back(nod);
124                 scc[nod] = sccs.size();
125                 index[nod] = -1;
126             } while (nod != node);
127             sccs.push_back(now);
128         }
129     }
130 }
131
132 void randomSort()
133 {
134     for (int i = 0; i < int(sccs.size()); ++i)
135         random_shuffle(E[i].begin(), E[i].end());
136
137     vector<int> list(sccs.size());
138     for (int i = 0; i < int(sccs.size()); ++i)
139         list[i] = i;
140     random_shuffle(list.begin(), list.end());
141
142     top.clear();
143     been = vector<bool>(sccs.size(), false);
144     for (int i = 0; i < int(sccs.size()); ++i)
145         dfs(list[i]);
146 }
147
148 void dfs(int nod)
149 {
150     if (been[nod])
151         return;
152     been[nod] = true;
153     for (vector<int>::iterator it = E[nod].begin();
154          it != E[nod].end(); ++it)
155         dfs(*it);
156
157     top.push_back(nod);
158 }
159
160 stack<int> S;
161 vector<vector<int>> sccs;
162 vector<int> index, lowlink, scc;
163 int indecs;
164 vector<vector<int>> edges_, E;
165 vector<int> top;
166 vector<bool> been;
167 };
168
169 int main()
170 {
171     ifstream cin("drumuri.in");
172     ofstream cout("drumuri.out");
173
174     int N, M;
175     cin >> N >> M;
176     Graph G(N);
177     for (int i = 0; i < M; ++i)
178     {
179         int x, y; cin >> x >> y;
180         G.addArc(x - 1, y - 1);
181     }
182
183     cout << G.kernels();
184 }
```

Listing 25.5.3: drumuri-100-alex.cpp

```

1 #include<cstdio>
2 #include <fstream>
3 #include<cstring>
```

```

4 #include<algorithm>
5 #include<vector>
6 #include<stack>
7 #include<queue>
8 using namespace std;
9
10 #define NMAX 150001
11
12 int N, M;
13 int used[NMAX];
14 vector<int> graph[NMAX], transpose[NMAX], dag[NMAX], transpose_dag[NMAX];
15 stack<int> Stack;
16
17 int num_of_scc;
18 int where[NMAX];
19 int num[NMAX];
20
21 int in[NMAX];
22 int out[NMAX], max_dist[NMAX];
23
24 vector<int> outer[NMAX];
25
26 void read_data()
27 {
28     ifstream fin ("drumuri.in");
29
30     int x, y;
31
32 //    scanf("%d %d", &N, &M);
33     fin >> N >> M;
34     for(int i = 1; i <= M; i++) {
35 //        scanf("%d %d", &x, &y);
36         fin >> x >> y;
37         graph[x].push_back(y);
38         transpose[y].push_back(x);
39     }
40 }
41
42 void dfs_scc(int node, int type, vector<int> *graph)
43 {
44
45     used[node] = type;
46     if (type == 2)
47     {
48         where[node] = num_of_scc;
49         num[num_of_scc]++;
50     }
51
52     for(vector<int>::iterator it = graph[node].begin();
53          it != graph[node].end(); it++)
54         if (used[*it] != type)
55             dfs_scc(*it, type, graph);
56
57     if (type == 1)
58         Stack.push(node);
59 }
60
61 void compute_scc()
62 {
63
64     for(int i = 1; i <= N; i++)
65         if (used[i] != 1)
66             dfs_scc(i, 1, graph);
67
68     for(; !Stack.empty(); Stack.pop())
69     {
70         if(used[Stack.top()] == 2) continue;
71         num_of_scc++;
72         dfs_scc(Stack.top(), 2, transpose);
73     }
74 }
75
76 void make_dag()
77 {
78     compute_scc();
79 }
```

```

80 //  for(int i = 1; i <= N; i++)
81 //      printf("%d %d\n", i, where[i]);
82
83     for(int node = 1; node <= N; node++)
84         for(vector<int>::iterator it = graph[node].begin();
85              it != graph[node].end(); it++)
86             if (where[node] != where[*it])
87             {
88                 dag[ where[node] ].push_back ( where[*it] );
89                 in[ where[*it] ]++;
90                 out[ where[node] ]++;
91                 transpose_dag[ where[*it] ].push_back ( where[node] );
92 //                 printf("transpose_dag : %d %d\n", where[*it], where[node]);
93             }
94     }
95
96 int is_solution[NMAX];
97
98 void compute_max_dist(int N, vector<int> *graph)
99 {
100     int node;
101
102     queue <int> Q;
103     for (node = 1; node <= N; node++)
104         if (out[node] == 0)
105             Q.push(node);
106
107     for(;!Q.empty(); Q.pop())
108     {
109         node = Q.front();
110         //printf("node %d\n", node);
111         for(vector<int>::iterator it = graph[node].begin();
112              it != graph[node].end(); it++)
113             {
114                 out[*it]--;
115                 if (out[*it] == 0)
116                     Q.push(*it);
117                 max_dist[*it] = max(max_dist[*it], 1 + max_dist[node]);
118             }
119     }
120 }
121
122 void solve()
123 {
124     vector<int> solutions;
125     int num = 0, dmax = -1;
126
127     compute_max_dist(num_of_scc, transpose_dag);
128     //for(int i = 1; i <= num_of_scc; i++)
129     //    printf("dist max : %d %d\n", i, max_dist[i]);
130
131     for(int i = 1; i <= num_of_scc; i++)
132         if (in[i] == 0)
133         {
134             outer[ max_dist[i] ].push_back (i);
135             num++;
136             dmax = max(dmax, max_dist[i]);
137         }
138
139     for ( ; dmax >= 0; )
140     {
141         //printf("num : %d dmax : %d\n", num, dmax);
142
143         if (num == 1)
144         {
145             //printf("is sol %d\n", outer[dmax][0]);
146             is_solution[ outer[dmax][0] ] = 1;
147         }
148
149         for(int i = 0; i < (int)outer[dmax].size(); i++)
150         {
151
152             int node = outer[dmax][i];
153             //printf("%d ", node);
154             for(vector<int>::iterator it = dag[node].begin();
155                 it != dag[node].end(); it++)

```

```

156         {
157             //printf("(%d) ", *it);
158             in[*it]--;
159             if (in[*it] == 0)
160             {
161                 outer[ max_dist[*it] ].push_back( *it );
162                 num++;
163             }
164         }
165         //printf("final");
166     }
167
168     num-= outer[dmax].size();
169     //printf("#%d\n", dmax);
170     outer[dmax].clear();
171
172     while (dmax >= 0 && outer[dmax].size() == 0)
173         dmax--;
174     }
175
176     for(int i = 1; i <= N; i++)
177     {
178 //        printf("#%d %d\n", i, where[i]);
179         if(is_solution[where[i]])
180             solutions.push_back(i);
181     }
182
183     ofstream fout ("drumuri.out");
184 /*
185     for (int i = 1; i <= num_of_scc; i++)
186         fout << ::num[i] << ' ';
187     fout << '\n';
188 */
189 //    printf("%d\n", solutions.size());
190     fout << solutions.size() << '\n';
191     for(int i = 0; i < (int)solutions.size(); i++)
192 //        printf("%d ", solutions[i]);
193         fout << solutions[i] << ' ';
194     }
195
196 int main()
197 {
198     freopen("drumuri.in", "r", stdin);
199     freopen("drumuri.out", "w", stdout);
200
201     read_data();
202     make_dag();
203     solve();
204
205     return 0;
206 }
```

Listing 25.5.4: drumuri-100-andrei.cpp

```

1 #include <cstdio>
2 #include <vector>
3 #include <algorithm>
4
5 using namespace std;
6
7 const int lg = 150005;
8
9 bool fst[lg];
10 vector<int> v[lg], w[lg], vec[lg], rev_vec[lg];
11 vector<pair<int, int> > muchii;
12 vector<int> top_sort;
13 int postordine[lg];
14 vector<vector<int> > comps;
15 int which[lg];
16 int nr_tare;
17 int cnt[2][lg];
18 int n, m;
19 bool good[lg];
20 int index[lg];
21
```

```

22 void df1(int nod)
23 {
24     fst[nod] = true;
25
26     for (int i = 0; i < (int)v[nod].size(); i++)
27         if (fst[v[nod][i]] == false)
28             df1(v[nod][i]);
29
30     postordine[postordine[0]] = nod;
31 }
32
33 void df2(int nod)
34 {
35     comps[comps.size() - 1].push_back(nod);
36     fst[nod] = false;
37     which[nod] = nr_tare;
38
39     for (int i = 0; i < (int)w[nod].size(); i++)
40         if (fst[w[nod][i]] == true)
41             df2(w[nod][i]);
42 }
43
44 void doSort(int nod)
45 {
46     fst[nod] = true;
47
48     for (int i = 0; i < (int)vec[nod].size(); i++)
49         if (fst[vec[nod][i]] == false)
50             doSort(vec[nod][i]);
51
52     top_sort.push_back(nod);
53 }
54
55 void calcAll(vector<int> top, vector<int> rec[lg], int s)
56 {
57     for (int i = 0; i < (int)top.size(); i++)
58         index[top[i]] = i;
59
60     for (int i = 0; i < (int)top.size(); i++)
61     {
62         cnt[s][top[i]]++;
63
64         int mn = -1;
65         for (int j = 0; j < (int)rec[top[i]].size(); j++)
66             if (mn == -1 || index[rec[top[i]][j]] < index[mn])
67                 mn = rec[top[i]][j];
68
69         if (mn != -1)
70             cnt[s][mn] += cnt[s][top[i]];
71     }
72 }
73
74 int main()
75 {
76     freopen("drumuri.in", "rt", stdin);
77     freopen("drumuri.out", "wt", stdout);
78
79     scanf("%d%d", &n, &m);
80
81     for (int i = 1; i <= m; i++)
82     {
83         int x, y;
84
85         scanf("%d%d", &x, &y);
86
87         v[x].push_back(y);
88         w[y].push_back(x);
89
90         muchii.push_back(make_pair(x, y));
91     }
92
93     for (int i = 1; i <= n; i++)
94         if (fst[i] == false)
95             df1(i);
96
97     for (int i = n; i > 0; i--)

```

```

98     {
99         if (fst[postordine[i]] == true)
100        {
101            nr_tare++;
102            comps.push_back(vector<int>());
103            df2(postordine[i]);
104        }
105    }
106
107    for (int i = 1; i <= n; i++)
108    {
109        //printf("%d %d\n", i, which[i]);
110    }
111
112    for (int i = 0; i < m; i++)
113        if (which[muchii[i].first] != which[muchii[i].second])
114        {
115            vec[which[muchii[i].first]].push_back(which[muchii[i].second]);
116            rev_vec[which[muchii[i].second]].push_back(which[muchii[i].first]);
117        }
118
119    for (int i = 1; i <= nr_tare; i++)
120        if (fst[i] == false)
121            doSort(i);
122
123    calcAll(top_sort, rev_vec, 0);
124
125    reverse(top_sort.begin(), top_sort.end());
126
127    for (int i = 0; i < (int)top_sort.size(); i++)
128    {
129        //printf("%d\n", top_sort[i]);
130    }
131
132    calcAll(top_sort, vec, 1);
133
134    for (int i = 1; i <= nr_tare; i++)
135    {
136        //printf("%d %d\n", i, cnt[0][i] + cnt[1][i]);
137    }
138
139    int nr_rez = 0;
140    for (int i = 1; i <= nr_tare; i++)
141        if (cnt[0][i] + cnt[1][i] == nr_tare + 1)
142            for (int j = 0; j < (int)comps[i - 1].size(); j++)
143            {
144                nr_rez++;
145                good[comps[i - 1][j]] = true;
146            }
147
148    printf("%d\n", nr_rez);
149    for (int i = 1; i <= n; i++)
150        if (good[i] == true)
151            printf("%d ", i);
152
153    printf("\n");
154
155    return 0;
156}

```

25.5.3 *Rezolvare detaliată

25.6 xnumere

Problema 6 - xnumere

100 de puncte

Din când în când un turist se gândește la o problemă dificilă (sau mai multe). Găsește pe drum un sir de N numere intregi de la 1 la K . În călătoria spre regăsirea sinelui, fiecare sir conține exact X numere distințe din mulțimea $\{1\dots K\}$. La sfârșitul călătoriei sale trage linia și vede numărul

de şiruri distințe. Bucuros că a reușit să numere şirurile, vrea să vadă dacă și voi puteți găsi răspunsul la problema sa (simplă, de altfel).

Cerințe

Determinați numărul de şiruri distințe de lungime N cu toate numerele din mulțimea $\{1\dots K\}$, fiecare şir având exact X elemente distințe.

Date de intrare

Fișierul **xnumere.in** va conține pe prima linie trei numere naturale: $K \ X \ N$.

Date de ieșire

Fișierul **xnumere.out** va conține un singur număr natural reprezentând răspunsul dat întrebării unui turist oarecare. Rezultatul va fi scris în fișier modulo 666013.

Restricții și precizări

- $1 \leq X \leq \min(K, 10^5)$
- $1 \leq N, K \leq 10^{15}$
- Pentru 10% din teste se garantează $N, K, X \leq 7$.
- Pentru 30% din teste se garantează $N \leq 10000, K \leq 100$.
- Pentru 60% din teste se garantează $K \leq 100$.
- Pentru 85% din teste se garantează $K \leq 1\,000$.
- 2 şiruri $A = (x_1, x_2, \dots, x_n)$ și $B = (y_1, y_2, \dots, y_n)$ sunt distințe dacă există cel puțin o poziție i pentru care $x_i \neq y_i$.

Exemple:

xnumere.in	xnumere.out	Explicații
2 2 4	14	şirurile sunt: (1,1,1,2),(1,1,2,1),(1,1,2,2),(1,2,1,1), (1,2,1,2),(1,2,2,1),(1,2,2,2),(2,1,1,1), (2,1,1,2),(2,1,2,1),(2,1,2,2),(2,2,1,1), (2,2,1,2),(2,2,2,1)
10 6 8	258420	

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

Memorie: total **64 MB** din care pentru stivă **16 MB**

25.6.1 Indicații de rezolvare

Stud. Dragoș Alin Rotaru - Universitatea București
Stud. Bogdan Cristian Tătăroiu - University of Cambridge

1. 10 p - $O(K^N)$

Se generează toate soluțiile cu *backtracking*.

2. 30p - $O(N * K)$

Se observă recurența următoare : $D[i][j]$ - numărul de şiruri de lungime i cu fix j numere distințe.

$$D[i][j] = D[i-1][j] * j + D[i-1][j-1] * (K-j+1)$$

în primul caz adăugăm la pasul i un element deja în mulțimea $\{1..j\}$ iar în al 2-lea adăugăm un element care se află în mulțimea $\{j+1..K\}$.

Starea inițială este $D[0][0]=1$. Răspunsul se află în $D[N][X]$.

3. 60p - $O(\log(N) * K^3)$

Notăm $R(i) = (D[i][0], D[i][1], \dots, D[i][K])$.

Optimizând soluția de 30 de puncte obținem o matrice M de dimensiune $K*K$, în care $M[i][i]=i$, $i=1, K$ și $M[j][j-1] = K-j+1$, oricare $j=2, K$.

Dacă înmulțim matricea M cu vectorul coloana $R(0)$ obținem $R(1)$.

Utilizând inducția, $R(i) = M^N * R(i-1)$.

Din $M^N * R(0)$ vom obține $R(N)$.

Pe linia X a vectorului R vom obține răspunsul căutat.

Vom calcula M^N folosind *exponențierea rapidă* de matrice ($M^N = M^{N/2} * M^{N/2}$ când N e par; $M^N = M^{N/2} * M^{N/2} * M$, pentru N impar)

4. 85p - $O(X^2 + X \log(N))$

Fie $d[i] =$ numărul de siruri de lungime N formate din fix i numere distincte cu valori între 1 și i .

Pentru a calcula $d[i]$, vom considera toate sirurile de lungime N cu valori între 1 și i (în număr de i^N) și vom elimina din acestea pe cele care nu sunt formate din fix i numere distincte.

Pentru orice $j < i$, stim

$d[j] =$ numărul de siruri de lungime N cu fix j numere distincte cu valori între 1 și j

și putem astfel calcula numărul de siruri de lungime N cu fix j numere distincte cu valori între 1 și i ca fiind

$d[j] * \text{comb}[i][j]$,

unde

$\text{comb}[i][j]$ reprezintă numărul de moduri de a alege j valori din i variante posibile.

Ca urmare a acestor observații obținem relația de recurență:

$$d[i] = i^N - \sum_{j=1}^{i-1} d[j] \cdot \text{comb}[i][j]$$

Avem $\text{comb}[K][X]$ moduri de a alege X valori distincte din cele K și $d[X]$ siruri de lungime N cu X valori distincte între 1 și X , deci numărul total de siruri de lungime N cu X valori distincte între 1 și K va fi determinat de numărul de moduri de a alege X numere distincte din cele K înmulțit cu $d[X]$:

$$d[X] * \text{comb}[K][X].$$

5. 100p - $O(X \log(N))$

După analiza atentă a problemei, se observă că rezultatul este de fapt $S[N, X] * A[K, X]$, unde $S[N, X]$ sunt *numerele lui Stirling de speță a doua* (în câte moduri putem împărți N obiecte în X clase) și $A[K, X]$ sunt aranjamente de K luate câte X (în câte moduri putem alege X obiecte din K în condițiile în care ordinea elementelor alese contează).

Să presupunem că avem valorile și ordinea celor X numere fixate:

$v[1], v[2], \dots, v[X]$. $S[N, X]$ reprezintă numărul de moduri de a împărți cele N poziții în X seturi distincte, fiecare set corespunzând unui din cele X numere.

Dacă o poziție face parte din set-ul i , atunci valoarea de pe poziția respectivă va fi $v[i]$.

Numărul de moduri de a fixa sirul de valori $v[1], v[2], \dots, v[X]$ este dat de $A[K, X]$.

$S[N, X]$ se poate calcula în timp $O(X \log N)$ folosind *principiul includerii și excluderii*.

$$S[N][X] = \frac{1}{X!} \sum_{j=1}^X (-1)^{X-j} \text{comp}[X][j] \cdot j^N$$

Demonstrația este simplă, dar nu trivială. O schiță a uneia poate fi gasită în [1], sub capitolele Onto functions și Stirling Numbers of the Second Kind.

$\text{comb}[X][j]$ se poate calcula în timp logaritmic din $\text{comb}[X][j-1]$ utilizând *inverse modulare*.

$A[K, X]$ se poate calcula în $O(X)$ ca produs de $(K - X + 1) * \dots * K$.

[1] http://www.rhitt.com/courses/367/sp03/docs/inclusion_exclusion.pdf

25.6.2 Cod sursă

Listing 25.6.1: xnumere-85-bogdan.c

```

1 /* (C) 2013 Bogdan Cristian Tataroiu */
2 #include <stdio.h>
3 #include <assert.h>
4

```

```

5 #define MAXN 10000000000000000LL
6 #define MAXX 100000
7 #define MAXK 10000000000000000LL
8 #define MOD 666013
9
10 int X;
11 long long N, K;
12
13 int comb[2][MAXX + 1];
14 int d[MAXX + 1];
15
16 inline int modPow(int v, long long power)
17 {
18     long long step = 1LL;
19     int rez = 1;
20     for (; step <= power; step <= 1)
21     {
22         if (power & step)
23             rez = (rez * (long long)v) % MOD;
24
25         v = (v * (long long)v) % MOD;
26     }
27     return rez;
28 }
29
30 int main()
31 {
32     assert(freopen("xnumere.in", "rt", stdin));
33     assert(freopen("xnumere.out", "wt", stdout));
34
35     assert(scanf("%lld %d %lld", &K, &X, &N) == 3);
36     assert(1 <= N && N <= MAXN);
37     assert(1 <= X && X <= K && X <= MAXX);
38     assert(1 <= K && K <= MAXK);
39
40     int i, j;
41     comb[0][0] = 1;
42     for (i = 1; i <= X; i++)
43     {
44         comb[i & 1][0] = 1;
45         for (j = 1; j <= i; j++)
46         {
47             comb[i & 1][j] = comb[(i - 1) & 1][j - 1] + comb[(i - 1) & 1][j];
48             if (comb[i & 1][j] >= MOD)
49                 comb[i & 1][j] -= MOD;
50         }
51
52         d[i] = modPow(i, N);
53         for (j = 1; j < i; j++)
54         {
55             d[i] -= (d[j] * (long long)comb[i & 1][j]) % MOD;
56             if (d[i] < 0)
57                 d[i] += MOD;
58         }
59     }
60
61     int SOL = d[X];
62     for (i = X + 1; i <= K; i++)
63         SOL = (SOL * (long long)i) % MOD;
64
65     for (i = 1; i <= K - X; i++)
66         SOL = (SOL * (long long)modPow(i, MOD - 2)) % MOD;
67
68     printf("%d\n", SOL);
69
70     return 0;
71 }
```

Listing 25.6.2: xnumere-100-bogdan.c

```

1 /* (C) 2013 Bogdan Cristian Tataroiu */
2 #include <stdio.h>
3 #include <assert.h>
4
5 #define MAXN 10000000000000000LL
```

```

6 #define MAXX 100000
7 #define MAXK 10000000000000000LL
8 #define MOD 666013
9
10 int X;
11 long long N, K;
12
13 inline int modPow(int v, long long power)
14 {
15     long long step = 1LL;
16     int rez = 1;
17     for (; step <= power; step <= 1)
18     {
19         if (power & step)
20             rez = (rez * (long long)v) % MOD;
21
22         v = (v * (long long)v) % MOD;
23     }
24     return rez;
25 }
26
27 int main()
28 {
29     assert(freopen("xnumere.in", "rt", stdin));
30     assert(freopen("xnumere.out", "wt", stdout));
31
32     assert(scanf("%lld %d %lld", &K, &X, &N) == 3);
33     assert(1 <= N && N <= MAXN);
34     assert(1 <= X && X <= K && X <= MAXX);
35     assert(1 <= K && K <= MAXK);
36
37     int i, SOL = 0, COMB = 1;
38
39     for (i = 0; i <= X; i++)
40     {
41         if (i >= 1)
42         {
43             COMB = (COMB * (long long)(X - i + 1)) % MOD;
44             COMB = (COMB * (long long)modPow(i, MOD - 2)) % MOD;
45         }
46
47         int V = (COMB * (long long)modPow(i, N)) % MOD;
48         if ((X - i) % 2 == 0)
49         {
50             SOL += V;
51             if (SOL >= MOD)
52                 SOL -= MOD;
53         }
54         else
55         {
56             SOL -= V;
57             if (SOL < 0)
58                 SOL += MOD;
59         }
60     }
61
62     long long p;
63     for (p = K - X + 1; p <= K; p++)
64         SOL = (SOL * (p % MOD)) % MOD;
65
66     for (p = 1; p <= X; p++)
67         SOL = (SOL * (long long)modPow(p, MOD - 2)) % MOD;
68
69     printf("%d\n", SOL);
70
71     return 0;
72 }
```

Listing 25.6.3: xnumere-10-dragos.cpp

```

1 #include <cstdio>
2 #include <fstream>
3 #include <vector>
4 #include <algorithm>
5
```

```

6 const int MAXN = 40;
7 const int MOD = 666013;
8
9 using namespace std;
10
11 int mark[MAXN];
12 int B[MAXN];
13 long long N;
14 int X, K , ans;
15 vector <int> A;
16
17 bool check()
18 {
19     int tmp = 0;
20
21     for( int i = 1; i <= K; ++i)
22         tmp += (mark[i] > 0);
23
24     if(tmp <= X)
25     {
26         // for(int i = 1; i <= B[0]; ++i)
27         // printf("%d ", B[i]);
28         // printf("\n");
29     }
30     return (tmp == X);
31 }
32
33 void back(int k)
34 {
35     if(k == N + 1)
36     {
37         ans += check();
38         return ;
39     }
40     else
41     {
42         for(int i = 1; i <= K; ++i)
43         {
44             mark[i]++;
45             B[++B[0]] = i;
46             back(k + 1);
47             mark[i]--;
48             --B[0];
49         }
50     }
51 }
52
53 int main()
54 {
55     ifstream fin("xnumere.in");
56     ofstream fout("xnumere.out");
57
58     fin >> K >> X >> N;
59     /*
60     for( int i = 1; i <= K; ++i) {
61         int x;
62         fin >> x;
63         A.push_back(x);
64     }
65
66     sort(A.begin() ,A.end());
67     A.resize( unique( A.begin(), A.end() ) - A.begin());
68     K = A.size();
69     */
70     back(1) ;
71
72     fout << ans % MOD << "\n";
73
74     return 0;
75 }
```

Listing 25.6.4: xnumere-30-dragos.cpp

```

1 #include <cstdio>
2 #include <cassert>
```

```

3 #include <iostream>
4 #include <fstream>
5 #include <vector>
6 #include <algorithm>
7
8 const int MOD = 666013;
9
10 using namespace std;
11
12 vector <int> A;
13 long long N;
14 int X, K;
15 int d[2][1000000];
16
17 int main()
18 {
19     ifstream fin("xnumere.in");
20     ofstream fout("xnumere.out");
21
22     fin >> K >> X >> N;
23     assert(N <= 10000);
24     assert(K <= 100);
25
26     d[0][0] = 1;
27
28     for(int i = 1; i <= N; ++i)
29     {
30         //clear
31         for(int j = 0; j <= K; ++j)
32             d[1][j] = 0;
33         // ~clear
34
35         for(int j = 1; j <= K; ++j)
36         {
37             d[1][j] = (1LL * d[0][j] * j) % MOD +
38                         (1LL * (K - j + 1) * d[0][j - 1]) % MOD ;
39             if(d[1][j] >= MOD)
40                 d[1][j] %= MOD;
41         }
42
43         //copy
44         for(int j = 0; j <= K; ++j)
45             d[0][j] = d[1][j];
46         //~copy
47     }
48
49     fout << d[1][X] << "\n";
50     return 0;
51 }
```

Listing 25.6.5: xnumere-60-adrian.cpp

```

1 #include <iostream>
2 #include <cstdio>
3
4 #define tip long long
5
6 using namespace std;
7
8 struct matrix
9 {
10     tip M[101][101];
11 };
12
13 struct column
14 {
15     tip C[101];
16 };
17
18 column R;
19 matrix A;
20
21 tip n, k, x, MOD=666013;
22
23 matrix operator*(matrix,matrix);
```

```

24 column operator*(matrix, column);
25
26 int main()
27 {
28     freopen("xnumere.in", "r", stdin);
29     freopen("xnumere.out", "w", stdout);
30
31     cin>>k>>x>>n;
32
33     tip i;
34     R.C[1]=k;
35     for(i=1; i<=k; i++) A.M[i][i]=i;
36     for(i=2; i<=k; i++) A.M[i][i-1]=k-i+1;
37     n--;
38     for(; n; n/=2)
39     {
40         if(n%2) R=A*A;
41         A=A*A;
42     }
43     cout<<R.C[x];
44     return 0;
45 }
46
47 matrix operator*(matrix a, matrix b)
48 {
49     matrix r;
50     tip v, i, j, q;
51     for(i=1; i<=k; i++)
52         for(j=1; j<=k; j++)
53         {
54             v=0;
55             for(q=1; q<=k; q++)
56                 v+=a.M[i][q]*b.M[q][j];
57             r.M[i][j]=v%MOD;
58         }
59     return r;
60 }
61
62 column operator*(matrix a, column b)
63 {
64     column r;
65     tip v, i, j;
66     for(i=1; i<=k; i++)
67     {
68         v=0;
69         for(j=1; j<=k; j++)
70             v+=a.M[i][j]*b.C[j];
71         r.C[i]=v%MOD;
72     }
73     return r;
74 }
```

Listing 25.6.6: xnumere-100-mugurel.cpp

```

1 // Mugurel Ionut Andreica
2 // O(X * log(N))
3 #include <stdio.h>
4 #include <string.h>
5
6 #define MOD 666013
7
8 int X, i, iinv;
9 long long K, N, answer, stirling, comb, kprod, xfact, term;
10
11 int ExtendedGCD(int A, int B, int &X, int &Y)
12 {
13     if (B == 0)
14     {
15         X = 1;
16         Y = 0;
17         return A;
18     }
19
20     int X0, Y0, D;
21     D = ExtendedGCD(B, A % B, X0, Y0);
```

```

22     X = Y0;
23     Y = X0 - (A / B) * Y0;
24     return D;
25 }
26
27 long long F[20][20];
28
29 void Brute()
30 {
31     int i, j;
32
33     memset(F, 0, sizeof(F));
34
35     F[0][0] = 1;
36
37     for (i = 1; i <= N; i++)
38         for (j = 1; j <= X; j++)
39         {
40             F[i][j] = 0;
41             F[i][j] = (F[i][j] + F[i - 1][j] * j) % MOD;
42             F[i][j] = (F[i][j] + F[i-1][j-1] * ((K - (j - 1)) % MOD)) % MOD;
43         }
44
45     fprintf(stderr, "brute: %lld\n", F[N][X]);
46 }
47
48 long long Power(long long P, long long Q)
49 {
50     long long answer;
51     if (Q == 0)
52         return 1;
53     if (Q == 1)
54         return (P % MOD);
55     answer = Power(P, Q / 2);
56     answer = (answer * answer) % MOD;
57     if (Q % 2 == 1)
58         answer = (answer * P) % MOD;
59     return answer;
60 }
61
62 int Inv(int i)
63 {
64     int inv, garbage;
65     ExtendedGCD(i, MOD, inv, garbage);
66     inv %= MOD;
67     while (inv < 0)
68         inv += MOD;
69     return inv;
70 }
71
72 int main()
73 {
74     freopen("xnumere.in", "r", stdin);
75     scanf("%lld %d %lld", &K, &X, &N);
76     //Brute();
77
78     xfact = 1;
79     for (i = 2; i <= X; i++)
80         xfact = (xfact * i) % MOD;
81     xfact = Inv(xfact);
82
83     stirling = 0;
84     comb = 1;
85
86     for (i = 1; i <= X; i++)
87     {
88         comb = (comb * ((X - i + 1) % MOD)) % MOD;
89         comb = (comb * Inv(i)) % MOD;
90         term = (comb * Power(i, N)) % MOD;
91         if ((X - i) % 2 == 0)
92             stirling = (stirling + term) % MOD;
93         else
94             stirling = (stirling - term + MOD) % MOD;
95     }
96
97     answer = (stirling * xfact) % MOD;

```

```
98     for (i = 0; i < X; i++)
99         answer = (answer * ((K - i) % MOD)) % MOD;
100
101    freopen("xnumere.out", "w", stdout);
102    printf("%lld\n", answer);
103    return 0;
104 }
```

25.6.3 *Rezolvare detaliată

Capitolul 26

ONI 2012

26.1 search

Problema 1 - search

100 de puncte

Pojărel are o listă de N fișiere, fiecare având numele format doar din litere mici ale alfabetului latin. El vrea să implementeze un motor de căutare care să funcționeze în timp real, adică imediat ce utilizatorul inserează sau șterge o literă din câmpul de căutare, să i se returneze numărul de fișiere care corespund sirului introdus la momentul respectiv. Un fișier corespunde căutării dacă numele acestuia conține sirul căutat ca subșir. Mai precis, caracterele din câmpul de căutare trebuie să apară în numele fișierului în aceeași ordine, însă nu neapărat pe poziții consecutive.

Cerințe

Fiind dată lista care conține numele fișierelor, determinați numărul de fișiere care va fi returnat după fiecare introducere sau ștergere a unui caracter din câmpul de căutare.

Date de intrare

Fișierul de intrare **search.in** conține pe prima linie două numere naturale N și M , reprezentând numărul de fișiere, respectiv numărul de operații care se fac asupra câmpului de căutare. Pe următoarele N linii se găsesc cele N nume de fișiere, formate doar din litere mici ale alfabetului latin. Urmează apoi M linii care descriu operațiile în ordinea în care sunt efectuate. Astfel, pe fiecare linie i se află un singur caracter care descrie operația i . Acest caracter este fie o literă, ceea ce înseamnă că s-a introdus litera respectivă în câmpul de căutare, fie caracterul ' $-$ ', ceea ce înseamnă că s-a șters ultima literă din câmpul de căutare.

Date de ieșire

Fișierul de ieșire **search.out** va conține M linii. Pe linia i ($1 \leq i \leq M$) se va afișa numărul de fișiere returnate de motorul de căutare după efectuarea primelor i operații.

Restricții și precizări

- $1 \leq N \leq 100$;
- $1 \leq M \leq 200\,000$;
- lungimea oricărui nume de fișier este maxim 5 000;
- două sau mai multe fișiere pot avea același nume;
- prima literă introdusă nu va fi ștearsă niciodată.

Exemple:

search.in	search.out	Explicații
4 5	4	- prima dată se introduce litera a, care se găsește în toate cele 4 nume de fișiere
palalila	3	- după a doua operație câmpul va avea valoarea aa și vor fi găsite primele 3 fișiere
alabala	2	- după a treia operație câmpul va avea valoarea aal și vor fi găsite fișierelor palalila și alabala
olimpiada	3	- după a patra operație câmpul va avea din nou valoarea aa și vor fi găsite primele 3 fișiere
iasi	1	- după a cincea operație câmpul va avea valoarea aab și va fi găsit doar fișierul alabala
a		
a		
l		
-		
b		

Timp maxim de executare/test: **0.8** secunde

Memorie: total **64 MB**

Dimensiune maximă a sursei: **10 KB**

26.1.1 Indicații de rezolvare

stud. Vlad Duță - Universitatea București

De fiecare dată când inserăm o literă în câmpul de căutare vom verifica în mod banal pentru fiecare nume de fișier dacă sirul de caractere introdus reprezintă un subșir.

Este împedite ca subșirul trebuie să se termine cât mai devreme în cadrul numelui fișierului, pentru a ne asigura că putem continua căutarea favorabil, fără a pierde din soluții.

Astfel putem determina poziția pe care se termină subșirul cu o clasice abordare de tip *greedy*: pentru fiecare literă din subșir găsim prima apariție de după poziția curentă în cadrul numelui de fișier și mutăm poziția curentă corespunzător.

Pentru a determina în mod eficient poziția în care ne vom muta, va trebui să calculăm următoarea matrice:

$A[i][j][k] =$ prima apariție a literei k în cuvântul i, începând cu poziția j.

De asemenea, pentru a realiza optim operația de stergere a unei litere, vom ține pentru fiecare nume de fișier cate o *stivă* cu pozițiile pe care se termină căutarea.

Când adaugăm o literă, determinăm poziția corespunzătoare și o adăugăm în stivă, iar când stergem o literă e suficient să eliminăm primul element din stivă, astfel ca operația de stergere devine un procedeu trivial.

Complexitatea soluției este $O(N * L * \text{alfabet} + M * N)$.

26.1.2 Cod sursă

Listing 26.1.1: search-cazacu-100.cpp

```

1 #include <cstdio>
2 #include <string.h>
3 #include <algorithm>
4
5 using namespace std;
6
7 #define NMAX 101
8 #define LMAX 5005
9 #define ALPHA 26
10 #define INF 0x3f3f3f3f
11
12 int N, M;
13 char word[NMAX][LMAX];
14 int nextt[NMAX][LMAX][ALPHA];
15 int stack[NMAX][LMAX], sizes;
16
17 void precompute ()
18 {
19     int len;
20
21     for (int i = 0; i <= N; i++)

```

```

22         for (int j = 0; j <= 5000; j++)
23             memset (nextt[i][j], INF, sizeof (nextt[i][j]));
24
25         for (int i = 1; i <= N; i++)
26         {
27             len = strlen (word[i]);
28             for (int j = len - 1; j >= 0; j--)
29                 for (int k = 0; k < 26; k++)
30                 {
31                     nextt[i][j][k] = nextt[i][j+1][k];
32                     if (word[i][j+1] == k + 'a') nextt[i][j][k] = j+1;
33                 }
34         }
35     }
36
37 void solve ()
38 {
39     int rez = 0;
40     char ch;
41
42     sizes = 1;
43     for (; M; M--)
44     {
45         scanf ("%c\n", &ch);
46         if (ch == '-')
47             sizes--;
48         else
49         {
50             for (int i = 1; i <= N; i++)
51                 if (stack[i][sizes] != INF)
52                     stack[i][sizes + 1] = nextt[i][ stack[i][sizes] ][ch - 'a'];
53                 else
54                     stack[i][sizes + 1] = INF;
55             sizes++;
56         }
57
58         rez = 0;
59         for (int i = 1; i <= N; i++)
60             if (stack[i][sizes] != INF) rez++;
61
62         printf ("%d\n", rez);
63     }
64 }
65
66 void readData ()
67 {
68     scanf ("%d %d\n", &N, &M);
69     for (int i = 1; i <= N; i++)
70     {
71         word[i][0] = ' ';
72         scanf ("%s\n", word[i]+1);
73     }
74 }
75
76 int main ()
77 {
78     freopen ("search.in", "r", stdin);
79     freopen ("search.out", "w", stdout);
80
81     readData ();
82     precompute ();
83     solve ();
84
85     return 0;
86 }
```

Listing 26.1.2: vlad_search_100.cpp

```

1 #include <cstdio>
2 #include <cstring>
3
4 #define Nmax 101
5 #define Lmax 5005
6 #define alfa 26
7
```

```

8 int N, M, lvl;
9 char S[Nmax][Lmax];
10 int first[Nmax][Lmax][alfa];
11 int st[Nmax][Lmax];
12
13 void build()
14 {
15     int i, j, l;
16
17     for (i=0; i<N; ++i)
18     {
19         l = strlen(S[i]) - 1;
20
21         memset(first[i][l+1], -1, sizeof(first[i][l+1]));
22         memset(first[i][l], -1, sizeof(first[i][l]));
23         first[i][l][S[i][l]-'a'] = l;
24
25         for (j=l-1; j>=0; --j)
26         {
27             memcpy(first[i][j], first[i][j+1], sizeof(first[i][j+1]));
28             first[i][j][S[i][j]-'a'] = j;
29         }
30     }
31
32     lvl = 0;
33     for (i=0; i<N; ++i)
34         st[i][0] = -1;
35 }
36
37 void go(char C)
38 {
39     int i, j;
40     ++lvl;
41     if (lvl >= Lmax) return;
42
43     for (i=0; i<N; ++i)
44     {
45         if (lvl>1 && st[i][lvl-1] == -1) st[i][lvl] = -1;
46         else st[i][lvl] = first[i][st[i][lvl-1]+1][C];
47     }
48 }
49
50 int main()
51 {
52     int i, nr;
53     char C;
54
55     freopen("search.in", "r", stdin);
56     freopen("search.out", "w", stdout);
57
58     scanf("%d %d\n", &N, &M);
59     for (i=0; i<N; ++i)
60         scanf("%s\n", &S[i]);
61
62     build();
63
64     while (M--)
65     {
66         scanf("%c\n", &C);
67
68         if (C != '-') go(C-'a');
69         else --lvl;
70
71         nr = 0;
72         if (lvl < Lmax)
73             for (int i=0; i<N; ++i)
74             {
75                 if (st[i][lvl] != -1) ++nr;
76             }
77         printf("%d\n", nr);
78     }
79
80     return 0;
81 }
```

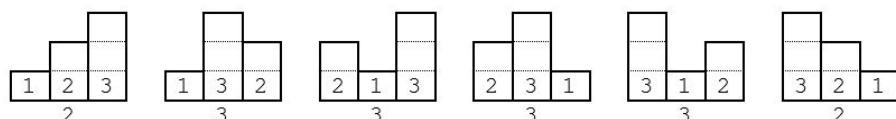
26.1.3 *Rezolvare detaliată

26.2 urat

Problema 2 - urat

100 de puncte

Aveți la dispoziție n scânduri de înălțimi $1, 2, 3, \dots, n$. Vrem să construim un gard așezând scândurile una lângă alta într-o ordine întâmplătoare. De exemplu, dacă $n = 3$, putem să construim gardul în 6 moduri:



Pentru orice tip de gard se calculează diferențele în valoare absolută dintre înălțimile oricărora două scânduri vecine din gard. Suma acestor diferențe se numește gradul de urătenie al gardului. În exemplul anterior, pentru $n = 3$, se observă că gardurile au în 4 cazuri gradul de urătenie egal cu 3 și în 2 cazuri au gradul de urătenie egal cu 2.

Cerințe

Cunoscând numărul n de scânduri realizați un program care:

- calculează gradul maxim de urătenie pe care îl poate avea un gard de n scânduri;
- calculează restul modulo 543217 al numărului de garduri cu grad maxim de urătenie care se pot construi cu cele n scânduri;
- determină un gard cu grad maxim de urătenie format din n scânduri, sub forma unei permutări de ordin n .

Date de intrare

Fișierul **urat.in** conține pe prima linie numărul natural n reprezentând numărul de scânduri.

Date de ieșire

Fișierul **urat.out** va conține trei linii:

- pe prima linie se va scrie un număr natural reprezentând gradul maxim de urătenie al unui gard format din n scânduri;
- pe a doua linie se va scrie un număr natural reprezentând restul modulo 543217 al numărului de garduri cu grad maxim de urătenie care se pot construi folosind cele n scânduri;
- pe a treia linie se vor scrie n numere naturale, oricare două consecutive separate prin câte un spațiu, reprezentând, în ordine de la stânga spre dreapta, înălțimile scândurilor dintr-un gard cu grad maxim de urătenie format cu cele n scânduri.

Restricții și precizări

- $1 < n \leq 500\,000$
- Pentru prima cerință se acordă 20% din punctaj, pentru a doua 60% iar pentru a treia 20%

Exemple:

urat.in	urat.out	Explicații
3	3 4 1 3 2	Gradul maxim de urătenie este 3. Există 4 tipuri de garduri cu grad maxim de urătenie dintre cele 6 posibile - vezi figura. Unul dintre gardurile de urătenie maximă folosește, de la stânga la dreapta, scândurile (1,3,2) - vezi al doilea gard din figură.

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

Memorie: total **64 MB**

Dimensiune maximă a sursei: **10 KB**

26.2.1 Indicații de rezolvare

prof. Adrian Panaete - C.N. "A.T.Laurian" Botoșani

Se observă că înălțimile scândurilor unui gard formează o permutare.

Problema cere de fapt permutările care au suma diferențelor între 2 termeni consecutivi maximă.

Fie p_1, p_2, \dots, p_n o permutare și $m = (n+1)/2$ media aritmetică a înălțimii gardurilor.

Notăm cu $G[p]$ gradul de urâtenie al gardului.

$$G[p] = |p_1 - p_2| + |p_2 - p_3| + \dots + |p_{i-1} - p_i| + |p_i - p_{i+1}| + \dots + |p_{n-1} - p_n|.$$

Se observă că oricare termen al permutării cu excepția lui p_1 și p_n intervin în exact 2 dintre cei $n-1$ termeni ai sumei.

Fie $p_i = x$ și $p_{i+1} = y$ avem 6 cazuri

1. $x \leq y \leq m \Rightarrow |x-y| = y-x = m-x-(m-y) = +|m-x|-|m-y|$
2. $x \leq m \leq y \Rightarrow |x-y| = y-x = m-x+(y-m) = +|m-x|+|m-y|$
3. $m \leq x \leq y \Rightarrow |x-y| = y-x = y-m-(x-m) = -|m-x|+|m-y|$
4. $y \leq x \leq m \Rightarrow |x-y| = x-y = m-y-(m-x) = -|m-x|+|m-y|$
5. $y \leq m \leq x \Rightarrow |x-y| = x-y = x-m+(m-y) = +|m-x|+|m-y|$
6. $m \leq y \leq x \Rightarrow |x-y| = x-y = x-m-(y-m) = +|m-x|-|m-y|$

Deducem că pentru fiecare termen din sumă putem scrie $|x-y| = \pm |m-x| \pm |m-y|$.

Deci

$$\begin{aligned} G[p] &= \pm|m-p_1| \pm|m-p_2| \pm|m-p_3| \pm\dots\pm|m-p_{n-2}| \pm|m-p_{n-1}| \pm|m-p_{n-1}| \pm|m-p_n| \\ &\leq |m-p_1| + |m-p_2| + |m-p_3| + \dots + |m-p_{n-2}| + |m-p_{n-1}| + |m-p_{n-1}| + |m-p_n| \\ &= 2 * (|m-p_1| + |m-p_2| + |m-p_3| + \dots + |m-p_n|) - |m-p_1| - |m-p_n| \\ &= 2 * (|m-1| + |m-2| + |m-3| + \dots + |m-n|) - |m-p_1| - |m-p_n| \end{aligned}$$

Pentru a maximiza valoarea se observă că p_1 și p_n ar trebui alese cele mai apropiate de medie.

Pentru $n = 2 * k$ (n par) media este $m = (n+1)/2 = k + 1/2$ deci p_1 și p_n se vor alege k și $k+1$.

Pentru $n = 2 * k + 1$ (n impar) media este $m = (n+1)/2 = k + 1$ deci una dintre valorile p_1 și p_n va fi obligatoriu $k+1$ iar cealaltă va fi una dintre valorile k sau $k+2$.

Se deduce că pentru orice n (par sau impar) un majorant al lui $G[p]$ independent de permutare va fi $MAX = 2 * (|m-1| + |m-2| + |m-3| + \dots + |m-n|) - 1$.

Cerința a.

Pentru ca o permutare să aibă $G[p] = MAX$ va trebui să indeplinească două condiții

I. Oricare doi termeni consecutivi să se găsească în cazurile 2 sau 4 din cele șase cazuri descrise mai sus \Rightarrow permutarea trebuie să alterneze valorile mai mici decât media cu valorile mai mari decât media

II. Primul și ultimul termen din permutare să fie alese cele mai apropiate posibil de media $m = (n+1)/2$

Există permutări cu aceste proprietăți deci MAX este valoarea cerută la punctul a. al problemei.

Definitivând calculul lui MAX se poate deduce formula $MAX = [n^2/2] - 1$

Cerința b.

In cazul $n = 2 * k$ avem 2 tipuri de permutări

Cele care au $p_1 = k$ și $p_n = k+1$ se obțin prin permutarea valorilor $1, 2, 3, \dots, k-1$ pe pozițiile impare și a valorilor $k+2, k+3, \dots, 2*k$ pe pozițiile pare.

Cele care au $p_1 = k+1$ și $p_n = k$ se obțin prin permutarea valorilor $1, 2, 3, \dots, k-1$ pe pozițiile pare și a valorilor $k+2, k+3, \dots, 2*k$ pe pozițiile impare.

Cum de fiecare tip se vor obține $(k-1)! * (k-1)!$ soluția va fi $2 * (k-1)! * (k-1)!$

In cazul $n = 2 * k + 1$ avem 4 tipuri de permutări

Cele care au $p_1 = k$ și $p_n = k+1$ se obțin prin permutarea valorilor $1, 2, 3, \dots, k-1$ pe pozițiile impare și a valorilor $k+2, k+3, \dots, 2*k, 2*k+1$ pe pozițiile pare.

Cele care au $p_1 = k+1$ și $p_n = k$ se obțin prin permutarea valorilor $1, 2, 3, \dots, k-1$ pe pozițiile impare și a valorilor $k+2, k+3, \dots, 2*k, 2*k+1$ pe pozițiile pare.

Cele care au $p_1 = k+2$ și $p_n = k+1$ se obțin prin permutarea valorilor $1, 2, 3, \dots, k-1$ pe pozițiile pare și a valorilor $k+2, k+3, \dots, 2*k, 2*k+1$ pe pozițiile impare.

Cele care au $p_1 = k + 1$ și $p_n = k + 2$ se obțin prin permutarea valorilor $1, 2, 3, \dots, k-1$ pe pozițiile pare și a valorilor $k+2, k+3, \dots, 2*k, 2*k+1$ pe pozițiile impare.

Cum de fiecare tip se vor obține $k! * (k - 1)!$ soluția va fi $4 * k! * (k - 1)!$

Cerință c.

Pentru $n = 2 * k$ (par) un exemplu de permutare cu $G[p] = MAX$ este
 $k, k+2, 1, k+3, 2, k+4, 3, \dots, 2*k-1, k-2, 2*k, k-1, k+1$

Pentru $n = 2 * k + 1$ (impar) un exemplu este

$k, k+2, 1, k+3, 2, k+4, 3, \dots, 2*k-1, k-2, 2*k, k-1, 2*k+1, k+1$

26.2.2 Cod sursă

Listing 26.2.1: uratAdrian100.cpp

```

1 #include<cstdio>
2 #include<iostream>
3
4 using namespace std;
5
6 long long n,i,k,r,Gr,Nr,lo,hi,MOD=543217;
7
8 int main()
9 {
10     freopen("urat.in","r",stdin);
11     freopen("urat.out","w",stdout);
12
13     cin>>n;
14
15     k=n/2;
16     r=n%2;
17     Gr=(n*n)/2-1;
18
19     cout<<Gr<<'\\n';
20
21     Nr=1;
22     lo=(n-2)/2;
23     hi=n-2-lo;
24     for(i=1;i<=lo;i++)
25         Nr=(Nr*i)%MOD;
26
27     Nr=(Nr*Nr)%MOD;
28     if(lo<hi)
29         Nr=(Nr*4*hi)%MOD;
30     else
31         Nr=(2*Nr)%MOD;
32
33     cout<<Nr<<'\\n';
34     cout<<k<<' ';
35
36     for(i=1;i<k;i++)
37         cout<<k+i+1<<' '<<i<<' ';
38
39     if(r)
40         cout<<n<<' ';
41
42     cout<<k+1<<'\\n';
43
44     return 0;
45 }
```

Listing 26.2.2: uratBKAdrian20.cpp

```

1 #include<cstdio>
2 #include<iostream>
3 #include<vector>
4 #include<algorithm>
5
6 using namespace std;
7
8 vector<int> x,y;
```

```

9 int n,i,k;
10 long long Uratenie,uratenie,cate;
11
12 int main()
13 {
14     freopen("urat.in","r",stdin);
15     freopen("urat.out","w",stdout);
16
17     cin>>n;
18     for(i=1;i<=n;i++) x.push_back(i);
19
20     for(;;)
21     {
22         for(uratenie=0,i=1;i<n;i++)
23             uratenie+=x[i]>x[i-1]?x[i]-x[i-1]:x[i-1]-x[i];
24
25         if(uratenie==Uratenie)
26             cate++;
27         else
28             if(uratenie>Uratenie)
29             {
30                 Uratenie=uratenie;
31                 y=x;
32                 cate=1;
33             }
34         k=next_permutation(x.begin(),x.end());
35         if(!k) break;
36     }
37
38     cout<<Uratenie<<'\n';
39     cout<<cate%543217<<'\n';
40
41     for(i=0;i<n;i++) cout<<y[i]<<' ';
42     return 0;
43 }
```

26.2.3 *Rezolvare detaliată

26.3 zlego

Problema 3 - zlego

100 de puncte

Recent, Bujorel a dat în mintea copiilor și s-a apucat să se joace cu piese de zlego. El are o piesă formată din N bucăți numerotate de la 1 la N , fiecare cu o înălțime și un coeficient de frumusețe date. Se definește un *zprefix* ca fiind un sir format din una sau mai multe bucăți consecutive care să înceapă cu piesa 1. Scopul este să aleagă un zprefix, să găsească toate aparitiiile acestuia în restul piesei și să facă suma costurilor de frumusețe ale acestor aparitii. Pentru o apariție a zprefixului ales care corespunde sevenței formate din bucățile numerotate de la i la j (înălțimea bucății 1 este egală cu înălțimea bucății i , înălțimea bucății 2 este egală cu înălțimea bucății $i+1$ etc.) costul său de frumusețe este coeficientul de frumusețe al ultimei bucăți, adică al lui j .

Bujorel e curios ce se întamplă pentru orice zprefix și vrea să afișeze suma costurilor de frumusețe ale tuturor aparitiilor fiecarui zprefix.

Cerințe

Date de intrare

Pe prima linie a fisierului **zlego.in** se află numarul de teste T . Fiecare test are următorul format: pe prima linie un număr natural N reprezentând dimensiunea piesei de zlego, pe următoarea linie se află N numere întregi, reprezentând înălțimile pieselor, despărțite prin câte un spațiu, iar pe cea de-a treia linie se află tot N numere întregi, despărțite prin câte un spațiu, cel de-al i -lea număr reprezentând coeficientul de frumusețe a celei de-a i -a piese.

Date de ieșire

Fisierul **zlego.out** trebuie să conțină, pentru fiecare din cele T teste, câte N linii, cea de-a i -a linie reprezentând suma costurilor de frumusețe ale aparitiilor zprefixului $[1, i]$.

Restricții și precizări

- $1 \leq N \leq 250.000$;
- $1 \leq T \leq 3$;
- Înălțimile și coeficienții de frumusețe ale bucătăilor piesei se încadreaza pe 32 de biti cu semn;
- Pentru 20% din teste $N \leq 100$;
- Pentru 50% din teste $N \leq 1000$;
- Atenție!: Bujorel recomandă tipuri de date pe 64 de biți pentru afișarea rezultatului.

Exemple:

zlego.in	zlego.out	Explicații
2	2	
3	2	
1 2 1	4	
2 2 2	44	
10	30	
1 1 2 1 1 1 2 1 1	11	
1 2 3 4 5 6 7 8 9 10	13	
	15	În cel de-al doilea test, pentru zprefixul [1, 1] obținem suma costurilor de frumusete ale aparițiilor acestuia $44 = 1+2+4+5+6+7+9+10$.
	6	
	7	
	8	
	9	
	10	

Timp maxim de executare/test: **1.2** secunde

Memorie: total **64 MB**

Dimensiune maximă a sursei: **10 KB**

26.3.1 Indicații de rezolvare

stud. Andrei Pârvu - Universitatea "Politehnica" București

Soluția 1 - 20 de puncte

Pentru fiecare zprefix de forma $[1,i]$ se calculează aparițiile sale folosind un algoritm trivial de potrivire de șiruri (într-o complexitate de $O(N^2)$) calculăm suma costurilor de frumusețe.

Complexitate finală: $O(N^3)$.

Soluția 2 - 50 de puncte

Aceasta soluție este asemănătoare cu cea de 20 de puncte, cu excepția faptului că se înlocuiește algoritmul trivial de potrivire de șiruri cu unul clasic de complexitate $O(N)$ (*KMP*, *Rabin-Karp* etc.).

Complexitate finală: $O(N^2)$.

Soluția 3 - 100 de puncte

Vom folosi funcția prefix a algoritmului KMP pentru a rezolva problema într-o complexitate liniară.

Reamintim că funcția prefix calculează un vector $\text{pi}[i]$ - cel mai lung *prefix* al secvenței $[1,i]$ care este și *sufix* al acesteia.

Facem observația că acest vector de prefix se comportă ca un *vector de tați* într-un arbore, adică se poate lega o mulțime între $\text{pi}[i]$ și i . Astfel, pentru un zprefix dat, aparițiile sale în tot șirul se află în subarborele său. Din acest motiv, putem ține o dinamică evidentă de forma

$D[i]$ - suma costurilor de frumusețe pentru nodul i ,
a cărei rezultate le afișăm la ieșire.

Complexitate finală: $O(N)$.

26.3.2 Cod sursă

Listing 26.3.1: zlego.cpp

```

1  /* Andrei Parvu
2   O(N) */
3 #include <cstdio>
4 #include <cstdlib>
5 #include <cassert>
6 #include <vector>
7
8 using namespace std;
9
10#define NMAX 250000
11
12int pi[NMAX + 5], cost[NMAX + 5], sir[NMAX + 5];
13long long sol[NMAX + 5], d[NMAX + 5];
14int n, teste;
15vector<int> v[NMAX + 5];
16
17void prefix()
18{
19    int x = 0, i;
20    v[0].push_back(1);
21
22    for (i = 2; i <= n; i++)
23    {
24        for (; x > 0 && sir[i] != sir[x + 1]; x = pi[x]);
25
26        if (sir[i] == sir[x + 1])
27            x++;
28
29        pi[i] = x;
30        v[x].push_back(i);
31    }
32}
33
34void df(int nod)
35{
36    int i;
37    d[nod] = cost[nod];
38
39    for (i = 0; i < (int)v[nod].size(); i++)
40    {
41        df(v[nod][i]);
42        d[nod] += d[v[nod][i]];
43    }
44
45    sol[nod] = d[nod];
46}
47
48int main()
49{
50    freopen("zlego.in", "rt", stdin);
51    freopen("zlego.out", "wt", stdout);
52
53    assert(scanf("%d", &teste));
54    assert(1 <= teste && teste <= 3);
55
56    for (; teste--; )
57    {
58        assert(scanf("%d", &n));
59        assert(1 <= n && n <= NMAX);
60
61        for (int i = 1; i <= n; i++)
62            v[i].clear();
63
64        for (int i = 1; i <= n; i++)
65            assert(scanf("%d", &sir[i]));
66
67        for (int i = 1; i <= n; i++)
68            assert(scanf("%d", &cost[i]));
69
70    prefix();

```

```

71     df(0);
72
73     for (int i = 1; i <= n; i++)
74         printf("%lld\n", sol[i]);
75     }
76
77     return 0;
78 }

```

Listing 26.3.2: zlego-20.cpp

```

1  /* Andrei Parvu
2   O(N^3) */
3 #include <cstdio>
4
5 const int NMAX = 250000;
6
7 int sir[NMAX + 5], cost[NMAX + 5], n, teste;
8
9 int main()
10 {
11     freopen("zlego.in", "rt", stdin);
12     freopen("zlego.out", "wt", stdout);
13
14     scanf("%d", &teste);
15
16     for (; teste--; )
17     {
18         scanf("%d", &n);
19
20         for (int i = 1; i <= n; i++)
21             scanf("%d", sir + i);
22
23         for (int i = 1; i <= n; i++)
24             scanf("%d", cost + i);
25
26         for (int i = 1; i <= n; i++)
27         {
28             long long suma = 0;
29
30             for (int j = 1; j <= n - i + 1; j++)
31             {
32                 bool prost = false;
33
34                 for (int k = 1; k <= i; k++)
35                     if (sir[k] != sir[j + k - 1])
36                     {
37                         prost = true;
38                         break;
39                     }
40
41                 if (!prost)
42                     suma += cost[j + i - 1];
43             }
44
45             printf("%lld\n", suma);
46         }
47     }
48
49     return 0;
50 }

```

Listing 26.3.3: zlego-50.cpp

```

1  /* Andrei Parvu
2   O(N^2) */
3 #include <cstdio>
4
5 const int NMAX = 250000;
6
7 int sir[NMAX + 5], cost[NMAX + 5], n, pi[NMAX + 5], teste;
8
9 void prefix()

```

```

10  {
11      int x = 0, i;
12
13      for (i = 2; i <= n; i++)
14      {
15          for (; x > 0 && sir[i] != sir[x + 1]; x = pi[x]);
16
17          if (sir[i] == sir[x + 1])
18              x++;
19
20          pi[i] = x;
21      }
22  }
23
24 int main()
25 {
26     freopen("zlego.in", "rt", stdin);
27     freopen("zlego.out", "wt", stdout);
28
29     scanf("%d", &teste);
30
31     for (; teste--; )
32     {
33         scanf("%d", &n);
34
35         for (int i = 1; i <= n; i++)
36             scanf("%d", sir + i);
37
38         for (int i = 1; i <= n; i++)
39             scanf("%d", cost + i);
40
41         prefix();
42
43         for (int i = 1; i <= n; i++)
44         {
45             int x = 0;
46             long long suma = 0;
47
48             for (int j = 1; j <= n; j++)
49             {
50                 for (; x > 0 && (sir[j] != sir[x + 1] || x + 1 > i); )
51                     x = pi[x];
52
53                 if (sir[j] == sir[x + 1])
54                     x++;
55
56                 if (x == i)
57                     suma += cost[j];
58             }
59
60             printf("%lld\n", suma);
61         }
62     }
63
64     return 0;
65 }
```

Listing 26.3.4: zlego-50-adrian.cpp

```

1 #include<cstdio>
2
3 #define N 250010
4
5 using namespace std;
6
7 long long c[N], val;
8 int n, i, k, m, L, x[N], Q[N];
9
10 void read(), solve(), prefix();
11
12 int main()
13 {
14     read();
15     return 0;
16 }
```

```

17
18 void read()
19 {
20     freopen("zlego.in", "r", stdin);
21     freopen("zlego.out", "w", stdout);
22
23     int teste;
24
25     scanf("%d", &teste);
26     while (teste--)
27     {
28         scanf("%d", &n);
29         for(i=1;i<=n;i++) scanf("%d", &x[i]);
30         for(i=1;i<=n;i++) scanf("%lld", &c[i]);
31
32         solve();
33     }
34 }
35
36 void solve()
37 {
38     for(i=1;i<=n;i++)
39         Q[i]=i;
40     m=n;
41     for(L=1;L<=n;L++)
42     {
43         for(i=1;i<=m;i++)
44         {
45             if(Q[i]+L-1>n) break;
46             if(x[Q[i]+L-1]==x[L])
47             {
48                 val+=c[Q[i]+L-1];
49                 Q[++k]=Q[i];
50             }
51         }
52         printf("%lld\n",val);
53         m=k; k=0; val=0;
54     }
55 }
```

Listing 26.3.5: zlego-50-mugurel.cpp

```

1 #include <stdio.h>
2
3 #define NMAX 250001
4
5 int v[NMAX], c[NMAX], p[NMAX];
6 int i, j, N;
7
8 int main()
9 {
10     freopen("zlego.in", "r", stdin);
11     freopen("zlego.out", "w", stdout);
12
13     int teste;
14
15     scanf("%d", &teste);
16
17     while (teste--)
18     {
19         scanf("%d", &N);
20         for (i = 1; i <= N; i++)
21             scanf("%d", &v[i]);
22         for (i = 1; i <= N; i++)
23             scanf("%d", &c[i]);
24
25         // functia prefix de la KMP
26         p[1] = j = 0;
27         for (i = 2; i <= N; i++)
28         {
29             while (v[j + 1] != v[i] && j > 0)
30                 j = p[j];
31             if (v[j + 1] == v[i])
32                 j++;
33             p[i] = j;
```

```

34      }
35
36     long long sc[NMAX] = {0};
37
38     for (i = 1; i <= N; i++)
39     {
40         sc[i] = c[i];
41         j = p[i];
42         while (j > 0)
43         {
44             sc[j] += c[i];
45             j = p[j];
46         }
47     }
48
49     // afisare
50     for (i = 1; i <= N; i++)
51     printf("%lld\n", sc[i]);
52 }
53
54     return 0;
55 }
```

Listing 26.3.6: zlego-100-adrian.cpp

```

1 #include<cstdio>
2
3 #define N 250010
4
5 using namespace std;
6
7 long long c[N];
8 int n,x[N],Pi[N];
9
10 void read(),solve(),prefix();
11 int main()
12 {
13     read();
14     return 0;
15 }
16
17 void read()
18 {
19     freopen("zlego.in","r",stdin);
20     freopen("zlego.out","w",stdout);
21     int teste;
22
23     scanf("%d", &teste);
24
25     while (teste--)
26     {
27         scanf("%d",&n);
28         for(int i=1;i<=n;i++) scanf("%d",&x[i]);
29         for(int i=1;i<=n;i++) scanf("%lld",&c[i]);
30
31         solve();
32     }
33 }
34
35 void solve()
36 {
37     prefix();
38     for(int i=n;i>=1;i--) c[Pi[i]]+=c[i];
39     for(int i=1;i<=n;i++) printf("%lld\n",c[i]);
40 }
41
42 void prefix()
43 {
44     int i,q=0,Pi[1]=0;
45     for(i=2;i<=n;++i)
46     {
47         while(q>0&&x[q+1]!=x[i]) q=Pi[q];
48         if(x[q+1]==x[i]) q++;
49         Pi[i]=q;
50     }
51 }
```

51 }

Listing 26.3.7: zlego-100-bogdan.cpp

```

1 /*
2  * (C) 2012 Bogdan-Cristian TĂĂtĂĂcroiu
3  */
4 #include <iostream>
5 #include <iomanip>
6 #include <assert>
7
8 using namespace std;
9
10 const int MAXN = 250000;
11 int N, x[MAXN], p[MAXN], c[MAXN];
12 long long sum[MAXN];
13
14 int main()
15 {
16     assert(freopen("zlego.in", "rt", stdin));
17     assert(freopen("zlego.out", "wt", stdout));
18
19     int T;
20     assert(scanf("%d", &T) == 1);
21     for (; T; T--)
22     {
23         assert(scanf("%d", &N) == 1);
24         assert(1 <= N && N <= MAXN);
25         for (int i = 0; i < N; i++)
26             assert(scanf("%d", x + i) == 1);
27
28         for (int i = 0; i < N; i++)
29             assert(scanf("%d", c + i) == 1);
30
31         memset(sum, 0, sizeof(sum));
32
33         p[0] = -1;
34         for (int i = 1; i < N; i++)
35         {
36             for (p[i] = p[i - 1];
37                  p[i] >= 0 && x[i] != x[p[i] + 1];
38                  p[i] = p[p[i]]);
39
40             if (x[i] == x[p[i] + 1])
41                 p[i] += 1;
42         }
43
44         for (int i = N - 1; i >= 0; i--)
45         {
46             if (p[i] == -1)
47                 continue;
48
49             sum[p[i]] += sum[i] + c[i];
50         }
51
52         for (int i = 0; i < N; i++)
53             printf("%lld\n", sum[i] + c[i]);
54     }
55
56     return 0;
57 }
```

Listing 26.3.8: zlego-100-mugurel.cpp

```

1 #include <iostream>
2 #include <iomanip>
3
4 #define NMAX 250001
5
6 int v[NMAX], c[NMAX], p[NMAX];
7 int i, j, N, teste;
8 long long sc[NMAX];
9
10 int main()
```

```

11  {
12      freopen("zlego.in", "r", stdin);
13      freopen("zlego.out", "w", stdout);
14
15      scanf("%d", &teste);
16
17      while (teste--)
18      {
19          scanf("%d", &N);
20          for (i = 1; i <= N; i++)
21              scanf("%d", &v[i]);
22          for (i = 1; i <= N; i++)
23              scanf("%d", &c[i]);
24
25          // functia prefix de la KMP
26          p[1] = j = 0;
27          for (i = 2; i <= N; i++)
28          {
29              while (v[j + 1] != v[i] && j > 0)
30                  j = p[j];
31              if (v[j + 1] == v[i])
32                  j++;
33              p[i] = j;
34          }
35
36          memset(sc, 0, sizeof(sc));
37          for (i = N; i >= 1; i--)
38          {
39              sc[i] += c[i];
40              sc[p[i]] += sc[i];
41          }
42
43          // afisare
44          for (i = 1; i <= N; i++)
45              printf("%lld\n", sc[i]);
46      }
47
48      return 0;
49 }

```

26.3.3 *Rezolvare detaliată

26.4 drumuri

Problema 4 - drumuri

100 de puncte

Se consideră un graf neorientat având inițial N noduri izolate numerotate de la 1 la N (deci graful are inițial 0 muchii). Asupra acestui graf se pot efectua 4 tipuri de operații, precum adăugarea unei muchii noi, stergerea unei muchii și două tipuri de interogări.

Există însă o constrângere: la orice moment de timp, componentele conexe ale grafului trebuie să aibă structura unui drum (adică fiecare nod are cel mult 2 vecini și graful nu conține cicluri). Cele 4 tipuri de operații sunt următoarele:

- Tipul 1: Adaugă o muchie între nodurile i și j . Această operație se efectuează cu succes dacă nu încalcă constrângerea. În caz contrar, operația nu va fi efectuată.
- Tipul 2: Elimină muchia dintre nodurile i și j . Această operație se efectuează cu succes dacă există muchie între nodurile i și j .
- Tipul 3: Determină toate nodurile aflate la distanță D de nodul i . Distanța dintre două noduri este egală cu numărul minim de muchii al unui drum în graf ce unește cele două noduri. Din cauza constrângerii, pot exista cel mult două noduri la distanță D față de un alt nod.
- Tipul 4: Determină nodurile-capăt ale drumului (componentei conexe) din care face parte nodul i (i ar putea fi chiar unul din capete sau singurul capăt, dacă el este un nod izolat).

Cerințe

Dându-se o secvență de M operații și pornind de la un graf fără nicio muchie, afișați rezultatul fiecărei operații din secvență (în ordinea în care acestea sunt date).

Date de intrare

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

- Pe prima linie numerele N și M separate, printr-un spațiu.
- Pe fiecare din următoarele M linii este descrisă o operație, printr-o succesiune de numere întregi separate prin spații. Primul număr de pe linie reprezintă tipul operației (de la 1 la 4).
 - Dacă operația este de tipul 1, atunci urmează două numere întregi i și j , având semnificația că se va adăuga o muchie între nodurile i și j din graf (dacă nu se încalcă constrângerea).
 - Dacă operația este de tipul 2, atunci urmează două numere întregi i și j , având semnificația că se va șterge muchia dintre nodurile i și j din graf (dacă există).
 - Dacă operația este de tipul 3, atunci urmează două numere întregi i și D , având semnificația că se dorește determinarea tuturor nodurilor aflate la distanță exact D de nodul i .
 - Dacă operația este de tipul 4, atunci urmează un singur număr întreg i , având semnificația că dorim să determinăm capetele drumului din care face parte nodul i .

Date de ieșire

Fișierul de ieșire **drumuri.out** va conține M linii. A i -a linie va conține rezultatul celei de-a i -a operații din fișierul de intrare. Dacă operația este de tipul 1 sau 2, rezultatul său este 1, dacă operația s-a efectuat cu succes, sau 0, în caz contrar.

În cazul operațiilor de tipul 3 și 4, rezultatul constă dintr-o mulțime de noduri. Afisați mai întâi numărul de noduri din mulțime, apoi, pe aceeași linie, elementele mulțimii în ordine crescătoare.

Restricții și precizări

- $1 \leq N \leq 40\,000$
- $1 \leq M \leq 400\,000$
- $0 \leq D \leq N - 1$
- $1 \leq i, j \leq N$
- Nu se vor adăuga muchii de la un nod la el însuși.
- Pentru 40% din teste $N \leq 5\,000$ și $M \leq 20\,000$.

Exemple:

drumuri.in	drumuri.out	Explicații
6 14	1	Graful are $N=6$ noduri și se dau $M=14$ operații:
1 2 4	1	1) Se adaugă muchie între nodurile 2 și 4 cu succes.
1 6 5	1 1	2) Se adaugă muchie între nodurile 6 și 5 cu succes.
4 1	1	3) Nodul 1 face parte dintr-un drum cu un singur nod: 1.
1 1 5	0	4) Se adaugă muchie între nodurile 1 și 5 cu succes.
1 1 6	2 1 6	5) Adăugarea muchiei între nodurile 1 și 6 eșuează, deoarece s-ar forma un ciclu în graf.
4 5	1 5	6) Nodul 5 face parte dintr-un drum cu 2 capete: 1 și 6.
3 6 1	2 2 4	7) Există un singur nod la distanță 1 de nodul 6: nodul 5.
4 4	1 3	8) Nodul 4 face parte dintr-un drum cu 2 capete: 2 și 4.
4 3	0	9) Nodul 3 face parte dintr-un drum cu un singur nod: 3.
3 3 1	1 3	10) Există zero noduri la distanță 1 de nodul 3.
3 3 0	0	11) Există un singur nod la distanță 0 de nodul 3: nodul 3.
2 3 2	1	12) Ștergerea muchiei dintre nodurile 3 și 2 eșuează, deoarece această muchie nu există.
2 5 6	0	13) Ștergerea muchiei dintre nodurile 5 și 6 se efectuează cu succes.
3 6 1		14) Există zero noduri la distanță 1 de nodul 6.

Timp maxim de executare/test: **6.5** secunde

Memorie: total **64 MB**

Dimensiune maximă a sursei: **20 KB**

26.4.1 Indicații de rezolvare

S.I. dr. ing. Mugurel Ionut Andreica - Universitatea "Politehnica" Bucuresti

Solutia 1 - 40 puncte

Se mentin drumurile sub forma de vectori, impreuna cu informatii asociate fiecarui nod referitor la drumul din care face parte si la pozitia sa in drum. Adaugarea unei muchii presupune concatenarea a doua drumuri (eventual in sens inversat), iar stergerea unei muchii presupune separarea unui drum in doua drumuri.

Cu ajutorul acestor informatii mentionate, se poate verifica in timp $O(1)$ daca adaugarea sau stergerea unei muchii sunt valide, respectiv se poate raspunde in timp $O(1)$ la fiecare interogare (de ex., daca nodul i se afla pe pozitia x in drum, iar drumul este retinut ca un vector, atunci nodurile aflate la distanta D de nodul i se afla pe pozitiile $x-D$, respectiv $x+D$, din cadrul vectorului). Concatenarea a doua drumuri si separarea unui drum in doua sunt operatii ce se executa in timp $O(N)$.

Solutia 2 - 40 puncte

Se mentine graful folosind o reprezentare "standard" a grafului (de ex., liste de vecini).

Pentru fiecare operatie putem parcurge graful (in timp $O(N)$), caci exista $O(N)$ noduri si muchii pentru a decide daca operatia se poate efectua, respectiv pentru a raspunde la interogare.

Solutia 3 - 40 puncte

Se mentin pentru fiecare nod cei maxim 2 vecini din graf.

In plus, daca un nod este capat de drum, putem memora si capatul opus al aceluiasi drum.

La adaugarea unei muchii setam cele doua noduri ca fiind vecine unul cu altul.

Muchia este adaugata daca ambele noduri sunt capete de drumuri (au gradul cel mult egal cu 1) si nu sunt capetele aceluiasi drum. La stergerea unei muchii "resetam" vecinul corespunzator fiecaruia din cele doua noduri la "inexistent" (si, eventual, calculam capatul opus al fiecaruia din cele 2 noduri, care au devenit acum capete de drumuri).

Pentru a raspunde la o interogare vom pleca din nodul i si vom parcurge drumul din care face parte in ambele sensuri, ori pana ajungem la capatul sau, ori pana am parcurs o distanta D.

Parcurgerea drumului presupune atentie la implementare, deoarece, in cadrul unui drum, nu putem mentine in mod consistent "vecinul stanga", respectiv "vecinul dreapta" al fiecarui nod (decat daca efectuam procesari suplimentare la adaugarea unei muchii, adica la concatenarea a doua drumuri).

Din fericire, mentionarea consistenta a vecinilor "stanga" si "dreapta" in cadrul unui drum nu este necesara.

Cand parcurgem drumul intr-un sens, mentionem atat nodul curent, cat si nodul anterior.

Apoi, urmatorul nod ce va fi parcurs este acel vecin al nodului curent care este diferit de cel anterior.

Aceasta solutia are complexitatea $O(N)$ pentru fiecare interogare si $O(1)$ pentru un tip de modificare a grafului si $O(N)$ pentru celalalt tip de modificare a grafului.

Solutia 4 - 100 puncte

Vom extinde solutia anterioara pentru a mentine, pentru fiecare nod, pe langa cei maxim 2 vecini, inca maxim 2 K-vecini, adica cele maxim 2 noduri aflate la distanta exact K de nodul respectiv.

Daca notam cei doi vecini ai nodului i prin $v1[i]$ si $v2[i]$ si cei doi K-vecini ai nodului prin $kv1[i]$ si $kv2[i]$, vom avea proprietatea ca $kv1[i]$ este K-vecinul nodului i din acelasi sens ca si vecinul $v1[i]$ (si similar pentru $kv2[i]$ si $v2[i]$).

La adaugarea unei muchii maxim K noduri de fiecare parte a muchiei vor obtine K-vecini noi in sensul dinspre muchia adaugata. La stergerea unei muchii, maxim K noduri de fiecare parte a muchiei isi vor pierde K-vecinii din sensul inspre muchia stearsa.

Folosind K-vecinii, putem raspunde la interogari in timp $O(K+N/K)$.

Mai intai vom "sari" din K in K, pana cand un nou salt de lungime K ar "sari" prea departe (ori peste distanta D, ori in afara drumului). Apoi vom continua sa ne deplasam din 1 in 1 (din vecin in vecin) pana ajungem la distanta D sau la capatul drumului.

Si acest mod de parcurgere necesita aceeasi atentie la implementare din solutia anterioara (intrucat nu mentionem in mod consistent vecini si K-vecini "stanga" si "dreapta", urmatorul vecin/K-vecin ce va fi parcurs este acel vecin/K-vecin al nodului curent care este diferit de vecinul/K-vecinul parcurs anterior).

Daca alegem $K=O(\sqrt{N})$, atunci complexitatea de timp a fiecarei operatii este $O(\sqrt{N})$.

26.4.2 *Cod sursă

Listing 26.4.1: drumuri.cpp

```

1  /* Mugurel Ionut Andreica */
2  #include <stdio.h>
3  #include <time.h>
4  #include <assert.h>
5
6  #define NMAX 40001
7  #define K 75
8
9  #define USE_FAST SOLUTION 1
10
11 int neigh[NMAX][2], kneigh[NMAX][2], p[NMAX];
12 int ki[K+1], kj[K+1], diri[K+1], dirj[K+1];
13 int t, i, j, d, dd, N, M, ci, cj;
14
15 void find_opposite_endpoint(int i,int j,int dmax,int& endpoint,int& distance)
16 {
17     endpoint = i;
18     distance = 0;
19     int dir;
20
21     if (neigh[i][0] == j)
22         dir = 1;
23     else
24         dir = 0;
25
26     endpoint = i;
27
28     while (1)
29     {
30         if (distance + K > dmax)
31             break;
32         if (kneigh[endpoint][dir] == 0)
33             break;
34         j = endpoint;
35         endpoint = kneigh[endpoint][dir];
36         distance += K;
37         if (kneigh[endpoint][dir] == j)
38             dir = 1 - dir;
39     }
40
41     while (1)
42     {
43         if (distance == dmax)
44             break;
45         if (neigh[endpoint][dir] == 0)
46             break;
47         j = endpoint;
48         endpoint = neigh[endpoint][dir];
49         distance++;
50         if (neigh[endpoint][dir] == j)
51             dir = 1 - dir;
52     }
53 }
54
55 void update_kneighs(int i, int j, int op)
56 {
57     int u, v;
58
59     ci = 0; u = i; v = j;
60     while (u > 0 && ci < K)
61     {
62         ki[ci] = u;

```

```

63         if (neigh[u][0] == v)
64     {
65         v = u;
66         u = neigh[u][1];
67         diri[ci] = 0;
68     }
69     else
70     {
71         v = u;
72         u = neigh[u][0];
73         diri[ci] = 1;
74     }
75     ci++;
76 }
77
78 cj = 0;
79 u = j;
80 v = i;
81 while (u > 0 && cj < K)
82 {
83     kj[cj] = u;
84     if (neigh[u][0] == v)
85     {
86         v = u;
87         u = neigh[u][1];
88         dirj[cj] = 0;
89     }
90     else
91     {
92         v = u;
93         u = neigh[u][0];
94         dirj[cj] = 1;
95     }
96     cj++;
97 }
98
99 for (u = 0; u < ci; u++)
100    if (op == 1)
101    {
102        v = K - u - 1;
103        if (cj > v)
104            kneigh[ki[u]][diri[u]] = kj[v];
105    }
106    else
107        kneigh[ki[u]][diri[u]] = 0;
108
109 for (u = 0; u < cj; u++)
110    if (op == 1)
111    {
112        v = K - u - 1;
113        if (ci > v)
114        {
115            kneigh[kj[u]][dirj[u]] = ki[v];
116        }
117    }
118    else
119        kneigh[kj[u]][dirj[u]] = 0;
120 }
121
122 void print_nodes(int u, int v)
123 {
124     if (u == v)
125         u = 0;
126     if (u > 0 && v > 0)
127     {
128         printf("2 ");
129         if (u < v)
130             printf("%d %d\n", u, v);
131         else
132             printf("%d %d\n", v, u);
133     }
134     else
135         if (u == 0 && v > 0)
136             printf("1 %d\n", v);
137         else
138             if (u > 0 && v == 0)

```

```

139             printf("1 %d\n", u);
140         else
141             printf("0\n");
142     }
143
144 int main()
145 {
146     int u, v, tstart = clock();
147
148     freopen("drumuri.in", "r", stdin);
149     freopen("drumuri.out", "w", stdout);
150
151     scanf("%d %d", &N, &M);
152     assert(N >= 1 && N <= 40000);
153
154     for (i = 1; i <= N; i++)
155         p[i] = i;
156
157     while (M--)
158     {
159         t = 0;
160         scanf("%d %d", &t, &i);
161         assert(t >= 1 && t <= 4);
162         assert(i >= 1 && i <= N);
163
164         if (t == 1)
165         {
166             scanf("%d", &j);
167             assert(j >= 1 && j <= N && i != j);
168
169             // Add a new edge i-j.
170             if (p[i] != j && p[i] > 0 && p[j] > 0 && i != j)
171             {
172                 // Connect the nodes i and j.
173                 if (neigh[i][0] == 0)
174                     neigh[i][0] = j;
175                 else
176                     neigh[i][1] = j;
177
178                 if (neigh[j][0] == 0)
179                     neigh[j][0] = i;
180                 else
181                     neigh[j][1] = i;
182
183                 // Update k-neighbors.
184                 if (USE_FAST SOLUTION)
185                     update_kneighs(i, j, 1);
186
187                 // Adjust the endpoint pairs.
188                 if (p[i] == i)
189                     ci = 0;
190                 else
191                     ci = 1;
192
193                 if (p[j] == j)
194                     cj = 0;
195                 else
196                     cj = 1;
197
198                 u = p[i]; v = p[j];
199                 p[u] = v;
200                 p[v] = u;
201
202                 if (ci)
203                     p[i] = 0;
204
205                 if (cj)
206                     p[j] = 0;
207
208                 printf("1\n");
209             }
210             else
211                 printf("0\n");
212         }
213         else
214             if (t == 2)

```

```

215      {
216          scanf("%d", &j);
217          assert(j >= 1 && j <= N && i != j);
218
219          // Remove an old edge i-j.
220          if (neigh[i][0] == j || neigh[i][1] == j)
221          {
222              find_opposite_endpoint(i, j, N, u, d);
223              v = p[u];
224
225              // Adjust k-neighbors.
226              if (USE_FAST SOLUTION)
227                  update_kneighs(i, j, 2);
228
229              // Clear the neighbors i and j.
230              if (neigh[i][0] == j)
231                  neigh[i][0] = 0;
232              else
233                  neigh[i][1] = 0;
234
235              if (neigh[j][0] == i)
236                  neigh[j][0] = 0;
237              else
238                  neigh[j][1] = 0;
239
240              // Adjust the endpoint pairs.
241              p[i] = u;
242              p[u] = i;
243              p[j] = v;
244              p[v] = j;
245
246              printf("1\n");
247          }
248          else
249              printf("0\n");
250      }
251      else
252          if (t == 3)
253          {
254              scanf("%d", &d);
255              assert(d >= 0 && d <= N - 1);
256
257              // Compute the neighbors located at distance d from
258              // the node i (on the existing paths).
259              find_opposite_endpoint(i, neigh[i][0], d, u, dd);
260              if (dd != d)
261                  u = 0;
262
263              find_opposite_endpoint(i, neigh[i][1], d, v, dd);
264              if (dd != d)
265                  v = 0;
266
267              print_nodes(u, v);
268          }
269          else
270              if (t == 4)
271              {
272                  // Compute the distances from the two endpoints of
273                  // the path to which node i belongs.
274                  find_opposite_endpoint(i, neigh[i][0], N, u, dd);
275                  find_opposite_endpoint(i, neigh[i][1], N, v, dd);
276                  print_nodes(u, v);
277              }
278      }
279
280      fprintf(stderr, "Duration=%3lf sec\n",
281             (double) (clock() - tstart) / CLOCKS_PER_SEC);
282      return 0;
283  }

```

Listing 26.4.2: drumuri-40-cosmin.cpp

```

1  /* Cosmin Tutunaru */
2  #include<cstdio>
3  #include<vector>

```

```

4 #include<algorithm>
5
6 #define infile "drumuri.in"
7 #define outfile "drumuri.out"
8 #define nMax 40013
9
10 using namespace std;
11
12 int v[nMax][2];
13 int n, m;
14
15 int findHead(int x, int y, int d = 1<<30)
16 {
17     int crt = 0;
18     while((v[x][0] && v[x][0] != y) || (v[x][1] && v[x][1] != y))
19     {
20         if(crt == d) break;
21         int z = x;
22         ++crt;
23
24         if(v[x][0] && v[x][0] != y) x = v[x][0];
25         else x = v[x][1];
26         y = z;
27     }
28
29     if(d == 1<<30) return x;
30     if(crt == d) return x;
31     return 0;
32 }
33
34 vector <int> findHeads(int x)
35 {
36     int a = findHead(v[x][0], x);
37     int b = findHead(v[x][1], x);
38     vector <int> v;
39
40     if(!a) a = x;
41     if(!b) b = x;
42
43     v.push_back(a);
44     if(a != b) v.push_back(b);
45
46     sort(v.begin(), v.end());
47     return v;
48 }
49
50
51 vector <int> findAtDist(int x, int d)
52 {
53     if(d == 0) return vector<int>(1, x);
54
55     int a = findHead(v[x][0], x, d-1);
56     int b = findHead(v[x][1], x, d-1);
57     vector <int> v;
58
59     if(a) v.push_back(a);
60     if(b) v.push_back(b);
61
62     sort(v.begin(), v.end());
63     return v;
64 }
65
66 bool add(int x, int y)
67 {
68     if(v[x][0] && v[x][1]) return false;
69     if(v[y][0] && v[y][1]) return false;
70
71     vector <int> a = findHeads(x);
72     vector <int> b = findHeads(y);
73
74     if(a == b && (a.size() || b.size())) return false;
75     !v[x][0] ? v[x][0] = y : v[x][1] = y;
76     !v[y][0] ? v[y][0] = x : v[y][1] = x;
77
78     return true;
79 }
```

```

80
81 bool rmv(int x, int y)
82 {
83     if(v[x][0] == y)
84     {
85         v[x][0] = 0;
86         v[y][0] == x ? v[y][0] = 0 : v[y][1] = 0;
87         return true;
88     }
89
90     if(v[x][1] == y)
91     {
92         v[x][1] = 0;
93         v[y][0] == x ? v[y][0] = 0 : v[y][1] = 0;
94         return true;
95     }
96
97     return false;
98 }
99
100 void write(vector <int> v)
101 {
102     printf("%d", v.size());
103     for(unsigned i = 0; i < v.size(); ++i)
104         printf(" %d", v[i]);
105
106     printf("\n");
107 }
108
109 void solve()
110 {
111     scanf("%d %d\n", &n, &m);
112
113     while(m--)
114     {
115         int type, x, y, d;
116         scanf("%d", &type);
117
118         switch(type)
119         {
120             case 1:
121                 scanf("%d %d", &x, &y);
122                 printf("%d\n", add(x, y));
123                 break;
124             case 2:
125                 scanf("%d %d", &x, &y);
126                 printf("%d\n", rmv(x, y));
127                 break;
128             case 3:
129                 scanf("%d %d", &x, &d);
130                 write(findAtDist(x, d));
131                 break;
132             case 4:
133                 scanf("%d", &x);
134                 write(findHeads(x));
135                 break;
136         }
137     }
138 }
139
140 int main()
141 {
142     freopen(infile, "r", stdin);
143     freopen(outfile, "w", stdout);
144
145     solve();
146
147     fclose(stdin);
148     fclose(stdout);
149     return 0;
150 }
```

Listing 26.4.3: drumuri-40-mugurel.cpp

```

2 #include <stdio.h>
3 #include <time.h>
4
5 #define NMAX 40001
6 #define K 200
7
8 #define USE_FAST SOLUTION 0
9
10 int neigh[NMAX][2], kneigh[NMAX][2], p[NMAX];
11 int ki[K+1], kj[K+1], diri[K+1], dirj[K+1];
12 int t, i, j, d, dd, N, M, ci, cj;
13
14 void find_opposite_endpoint(int i,int j,int dmax,int& endpoint,int& distance)
15 {
16     endpoint = i;
17     distance = 0;
18     int dir;
19
20     if (neigh[i][0] == j)
21         dir = 1;
22     else
23         dir = 0;
24
25     endpoint = i;
26
27     while (1)
28     {
29         if (distance + K > dmax)
30             break;
31         if (kneigh[endpoint][dir] == 0)
32             break;
33         j = endpoint;
34         endpoint = kneigh[endpoint][dir];
35         distance += K;
36         if (kneigh[endpoint][dir] == j)
37             dir = 1 - dir;
38     }
39
40     while (1)
41     {
42         if (distance == dmax)
43             break;
44         if (neigh[endpoint][dir] == 0)
45             break;
46         j = endpoint;
47         endpoint = neigh[endpoint][dir];
48         distance++;
49         if (neigh[endpoint][dir] == j)
50             dir = 1 - dir;
51     }
52 }
53
54 void update_kneighs(int i, int j, int op)
55 {
56     int u, v;
57
58     ci = 0; u = i; v = j;
59     while (u > 0 && ci < K)
60     {
61         ki[ci] = u;
62         if (neigh[u][0] == v)
63         {
64             v = u;
65             u = neigh[u][1];
66             diri[ci] = 0;
67         }
68         else
69         {
70             v = u;
71             u = neigh[u][0];
72             diri[ci] = 1;
73         }
74
75         ci++;
76     }
77 }
```

```

78     cj = 0;
79     u = j;
80     v = i;
81     while (u > 0 && cj < K)
82     {
83         kj[cj] = u;
84         if (neigh[u][0] == v)
85         {
86             v = u;
87             u = neigh[u][1];
88             dirj[cj] = 0;
89         }
90         else
91         {
92             v = u;
93             u = neigh[u][0];
94             dirj[cj] = 1;
95         }
96         cj++;
97     }
98
99
100    for (u = 0; u < ci; u++)
101    {
102        if (op == 1)
103        {
104            v = K - u - 1;
105            if (cj > v)
106                kneigh[kj[u]][diri[u]] = kj[v];
107        }
108        else
109            kneigh[kj[u]][diri[u]] = 0;
110    }
111
112    for (u = 0; u < cj; u++)
113    {
114        if (op == 1)
115        {
116            v = K - u - 1;
117            if (ci > v)
118                kneigh[kj[u]][dirj[u]] = ki[v];
119        }
120        else
121            kneigh[kj[u]][dirj[u]] = 0;
122    }
123 }
124
125 void print_nodes(int u, int v)
126 {
127     if (u == v)
128         u = 0;
129     if (u > 0 && v > 0)
130     {
131         printf("2 ");
132         if (u < v)
133             printf("%d %d\n", u, v);
134         else
135             printf("%d %d\n", v, u);
136     }
137     else
138         if (u == 0 && v > 0)
139             printf("1 %d\n", v);
140         else
141             if (u > 0 && v == 0)
142                 printf("1 %d\n", u);
143             else
144                 printf("0\n");
145 }
146
147 int main()
148 {
149     int u, v, tstart = clock();
150
151     freopen("drumuri.in", "r", stdin);
152     freopen("drumuri.out", "w", stdout);
153 }
```

```

154     scanf("%d %d", &N, &M);
155
156     for (i = 1; i <= N; i++)
157         p[i] = i;
158
159     while (M--)
160     {
161         scanf("%d %d", &t, &i);
162         if (t == 1)
163         {
164             scanf("%d", &j);
165
166             // Add a new edge i-j.
167             if (p[i] != j && p[i] > 0 && p[j] > 0 && i != j)
168             {
169                 // Connect the nodes i and j.
170                 if (neigh[i][0] == 0)
171                     neigh[i][0] = j;
172                 else
173                     neigh[i][1] = j;
174
175                 if (neigh[j][0] == 0)
176                     neigh[j][0] = i;
177                 else
178                     neigh[j][1] = i;
179
180                 // Update k-neighbors.
181                 if (USE_FAST SOLUTION)
182                     update_kneighs(i, j, 1);
183
184                 // Adjust the endpoint pairs.
185                 if (p[i] == i)
186
187                     ci = 0;
188                 else
189                     ci = 1;
190
191                 if (p[j] == j)
192                     cj = 0;
193                 else
194                     cj = 1;
195
196                 u = p[i]; v = p[j];
197                 p[u] = v;
198                 p[v] = u;
199
200                 if (ci)
201                     p[i] = 0;
202
203                 if (cj)
204                     p[j] = 0;
205
206                 printf("1\n");
207             }
208             else
209                 printf("0\n");
210         }
211     else
212     if (t == 2)
213     {
214         scanf("%d", &j);
215
216         // Remove an old edge i-j.
217         if (neigh[i][0] == j || neigh[i][1] == j)
218         {
219             find_opposite_endpoint(i, j, N, u, d);
220             v = p[u];
221
222             // Adjust k-neighbors.
223             if (USE_FAST SOLUTION)
224                 update_kneighs(i, j, 2);
225
226             // Clear the neighbors i and j.
227             if (neigh[i][0] == j)
228                 neigh[i][0] = 0;
229             else

```

```

230             neigh[i][1] = 0;
231
232             if (neigh[j][0] == i)
233                 neigh[j][0] = 0;
234             else
235                 neigh[j][1] = 0;
236
237             // Adjust the endpoint pairs.
238             p[i] = u;
239             p[u] = i;
240             p[j] = v;
241             p[v] = j;
242
243             printf("1\n");
244         }
245         else
246             printf("0\n");
247     }
248     else
249     {
250         if (t == 3)
251         {
252             scanf("%d", &d);
253
254             // Compute the neighbors located at distance d from
255             // the node i (on the existing paths).
256             find_opposite_endpoint(i, neigh[i][0], d, u, dd);
257             if (dd != d)
258                 u = 0;
259
260             find_opposite_endpoint(i, neigh[i][1], d, v, dd);
261             if (dd != d)
262                 v = 0;
263
264             print_nodes(u, v);
265         }
266         else
267         {
268             // Compute the distances from the two endpoints
269             // of the path to which node i belongs.
270             find_opposite_endpoint(i, neigh[i][0], N, u, dd);
271             find_opposite_endpoint(i, neigh[i][1], N, v, dd);
272             print_nodes(u, v);
273         }
274     }
275
276     fprintf(stderr, "Duration=% .3lf sec\n",
277             (double) (clock() - tstart) / CLOCKS_PER_SEC);
278
279     return 0;
}

```

Listing 26.4.4: drumuri-40-mugurel2.cpp

```

1  /* Mugurel Andreica */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5  #include <set>
6  #include <vector>
7  #include <string.h>
8
9  using namespace std;
10
11 #define NMAX 40001
12
13 #define myabs(x) ((x) < 0 ? (- (x)) : (x))
14
15 set<int> available_positions;
16
17 #ifdef USE_STL_VECTORS
18 vector<int> path[NMAX];
19#else
20 unsigned short int* path[NMAX];
21#endif
22

```

```

23 int path_len[NMAX], path_id[NMAX], path_pos[NMAX];
24 int N, M, i, j, d, op, x, y, z;
25
26 void path_union(int i, int revi, int j, int revj) {
27 #ifdef USE_STL_VECTORS
28     if (path_len[i] < path_len[j]) {
29         path_union(j, 1 - revj, i, 1 - revi);
30         return;
31     }
32 #endif
33
34     if (revi)
35     {
36         for (x = 0, y = path_len[i] - 1; x < y; ++x, --y)
37         {
38             z = path[i][x];
39             path[i][x] = path[i][y];
40             path[i][y] = z;
41             path_pos[path[i][x]] = x;
42             path_pos[path[i][y]] = y;
43         }
44     }
45
46 #ifdef USE_STL_VECTORS
47     if (revj) {
48         for (x = path_len[j] - 1; x >= 0; --x) {
49             path[i].push_back(path[j][x]);
50         }
51     } else {
52         for (x = 0; x < path_len[j]; ++x) {
53             path[i].push_back(path[j][x]);
54         }
55     }
56 #else
57     path[i] = (unsigned short int*) realloc(
58                                         path[i], (path_len[i]+path_len[j]) *
59                                         sizeof(unsigned short int));
60
61     if (revj)
62         for (x = path_len[j] - 1, y = path_len[i]; x >= 0; --x, ++y)
63             path[i][y] = path[j][x];
64     else
65         memcpy(&(path[i][path_len[i]]),
66                path[j], path_len[j] *
67                sizeof(unsigned short int));
68
69 #endif
70
71     for (x = path_len[i]; x < path_len[i] + path_len[j]; ++x)
72     {
73         path_id[path[i][x]] = i;
74         path_pos[path[i][x]] = x;
75     }
76
77     path_len[i] += path_len[j];
78     available_positions.insert(j);
79
80 #ifdef USE_STL_VECTORS
81     path[j].clear();
82 #else
83     delete path[j];
84 #endif
85 }
86
87 void path_split(int i, int left_len)
88 {
89     j = *available_positions.begin();
90     available_positions.erase(j);
91     path_len[j] = path_len[i] - left_len;
92
93 #ifdef USE_STL_VECTORS
94 #else
95     path[j] = new unsigned short int[path_len[j]];
96 #endif
97
98     for (x = left_len; x < path_len[i]; ++x)

```

```

99     {
100 #ifdef USE_STL_VECTORS
101     path[j].push_back(path[i][x]);
102 #else
103     path[j][x - left_len] = path[i][x];
104 #endif
105     path_id[path[i][x]] = j;
106     path_pos[path[i][x]] = x - left_len;
107 }
108
109 path_len[i] = left_len;
110
111 #ifdef USE_STL_VECTORS
112     path[i].resize(left_len);
113 #else
114     path[i] = (unsigned short int*) realloc(
115                                         path[i],
116                                         left_len * sizeof(unsigned short int));
117 #endif
118 }
119
120 void print_nodes(int u, int v)
121 {
122     if (u == v)
123         u = 0;
124     if (u > 0 && v > 0)
125     {
126         printf("2 ");
127         if (u < v)
128             printf("%d %d\n", u, v);
129         else
130             printf("%d %d\n", v, u);
131     }
132     else
133         if (u == 0 && v > 0)
134             printf("1 %d\n", v);
135         else
136             if (u > 0 && v == 0)
137                 printf("1 %d\n", u);
138             else
139                 printf("0\n");
140 }
141
142 int main()
143 {
144     int tstart = clock();
145     freopen("drumuri.in", "r", stdin);
146     freopen("drumuri.out", "w", stdout);
147
148     scanf("%d %d", &N, &M);
149     for (i = 1; i <= N; i++)
150     {
151         #ifdef USE_STL_VECTORS
152         path[i].push_back(i);
153         #else
154         path[i] = new unsigned short int[1];
155         path[i][0] = i;
156         #endif
157         path_len[i] = 1;
158         path_id[i] = i;
159         path_pos[i] = 0;
160     }
161
162     while (M--)
163     {
164         scanf("%d", &op);
165
166         if (op == 1)
167         {
168             scanf("%d %d", &i, &j);
169             if (path_id[i] != path_id[j] &&
170                 (path_pos[i] == 0 || path_pos[i] == path_len[path_id[i]] - 1) &&
171                 (path_pos[j] == 0 || path_pos[j] == path_len[path_id[j]] - 1))
172             {
173
174                 if (path_pos[i] == 0 && path_pos[j] > 0)

```

```

175             path_union(path_id[j], 0, path_id[i], 0);
176         else if (path_pos[i] == 0 && path_pos[j] == 0)
177             path_union(path_id[i], 1, path_id[j], 0);
178         else if (path_pos[i] > 0 && path_pos[j] == 0)
179             path_union(path_id[i], 0, path_id[j], 0);
180         else
181             // path_pos[i] > 0 && path_pos[j] > 0
182             path_union(path_id[i], 0, path_id[j], 1);
183
184         printf("1\n");
185     }
186     else
187         printf("0\n");
188 }
189 else
190     if (op == 2)
191     {
192         scanf("%d %d", &i, &j);
193         if (path_id[i] == path_id[j] &&
194             myabs(path_pos[i] - path_pos[j]) == 1)
195         {
196             path_split(path_id[i], min(path_pos[i], path_pos[j]) + 1);
197             printf("1\n");
198         }
199         else
200             printf("0\n");
201     }
202     else if (op == 3)
203     {
204         scanf("%d %d", &i, &d);
205         if (path_pos[i] >= d)
206             x = path[path_id[i]][path_pos[i] - d];
207         else
208             x = 0;
209
210         if (path_pos[i] + d < path_len[path_id[i]])
211             y = path[path_id[i]][path_pos[i] + d];
212         else
213             y = 0;
214
215         print_nodes(x, y);
216     }
217     else
218     {
219         scanf("%d", &i);
220         print_nodes(path[path_id[i]][0],
221                     path[path_id[i]][path_len[path_id[i]] - 1]);
222     }
223 }
224
225 fprintf(stderr, "Duration=% .3lf sec\n",
226         (double) (clock() - tstart) / CLOCKS_PER_SEC);
227 return 0;
228 }
```

Listing 26.4.5: drumuri-100-bogdan.cpp

```

1 /**
2 * (C) 2012 Bogdan-Cristian TĂĂtĂĂcroiu
3 */
4 #include <cstdio>
5 #include <cassert>
6 #include <algorithm>
7 #include <set>
8
9 using namespace std;
10
11 const int MAXN = 40000;
12 const int MAXM = 400000;
13 const int K = 75;
14
15 int N, M;
16 int endPair[MAXN];
17 int con[MAXN][2], conK[MAXN][2];
18
```

```

19 inline int walk(int nod, int dir, int count, bool noSkipping = false)
20 {
21     int prv;
22     if (!noSkipping)
23     {
24         prv = conK[nod][1 ^ dir];
25         for (; count >= K && nod != -1; count -= K)
26         {
27             int nxt = conK[nod][0] ^ conK[nod][1] ^ prv;
28             prv = nod;
29             nod = nxt;
30             dir = conK[nod][0] == prv;
31         }
32     }
33
34     prv = con[nod][1 ^ dir];
35     for (; count > 0 && nod != -1; count -= 1)
36     {
37         int nxt = con[nod][0] ^ con[nod][1] ^ prv;
38         prv = nod;
39         nod = nxt;
40         dir = con[nod][0] == prv;
41     }
42     return nod;
43 }
44
45 inline int getEnd(int nod, int dir)
46 {
47     int prv = conK[nod][1 ^ dir];
48     for ( ; )
49     {
50         int nxt = conK[nod][0] ^ conK[nod][1] ^ prv;
51         if (nxt == -1)
52             break;
53
54         prv = nod;
55         nod = nxt;
56         dir = conK[nod][0] == prv;
57     }
58     prv = con[nod][1 ^ dir];
59     for ( ; )
60     {
61         int nxt = con[nod][0] ^ con[nod][1] ^ prv;
62         if (nxt == -1)
63             break;
64
65         prv = nod;
66         nod = nxt;
67         dir = con[nod][0] == prv;
68     }
69     return nod;
70 }
71
72 inline void addEdge(int X, int Y)
73 {
74     int XK[K], XKdir[K], YK[K], YKdir[K];
75     int nod, dir, prv;
76     nod = X; dir = con[nod][0] == -1; prv = con[nod][1 ^ dir];
77     for (int i = 0; i < K; i++)
78     {
79         XK[i] = nod;
80         if (nod != -1)
81         {
82             int nxt = con[nod][0] ^ con[nod][1] ^ prv;
83             dir = con[nod][1] == nxt;
84             XKdir[i] = 1 ^ dir;
85             prv = nod;
86             nod = nxt;
87         }
88     }
89
90     nod = Y;
91     dir = con[nod][0] == -1;
92     prv = con[nod][1 ^ dir];
93     for (int i = 0; i < K; i++)
94     {

```

```

95         YK[i] = nod;
96         if (nod != -1)
97         {
98             int nxt = con[nod][0] ^ con[nod][1] ^ prv;
99             dir = con[nod][1] == nxt;
100            XKdir[i] = 1 ^ dir;
101            prv = nod;
102            nod = nxt;
103        }
104    }
105
106    for (int i = 0; i < K; i++)
107    {
108        if (XK[i] != -1)
109        {
110            assert(conK[XK[i]][XKdir[i]] == -1);
111            conK[XK[i]][XKdir[i]] = YK[K - i - 1];
112        }
113
114        if (YK[i] != -1)
115        {
116            assert(conK[YK[i]][YKdir[i]] == -1);
117            conK[YK[i]][YKdir[i]] = XK[K - i - 1];
118        }
119    }
120
121    con[X][XKdir[0]] = Y;
122    con[Y][YKdir[0]] = X;
123}
124
125 inline void removeEdge(int X, int Y)
126{
127    int XK[K], XKdir[K], YK[K], YKdir[K];
128    int nod, dir, prv;
129    nod = X;
130    dir = con[nod][0] == Y;
131    prv = con[nod][1 ^ dir];
132    for (int i = 0; i < K; i++)
133    {
134        XK[i] = nod;
135        if (nod != -1)
136        {
137            int nxt = con[nod][0] ^ con[nod][1] ^ prv;
138            dir = con[nod][1] == nxt;
139            XKdir[i] = 1 ^ dir;
140            prv = nod;
141            nod = nxt;
142        }
143    }
144
145    nod = Y;
146    dir = con[nod][0] == X;
147    prv = con[nod][1 ^ dir];
148    for (int i = 0; i < K; i++)
149    {
150        YK[i] = nod;
151        if (nod != -1)
152        {
153            int nxt = con[nod][0] ^ con[nod][1] ^ prv;
154            dir = con[nod][1] == nxt;
155            YKdir[i] = 1 ^ dir;
156            prv = nod;
157            nod = nxt;
158        }
159    }
160    for (int i = 0; i < K; i++)
161    {
162        if (XK[i] != -1)
163            conK[XK[i]][XKdir[i]] = -1;
164
165        if (YK[i] != -1)
166            conK[YK[i]][YKdir[i]] = -1;
167    }
168
169    con[X][XKdir[0]] = -1;
170    con[Y][YKdir[0]] = -1;

```

```

171 }
172
173 int main()
174 {
175     assert(freopen("drumuri.in", "rt", stdin));
176 #ifndef DEBUG
177     assert(freopen("drumuri.out", "wt", stdout));
178 #endif
179
180     assert(scanf("%d %d", &N, &M) == 2);
181     assert(1 <= N && N <= MAXN);
182     assert(1 <= M && M <= MAXM);
183     for (int i = 0; i < N; i++)
184     {
185         endPair[i] = i;
186         con[i][0] = con[i][1] = -1;
187         conK[i][0] = conK[i][1] = -1;
188     }
189
190     for (; M--)
191     {
192         int type;
193         assert(scanf("%d", &type) == 1);
194         assert(1 <= type && type <= 4);
195         if (type == 1)
196         {
197             int X, Y;
198             assert(scanf("%d %d", &X, &Y) == 2);
199             X -= 1;
200             Y -= 1;
201             assert(0 <= X && X < N && 0 <= Y && Y < N);
202
203             if (con[X][0] != -1 && con[X][1] != -1)
204             {
205                 printf("0\n");
206                 continue;
207             }
208             if (con[Y][0] != -1 && con[Y][1] != -1)
209             {
210                 printf("0\n");
211                 continue;
212             }
213             if (endPair[X] == Y)
214             {
215                 assert(endPair[Y] == X);
216                 printf("0\n");
217                 continue;
218             }
219
220             addEdge(X, Y);
221
222             int endPairX = endPair[X], endPairY = endPair[Y];
223             endPair[endPairX] = endPairY;
224             endPair[endPairY] = endPairX;
225             printf("1\n");
226         }
227         else if (type == 2)
228         {
229             int X, Y;
230             assert(scanf("%d %d", &X, &Y) == 2);
231             X -= 1; Y -= 1;
232             assert(0 <= X && X < N && 0 <= Y && Y < N);
233
234             if (con[X][0] != Y && con[X][1] != Y)
235             {
236                 assert(con[Y][0] != X && con[Y][1] != X);
237                 printf("0\n");
238                 continue;
239             }
240             assert(con[Y][0] == X || con[Y][1] == X);
241
242             removeEdge(X, Y);
243             endPair[X] = getEnd(X, con[X][0] == -1);
244             endPair[Y] = getEnd(Y, con[Y][0] == -1);
245             endPair[endPair[X]] = X;
246             endPair[endPair[Y]] = Y;

```

```

247         printf("1\n");
248     }
249     else if (type == 3)
250     {
251         int nod, dist, n1, n2;
252         assert(scanf("%d %d", &nod, &dist) == 2);
253         nod -= 1;
254         assert(0 <= nod && nod < N);
255
256         n1 = walk(nod, 0, dist);
257         n2 = walk(nod, 1, dist);
258
259         set<int> ans;
260         if (n1 != -1)
261             ans.insert(n1);
262
263         if (n2 != -1)
264             ans.insert(n2);
265
266         printf("%d", (int)ans.size());
267         for (set<int> :: iterator it = ans.begin(); it != ans.end(); it++)
268             printf(" %d", *it + 1);
269
270         printf("\n");
271     }
272     else if (type == 4)
273     {
274         int nod, n1, n2;
275         assert(scanf("%d", &nod) == 1);
276         nod -= 1;
277         assert(0 <= nod && nod < N);
278
279         n1 = getEnd(nod, 0);
280         n2 = getEnd(nod, 1);
281         set<int> ans;
282         ans.insert(n1);
283         ans.insert(n2);
284         printf("%d", (int)ans.size());
285         for (set<int> :: iterator it = ans.begin(); it != ans.end(); it++)
286             printf(" %d", *it + 1);
287
288         printf("\n");
289     }
290
291     //{
292     //    for (int i = 0; i < N; i++) {
293     //        assert(conK[i][0] == walk(i, 0, K, true));
294     //        assert(conK[i][1] == walk(i, 1, K, true));
295     //    }
296     //    fflush(stdout);
297     //}
298 }
299
300 return 0;
301 }
```

Listing 26.4.6: drumuri-100-tibi.cpp

```

1  /* Tiberiu Savin */
2  #include <stdio.h>
3  #include <vector>
4  #include <algorithm>
5
6  using namespace std;
7
8  #define NMAX 40005
9
10 int N, M, K;
11 int left[NMAX], right[NMAX];
12 int super_left[NMAX], super_right[NMAX];
13
14 int go_deep(int nod, int lst)
15 {
16     int lst2;
17     if (left[nod] == lst)
```

```

18      {
19          lst2 = lst;
20          lst = nod;
21          nod = super_right[nod];
22      }
23  else
24  {
25      lst2 = lst;
26      lst = nod;
27      nod = super_left[nod];
28  }
29
30  int super_jump = 1;
31  if (nod == 0)
32  {
33      super_jump = 0;
34      nod = lst;
35      lst = lst2;
36  }
37
38  while (super_jump)
39  {
40      if ((super_left[nod] == lst && super_right[nod] == 0) ||
41          (super_right[nod] == lst && super_left[nod] == 0))
42      {
43          break;
44      }
45
46      if (super_left[nod] == lst)
47      {
48          lst = nod;
49          nod = super_right[nod];
50      }
51      else
52      {
53          lst = nod;
54          nod = super_left[nod];
55      }
56
57  }
58
59  if (super_jump)
60  {
61      if (super_left[nod] == lst)
62      {
63          lst = nod;
64          nod = right[nod];
65      }
66      else
67      {
68          lst = nod;
69          nod = left[nod];
70      }
71  }
72
73  if (!nod)
74      return lst;
75
76  while (1)
77  {
78      if ((left[nod] == lst && right[nod] == 0) ||
79          (right[nod] == lst && left[nod] == 0))
80      {
81          return nod;
82      }
83
84      if (left[nod] == lst)
85      {
86          lst = nod;
87          nod = right[nod];
88      }
89      else
90      {
91          lst = nod;
92          nod = left[nod];
93      }

```

```

94      }
95  }
96
97 bool check_add(int x, int y)
98 {
99     int omolog;
100    if (left[x] == 0 && right[x] == 0)
101        omolog = x;
102
103    if (left[x] == 0)
104        omolog = go_deep(right[x], x);
105    else
106        omolog = go_deep(left[x], x);
107
108    if (omolog == y)
109        return false;
110
111    if (left[x] && right[x])
112        return false;
113
114    if (left[y] && right[y])
115        return false;
116
117    return true;
118 }
119
120 void add_edge(int x, int y)
121 {
122     if (!check_add(x, y))
123     {
124         printf("0\n");
125         return;
126     }
127     printf("1\n");
128
129     if (left[x] == 0)
130         left[x] = y;
131     else
132         right[x] = y;
133
134     if (left[y] == 0)
135         left[y] = x;
136     else
137         right[y] = x;
138
139 // Maintain super neighbours
140 // try to go K steps in one direction, and then go back
141 int lst = x;
142 int nod = y;
143 int dist = K - 1;
144 for (int i = 1; i <= K - 1; i++)
145 {
146     if (left[nod] == lst)
147     {
148         lst = nod;
149         nod = right[nod];
150     }
151     else
152     {
153         lst = nod;
154         nod = left[nod];
155     }
156
157     if (nod == 0)
158     {
159         nod = lst;
160         if (left[nod])
161             lst = left[nod];
162         else
163             lst = right[nod];
164
165         dist = i - 1;
166         break;
167     }
168 }
169

```

```

170     int nod1 = nod;
171     int lst1 = lst;
172
173     nod = x;
174     lst = y;
175
176     for (int i = 1; i <= K - dist - 1; i++)
177     {
178         if (left[nod] == lst)
179         {
180             lst = nod;
181             nod = right[nod];
182         }
183         else
184         {
185             lst = nod;
186             nod = left[nod];
187         }
188
189         if (nod == 0)
190         {
191             // no super-neighbours to maintain
192             return;
193         }
194     }
195
196     int nod2 = nod;
197     int lst2 = lst;
198
199     // Reverse direction
200
201     if (left[nod1] == lst1)
202         lst1 = right[nod1];
203     else
204         lst1 = left[nod1];
205
206     while (nod1 != x)
207     {
208         if (left[nod1] == lst1)
209             super_right[nod1] = nod2;
210         else
211             super_left[nod1] = nod2;
212
213         if (left[nod2] == lst2)
214             super_left[nod2] = nod1;
215         else
216             super_right[nod2] = nod1;
217
218         if (left[nod1] == lst1)
219         {
220             lst1 = nod1;
221             nod1 = right[nod1];
222         }
223         else
224         {
225             lst1 = nod1;
226             nod1 = left[nod1];
227         }
228
229         if (left[nod2] == lst2)
230         {
231             lst2 = nod2;
232             nod2 = right[nod2];
233         }
234         else
235         {
236             lst2 = nod2;
237             nod2 = left[nod2];
238         }
239
240         // we could reach the end
241         if (nod2 == 0)
242             break;
243     }
244 }
```

```

246 void erase_super_neighbours(int nod)
247 {
248     int lst = 0;
249     for (int i = 1; i <= K; i++)
250     {
251         if (left[nod] == lst)
252             super_left[nod] = 0;
253         else
254             super_right[nod] = 0;
255
256         if (left[nod] == lst)
257         {
258             lst = nod;
259             nod = right[nod];
260         }
261         else
262         {
263             lst = nod;
264             nod = left[nod];
265         }
266
267         if (nod == 0)
268             break;
269     }
270 }
271
272 void erase_edge(int x, int y)
273 {
274     bool success = 0;
275     if (left[x] == y)
276     {
277         left[x] = 0;
278         success = 1;
279     }
280     else if (right[x] == y)
281     {
282         right[x] = 0;
283         success = 1;
284     }
285
286     if (left[y] == x)
287         left[y] = 0;
288     else if (right[y] == x)
289         right[y] = 0;
290
291     // Erase super-neighbours
292     if (success)
293     {
294         erase_super_neighbours(x);
295         erase_super_neighbours(y);
296     }
297
298     printf("%d\n", success);
299 }
300
301 int go(int dist, int nod, int lst)
302 {
303     int lst2 = 0;
304     //printf("right=%d left=%d init=%d ", right[830], left[830], nod);
305
306
307     if (nod == 0)
308         return 0;
309
310     if (left[nod] == lst)
311     {
312         lst2 = lst;
313         lst = nod;
314         nod = super_right[nod];
315     }
316     else
317     {
318         lst2 = lst;
319         lst = nod;
320         nod = super_left[nod];
321     }

```

```

322
323     int step, super_jump = 1;
324     if (nod != 0)
325         step = K + 1;
326
327     if (step >= dist || nod == 0)
328     {
329         nod = lst;
330         lst = lst2;
331         step = 1;
332         super_jump = 0;
333     }
334
335     for (int i = step; i < dist && super_jump; i += K)
336     {
337         if (super_left[nod] == lst)
338         {
339             lst2 = lst;
340             lst = nod;
341             nod = super_right[nod];
342         }
343         else
344         {
345             lst2 = lst;
346             lst = nod;
347             nod = super_left[nod];
348         }
349
350         if (nod)
351         {
352             step = step + K;
353             if (step >= dist) {
354                 step -= K;
355                 nod = lst;
356                 lst = lst2;
357             }
358         }
359         else
360         {
361             nod = lst;
362             lst = lst2;
363             break;
364         }
365     }
366
367     int first = 1;
368     // printf("step=%d nod=%d ", step, nod);
369     for (int i = step; i < dist; i++)
370     {
371         int left_nod;
372         if (first && super_jump)
373             left_nod = super_left[nod];
374         else
375             left_nod = left[nod];
376
377         first = 0;
378
379         if (left_nod == lst)
380         {
381             lst = nod;
382             nod = right[nod];
383         }
384         else
385         {
386             lst = nod;
387             nod = left[nod];
388         }
389
390         if (nod == 0)
391             return 0;
392     }
393
394     return nod;
395 }
396
397 void print_ans(int nod1, int nod2)

```

```

398 {
399     vector<int> ret;
400     if (nod1)
401         ret.push_back(nod1);
402
403     if (nod2 && nod1 != nod2)
404         ret.push_back(nod2);
405
406     printf("%d ", (int) ret.size());
407
408     sort(ret.begin(), ret.end());
409
410     for (int i = 0; i < (int) ret.size(); i++)
411         printf("%d ", ret[i]);
412
413     printf("\n");
414 }
415
416 void query_dist(int x, int d)
417 {
418     if (d == 0)
419     {
420         printf("1 %d\n", x);
421         return;
422     }
423
424     int nod1 = go(d, left[x], x);
425     int nod2 = go(d, right[x], x);
426     print_ans(nod1, nod2);
427 }
428
429 void query_road(int x)
430 {
431     int nod1, nod2;
432     if (left[x] == 0)
433         nod1 = x;
434     else
435         nod1 = go_deep(left[x], x);
436
437     if (right[x] == 0)
438         nod2 = x;
439     else
440         nod2 = go_deep(right[x], x);
441
442     print_ans(nod1, nod2);
443 }
444
445 int main()
446 {
447     freopen("drumuri.in", "r", stdin);
448     freopen("drumuri.out", "w", stdout);
449
450     scanf("%d %d ", &N, &M);
451     K = 100;
452
453     for (int i = 1; i <= M; i++)
454     {
455         int cod;
456         scanf("%d ", &cod);
457         if (cod == 1)
458         {
459             int x, y;
460             scanf("%d %d ", &x, &y);
461             add_edge(x, y);
462         }
463         else if (cod == 2)
464         {
465             int x, y;
466             scanf("%d %d ", &x, &y);
467             erase_edge(x, y);
468         }
469         else if (cod == 3)
470         {
471             int x, d;
472             scanf("%d %d ", &x, &d);
473             query_dist(x, d);

```

```

474         }
475     else
476     {
477         int x;
478         scanf ("%d ", &x);
479         query_road(x);
480     }
481 }
482 }
```

26.4.3 *Rezolvare detaliată

26.5 minerale

Problema 5 - minerale

100 de puncte

La începutul începutului, pe Terra a existat o singură substanță chimică S , numită substanță primordială. Ulterior, din ea s-au format alte substanțe chimice, unele stabile (notate prin litere mici ale alfabetului englez), iar altele instabile (notate prin litere mari ale alfabetului englez).

Substanțele stabile aveau proprietatea că nu se mai puteau transforma în alte substanțe, în timp ce o substanță instabilă se putea transforma fie într-o substanță stabilă, fie în două substanțe instabile. În urma reacțiilor chimice repetitive s-au format mineralele, conglobate de substanțe stabile.

Unele dintre aceste minerale s-au format prin reacții chimice care au pornit chiar din substanța primordială S , în timp ce altele fie s-au format prin reacții chimice care au pornit de la o altă substanță instabilă existentă în acel moment pe Terra, dar nu din substanța primordială S , fie, pur și simplu, au fost aduse din spațiul cosmic, deci nu s-au format în urma unor reacții chimice pornite dintr-o substanță instabilă de pe Terra.

Cerințe

Cei mai mari cercetători contemporani ai haosului primordial, BitDAC și rOCKTETUâZ, au reușit să identifice toate reacțiile chimice care s-au produs de-a lungul timpului pe Terra. Scrieți pentru ei un program care să stabilească dacă un mineral, cu formula dată sub forma unui sir de substanțe stabile, provine dintr-o substanță instabilă de pe Terra, inclusiv din substanța primordială S , sau este de origine extraterestră.

Date de intrare

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

- pe prima linie două numere naturale r și m , reprezentând, în această ordine, numărul reacțiilor chimice și numărul mineralelor ale căror origini vor fi analizate (substanțele stabile vor fi notate prin litere mici ale alfabetului englez, iar substanțele instabile prin litere mari ale alfabetului englez);
- pe fiecare dintre următoarele r rânduri se va găsi câte o reacție chimică, respectiv fie un sir de caractere de forma A b , unde A este o substanță instabilă și b o substanță stabilă, fie un sir de caractere de forma A BC , unde A , B și C sunt 3 substanțe instabile.
- pe fiecare dintre următoarele m rânduri se va găsi formula unui mineral, respectiv un sir de caractere format din cel mult 100 de litere mici ale alfabetului englez.

Date de ieșire

Fișierul de ieșire **minerale.out** va conține m linii, iar pe fiecare linie se va scrie unul dintre numerele 0, 1 sau 2, după cum mineralul cu numărul de ordine egal cu numărul de ordine al liniei este de origine extraterestră, provine din substanța primordială S sau provine dintr-o altă substanță instabilă alta decât substanța primordială S .

Restricții și precizări

- Substanța primordială S este întotdeauna o substanță instabilă.
- Orice mineral poate avea în compoziția sa cel mult 100 de substanțe stable.
- În cazul în care un mineral provine atât din substanța primordială S , cât și din alte substanțe instabile se va considera că el provine din substanța primordială S .
 - $1 \leq r \leq 400$
 - $1 \leq m \leq 100$

Exemplu:

minereale.in	minereale.out	Explicații
8 3	1	Mineralul aaaab se poate obține din substanța primordială S prin următoarele reacții chimice: $S \rightarrow AB \rightarrow aB \rightarrow aAC \rightarrow aaC \rightarrow aaAB \rightarrow aaaB \rightarrow aaaAC \rightarrow aaaaC \rightarrow aaaab$.
S AB	2	
S CA	0	Mineralul aaabb nu se poate obține din substanța primordială S , dar se poate obține din substanța instabilă B prin următoarele reacții chimice: $B \rightarrow BC \rightarrow ACC \rightarrow aABC \rightarrow aaBC \rightarrow aaACC \rightarrow aaaCC \rightarrow aaabC \rightarrow aaabb$.
A a		Mineralul abab nu se poate obține din nici o substanță instabilă.
B BC		
B BC		
B AC		
C AB		
C b		
aaaab		
aaabb		
abab		

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

Memorie: total **64 MB**

Dimensiune maximă a sursei: **10 KB**

26.5.1 Indicații de rezolvare

lect. dr. Radu-Eugen Boriga - Univ. "Titu Maiorescu" București

Fie un mineral cu formula $s = s_1s_2s_3...s_n$ de lungime n și $s_{i,j} = s_i s_{i+1} ... s_j$, unde $s_1, s_2, ..., s_n$ sunt toate substanțe instabile.

Notăm cu $M_{i,j}$ mulțimea substanțelor instabile din care poate proveni $s_{i,j}$.

Mai întâi, se observă că mulțimea $M_{i,i}$ se poate determina din reacțiile de forma $A s_i$, respectiv mulțimea $M_{i,i}$ este formată din substanțele instabile A din care poate proveni substanța stabilă s_i .

În continuare, vom folosi *metoda programării dinamice* pentru a calcula $M_{i,j}$, astfel:

pentru fiecare număr natural k cuprins între i și $j - 1$, se consideră toate sirurile de substanțe instabile BC din produsul cartezian $M_{i,k} \times M_{k+1,j}$ și se determină mulțimea tuturor substanțelor instabile A din care poate proveni sirul BC .

În acest moment, natura mineralului cu formula $s = s_1s_2s_3...s_n$ se poate determina analizând mulțimea $M_{1,n}$, astfel:

- dacă $M_{1,n}$ este mulțimea vidă, atunci se va afișa valoarea 0;

- altfel, dacă $M_{1,n}$ conține substanța primordială S , atunci se va afișa valoarea 1;

- altfel, se va afișa valoarea 2.

În funcție de modul în care se determină elementele mulțimii $M_{i,j}$, se pot obține soluții de complexitate diferite.

Pentru a obține punctaj maxim, este necesară o implementare cu complexitatea $O(m * k^2 * n^3)$, unde k reprezintă numărul substanțelor instabile distințe din reacții.

26.5.2 Cod sursă

Listing 26.5.1: minerale.cpp

```

1  /* Adrian Panaete */
2  #include<cstdio>
3  #include<cstring>
4
5  using namespace std;
6
7  int r,t,S,A[30],B[30][30],LG,L,Sol[110][110],ST,DR,MI,SOL(int,int);
8  char X[4],Y[4],T[110];
9
10 int main()
11 {
12     freopen("minerale.in","r",stdin);
13     freopen("minerale.out","w",stdout);
14
15     scanf("%d%d",&r,&t);
16
17     S=1<<('S'-'A');
18     for(;r;r--)
19     {
20         scanf("%s%s",X,Y);
21         if(Y[0]>='a')
22             A[Y[0]-'a'] |= (1<<(X[0]-'A'));
23         else
24             B[Y[0]-'A'][Y[1]-'A'] |= (1<<(X[0]-'A'));
25     }
26
27     for(;t;t--)
28     {
29         scanf("%s",T+1);
30         LG=strlen(T+1);
31         for(L=1;L<=LG;L++) Sol[L][L]=A[T[L]-'a'];
32         for(L=2;L<=LG;L++)
33             for(ST=1,DR=L;DR<=LG;ST++,DR++)
34             {
35                 Sol[ST][DR]=0;
36                 for(MI=ST;MI<DR;MI++)
37                     Sol[ST][DR] |= SOL(Sol[ST][MI],Sol[MI+1][DR]);
38             }
39
40         if(Sol[1][LG]&S) printf("1\n");
41         else
42             if(Sol[1][LG]) printf("2\n");
43             else printf("0\n");
44     }
45
46     return 0;
47 }
48
49 int SOL(int U,int V)
50 {
51     int ret=0,nu=0,nv=0,I,J,MU[26],MV[26];
52     for(I=0,J=1;U;I++,J<<=1)
53         if(J&U)
54         {
55             MU[++nu]=I;
56             U-=J;
57         }
58     for(I=0,J=1;V;I++,J<<=1)
59         if(J&V)
60         {
61             MV[++nv]=I;
62             V-=J;
63         }
64     for(I=1;I<=nu;I++)
65         for(J=1;J<=nv;J++)
66             ret |= B[MU[I]][MV[J]];
67     return ret;
68 }
```

Listing 26.5.2: minerale-20-vlad.cpp

```

1  /* Vlad Tudose */
2  #include <cstdio>
```

```

3 #include <cstring>
4
5 #define Lmax 128
6 #define Rmax 128
7
8 int R, M;
9 int rez;
10 char X[Rmax], Y[Rmax][3];
11 char cuv[Lmax];
12
13 void back(char *S, int lvl)
14 {
15     char SS[Lmax];
16
17     if (lvl > 25 || rez == 1) return;
18
19     int L = strlen(S);
20     if (L == 1)
21     {
22         if (S[0] == 'S') rez = 1;
23         else if (S[0] <= 'Z' && rez != 1) rez = 2;
24         return;
25     }
26
27     for (int i=0; i<L; ++i)
28         for (int j=0; j<R; ++j)
29         {
30             int Lk = strlen(Y[j]);
31             int k;
32             for (k=0; k<Lk; ++k)
33                 if (S[i+k] != Y[j][k]) break;
34             if (k < Lk) continue;
35
36             memcpy(SS, S, i+1);
37             SS[i] = X[j];
38             memcpy(SS+i+1, S+i+Lk, L-i-Lk+1);
39             SS[L-Lk+1] = 0;
40
41             back(SS, lvl+1);
42         }
43     }
44
45 int main()
46 {
47     freopen("minerale.in", "r", stdin);
48     freopen("minerale.out", "w", stdout);
49
50     scanf("%d %d\n", &R, &M);
51     for (int i=0; i<R; ++i)
52         scanf("%c %s\n", &X[i], &Y[i]);
53
54     for (int i=0; i<M; ++i)
55     {
56         scanf("%s\n", &cuv);
57         rez = 0;
58         back(cuv, 0);
59         printf("%d\n", rez);
60     }
61
62     return 0;
63 }
```

Listing 26.5.3: minerale-70-mugurel.cpp

```

1 /* Mugurel Andreica */
2 #include <stdio.h>
3 #include <vector>
4 #include <set>
5 #include <string.h>
6
7 using namespace std;
8
9 #define NMAX 30
10 #define PMAX 401
11 #define LMAX 105
```

```

12
13 vector<char> NT[256], okNT[LMAX][LMAX], prod[NMAX][NMAX];
14 set<char> s;
15
16 char pleft[PMAX][10], pright[PMAX][10], sir[LMAX];
17 int i, j, k, R, M, L, len, oki, okj, okk;
18 char A, B, C;
19
20 int main()
21 {
22     freopen("minerale.in", "r", stdin);
23     freopen("minerale.out", "w", stdout);
24     scanf("%d %d", &R, &M);
25
26     for (i = 1; i <= R; i++)
27     {
28         scanf("%s %s", pleft[i], pright[i]);
29         L = strlen(pright[i]);
30         if (L == 1)
31             NT[pright[i][0]].push_back(pleft[i][0] - 'A');
32         else
33             prod[pright[i][0]-'A'][pright[i][1]-'A'].push_back(pleft[i][0]-'A');
34         //fprintf(stderr, "%s->%s\n", pleft[i], pright[i]);
35     }
36
37     while (M--)
38     {
39         scanf("%s", sir + 1);
40         L = strlen(sir + 1);
41         //fprintf(stderr, "sir=%s\n", sir + 1);
42
43         for (i = 1; i <= L; i++)
44         {
45             okNT[i][i].clear();
46             for (j = 0; j < NT[sir[i]].size(); j++)
47             {
48                 okNT[i][i].push_back(NT[sir[i]][j]);
49             }
50
51             /*
52             fprintf(stderr, "(%d,%d):", i, i);
53             for (k = 0; k < okNT[i][i].size(); ++k)
54                 fprintf(stderr, " %d", okNT[i][i][k]);
55             fprintf(stderr, "\n");
56             */
57         }
58
59         for (len = 2; len <= L; len++)
60             for (i = 1; i <= L - len + 1; i++)
61             {
62                 j = i + len - 1;
63                 okNT[i][j].clear();
64                 s.clear();
65
66                 for (k = i; k < j; k++)
67                 {
68                     for (oki = 0; oki < okNT[i][k].size(); oki++)
69                     {
70                         A = okNT[i][k][oki];
71                         for (okj = 0; okj < okNT[k + 1][j].size(); okj++)
72                         {
73                             B = okNT[k + 1][j][okj];
74                             //fprintf(stderr, "(%d,%d): %d %d\n", i, j, A, B);
75                             for (okk = 0; okk < prod[A][B].size(); okk++)
76                             {
77                                 C = prod[A][B][okk];
78                                 //fprintf(stderr, "(%d,%d): %d -> %d %d\n",
79                                         // i, j, C, A, B);
80
81                                 if (s.find(C) == s.end())
82                                 {
83                                     okNT[i][j].push_back(C);
84                                     s.insert(C);
85                                 }
86                             }
87                         }
88                     }
89                 }
90             }
91         }
92     }
93 }
```

```

88         }
89     }
90
91     /*
92     fprintf(stderr, "### (%d,%d):", i, j);
93     for (k = 0; k < okNT[i][j].size(); ++k)
94         fprintf(stderr, " %d", okNT[i][j][k]);
95     fprintf(stderr, "\n");
96 */
97 }
98
99 if (okNT[1][L].size() == 0)
100 {
101     printf("0\n");
102 }
103 else
104 {
105     for (i = 0; i < okNT[1][L].size(); i++)
106     {
107         if (okNT[1][L][i] == ('S' - 'A'))
108             break;
109     }
110
111     if (i < okNT[1][L].size())
112         printf("1\n");
113     else
114         printf("2\n");
115     }
116 }
117
118 return 0;
119 }
```

26.5.3 *Rezolvare detaliată

26.6 tarabe

Problema 6 - tarabe

100 de puncte

Sile Marele Cumpărător (SMC) a plecat la piață! El știe că piață este formată din N tarabe, fiecare tarabă având spre vânzare o infinitate de produse. De asemenea, SMC știe prețul inițial A_i al produsului de la taraba i și rația B_i cu care acest preț crește odată cu fiecare cumpărare a unui produs. Cu alte cuvinte, dacă primul produs cumpărat de la taraba i va avea costul A_i , al doilea va avea costul $A_i + B_i$, al treilea va avea costul $A_i + 2 * B_i$ etc.

Cerințe

SMC dorește să cumpere K produse, astfel încât suma prețurilor produselor cumpărate să fie minimă. Deoarece SMC nu știe aritmetică, voi trebui să îl ajutați, scriindu-i un program!

Date de intrare

Pe prima linie a fișierului **tarabe.in** se găsesc două numere naturale N și K , reprezentând numărul de tarabe și numărul de produse pe care SMC vrea să le cumpere, despărțite printr-un spațiu.

Urmează N linii, pe cea de-a i -a linie fiind scrise două numere naturale B_i și A_i reprezentând rația, respectiv costul inițial al unui produs la taraba i .

Date de ieșire

Pe prima și singura linie a fișierului de ieșire **tarabe.out** trebuie să afișați un singur număr, respectiv suma minimă a prețurilor pe care o poate obține SMC dacă va cumpara exact K produse.

Restricții și precizări

- $1 \leq N \leq 200\,000$;
- $1 \leq K \leq 1\,000\,000\,000$;
- $1 \leq A_i, B_i \leq 1\,000$;
- Pentru 20% din teste $N, K \leq 1\,000$;
- Pentru 60% din teste $N, K \leq 200\,000$;
- Atenție! SMC garantează că pentru rezultat pot fi folosite tipuri de date pe *64 de biți cu semn*.

Exemplu:

tarabe.in	tarabe.out	Explicații
4 7		În ordine, produsele cumpărate vor fi: de la taraba 3 cu costul 2, de la tabara 2 cu costul 2, de la taraba 1 cu costul 3, de la taraba 3 cu costul 2+5, de la taraba 4 cu costul 10, de la taraba 3 cu costul 2+5+5 și de la taraba 2 cu costul 2+10.
9 3		
10 2		
5 2		
4 10	48	Nu există nicio altă variantă care să asigure o sumă mai mică a prețurilor

Timp maxim de executare/test: **0.8** secunde

Memorie: total **64 MB**

Dimensiune maximă a sursei: **10 KB**

26.6.1 Indicații de rezolvare

stud. Andrei Parvu - Universitatea "Politehnica" Bucuresti
 stud. Tiberiu Savin - Universitatea Bucuresti
 stud. Bogdan Tataroiu - Cambridge University
 S.L. Dr. Ing. Mugurel Andreica - Universitatea "Politehnica" Bucuresti

Solutia 1 - 20 de puncte

Pentru fiecare dintre cele K produse, se parcurg cele N tarabe și se selectează tot timpul produsul cu pretul minim.

Complexitate: $O(N * K)$

Solutia 2 - 60 de puncte

Se menține un *heap* al celor mai mici N produse. La fiecare pas se extrage minimul din heap, se adaugă la soluție, se incrementează cu B_i -ul corespunzător și se introduce din nou în heap.

Complexitate: $O(K * \log N)$.

Solutia 3 - 100 de puncte

Ne vine ideea evidentă de a *cauta binar* pretul celui mai scump produs cumpărat.

Odată fixat acest pret, se pot calcula numărul de produse care au pretul mai mic sau egal decât acesta (pentru o taraba i acest număr este $\lceil (cost - fixat - B_i) / A_i \rceil + 1$). Dacă acest număr este mai mare decât K , atunci cautăm un cost final mai mic decât cel curent, iar dacă numărul de produse este mai mic decât K , cautăm un cost final mai mare.

Bineînteles, la fiecare pas, putem calcula și suma prețurilor tuturor produselor cu costul mai mic sau egal cu cel fixat.

La final, s-ar putea să obținem un număr de produse K' mai mare sau egal decât K , dar în acest caz, vom scădea din suma prețurilor acestor produse $(K' - K) * \text{pretul_maxim_produs}$.

Complexitate finală: $O(N * \log K)$.

26.6.2 Cod sursă

Listing 26.6.1: tarabe.cpp

```

1 /* Andrei Parvu
2  O(N * logK) */
3 #include <cstdio>
4 #include <queue>
5 #include <cassert>

```

```

6
7 using namespace std;
8
9 const int NMAX = 200000, KMAX = 1000000000;
10
11 #define x first
12 #define y second
13 #define ii pair<long long, int>
14
15 int N, K;
16 ii dreapta[NMAX + 5];
17 long long sumFound, totalFound, found;
18
19 void bs()
20 {
21     long long li = 1, ls = 1LL << 32, x;
22
23     while (li <= ls)
24     {
25         x = (li + ls) / 2;
26
27         long long total = 0, suma = 0;
28         for (int i = 1; i <= N; i++)
29         {
30             long long cate = 0;
31
32             if (x >= dreapta[i].y)
33                 cate = (x - dreapta[i].y) / dreapta[i].x + 1;
34
35             total += cate;
36             suma += cate * dreapta[i].y + (cate - 1) * cate / 2 * dreapta[i].x;
37         }
38
39         if (total >= K)
40         {
41             found = x;
42             sumFound = suma;
43             totalFound = total;
44             ls = x - 1;
45         }
46         else
47             li = x + 1;
48     }
49 }
50
51
52 int main()
53 {
54     freopen("tarabe.in", "rt", stdin);
55     freopen("tarabe.out", "wt", stdout);
56
57     assert(scanf("%d %d", &N, &K) == 2);
58     assert(1 <= N && N <= NMAX && 1 <= K && K <= KMAX);
59
60     for (int i = 1; i <= N; i++)
61     {
62         assert(scanf("%lld %d", &dreapta[i].x, &dreapta[i].y) == 2); //x*timp+y
63         assert(1 <= dreapta[i].x && dreapta[i].x <= 1000 &&
64             1 <= dreapta[i].y && dreapta[i].y <= 1000);
65     }
66
67     bs();
68
69     sumFound -= (totalFound - K) * found;
70
71     printf("%lld\n", sumFound);
72
73     return 0;
74 }
```

Listing 26.6.2: tarabe-20.cpp

```

1 /* Andrei Parvu
2      O(N * K) */
3 #include <cstdio>
```

```

4 #include <algorithm>
5
6 using namespace std;
7
8 const int NMAX = 200005;
9
10#define x first
11#define y second
12#define ii pair<long long, int>
13
14 int N, K;
15 ii dreapta[NMAX];
16 long long curent[NMAX], raspuns;
17
18 int main()
19 {
20     freopen("tarabe.in", "rt", stdin);
21     freopen("tarabe.out", "wt", stdout);
22
23     scanf("%d %d", &N, &K);
24
25     for (int i = 1; i <= N; i++)
26     {
27         scanf("%lld %d", &dreapta[i].x, &dreapta[i].y); //x*timp+y
28         curent[i] = dreapta[i].y;
29     }
30
31     for (int i = 1; i <= K; i++)
32     {
33         ii acum = make_pair(curent[1], 1);
34
35         for (int j = 2; j <= N; j++)
36             if (curent[j] < acum.x)
37                 acum = make_pair(curent[j], j);
38
39         raspuns += acum.x;
40         curent[acum.y] += dreapta[acum.y].x;
41     }
42
43     printf("%lld\n", raspuns);
44
45     return 0;
46 }
```

Listing 26.6.3: tarabe-60.cpp

```

1 /* Andrei Parvu
2      O(K * logN) */
3 #include <cstdio>
4 #include <queue>
5
6 using namespace std;
7
8 const int NMAX = 200005;
9
10#define x first
11#define y second
12#define ii pair<long long, int>
13
14 int N, K;
15 ii dreapta[NMAX], acum;
16 long long curent[NMAX], raspuns;
17
18 int main()
19 {
20     freopen("tarabe.in", "rt", stdin);
21     freopen("tarabe.out", "wt", stdout);
22
23     scanf("%d %d", &N, &K);
24
25     priority_queue<ii, vector<ii> > hp;
26
27     for (int i = 1; i <= N; i++)
28     {
29         scanf("%lld %d", &dreapta[i].x, &dreapta[i].y); //x * timp + y
```

```

30     curent[i] = dreapta[i].y;
31     hp.push(make_pair(-curent[i], i));
32 }
33
34 for (int i = 1; i <= K; i++)
35 {
36     ii acum = hp.top();
37     hp.pop();
38
39     raspuns -= acum.x;
40     curent[acum.y] += dreapta[acum.y].x;
41     hp.push(make_pair(-curent[acum.y], acum.y));
42 }
43
44 printf("%lld\n", raspuns);
45 return 0;
46 }
```

Listing 26.6.4: tarabe-100-mugurel.cpp

```

1 /* Mugurel Andreica */
2 #include <stdio.h>
3
4 #define NMAX 200001
5
6 int A[NMAX], B[NMAX];
7 long long cnt, li, ls, Cost, C, S = 0;
8 int i, j, N, K;
9
10 int ok(long long C, char compute_sum)
11 {
12     cnt = 0;
13     for (i = 1; i <= N; i++)
14     {
15         if (C < A[i])
16             continue;
17         long long x = ((C - A[i]) / B[i]) + 1;
18         cnt += x;
19         if (compute_sum)
20             S += x * A[i] + (x * (x - 1) / 2) * B[i];
21     }
22     return (cnt >= K);
23 }
24
25 int main()
26 {
27     freopen("tarabe.in", "r", stdin);
28     scanf("%d %d", &N, &K);
29     for (i = 1; i <= N; i++)
30         scanf("%d %d", &B[i], &A[i]);
31
32     C = 1;
33     while (!ok(C, 0))
34         C <<= 1;
35
36     if (C > 1)
37     {
38         li = C >> 1;
39         ls = C;
40         while (li <= ls)
41         {
42             Cost = (li + ls) >> 1;
43             if (ok(Cost, 0))
44             {
45                 C = Cost;
46                 ls = Cost - 1;
47             }
48             else
49                 li = Cost + 1;
50         }
51     }
52
53     ok(C, 1);
54     S -= C * (cnt - K);
55 }
```

```
56     freopen("tarabe.out", "w", stdout);
57     printf("%lld\n", S);
58
59     return 0;
60 }
```

26.6.3 *Rezolvare detaliată

Capitolul 27

ONI 2011

27.1 fotbal

Problema 1 - fotbal

100 de puncte

Se apropie sezonul competițional în Liga I de fotbal din Gheorgheni. Campionatul se desfășoară după următoarele reguli:

- În cazul în care după 90 de minute de joc scorul este egal, meciul va continua până când una din echipe va înscrie un gol, urmând ca aceasta să fie declarată câștigătoare. În acest fel nu vor exista meciuri terminate la egalitate.
- La finalul unui meci, echipa câștigătoare va primi 1 punct, iar echipa învinsă va pierde 1 punct. Programul competiției constă din M meciuri stabilite dinainte de către Federația de Fotbal din Gheorgheni. Este posibil ca unele echipe să nu joace deloc aşa cum de asemenea este posibil ca unele echipe să joace împreună de mai multe ori, chiar cu rezultate diferite.

Alexandra este foarte pasionată de fotbal și ca o microbiștă adevărată își dorește un campionat cât mai echilibrat. Astfel ea se întreabă care poate fi diferența minimă de punctaj între echipa aflată pe primul loc și echipa aflată pe ultimul loc în campionat după disputarea celor M meciuri. De asemenea ea ar vrea să stabilească câștigătorul fiecărui meci astfel încât să se obțină această diferență minimă de punctaj.

Cerințe

Alexandra este o fată foarte ocupată aşa că nu are timp să rezolve astfel de probleme. Totuși a fost de acord să vă spună vouă câte echipe sunt în campionat, numărul de meciuri care urmează să se dispute precum și echipele care se vor înfrunta în fiecare meci. Voi trebui să realizezi un program care să răspundă întrebărilor Andrei.

Date de intrare

Fișierul **fotbal.in** conține pe prima linie două numere naturale N și M , separate printr-un spațiu, reprezentând numărul de echipe din campionat respectiv numărul de meciuri programate. Pe următoarele M linii se vor afla câte două numere x și y , separate printr-un spațiu, perechea de pe linia $i + 1$ din fișier reprezentând echipele care se vor înfrunta în meciul i .

Date de ieșire

Fișierul **fotbal.out** va conține pe prima linie diferența minimă de scor între echipa aflată pe primul loc și echipa aflată pe ultimul loc în clasament.

Următoarele M linii vor conține câte un număr, numărul de pe linia $i + 1$ din fișier reprezentând numărul de ordine al echipei câștigătoare din meciul i .

Restricții și precizări

- $1 \leq N, M \leq 50.000$
- Nu va exista niciun meci în care o echipă să joace cu ea însăși
- Pentru 5% din punctaj $N = 2$
- Pentru 15% din punctaj $N, M \leq 20$
- Pentru 50% din punctaj $N, M \leq 2.000$
- Pentru determinarea diferenței minime se acordă 20% din punctaj

Exemplu:

fotbal.in	fotbal.out	Explicații
4 2	2	Primul meci este câștigat de echipa 1, iar al doilea meci este
1 2	1	câștigat de către echipa 3. La sfârșitul campionatului echipele 1
3 4	3	și 3 vor avea câte 1 punct, iar echipele 2 și 4 vor avea -1 puncte.

Timp maxim de executare/test: **2.0** secunde

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

27.1.1 Indicații de rezolvare

Tiberiu Savin - Universitatea Bucuresti

Vom construi un graf în care echipele reprezintă noduri, iar meciurile vor fi muchii.

Puteți observa că dacă alegem un *lanț simplu* $x_1 x_2 \dots x_k$ în acest graf și considerăm că echipa x_1 bate echipa x_2 , x_2 bate x_3 , ..., x_{k-1} bate x_k atunci toate echipele vor avea scorul 0 mai puțin echipa x_1 și x_k care vor avea 1 punct respectiv -1 puncte, mai puțin în cazul în care $x_k = x_1$ (*ciclu simplu*), caz în care toate echipele vor avea 0 puncte.

Astfel diferența minima dintre echipa clasată pe primul loc și echipa clasată pe ultimul loc poate fi ori 0, ori 2, dacă graful este sau nu eulerian.

In cazul în care *graful nu este eulerian* vom adăuga un nod fictiv în graf pe care îl vom conecta de nodurile cu grad impar transformând astfel graful într-un *graf eulerian*.

Vom construi un *ciclu eulerian* în acest graf și de fiecare dată cand vom trece prin nodul fictiv înseamnă ca acolo se termină un lant și începe un lant nou.

Este evident că un nod poate fi "capăt de lant" cel mult odată deoarece nu are sens să tragem mai mult de o muchie între un nod și nodul fictiv.

27.1.2 Cod sursă

Listing 27.1.1: fotbal_100.cpp

```

1 //Andrei Parvu
2 //O(N + M)
3 #include <iostream>
4 #include <vector>
5 #include <cassert>
6
7 using namespace std;
8
9 #define x first
10 #define y second
11
12 const int lg = 100005, MAXM = 200005;
13
14 int n, m, i, x, y, eulerian = 1, ind, prc[MAXM], care[MAXM], sol[lg];
15 vector< pair<int, int> > v[lg];
16
17 bool trecut[lg], scos[MAXM];
18
19 void ciclu(int nod, int muchie)
20 {
21     trecut[nod] = 1;
22
23     for (int i = 0; i < (int)v[nod].size(); i++)
24         if (!scos[v[nod][i].y])
25         {
26             scos[v[nod][i].y] = 1;
27             ciclu(v[nod][i].x, v[nod][i].y);
28         }
29
30     prc[++ind] = nod;
31     care[ind] = muchie;
32 }
33

```

```

34 int main()
35 {
36     freopen("fotbal.in", "rt", stdin);
37     freopen("fotbal.out", "wt", stdout);
38
39     assert( scanf("%d%d", &n, &m) == 2 );
40     assert(1 <= n && n <= 100000 && 1 <= m && m <= 100000);
41
42     for (i = 1; i <= m; i++)
43     {
44         assert( scanf("%d%d", &x, &y) == 2 );
45         assert(1 <= x && x <= n && 1 <= y && y <= n);
46
47         v[x].push_back(make_pair(y, i));
48         v[y].push_back(make_pair(x, i));
49     }
50
51     for (i = 1; i <= n; i++)
52     {
53         if (v[i].size() % 2 == 1)
54         {
55             v[i].push_back(make_pair(n + 1, m + i));
56             v[n + 1].push_back(make_pair(i, m + i));
57             eulerian = 0;
58         }
59
60         for (i = 1; i <= n; i++)
61         {
62             if (!trecut[i])
63                 ciclu(i, 0);
64
65             if (eulerian == 1)
66                 printf("0\n");
67             else
68                 printf("2\n");
69
70             for (i = ind - 1; i; i--)
71             {
72                 if (0 < care[i] && care[i] <= m)
73                     sol[ care[i] ] = prc[i];
74
75             }
76             printf("%d\n", sol[i]);
77
78         }
79     }

```

Listing 27.1.2: fotbal100.cpp

```

1 //Tibi Savin
2 //O(N + M)
3 #include <stdio.h>
4 #include <vector>
5
6 using namespace std;
7
8 #define NMAX 200002
9
10 int N, M, R; // R e nodul fictiv
11 vector< pair<int, int> > G[NMAX];
12 int gr[NMAX];
13 int winner[NMAX];
14 int used_edges[NMAX];
15 int used_nodes[NMAX];
16 int position;
17
18 void euler(int nod)
19 {
20     used_nodes[nod] = 1;
21
22     for (int i = 0; i < G[nod].size(); i++)
23     {
24         int node, edge;
25
26         node = G[nod][i].first;

```

```

27     edge = G[nod][i].second;
28
29     if (used_edges[edge])
30         continue;
31
32     used_edges[edge] = 1;
33     euler(node);
34     winner[edge] = node;
35 }
36 }
37
38 int main()
39 {
40     freopen("fotbal.in", "r", stdin);
41     freopen("fotbal.out", "w", stdout);
42
43     scanf("%d %d ", &N, &M);
44
45     for (int i = 1; i <= M; i++)
46     {
47         int x, y;
48         scanf("%d %d ", &x, &y);
49
50         G[x].push_back(make_pair(y, i));
51         G[y].push_back(make_pair(x, i));
52
53         gr[x]++;
54         gr[y]++;
55     }
56
57     R = 0;
58     int score_diff = 0;
59     int cnt = M;
60     for (int i = 1; i <= N; i++)
61     {
62         if (gr[i] % 2 == 1)
63         {
64             score_diff = 2;
65             G[R].push_back(make_pair(i, ++cnt));
66             G[i].push_back(make_pair(R, cnt));
67         }
68     }
69
70     for (int i = 0; i <= N; i++)
71         if (!used_nodes[i])
72             euler(i);
73
74     printf("%d\n", score_diff);
75     for (int i = 1; i <= M; i++)
76         printf("%d\n", winner[i]);
77
78     return 0;
79 }
```

27.1.3 *Rezolvare detaliată

27.2 ikebana

Problema 2 - ikebana

100 de puncte

O florărie vrea să ajungă în Guinness book cu cel mai mare aranjament floral. Ei au la dispoziție t tipuri de flori, dintre care patru tipuri sunt mai speciale: gerbera, orhideea, azaleea și hortensia. Lucrătorii au hotărât să pună florile distanțate uniform pe mai multe rânduri, pe fiecare rând exact n flori. Nu vor exista două rânduri identice, dar toate rândurile vor respecta anumite cerințe:

- Ei au observat că hortensiile au o viață mult mai îndelungată, dacă se învecinează pe același rând cu o azalee și cu o orhidee, indiferent dacă ordinea este azalee-hortensie-orhidee sau orhidee-hortensie-azalee.

- Gerberele vor fi amplasate în aşa fel încât între oricare două gerbere să existe cel puțin p flori, fiind oricare dar nu gerbera.

De exemplu dacă avem la dispoziție $t = 5$ tipuri de flori: azalee (notate cu a), hortensii (notate cu h), orhidee (notate cu o), gerbere (notate cu g), și begonii (notate cu b), între două gerbere se vor amplasa minim $p = 3$ flori, iar rândul va fi format din $n = 6$ flori, atunci următoarele aranjamente florale sunt corecte: "aoaaoo", "ahohag", "gbbbgo", "gbbbog", "bbbbbb".

Următoare aranjamente nu sunt însă corecte: "ohoaha" (hortensiile nu sunt între o orhidă și o azaleă), "gogbao" (cele două gerbere nu sunt despărțite de minim trei flori), "ahohah" (ultima hortensie nu se înceinează cu o orhidă).

Pentru $n = 6$, $p = 3$, $t = 5$, numărul diferitelor aranjamente florale este 2906.

Cerințe

Cunoscând valorile lui n , p și t , să se determine numărul liniilor distincte ce se pot obține cu cerințele de mai sus.

Date de intrare

Fișierul **ikebana.in** conține pe un singur rând 3 numere naturale n , p și t separate prin câte un spațiu.

n - reprezintă numărul de flori de pe un rând;

p - numărul minim de flori ce trebuie să despartă două gerbere dintr-un rând;

t - numărul tipurilor de flori distincte ce stau la dispoziția florăriei.

Date de ieșire

Fișierul **ikebana.out** va conține pe unicul rând un singur număr: numărul de aranjări distincte modulo 666013.

Restricții și precizări

- $1 \leq n \leq 1.000.000.000$;
- $3 \leq p \leq 20$;
- $4 \leq t \leq 20$;

Exemple:

ikebana.in	ikebana.out	Explicații
6 3 5	2906	Numărul aranjamentelor distincte este de 2906
10 6 8	620160	Numărul aranjamentelor distincte este de 181775696, și avem: $181775696 \% 666013 = 620160$

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

Memorie: total **32 MB** din care pentru stivă **1 MB**

27.2.1 Indicații de rezolvare

prof. Szabo Zoltan - Gr. Sc. "Petru Maior" Reghin)

1. Soluție de 20 de puncte

Se rezolvă cu metoda *backtracking*, generând fiecare soluție în parte.

2. Soluție de 60 de puncte ($O(n)$)

Pornim de la următoarele formule recursive cu semnificația că $a[i][j]$ reprezintă numărul de aranjamente florale ce conțin j flori, pe ultima poziție fiind o azalee, iar între ea și cea mai apropiată gerberă din față sa se găsească $i - 1$ flori.

$o[i][j]$ reprezintă numărul de aranjamente florale ce conțin j flori, pe ultima poziție fiind o orhidă, iar între ea și cea mai apropiată gerberă din față sa se găsească $i - 1$ flori.

$ah[i][j]$ reprezintă numărul de aranjamente florale ce conțin j flori, pe ultimele două poziții fiind o azalee și o hortensie, iar între ultima hortensie și cea mai apropiată gerberă din față sa se găsească $i - 1$ flori.

$oh[i][j]$ reprezintă numărul de aranjamente florale ce conțin j flori, pe ultimele două poziții fiind o orhidee și o hortensie, iar între ultima hortensie și cea mai apropiată gerberă din față să se găsească $i - 1$ flori.

$gl[j]$ reprezintă numărul de aranjamente florale ce conțin j flori, pe ultima poziție fiind o gerberă.

$b[i][j]$ reprezintă numărul de aranjamente florale ce conțin j flori, pe ultima poziție fiind o floare diferită de azalee, orhidee, hortensie și gerberă, iar între ea și cea mai apropiată gerberă din față să se găsească $i - 1$ flori.

Formulele recursive pentru fiecare tip de floare în parte (valorile inițiale le depistați singuri):	Având în vedere că $a[i][j] = o[i][j]$ și $ah[i][j] = oh[i][j]$ pentru orice $i \neq j$, numărul de siruri recursive se reduc astfel:	
$b[i][j] = a[i-1][j-1] + o[i-1][j-1] + t*b[i-1][j-1] \quad pt\ j=2..n$ $\quad \quad \quad \quad pt\ i=2..p-1$	$b[i][j] = 2*a[i-1][j-1] + t*b[i-1][j-1] \quad pt\ j=2..n$ $\quad \quad \quad \quad pt\ i=2..p-1$	
$a[i][j] = a[i-1][j-1] + o[i-1][j-1] + oh[i-1][j-1] + t*b[i-1][j-1] \quad pt\ j=2..n$ $\quad \quad \quad \quad pt\ i=2..p-1$	$a[i][j] = 2*a[i-1][j-1] + ah[i-1][j-1] + t*b[i-1][j-1] \quad pt\ j=2..n$ $\quad \quad \quad \quad pt\ i=2..p-1$	
$o[i][j] = a[i-1][j-1] + o[i-1][j-1] + ah[i-1][j-1] + t*b[i-1][j-1] \quad pt\ j=2..n$ $\quad \quad \quad \quad pt\ i=2..p-1$		
$ah[i][j] = a[i-1][j-1] \quad pt\ j=2..n$ $\quad \quad \quad \quad pt\ i=2..p-1$	$ah[i][j] = a[i-1][j-1] \quad pt\ j=2..n$ $\quad \quad \quad \quad pt\ i=2..p-1$	
$oh[i][j] = o[i-1][j-1] \quad pt\ j=2..n$ $\quad \quad \quad \quad pt\ i=2..p-1$		
$b[p][j] = a[p-1][j-1] + o[p-1][j-1] + t*b[p-1][j-1] + a[p][j-1] + o[p][j-1] + t*b[p][j-1] \quad pt\ j=2..n$	$b[p][j] = 2*a[p-1][j-1] + t*b[p-1][j-1] + 2*a[p][j-1] + t*b[p][j-1] \quad pt\ j=2..n$	
$a[p][j] = a[p-1][j-1] + o[p-1][j-1] + oh[p-1][j-1] + t*b[p-1][j-1] + a[p][j-1] + o[p][j-1] + oh[p][j-1] + t*b[p][j-1] \quad pt\ j=2..n$	$a[p][j] = 2*a[p-1][j-1] + ah[p-1][j-1] + t*b[p-1][j-1] + 2*a[p][j-1] + ah[p][j-1] + t*b[p][j-1] \quad pt\ j=2..n$	
$o[p][j] = a[p-1][j-1] + o[p-1][j-1] + oh[p-1][j-1] + t*b[p-1][j-1] + a[p][j-1] + o[p][j-1] + oh[p][j-1] + t*b[p][j-1] \quad pt\ j=2..n$		
$ah[p][j] = a[p-1][j-1] + a[p][j-1] \quad pt\ j=2..n$	$ah[p][j] = a[p-1][j-1] + a[p][j-1] \quad pt\ j=2..n$	
$oh[p][j] = o[p-1][j-1] + o[p][j-1] \quad pt\ j=2..n$		
$gl[j] = a[j-1][j-1] + o[j-1][j-1] + t*b[j-1][j-1] \quad pt\ j=2..p$	$gl[j] = 2*a[j-1][j-1] + t*b[j-1][j-1] \quad pt\ j=2..p$	

$g[j] = a[p][j-1] + o[p][j-1] + t * b[p][j-1] \quad \text{pt } j = p+1 \dots n$	$g[j] = 2 * a[p][j-1] + t * b[p][j-1] \quad \text{pt } j = p+1 \dots n$
---	---

Soluția problemei este $g[n] + 2 * a[i][n] + t * b[i][n]$ $i = 1..p$

3. Soluție de 100 de puncte ($O(p^3 * \log n)$)

Din recurența anterioară construim formula cu înmulțire de matrici de forma

$$A^{n-p} * B[p] = B[n],$$

unde matricea A este de dimensiuni $3p \times 3p$, iar matricile $B[p]$ și $B[n]$ sunt de dimensiune $3p \times 1$.

Exponențierea se poate rezolva în $O(\log n)$, deci această metodă are complexitatea $O(p^3 * \log n)$.

27.2.2 Cod sursă

Listing 27.2.1: ikebana-100.c

```

1  /* Tataroiu Bogdan-Cristian */
2  #include <stdio.h>
3  #include <string.h>
4  #include <assert.h>
5
6  #define MAXN 1000000000
7  #define MAXK 20
8  #define MAXT 20
9  #define MOD 666013
10
11 #define cell(j, k) ((j) * 3 + (k))
12 #define var_count (cell(MAXK, 2) + 1)
13
14 void mul(int A[var_count][var_count], int B[var_count][var_count])
15 {
16     int C[var_count][var_count], i, j, k;
17
18     memset(C, 0, sizeof(C));
19
20     for (k = 0; k < var_count; k++)
21         for (i = 0; i < var_count; i++)
22             for (j = 0; j < var_count; j++)
23                 C[i][j] = (C[i][j] + A[i][k] * (long long)B[k][j]) % MOD;
24
25     memcpy(A, C, sizeof(C));
26 }
27
28 void toPower(int res[var_count][var_count], int M[var_count][var_count],
29               int power)
30 {
31     if (power == 0)
32     {
33         memset(res, 0, sizeof(int) * var_count * var_count);
34         int i;
35         for (i = 0; i < var_count; i++)
36             res[i][i] = 1;
37
38         return;
39     }
40
41     if (power & 1)
42     {
43         toPower(res, M, power - 1);
44         mul(res, M);
45     }
46     else
47     {
48         toPower(res, M, power >> 1);
49         mul(res, res);
50     }
51 }
52

```

```

53 int main()
54 {
55     assert(freopen("ikebana.in", "rt", stdin));
56     assert(freopen("ikebana.out", "wt", stdout));
57
58     int N, K, T;
59     assert(scanf("%d %d %d", &N, &K, &T) == 3);
60     assert(1 <= N && N <= MAXN);
61     assert(3 <= K && K <= MAXK);
62     assert(4 <= T && T <= MAXT);
63
64     /*
65      * cnt[i][j][k] = numarul de aranjamente de lungime i, cu ultima gherbere
66      * plasata acum j pozitii si ultima planta plasata de tipul k
67      *   k == 0 pentru azalee sau orhidee
68      *   k == 1 pentru hortensie
69      *   k == 2 pentru celelalte tipuri de plante
70      * Modelam relatia de recurenta sub forma unui sistem matriceal:
71      * REC * cnt(i) = cnt(i + 1)
72      * unde cnt(i) va contine elementele liniarizate sub forma unei coloane de
73      * "var_count" elemente.
74     */
75     int init[var_count], rec[var_count][var_count], final[var_count];
76     memset(init, 0, sizeof(init));
77     memset(rec, 0, sizeof(rec));
78     memset(final, 0, sizeof(final));
79
80     /* Construieste starea initiala */
81     init[cell(K, 0)] = 2;
82     init[cell(K, 1)] = 0;
83     init[cell(0, 2)] = 1;
84     init[cell(K, 2)] = (T - 4);
85
86     /* Construieste matricea de recurenta */
87     int j, k;
88     for (j = 1; j <= K; j++)
89     {
90         /*
91          cnt(i, j, 0) += (cnt(i - 1, j - 1, 0) + cnt(i - 1, j - 1, 2)) * 2 + \
92                      cnt(i - 1, j - 1, 1);
93        */
94        rec[cell(j, 0)][cell(j - 1, 0)] = 2;
95        rec[cell(j, 0)][cell(j - 1, 2)] = 2;
96        rec[cell(j, 0)][cell(j - 1, 1)] = 1;
97
98        /*
99          cnt(i, j, 1) += cnt(i - 1, j - 1, 0);
100        */
101       rec[cell(j, 1)][cell(j - 1, 0)] = 1;
102
103       /*
104          cnt(i, j, 2) += (cnt(i - 1, j - 1, 0) + cnt(i - 1, j - 1, 2)) * \
105                      (T - 4);
106        */
107       rec[cell(j, 2)][cell(j - 1, 0)] = (T - 4);
108       rec[cell(j, 2)][cell(j - 1, 2)] = (T - 4);
109    }
110
111    /* cnt(i, K, ?) += cnt(i - 1, K, ?) */
112    /*
113      cnt(i, K, 0) += (cnt(i - 1, K, 0) + cnt(i - 1, K, 2)) * 2 + \
114                  cnt(i - 1, K, 1);
115    */
116    rec[cell(K, 0)][cell(K, 0)] = 2;
117    rec[cell(K, 0)][cell(K, 2)] = 2;
118    rec[cell(K, 0)][cell(K, 1)] = 1;
119
120    /*
121      cnt(i, K, 1) += cnt(i - 1, K, 0);
122    */
123    rec[cell(K, 1)][cell(K, 0)] = 1;
124
125    /*
126      cnt(i, 0, 2) += (cnt(i - 1, K, 0) + cnt(i - 1, K, 2));
127    */
128    rec[cell(0, 2)][cell(K, 0)] = 1;

```

```

129     rec[cell(0, 2)][cell(K, 2)] = 1;
130
131     /*
132     cnt(i, K, 2) += (cnt(i - 1, K, 0) + cnt(i - 1, K, 2)) * (T - 4);
133     */
134     rec[cell(K, 2)][cell(K, 0)] = (T - 4);
135     rec[cell(K, 2)][cell(K, 2)] = (T - 4);
136
137     /* Calculeaza starea finala */
138     /* final = rec ^ (N - 1) * init */
139     int power[var_count][var_count];
140     toPower(power, rec, N - 1);
141     for (j = 0; j < var_count; j++)
142     {
143         final[j] = 0;
144         for (k = 0; k < var_count; k++)
145         {
146             final[j] = (final[j] + power[j][k] * init[k]) % MOD;
147         }
148     }
149
150     int SUM = 0;
151     /* Sum of cnt(N, j, k), 0 <= j <= J, k in {0, 2} */
152     for (j = 0; j <= K; j++)
153         for (k = 0; k < 3; k += 2)
154         {
155             if (k == 1)
156                 continue;
157             SUM += final[cell(j, k)];
158             if (SUM >= MOD)
159                 SUM -= MOD;
160         }
161     printf("%d\n", SUM);
162
163     return 0;
164 }
```

27.2.3 *Rezolvare detaliată

27.3 posta

Problema 3 - posta

100 de puncte

Într-o țară minunată, cele N orașe sunt legate între ele prin $N - 1$ șosele astfel încât din fiecare oraș se poate ajunge în oricare alt oraș. Se știe costul benzinei necesare pentru parcurgerea fiecărei șosele și costul de intrare în fiecare oraș. În rîbă, managerul poștei, trebuie să aleagă sediul poștei, știind că va avea de livrat colete în M orașe precizate, plecând de la sediul poștei și revenind după livrarea coletelor tot la sediul poștei.

El trebuie să aleagă sediul astfel încât să minimizeze costul transporturilor, ținând cont că poșta va întocmi un contract cu guvernul prin care va fi scutită de:

- toate taxele de intrare din orașul în care își stabilește sediul;
- prima intrare în oricare alt oraș, iar pentru restul intrărilor se plătește taxa.

Cerințe

Cunoscând numărul de orașe, șoselele, taxele de intrare în fiecare oraș și cele M orașe în care se livrează coletele, ajutați-l pe rîbă să calculeze costul minim necesar pentru livrarea comenziilor.

Date de intrare

Fișierul de intrare **posta.in** conține pe prima linie două numere naturale N și M separate printr-un spațiu, cu semnificația din enunț. Pe următoarele $N - 1$ linii se vor afla câte trei numere x, y, z , separate prin câte un spațiu, cu semnificația că există șosea de la orașul x la orașul y având costul z . Pe următoarea linie se află N numere naturale reprezentând taxa de intrare din fiecare oraș. Ultima linie conține M numere naturale reprezentând orașele în care poșta trebuie să livreze comenzi.

Date de ieșire

Fișierul de ieșire **posta.out** conține costul minim al unui transport.

Restricții și precizări

- $2 \leq M \leq N \leq 100.000$
- Toate taxele și costurile sunt numere naturale strict pozitive mai mici sau egale cu 100.000.
- Mașina poate trece prin oricare oraș sau pe orice șosea de oricâte ori.
- Orașele în care livrează comenzi sunt distințe.
- Pentru 10% din teste $M \leq 3$
- Pentru 30% din teste $N \leq 1.000$

Exemple:

posta.in	posta.out	Explicații
7 3		Se va alege sediul în orașul 1 și se va parcurge următorul traseu: 1 → 2 → 4 → 2 → 1 → 5 → 6 → 5 → 1
1 2 3		Costul benzinei este 26
2 3 5		Costul taxelor este 2 (orașul 2 + orașul 5)
2 4 2		în orașele 4 și 6 nu se plătește taxă deoarece se intră o singură dată.
4 7 4		
1 5 7		
5 6 1		
2 1 1 2 1 2 1		
1 4 6		

Timp maxim de executare/test: **0.3** secunde

Memorie: total **32 MB** din care pentru stivă **1 MB**

27.3.1 Indicații de rezolvare

Cosmin-Mihai Tutunaru - Universitatea Babes-Bolyai

Se definește un nod special ca fiind un oraș în care trebuie livrate colete.

Soluție de 30 pct

Se încearcă fiecare nod ca sediu și vom face o *parcuregere în adâncime* din acel nod pentru a afla costul.

Din fiecare nod vom parcurge mai întâi toți fii recursiv ca să aflăm costul transportului în fiecare subarbore.

Pentru a calcula costul unui subarbore X , trebuie să calculăm

$nrMuchii =$ numărul de muchii incidente în X ce duc către un subarbore în care avem cel puțin un oraș special,

iar $costMuchii$ este suma costurilor tuturor muchiilor numărate.

Acum, costul subarborelui X este $(nrMuchii - 1) * Taxa[X] + costMuchii * 2$.

Vom alege minimul dintre toate rădăcinile arborelui.

Complexitate: $O(N^2)$

Soluție de 60-70 pct

Soluția este asemănătoare cu cea anterioară, având următoarele modificări:

- Când facem parcurgerea dintr-o anumită rădăcină, salvăm costul pentru fiecare subarbore având ca tată nodul din care este parcurs, iar ulterior dacă vom mai ajunge în acel subarbore având același tată nu va mai trebui să calculăm în continuare pentru restul nodurilor.
- Când alegem rădăcinile arborelui, le vom alege în ordinea unei parcurgeri în adâncime, pentru a nu se schimba des salvările făcute în noduri.

Complexitate: $O(N * maxFii)$, unde $maxFii$ este numărul maxim de fii ai unui nod.

Soluție de 100 pct

Vom elimina toți subarborii ce nu conțin niciun nod special, cu ajutorul unei parcurgeri. Trebuie să pornim parcurgerea dintr-un nod special, deoarece altfel riscăm ca nodul din care plecăm să trebuiască să-l eliminăm. Pentru fiecare nod X rămas în arbore definim $inc[X] =$ numărul de muchii (rămase în arbore) incidente în el. De asemenea, mai trebuie calculat $cost =$ suma costurilor tuturor muchiilor rămase în arbore. Dacă și în orașul sediu s-ar plăti intrările ca în oricare alt oraș, soluția ar fi:

Mai rămâne să calculăm costul pentru toate orașele ca sediu, scăzând din suma anterioară taxele fiecărui oraș.

Complexitate: $O(N)$

27.3.2 Cod sursă

Listing 27.3.1: posta_100.cpp

```

1 #define TuTTY "Cosmin-Mihai Tutunaru"
2 #include<cstdio>
3 #include<vector>
4 #include<algorithm>
5
6 #define infile "posta.in"
7 #define outfile "posta.out"
8 #define nMax 100013
9 #define ll long long
10
11 using namespace std;
12
13 vector < pair < int, int > > v[nMax];
14 int directions[nMax];
15 int tax[nMax];
16 bool w[nMax];
17 int n, m;
18 ll sol = ((ll)1<<61);
19
20 void read()
21 {
22     scanf("%d %d\n", &n, &m);
23
24     for(int i = 2; i <= n; ++i)
25     {
26         int x, y, z;
27         scanf("%d %d %d\n", &x, &y, &z);
28         v[x].push_back(make_pair(y, z));
29         v[y].push_back(make_pair(x, z));
30     }
31
32     for(int i = 1; i <= n; ++i)
33         scanf("%d", &tax[i]);
34
35     for(int i = 1; i <= m; ++i)
36     {
37         int x;
38         scanf("%d", &x);
39         w[x] = true;
40     }
41 }
42
43 ll dfs(int x, int f)
44 {
45     ll val = 0;
46     int count = 0;
47     for(unsigned i = 0; i < v[x].size(); ++i)
48         if(v[x][i].first != f)
49         {
50             ll cur = dfs(v[x][i].first, x);
51             if(cur || w[v[x][i].first])
52                 val += cur + v[x][i].second * 2, count++;
53         }
}
```

```

54     directions[x] = count;
55     return val;
56 }
58
59 void solve()
60 {
61     if(m == 0)
62     {
63         sol = 0;
64         return;
65     }
66
67     ll dist = 0, taxes = 0;
68     for(int i = 1; i <= n; ++i)
69     {
70         if(w[i])
71         {
72             dist = dfs(i, 0);
73             directions[i] -= 1;
74             break;
75         }
76
77         for(int i = 1; i <= n; ++i)
78             taxes += (ll)tax[i] * (ll)directions[i];
79
80         sol = min(sol, dist + taxes - (ll)tax[i] * (ll)directions[i]);
81     }
82
83 void write()
84 {
85     printf("%lld\n", sol);
86 }
87
88 int main()
89 {
90     freopen(infile, "r", stdin);
91     freopen(outfile, "w", stdout);
92
93     read();
94     solve();
95     write();
96
97     fclose(stdin);
98     fclose(stdout);
99     return 0;
100 }
```

Listing 27.3.2: posta100.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <cstdio>
4 #include <vector>
5 #include <algorithm>
6 #include <cassert>
7
8 using namespace std;
9
10 const char iname[] = "posta.in";
11 const char oname[] = "posta.out";
12
13 const int MAX_N = 100005;
14 const int MAX_C = 100000;
15
16 typedef long long i64;
17
18 vector < pair <int, int> > adj[MAX_N];      int nodes, inodes;
19 vector <int> cst, im, vis, cntv;
20
21
22 pair <int, i64> go(int u)
23 {
24     vis[u] = true;
25 }
```

```

26     vector<pair<int, int>>::iterator it;
27     i64 cost = 0;
28
29     for (it = adj[u].begin(); it != adj[u].end(); ++ it)
30     {
31         int v = (*it).first;
32         if (vis[v] == false)
33         {
34             pair<int, i64> ret = go(v);
35             if (ret.first == false)
36                 continue;
37             cost += ret.second + (*it).second * 2;
38             cntv[u]++;
39         }
40     }
41
42     cost += cntv[u] * cst[u];
43     return make_pair(im[u] || cost, cost);
44 }
45
46 int main()
47 {
48     FILE *fi = fopen(iname, "r");
49
50     assert(fscanf(fi, "%d %d", &nodes, &inodes) == 2);
51     assert(1 <= nodes && nodes <= MAX_N - 5);
52     assert(1 <= inodes && inodes <= nodes);
53
54     for (int i = 1; i < nodes; ++ i)
55     {
56         int u, v, cst;
57         assert(fscanf(fi, "%d %d %d", &u, &v, &cst) == 3);
58         assert(1 <= u && u <= nodes);
59         assert(1 <= v && v <= nodes);
60         assert(1 <= cst && cst <= MAX_C);
61         adj[u].push_back(make_pair(v, cst));
62         adj[v].push_back(make_pair(u, cst));
63     }
64
65     cst.resize(nodes + 1);
66     for (int i = 1; i <= nodes; ++ i)
67     {
68         assert(fscanf(fi, "%d", &cst[i]) == 1);
69         assert(1 <= cst[i] && cst[i] <= MAX_C);
70     }
71
72     im.resize(nodes + 1);
73     im.assign(im.size(), 0);
74
75     int start = 0;
76     for (int i = 1; i <= inodes; ++ i)
77     {
78         int u;
79         assert(fscanf(fi, "%d", &u) == 1);
80         assert(1 <= u && u <= nodes);
81         im[u] = true;
82         start = u;
83     }
84
85     fclose(fi);
86
87     vis.resize(nodes + 1);
88     vis.assign(vis.size(), 0);
89     cntv.resize(nodes + 1);
90     cntv.assign(cntv.size(), 0);
91
92     pair<int, i64> ret = go(start);
93
94     assert(ret.first == true);
95     ret.second -= cst[start];
96
97     i64 minres = 1LL << 62;
98     for (int i = 1; i <= nodes; ++ i)
99     {
100         if (cntv[i])
101             minres = min(minres, ret.second - ((i64) (cntv[i])) * cst[i]);
101

```

```

102     fprintf(fopen(online, "w"), "%lld\n", minres);
103
104     return 0;
105 }
```

27.3.3 *Rezolvare detaliată

27.4 pamant

Problema 4 - pamant

100 de puncte

Pe pământul din apropierea localității Gheorgheni s-au întâlnit toți copiii și doresc organizarea unui joc mai deosebit. Copiii au fost numerotați de la 1 la N și știm care sunt prietenii fiecărui copil.

O echipă este un grup maximal de copii cu proprietatea că oricare ar fi copiii P și Q din echipă, există un sir de copii C_1, \dots, C_k astfel încât $P = C_1, Q = C_k$, și oricare ar fi $1 \leq i < k$, C_i este prieten cu C_{i+1} .

Fiecare echipă va primi un cod, egal cu cel mai mic număr de ordine al unui copil din echipa respectivă.

Dorim să aflăm care sunt copiii vulnerabili, adică acei copii care dacă ar fi eliminați ar duce la spargerea echipei sale în două sau mai multe echipe.

Cerințe

Scrieți un program care să identifice toate echipele formate conform regulilor de mai sus, precum și care sunt copiii vulnerabili.

Date de intrare

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

- pe prima linie două numere naturale N și M reprezentând numărul de copii și respectiv numărul relațiilor de prietenie.
- următoarele M linii conțin câte două numere naturale distincte x și y , cu semnificația că x și y sunt numerele de ordine a doi copii în relație de prietenie.

Date de ieșire

- Prima linie a fișierului de ieșire **pamant.out** conține o singură valoare naturală A , reprezentând numărul de echipe.
- A doua linie conține A numere naturale separate prin câte un spațiu reprezentând codurile echipelor, în ordine crescătoare.
- A treia linie conține o valoare naturală B reprezentând numărul de copii vulnerabili.
- A patra linie conține B valori naturale, separate prin câte un spațiu, reprezentând numerele de ordine, scrise în ordine crescătoare, ale copiilor vulnerabili.

Restricții și precizări

- $1 \leq N \leq 100.000$
- $1 \leq M \leq 200.000$
- Se acordă 40% din punctaj pentru corectitudinea primelor două linii din fișierul de ieșire și 60% pentru celelalte două linii.
 - Relațiile de prietenie sunt reciproce: dacă x este prieten cu y , atunci și y este prieten cu x .
 - Dacă x este prieten cu y și y este prieten cu z nu înseamnă că x este prieten cu z .
 - Pentru 30% din teste $N \leq 1.000$.

Exemple:

pamant.in	pamant.out	Explicații
10 7	4	Există 4 echipe și anume: - prima echipă: 1 2 8
1 2	1 3 4 9	- a doua echipă: 3 5 7 10
2 8	2	- a treia echipă: 4 6
4 6	2 5	- a patra echipă: 9
3 5		
3 10		Există doi copii speciali și anume 2 și 5.
5 10		
5 7		

Timp maxim de executare/test: **0.4** secunde

Memorie: total 32 MB din care pentru stivă 8 MB

27.4.1 Indicații de rezolvare

prof. Carmen Popescu, Colegiul National "Gheorghe Lazar", Sibiu

Problema este modelată cu ajutorul unui *graf neorientat*.

Prima parte a cerintei constă în determinarea *componentelor conexe*.

Algoritmul de determinare a *punctelor critice* (copiii speciali) se bazează pe *parcurgerea DF* a grafului.

Se pornește de la primul nod nevizitat și parcugem DF graful, incrementând totodată numărul de componente.

Un nod x NU este punct de articulație dacă fiecare fiu y al său are printre descendenți vreun nod conectat cu un nod situat în arbore deasupra lui x .

Rădăcina este punct de articulație dacă are cel puțin doi fi.

Pentru punctaj maxim determinarea componentelor conexe se va face simultan cu determinarea punctelor de articulație, determinând la parcurgere în plus cea mai mică etichetă a nodurilor atinse.

27.4.2 Cod sursă

Listing 27.4.1: pamant0.cpp

```

1 // prof. Carmen Popescu
2 #include <iostream>
3 #include <fstream>
4 #include <set>
5 #include <vector>
6
7 using namespace std;
8
9 #define MAX 200000
10 ifstream f("pamant.in");
11 ofstream g("pamant.out");
12
13 int nr,n,m,d[MAX],v[MAX],start,nrfii,nc=0,nrp=0,k,mn;
14
15 set<int> s[MAX];      // liste de adiacenta
16 set<int> art;          // multimea punctelor de articulatie
17 set<int> comp;         // codurile proprietatilor
18
19 void VIZ(int i,int tata)
20 {
21     set<int>::iterator it;
22     int j;
23
24     if (i<mn)
25         mn=i;
26
27     d[i]=++nr;
28     v[i]=nr;
29
30     for (it=s[i].begin(); it!=s[i].end(); it++)
31     {
32         j=*it;

```

```

33         if (d[j]==0)
34     { // j este "atins" pentru prima data
35         if (i==start)
36             nrfii++;
37
38         VIZ(j,i);
39
40         if (v[j]>=d[i])
41     { // i e punct de articulatie
42             if (tata!=-1)
43                 art.insert(i);
44         }
45         if (v[j]<v[i])
46             v[i]=v[j];
47     }
48     else
49         if (j!=tata)
50             if (v[j]<v[i])      // (i,j) = muchie de intoarcere
51                 v[i]=v[j];
52     }
53 }
54
55 void citire()
56 {
57     int x,y;
58     f>>n>>m;
59     for (int i=0;i<m;i++)
60     {
61         f>>x>>y;
62         s[x].insert(y);
63         s[y].insert(x);
64     }
65 }
66
67 int main()
68 {
69     int i;
70     set<int>::iterator it;
71     set<int>::iterator it1;
72     citire();
73
74     // pentru fiecare componenta conexa se cauta punctele de articulatie
75     nr=k=0;
76     for (i=1;i<=n;i++)
77     {
78         mn=n+1;
79         if (d[i]==0)
80         {
81             start=i;
82             nrfii=0;
83             VIZ(start,-1);
84
85             // radacina are cel putin 2 fii => pct de articulatie
86             if (nrfii>1)
87                 art.insert(start);
88
89             k++;
90             comp.insert(mn);
91         }
92     }
93
94     g<<k<<"\n";
95     for (it=comp.begin();it!=comp.end();it++)
96         g<<*it<<" ";
97     g<<"\n";
98
99     // numarul de puncte de articulatie
100    g<<art.size()<<"\n";
101
102    // afisarea punctelor de articulatie
103    for (it=art.begin();it!=art.end();it++)
104        g<<*it<<" ";
105    g<<"\n";
106 }

```

Listing 27.4.2: pamant1.cpp

```

1 // Marius Stroe
2 #include <fstream>
3 #include <iostream>
4 #include <set>
5 #include <stack>
6 #include <vector>
7 #include <algorithm>
8 #include <cassert>
9
10 using namespace std;
11
12 const char iname[] = "pamant.in";
13 const char oname[] = "pamant.out";
14
15 #define MAXN 100005
16 #define Min(a, b) ((a) < (b) ? (a) : (b))
17
18 vector <int> adj[MAXN], dfn, low, ans, con;
19
20 stack <pair <int, int> > stk;
21
22 void read_in(vector <int>* adj, int &n)
23 {
24     ifstream in(iname);
25     int cnt_edges, x, y;
26
27     assert(in >> n >> cnt_edges);
28     assert(1 <= n && n <= 100000);
29     assert(1 <= cnt_edges && cnt_edges <= 200000);
30
31     for (; cnt_edges > 0; -- cnt_edges)
32     {
33         assert(in >> x >> y);
34         assert(x != y && 1 <= x && x <= n && 1 <= y && y <= n);
35
36         adj[x].push_back(y);
37         adj[y].push_back(x);
38     }
39     in.close();
40 }
41
42 void cache(int node)
43 {
44     ans.push_back(node);
45 }
46
47 void DF(const int n, const int fn, int number)
48 {
49     vector <int>::iterator it,
50     int critical = false;
51     int sons = 0;
52
53     dfn[n] = low[n] = number;
54     for (it = adj[n].begin(); it != adj[n].end(); ++ it)
55     {
56         if (*it == fn) continue ;
57         if (dfn[*it] == -1)
58         {
59             stk.push( make_pair(n, *it) );
60             DF(*it, n, number + 1);
61             low[n] = Min(low[n], low[*it]);
62             critical |= (low[*it] >= dfn[n]);
63             sons++;
64         }
65         else
66             low[n] = Min(low[n], dfn[*it]);
67     }
68
69     if (n != fn)
70     {
71         if (critical)
72             ans.push_back(n);
73     }
74     else
75         if (sons > 1)

```

```

76         ans.push_back(n);
77     }
78
79     int main(void)
80     {
81         int n;
82         read_in(adj, n);
83
84         dfn.resize(n + 1), dfn.assign(n + 1, -1);
85         low.resize(n + 1);
86
87         for (int i = 1; i <= n; ++ i)
88             if (dfn[i] == -1)
89                 con.push_back(i), DF(i, i, 0);
90
91         ofstream out(oname);
92
93         sort(con.begin(), con.end());
94
95         out << con.size() << "\n";
96         for (int i = 0; i < (int) con.size(); ++ i)
97             out << con[i] << " ";
98
99         out << "\n";
100
101        sort(ans.begin(), ans.end());
102
103        out << ans.size() << "\n";
104
105        for (int i = 0; i < (int) ans.size(); ++ i)
106            out << ans[i] << " ";    out << "\n";
107
108        out.close();
109
110        return 0;
111    }

```

Listing 27.4.3: pamant2.cpp

```

1 #define TuTTy "Cosmin-Mihai Tutunaru"
2 #include<cstdio>
3 #include<vector>
4 #include<algorithm>
5
6 #define infile "pamant.in"
7 #define outfile "pamant.out"
8 #define nMax 100013
9
10 using namespace std;
11
12 vector <int> comp;
13 vector <int> crit;
14 vector <int> v[nMax];
15
16 bool vz[nMax];
17 int ord[nMax];
18 int curOrd;
19 int n, m;
20
21 void getComp(int x)
22 {
23     vz[x] = true;
24
25     for(unsigned i = 0; i < v[x].size(); ++i)
26         if(!vz[v[x][i]])
27             getComp(v[x][i]);
28 }
29
30 void getCrit(int x, int f)
31 {
32     ord[x] = ++curOrd;
33
34     for(unsigned i = 0; i < v[x].size(); ++i)
35         if(!ord[v[x][i]])
36         {

```

```

37     getCrit(v[x][i], x);
38     if(ord[v[x][i]] > ord[x] && v[x].size() > 1)
39         crit.push_back(x);
40     }
41
42     for(unsigned i = 0; i < v[x].size(); ++i)
43         if(v[x][i] != f)
44             ord[x] = min(ord[x], ord[v[x][i]]);
45     }
46
47 void read()
48 {
49     scanf("%d %d\n", &n, &m);
50     for(int i = 1; i <= m; ++i)
51     {
52         int x, y;
53         scanf("%d %d\n", &x, &y);
54         v[x].push_back(y);
55         v[y].push_back(x);
56     }
57 }
58
59 void solve()
60 {
61     for(int i = 1; i <= n; ++i)
62         if(!vz[i])
63         {
64             comp.push_back(i);
65             getComp(i);
66             getCrit(i, 0);
67         }
68     sort(crit.begin(), crit.end());
69
70     crit.resize(unique(crit.begin(), crit.end()) - crit.begin());
71 }
72
73 void write()
74 {
75     printf("%d\n", comp.size());
76     for(unsigned i = 0; i < comp.size(); ++i)
77         printf("%d ", comp[i]);
78     printf("\n");
79
80     printf("%d\n", crit.size());
81     for(unsigned i = 0; i < crit.size(); ++i)
82         printf("%d ", crit[i]);
83     printf("\n");
84 }
85
86 int main()
87 {
88     freopen(infile, "r", stdin);
89     freopen(outfile, "w", stdout);
90
91     read();
92     solve();
93     write();
94
95     fclose(stdin);
96     fclose(stdout);
97     return 0;
98 }

```

27.4.3 *Rezolvare detaliată

27.5 radare

Problema 5 - radare

100 de puncte

Odată cu vacanța de Paște, Alex s-a hotărât să plece cu mașina la mare. Harta României este

modelată ca un arbore (graf conex aciclic neorientat) ce are N noduri, reprezentând orașele țării. Este binecunoscut faptul că Alex este vitezoman, de aceea el vrea să se ferească de eventualele radare, care sunt plasate pe muchiile arborelui (pe o muchie se poate afla maxim un radar).

Dacă pe o muchie există un radar, atunci Alex nu va risca și nu va merge pe acea muchie.

Fiecare oraș are un timp de vizitare dat, iar cu toții știm că Alex este foarte ocupat, aşadar el nu vrea să piardă prea mult timp vizitând. Alex pleacă din Gheorgheni (nodul 1, considerat rădăcină) și dorește să știe în câte moduri ar putea fi plasate radarele pe hartă astfel încât timpul total de vizitare al orașelor accesibile din nodul 1 să fie egal cu P , știind că Alex nu va merge pe nicio muchie ce conține un radar.

Problema aceasta este destul de simplă pentru Alex, să că s-a gândit să v-o dea vouă.

Cerințe

Scrieți un program care să răspundă problemei lui Alex, adică să determine în câte moduri pot fi plasate radare pe muchii astfel încât timpul total de vizitare al orașelor accesibile din nodul 1 să fie egal cu P .

Date de intrare

Fișierul **radare.in** conține pe prima linie N și P , cu semnificația din enunț. Următoarele $N - 1$ linii vor conține fiecare câte două numere întregi x și y , însemnând că există o muchie între orașele x și y . Pe ultima linie se vor afla N numere naturale cuprinse între 1 și P inclusiv, al i -lea dintre acestea reprezentând timpul de vizitare al orașului i .

Date de ieșire

Fișierul **radare.out** va conține un singur număr reprezentând răspunsul problemei lui Alex modulo 31333.

Restricții și precizări

- $1 \leq N \leq 3\,000$
- $1 \leq P \leq 3\,500$
- $1 \leq x, y \leq N$
- $1 \leq$ timpul de vizitare al unui oraș P
- dacă un radar este plasat pe o muchie, Alex NU va merge pe acea muchie
- două configurații de radare diferă între ele dacă există cel puțin o muchie (a, b) care într-o una din configurații să conțină un radar, iar în cealaltă nu.
- pentru 30% din teste $N \leq 15$
- pentru 50% din teste, timpii de vizitare pentru fiecare oraș vor fi 1
- pentru 60% teste, $N \leq 300$

Exemple:

radare.in	radare.out	Explicații
6 3		Cele 5 moduri posibile sunt:
1 2		1. există radar pe muchia (2,4)
1 3		2. sunt radare pe muchiile (2,4), (4,5)
2 4		3. sunt radare pe muchiile (2,4), (4,6)
4 5		4. sunt radare pe muchiile (2, 4), (4,5), (4,6)
4 6		5. sunt radare pe muchiile (1,3), (4,5), (4,6)
1 1 1 1 1 1	5	

Timp maxim de executare/test: **0.4** secunde

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

27.5.1 Indicații de rezolvare

stud. Andrei Pârvu, Univeritatea POLITEHNICA București
stud. Bogdan Tătăroiu, Univeritatea POLITEHNICA București

Solutia 1 - 30 puncte - $O(2^N * N)$

Se vor genera toate submultimile de muchii, o muchie selectata reprezentand ca exista un radar pe acea muchie. Cu ajutorul unei parcurgeri (in adancime sau latime) se vor determina nodurile conectate de nodul 1, si se vor numara toate variantele care au suma P .

Solutia 2 - 50 puncte - $O(N * P^2)$

Vom construi o dinamica $A[i][j]$ - numarul de posibilitati de a selecta muchii din subarborele lui i , astfel incat din radacina subarborelui sa se poata ajunge in exact j noduri. Aceasta solutie se aplica pentru testelete in care timpii de vizitare a tutoror oraselor sunt 1.

Pentru a calcula valoarea dinamicei in *subarborele* i , se calculeaza mai intai valorile pentru fiecare din fiii acestuia.

Vom construi inca o dinamica $B[a][b]$ - numarul de posibilitati de a selecta muchii din subarboreii primilor a fii ai lui i astfel incat sa se poata ajunge in exact b noduri din radacina.

Modul de calculare a acestei dinamici este foarte asemanator cu *problema Rucsacului (Knapsack)*, avand 2 cazuri pentru fiecare fiu: fie taiem muchia dintre i si fiu (caz in care muchiile din subarborele fiului pot avea orice configuratie), fie variem numarul de noduri c din subarborele fiului si ne folosim de $A[fiu][c]$.

Solutia 3 - 60 puncte - $O(N * P^2)$

Pentru 60 de puncte, solutia se construieste in mod similar cu solutia de 50 de puncte. $A[i][j]$ - numarul de posibilitati de a selecta muchii din subarborele lui i , astfel incat din radacina subarborelui sa se poata ajunge in noduri avand suma totala a timpilor de vizitare exact j . Implementarea este asemeneatoare cu cea de 50 de puncte.

Solutia 4 - 100 puncte - $O(N * P)$

Solutia de 100 de puncte porneste de la urmatoarea observatie: daca pe o muchie (x, y) este un radar (x fiind mai apropiat de radacina decat y) atunci muchiile din subarborele lui y pot avea orice configuratie. Astfel ne vine ideea de a *liniariza arborele* si de a face urmatoarea dinamica:

$A[i][j]$ - numarul de moduri de a ajunge pe pozitia i a liniarizarii, suma nodurilor accesibile din 1 fiind j .

In acest moment avem doua variante: fie taiem muchia parinte a nodului cu timpul de intrare i , fie adaugam acest nod la configuratia curenta.

Daca muchia este taiata, putem sari intregul subarbore al nodului, ajungand pe pozitia *temp_sfarsit[nod]*. Astfel:

- $A[i + 1][j + \text{valoare}[\text{nod}_\text{temp}_i]] += A[i][j]$, daca nu taiem muchia
- $A[\text{temp}_\text{sfarsit}[\text{nod}_\text{temp}_i]][j] += A[i][j]$, daca taiem muchia

27.5.2 Cod sursă

Listing 27.5.1: radare-60.c

```

1 /* Tătăroiu Bogdan-Cristian - 60 de puncte */
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 #include <assert.h>
6
7 #define MAXN 3000
8 #define MAXP 4000
9 #define MOD 31333
10
11 int N, P, mx[MAXN - 1], my[MAXN - 1], deg[MAXN];
12 int pow2[MAXN], treeSize[MAXN], val[MAXN], sum[MAXN];
13 int *con[MAXN], **cnt, **aux;
14
15 void dfs(int k, int p)

```

```

16  {
17      int *it, leaf = 1;
18      treeSize[k] = 1;
19      sum[k] = val[k];
20      for (it = con[k]; *it != -1; it++)
21      {
22          if (*it == p)
23              continue;
24
25          dfs(*it, k);
26          leaf = 0;
27          treeSize[k] += treeSize[*it];
28          sum[k] += sum[*it];
29      }
30
31      if (leaf)
32      {
33          memset(cnt[k], 0, sizeof(int) * P);
34          cnt[k][val[k] - 1] = 1;
35          return;
36      }
37
38      memset(aux[0], 0, sizeof(int) * P);
39      aux[0][val[k] - 1] = 1;
40      int step = 0, i, j;
41      for (it = con[k]; *it != -1; it++)
42      {
43          if (*it == p)
44              continue;
45
46          for (i = 0; i < P; i++)
47              aux[1 ^ step][i] = (aux[step][i] * pow2[treeSize[*it] - 1]) % MOD;
48
49          for (i = 0; i < P; i++)
50          {
51              if (!aux[step][i])
52                  continue;
53
54              for (j = 0; i + j + 1 < P && j < sum[*it]; j++)
55              {
56                  if (!cnt[*it][j])
57                      continue;
58
59                  aux[1 ^ step][i + j + 1] = (aux[1 ^ step][i + j + 1] +
60                                              aux[step][i] * cnt[*it][j]) % MOD;
61              }
62          }
63
64          step ^= 1;
65      }
66
67      memcpy(cnt[k], aux[step], sizeof(int) * P);
68  }
69
70  int main()
71  {
72      assert(freopen("radare.in", "rt", stdin));
73      assert(freopen("radare.out", "wt", stdout));
74
75      assert(scanf("%d %d", &N, &P) == 2);
76      assert(1 <= N && N <= MAXN);
77      assert(1 <= P && P <= MAXP);
78      int i;
79
80      /* Calculeaza  $2^i \% \text{MOD}$  pentru i intre 0 si N - 1 */
81      pow2[0] = 1;
82      for (i = 1; i < N; i++)
83      {
84          pow2[i] = (pow2[i - 1] << 1);
85          if (pow2[i] >= MOD)
86              pow2[i] -= MOD;
87      }
88
89      /* Citeste muchiile */
90      for (i = 0; i + 1 < N; i++)
91      {

```

```

92         assert(scanf("%d %d", mx + i, my + i) == 2);
93         assert(1 <= mx[i] && mx[i] <= N);
94         assert(1 <= my[i] && my[i] <= N && mx[i] != my[i]);
95         mx[i] -= 1; my[i] -= 1;
96         deg[mx[i]] += 1;
97         deg[my[i]] += 1;
98     }
99
100    /* Aloca memorie pentru muchiile fiecarui nod */
101   for (i = 0; i < N; i++)
102   {
103       con[i] = malloc(sizeof(int) * (deg[i] + 1));
104       con[i][deg[i]] = -1;
105   }
106
107   for (i = 0; i + 1 < N; i++)
108   {
109       con[mx[i]][--deg[mx[i]]] = my[i];
110       con[my[i]][--deg[my[i]]] = mx[i];
111   }
112
113    /* Aloca memorie pentru cnt - dimensiune N x (P + 1) */
114   cnt = malloc(sizeof(int *) * N);
115   for (i = 0; i < N; i++)
116       cnt[i] = malloc(sizeof(int) * P);
117
118    /* Aloca memorie pentru aux - dimensiune 2 x (P + 1) */
119   aux = malloc(sizeof(int *) * 2);
120   for (i = 0; i < 2; i++)
121       aux[i] = malloc(sizeof(int) * P);
122
123    /* Citeste valorile */
124   for (i = 0; i < N; i++)
125   {
126       assert(scanf("%d", val + i) == 1);
127       assert(1 <= val[i] && val[i] <= P);
128   }
129
130   dfs(0, -1);
131
132   printf("%d\n", cnt[0][P - 1]);
133
134    /* Elibereaza memoria alocata */
135   for (i = 0; i < N; i++)
136   {
137       free(con[i]);
138       free(cnt[i]);
139   }
140
141   free(cnt);
142   free(aux[0]); free(aux[1]);
143   free(aux);
144
145   return 0;
146 }
```

Listing 27.5.2: radare_100.cpp

```

1 //Andrei Parvu
2 //O(N * P)
3 #include <cstdio>
4 #include <vector>
5
6 using namespace std;
7
8 const int lg = 3003, lg2 = 3503, mod = 31333;
9
10 int n, p, i, j, x, y, pt[lg], ind, prc[lg],
11      ult[lg], cnt[lg], D[lg][lg2], cur, val[lg];
12 bool fst[lg];
13 vector<int> v[lg];
14
15 void df(int nod)
16 {
17     fst[nod] = 1;
```

```

18     prc[++ ind] = nod;
19
20     cnt[nod] = 1;
21     for (int i = 0; i < (int)v[nod].size(); i++)
22         if (!fst[ v[nod][i] ])
23         {
24             df(v[nod][i]);
25             cnt[nod] += cnt[ v[nod][i] ];
26         }
27
28     ult[nod] = ind;
29 }
30
31 int main()
32 {
33     freopen("radare.in", "rt", stdin);
34     freopen("radare.out", "wt", stdout);
35
36     scanf("%d%d", &n, &p);
37     for (i = 1; i < n; i++)
38     {
39         scanf("%d%d", &x, &y);
40         v[x].push_back(y);
41         v[y].push_back(x);
42     }
43
44     for (i = 1; i <= n; i++)
45         scanf("%d", &val[i]);
46
47     df(1);
48
49     pt[0] = 1;
50     for (i = 1; i <= n; i++)
51     {
52         pt[i] = pt[i - 1] << 1;
53
54         if (pt[i] >= mod)
55             pt[i] -= mod;
56     }
57
58     D[1][0] = 1;
59     for (i = 1; i <= n; i++)
60         for (j = 0; j <= p; j++)
61             if (D[i][j] > 0)
62             {
63                 cur = prc[i];
64
65                 if (j + val[cur] <= p)
66                 {
67                     D[i + 1][j + val[cur]] += D[i][j];
68                     if (D[i + 1][j + val[cur]] >= mod)
69                         D[i + 1][j + val[cur]] -= mod;
70                 }
71
72                 D[ult[cur] + 1][j] += (D[i][j] * pt[cnt[cur] - 1]) % mod;
73
74                 if (D[ult[cur] + 1][j] >= mod)
75                     D[ult[cur] + 1][j] -= mod;
76             }
77
78     printf("%d\n", D[n + 1][p]);
79
80     return 0;
81 }
```

Listing 27.5.3: radare60.cpp

```

1 /* Perticas Catalin - 60 de puncte */
2 #include <cstdio>
3 #include <iostream>
4 #include <vector>
5 #include <string>
6 #include <cstring>
7 #include <ctime>
8
```

```

9  using namespace std;
10
11 #define NM 3005
12 #define MOD 31333
13 #define PB push_back
14
15 vector <int> G[NM], ARB[NM];
16 int viz[NM], N, P, W[NM][NM], PW[NM][NM], NRN[NM], p2[NM], cost[NM], RNRN[NM];
17
18 void formeazaARB(int nod)
19 {
20     viz[nod] = 1;
21
22     for (int i = 0; i < G[nod].size(); ++i)
23     {
24         int nnod = G[nod][i];
25         if (viz[nnod]) continue;
26         ARB[nod].PB(nnod);
27         formeazaARB(nnod);
28     }
29 }
30
31 void getNRN (int nod)
32 {
33     for (int i = 0; i < ARB[nod].size(); ++i)
34     {
35         int nnod = ARB[nod][i];
36         getNRN(nnod);
37         NRN[nod] += NRN[nnod];
38         RNRN[nod] += RNRN[nnod];
39     }
40
41     NRN[nod] += cost[nod];
42     ++RNRN[nod];
43 }
44
45 void bagaDIN(int nod)
46 {
47     if (ARB[nod].size() == 0)
48     {
49         W[nod][cost[nod]] = 1;
50         return ;
51     }
52
53     for (int i = 0; i < ARB[nod].size(); ++i)
54     {
55         int nnod = ARB[nod][i];
56         bagaDIN (nnod);
57     }
58
59 //memset (PW, 0, sizeof(PW));
60
61     int sum = cost[nod];
62     int n = ARB[nod].size();
63
64     for (int i = 0; i < n; ++i)
65     {
66         int nnod = ARB[nod][i];
67
68         int lim = min (P, sum + NRN[nnod]);
69
70         for (int j = cost[nod]; j <= lim; ++j) PW[i+1][j] = 0;
71
72         sum += NRN[nnod];
73     }
74
75     PW[0][cost[nod]] = 1;
76     sum = cost[nod];
77
78     for (int i = 0; i < n; ++i)
79     {
80         int nnod = ARB[nod][i];
81
82         if (sum > P) sum = P;
83
84         for (int j = cost[nod]; j <= sum; ++j)

```

```

85         {
86             PW[i+1][j] = (PW[i+1][j]+(PW[i][j]*p2[RNRN[nnod]-1])%MOD)%MOD;
87
88             int lim = min (NRN[nnod], P-j);
89
90             for (int k = cost[nnod]; k <= lim; ++k)
91                 PW[i+1][j+k] = (PW[i+1][j+k]+(PW[i][j]*W[nnod][k])%MOD)%MOD;
92         }
93
94         sum += NRN[nnod];
95     }
96
97     int lim = min(NRN[nod], P);
98
99     for (int i = cost[nod]; i <= lim; ++i) W[nod][i] = PW[n][i];
100 }
101
102 int main()
103 {
104     int start = clock();
105
106     freopen ("radare.in", "r", stdin);
107     freopen ("radare.out", "w", stdout);
108
109     scanf ("%d %d", &N, &P);
110
111     p2[0] = 1;
112
113     for (int i = 1; i <= N; ++i) p2[i] = (p2[i-1] * 2) % MOD;
114
115     for (int i = 1; i < N; ++i)
116     {
117         int a, b;
118         scanf ("%d %d", &a, &b);
119
120         G[a].PB(b);
121         G[b].PB(a);
122     }
123
124     for (int i = 1; i <= N; ++i) scanf ("%d", &cost[i]);
125
126     formeazaARB(1);
127     getNRN(1);
128     bagaDIN(1);
129
130     printf ("%d", W[1][P]);
131
132     return 0;
133 }
```

27.5.3 *Rezolvare detaliată

27.6 xmoto

Problema 6 - xmoto

100 de puncte

Ali Lalap se joacă Xmoto pe telefonul mobil. Scopul jocului este de a parcurge cu motocicleta circuitul din Gheorgheni. Traseul este format din N tronsoane. Consumul de benzină pentru tronsonul i ($1 \leq i \leq N$) este definit astfel:

$$a_i * v + k_i \text{ litri, dacă } v \leq v_i$$

$$b_i * v + q_i \text{ litri, dacă } v > v_i,$$

unde a_i, b_i, k_i, q_i, v_i sunt valori constante, iar v este viteza cu care se deplasează Ali Lalap pe acel tronson.

Pentru a nu forța motocicleta, lui Ali Lalap îi place să meargă cu viteza constantă și ar vrea să știe căte posibilități de a alege viteza cu care să parcurgă traseul există astfel încât să consume L litri de combustibil.

Cerințe

Calculați pentru câte valori distincte ale vitezei consumul total va fi de L litri.

Date de intrare

Pe prima linie din fișierul de intrare **xmoto.in** se află numerele naturale N și L .

Următoarele N linii conțin fiecare cele patru numere reale a_i , b_i , k_i , q_i urmate de un număr întreg v_i cu semnificațiile din enunț.

Date de ieșire

Fisierul de ieșire **xmoto.out** va conține pe prima linie un singur număr M , reprezentând numărul maxim de valori ale vitezei cu care parcurgând în întregime traseul se obține un consum total de L litri.

Pe următoarele M linii se vor afișa M numere reale, distincte, cu 6 zecimale și sortate crescator w_1, w_2, \dots, w_M , astfel încât dacă se parurge traseul cu viteza w_i ($1 \leq i \leq M$) să se obțină un consum total de L litri.

Restricții și precizări

- $N \leq 50\ 000$
- numerele reale a_i, b_i aparțin intervalului $[-100, 100]$
- $-1\ 000\ 000 \leq k_i, q_i \leq 1\ 000\ 000$
- $L \leq 100\ 000\ 000$
- Pe fiecare tronson consumul va fi strict pozitiv pentru orice viteză din intervalul $(0, 10\ 000]$
- Viteza maximă a motocicletei este de $10\ 000$ km/h
- Motocicleta rulează de la început până la sfârșit cu aceeași viteză (nu se pierde timp cu plecarea de pe loc, nu există accelerări/frânări)
 - Toate vitezele (viteză maximă, vitezele vi, viteza care trebuie determinată) sunt exprimate în aceeași unitate de măsură
 - Se consideră corectă orice soluție în care vitezele diferă cu cel mult 10^{-6} față de rezultatul corect.
 - Verificarea consumului total se va face cu o precizie de 10^{-6} față de L .
 - Se garantează că M este finit.

Punctaje

- Pentru determinarea corecta a lui M fără a calcula corect si cele M viteze se vor obtine 50% din punctajul pe test
- Pentru 10% din teste $N = 1$
- Pentru 25% din teste $a_i, b_i > 0$
- Pentru 50% din teste $N \leq 1\ 000$

Exemple:

xmoto.in	xmoto.out	Explicații
2 150 3.0 -2.0 2.0 22000.0 60 2.0 4.0 4.0 2.0 50	1 28.800000	28.8 ≤ 60 deci consumul pe primul tronson este $x=3*28.8+2=88.4$ 28.8 ≤ 50 deci consumul pe al doilea tronson este $y=2*28.8+4=61.6$ Consumul total: $x+y=88.4+61.6=150$

Timp maxim de executare/test: **0.4** secunde

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

27.6.1 Indicații de rezolvare

Duta Vlad si Savin Tiberiu

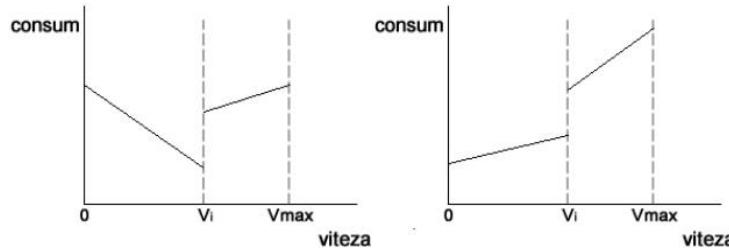
Pentru fiecare tronson i , funcția consum definită

$$f(v) = a_i * v + k_i, \quad v \leq v_i$$

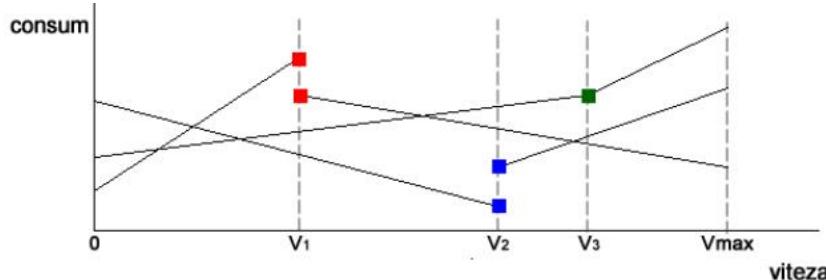
$$f(v) = b_i * v + q_i, \quad v > v_i$$

este formată din două *funcții liniare și monotone*.

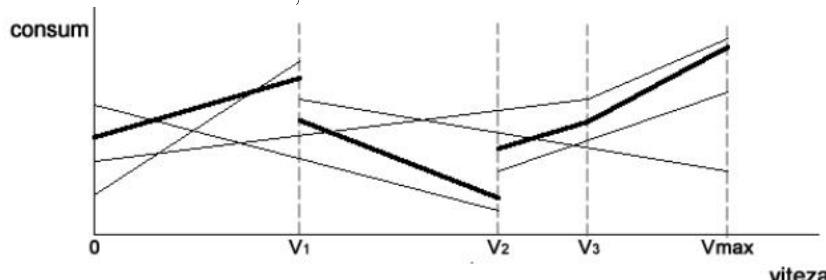
Cateva exemple de grafice ale acestor functii:



Daca sortam toate functiile fi dupa v_i si suprapunem graficele obtinem o figura asemanatoare cu



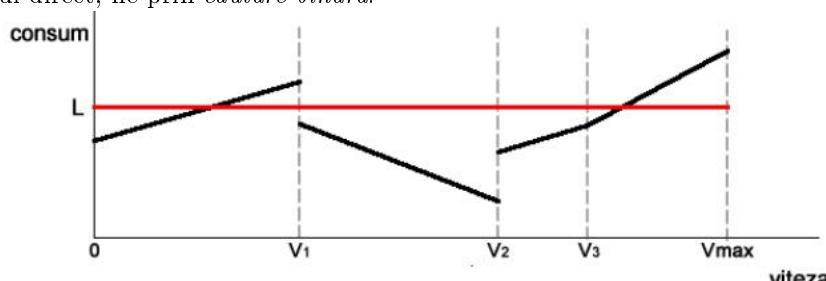
Se observa ca pe fiecare interval $(v_i, v_{i+1}]$ consumul total $C(v) = \sum f_j(v)$ este o suma de functii liniare si monotone, deci este tot o functie liniara si monotonă



De altfel, pe fiecare interval $(v_i, v_{i+1}]$, consumul total $C(v) = \sum x_i * v + \sum y_i$, unde $x_i = a_i$, $v \leq v_i$, $x_i = b_i$ altfel

$y_i = k_i$, $v \leq v_i$, $y_i = q_i$ altfel

Daca $\min(C(v_i), C(v_{i+1})) \leq L \leq \max(C(v_i), C(v_{i+1}))$ atunci cu siguranta există o valoare v în intervalul $(v_i, v_{i+1}]$ astfel încât $C(v) = L$. Valoarea efectivă se poate calcula fie prin calcul direct, fie prin *cautare binară*.



Orice solutii corecte de complexitate $O(N \log N)$ sau $O(N \log V)$ obtin 100 puncte
Solutii de complexitate $O(N^2)$, $O(N * V)$ obtin 50-60 puncte

O solutie care trateaza doar cazul in care functia este monotonă pe intreg intervalul $[0, V_{max}]$ si calculeaza rezultatul in $O(N \log V)$ obtine 25 puncte.

Există multe alte abordări ale problemei care în funcție corectitudine și diverse optimizări obtin până la 50 de puncte.

```

sortare(); //sorteaza in functie de vi
for (i=1; i<=N; ++i)
{
    v = (L - Sk) / Sa; // Sa * v + Sk = L
    Sa = Sa - A[i] + B[i]; //se schimba parametrii
    Sk = Sk - K[i] + Q[i];
    if (v > Vi[i-1] && v <= Vi[i]) solutie(v); //solutie corecta
}

```

27.6.2 Cod sursă

https://www.infoarena.ro/job_detail/2581038?action=view-source

Listing 27.6.1: xmoto.cpp

```

1 // https://www.infoarena.ro/job_detail/2581038?action=view-source
2 // Ciocoiu Vlad vlad082002
3 #include <iostream>
4 #include <vector>
5 #include <iomanip>
6 #include <algorithm>
7 #include<iomanip>
8
9 using namespace std;
10
11 ifstream fin("xmoto.in");
12 ofstream fout("xmoto.out");
13
14 int n, l;
15
16 vector<pair<double, pair<pair<double, double>, pair<double, double>>> v;
17 vector<double> intervale;
18 pair<double, double> consum[50005]; //consum[i] e consumul pe intervalul
19 //(( intervale[i-1], intervale[i] ] )
20 vector<double> ans;
21 pair<double, double> aib[50005];
22
23 void update(int poz, pair<double, double> x)
24 {
25     while(poz <= intervale.size())
26     {
27         aib[poz].first += x.first;
28         aib[poz].second += x.second;
29         poz += (poz& - poz);
30     }
31 }
32
33 pair<double, double> query(int poz)
34 {
35     pair<double, double> s = {0.0, 0.0};
36     while(poz > 0)
37     {
38         s.first += aib[poz].first;
39         s.second += aib[poz].second;
40         poz -= (poz& - poz);
41     }
42
43     return s;
44 }
45
46 int cb(double x)
47 {
48     int st = 0, dr = intervale.size() - 1;
49
50     while(st <= dr)
51     {
52         int m = (st+dr)/2;
53         if(intervale[m] == x)
54             return m;
55         else if(intervale[m] <= x)
56             st = m+1;
57         else
58             dr = m-1;
59     }
60
61     return -1;
62 }
63
64 void citire()
65 {
66     fin >> n >> l;
67     double a, b, c, d, e;
68     intervale.push_back(0.0);
69 }
```

```

70     for(int i = 1; i <= n; i++)
71     {
72         fin >> a >> b >> c >> d >> e;
73         v.push_back({e, {{a, c}, {b, d}}});
74         intervale.push_back(e);
75     }
76
77     intervale.push_back(10000.0);
78 }
79
80 void solve()
81 {
82     sort(intervale.begin(), intervale.end());
83
84     for(auto it: v)
85     {
86         double q = it.first;
87         int poz = cb(q);
88         update(poz, it.second.first);
89         update(intervale.size()-1, it.second.second);
90         update(poz, {-it.second.second.first, -it.second.second.second});
91     }
92
93     pair<double, double> q = query(intervale.size()-1);
94
95     for(int i = 1; i < intervale.size(); i++)
96     {
97         double maxim, minim;
98         double a = double(intervale[i-1]);
99         double b = double(intervale[i]);
100
101        pair<double, double> p = query(i-1);
102
103        p.first = q.first-p.first;
104        p.second = q.second-p.second;
105
106        minim = p.first*a+p.second;
107        maxim = p.first*b+p.second;
108
109        if(minim > maxim)
110            swap(minim, maxim);
111
112        if(minim <= l && l <= maxim)
113            ans.push_back((l-p.second)/p.first);
114    }
115
116    fout << ans.size() << '\n';
117    for(auto x: ans)
118        fout << fixed << setprecision(6) << x << '\n';
119 }
120
121 int main()
122 {
123     citire();
124     solve();
125     return 0;
126 }
```

27.6.3 *Rezolvare detaliată

Capitolul 28

ONI 2010

28.1 conex

Problema 1 - conex

100 de puncte

Numim *graf orientat* o pereche de mulțimi, notată $G = (V, A)$, unde V este o mulțime de elemente denumite noduri, iar A este o colecție de *perechi ordonate* de elemente din V denumite arce.

Numim *drum* într-un graf o succesiune de noduri x_1, x_2, \dots, x_p cu proprietatea că oricare ar fi $1 \leq i < p$, există în graf arcul (x_i, x_{i+1}) .

Un graf orientat se numește *tare conex* dacă oricare ar fi x și y noduri din graf, există în graf cel puțin un drum de la x la y și cel puțin un drum de la y la x .

Cerințe

Scripteți un program care citește succesiv arcele unui graf orientat cu N noduri și verifică după fiecare arc citit dacă *graful parțial* curent (graful format din cele N vârfuri și arcele citite până la momentul respectiv) este sau nu tare conex.

Interacțiune

Programul vostru nu va citi și nu va scrie din/în niciun fișier.

În schimb, va interacționa cu un program al comisiei care rulează în paralel.

Interacțiunea se desfășoară conform protocolului descris în continuare.

Programul vostru citește o linie de la intrarea standard, care conține două numere naturale $N M$ separate printr-un spațiu (unde N reprezintă numărul de noduri din graf, iar M reprezintă numărul de arce din graf).

La momentul inițial considerați că graful parțial curent are N vârfuri și 0 arce.

În continuare programul vostru va efectua un anumit număr de runde, în modul următor:

- afișează la ieșirea standard o linie care conține valoarea 0 dacă graful parțial curent nu este tare conex, respectiv valoarea 1, dacă graful parțial curent este tare conex;
- dacă valoarea afișată a fost 0, atunci programul vostru va citi de la intrarea standard o linie pe care se află două numere naturale $x y$, separate prin spațiu, cu semnificația că la graful parțial curent se adaugă arcul de la x la y ;
- dacă valoarea afișată a fost 1, execuția programului vostru trebuie să se termine; se garantează ca graful dat este tare conex.

Instructiuni de programare

Programatorii în C trebuie să apeleze "fflush(stdout);;" după ce au terminat de scris o linie completă la ieșire. Spre exemplu:

```
printf("0\n"); fflush(stdout); | printf("1\n"); fflush(stdout);
```

Programatorii în C++ trebuie să apeleze "cout.flush();;" după ce au terminat de scris o linie completă la ieșire. Spre exemplu:

```
cout << "0\n"; cout.flush(); | cout << "1\n"; cout.flush();
```

Programatorii în Pascal trebuie să apeleze "flush(output);" după ce au terminat de scris o linie completă la ieșire. Spre exemplu:

```
WriteLn('0'); flush(Output); | WriteLn('1'); flush(Output);
```

Recomandăm ca programatorii C/C++ să citească datele cu funcția scanf(), fără a specifica marcajul de sfârșit de linie '\n' în sirul care descrie formatul de citire.

De exemplu, citirea valorilor variabilelor N și M se va realiza astfel:

```
scanf ("%d %d", &N, &M);
```

Exemplu de interacțiune

Citește N și M	4 8
Graful parțial nu este tare conex	Scrie o linie cu valoarea 0
Citește următorul arc	1 2
Graful parțial nu este tare conex	Scrie o linie cu valoarea 0
Citește următorul arc	3 1
Graful parțial nu este tare conex	Scrie o linie cu valoarea 0
Citește următorul arc	2 4
Graful parțial nu este tare conex	Scrie o linie cu valoarea 0
Citește următorul arc	2 3
Graful parțial nu este tare conex	Scrie o linie cu valoarea 0
Citește următorul arc	4 3
Graful parțial este tare conex	Scrie o linie cu valoarea 1

Execuția programului s-a terminat.

Restricții și precizări

- $1 \leq N \leq 10\,000$
- $1 \leq M \leq 40\,000$
- Se garantează ca graful dat este tare conex.
- Arcele nu sunt neapărat distințe.
- Timpul de execuție al programului de interacțiune va fi maxim 0.1 secunde.

Punctaj:

Punctajul pentru un test va fi zero dacă apare una dintre următoarele situații:

- programul vostru depășește timpul maxim de execuție/test;
- programul vostru nu respectă protocolul de comunicare;
- programul vostru nu scrie răspunsul corect.

Timp maxim de executare/test: : 0.3 secunde, din care 0.1 secunde sunt alocate pentru programul de interacțiune.

Memorie: total **16 MB**

28.1.1 Indicații de rezolvare

Puni Andrei Paul

Vom reține mulțimea nodurilor spre care 1 are drum și mulțimea nodurilor care au drum spre 1, pe care le vom nota Q și W .

Când se primește un arc x, y se adaugă *listelor de adiacență* iar în caz că x face parte din Q și y nu face parte din Q se face o *parcursere în adâncime* folosind muchile care nu au mai fost folosite pana in acel moment; analog y face parte din W si x nu face parte din W .

Cand cardinalul ambelor multimi devine N graful devine conex. Complexitatea totală a algoritmului este $(N + M)$.

28.1.2 Cod sursă

Listing 28.1.1: conex4.cpp

```

1 //programul asta presupune ca ultimul arc face graful conex
2 #include <stdio.h>
3 #define Nmax 100100
4
5 int n,m;
6
7 int main()
8 {
9     scanf("%d %d",&n, &m);
10    int a,b, i;
11
12    for (i=1;i<=m;++i)
13    {
14        printf("0\n");
15        fflush(stdout);
16        scanf("%d %d", &a, &b);
17    }
18
19    printf("1\n");
20    fflush(stdout);
21
22    return 0;
23 }
```

Listing 28.1.2: conex_10p.cpp

```

1 //f.brut marius
2 #include<stdio.h>
3 #include<vector>
4 #define pb push_back
5
6 using namespace std;
7
8 const int maxn = 10001;
9 vector<int> VECT[maxn];
10 int VER[maxn],CHECK;
11 int NR,N,M;
12
13 void dfs(int nod)
14 {
15     if (VER[nod]) return ;
16     VER[nod] = 1;
17     ++NR;
18     for(int i = 0;i < (int)VECT[nod].size(); ++i)
19         dfs(VECT[nod][i]);
20 }
21
22 int main()
23 {
24     scanf("%d %d",&N,&M);
25     for(int i = 1;i <= M; ++i)
26     {
27         printf("0\n");
28         fflush(stdout);
29         int x,y;
30         scanf("%d %d", &x, &y);
31         VECT[x].pb(y);
32         CHECK = 1;
33         for(int j = 1;j <= N; ++j)
34         {
35             NR = 0;
36             for(int k = 1;k <= N; ++k) VER[k] = 0;
37             dfs(j);
38             if (NR != N) CHECK = 0;
39         }
40         if (CHECK)
41         {
42             printf("1\n");
43             fflush(stdout);
44         }
45     }
46 }
```

```

44         return 0;
45     }
46 }
47 return 0;
48 }
```

Listing 28.1.3: conex_50p_3.cpp

```

1 //by Puni Andrei-Paul
2 //Time complexity: O((n + m)*m)
3 //Space complexity: O(n + m)
4 //Method: Kosaraju's_algorithm / depth first search
5 //Working time: 15 minutes
6 //brut optimizat
7 #include <iostream>
8 #include <vector>
9 #include<string.h>
10
11 using namespace std;
12
13 #define Nmax 100100
14
15 int n,m;
16
17 vector <int> l[Nmax], lt[Nmax];
18 int post[Nmax], nrEl;
19 char viz[Nmax];
20
21 void df(int nod)
22 {
23     viz[nod] = 1;
24     for (int i = 0; i < (int)l[nod].size(); ++i)
25         if (viz[l[nod][i]] == 0)
26             df(l[nod][i]);
27     post[+post[0]] = nod;
28 }
29
30 void dft(int nod)
31 {
32     ++nrEl;
33     viz[nod] = 0;
34     for (int i = 0; i < (int)lt[nod].size(); ++i)
35         if (viz[lt[nod][i]] == 1)
36             dft(lt[nod][i]);
37 }
38
39 int main()
40 {
41     cin >> n >> m;
42
43     int a,b;
44
45     for (int i=1;i<=m;++i)
46     {
47         cout << "0\n";
48         cout.flush();
49
50         cin >> a >> b;
51
52         l[a].push_back(b);
53         lt[b].push_back(a);
54
55         if (i < n) continue;
56
57         nrEl = 0;
58
59         memset(viz, 0, sizeof viz);
60
61         post[0] = 0;
62
63         for (int j=1;j<=n;++j)
64             if (viz[j] == 0)
65                 df(j);
66
67         dft(post[post[0]]);
68 }
```

```

68         if (nrEl == n)
69             break;
70     }
71
72     cout << "1\n";
73     cout.flush();
74
75     return 0;
76 }

```

Listing 28.1.4: conex_50p.cpp

```

1 //by Puni Andrei-Paul
2 //Time complexity: O((n + m)*m)
3 //Space complexity: O(n + m)
4 //Method: Kosaraju's_algorithm / depth first search
5 //Working time: 15 minutes
6 //brut opt c
7
8 #include <iostream>
9 #include <cstdio>
10 #include <vector>
11 #include<string.h>
12
13 using namespace std;
14
15 #define Nmax 10100
16
17 int n,m;
18
19 vector <int> l[Nmax], lt[Nmax];
20 int post[Nmax], nrEl;
21 char viz[Nmax];
22
23 void df(int nod)
24 {
25     viz[nod] = 1;
26     for (int i = 0; i < (int)l[nod].size(); ++i)
27         if (viz[l[nod][i]] == 0)
28             df(l[nod][i]);
29     post[post[0]] = nod;
30 }
31
32 void dft(int nod)
33 {
34     ++nrEl;
35     viz[nod] = 0;
36     for (int i = 0; i < (int)lt[nod].size(); ++i)
37         if (viz[lt[nod][i]] == 1)
38             dft(lt[nod][i]);
39 }
40
41 int main()
42 {
43     //cin >> n >> m;
44     scanf("%d %d", &n, &m);
45
46     int a,b;
47
48     for (int i=1;i<=m;++i)
49     {
50         //cout << "nu este\n";
51         //cout.flush();
52         printf("0\n");
53         fflush(stdout);
54
55         //cin >> a >> b;
56         scanf("%d%d", &a, &b);
57
58         l[a].push_back(b);
59         lt[b].push_back(a);
60
61         if (i < n) continue;

```

```

63         nrEl = 0;
64
65         memset(viz, 0, sizeof viz);
66
67         post[0] = 0;
68
69         for (int j=1; j<=n; ++j)
70             if (viz[j] == 0)
71                 df(j);
72
73         dft(post[post[0]]);
74
75         if (nrEl == n)
76             break;
77     }
78
79 //cout << "este\n";
80 //cout.flush();
81
82 printf("1\n");
83 fflush(stdout);
84
85 return 0;
86
87 }
```

Listing 28.1.5: conex_50p2.cpp

```

1 //by Puni Andrei-Paul
2 //Time complexity: O((n + m)*m)
3 //Space complexity: O(n + m)
4 //Method: Kosaraju's_algorithm / depth first search
5 //Working time: 15 minutes
6 //brut c
7 #include <iostream>
8 #include <vector>
9 #include <cstdio>
10 #include<string.h>
11
12 using namespace std;
13
14 #define Nmax 10100
15
16 int n,m;
17
18 vector <int> l[Nmax], lt[Nmax];
19 int post[Nmax], nrEl;
20 char viz[Nmax];
21
22 void df(int nod)
23 {
24     viz[nod] = 1;
25     for (int i = 0; i < (int)l[nod].size(); ++i)
26         if (viz[l[nod][i]] == 0)
27             df(l[nod][i]);
28     post[+post[0]] = nod;
29 }
30
31 void dft(int nod)
32 {
33     ++nrEl;
34     viz[nod] = 0;
35     for (int i = 0; i < (int)lt[nod].size(); ++i)
36         if (viz[lt[nod][i]] == 1)
37             dft(lt[nod][i]);
38 }
39
40 int main()
41 {
42     //cin >> n >> m;
43     scanf("%d%d", &n, &m);
44
45     int a,b;
46
47     for (int i=1;i<=m;++i)
```

```

48      {
49          //cout << "nu este\n";
50          //cout.flush();
51          printf("0\n");
52          fflush(stdout);
53
54          //cin >> a >> b;
55          scanf("%d%d", &a, &b);
56
57          l[a].push_back(b);
58          lt[b].push_back(a);
59
60          nrEl = 0;
61
62          memset(viz, 0, sizeof viz);
63
64          post[0] = 0;
65
66          for (int j=1;j<=n;++j)
67              if (viz[j] == 0)
68                  df(j);
69
70          dft(post[post[0]]);
71
72          if (nrEl == n)
73              break;
74      }
75
76      //cout << "este\n";
77      //cout.flush();
78
79      printf("1\n");
80      fflush(stdout);
81
82
83      return 0;
84  }

```

Listing 28.1.6: conex_100p.cpp

```

1 //Marius Dragus
2 // Solutie O(N + M)
3 #include<stdio.h>
4 #include<vector>
5 #define pb push_back
6
7 using namespace std;
8 const int maxn = 100000;
9
10 vector<int> VECT[maxn];
11 vector<int> VECTINV[maxn];
12
13 int NR,NR1,VER[maxn],VER1[maxn];
14 int N,M;
15
16 void dfs(int nod)
17 {
18     if (VER[nod]) return;
19     ++NR;
20     VER[nod] = 1;
21     for(int i = 0;i < VECT[nod].size(); ++i)
22         dfs(VECT[nod][i]);
23 }
24
25 void dfsinv(int nod)
26 {
27     if (VER1[nod]) return;
28     ++NR1;
29     VER1[nod] = 1;
30     for(int i = 0;i < VECTINV[nod].size(); ++i)
31         dfsinv(VECTINV[nod][i]);
32 }
33
34 int main()
35 {

```

```

36     scanf("%d %d", &N, &M);
37     NR = NR1 = VER[1] = VER1[1] = 1;
38     for(int i = 1; i <= M; ++i)
39     {
40         printf("0\n");
41         fflush(stdout);
42         int x,y;
43         scanf("%d %d", &x, &y);
44         VECT[x].pb(y);
45         VECTINV[y].pb(x);
46         if (VER[x] == 1)
47         {
48             VER[x] = 0;
49             NR--;
50             dfs(x);
51         }
52         if (VER1[y] == 1)
53         {
54             VER1[y] = 0;
55             --NR1;
56             dfsinv(y);
57         }
58         if (NR == N && NR1 == N)
59         {
60             printf("1\n");
61             fflush(stdout);
62             return 0;
63         }
64     }
65     return 0;
66 }
```

Listing 28.1.7: conex_100p2.cpp

```

1 //by Puni Andrei-Paul
2 //Time complexity: O(n + m)
3 //Space complexity: O(n + m)
4 //Method: depth first search
5 //Working time: 10 minutes
6 #include <iostream>
7 #include <vector>
8 #include <queue>
9 #include <stdio.h>
10
11 using namespace std;
12
13 #define Nmax 10100
14
15 int n,m, nr = 1, nrt = 1;
16
17 //o sa tin muchile in coada ..
18 queue<int> l[Nmax], lt[Nmax];
19
20 bool ajunge[Nmax], ajunget[Nmax];
21
22 void df(int nod)
23 {
24     if (!ajunge[nod])
25     {
26         ++nr;
27         ajunge[nod] = true;
28     }
29
30     while (!l[nod].empty())
31     {
32         if (!ajunge[l[nod].front()])
33             df(l[nod].front());
34         l[nod].pop();
35     }
36 }
37
38 void dft(int nod)
39 {
40     if (!ajunget[nod])
41     {
```

```

42         ++nrt;
43         ajunget[nod] = true;
44     }
45
46     while (!lt[nod].empty())
47     {
48         if (!ajunget[lt[nod].front()])
49             dft(lt[nod].front());
50         lt[nod].pop();
51     }
52 }
53
54 int main()
55 {
56     //cin >> n >> m;
57     scanf("%d%d", &n, &m);
58
59     int a,b;
60
61     ajunge[1] = true;
62     ajunget[1] = true;
63
64     while (nr < n || nrt < n)
65     {
66         //cout << "nu este\n";
67         //cout.flush();
68         printf("0\n");
69         fflush(stdout);
70
71         //cin >> a >> b;
72         scanf("%d%d", &a, &b);
73
74         l[a].push(b);
75         lt[b].push(a);
76
77         if (ajunge[a] && !ajunge[b])
78             df(a);
79
80         if (ajunget[b] && !ajunget[a])
81             dft(b);
82     }
83
84     //cout << "este\n";
85     //cout.flush();
86
87     printf("1\n");
88     fflush(stdout);
89
90     return 0;
91 }
```

Listing 28.1.8: conex3.cpp

```

1 //by Puni Andrei-Paul
2 //Time complexity: O((n + m)*m)
3 //Space complexity: O(n + m)
4 //Method: Kosaraju's_algorithm / depth first search
5 //Working time: 15 minutes
6 //brut
7 #include <iostream>
8 #include <vector>
9 #include<string.h>
10
11 using namespace std;
12
13 #define Nmax 100100
14
15 int n,m;
16
17 vector <int> l[Nmax], lt[Nmax];
18 int post[Nmax], nrEl;
19 char viz[Nmax];
20
21 void df(int nod)
22 {
```

```

23     viz[nod] = 1;
24     for (int i = 0; i < (int)l[nod].size(); ++i)
25         if (viz[l[nod][i]] == 0)
26             df(l[nod][i]);
27     post[post[0]] = nod;
28 }
29
30 void dft(int nod)
31 {
32     ++nrEl;
33     viz[nod] = 0;
34     for (int i = 0; i < (int)lt[nod].size(); ++i)
35         if (viz[lt[nod][i]] == 1)
36             dft(lt[nod][i]);
37 }
38
39 int main()
40 {
41     cin >> n >> m;
42
43     int a,b;
44
45     for (int i=1;i<=m;++i)
46     {
47         cout << "0\n";
48         cout.flush();
49
50         cin >> a >> b;
51         continue;
52         l[a].push_back(b);
53         lt[b].push_back(a);
54
55         nrEl = 0;
56
57         memset(viz, 0, sizeof viz);
58
59         post[0] = 0;
60
61         for (int j=1;j<=n;++j)
62             if (viz[j] == 0)
63                 df(j);
64
65         dft(post[post[0]]);
66
67         if (nrEl == n)
68             break;
69     }
70
71     cout << "1\n";
72     cout.flush();
73
74     return 0;
75 }
```

28.1.3 *Rezolvare detaliată

28.2 kmax

Problema 2 - kmax

100 de puncte

Aurorei îi plac mult permutările. Ea definește o kmax-permutare ca fiind o permutare cu următoarea proprietate: pentru orice subsecvență cu elementele în ordine crescătoare, lungimea subsecvenței este cel mult egală cu k . Acum, Aurora se întreabă câte kmax-permutări cu N elemente există.

Cerințe

Pentru valorile N , k și R date, aflați numărul de kmax-permutări cu N elemente. Rezultatul va fi calculat modulo R .

Date de intrare

Pe prima linie a fișierului de intrare **kmax.in** se vor afla numerele naturale N , k și R , având semnificațiile din enunț, separate între ele prin câte un spațiu.

Date de ieșire

Fișierul de ieșire **kmax.out** va conține o singură linie pe care veți scrie numărul de k -max-permutări cu N elemente, modulo R .

Restricții și precizări

- $1 \leq k \leq N \leq 300$
- $10 \leq R \leq 30\,000$
- O subsecvență a unei permutări este formată din elemente situate pe poziții consecutive.
- Pentru 20% din teste $N \leq 10$
- Pentru 60% din teste $N \leq 150$

Exemplu:

kmax.in	kmax.out	Explicații
4 2 997	17	Dintre cele 24 de permutări de 4 elemente NU sunt 2max-permutări următoarele 7: 1 2 3 4 1 2 4 3 1 3 4 2 2 1 3 4 2 3 4 1 3 1 2 4 4 1 2 3 După cum se observă, toate cele șapte permutări de mai sus au cel puțin o secvență cu elementele în ordine crescătoare de lungime mai mare decât 2.
30 10 27017	21690	

Timp maxim de executare/test: **0.4** secunde

Memorie: total **16 MB**

28.2.1 Indicații de rezolvare

Gheorghe Cosmin

Vom rezolva problema cu programare dinamică:

Fie $\text{din}[i][j] =$ numărul de permutări de lungime i pentru care ultima secvență crescătoare (cea din capatul dreapta) are lungime j .

Recurența se construiește în modul următor.

Dacă vrem să calculăm $\text{din}[i][j]$, vom fixa cel mai mare element pe pozitia q .

Vom împărți permutarea astfel în două permutări mai mici independente, cea din stânga de lungime $q - 1$ și cea din dreapta de lungime $i - q - 1$.

Se poate observa că permutarea stânga se poate termina în o secvență crescătoare de lungime $lu \leq K - 1$ iar cea dreapta într-o secvență crescătoare de lungime fix j (vrem să calculăm $\text{din}[i][j]$, adică ultima secvență crescătoare să aibă lungime j).

Astfel recurența este următoarea:

$$\text{din}[i][j] = \text{din}[q - 1][lu] * \text{din}[i - q - 1][j] * \text{comb}[i - 1][q - 1]$$

Se observă că împărțim o permutare mai mare în două permutări mai mici și folosim combinări pentru a lua toate posibilitățile de a poziționa anumite elemente în prima permutare și în a doua.

Trebuie avut grijă la câteva cazuri particulare în recurența de mai sus și anume trebuie definită bine dinamica pentru permutări de lungime 0 (adică $\text{din}[0][j]$) și cazul când punem elementul maxim pe ultima poziție din permutare.

28.2.2 Cod sursă

Listing 28.2.1: kmax_20.cpp

```

1 //Gheorghe Cosmin
2 //Solutie O(N! * N)
3 #include <stdio.h>
4 #include <algorithm>
5
6 using namespace std;
7
8 int N, K, MOD;
9
10 int get_brut(int N, int K)
11 {
12     int a[N + 10], rez = 0, i;
13
14     a[0] = 0;
15     for (i = 1; i <= N; i++) a[i] = i;
16
17     do
18     {
19         int k = 0;
20         for (i = 1; i <= N; i++)
21         {
22             if (a[i] > a[i - 1]) k++;
23             else k = 1;
24             if (k > K) break;
25         }
26
27         rez += i == N + 1;
28     } while (next_permutation(a + 1, a + N + 1));
29
30     return rez;
31 }
32
33 int main()
34 {
35     int i, j, k, MOD;
36
37     freopen("kmax.in", "r", stdin);
38     freopen("kmax.out", "w", stdout);
39
40     scanf("%d %d %d", &N, &K, &MOD);
41     printf("%d\n", get_brut(N, K) % MOD);
42
43     return 0;
44 }
```

Listing 28.2.2: kmax_100.cpp

```

1 //Gheorghe Cosmin
2 //Complexitate O(N^2 * K)
3 #include <stdio.h>
4 #include <algorithm>
5
6 using namespace std;
7
8 #define NMAX 310
9
10 int N, K, MOD;
11
12 int comb[NMAX][NMAX];
13 int din[NMAX][NMAX];
14
15 int get_brut(int N, int K)
16 {
17     int a[N + 10], rez = 0, i;
18
19     a[0] = 0;
20     for (i = 1; i <= N; i++) a[i] = i;
21
22     do
```

```

23     {
24         int k = 0;
25         for (i = 1; i <= N; i++)
26         {
27             if (a[i] > a[i - 1]) k++;
28             else k = 1;
29             if (k > K) break;
30         }
31         rez += i == N + 1;
32     } while (next_permutation(a + 1, a + N + 1));
33
34     return rez;
35 }
36
37 int main()
38 {
39     int i, j, k;
40
41     freopen("kmax.in", "r", stdin);
42     freopen("kmax.out", "w", stdout);
43
44     scanf("%d %d %d", &N, &K, &MOD);
45
46     comb[0][0] = 1;
47     for (i = 1; i <= N; i++)
48     {
49         comb[i][0] = 1;
50         for (j = 1; j <= i; j++)
51         {
52             comb[i][j] = comb[i-1][j-1] + comb[i-1][j];
53             if (comb[i][j] >= MOD) comb[i][j] -= MOD;
54         }
55     }
56
57     int jeg = 0;
58
59     for (i = 1; i <= N; i++)
60     {
61         // setez potizia maximului
62
63         for (k = 1; k <= K; k++)
64         {
65             din[i][k] = din[i][k - 1];
66             if (k > i) continue;
67             for (j = 1; j <= i; j++)
68             {
69
70                 // for (int kk = 1; kk <= k; kk++) jeg += kk;
71
72                 int dr = (din[i - j][k] - din[i - j][k - 1]);
73                 if (dr < 0) dr += MOD;
74                 if (j == i) dr = 1;
75
76                 int st = din[j - 1][K - 1];
77                 if (j == i)
78                 {
79                     st = din[j - 1][k - 1] - din[j - 1][max(0, k - 2)];
80                     if (st < 0) st += MOD;
81                 }
82                 if (j == 1) st = 1;
83
84                 din[i][k] += (((st * dr) % MOD) * comb[i - 1][j - 1]) % MOD;
85
86                 // printf("%d %d %d -> %d (%d %d)\n", i, k, j, din[i][k], st, dr);
87
88                 if (din[i][k] >= MOD) din[i][k] -= MOD;
89             }
90             // printf("%d %d -> %d\n", i, k, din[i][k]);
91         }
92     }
93
94     printf("%d\n", din[N][K]);
95
96     // printf("%d\n", jeg);
97     // printf("brut = %d\n", get_brut(N, K) % MOD);

```

```

99     return 0;
100 }

```

Listing 28.2.3: kmax_100b.cpp

```

1 // Savin Tiberiu
2 // Solutie O(N^2 * K)
3
4 #include <stdio.h>
5 #include <algorithm>
6
7 using namespace std;
8
9 #define NMAX 310
10
11 int A[NMAX][NMAX];
12 int comb[NMAX][NMAX];
13 int N, K;
14 int MOD;
15
16 int main()
17 {
18     freopen("kmax.in", "r", stdin);
19     freopen("kmax.out", "w", stdout);
20
21     scanf("%d %d %d ", &N, &K, &MOD);
22
23     int i, j, l;
24     for (i = 1; i <= K; i++)
25     {
26         A[1][i] = 1;
27     }
28
29     // combinari
30     for (i = 1; i <= N; i++)
31     {
32         comb[i][1] = i;
33         comb[i][i] = 1;
34     }
35
36     for (i = 3; i <= N; i++)
37     {
38         for (j = 2; j < i; j++)
39             comb[i][j] = (comb[i - 1][j] + comb[i - 1][j - 1]) % MOD;
40
41         for (j = 1; j <= K; j++)
42         {
43             A[i][j] = (A[i - 1][j - 1] + A[i - 1][j]) % MOD;
44             for (l = 1; l < i; l++)
45             {
46                 int x;
47                 x = (A[l][K - 1] * A[i - 1 - l][j]) % MOD;
48                 x = (x * comb[i - 1][l]) % MOD;
49                 A[i][j] = (A[i][j] + x) % MOD;
50             }
51         }
52     }
53     printf("%d ", A[N][K]);
54     return 0;
55 }

```

28.2.3 *Rezolvare detaliată

28.3 submatrix

Problema 3 - submatrix

100 de puncte

Miruna a găsit pe fundul mării o matrice cu n linii și m coloane având elementele numere naturale. Din motive necunoscute, Mirunel, prietenul misterios al Mirunei, vrea să afle care este

latura celei mai mari submatrice pătratice care conține maxim k numere distincte.

Submatricea cu colțul stânga-sus (xs, ys) și colțul dreapta-jos (xd, yd) este formată din toate elementele din matrice având indicele liniei în intervalul $[xs, xd]$ și indicele coloanei în intervalul $[ys, yd]$.

Cerințe

Scrieți un program care să determine latura maximă a unei submatrice care respectă condițiile lui Mirunel.

Date de intrare

Fișierul de intrare **submatrix.in** conține pe prima linie trei numere naturale n, m și k separate prin câte un singur spațiu având semnificația din enunț.

Pe următoarele n linii se găsesc câte m numere naturale separate prin spațiu, reprezentând elementele matricei.

Date de ieșire

Fișierul de ieșire **submatrix.out** va conține o singură linie pe care va fi scris un singur număr natural, reprezentând latura unei submatrice cu proprietatea din enunț.

Restricții și precizări

- $1 \leq n, m \leq 300$
- $41 \leq k \leq n * m$
- Pentru 30% din teste $1 \leq n, m \leq 30$
- Pentru 70% din teste $41 \leq n, m \leq 150$

Exemplu:

submatrix.in	submatrix.out	Explicații
<pre> 5 7 3 6 5 7 3 6 6 7 5 7 5 5 7 3 7 3 3 5 3 5 6 7 7 7 5 5 6 7 7 7 6 5 6 3 5 </pre>	3	O soluție este submatricea având colțul din stânga-sus pe poziția (2, 3) și latura 3, care conține doar elementele 3, 5 și 7.

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

Memorie: total **16 MB**

28.3.1 Indicații de rezolvare

Andrei Grigorean

Vom calcula pentru fiecare poziție latura maximă a submatricei pătratice care are colțul din dreapta jos în poziția respectivă și care conține maxim k numere distincte.

Vom reține aceste valori într-o matrice *best*.

Observăm că $best[i][j] \leq best[i-1][j-1] + 1$.

Vom calcula valorile matricei *best* pentru fiecare diagonală independent.

În momentul în care dorim să trecem dintr-o stare (i, j) în starea $(i + 1, j + 1)$, vom încerca să extindem pătratul cu colțul în (i, j) cu o unitate.

Dacă noua submatrice va conține mai mult de k numere distincte, o vom micșora atât cât este necesar - micșorarea se face păstrând colțul din dreapta jos în $(i + 1, j + 1)$.

Pentru a putea efectua rapid operațiile descrise, vom *normalize* valorile din matrice și ne vom menține un *vector de frecvențe*.

Această soluție are complexitatea $O(N^3)$, deoarece pentru fiecare diagonală vom parurge fiecare element al matricei de maxim două ori.

28.3.2 Cod sursă

Listing 28.3.1: submatrix_20.cpp

```

1 // Andrei Grigorean - O(N^5 log N)
2 #include <cassert>
3 #include <cstdio>
4 #include <algorithm>
5 #include <set>
6
7 using namespace std;
8
9 const int MAX_N = 300;
10
11 int n, m, k;
12 int a[MAX_N][MAX_N];
13 int best;
14
15 void read()
16 {
17     assert(freopen("submatrix.in", "r", stdin) != NULL);
18     assert(freopen("submatrix.out", "w", stdout) != NULL);
19
20     assert(scanf("%d %d %d", &n, &m, &k) == 3);
21     assert(1 <= n && n <= MAX_N);
22     assert(1 <= m && m <= MAX_N);
23     assert(1 <= k && k <= n * m);
24     for (int i = 0; i < n; ++i)
25         for (int j = 0; j < m; ++j)
26             assert(scanf("%d", &a[i][j]) == 1);
27 }
28
29 void solve()
30 {
31     set<int> values;
32
33     for (int x = 0; x < n; ++x)
34         for (int y = 0; y < m; ++y)
35             for (int size = 1; x + size <= n && y + size <= m; ++size)
36             {
37                 values.clear();
38
39                 for (int i = x; i < x + size; ++i)
40                     for (int j = y; j < y + size; ++j)
41                         values.insert(a[i][j]);
42
43                 if (int(values.size()) <= k)
44                     best = max(best, size);
45             }
46 }
47
48 void write()
49 {
50     printf("%d\n", best);
51 }
52
53 int main()
54 {
55     read();
56     solve();
57     write();
58 }
```

Listing 28.3.2: submatrix_40_adrian.cpp

```

1 /*
2     Airinei Adrian, Iasi
3     Complexitate O(N^4) optimizat
4 */
5 #include <cstdio>
6 #include <cstring>
7 #include <vector>
8 #include <algorithm>
```

```

9
10 using namespace std;
11
12 #define MAXN 310
13
14 int N, M, K;
15 int A[MAXN][MAXN], cnt[MAXN*MAXN];
16
17 int solve(void)
18 {
19     int i, j, k, L, x, y, num, res = 1;
20
21     for(i = 1; i <= N; i++)
22         for(j = 1; j <= M; j++)
23         {
24             num = 0;
25             for(L = 1; i-L+1 >= 1 && j-L+1 >= 1; L++)
26             {
27                 for(x = i; x >= i-L+1; x--)
28                     if(cnt[A[x][j-L+1]] == 0)
29                         num++, cnt[A[x][j-L+1]] = 1;
30                 for(y = j; y > j-L+1; y--)
31                     if(cnt[A[i-L+1][y]] == 0)
32                         num++, cnt[A[i-L+1][y]] = 1;
33                 if(num > K) break;
34             }
35             res = max(res, L-1);
36             for(x = i-L+1; x <= i; x++)
37                 for(y = j-L+1; y <= j; y++)
38                     cnt[A[x][y]] = 0;
39         }
40     return res;
41 }
42
43 vector<int> V;
44 int main(void)
45 {
46     freopen("submatrix.in", "rt", stdin);
47     freopen("submatrix.out", "wt", stdout);
48
49     int i, j, k;
50     scanf("%d %d %d", &N, &M, &K);
51     for(i = 1; i <= N; i++)
52         for(j = 1; j <= M; j++)
53             scanf("%d", &A[i][j]), V.push_back(A[i][j]);
54
55     vector<int>::iterator it;
56     sort(V.begin(), V.end());
57     V.resize(unique(V.begin(), V.end())-V.begin());
58     for(i = 1; i <= N; i++)
59         for(j = 1; j <= M; j++)
60         {
61             it = lower_bound(V.begin(), V.end(), A[i][j]);
62             A[i][j] = (int)(it-V.begin());
63         }
64
65     printf("%d\n", solve());
66
67     return 0;
68 }
```

Listing 28.3.3: submatrix_70.cpp

```

1 // Andrei Grigorean - O(N^3 log N)
2 #include <cassert>
3 #include <cstdio>
4 #include <algorithm>
5 #include <map>
6
7 using namespace std;
8
9 const int MAX_N = 300;
10
11 int n, m, k;
12 int a[MAX_N][MAX_N];
```

```

13
14 map<int, int> freq;
15 int distinct;
16
17 int best;
18
19 void read()
20 {
21     assert(freopen("submatrix.in", "r", stdin) != NULL);
22     assert(freopen("submatrix.out", "w", stdout) != NULL);
23
24     assert(scanf("%d %d %d", &n, &m, &k) == 3);
25     assert(1 <= n && n <= MAX_N);
26     assert(1 <= m && m <= MAX_N);
27     assert(1 <= k && k <= n * m);
28     for (int i = 0; i < n; ++i)
29         for (int j = 0; j < m; ++j)
30             assert(scanf("%d", &a[i][j]) == 1);
31 }
32
33 inline void insert(int x, int y)
34 {
35     if (++freq[a[x][y]] == 1) ++distinct;
36 }
37
38 inline void remove(int x, int y)
39 {
40     if (--freq[a[x][y]] == 0) --distinct;
41 }
42
43 void solve()
44 {
45     for (int line = 0; line < n; ++line)
46     {
47         freq.clear();
48         distinct = 0;
49
50         for (int p1 = 0, p2 = 0; p1 < m; ++p1)
51         {
52             // Check for too large squares
53             if (p1 - p2 > line)
54             {
55                 for (int i = 0; i <= line; ++i)
56                     remove(i, p2);
57                 for (int i = 1; i < p1 - p2; ++i)
58                     remove(0, p2 + i);
59                 ++p2;
60             }
61
62             // Increase the size of the square by 1
63             for (int i = 0; i <= p1 - p2; ++i)
64                 insert(line - i, p1);
65             for (int i = 0; i < p1 - p2; ++i)
66                 insert(line - (p1 - p2), p2 + i);
67
68             // Decrease the size of the square while necessary
69             while (distinct > k)
70             {
71                 for (int i = 0; i <= p1 - p2; ++i)
72                     remove(line - i, p2);
73                 for (int i = 1; i <= p1 - p2; ++i)
74                     remove(line - (p1 - p2), p2 + i);
75                 ++p2;
76             }
77
78             // Check for solution
79             best = max(best, p1 - p2 + 1);
80         }
81     }
82 }
83
84 void write()
85 {
86     printf("%d\n", best);
87 }
88

```

```

89 int main()
90 {
91     read();
92     solve();
93     write();
94 }
```

Listing 28.3.4: submatrix_100.cpp

```

1 // Andrei Grigorean - O(N^3)
2 #include <cassert>
3 #include <cstdio>
4 #include <cstring>
5 #include <algorithm>
6 #include <map>
7
8 using namespace std;
9
10 #define x first
11 #define y second
12
13 const int MAX_N = 300;
14
15 int n, m, k;
16 int a[MAX_N][MAX_N];
17
18 int freq[MAX_N * MAX_N];
19 int distinct;
20
21 int best;
22 int ans[MAX_N][MAX_N];
23
24 void read()
25 {
26     assert(freopen("submatrix.in", "r", stdin) != NULL);
27     assert(freopen("submatrix.out", "w", stdout) != NULL);
28
29     assert(scanf("%d %d %d", &n, &m, &k) == 3);
30     assert(1 <= n && n <= MAX_N);
31     assert(1 <= m && m <= MAX_N);
32     assert(1 <= k && k <= n * m);
33     for (int i = 0; i < n; ++i)
34         for (int j = 0; j < m; ++j)
35             assert(scanf("%d", &a[i][j]) == 1);
36 }
37
38 void normalize()
39 {
40     int values_no = 0;
41     map<int, int> values;
42
43     for (int i = 0; i < n; ++i)
44         for (int j = 0; j < m; ++j)
45             if (values.find(a[i][j]) == values.end())
46                 values[a[i][j]] = values_no++;
47
48     for (int i = 0; i < n; ++i)
49         for (int j = 0; j < m; ++j)
50             a[i][j] = values[a[i][j]];
51 }
52
53 inline void insert(int x, int y)
54 {
55     if (++freq[a[x][y]] == 1) ++distinct;
56 }
57
58 inline void remove(int x, int y)
59 {
60     if (--freq[a[x][y]] == 0) --distinct;
61 }
62
63 void solve()
64 {
65     normalize();
```

```

67     pair<int, int> start = make_pair(n - 1, 0);
68
69     for (int step = 0; step < n + m - 1; ++step)
70     {
71         pair<int, int> p1 = start;
72         pair<int, int> p2 = start;
73
74         distinct = 0;
75         memset(freq, 0, sizeof(freq));
76
77         while (p1.x < n && p1.y < m)
78         {
79             // Increase the size of the square by 1
80             for (int i = 0; i <= (p1.x - p2.x); ++i)
81                 insert(p1.x - i, p1.y);
82             for (int i = 1; i <= (p1.x - p2.x); ++i)
83                 insert(p1.x, p1.y - i);
84
85             // Decrease the size of the square while necessary
86             while (distinct > k)
87             {
88                 for (int i = 0; i <= (p1.x - p2.x); ++i)
89                     remove(p2.x + i, p2.y);
90                 for (int i = 1; i <= (p1.x - p2.x); ++i)
91                     remove(p2.x, p2.y + i);
92                     ++p2.x; ++p2.y;
93             }
94
95             // Check for solution
96             best = max(best, p1.x - p2.x + 1);
97             ans[p1.x][p1.y] = p1.x - p2.x + 1;
98
99             ++p1.x; ++p1.y;
100        }
101
102        if (start.x == 0) ++start.y;
103        else --start.x;
104    }
105 }
106
107 void write()
108 {
109     printf("%d\n", best);
110 }
111
112 int main()
113 {
114     read();
115     solve();
116     write();
117 }
```

Listing 28.3.5: submatrix_100_adrian.cpp

```

1  /*
2   * Airinei Adrian, Iasi
3   * Complexitate O(N^3)
4   */
5 #include <iostream>
6 #include <cstring>
7 #include <algorithm>
8 #include <vector>
9
10 using namespace std;
11
12 #define MAXN 310
13
14 int N, M, K, A[MAXN][MAXN], cnt[MAXN*MAXN], res;
15 vector<int> V;
16
17 void diag(int i, int j)
18 {
19     int x, y, k, num = 1, L;
20     L = 1, cnt[A[i][j]] = 1;
21     while(i+1 <= N && j+1 <= M)
```

```

22     {
23         i++, j++, L++;
24         for(x = i-L+1; x <= i; x++)
25             if(cnt[A[x][j]] == 0)
26                 num++, cnt[A[x][j]] = 1;
27             else
28                 cnt[A[x][j]]++;
29         for(y = j-L+1; y < j; y++)
30             if(cnt[A[i][y]] == 0)
31                 num++, cnt[A[i][y]] = 1;
32             else
33                 cnt[A[i][y]]++;
34         while(num > K)
35         {
36             for(x = i-L+1; x <= i; x++)
37             {
38                 cnt[A[x][j-L+1]]--;
39                 if(cnt[A[x][j-L+1]] == 0)
40                     num--;
41             }
42             for(y = j; y > j-L+1; y--)
43             {
44                 cnt[A[i-L+1][y]]--;
45                 if(cnt[A[i-L+1][y]] == 0)
46                     num--;
47             }
48             L--;
49         }
50         res = max(res, L);
51     }
52 }
53 void read_and_solve(void)
54 {
55     int i, j, k;
56     scanf("%d %d %d", &N, &M, &K);
57     for(i = 1; i <= N; i++)
58         for(j = 1; j <= M; j++)
59             scanf("%d", &A[i][j]), V.push_back(A[i][j]);
60
61     vector<int>::iterator it;
62     sort(V.begin(), V.end());
63     V.resize(unique(V.begin(), V.end())-V.begin());
64
65     for(i = 1; i <= N; i++)
66         for(j = 1; j <= M; j++)
67         {
68             it = lower_bound(V.begin(), V.end(), A[i][j]);
69             A[i][j] = (int)(it-V.begin());
70         }
71     res = 1;
72     for(i = 1; i <= N; i++)
73     {
74         memset(cnt, 0, sizeof(cnt));
75         diag(i, 1);
76     }
77     for(j = 2; j <= M; j++)
78     {
79         memset(cnt, 0, sizeof(cnt));
80         diag(1, j);
81     }
82 }
83
84 printf("%d\n", res);
85 }
86
87 int main(void)
88 {
89     freopen("submatrix.in", "rt", stdin);
90     freopen("submatrix.out", "wt", stdout);
91
92     read_and_solve();
93
94     return 0;
95 }
```

28.3.3 *Rezolvare detaliată

28.4 minuni

Problema 4 - minuni

100 de puncte

Regele Gefghev s-a gândit să viziteze o țară în care se întâmplă multe minuni, țara Minunilor. Această țară este alcătuită din N orașe, numerotate de la 1 la N . Între orașele i și $i+1$ ($1 \leq i < N$) există o stradă modernă pe care se poate circula doar de la orașul i la orașul $i+1$. Fiind un tip istet, Gefghev și-a dat seama că el poate ajunge de la orice oraș i la un oraș j ($j > i$) mergând în total pe $j-i$ străzi.

Tronomir, angajat al direcției de întreținere a drumurilor, a pus la cale un plan de construcție a M noi străzi pentru a-i fi mai ușor lui Gefghev să călăorească. Străzile vor fi construite rând pe rând, în M zile consecutive, în ziua i fiind construită cea de-a i -a stradă. O stradă va fi construită de la orașul a la orașul b ($1 \leq a < b - 1 < N$) și pe această stradă se va putea circula doar de la orașul a către orașul b .

Fiind un tip preocupat de călătorii, Gefghev se gândește acum să afle, după fiecare stradă construită, între care orașe poate ajunge acum mai repede decât putea înainte de construcția străzii. Altfel spus, el vrea să știe câte perechi de orașe (x, y) ($1 \leq x < y \leq N$) și-au schimbat distanța minimă dintre ele.

Distanța minimă dintre două orașe x și y este numărul minim de străzi care trebuie să fie parcurse pentru a ajunge din orașul x în orașul y .

Cerințe

Scrieți un program care determină pentru regele Gefghev, după fiecare stradă construită, numărul de perechi de orașe (x, y) ($1 \leq x < y \leq N$) pentru care distanța minimă de la x la y s-a modificat după construirea străzii respective.

Date de intrare

Pe prima linie a fișierului de intrare **minuni.in** se află două numere întregi N și M separate printr-un singur spațiu, cu semnificația din enunț. Pe următoarele M linii se află câte două numere naturale separate de un singur spațiu, a și b , cu semnificația că s-a construit o stradă de la orașul a la orașul b . Străzile sunt construite în ordinea din fișierul de intrare.

Date de ieșire

În fișierul de ieșire **minuni.out** se vor afla M numere naturale, câte un număr pe o linie. Pe linia i se va scrie numărul de perechi de orașe (x, y) ($1 \leq x < y \leq N$) pentru care distanța minimă de la x la y s-a modificat, după construirea celei de a i -a străzi din fișierul de intrare.

Restricții și precizări

- $1 \leq N \leq 1\,000\,000\,000$
- $1 \leq M \leq 100\,000$
- Toate orașele între care s-au construit străzi noi sunt distințe.
- Nu există două străzi dintre cele nou construite, una (a, b) și cealaltă (c, d) astfel încât $a \leq c \leq b \leq d$.
- Pentru 30% din teste $M \leq 1\,000$.

Exemple:

minuni.in	minuni.out	Explicații
8 3	10	Există 8 orașe. Se vor construi 3 noi străzi. După construcția străzii 2 4, se vor modifica distanțele dintre următoarele perechi de orașe: (1 4), (1 5), (1 6), (1 7), (1 8), (2 4), (2 5), (2 6), (2 7), (2 8).
2 4	4	
1 5	6	
6 8		

Timp maxim de executare/test: **0.4** secunde

Memorie: total **64 MB**

28.4.1 Indicații de rezolvare

stud. Airinei Adrian

Asociem problemei un graf orientat cu N noduri, initial fiind muchie intre oricare doua noduri consecutive.

Trebuie sa facem urmatoarele observatii atunci cand introducem o muchie noua $x \rightarrow y$ ($x < y$)

- daca nu exista o muchie $a \rightarrow b$ introdusa anterior astfel incat $a < x$ si $b > y$ atunci raspunsul este $x^*(N-y+1)$
- daca exista o muchie $a \rightarrow b$ introdusa anterior astfel incat $a < x$ si $b > y$, fie $a \rightarrow b$ muchia cu valoarea "a" maxima

Atunci observam ca drumurile care isi schimba distanta minima intre ele sunt perechi de forma (p,q) astfel

1. $p \leq x$ si $y \leq q < b$
2. $a < p \leq x$ si $q \geq y$
3. trebuie sa scadem acum perechile unde $a < p \leq x$ si $y \leq q < b$ pentru ca le-am numarat de doua ori

Astfel raspunsul este $x^*(b-y)+(x-a)^*(N-y+1)-(x-a)^*(b-y)$.

Deci trebuie sa aflam pentru fiecare muchie (x,y) introdusa muchia (a,b) despre care am facut referire anterior.

Cea mai usoara modalitate de a face acest lucru este sa observam ca daca fiecarei muchii (x,y) i-am asocia o pereche de paranteze, o paranteza deschisa pentru x si una inchisa pentru y toate operatiile noastre ar determina la sfarsit un sir corect parantezat (asta datorita restrictiei impuse in enunt).

Pentru a k-a operatie (x,y) vom introduce intr-un vector doua triplete, unul $(x,0,k)$ si unul $(y,1,k)$ avand semnificatia ca am introdus o paranteza deschisa la pozitia x si timpul k (timpul este indicele operatiei) si o paranteza inchisa la timpul y si timpul k .

Vom sorta acest vector dupa prima valoare (corespunzatoare pozitiei).

Atunci de fiecare data cand apare un triplet $(x,0,k)$, semn ca deschidem o paranteza, vrem sa stim dintre toate parantezele care sunt deschise dar nu au fost inca inchise care este cea cu timpul mai mic decat k , cat mai tarziu introdusa in *stiva*.

Acest lucru se poate face usor folosind un *arbore de intervale*, doar cu operatiile de afilare a maximului pe un interval si modificare a unei valori.

In arborele de intervale vom introduce pentru fiecare nod asociat unui timp x momentul la care a fost introdus in stiva.

Astfel, trebuie sa aflam la un moment dat care nod din intervalul $[1,a]$ are valoarea maxima.

Daca apare un triplet $(y,1,k)$ trebuie sa *stergesem* nodul respectiv din arborele de intervale.

La sfarsit vom parcurgerea din nou operatiile si vom afisa raspunsul pentru fiecare.

Complexitatea va fi $O(M * \log M)$ timp si $O(M)$ memorie.

Mai exista alte solutii mai mult sau mai putin complicate folosind diferite structuri de date, in complexitate $O(M * \log M)$ sau $O(M * \log M * \log M)$.

28.4.2 Cod sursă

Listing 28.4.1: minuni1.cpp

```

1  /*
2   *          Airinei Adrian
3   *          Complexitate O(M*logM)
4  */
5 #include <cstdio>
6 #include <cstring>
7 #include <vector>
8 #include <set>
9 #include <algorithm>
```

```

10
11 using namespace std;
12
13 #define MAXN 100100
14 #define llong long long
15 #define MAXARB (1<<18)
16 #define left (nod << 1)
17 #define right ((nod << 1) | 1)
18 #define mid ((st+dr) >> 1)
19
20 int N, M;
21 int viz[MAXN], Ind[MAXN], T[MAXN];
22 pair<int, int> query[MAXN], A[2*MAXN];
23
24 int Max[MAXARB];
25
26 void update(int nod, int st, int dr, int a, int val)
27 {
28     if(st == dr) { Max[nod] = val; return ; }
29     if(a <= mid) update(left, st, mid, a, val);
30     if(a > mid) update(right, mid+1, dr, a, val);
31     Max[nod] = max(Max[left], Max[right]);
32 }
33
34 int querya(int nod, int st, int dr, int a, int b)
35 {
36     if(st > dr || a > b) return 0;
37     if(a <= st && dr <= b)
38         return Max[nod];
39     int q1 = 0, q2 = 0;
40     if(a <= mid) q1 = querya(left, st, mid, a, b);
41     if(b > mid) q2 = querya(right, mid+1, dr, a, b);
42     return max(q1, q2);
43 }
44
45 void preproc(void)
46 {
47     int i, j, k, timp = 0;
48     for(i = 1; i <= M; i++)
49     {
50         A[i].first = query[i].first, A[i].second = i,
51         A[M+i].first = query[i].second, A[M+i].second = i;
52         sort(A+1, A+2*M+1);
53         for(i = 1; i <= 2*M; i++)
54         {
55             if(viz[A[i].second] == 0) // deschid
56             {
57                 viz[A[i].second] = 1;
58                 j = querya(1, 1, M, 1, A[i].second-1);
59                 Ind[A[i].second] = T[j];
60                 T[++timp] = A[i].second;
61                 update(1, 1, M, A[i].second, timp);
62             }
63             else // inchid
64             {
65                 update(1, 1, M, A[i].second, 0);
66             }
67         }
68     }
69     void read_and_solve(void)
70 {
71     int i, j, k, a, b;
72
73     scanf("%d %d\n", &N, &M);
74
75     for(i = 1; i <= M; i++)
76     {
77         scanf("%d %d\n", &a, &b);
78         query[i].first = a, query[i].second = b;
79     }
80
81     preproc();
82
83     for(i = 1; i <= M; i++)
84     {
85         if(Ind[i] == 0)

```

```

86         printf("%lld\n", (llong)query[i].first*(N-query[i].second+1));
87     else
88     {
89         int p = query[Ind[i]].first, q = query[Ind[i]].second;
90         a = query[i].first, b = query[i].second;
91         printf("%lld\n",
92                 (llong)a*(q-b)+(llong)(a-p)*(N-b+1)-(llong)(a-p)*(q-b));
93     }
94 }
95 }
96
97 int main(void)
98 {
99     freopen("minuni.in", "rt", stdin);
100    freopen("minuni.out", "wt", stdout);
101
102    read_and_solve();
103
104    return 0;
105 }
```

Listing 28.4.2: minuni2.cpp

```

1 /*
2  * Airinei Adrian
3  * Complexitate O(X^2), X = adancimea maxima a sechetei parantezate
4 */
5 #include <iostream>
6 #include <cstring>
7 #include <vector>
8 #include <set>
9 #include <algorithm>
10
11 using namespace std;
12
13 #define MAXN 100100
14 #define ll long long
15 #define MAXARB (1<<18)
16 #define left (nod << 1)
17 #define right ((nod << 1) | 1)
18 #define mid ((st+dr) >> 1)
19
20 int N, M;
21 int viz[MAXN], Ind[MAXN], T[MAXN];
22 pair<int, int> query[MAXN], A[2*MAXN];
23
24 void preproc(void)
25 {
26     int i, j, k, timp = 0;
27     for(i = 1; i <= M; i++)
28         A[i].first = query[i].first, A[i].second = i,
29         A[M+i].first = query[i].second, A[M+i].second = i;
30     sort(A+1, A+2*M+1);
31     for(i = 1; i <= 2*M; i++)
32     {
33         if(viz[A[i].second] == 0) // deschid
34         {
35             viz[A[i].second] = 1;
36             for(j = timp; j >= 1; j--)
37                 if(T[j] < A[i].second)
38                     break;
39             Ind[A[i].second] = T[j];
40             T[++timp] = A[i].second;
41         }
42         else // inchid
43         {
44             timp--;
45         }
46     }
47 }
48
49 void read_and_solve(void)
50 {
51     int i, j, k, a, b;
```

```

53     scanf("%d %d\n", &N, &M);
54
55     for(i = 1; i <= M; i++)
56     {
57         scanf("%d %d\n", &a, &b);
58         query[i].first = a, query[i].second = b;
59     }
60
61     preproc();
62
63     for(i = 1; i <= M; i++)
64     {
65         if(Ind[i] == 0)
66             printf("%lld\n", (llong)query[i].first*(N-query[i].second+1));
67         else
68         {
69             int p = query[Ind[i]].first, q = query[Ind[i]].second;
70             a = query[i].first, b = query[i].second;
71             printf("%lld\n",
72                     (llong)a*(q-b)+(llong)(a-p)*(N-b+1)-(llong)(a-p)*(q-b));
73         }
74     }
75 }
76
77 int main(void)
78 {
79     freopen("minuni.in", "rt", stdin);
80     freopen("minuni.out", "wt", stdout);
81
82     read_and_solve();
83
84     return 0;
85 }
```

Listing 28.4.3: minuni3.cpp

```

1 //Cosmin
2 #include <stdio.h>
3 #include <vector>
4 #include <algorithm>
5
6 using namespace std;
7
8 #define MMAX 100010
9 #define INF 1000000010
10 #define LL long long
11
12 #define MP make_pair
13 #define ff first
14 #define ss second
15
16 int N, M;
17
18 int xx[MMAX], yy[MMAX];
19
20 vector<pair<int, int>> vec;
21 int poz[MMAX];
22
23 int aint[MMAX << 4];
24
25 void update(int x, int y, int q, int poz, int val)
26 {
27     if (x == y) { aint[q] = val; return; }
28
29     int mij = (x + y) >> 1;
30
31     if (poz <= mij) update(x, mij, q << 1, poz, val);
32     if (poz > mij) update(mij + 1, y, (q << 1) + 1, poz, val);
33
34     aint[q] = max(aint[q << 1], aint[(q << 1) + 1]);
35 }
36
37 vector<pair<pair<int, int>, int>> inter;
38
39 void get_inter(int x, int y, int q, int a, int b)
```

```

40  {
41      if (a <= x && y <= b)
42      {
43          inter.push_back(MP(MP(x, y), q));
44          return;
45      }
46
47      int mij = (x + y) >> 1;
48
49      if (a <= mij) get_inter(x, mij, q << 1, a, b);
50      if (mij < b) get_inter(mij + 1, y, (q << 1) + 1, a, b);
51  }
52
53  int get_val(int x, int y, int q, int val)
54  {
55      if (x == y) return vec[x].ss;
56
57      int mij = (x + y) >> 1;
58
59      if (aint[(q << 1) + 1] >= val)
60          return get_val(mij + 1, y, (q << 1) + 1, val);
61
62      return get_val(x, mij, q << 1, val);
63  }
64
65  int afla_tata(int x, int y, int val) // trebuie sa afli valoarea cea mai
66                                // din dreapta intervalului x-y care e >= val
67  {
68      if (x > y) return 0;
69
70      inter.clear();
71
72      get_inter(1, M, 1, x, y);
73
74      int n = inter.size();
75
76      for (int i = n - 1; i >= 0; i--)
77      {
78          int x = inter[i].ff.ff;
79          int y = inter[i].ff.ss;
80          int q = inter[i].ss;
81
82          if (aint[q] >= val) return get_val(x, y, q, val);
83      }
84
85      return 0;
86  }
87
88  int main()
89  {
90      int i, x, y, px, mt, xt, yt;
91
92      freopen("minuni.in", "r", stdin);
93      freopen("minuni.out", "w", stdout);
94
95      scanf("%d %d", &N, &M);
96
97      vec.push_back(MP(-1, 0));
98
99      xx[0] = 0; yy[0] = N + 1;
100     for (i = 1; i <= M; i++)
101     {
102         scanf("%d %d", &xx[i], &yy[i]);
103         vec.push_back(MP(xx[i], i));
104     }
105
106     sort(vec.begin(), vec.end());
107
108     for (i = 1; i <= M; i++) poz[vec[i].ss] = i;
109
110     for (i = 1; i <= M; i++)
111     {
112         x = xx[i], y = yy[i];
113         px = poz[i];
114
115         mt = afla_tata(1, px - 1, y);

```

```

116         xt = xx[mt]; yt = yy[mt];
117
118         LL nr = (LL) x * (yt-y)+(LL) (x-xt)*(N-y+1)-(LL) (x-xt)*(yt-y);
119
120         printf("%lld\n", nr);
121         update(1, M, 1, px, y);
122     }
123
124     return 0;
125 }
```

Listing 28.4.4: minuni4.cpp

```

1 // Andrei Grigorean - 100 puncte
2 #include <cassert>
3 #include <cstdio>
4 #include <algorithm>
5 #include <set>
6 #include <vector>
7
8 using namespace std;
9
10 const int MAX_LEN = 1000000000;
11 const int MAX_N = 100005;
12 const int INF = 0x3f3f3f3f;
13
14 struct Edge
15 {
16     int x, y;
17 };
18
19 struct Event
20 {
21     int time;
22     int type;
23     int index;
24
25     bool operator < (const Event &other) const
26     {
27         return time < other.time;
28     }
29 };
30
31 int length, n;
32 Edge edges[MAX_N];
33
34 int st[MAX_N];
35 vector<Event> events;
36 int aint[4 * MAX_N];
37 long long ans[MAX_N];
38
39 // Debug stuff
40 set<int> usedCities;
41
42 void read()
43 {
44     assert(freopen("minuni.in", "r", stdin) != NULL);
45     assert(freopen("minuni.out", "w", stdout) != NULL);
46
47     assert(scanf("%d %d", &length, &n) == 2);
48     assert(1 <= length && length <= MAX_LEN);
49     assert(1 <= n && n <= MAX_N);
50
51     for (int i = 1; i <= n; ++i)
52     {
53         assert(scanf("%d %d", &edges[i].x, &edges[i].y) == 2);
54         assert(1 <= edges[i].x && edges[i].x <= length);
55         assert(1 <= edges[i].y && edges[i].y <= length);
56
57         assert(usedCities.find(edges[i].x) == usedCities.end());
58         usedCities.insert(edges[i].x);
59
60         assert(usedCities.find(edges[i].y) == usedCities.end());
61         usedCities.insert(edges[i].y);
62     }
}
```

```

63     usedCities.insert(edges[i].y);
64 }
65 }
66
67 void update(int nod, int st, int dr, int a, int b)
68 {
69     if (st == dr)
70     {
71         aint[nod] = b;
72         return;
73     }
74     int mij = (st + dr) / 2;
75     if (a <= mij)
76         update(nod * 2, st, mij, a, b);
77     else
78         update(nod * 2 + 1, mij + 1, dr, a, b);
79
80     aint[nod] = max(aint[nod * 2], aint[nod * 2 + 1]);
81 }
82
83 int query(int nod, int st, int dr, int a, int b)
84 {
85     if (a <= st && dr <= b)
86         return aint[nod];
87
88     int ans = 0;
89     int mij = (st + dr) / 2;
90
91     if (a <= mij)
92         ans = max(ans, query(nod * 2, st, mij, a, b));
93     if (b > mij)
94         ans = max(ans, query(nod * 2 + 1, mij + 1, dr, a, b));
95
96     return ans;
97 }
98
99 void solve()
100 {
101     // Find events
102     for (int i = 1; i <= n; ++i)
103     {
104         events.push_back((Event){edges[i].x, 0, i});
105         events.push_back((Event){edges[i].y, 1, i});
106     }
107
108     sort(events.begin(), events.end());
109
110     // Do the monkey
111     for (int i = 0; i < int(events.size()); ++i)
112         if (events[i].type == 0)
113         {
114             int edge = events[i].index;
115
116             int stackIndex = query(1, 1, n, 1, edge);
117             int father = stackIndex ? st[stackIndex] : 0;
118
119             if (father)
120             {
121                 long long a = edges[father].x;
122                 long long b = edges[events[i].index].x;
123                 long long c = edges[events[i].index].y;
124                 long long d = edges[father].y;
125
126                 ans[events[i].index] = b * (d - c + 1) - a +
127                               (b - a + 1) * (length - c + 1) - (length-d +1) -
128                               (b - a + 1) * (d - c + 1) + 1;
129             }
130             else
131             {
132                 long long a = edges[edge].x;
133                 long long b = edges[edge].y;
134                 ans[events[i].index] = a * (length - b + 1);
135             }
136
137             st[++st[0]] = events[i].index;
138             update(1, 1, n, events[i].index, st[0]);

```

```

139     }
140     else
141     {
142         int index = st[st[0]];
143         --st[0];
144         assert(st[0] >= 0);
145         update(1, 1, n, index, 0);
146     }
147
148     assert(st[0] == 0);
149 }
150
151 void write()
152 {
153     for (int i = 1; i <= n; ++i)
154         printf("%lld\n", ans[i]);
155 }
156
157 int main()
158 {
159     read();
160     solve();
161     write();
162 }
```

Listing 28.4.5: minuni5.cpp

```

1 /*
2     Airinei Adrian
3     Complexitate O(M^2), brute-force
4 */
5 #include <cstdio>
6 #include <cstring>
7 #include <vector>
8 #include <algorithm>
9
10 using namespace std;
11
12 #define MAXN 100100
13 #define llong long long
14
15 int N, M;
16 int viz[MAXN], Ind[MAXN];
17 pair<int, int> query[MAXN];
18
19 void read_and_solve(void)
20 {
21     int i, j, k, a, b, amax;
22
23     scanf("%d %d\n", &N, &M);
24
25     for(i = 1; i <= M; i++)
26     {
27         scanf("%d %d\n", &a, &b);
28         query[i].first = a, query[i].second = b;
29         amax = 0;
30         k = 0;
31         for(j = i-1; j >= 1; j--)
32             if(query[j].first < a && query[j].second > b && query[j].first > amax)
33                 { amax = query[j].first; k = j; }
34
35         if(k == 0)
36             printf("%lld\n", (llong)query[i].first * (N - query[i].second + 1));
37         else
38         {
39             int p = query[k].first, q = query[k].second;
40             printf("%lld\n",
41                   (llong)a * (q - b) + (llong)(a - p) * (N - b + 1) - (llong)(a - p) * (q - b));
42         }
43     }
44 }
45
46 int main(void)
47 {
48     freopen("minuni.in", "rt", stdin);
```

```

49     freopen("minuni.out", "wt", stdout);
50
51     read_and_solve();
52
53     return 0;
54 }
```

28.4.3 *Rezolvare detaliată

28.5 diff

Problema 5 - diff

100 de puncte

Aurora tocmai a ajuns învățătoare la școală din cartier. În prima zi de școală ea a așezat toti cei N copii din școală într-un singur rând, apoi a numerotat copiii de la 1 la N , de la stânga la dreapta. Acum Aurora pune M întrebări de tipul: "există doi copii x și y ($x \leq y$) astfel încât diferența dintre numărul de băieți și numărul de fete situați în sir între copilul x și copilul y (inclusiv x și y) să fie exact K ; dacă da, dați un exemplu!"?

Cerințe

Scrieți un program care să răspundă la întrebările Aurorei.

Date de intrare

Pe prima linie a fișierului de intrare **diff.in** se vor afla numerele naturale N și M , separate printr-un spațiu, reprezentând numărul de copii și respectiv numărul de întrebări ale Aurorei.

Următoarea linie va conține N numere 0 sau 1 separate prin câte un spațiu; al i -lea număr de pe linie va fi 0 în cazul în care copilul i este fată, respectiv 1, în cazul în care copilul i este băiat.

Următoarele M linii vor conține cele M întrebări, câte o întrebare pe o linie. Pe cea de a i -a linie dintre cele M se află numărul natural K_i , specificat în cea de a i -a întrebare a Aurorei.

Date de ieșire

În fișierul de ieșire **diff.out** veți scrie M linii. Pe cea de a i -a linie ($1 \leq i \leq M$) se vor scrie două numere naturale x și y ($1 \leq x \leq y \leq N$), cu semnificația că diferența dintre numărul de băieți și numărul de fete situați în sir între copilul x și copilul y (inclusiv x și y) este exact K_i sau -1 în cazul în care nu există soluție.

Restricții și precizări

- $1 \leq N \leq 100\,000$
- $1 \leq M \leq 200\,000$
- $-1\,000\,000.000 \leq K_i \leq 1\,000\,000\,000$, pentru $1 \leq i \leq M$
- Pot exista mai multe răspunsuri corecte la o întrebare; afișați oricare dintre acestea.
- În răspunsul la o întrebare x poate fi egal cu y , caz în care este vorba de un singur copil.
- Pentru 20% din teste $N \leq 300$ și $M \leq 300$
- Pentru 40% din teste $N \leq 100\,000$ și $M \leq 500$
- Pentru 40% din teste $N \leq 3\,000$ și $M \leq 200\,000$

Exemple:

diff.in	diff.out	Explicații
10 4	6 10	Există 10 copii. Aurora formulează 4 întrebări.
0 0 1 0 0 1 1	1 5	La prima întrebare trebuie să fie determinați doi copii x și y astfel încât diferența dintre numărul de băieți și numărul de fete situați între x și y să fie 3. O soluție posibilă este x=6 și y=10 (între 6 și 10 există 4 băieți și o fată).
0 1 1	-1	
3	2 3	
-3		
10		
0		

Timp maxim de executare/test: **0.3** secunde

Memorie: total **16 MB**

28.5.1 Indicații de rezolvare

stud. Cosmin Gheorghe

Se poate usor observa ca daca incluim fiecare numar 0 din sir cu -1 problema se reduce la a gasi un interval de suma egala cu K_i , pentru fiecare query.

De asemenea suma maxima posibila ce se poate obtine este egala cu N si suma minima posibila este egala cu $-N$.

Asadar există maxim $2 * N$ sume posibile.

Dacă am reușit să precalculez cate un interval pentru fiecare suma posibila (sau -1 dacă suma nu se poate obține) am putea răspunde la fiecare întrebare în timp $O(1)$.

Pentru aceasta vom proceda în felul următor.

O să calculăm *sirul sumelor parțiale* S_i , unde fiecare S_i reprezintă suma primelor i elemente.

Calculăm pe rand toate sumele ce se pot obține cu primele i elemente pentru orice $i \leq N$.

Observăm că dacă avem sumele posibile pentru primele i elemente, cand vrem să calculăm sumele obținute cu primele $i + 1$ elemente trebuie doar să adăugăm sumele noi obținute ce se termină în poziția $i + 1$.

Dacă ne uitam la suma de pe un interval ca diferența dintre două elemente din sirul de sume parțiale, toate sumele noi posibile vor fi de forma $S_{i+1} - S_j$ (cu $j < i + 1$).

Pentru că elementul de pe poziția $i + 1$ este egal ori cu +1 sau -1, se observă că singurele sume parțiale S_j care pot fi de forma sume noi trebuie să fie ori maximul ori minimul dintre toate S_j -urile.

Asadar trebuie doar să retinem suma maxima și minima parțială ce apare înaintea poziției $i + 1$, și să verificăm pentru fiecare dintre aceasta dacă sumele generate de ele nu au fost obținute înainte.

Pentru a găsi ușor o sumă deja obținută sau nu vom folosi un *vector de frecvențe* și încă doi vectori pentru a retine pozitiile sevențelor ce generează sumele respective.

După ce precalculez toate aceste valori putem răspunde la fiecare întrebare în timp $O(1)$.

Această soluție are complexitatea $O(N + M)$.

28.5.2 Cod sursă

Listing 28.5.1: diff1.cpp

```

1 #include <stdio.h>
2
3 #define NMAX 100010
4
5 int N, M;
```

```

6
7 int off = NMAX;
8 char ap[NMAX + NMAX];
9 int xx[NMAX + NMAX];
10 int yy[NMAX + NMAX];
11
12 int main()
13 {
14     int i, q;
15
16     freopen("diff.in", "r", stdin);
17     freopen("diff.out", "w", stdout);
18
19     scanf("%d %d", &N, &M);
20
21     int s = 0, smin = 0, pmin = 0, smax = 0, pmax = 0;
22     for (i = 1; i <= N; i++)
23     {
24         scanf("%d", &q);
25
26         if (q == 0) q = -1;
27
28         s += q;
29
30         if (!ap[off + s - smin])
31         {
32             ap[off + s - smin] = 1;
33             xx[off + s - smin] = pmin + 1;
34             yy[off + s - smin] = i;
35         }
36
37         if (!ap[off + s - smax])
38         {
39             ap[off + s - smax] = 1;
40             xx[off + s - smax] = pmax + 1;
41             yy[off + s - smax] = i;
42         }
43
44         if (s < smin) smin = s, pmin = i;
45         if (s > smax) smax = s, pmax = i;
46     }
47
48     for (i = 1; i <= M; i++)
49     {
50         scanf("%d", &s);
51
52         if (!ap[off + s])
53             printf("-1\n");
54         else
55             printf("%d %d\n", xx[off + s], yy[off + s]);
56     }
57
58     return 0;
59 }
```

Listing 28.5.2: diff2.cpp

```

1 #include <stdio.h>
2
3 #define NMAX 100002
4 #define SHIFT 100000
5
6 int N, M;
7 int Hst[2 * NMAX], Hdr[2 * NMAX];
8 int V[NMAX];
9 int S[NMAX];
10
11 void init_hash()
12 {
13     int i;
14
15     for (i = 0; i <= 2 * NMAX; i++)
16     {
17         Hst[i] = -1;
18         Hdr[i] = -1;
```

```

19     }
20 }
21
22 void build_hash()
23 {
24     int i;
25     int min = 0;
26     int max = 0;
27     int pmin = 0;
28     int pmax = 0;
29     for (i = 1; i <= N; i++)
30     {
31         int x, y;
32         x = S[i] - max;
33         y = S[i] - min;
34
35         Hst[SHIFT + x] = pmax;
36         Hdr[SHIFT + x] = i;
37
38         Hst[SHIFT + y] = pmin;
39         Hdr[SHIFT + y] = i;
40
41         if (S[i] < min)
42         {
43             min = S[i];
44             pmin = i;
45         }
46
47         if (S[i] > max)
48         {
49             max = S[i];
50             pmax = i;
51         }
52     }
53 }
54
55 int main()
56 {
57     freopen("diff.in", "r", stdin);
58     freopen("diff.out", "w", stdout);
59
60     scanf("%d %d ", &N, &M);
61
62     int i;
63     for (i = 1; i <= N; i++)
64     {
65         scanf("%d ", &V[i]);
66
67         if (V[i] == 0)
68         {
69             V[i] = -1;
70         }
71
72         S[i] = S[i - 1] + V[i];
73     }
74
75     init_hash();
76     build_hash();
77
78     for (i = 1; i <= M; i++)
79     {
80         int D;
81         scanf("%d ", &D);
82
83         if (Hst[SHIFT + D] == -1)
84             printf("-1\n");
85         else
86             printf("%d %d\n", Hst[SHIFT + D] + 1, Hdr[SHIFT + D]);
87     }
88
89     return 0;
90 }
```

Listing 28.5.3: diff3.cpp

```

1 #include <cassert>
2 #include <cstdio>
3 #include <algorithm>
4
5 using namespace std;
6
7 const int MAX_N = 100000;
8 const int INF = 0x3f3f3f3f;
9 const int MAX_S = 1000000000;
10
11 int n, m;
12 int v[MAX_N];
13
14 pair<int, int> answer[2 * MAX_N + 1];
15 #define answer (answer + MAX_N)
16
17 void do_the_monkey(int pos, int add)
18 {
19     int sum = 0;
20     for (int i = pos; 0 <= i && i < n; i += add)
21     {
22         sum += v[i];
23         answer[sum] = make_pair(min(pos, i) + 1, max(pos, i) + 1);
24     }
25 }
26
27 int main()
28 {
29     assert(freopen("diff.in", "r", stdin) != NULL);
30     assert(freopen("diff.out", "w", stdout) != NULL);
31
32     assert(scanf("%d %d", &n, &m) == 2);
33     assert(1 <= n && n <= MAX_N);
34     assert(1 <= m && m <= 2 * n);
35
36     for (int i = 0; i < n; ++i)
37     {
38         assert(scanf("%d", &v[i]) == 1);
39         assert(v[i] == 0 || v[i] == 1);
40         if (!v[i]) v[i] = -1;
41     }
42
43     do_the_monkey(0, 1);
44     do_the_monkey(n-1, -1);
45
46     int minSum = INF, posMin = -1;
47     int maxSum = -INF, posMax = -1;
48     int sum = 0;
49     for (int i = 0; i < n; ++i)
50     {
51         sum += v[i];
52         if (sum < minSum)
53         {
54             minSum = sum;
55             posMin = i;
56         }
57         if (sum > maxSum)
58         {
59             maxSum = sum;
60             posMax = i;
61         }
62     }
63
64     do_the_monkey(posMin, 1);
65     do_the_monkey(posMin, -1);
66     do_the_monkey(posMax, 1);
67     do_the_monkey(posMax, -1);
68
69     for (int i = 0; i < m; ++i)
70     {
71         int sum;
72         assert(scanf("%d", &sum) == 1);
73         assert(-MAX_S <= sum && sum <= MAX_S);
74
75         if (answer[sum].first == 0)
76             printf("-1\n");

```

```

77     else
78         printf("%d %d\n", answer[sum].first, answer[sum].second);
79     }
80 }
```

Listing 28.5.4: diff4.cpp

```

1 #include <stdio.h>
2
3 #define NMAX 100002
4
5 int V[NMAX];
6 int S[NMAX];
7 int N, M;
8
9 int main()
10 {
11     freopen("diff.in", "r", stdin);
12     freopen("diff.out", "w", stdout);
13
14     scanf("%d %d ", &N, &M);
15
16     int i;
17     for (i = 1; i <= N; i++)
18     {
19         int x;
20         scanf("%d ", &x);
21         if (x == 0)
22             S[i] = S[i - 1] - 1;
23         else
24             S[i] = S[i - 1] + 1;
25     }
26
27     for (i = 1; i <= M; i++)
28     {
29         int x, y, D, sx = -1, sy = -1;
30         scanf("%d ", &D);
31         for (x = 1; x <= N; x++)
32             for (y = 1; y <= x; y++)
33                 if (S[x] - S[y - 1] == D)
34                 {
35                     sx = x;
36                     sy = y;
37                 }
38
39         if (sx == -1)
40             printf("-1\n");
41         else
42             printf("%d %d\n", sy, sx);
43     }
44
45     return 0;
46 }
```

Listing 28.5.5: diff5.cpp

```

1 #include <stdio.h>
2
3 #define NMAX 100002
4
5 int V[NMAX];
6 int S[NMAX];
7 int N, M;
8 int H[NMAX][4];
9
10 int main()
11 {
12     freopen("diff.in", "r", stdin);
13     freopen("diff.out", "w", stdout);
14
15     scanf("%d %d ", &N, &M);
16
17     int i;
18     int min = 0, max = 0, pmin = 0, pmax = 0;
```

```

19   for (i = 1; i <= N; i++)
20   {
21     int x, y;
22     scanf("%d ", &x);
23     if (x == 0)
24       S[i] = S[i - 1] - 1;
25     else
26       S[i] = S[i - 1] + 1;
27
28     x = S[i] - S[pmin];
29     y = S[i] - S[pmax];
30     H[i][0] = x;
31     H[i][1] = pmin;
32     H[i][2] = y;
33     H[i][3] = pmax;
34
35     if (S[i] < min)
36     {
37       min = S[i];
38       pmin = i;
39     }
40
41     if (S[i] > max)
42     {
43       max = S[i];
44       pmax = i;
45     }
46   }
47
48   for (i = 1; i <= M; i++)
49   {
50     int x, D, sx = -1, sy = -1;
51     scanf("%d ", &D);
52     for (x = 1; x <= N; x++)
53     {
54       if (H[x][0] == D)
55       {
56         sx = x;
57         sy = H[x][1] + 1;
58       }
59
60       if (H[x][2] == D)
61       {
62         sx = x;
63         sy = H[x][3] + 1;
64       }
65     }
66
67     if (sx == -1)
68       printf("-1\n");
69     else
70       printf("%d %d\n", sy, sx);
71   }
72
73   return 0;
74 }
```

28.5.3 *Rezolvare detaliată

28.6 stalpi

Problema 6 - stalpi

100 de puncte

Veronel dorește să-și repare gardul care-i separă curtea de cea a vecinului său. Gardul este susținut de n stâlpi, amplasați coliniar, numerotați în ordine de la stânga la dreapta: 1, 2, ..., n . Aceștia se găsesc la distanțele d_i metri ($i = 2, 3, \dots, n$) față de primul stâlp.

Stâlpii 2, 3, ..., $n - 1$ pot fi mutați spre stânga sau spre dreapta. Stâlpul 1 și stâlpul n nu se pot muta.

Pentru simplitate, Veronel calculează efortul deplasării unui stâlp ca fiind egal cu distanța de deplasare. Fie D cea mai mare distanță dintre doi stâlpi consecutivi, după efectuarea tuturor mutărilor.

Cerințe

Cunoscând efortul total maxim E pe care Veronel este dispus să-l facă pentru deplasarea stâlpilor să se determine cea mai mică valoare posibilă pentru D , care se poate obține conform condițiilor din enunț, astfel încât să nu se depășească efortul E . Efortul total este definit ca suma eforturilor pe care Veronel le face pentru deplasarea stâlpilor.

Date de intrare

Pe prima linie a fișierului de intrare **stalpi.in** se află două numere naturale n și E separate printr-un singur spațiu, cu semnificația din enunț. Pe linia următoare se află $n - 1$ numere naturale d_2, d_3, \dots, d_n , separate prin câte un singur spațiu, reprezentând distanțele inițiale ale stâlpilor 2, 3, ..., n față de stâlpul 1.

Date de ieșire

Fișierul de ieșire **stalpi.out** va conține o singură linie pe care se va scrie un număr natural D , cu semnificația din enunț.

Restricții și precizări

- Stâlpii pot fi mutați doar pe poziții a căror distanță față de stâlpul 1 este exprimată prin valori naturale.
 - $0 \leq d_2 \leq d_3 \leq \dots \leq d_n \leq 10\ 000$
 - $3 \leq n \leq 50$
 - $1 \leq E \leq 400\ 000$
 - Pentru 20% din teste $n = 3$ și $d_n \leq 100$
 - Pentru 40% din teste $n \leq 15$ și $d_n \leq 200$
 - Pentru 70% din teste $n \leq 15$ și $d_n \leq 1\ 000$
 - Pentru 100% din teste $n \leq 50$ și $d_n \leq 10\ 000$

Exemplu:

stalpi.in	stalpi.out	Explicații
4 10 10 30 50	17	Se mută stâlpul 2 cu 7 metri spre dreapta și stâlpul 3 cu 3 metri spre dreapta.

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

Memorie: total **64 MB**

28.6.1 Indicații de rezolvare

prof. Constantin Gălățan

Se observă că putem cauta binar soluția. Altă observație pe care o putem face este că orice soluție optimă respectă condiția

$$df[2] \leq df[3] \leq \dots \leq df[n],$$

unde $df[i]$ este distanța finală fata de primul stâlp la care este plasat stâlpul i .

Acum vrem pentru o distanță D fixată să stim care este efortul minim astfel încât distanța dintre oricare doi stâlpi consecutivi sa fie mai mică sau egală decât D .

Pentru a realiza acest lucru vom folosi tehnica programării dinamice.

Vom calcula pentru primii i stâlpi efortul minim necesar astfel încât stâlpul i sa fie plasat la distanța j fata de primul stâlp, și distanța între oricare doi stâlpi consecutivi sa fie mai mică sau egală decat D .

Aceste valori le vom retine într-o matrice $Best[i][j]$. Recurența este urmatoarea:

$$Best[i][j] = \min(Best[i-1][j-k]) + |Poz[i]-j|, 0 \leq k \leq D$$

Complexitatea acestei soluții este $O(D^2 * N * \log(D))$.

Pentru a reduce complexitatea la $O(D * N * \log(D))$ vom folosi un *dequeue*.

In loc de dequeue se pot folosi alte structuri care permit aflarea minimului pe un interval în timp logaritmic pentru a obține complexitatea $O(D * N * \log(D) * \log(N))$.

28.6.2 Cod sursă

Listing 28.6.1: stalpi1.cpp

```

1 //by Puni Andrei-Paul
2 //Time complexity: O(N*D*log D)
3 //Space complexity: O(N*D)
4 //Method: DP + deque
5 //Working time: 45 minutes
6
7 #include <cstdio>
8 #include <cassert>
9 #include <cstdlib>
10 #include <cstring>
11 #include <vector>
12
13 using namespace std;
14
15 #define Nmax 51
16 #define Emax 400000
17 #define Dmax 10001
18 #define Inf best[0][0]
19
20 int N,E, d[Nmax];
21
22 int best [Nmax] [Dmax];
23
24 int dq[Dmax], st, dr;
25
26
27 bool se_poate(int D)
28 {
29     memset(best, 0x3f, sizeof best);
30
31     //si in rest infinit ... pt ca nu pot sa il misc pe 1 ...
32     best[1][0] = 0;
33
34     for (int i = 2; i <= N; ++i)
35     {
36         //init dq
37         st = 1; dr = 0;
38
39         for (int j = 0; j <= d[N]; ++j)
40         {
41             //le scoti pe alea de care nu ai nevoie
42             while (st <= dr && abs(j - dq[st]) > D)
43                 ++st;
44
45             //le scoti pe alea care nus destul de bune
46             while (st <= dr && best[i-1][dq[dr]] >= best[i-1][j])
47                 --dr;
48
49             //bagi aia noua
50             dq[++dr] = j;
51
52             best[i][j] = abs(d[i]-j) + best[i-1][dq[st]];
53             if (best[i][j] > Inf)
54                 best[i][j] = Inf;
55         }
56     }
57
58     return (best[N][d[N]] <= E);
59 }
60
61 int main()
62 {
63     assert(freopen("stalpi.in", "r", stdin) != NULL);
64     assert(freopen("stalpi.out", "w", stdout) != NULL);
65
66     assert(scanf("%d%d", &N, &E) == 2);
67     assert(3 <= N && N < Nmax);
68     assert(1 <= E && E <= Emax);
69
70     for (int i = 2; i <= N; ++i)

```

```

71     {
72         assert(scanf("%d", &d[i]) == 1);
73         assert(d[i-1] <= d[i] && d[i] < Dmax);
74     }
75
76     int ret = d[N];
77
78     for (int i = 1<<20; i > 0; i /= 2)
79     if (ret - i >= 0)
80     if (se_poate(ret - i))
81         ret -= i;
82
83     printf("%d\n", ret);
84
85     return 0;
86 }
```

Listing 28.6.2: stalpi2.cpp

```

1 #include <vector>
2 #include <algorithm>
3 #include <iostream>
4 #include <fstream>
5
6 using namespace std;
7
8 #define MAXD 10001
9 #define MAXN 51
10#define INT_MAX 0x3f3f3f3f
11
12 int n;
13 int a[2][MAXD];
14 int R;
15 int p[MAXN];
16 int D;
17 int E;
18
19 int Test(int r);
20
21 ifstream fin("stalpi.in");
22 ofstream fout("stalpi.out");
23
24 int main()
25 {
26     fin >> n >> E;
27
28     p[0] = 0;
29     for (int i = 1; i < n; ++i)
30         fin >> p[i];
31     D = p[n-1];
32     int m = 0;
33     int lo = 0, hi = D;
34     while (lo <= hi)
35     {
36         m = (lo + hi) / 2;
37
38         if (Test(m))
39             hi = m - 1;
40         else
41             lo = m + 1;
42     }
43     R = hi + 1;
44     fout << R << '\n';
45
46     fin.close();
47     fout.close();
48 }
49
50 int Test(int r)
51 {
52
53     for (int i = 0; i < 2; ++i)
54         for (int j = 0; j <= D; ++j)
55             a[i][j] = INT_MAX;
56 }
```

```

57     for ( int j = 0; j <= D; ++j )
58     {
59         if ( j > r )
60             continue;
61         if ( j == p[1] )
62             a[1][j] = 0;
63         else
64             a[1][j] = abs(j - p[1]);
65     }
66
67     int cr = 0, pr = 1;
68     for ( int i = 2; i < n - 1; ++i, cr = !cr, pr = !pr )
69     {
70         for ( int j = 0; j <= D; ++j )
71             for ( int k = 0; k <= r && k <= j; ++k )
72                 if ( j - k >= 0 && a[pr][j-k] != INT_MAX )
73                     a[cr][j] = min(a[cr][j], a[pr][j-k] + abs(j - p[i]));
74
75     for ( int j = D - r; j <= D; ++j )
76         if ( a[pr][j] <= E )
77             return true;
78     return false;
79 }
```

Listing 28.6.3: stalpi3.cpp

```

1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<string.h>
4 #include<algorithm>
5
6 using namespace std;
7
8 const int maxn = 50;
9 const int maxd = 10000;
10 const int INF = 10000000;
11
12 int DIN[maxn][maxd];
13 int D[maxn], E, N;
14
15 int main()
16 {
17     int s = 0;
18     freopen("stalpi.in", "r", stdin);
19     freopen("stalpi.out", "w", stdout);
20     scanf("%d %d\n", &N, &E);
21
22     for(int i = 2; i <= N; ++i)
23     {
24         scanf("%d", &D[i]);
25         s = D[i];
26     }
27
28     for(int i = 1; i <= s; ++i)
29     {
30         memset(DIN, 0, sizeof(DIN));
31
32         for(int k = 1; k <= N; ++k)
33             for(int j = 0; j <= s; ++j) DIN[k][j] = INF;
34         DIN[1][0] = 0;
35         for(int j = 2; j < N; ++j)
36             for(int k = 0; k <= s; ++k)
37                 for(int l = k; l >= 0 && k - l <= i; --l)
38                     DIN[j][k] = min(DIN[j][k], DIN[j-1][l] + abs(k - D[j]));
39
40         for(int j = s; j > 0 && s - j <= i; --j)
41             if (DIN[N-1][j] <= E)
42                 printf("%d\n", i); return 0;
43     }
44
45     return 0;
46 }
```

Listing 28.6.4: stalpi4.cpp

```
1 #include <stdio.h>
2 #include <vector>
3 #include <algorithm>
4
5 using namespace std;
6
7 #define NMAX 10002
8
9 int N, F;
10 int D[NMAX];
11 int poz[NMAX];
12
13 int main()
14 {
15     freopen("stalpi.in", "r", stdin);
16     freopen("stalpi.out", "w", stdout);
17
18     scanf("%d %d ", &N, &F);
19
20     int i;
21     for (i = 1; i < N; i++)
22         scanf("%d ", &D[i]);
23
24     int dist = D[N - 1] / (N - 1);
25
26     for (i = 1; i < N; i++)
27     {
28         poz[i] = dist * i;
29         if (i <= D[N - 1] % (N - 1))
30             poz[i]++;
31     }
32     poz[N - 1] = D[N - 1];
33
34     int E = 0, diff;
35     for (i = 1; i < N; i++)
36     {
37         if (poz[i] < D[i])
38         {
39             diff = D[i] - poz[i];
40             E += diff;
41         }
42         else
43         {
44             diff = poz[i] - D[i];
45             E += diff;
46         }
47     }
48
49     int nr = E - F;
50     for (int i = 1; nr; i++)
51     {
52         if (i == N - 1)
53             i = 1;
54
55         if (poz[i] < D[i])
56         {
57             poz[i]++;
58             nr--;
59         }
60         else
61             if (poz[i] > D[i])
62             {
63                 nr--;
64                 poz[i]--;
65             }
66     }
67
68     int ret = D[1];
69     for (i = 2; i < N; i++)
70         if (poz[i] - poz[i - 1] > ret)
71             ret = poz[i] - poz[i - 1];
72
73     printf("%d ", ret);
74     return 0;
75 }
```

Listing 28.6.5: stalpi5.cpp

```

1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<string.h>
4 #include<algorithm>
5
6 using namespace std;
7
8 const int maxn = 52;
9 const int maxd = 10020;
10 const int INF = 10000000;
11
12 int DIN[maxn][maxd];
13 int DAUX[maxn], D[maxn], E, N;
14
15 bool ver(int d)
16 {
17     for(int i = 1; i <= N; ++i)
18         DAUX[i] = D[i];
19     int ef = E;
20     int nr = 5;
21     while(nr)
22     {
23         int move = 0;
24         for(int i = 2; i < N; ++i)
25         {
26             if (DAUX[i] - DAUX[i - 1] > d)
27             {
28                 move = 1;
29                 ef += abs(DAUX[i] - D[i]);
30                 ef -= DAUX[i] - DAUX[i - 1] - d;
31                 DAUX[i] = DAUX[i - 1] + d;
32             }
33         }
34         for(int i = N - 1; i > 1; --i)
35         {
36             if (DAUX[i + 1] - DAUX[i] > d)
37             {
38                 move = 1;
39                 ef += abs(DAUX[i] - D[i]);
40                 ef -= DAUX[i + 1] - DAUX[i] - d;
41                 DAUX[i] = DAUX[i + 1] - d;
42             }
43         }
44     }
45
46     if (!move) return true;
47     if (ef < 0) return false;
48     --nr;
49 }
50
51 return false;
52 }
53
54 int main()
55 {
56     int s = 0;
57     freopen("stalpi.in", "r", stdin);
58     freopen("stalpi.out", "w", stdout);
59     scanf("%d %d\n", &N, &E);
60
61     for(int i = 2; i <= N; ++i)
62     {
63         scanf("%d", &D[i]);
64         s = D[i];
65     }
66     int x = 0, p = 0;
67     for( p = 1; p <= D[N]; p <= 1);
68
69     for(; p; p >= 1)
70         if (!ver(x + p))
71             x += p;
72
73     printf("%d\n", x + 1);
74     return 0;

```

28.6.3 *Rezolvare detaliată

Capitolul 29

ONI 2009

29.1 reinvent

Problema 1 - reinvent

100 de puncte

Zăharel și Sică s-au gândit să se reinventeze din punct de vedere spiritual. În prima fază, vor să se mute în orașul Sala. Orașul Sala conține N case (numerotate de la 1 la N) unite prin M străzi bidirectionale de lungimi egale. Ei au la dispoziție fonduri limitate și pot să se mute doar într-un cartier mărginaș format din X case. Fiindcă sunt buni prieteni cei doi vor să se mute în două case distincte, cât mai apropiate între ele.

Cerințe

Determinați distanța minimă dintre două case distincte din cele X din cartier.

Date de intrare

Pe prima linie a fișierului de intrare **reinvent.in** se află trei numere întregi N , M și X separate printr-un singur spațiu, cu semnificația din enunț. Pe următoarele M linii se află câte două numere întregi distincte separate printr-un singur spațiu, reprezentând o stradă bidirectională din oraș. Ultima linie din fișier conține X numere naturale distincte separate prin câte un spațiu, reprezentând casele din cartier.

Date de ieșire

În fișierul de ieșire **reinvent.out** se află un singur număr natural, reprezentând distanța minimă între două case distincte din cartier.

Restricții și precizări

- $1 \leq N, M \leq 100.000$
- $2 \leq X \leq N$
- Pentru 30% din teste $N \leq 1024$
- Distanța între două case se măsoară prin numărul minim de străzi necesare pentru a ajunge de la o casă la cealaltă
- Între oricare două case există cel mult o stradă bidirectională.
- Există cel puțin două case din cartier astfel încât să existe drum între ele

Exemple:

reinvent.in	reinvent.out	Explicații
5 6 2	3	Distanța minimă între casele 1 și 5 din cartier este 3. Un drum posibil format din 3 străzi este 1 - 2 - 3 - 5.
1 2		
2 3		
2 4		
3 4		
1 4		
3 5		
1 5		

Timp maxim de executare/test: **0.3** secunde

Memorie: total **16 MB**

29.1.1 Indicații de rezolvare

Mircea Pașoi

Modelăm orașul Sala cu un *graf neorientat* cu N noduri și M muchii.

Inserăm într-o *coadă* cele X noduri initiale și aplicăm algoritmul de *parcuregere în lățime* (BF).

Pe parcursul algoritmului, ținem pentru fiecare nod din coadă și rădăcina lui (un nod din cele X). În momentul în care un nod " n " apare în coadă cu două rădăcini distincte (fie acestea "r1" și "r2"), avem o soluție: un drum de la $r1$ la $r2$ care trece prin n . Faptul ca folosim o *parcuregere BF* ne garantează că soluția găsită este minimă. Complexitatea rezolvării este $O(N + M)$.

29.1.2 Cod sursă

Listing 29.1.1: reinvent.cpp

```

1  /* Mircea Pasoi, solutie O(N+M) */
2  #include <stdio.h>
3  #include <string.h>
4  #include <queue>
5
6  using namespace std;
7
8  #define MAX_N 100005
9  #define FIN "reinvent.in"
10 #define FOUT "reinvent.out"
11
12 int N, M, X, src[MAX_N], dist[MAX_N];
13 vector<int> G[MAX_N];
14 queue<int> Q;
15
16 int main(void)
17 {
18     int i, j, n;
19
20     freopen(FIN, "r", stdin);
21     freopen(FOUT, "w", stdout);
22
23     scanf("%d %d %d", &N, &M, &X);
24     for (; M; --M)
25     {
26         scanf("%d %d", &i, &j);
27         G[i].push_back(j);
28         G[j].push_back(i);
29     }
30
31     memset(src, -1, sizeof(src));
32
33     for (; X; --X)
34     {
35         scanf("%d", &n);
36         Q.push(n);
37         src[n] = n;
38     }
39
40     while (!Q.empty())
41     {
42         n = Q.front();
43         Q.pop();
44         for (vector<int>::iterator it = G[n].begin(); it != G[n].end(); ++it)
45             if (src[*it] == -1)
46             {
47                 Q.push(*it);
48                 src[*it] = src[n];
49                 dist[*it] = dist[n]+1;
50             }
51         else
52             if (src[*it] != src[n])
53             {
54                 printf("%d\n", dist[*it]+dist[n]+1);
55             }
56     }
57 }
```

```

56         }
57     }
58
59     return 0;
60 }
```

Listing 29.1.2: reinvent2.cpp

```

1  /* Adrian Airinei, sursa "ciucuiala" */
2  #include <cstdio>
3  #include <ctime>
4  #include <algorithm>
5  #include <queue>
6  #include <vector>
7  #include <cstdlib>
8  #include <cstring>
9
10 using namespace std;
11
12 #define pb push_back
13 #define MAXN 100100
14
15 int N, M, X, dmin[MAXN];
16 vector<int> G[MAXN];
17 int ok[MAXN], viz[MAXN], aux[MAXN];
18
19 int bfs(int x)
20 {
21     queue<int> Q;
22     vector<int>::iterator it;
23     aux[0] = 1, aux[1] = x;
24     Q.push(x), viz[x] = 1, dmin[x] = 0;
25     while(1)
26     {
27         int y = Q.front(); Q.pop();
28         if(ok[y] && y != x)
29         {
30             for(int i = 1; i <= aux[0]; i++)
31                 viz[aux[i]] = 0;
32             return dmin[y];
33         }
34         for(it = G[y].begin(); it != G[y].end(); it++)
35             if(!viz[*it])
36                 viz[*it] = 1, dmin[*it] = dmin[y]+1,
37                 Q.push(*it), aux[+aux[0]] = *it;
38     }
39 }
40
41 int A[MAXN];
42
43 int main(void)
44 {
45     freopen("reinvent.in", "rt", stdin);
46     freopen("reinvent.out", "wt", stdout);
47
48     int i, j, a, b;
49     double start = clock();
50
51     srand(time(0));
52
53     scanf ("%d %d %d\n", &N, &M, &X);
54
55     for(i = 1; i <= M; i++)
56     {
57         char sir[256];
58         fgets(sir, 256, stdin);
59         int ind = 0;
60         for(a = 0; sir[ind] >= '0' && sir[ind] <= '9'; ind++)
61             a = a * 10 + (sir[ind]-'0');
62         ind++;
63         for(b = 0; sir[ind] >= '0' && sir[ind] <= '9'; ind++)
64             b = b * 10 + (sir[ind]-'0');
65         //scanf("%d %d", &a, &b);
66         G[a].pb(b), G[b].pb(a);
67     }
```

```

68     for(i = 1; i <= X; i++)
69     {
70         scanf("%d ", &a);
71         ok[a] = 1, A[++A[0]] = a;
72     }
73
74     int res = (1<<30);
75     while( (clock()-start)/CLOCKS_PER_SEC < 0.18)
76     {
77         int q;
78         q = A[rand()%X+1];
79         res = min(res, bfs(q));
80     }
81 }
82
83 printf("%d", res);
84
85 return 0;
86 }
```

Listing 29.1.3: reinvent3.cpp

```

1 /* Airinei Adrian, sursa brute */
2 #include <iostream>
3 #include <algorithm>
4 #include <queue>
5 #include <vector>
6
7 using namespace std;
8
9 #define pb push_back
10#define MAXN 100100
11
12 int N, M, X, dmin[MAXN];
13 vector<int> G[MAXN];
14 int ok[MAXN], viz[MAXN], aux[MAXN];
15
16 int bfs(int x)
17 {
18     queue<int> Q;
19     vector<int>::iterator it;
20     aux[0] = 1, aux[1] = x;
21     Q.push(x), viz[x] = 1, dmin[x] = 0;
22     while(1)
23     {
24         int y = Q.front(); Q.pop();
25         if(ok[y] && y != x) {
26             for(int i = 1; i <= aux[0]; i++)
27                 viz[aux[i]] = 0;
28             return dmin[y];
29         }
30         for(it = G[y].begin(); it != G[y].end(); it++)
31             if(!viz[*it])
32                 viz[*it] = 1, dmin[*it] = dmin[y]+1,
33                 Q.push(*it), aux[+aux[0]] = *it;
34     }
35 }
36
37 int main(void)
38 {
39     freopen("reinvent.in", "rt", stdin);
40     freopen("reinvent.out", "wt", stdout);
41
42     int i, j, a, b;
43     scanf ("%d %d %d\n", &N, &M, &X);
44     for(i = 1; i <= M; i++)
45     {
46         char sir[256];
47         fgets(sir, 256, stdin);
48         int ind = 0;
49         for(a = 0; sir[ind] >= '0' && sir[ind] <= '9'; ind++)
50             a = a * 10 + (sir[ind]-'0');
51         ind++;
52         for(b = 0; sir[ind] >= '0' && sir[ind] <= '9'; ind++)
53             b = b * 10 + (sir[ind]-'0');
```

```

54     //scanf("%d %d", &a, &b);
55     //scanf("%d %d", &a, &b);
56     G[a].pb(b), G[b].pb(a);
57 }
58
59     for(i = 1; i <= X; i++)
60         scanf("%d ", &a), ok[a] = 1;
61     int res = (1<<30);
62     for(i = 1; i <= N; i++)
63         if(ok[i])
64             res = min(res, bfs(i));
65     printf("%d\n", res);
66     return 0;
67 }
```

29.1.3 *Rezolvare detaliată

29.2 revolutie

Problema 2 - revolutie

100 de puncte

În țara Utopia a avut loc recent o revoluție digitală, în urma căreia s-a hotărât să se îñtrerupă serviciile de telefonie mobilă. Din fericire, Miruna s-a infiltrat în sediul central al principalului furnizor de telefonie din Utopia. Pentru a repune în funcțiune rețeaua, Miruna trebuie să treacă de un filtru de autentificare: ea are în față o matrice pătratică de dimensiune N având elemente din multimea $\{0, 1\}$. Asupra acestei matrice se pot efectua următoarele operații:

- se aleg două linii și se interschimbă;
- se aleg două coloane și se interschimbă.

Pentru a trece de filtrul de autentificare Miruna trebuie să obțină pe diagonala principală (toate elementele de forma $A[i][i]$) valori egale cu 1.

Cerințe

Determinați pentru Miruna o secvență de maxim $4 * N$ operații astfel încât să reușească să treacă de filtrul de autentificare.

Date de intrare

Fișierul de intrare **revolutie.in** va conține pe prima linie numărul N , reprezentând dimensiunea matricei. Pe următoarele N linii se află câte N valori din multimea $\{0, 1\}$ reprezentând valorile matricei.

Date de ieșire

Fișierul de ieșire **revolutie.out** va conține pe prima linie un singur număr întreg T reprezentând numărul de operații efectuate. Fiecare din următoarele T linii va descrie câte o operație: primul caracter de pe linie va fi L dacă s-a aplicat o operație asupra liniilor și C dacă s-a aplicat o operație asupra coloanelor. Vor urma două valori între 1 și N reprezentând indicii liniilor/coloanelor interschimbată. În caz că nu există soluție, se va afișa valoarea -1.

Restricții și precizări

- $1 \leq N \leq 127$
- Liniile și coloanele sunt numerotate de la 1 la N
- T trebuie să aparțină intervalului $[0, 4 * N]$
- 30% dintre fișierele de test vor avea $1 \leq N \leq 10$
- În cazul în care există mai multe soluții, se va afișa oricare dintre ele
- Miruna se opune fenomenului de politeness

Exemplu:

revolutie.in	revolutie.out	Explicații
2	1	Interschimbând primele două linii obținem matricea:
0 1	L 1 2	1 0
1 0		0 1

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

Memorie: total **16 MB**

29.2.1 Indicații de rezolvare

Andrei Grigorean

Vom modela problema sub forma unui *graf bipartit* cu N noduri în stânga și N noduri în dreapta.

Vom introduce o mulțime între nodul cu numărul i din stânga și nodul cu numărul j din stânga dacă și numai dacă elementul $A[i][j]$ este egal cu 1.

A efectua o operație de interschimbare a două linii este echivalent cu a interschimba două noduri din partea stângă a grafului, iar o operație de interschimbare a două coloane este echivalent cu a interschimba două noduri din partea dreaptă a grafului.

De aici putem trage concluzia că vom reuși să obținem pe diagonala principală a matricei elemente egale cu 1 dacă și numai dacă există un *cuplaj* de cardinal N în graf.

După ce identificăm un cuplaj putem construi o soluție interschimbând doar linii: vom efectua la început o operație pentru linia 1 și linia corespunzătoare nodului din stânga cuplat cu nodul 1 din dreapta, apoi pentru linia 2 și linia corespunzătoare nodului din stânga cuplat cu nodul 2 din dreapta, etc.

29.2.2 Cod sursă

Listing 29.2.1: revolutie.cpp

```

1 #include <cstdio>
2 #include <cstring>
3 #include <algorithm>
4 #include <string>
5 #include <sstream>
6 #include <fstream>
7
8 using namespace std;
9
10 const int MAX_N = 128;
11 const int MAX_T = 10;
12
13 int n;
14 int a[MAX_N][MAX_N];
15 int cup[MAX_N];
16 char vis[MAX_N];
17
18 int cupleaza(int i)
19 {
20     if (vis[i]) return 0;
21     vis[i] = 1;
22     for (int j = 1; j <= n; ++j)
23         if (a[i][j] && (!cup[j] || (cup[j] != i && cupleaza(cup[j]))))
24         {
25             cup[j] = i;
26             return 1;
27         }
28     return 0;
29 }
30
31 int main()
32 {
33     for (int t = 1; t <= MAX_T; ++t)
34     {
35         ostringstream filein;
36         ostringstream fileout;

```

```

37         filein << t << "-revolutie.in";
38         fileout << t << "-revolutie.ok";
39
40         ifstream fin(filein.str().c_str());
41         ofstream fout(fileout.str().c_str());
42
43         fin >> n;
44         for (int i = 1; i <= n; ++i)
45             for (int j = 1; j <= n; ++j)
46                 fin >> a[i][j];
47
48         memset(cup, 0, sizeof(cup));
49
50         int sol = 0;
51         for (int i = 1; i <= n; ++i)
52         {
53             memset(vis, 0, sizeof(vis));
54             sol += cupleaza(i);
55         }
56
57         if (sol != n)
58         {
59             fout << "-1\n";
60             return 0;
61         }
62
63         fout << n << "\n";
64         for (int i = 1; i <= n; ++i)
65             for (int j = i; j <= n; ++j)
66                 if (cup[j] == i)
67                 {
68                     fout << "C " << i << " " << j << "\n";
69                     swap(cup[i], cup[j]);
70                 }
71         }
72     }
73 }
```

Listing 29.2.2: revolutie_eval.cpp

```

1 #include <cstdio>
2 #include <cstring>
3 #include <cstdlib>
4 #include <algorithm>
5
6 using namespace std;
7
8 const char FILEIN[] = "revolutie.in";
9 const char FILEOUT[] = "revolutie.out";
10
11 const int MAX_N = 128;
12
13 int n;
14 int a[MAX_N][MAX_N];
15
16 void terminate(int score, char* msg)
17 {
18     printf("%d %s\n", score, msg);
19     exit(0);
20 }
21
22 void lines(int x, int y)
23 {
24     for (int i = 1; i <= n; ++i)
25         swap(a[x][i], a[y][i]);
26 }
27
28 void columns(int x, int y)
29 {
30     for (int i = 1; i <= n; ++i)
31         swap(a[i][x], a[i][y]);
32 }
33
34 int main()
35 {
```

```

36     freopen(FILEIN, "r", stdin);
37     scanf("%d", &n);
38     for (int i = 1; i <= n; ++i)
39         for (int j = 1; j <= n; ++j)
40             scanf("%d", &a[i][j]);
41
42     if (!freopen(FILEOUT, "r", stdin))
43         terminate(0, "Fisier de iesire lipsa!");
44
45     int op;
46     if (scanf("%d", &op) != 1)
47         terminate(0, "Raspuns gresit!");
48
49     for (; op; --op) {
50         char ch; int i1, i2;
51         if (scanf(" %c %d %d", &ch, &i1, &i2) != 3)
52             terminate(0, "Raspuns gresit!");
53         if (ch != 'L' && ch != 'C') terminate(0, "Raspuns gresit!");
54         if (i1 < 1 || i1 > n) terminate(0, "Raspuns gresit!");
55         if (i2 < 1 || i2 > n) terminate(0, "Raspuns gresit!");
56         if (ch == 'L') lines(i1, i2);
57         if (ch == 'C') columns(i1, i2);
58     }
59
60     for (int i = 1; i <= n; ++i)
61         if (a[i][i] != 1)
62             terminate(0, "Raspuns gresit!");
63     terminate(10, "OK!");
64 }
```

Listing 29.2.3: revolutie_gener.cpp

```

1 #include <ctime>
2 #include <cstdlib>
3 #include <fstream>
4 #include <cstring>
5 #include <sstream>
6 #include <algorithm>
7
8 using namespace std;
9
10 const int MAX_N = 128;
11
12 int N[] = {4, 7, 10, 30, 50, 70, 90, 100, 110, 125};
13 int T[] = {6, 15, 25, 300, 400, 2500, 1500, 3000, 2999};
14 int A[MAX_N][MAX_N];
15
16 inline int random(int r) { return (rand() % r) + 1; }
17
18 void generate(int t)
19 {
20     memset(A, 0, sizeof(A));
21     int v[MAX_N];
22     for (int i = 1; i <= N[t]; ++i)
23         v[i] = i;
24     for (int i = N[t]; i; --i)
25         swap(v[i], v[random(i)]);
26
27     for (int i = 1; i <= N[t]; ++i)
28         A[i][v[i]] = 1;
29
30     for (int nr1 = 0; nr1 <= T[t]; ++nr1)
31         A[random(N[t])][random(N[t])] = 1;
32 }
33
34 int main()
35 {
36     srand((int)time(0));
37     for (int t = 0; t < sizeof(N)/sizeof(N[0]); ++t)
38     {
39         generate(t);
40
41         ostringstream file;
42         file << t + 1 << "-revolutie.in";
43         ofstream fout(file.str().c_str());
```

```

44     fout << N[t] << "\n";
45     for (int i = 1; i <= N[t]; ++i) {
46         for (int j = 1; j < N[t]; ++j)
47             fout << A[i][j] << " ";
48         fout << A[i][N[t]];
49         fout << "\n";
50     }
51 }
52 }
53 }
```

29.2.3 *Rezolvare detaliată

29.3 sirag

Problema 3 - sirag

100 de puncte

În căutarea visului său de preamărire, Lunasorab a dat peste un şirag de perle de lungime N , de diferite tipuri, reprezentate printr-o literă mică a alfabetului latin. Dintre acestea, perlele marcate cu $*$ sunt considerate magice. Ele se pot transforma (şi trebuie transformate) într-o perlă de orice tip nemagic. Acum Lunasorab a primit un număr natural L , şi ştie că dacă poate găsi o subsecvenţă de lungime cat mai mare în şiragul iniţial, formată din concatenarea repetată a unei subsecvenţe de lungime L a şiragului cu ea însăşi, o poate vinde pe piaţa neagră contra unei sume considerabile (Lunasorab nu se sfieşte în a folosi metode neconvenţionale pentru a-şi atinge scopurile).

Deoarece Lunasorab nu stă prea bine la capitolul stiinţe exacte (are alte talente), vă cere vouă ajutorul.

De asemenea, fiind mai neîncrezător din fire, Lunasorab vă cere răspunsul pentru mai multe şiraguri.

Cerinţe

Pentru fiecare şirag din fişierul de intrare, aflaţi cea mai lungă subsecvenţă a şiragului, care se poate obţine prin concatenarea unei subsecvenţe de lungime L .

Date de intrare

Fişierul de intrare **sirag.in** va conţine pe prima linie numărul natural T , reprezentând numărul de teste.

Pe următoarele $2 * T$ linii vor urma cele T teste, formataate după cum urmează: pe prima linie corespunzătoare unui test vor fi numerele naturale N şi L , separate printr-un singur spatiu, reprezentând lungimea şiragului, respectiv lungimea subsecvenţei folosite în repetiţie. Pe a doua linie se vor afla N caractere, reprezentând tipul celor N perle.

Date de ieşire

Fişierul de ieşire **sirag.out** va avea T linii, fiecare linie continând un număr natural M , ce reprezintă lungimea celei mai lungi subsecvenţe cu proprietatea menţionată pentru testul respectiv.

Restricţii şi precizări

- $1 \leq T \leq 10$
- $1 \leq N \leq 100.000$
- $1 \leq L \leq N$
- Tipurile perelor vor fi reprezentate ori printr-o literă mică a alfabetului latin, ori prin caracterul $*$ pentru perle magice
 - Pentru 10% din teste $L = 1$
 - Pentru 30% din teste $N \leq 2000$

Exemple:

sirag.in	sirag.out	Explicații
1 6 3 a*cab*	6	Şirul se poate transforma în abcabc, înlocuind prima perlă magică cu b, iar a doua cu c. Atunci secvența abc se repetă de 2 ori, și deci se poate obține o subsecvență de lungime 6 (tot siragul) cu proprietatea menționată.

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

Memorie: total **16 MB**

29.3.1 Indicații de rezolvare

Cătălin Ștefan Tiseanu, Florin Manea, Adrian Diaconu

1. Să începem mai întâi cu soluția de 10 de puncte ($L = 1$) și să vedem cum o putem extinde.

Observăm că este suficient să găsim o subsecvență care să conțină doar perle magice și perle de același tip ($P1$).

Să notăm cu w sirul celor N perle.

Pentru aceasta calculăm vectorul ajutător A , cu următoarea semnificație:

$A[i] =$ cel mai mare j , $1 \leq j \leq i$, astfel încât $w[j] \neq *$, dacă există i , altfel

Nu este greu de văzut cum se poate calcula acest vector în $O(N)$.

Odată ce l-am calculat, avem nevoie de vectorul T , cu următoarea semnificație:

$T[i] =$ cea mai lungă lungime a unei subsecvențe a sirului inițial, ce se termină pe i , și respectă proprietatea $P1$.

Din nou, și T poate fi calculat în $O(N)$.

Maximul valorilor din vector T va reprezenta soluția căutată.

2. Să vedem acum cum putem obține soluția de 100 de puncte, pentru orice valoare a lui L .

Vom folosi tot cei 2 vectori, A și T , numai că în loc să se refere la valori consecutive, se vor referi la valori din L în L .

Mai exact, vom împărți valorile sirului inițial în L clase, în funcție de restul împărțirii la L .

Acum, pentru fiecare dintre aceste L clase, vom calcula cei doi vectori numai în interiorul clasei respective, ca la soluția de 10 de puncte.

Având vectorul T calculat, observăm că o subsecvență căutată este formată din L benzi (clase) consecutive.

Mai exact, lungimea maximă a unei astfel de subsecvențe care se termină pe poziția i , este minimul valorilor $T[i - L + 1 \dots i]$.

Acum, problema s-a redus la determinarea minimului pe secvențe consecutive de lungime L .

Observăm că putem face această determinare în timp liniar, folosind structura de date *double-ended queue* (deque).

De asemenea, putem folosi un *heap*, pentru o soluție $O(NlogL)$.

29.3.2 Cod sursă

Listing 29.3.1: brute_sirag.cpp

```

1 #include <iostream>
2 #include <string.h>
3
4 using namespace std;
5
6 #define MAXN 100010
7 #define Sigma 26
8
9 int T, N, L, Sol;
10 char A[MAXN];
11 int match[MAXN];
12
13 int main()
14 {
15     ifstream fin("sirag.in");
16     ofstream fout("sirag.out");
17

```

```

18     int i, j, k, found;
19     fin >> T;
20     for ( ; T; T--)
21     {
22         fin >> N >> L >> (A+1);
23         Sol = 0;
24
25         for (i = 1; i <= N; i++)
26         {
27             memset(match, -1, sizeof(match));
28             for (j = i; j-L >= 0; j -= L)
29             {
30                 found = 0;
31                 for (k = 0; k < L; k++)
32                     if (A[j-k] != '*')
33                     {
34                         if (match[k] == -1) match[k] = A[j-k] - 'a';
35                         else if (match[k] != A[j-k] - 'a')
36                         {
37                             found = 1;
38                             break;
39                         }
40                     }
41
42                     if (found) break;
43                     Sol = max(Sol, i-j+L);
44                 }
45             }
46
47             fout << Sol << endl;
48         }
49
50     return 0;
51 }
```

Listing 29.3.2: brute_sirag2.cpp

```

1 //Code by Patcas Csaba
2 //Time complexity: O(N^2)
3 //Space complexity: O(L)
4 //Working time: 30 minutes
5
6 #include <stdio.h>
7 #include <vector>
8 #include <ctype.h>
9
10 using namespace std;
11
12 #define in_file "sirag.in"
13 #define out_file "sirag.out"
14 #define nBuf 8000005
15
16 #define FORN(i,n) for (int i=0;i<n;++i)
17 #define FOR(i,a,b) for (int i=a;i<=b;++i)
18 #define repeat do{
19 #define until(x) }while(!(x));
20
21 int n,l,ind,solution;
22 char buf[nBuf];
23 vector <int> a;
24
25 void solve()
26 {
27     solution=1;
28     vector <int> cand(l);
29     FORN(q,n+1-l-1)
30     {
31         FOR(w,q,q+l-1) cand[w-q]=a[w];
32         int start=q+1, now=l;
33         while (start+l<=n)
34         {
35             ++now;
36             bool ok=true;
37             FOR(w,start,start+l-1)
38                 if (a[w]!='*')
```

```

39             if (cand[w-start]=='*') cand[w-start]=a[w];
40             else
41                 if (cand[w-start] != a[w])
42                 {
43                     ok=false;
44                     break;
45                 }
46             if (ok) solution=max(solution,now);
47             else break;
48             start+=l;
49         }
50     }
51 }
52
53 int main()
54 {
55     FILE* fin=fopen(in_file,"r");
56     FILE* fout=fopen(out_file,"w");
57     int TestCases;
58     fscanf(fin, "%d", &TestCases);
59
60     for ( int i = 1; i <= TestCases; ++i ) {
61         printf("%d ",i);
62         fscanf(fin,"%d %d \n",&n,&l);
63         fscanf(fin, "%s \n", buf);
64
65         a.clear();
66         a.resize(n);
67         ind=0;
68         FORN(q,n)
69         {
70             a[q]=buf[q];
71             if (!(q%1000)) printf("%d ",q);
72         }
73
74         solve();
75
76         fprintf(fout,"%d\n",solution*l);
77     }
78
79     fclose(fin);
80     fclose(fout);
81
82     return 0;
83 }
```

Listing 29.3.3: brute_sirag3.cpp

```

1 //Code by Patcas Csaba
2 //Time complexity: O(N^2)
3 //Space complexity: O(L+N)
4 //Working time: 30 minutes
5
6 #include <stdio.h>
7 #include <vector>
8 #include <ctype.h>
9
10 using namespace std;
11
12 #define in_file "sirag.in"
13 #define out_file "sirag.out"
14 #define nBuf 8000005
15
16 #define FORN(i,n) for (int i=0;i<n;++i)
17 #define FOR(i,a,b) for (int i=a;i<=b;++i)
18
19 int n,l,ind,solution;
20 char buf[nBuf];
21 vector <int> a;
22 vector <bool> was;
23
24 void solve()
25 {
26     was.clear();
27     was.resize(n,false);
```

```

28     solution=1;
29     vector <int> cand(l);
30     FORN(q,n+1-l-1)
31         if (!was[q])
32         {
33             FOR(w,q,q+l-1) cand[w-q]=a[w];
34             int start=q+l, now=1;
35             while (start+l<=n)
36             {
37                 ++now;
38                 bool ok=true;
39                 FOR(w,start,start+l-1)
40                     if ( a[w] != '*' )
41                         if ( cand[w-start] == '*' ) cand[w-start]=a[w];
42                     else
43                         if (cand[w-start] != a[w] )
44                         {
45                             ok=false;
46                             break;
47                         }
48                 if (ok)
49                 {
50                     if ( a[start] != '*' ) was[start]=true;
51                     solution=max(solution,now);
52                 }
53                 else break;
54                 start+=l;
55             }
56         }
57     }
58
59     int main()
60     {
61         FILE* fin=fopen(in_file, "r");
62         FILE* fout=fopen(out_file, "w");
63         int TestCases;
64         fscanf(fin, "%d", &TestCases);
65
66         for ( int i = 1; i <= TestCases; ++i ) {
67             fscanf(fin, "%d %d \n", &n, &l);
68             fscanf(fin, "%s \n", buf);
69             a.clear();
70             a.resize(n);
71             ind=0;
72             FORN(q,n) a[q]=buf[q];
73             solve();
74             fprintf(fout, "%d\n", solution*l);
75             printf("%d\n", solution*l);
76         }
77
78         fclose(fin);
79         fclose(fout);
80
81         return 0;
82     }

```

Listing 29.3.4: sirag.cpp

```

1  /*
2  Autor: Catalin Stefan Tiseanu
3  Punctaj: 100
4  Complexitate: O(N)
5  */
6  #include <cstdio>
7
8  #define HOLE '*'
9  #define MAX_N (1<<17)
10
11 int N, L, A[MAX_N], T[MAX_N], deque_min[MAX_N];
12 char w[MAX_N];
13
14 int solve()
15 {
16     int ql = 1, qr = 0, sol = 0;
17

```

```

18     for ( int i = 1; i <= N; i++ )
19     {
20         if ( w[i] != HOLE || i <= L )
21             A[i] = i;
22         else
23             A[i] = A[i-L];
24
25         if ( ( i > L ) &&
26             (w[i] == HOLE || w[ A[i-L] ] == HOLE || w[i] == w[ A[i-L] ] ) )
27             T[i] = T[i-L] + 1;
28         else if ( i > L )
29             T[i] = ( i - A[i-L] ) / L;
30         else
31             T[i] = 1;
32
33         while ( qr >= ql && T[ deque_min[qr] ] >= T[i] )
34             --qr;
35         deque_min[++qr] = i;
36         while ( qr >= ql && i - deque_min[ql] + 1 > L )
37             ++ql;
38
39         if ( T[ deque_min[ql] ] > sol )
40             sol = T[ deque_min[ql] ];
41     }
42
43     return sol * L;
44 }
45
46 int main()
47 {
48     freopen("sirag.in", "r", stdin);
49     freopen("sirag.out", "w", stdout);
50
51     int T;
52
53     scanf("%d", &T );
54
55     for ( ; --T; )
56     {
57         scanf("%d %d", &N, &L );
58         scanf("%s", w + 1);
59         printf("%d\n", solve() );
60     }
61
62     return 0;
63 }
```

Listing 29.3.5: sirag_10.cpp

```

1  /*
2  Autor: Catalin Stefan Tiseanu
3  Punctaj: 10
4  Complexitate: O(N) [ doar pentru L = 1 ]
5  */
6  #include <cstdio>
7
8  #define HOLE '*'
9  #define MAX_N 1<<20
10
11 int N, L, A[MAX_N], T[MAX_N];
12 char w[MAX_N];
13
14 int solve()
15 {
16     int ql = 1, qr = 0, sol = 0;
17
18     T[0] = 0;
19     A[0] = 0;
20
21     for ( int i = 1; i <= N; i++ )
22     {
23         if ( w[i] != HOLE )
24             A[i] = i;
25         else
26             A[i] = A[i-1];
```

```

27         if (i>1 && (w[i]==HOLE || w[A[i-1]]==HOLE || w[i]==w[A[i-1]]))
28             T[i] = T[i-1] + 1;
29         else if ( i > 1 )
30             T[i] = i - A[i-1];
31         else
32             T[i] = 1;
33
34         if ( T[i] > sol )
35             sol = T[i];
36     }
37 }
38
39     return sol;
40 }
41
42 int main()
43 {
44     freopen("sirag.in", "r", stdin);
45     freopen("sirag.out", "w", stdout);
46
47     int T;
48
49     scanf("%d", &T );
50
51     for ( ; T--; )
52     {
53         scanf("%d %d", &N, &L);
54         scanf("%s", w + 1);
55         printf("%d\n", solve() );
56     }
57
58     return 0;
59 }
```

29.3.3 *Rezolvare detaliată

29.4 matrice

Problema 4 - matrice

100 de puncte

Laura a primit de ziua ei o matrice pătratică de dimensiuni $N \times N$ de numere întregi. Neștiind ce să facă cu ea, a început să-și pună diverse întrebări. Fata consideră că un drum de la (x_1, y_1) la (x_2, y_2) este o secvență de celule care începe în celula (x_1, y_1) , se termină în (x_2, y_2) și oricare două celule consecutive au o latură în comun (deplasarea se poate face spre nord, est, sud, vest).

Laura a definit costul unui drum ca fiind valoarea minimă a unei celule de pe acel drum.

Apoi ea a început să-și pună Q întrebări de forma: Care este costul maxim pe care îl poate avea un drum de la (x_1, y_1) la (x_2, y_2) ? Întrebările au început să i se pară dificile și vă cere ajutorul.

Cerințe

Aflați răspunsul pentru fiecare dintre cele Q întrebări.

Date de intrare

Pe prima linie a fișierului de intrare **matrice.in** se află 2 numere întregi N și Q cu semnificația din enunț.

Pe următoarele N linii se află câte N numere întregi reprezentând matricea primită de Laura. Fiecare dintre următoarele Q linii conține câte patru numere întregi x_1 y_1 x_2 y_2 care descriu câte o întrebare.

Date de ieșire

În fișierul de ieșire **matrice.out** se află răspunsul la cele Q întrebări, câte unul pe linie, în aceeași ordine în care au apărut în fișierul de intrare.

Restricții și precizări

- $1 \leq N \leq 300$
- $1 \leq Q \leq 20\,000$
- Elementele matricei sunt numere întregi cuprinse între 1 și 1 000 000.
- Pentru 15% din teste $N \leq 50$, $Q \leq 10$ și valorile matricei sunt cuprinse între 1 și 250.
- Pentru alte 20% din teste $N \leq 100$, $Q \leq 100$.
- Nu există nicio întrebare pentru care perechea (x_1, y_1) să coincidă cu perechea (x_2, y_2) .

Exemple:

matrice.in	matrice.out	Explicații
5 3	5	Pentru prima întrebare răspunsul este dat de următorul drum:
9 5 9 8 7	6	9 5 9 8 7
2 1 1 3 8	1	2 1 1 3 8
1 3 9 4 6		1 3 9 4 6
4 1 8 6 7		4 1 8 6 7
2 4 5 5 6		2 4 5 5 6
1 1 3 3		
5 5 1 5		
2 2 4 3		

Timp maxim de executare/test: **2.0** secunde

Memorie: total **16 MB**

29.4.1 Indicații de rezolvare

Paul - Dan Băltescu

Soluția ce obține 15 puncte are complexitate $O(N^2 * Q * H_{max})$.

Pentru fiecare query, se fixează pe rând toate înălțimile h cuprinse între 1 și H_{max} și se verifică cu ajutorul unei *parcurgeri în lățime* (Lee) dacă există drum de la (x_1, y_1) la (x_2, y_2) trecând doar prin celule cu înălțimea $\geq h$.

O soluție ce obține 35 de puncte se bazează pe observația că răspunsul la fiecare query poate fi *căutat binar*. Dacă pentru un h oarecare putem obține un drum de la (x_1, y_1) la (x_2, y_2) trecând doar prin celule de înălțime cel puțin h , atunci vom încerca să mărim h , altfel îl vom micșora. Astfel complexitatea devine $O(N^2 * Q * \log H_{max})$.

Pentru a obține 35 de puncte mai există o abordare.

Pentru fiecare query *căutăm binar* rezultatul h . Pentru un h fixat, se sortează elementele matricii crescător.

Considerăm *graful asociat matricii* G în care fiecare celulă devine nod, iar relație de vecinătate (nord, est, sud, vest) este o muchie.

Parcurgem elementele matricii astfel sortate și introducem pentru fiecare celulă nodul și muchiile corespunzătoare în G . Când valoarea elementelor parcurse devine $\leq h$ ne punem problema dacă (x_1, y_1) și (x_2, y_2) se află în aceeași *componentă conexă*.

Aceste operații pot fi efectuate în timp aproape constant cu ajutorul *structurilor* pentru *mulțimi de date disjuncte* și astfel complexitatea totală este $O(N^2 \log N + N^2 * Q * \log H_{max})$.

Solutia de 100 de puncte se bazează pe cea de-a doua soluție de 35 de puncte și pe o observație suplimentară. Se observă că abordarea precedentă nu merită făcută pentru un singur query, dar poate fi folosită pentru a răspunde la mai multe întrebări în același timp.

De aceea, pentru fiecare din cele Q query-uri vom căuta rezultatele binar în paralel.

Complexitatea obținută este $O(N^2 \log N + (N^2 + Q \log Q) \log H_{max})$.

Menționez că complexitatea se poate reduce la $O(N^2 \log N + (N^2 + Q) \log H_{max})$ cu ajutorul unei *interclasări* a query-urilor, dar nu este necesar.

29.4.2 Cod sursă

Listing 29.4.1: brute15.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 #define MAXN 310
6 #define MAXX 90010
7 #define MAXL 4
8
9 const int dirx[MAXL] = {-1, 0, 1, 0};
10 const int diry[MAXL] = {0, -1, 0, 1};
11
12 int N, Q, L;
13 int A[MAXN][MAXN];
14 int U[MAXN][MAXN];
15 int SX[MAXX], SY[MAXX];
16
17 int BFS(int X1, int Y1, int X2, int Y2, int H)
18 {
19     int i, j, cx, cy;
20
21     for (i = 1; i <= N; i++)
22         for (j = 1; j <= N; j++) U[i][j] = 0;
23
24     L = 1;
25     SX[L] = X1;
26     SY[L] = Y1;
27     U[X1][Y1] = 1;
28
29     for (i = 1; i <= L; i++)
30         for (j = 0; j < MAXL; j++)
31         {
32             cx = SX[i] + dirx[j];
33             cy = SY[i] + diry[j];
34
35             if (cx > 0 && cy > 0 && cx <= N && cy <= N && A[cx][cy] >= H && !U(cx)[cy])
36             {
37                 L++;
38                 SX[L] = cx;
39                 SY[L] = cy;
40                 U[cx][cy] = 1;
41
42                 if (cx == X2 && cy == Y2) return 1;
43             }
44         }
45
46     return 0;
47 }
48
49 int main()
50 {
51     ifstream fin("matrice.in");
52     ofstream fout("matrice.out");
53
54     int i, j, X1, Y1, X2, Y2;
55
56     fin >> N >> Q;
57
58     for (i = 1; i <= N; i++)
59         for (j = 1; j <= N; j++) fin >> A[i][j];
60
61     for (i = 1; i <= Q; i++)
62     {
63         fin >> X1 >> Y1 >> X2 >> Y2;
64
65         for (j = A[X1][Y1]; j > 0; j--)
66             if (BFS(X1, Y1, X2, Y2, j))
67             {
68                 fout << j << endl;
69                 break;
70             }
}

```

```

71     }
72     return 0;
73 }

```

Listing 29.4.2: brute35.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 #define MAXN 310
6 #define MAXX 90010
7 #define MAXL 4
8
9 const int dirx[MAXL] = {-1, 0, 1, 0};
10 const int diry[MAXL] = {0, 1, 0, -1};
11
12 int N, Q, L, Sol;
13 int A[MAXN][MAXN], U[MAXN][MAXN];
14 int SX[MAXX], SY[MAXX];
15
16 int BFS(int X1, int Y1, int X2, int Y2, int H)
17 {
18     int i, j, cx, cy;
19
20     for (i = 1; i <= N; i++)
21         for (j = 1; j <= N; j++) U[i][j] = 0;
22
23     L = 1;
24     SX[L] = X1, SY[L] = Y1;
25     U[X1][Y1] = 1;
26
27     for (i = 1; i <= L; i++)
28         for (j = 0; j < MAXL; j++)
29         {
30             cx = SX[i] + dirx[j];
31             cy = SY[i] + diry[j];
32
33             if (cx > 0 && cy > 0 && cx <= N && cy <= N && !U[cx][cy] && A[cx][cy] >= H)
34             {
35                 L++;
36                 SX[L] = cx;
37                 SY[L] = cy;
38                 U[cx][cy] = 1;
39
40                 if (cx == X2 && cy == Y2) return 1;
41             }
42         }
43
44     return 0;
45 }
46
47 int main()
48 {
49     ifstream fin("matrice.in");
50     ofstream fout("matrice.out");
51
52     int i, j, X1, Y1, X2, Y2;
53
54     fin >> N >> Q;
55
56     for (i = 1; i <= N; i++)
57         for (j = 1; j <= N; j++) fin >> A[i][j];
58
59     for (i = 1; i <= Q; i++)
60     {
61         fin >> X1 >> Y1 >> X2 >> Y2;
62
63         int front = 1, middle, back = A[X1][Y1];
64         Sol = 0;
65
66         while (front <= back)
67         {
68             middle = (front + back) / 2;

```

```

69             if (BFS(X1, Y1, X2, Y2, middle))
70             {
71                 front = middle + 1;
72                 Sol = middle;
73             }
74             else back = middle - 1;
75         }
76     }
77
78     fout << Sol << endl;
79 }
80
81     return 0;
82 }
```

Listing 29.4.3: brute.cpp

```

1 #include <iostream>
2 #include <algorithm>
3
4 using namespace std;
5
6 #define MAXN 310
7 #define MAXX 90010
8 #define MAXQ 20010
9 #define MAXL 4
10
11 const int dirx[MAXL] = {-1, 0, 1, 0};
12 const int diry[MAXL] = {0, 1, 0, -1};
13
14 int N, Q, L;
15 char B[MAXN][MAXN];
16 int W[MAXN][MAXN];
17 int X[MAXX], Y[MAXX], C[MAXX], P[MAXX];
18 int G[MAXX];
19 int Sol[MAXQ];
20 int Query[MAXQ], P2[MAXQ];
21 int X1[MAXQ], Y1[MAXQ], X2[MAXQ], Y2[MAXQ];
22
23 inline int cmp(int x, int y)
24 {
25     return C[x] > C[y];
26 }
27
28 inline int cmp2(int x, int y)
29 {
30     return Query[x] > Query[y];
31 }
32
33 inline int find_dad(int x)
34 {
35     if (x != G[x]) G[x] = find_dad(G[x]);
36     return G[x];
37 }
38
39 int main()
40 {
41     ifstream fin("matrice.in");
42     ofstream fout("matrice.out");
43
44     int i, j, k, p, step, cx, cy;
45
46     fin >> N >> Q;
47
48     for (i = 1; i <= N; i++)
49         for (j = 1; j <= N; j++)
50         {
51             L++;
52             W[i][j] = L;
53             X[L] = i, Y[L] = j;
54             fin >> C[L];
55             P[L] = L;
56         }
57
58     for (i = 1; i <= Q; i++) fin >> X1[i] >> Y1[i] >> X2[i] >> Y2[i];
```

```

59     sort(P+1, P+L+1, cmp);
60
61     for (step = 1; step < C[P[1]]; step <= 1);
62
63     for (; step; step >= 1)
64     {
65         for (i = 1; i <= Q; i++)
66         {
67             Query[i] = Sol[i] + step;
68             P2[i] = i;
69         }
70
71         sort(P2+1, P2+Q+1, cmp2);
72
73         for (i = 1; i <= L; i++) G[i] = i;
74
75         for (i = 1; i <= N; i++)
76             for (j = 1; j <= N; j++) B[i][j] = 0;
77
78         for (i = 1, j = 1; i <= L; )
79         {
80             for (k = j; j <= Q && Query[P2[j]] > C[P[i]]; j++)
81                 if (find_dad( W[X1[P2[j]]][Y1[P2[j]]] ) ==
82                     find_dad( W[X2[P2[j]]][Y2[P2[j]]] )) Sol[P2[j]] += step;
83
84             for (k = i; i <= L && C[P[i]] == C[P[k]]; i++)
85             {
86                 B[X[P[i]]][Y[P[i]]] = 1;
87
88                 for (p = 0; p < MAXL; p++)
89                 {
90                     cx = X[P[i]] + dirx[p];
91                     cy = Y[P[i]] + diry[p];
92
93                     if (cx > 0 && cy > 0 && cx <= N && cy <= N && B[cx][cy])
94                         G[G[ find_dad(W[cx][cy]) ]] = G[P[i]];
95
96                 }
97             }
98         }
99
100        for (; j <= Q; j++)
101            if (find_dad( W[X1[P2[j]]][Y1[P2[j]]] ) ==
102                find_dad( W[X2[P2[j]]][Y2[P2[j]]] )) Sol[P2[j]] += step;
103    }
104
105    for (i = 1; i <= Q; i++) fout << Sol[i] << endl;
106
107    return 0;
108 }

```

29.4.3 *Rezolvare detaliată

29.5 numere

Problema 5 - numere

100 de puncte

Fie un număr natural X format din maximum 20 cifre, toate nenule. Adrian dorește să construiască pe rând, în ordine crescătoare a valorii lor, toate numerele distințe care se pot forma prin schimbarea poziției cifrelor numărului X . Pentru că n este numărul său norocos, el dorește să afle al n -lea număr care se obține în acest fel.

Cerințe

Scrieți un program care determină al n -lea număr, cu numerotare de la 1, care se poate forma din cifrele lui X .

Date de intrare

Fișierul de intrare **numere.in** conține pe prima linie cele două numere naturale n și X separate printr-un singur spațiu.

Date de ieșire

Fișierul de ieșire **numere.out** va conține pe prima linie numărul natural Y , care reprezintă al n -lea număr care se poate forma cu toate cifrele numărului X . Dacă al n -lea număr generat în ordine crescătoare nu există, se va afișa -1.

Restricții și precizări

- Pentru 20% din teste $n \geq 200$ iar X are cel mult 9 cifre
- Pentru celelalte teste $200 \leq n \leq 3 * 10^{11}$

Exemple:

numere.in	numere.out	Explicații
2 8264	2486	Considerând ordinea crescătoare a valorii, primul număr care se poate forma este 2468 iar al doilea 2486
3 523525	225535	Primele trei numere care se formează sunt: 223555 225355 225535

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

Memorie: total **16 MB**

29.5.1 Indicații de rezolvare

prof. Constantin Gălățan

Fie numărul X cu nc cifre, având cifrele c_1, c_2, \dots, c_{nc} .

Să ordonăm crescător cifrele sale astfel încât $c_1 < c_2 < \dots < c_{nc}$. Frecvențele de apariție ale cifrelor în numărul X sunt f_1, f_2, \dots, f_9 .

Atunci numărul de permutări distincte ale cifrelor sale este $nr = nc! / f_1! * f_2! * \dots * f_9!$

Așadar, din numărul total al permutărilor se elimină aceleia care permutează între ele doar cifrele identice.

De exemplu: $X = 6131$. Atunci $nr = 4! / 2! * 1! * 1!$

Pentru a evita generarea efectivă a tuturor permutărilor până la întâlnirea celei de-a n -a permutări, este necesar să facem câteva observații.

Se procedează astfel:

1. Calculăm numărul p al permutărilor distincte care încep cu cea mai mică cifră c_1 :

$$p = (nc - 1)! / (f_1 - 1)! * f_2! * \dots * f_9! = nr * f_1 / nc$$

2. Dacă $p > n$, atunci permutarea căutată începe cu cifra c_1 pe care o reținem. Vom continua căutarea celei de a doua cifre a permutării eliminând cifra c_1 din număr (se decrementează f_1) și calculând numărul de permutări care încep cu c_1 , iar a doua cifră este cea mai mică dintre cele rămase.

3. Dacă $p < n$, atunci permutarea începe cu o cifră mai mare și se reia pasul 1, cumulând valorile anterior calculate pentru p .

4. Dacă $p = n$ și s-au utilizat toate cifrele numărului X , atunci se afișează cifrele reținute la pasul 2.

29.5.2 Cod sursă

Listing 29.5.1: back20p.cpp

```

1 #include <stdio.h>
2 #include <string.h>
3
4 #define LUNG_NR 21

```

```

5
6 FILE *fin = fopen("numere.in", "r"),
7     *fout = fopen("numere.out", "w");
8
9 char x[LUNG_NR];
10 int nc;
11 int fr[60];
12 double sol, n;
13 int y[21];
14 void Back(int k);
15 void DoIt();
16
17 int main()
18 {
19     fscanf(fin, "%lf%s", &n, x);
20     nc = strlen(x);
21
22     for (int i = 0; i < nc; i++)
23         fr[x[i]]++;
24     Back(1);
25 //    printf("%ld", sol);
26     if (n > sol)
27         fprintf(fout, "-1\n");
28
29     fclose(fin);
30     fclose(fout);
31     return 0;
32 }
33
34 void DoIt()
35 {
36     sol++;
37     if (sol == n)
38     {
39         for (int i = 1; i <= nc; i++)
40             fprintf(fout, "%d", y[i]);
41             fprintf(fout, "\n");
42     }
43 }
44
45 void Back(int k)
46 {
47     for (int i = 1; i <= 9; i++)
48     {
49         y[k] = i;
50         if (fr[48 + i])
51             if (k == nc)
52                 DoIt();
53             else
54             {
55                 fr[48 + i]--;
56                 Back(k + 1);
57                 fr[48 + i]++;
58             }
59     }
60 }
```

Listing 29.5.2: numere.cpp

```

1 #include <stdio.h>
2 #include <string.h>
3
4 #define LUNG_NR 21
5
6 FILE *fin = fopen("numere.in", "r"),
7     *fout = fopen("numere.out", "w");
8
9 char x[LUNG_NR];
10 char y[LUNG_NR];
11 int m, nc;
12 double fact[LUNG_NR];
13 int fr[60];
14 double nr, n;
15 int i;
16
```

```

17 int main()
18 {
19     fscanf(fin, "%lf%s", &n, x);
20     nc = strlen(x);
21
22     fact[0] = 1;
23     for( i = 1; i <= nc; i++)
24         fact[i] = fact[i-1] * i;
25
26 // printf("%lf\n", fact[nc]);
27
28     for ( i = 0; i < nc; i++)
29         fr[x[i]]++;
30
31     nr = fact[nc];
32     for(i = '1'; i <= '9'; i++)
33         nr /=fact[fr[i]];
34
35 // printf("%lf", nr);
36     if ( n > nr )
37         fprintf(fout, "-1\n");
38     else
39     {
40         m = 0;
41         while ( nc )
42         {
43             for (i = '1'; i <= '9'; i++)
44                 if ( fr[i] )
45                 {
46                     nr = nr * fr[i] / nc;
47
48                     if ( n <= nr )
49                     {
50                         y[m++] = i;
51                         fr[i]--;
52                         break;
53                     }
54                     n -= nr;
55                     nr = nr * nc / fr[i];
56                 }
57             nc--;
58         }
59         y[m] = '\0';
60         fprintf(fout, "%s\n", y);
61     }
62     fclose(fin);
63     fclose(fout);
64
65     return 0;
66 }
```

Listing 29.5.3: numere100p.cpp

```

1 // Mircea Pasoi
2 #include <stdio.h>
3 #include <string>
4
5 using namespace std;
6
7 #define MAX_X 22
8 #define FIN "numere.in"
9 #define FOUT "numere.out"
10
11 char X[MAX_X];
12 long long N, fact[MAX_X];
13
14 long long count(int cnt[])
15 {
16     int i, sum = 0;
17
18     for (i = 1; i < 10; ++i)
19         sum += cnt[i];
20     long long res = fact[sum];
21     for (i = 1; i < 10; ++i)
22         if (cnt[i]) res /= fact[cnt[i]];
```

```

23     return res;
24 }
25
26 string solve(long long n, int cnt[])
27 {
28     long long num = 0, t;
29
30     for (int i = 1; i < 10; ++i)
31     {
32         if (!cnt[i]) continue;
33         cnt[i]--;
34         t = count(cnt);
35         num += t;
36         if (num >= n)
37             return (char)(i+'0') + solve(n-num+t, cnt);
38         else
39             cnt[i]++;
40     }
41
42     return "";
43 }
44
45 int main(void)
46 {
47     int i, cnt[10] = {0};
48
49     freopen(FIN, "r", stdin);
50     freopen(FOUT, "w", stdout);
51
52     scanf("%lld %s\n", &N, X);
53
54     fact[0] = 1;
55     for (i = 1; i < MAX_X; ++i)
56         fact[i] = fact[i-1]*i;
57
58     for (i = 0; X[i]; ++i)
59         cnt[X[i]-'0']++;
60
61     printf("%s\n", solve(N, cnt).c_str());
62
63     return 0;
64 }
```

29.5.3 *Rezolvare detaliată

29.6 pikachu

Problema 6 - pikachu

100 de puncte

Miruna și partenerul ei de aventură, Pikachu, sunt în fața unei noi provocări. Cele două personaje au ajuns lângă un lanț muntos format din N vârfuri așezate în linie dreaptă unul după altul. Pentru fiecare vârf muntos se cunoaște înălțimea lui. Folosindu-se de puterile sale extraordinare, Pikachu este capabil să scadă sau să crească înălțimea unui vârf muntos cu o unitate într-o secundă. Din motive necunoscute muritorilor de rând, cei doi prieteni vor să obțină cel puțin K vârfuri montane consecutive care au aceeași înălțime, într-un timp cât mai scurt.

Cerințe

Determinați timpul minim în care Pikachu poate îndeplini această sarcină.

Date de intrare

Fișierul de intrare **pikachu.in** va conține pe prima linie două numere N și K având semnificația din enunț. Pe cea de a doua linie se vor găsi N numere naturale reprezentând înălțimile vârfurilor muntoase.

Date de ieșire

Fisierul de ieșire **pikachu.out** va conține un singur număr natural T , reprezentând timpul minim necesar pentru a obține cel puțin K vârfuri consecutive cu aceeași înălțime.

Restricții și precizări

- $1 \leq N \leq 10^5$
- $1 \leq K \leq N$
- Înălțimile munților sunt numere pozitive care se pot reprezenta pe întregi de 32 de biți cu semn
 - 20% din teste au $1 \leq N, K \leq 100$, iar înălțimile aparțin intervalului $[1, 100]$
 - Alte 20% din teste au $1 \leq N, K \leq 5000$
 - Rezultatul se poate reprezenta pe un întreg de 64 biți cu semn

Exemplu:

pikachu.in	pikachu.out	Explicații
5 3 5 2 4 3 7	2	În prima secundă se crește înălțimea muntelui de pe poziția a doua, iar în a doua secundă se scade înălțimea muntelui de pe poziția a treia. În urma acestor operații subsecvența care începe pe a doua poziție și se termină pe a patra poziție va conține doar vârfuri de înălțime 3.

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

Memorie: total **16 MB**

29.6.1 Indicații de rezolvare

Andrei Grigorean

Vom analiza cazul în care N este egal cu K .

În aceasta situație răspunsul optim se obține aducând elementele la valoarea medianei (definim *mediana unui sir* cu N elemente ca fiind al $N/2$ -lea element în ordine crescătoare).

Aceasta este o afirmație cunoscută, demonstrația poate fi găsită în cartea "Introducere în algoritm" de Cormen, Leisserson și Rivest.

Pentru cazurile în care K este mai mic decât N , observăm că nu are rost să considerăm subsecvențe de lungime mai mare decât K .

Pentru subsecvență care începe pe prima poziție putem calcula ușor mediana și costul aducerii celorlalte elemente la această valoare în complexitate $O(K \log K)$.

În continuare vom procesa următoarele subsecvențe în ordine, încercând să recalculez mediana și costul în mod eficient.

Costul cerut este egal cu

$(K/2) * \text{mediana} - \text{suma elementelor mai mici ca mediana} + \text{suma elementelor mai mari ca mediana} - (K - K/2 - 1) * \text{mediana}$.

Observăm că trebuie două subsecvențe consecutive diferă prin doar două elemente.

De aici tragem concluzia că avem nevoie de o structură de date care să suporte în mod eficient următoarele operații:

- inserarea unui element
- ștergerea unui element
- găsirea medianei

Pentru aceste operații putem folosi un *arbore de intervale* sau un *arbore echilibrat* (este suficient un *set* din *STL*). Costul se poate recalcula la fiecare pas în timp constant.

29.6.2 Cod sursă

Listing 29.6.1: pikachu.cpp

```

1 #include <cstdio>
2 #include <cstring>
3 #include <vector>
```

```

4 #include <cstdlib>
5 #include <algorithm>
6
7 using namespace std;
8
9 #define MAXN 100100
10 #define MAXARB (1 << 18)
11 #define pb push_back
12 #define mp make_pair
13 #define PII pair<int, int>
14 #define left (nod<<1)
15 #define right ((nod<<1) | 1)
16 #define mid ((st+dr) >> 1)
17 #define llong long long
18
19 int N, K, sir[MAXN], arb[MAXARB], med[MAXN];
20 vector<int> A;
21 llong res;
22 llong arb2[MAXARB];
23
24 int getindex(int x)
25 {
26     return (int)(lower_bound(A.begin(), A.end(), x)-A.begin());
27 }
28
29 int query(int nod, int st, int dr, int k)
30 {
31     if(st == dr)
32         return st;
33     if(arb[left] >= k) return query(left, st, mid, k);
34     return query(right, mid+1, dr, k-arb[left]);
35 }
36
37 void update(int nod, int st, int dr, int k, int sgn)
38 {
39     arb[nod] += sgn;
40     if(st == dr)
41         return;
42     if(k <= mid) update(left, st, mid, k, sgn);
43     else update(right, mid+1, dr, k, sgn);
44 }
45
46 void update2(int nod, int st, int dr, int k, int sgn)
47 {
48     arb[nod] += sgn;
49     arb2[nod] += (llong)sgn*A[k];
50     if(st == dr) return;
51     if(k <= mid) update2(left, st, mid, k, sgn);
52     else update2(right, mid+1, dr, k, sgn);
53 }
54
55 llong sum(int nod, int st, int dr, int k)
56 {
57     if(st == dr) return arb2[nod];
58     if(arb[left] >= k) return sum(left, st, mid, k);
59     return arb2[left]+sum(right, mid+1, dr, k-arb[left]);
60 }
61
62 void solve(void)
63 {
64     int i, j, k, s, lamisto;
65     llong aux, aux2;
66     sort(A.begin(), A.end());
67     for(i = 0; i < K; i++)
68         update(1, 0, N-1, getindex(sir[i]), +1);
69     if(K%2 == 1) s = 1;
70     else s = 0;
71     med[0] = A[query(1, 0, N-1, K/2+s)];
72     for(i = 1; i+K-1 < N; i++)
73     {
74         int ttt = getindex(sir[i-1]);
75         int qqq = getindex(sir[i+K-1]);
76         update(1, 0, N-1, getindex(sir[i-1]), -1);
77         update(1, 0, N-1, getindex(sir[i+K-1]), +1);
78         med[i] = A[query(1, 0, N-1, K/2+s)];
79     }

```

```

80     for(i = 0; i < MAXARB; i++) arb[i] = 0;
81     for(i = 0; i < K; i++)
82         update2(1, 0, N-1, getindex(sir[i]), +1);
83     if(K%2 == 0) lamisto = 1;
84     else lamisto = 0;
85     aux = sum(1, 0, N-1, K/2-lamisto), aux2 = sum(1, 0, N-1, K)-med[0];
86     res = (llong)med[0]*(K/2-lamisto)-aux;
87     res += (llong)(aux2-aux)-med[0]*(llong)(K/2);
88     for(i = 1; i+K-1 < N; i++)
89     {
90         update2(1, 0, N-1, getindex(sir[i-1]), -1);
91         update2(1, 0, N-1, getindex(sir[i+K-1]), +1);
92         aux = sum(1, 0, N-1, K/2-lamisto), aux2 = sum(1, 0, N-1, K)-med[i];
93         llong q;
94         q = (llong)med[i]*(K/2-lamisto)-aux;
95         q += (llong)(aux2-aux)-med[i]*(llong)(K/2);
96         res = min(res, q);
97     }
98 }
99
100 int main(void)
101 {
102     freopen("pikachu.in", "rt", stdin);
103     freopen("pikachu.out", "wt", stdout);
104
105     int i;
106     scanf("%d %d", &N, &K);
107
108     for(i = 0; i < N; i++)
109         scanf("%d", &sir[i]), A.pb(sir[i]);
110
111     solve();
112
113     printf("%lld\n", res);
114
115     return 0;
116 }
```

Listing 29.6.2: pikachu2.cpp

```

1 #include <iostream>
2 #include <algorithm>
3
4 using namespace std;
5
6 #define MAXN 100010
7 #define MAXX 262144
8 #define ll long long
9 #define INF 10000000000000000LL
10
11 int N, K, L;
12 int X, Y, V, Value, Med;
13 int A[MAXN], P[MAXN], W[MAXN];
14 int C[MAXX];
15 ll Sum[MAXX];
16 ll Soi;
17
18 inline int cmp(int x, int y)
19 {
20     return A[x] < A[y];
21 }
22
23 void update(int p, int r, int nod)
24 {
25     if (p == r)
26     {
27         C[nod] = V;
28         Sum[nod] = Value;
29     }
30     else
31     {
32         int q = (p+r) / 2;
33
34         if (X <= q) update(p, q, nod*2);
35         else update(q+1, r, nod*2+1);
36     }
37 }
```

```

36         Sum[nod] = Sum[nod*2] + Sum[nod*2+1];
37         C[nod] = C[nod*2] + C[nod*2+1];
38     }
39 }
40 }
41
42 int locate(int p, int r, int nod)
43 {
44     if (p == r) return p;
45
46     int q = (p+r) / 2;
47
48     if (V <= C[nod*2]) return locate(p, q, nod*2);
49
50     V -= C[nod*2];
51     return locate(q+1, r, nod*2+1);
52 }
53
54 ll query(int p, int r, int nod)
55 {
56     if (X <= p && r <= Y) return Sum[nod];
57     else
58     {
59         int q = (p+r) / 2;
60         ll rez = 0;
61
62         if (X <= q) rez = query(p, q, nod*2);
63         if (q+1 <= Y) rez += query(q+1, r, nod*2+1);
64
65         return rez;
66     }
67 }
68
69 int main()
70 {
71     ifstream fin("pikachu.in");
72     ofstream fout("pikachu.out");
73
74     int i;
75     ll S;
76
77     fin >> N >> K;
78
79     for (i = 1; i <= N; i++)
80     {
81         fin >> A[i];
82         P[i] = i;
83     }
84
85     sort(P+1, P+N+1, cmp);
86
87     for (i = 1; i <= N; i++) W[P[i]] = i;
88
89     for (L = 1; L < N; L <= 1);
90
91     Sol = INF;
92
93     for (i = 1; i <= N; i++)
94     {
95         X = W[i];
96         V = 1, Value = A[i];
97         update(1, L, 1);
98
99         if (i - K > 0)
100        {
101            X = W[i-K];
102            V = 0, Value = 0;
103            update(1, L, 1);
104        }
105
106        if (i - K >= 0)
107        {
108            V = K/2 + 1;
109            Med = locate(1, L, 1);
110
111            X = 1, Y = Med - 1;

```

```

112         S = (1LL * A[P[Med]] * (K/2)) - query(1, L, 1);
113
114         X = Med + 1, Y = N;
115         S += query(1, L, 1) - (1LL * A[P[Med]]*(K-(K/2+1)));
116
117         Sol = min(Sol, S);
118     }
119 }
120
121     fout << Sol << endl;
122
123     return 0;
124 }
```

Listing 29.6.3: pikachu3.cpp

```

1 //Code by Patcas Csaba
2 #include <stdio.h>
3 #include <vector>
4 #include <set>
5 #include <algorithm>
6 #include <iostream>
7
8 using namespace std;
9
10#define in_file "pikachu.in"
11#define out_file "pikachu.out"
12
13#define VI vector <int>
14#define FORN(i,n) for(int i=0;i<n;++i)
15#define FOR(i,a,b) for(int i=a;i<=b;++i)
16#define S size()
17#define X first
18#define Y second
19#define PB push_back
20#define MP make_pair
21#define ALL(x) x.begin(),x.end()
22#define B begin()
23#define E end()
24
25 int n,k;
26 VI a;
27 long long solution;
28
29 void solve()
30 {
31     set<int> active;
32     VI aux;
33     FOR(i,1,k)
34     {
35         aux.PB(a[i]);
36         active.insert(a[i]);
37     }
38
39     nth_element(aux.B,aux.B+(k+1)/2-1,aux.E);
40     int median=aux[(k+1)/2-1];
41     long long sum=0;
42     FOR(i,1,k) sum+=abs(median-a[i]);
43     solution=sum;
44     FOR(i,k+1,n)
45     {
46         sum-=abs(median-a[i-k]);
47         set<int>::iterator it=active.find(median);
48         --it;
49         int before=*it;
50         ++it;
51         int after=*it;
52
53         int oldmedian=median;
54
55         active.erase(active.find(a[i-k]));
56         active.insert(a[i]);
57
58         if (a[i-k]==median)
59             if (a[i]>=before && a[i]<=after) median=a[i];
```

```

60         else
61             if (a[i]<before) median=before;
62             else median=after;
63         if (a[i-k]==before)
64             if (a[i]>=median && a[i]<=after) median=a[i];
65             else
66                 if (a[i]<median) ;
67                 else median=after;
68         if (a[i-k]==after)
69             if (a[i]>=before && a[i]<=median) median=a[i];
70             else
71                 if (a[i]<before) median=before;
72                 else ;
73         if (a[i-k]<before)
74             if (a[i]>after) median=after;
75             else if (a[i]>median) median=a[i];
76         if (a[i-k]>after)
77             if (a[i]<before) median=before;
78             else if (a[i]<median) median=a[i];
79             sum+=abs(median-a[i]);
80         if (!(k&1)) sum+=oldmedian-median;
81
82     solution=min(solution,sum);
83 }
85
86 void solve2()
87 {
88     if (n==1) solution=0;
89     else
90     {
91         solution=abs(a[1]-a[2]);
92         FOR(i,3,n)
93             solution=min(solution,(long long) (abs(a[i]-a[i-1])));
94     }
95 }
96
97 int main()
98 {
99     FILE* fin=fopen(in_file,"r");
100    fscanf(fin,"%d%d",&n,&k);
101    a.resize(n+1);
102    FOR(i,1,n) fscanf(fin,"%d",&a[i]);
103    fclose(fin);
104
105    if (k==1) solution=0;
106    else if (k==2) solve2();
107    else solve();
108
109    FILE* fout=fopen(out_file,"w");
110    fprintf(fout,"%lld",solution);
111    fclose(fout);
112
113    return 0;
114 }
```

Listing 29.6.4: pikachu4.cpp

```

1 /*
2 Autor: Catalin Stefan Tiseanu
3 Punctaj:
4 Complexitate: O(N * K)
5 */
6
7 #include <cstdio>
8 #include <algorithm>
9 #include <vector>
10 #include <iostream>
11 #include <cmath>
12
13 using namespace std;
14
15 #define MAX_N (1<<17)
16
17 int N, K, A[MAX_N], buf[MAX_N];
```

```
18
19 int main()
20 {
21     freopen("pikachu.in", "r", stdin);
22     freopen("pikachu.out", "w", stdout);
23
24     scanf("%d %d", &N, &K);
25     for ( int i = 1; i <= N; ++i )
26         scanf("%d", A + i );
27
28     long long best_cost = (1LL<<62);
29
30     for ( int i = K; i <= N; ++i )
31     {
32         long long cur_cost = 0, med;
33         int left = i - K + 1;
34
35         for ( int j = left; j <= i; ++j )
36             buf[ j - left + 1 ] = A[j];
37
38         nth_element(buf + 1, buf + (K+1)/2, buf + K + 1 );
39         med = buf[(K+1)/2];
40
41         for ( int j = i - K + 1; j <= i; ++j )
42             cur_cost += abs( A[j] - med );
43
44         if( cur_cost < best_cost )
45             best_cost = cur_cost;
46     }
47
48     cout << best_cost << endl;
49
50     return 0;
51 }
```

29.6.3 *Rezolvare detaliată

Capitolul 30

ONI 2008

30.1 albinuta

Problema 1 - albinuta

100 de puncte

Albinuța D are N flori preferate și un mod aparte de a le culege polenul. Aceasta și-a întocmit o hartă a florilor sub forma unui graf orientat cu N noduri și M muchii, în care florile sunt nodurile grafului și sunt numerotate cu $1, 2, \dots, N$. Pentru fiecare nod se cunoaște lista de adiacență, ordonată conform preferințelor albinuței.

Traseul parcurs de albinuță pentru culegerea polenului, pornește din nodul cu numărul 1, la momentul de timp $T = 1$. La fiecare moment de timp T albinuța alege al T -lea nod din lista de adiacență, numărând circular T poziții, începând cu primul nod din lista. Ea va ajunge la momentul $T + 1$ în nodul astfel ales. De exemplu, dacă la momentul de timp $T = 12$ lista de adiacență a nodului curent, ordonată conform preferințelor albinuței, este: 1 6 2 9 5, atunci la momentul $T = 13$ albinuța va ajunge în nodul 6.

Un apicoltor a descoperit harta folosită de albinuță și se întreabă în ce floare se va afla aceasta la anumite momente de timp T_i ($1 \leq i \leq Q$)

Cerințe

Scrieți un program care să determine pentru fiecare moment de timp T_i dat, floarea pe care se va afla albinuța.

Date de intrare

Fișierul de intrare **albinuta.in** va conține pe prima linie două numere naturale naturale N și Q , separate printr-un singur spațiu. Fiecare linie k , dintre următoarele N , conține numărul natural L_k , reprezentând numărul de noduri adiacente cu nodul k , urmat de L_k numere naturale reprezentând *lista de adiacență* a acestuia, ordonată conform preferințelor albinuței. Numerele conținute de cele N linii sunt separate prin câte un spațiu. Următoarele Q linii vor conține, în ordine, numerele T_1, T_2, \dots, T_Q , câte unul pe linie.

Date de ieșire

Fișierul de ieșire **albinuta.out** va conține Q linii.

Pe linia i se va scrie numărul nodului în care se va afla albinuța la momentul de timp T_i .

Restricții și precizări

- $M = L_1 + L_2 + \dots + L_N$
- $1 \leq N, M, Q \leq 50$
- $1 \leq L_k \leq M$, (\forall) $1 \leq k \leq N$
- $1 \leq T_i \leq 10^9$, (\forall) $1 \leq i \leq Q$
- Într-o listă de adiacență, un nod poate apărea de mai multe ori.
- Un nod poate apărea în lista lui de adiacență
- Pentru 30% din teste, $1 \leq T_i \leq 10^6$

Exemple:

albinuta.in	albinuta.out	Explicații
6 5	1	Albinuța va urma traseul: 1 2 3 6 1 la momentele de timp 1, 2, 3, 4, 5.
2 2 1	2	
2 1 3	3	
3 4 5 6	6	
1 5	1	
1 6		
1 1		
1		
2		
3		
4		
5		

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

30.1.1 Indicații de rezolvare

Emilian Miron

Putem caracteriza starea albinuței ca un tuplu (t, n) cu semnificația: la timpul t albinuța se află în nodul n . Notând cu V_k vectorul de adiacență al nodului k ($V_{k,i}$ - al i -lea vecin al nodului k , $1 \leq i \leq L_k$) obținem următoarea regulă de tranziție:

Pentru (t, n) următorul nod ales va avea poziția $1 + (t - 1)\%L_n$:

$$(t, n) \rightarrow (t + 1, V_{n,(1+(t-1)\%L_n)})$$

Observăm din relația de mai sus că pentru a afla succesorul nodului n la un moment t nu contează valoarea exactă a lui t ci doar restul împărțirii acestuia la L_n sau la orice multiplu de L_n . Restul împărțirii lui t la $cmmmc(L_k)$ este deci suficient pentru a caracteriza starea albinuței.

CMMMC-ul maxim pentru datele de intrare este $MAX = 4*9*5*7*11*13$, deci spațiul stărilor este o mulțime finită de maxim $N * MAX$ elemente. De la un anumit timp starea albinuței va cicla.

Graful stărilor va fi similar cu cel din dreapta, format dintr-o *coadă* de X elemente urmată de un ciclu de lungime Y .

Dacă $T > X + Y$ putem reduce T la $X + 1 + (T - X - 1)\%Y$.

Memoria disponibilă este suficientă pentru a păstra un vector de marcare pentru stări care ne permite să determinăm X și Y , iar pentru fiecare T_k putem să recalculem drumul albinuței până la T_k redus la maxim $X + Y$.

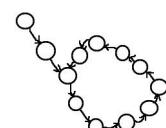
Complexitatea soluției este $O(N * MAX + Q * N * MAX)$, cu memorie $O(N * MAX)$.

Se poate reține și vectorul de noduri parcuse până la $X + Y$, răspunzând apoi la întrebări în $O(1)$ pentru fiecare T , însă trebuie avut *grijă la limita de memorie*.

O soluție alternativă este presupunerea că traseul albinuței va cicla. De la un moment de timp i (egal cu X din soluția de mai sus) sirul T_i, T_{i+1}, \dots va fi un *sir periodic*. Se încearcă succesiv valorile i până când se găsește un sir periodic. Se poate determina în timp liniar dacă un sir este periodic.

Este posibilă rezolvarea cu $O(1)$ memorie adițională cu ajutorul unui *algoritm de detectie a ciclurilor* similar cu ideea de la *algoritmul de factorizare Pollard-Rho*.

http://en.wikipedia.org/wiki/Cycle_detection



30.1.2 Cod sursă

https://www.infoarena.ro/job_detail/2596063?action=view-source

Listing 30.1.1: albinuta.cpp

```

1 // https://www.infoarena.ro/job_detail/2596063?action=view-source
2 // Marius Andronie - malakay
3
4 #include <iostream>
5 #include <fstream>
```

```

6 #include <vector>
7 #include <algorithm>
8
9 using namespace std;
10
11 ifstream f("albinuta.in");
12 ofstream g("albinuta.out");
13
14 int n,q,L=1,viz[51][180180],l,r;
15
16 vector<int>e[51],ee;
17
18 void qry()
19 {
20     int in;
21     f>>in;
22
23     if(in<=r) g<<ee[in]<<"\n";
24     else
25     {
26         g<<ee[l+(in-1)% (r-l+1)]<<"\n";
27     }
28 }
29
30 int main()
31 {
32     f>>n>>q;
33
34     for(int i=1,k;i<=n;i++)
35     {
36         f>>k;
37         for(int j=1,a;j<=k;++j)
38         {
39             f>>a;
40             e[i].push_back(a);
41         }
42
43         L=(L*k)/__gcd(L,k);
44     }
45
46     int t=1,nod=1;
47
48     ee.push_back(0);
49
50     while(1)
51     {
52         ee.push_back(nod);
53         if(viz[nod][t%L]>0)
54         {
55             l=viz[nod][t%L];
56             r=t;
57             break;
58         }
59
60         viz[nod][t%L]=t;
61         nod=e[nod][(t-1)%e[nod].size()];
62         ++t;
63     }
64
65     ++l;
66     while(q--)
67     {
68         qry();
69     }
70
71     return 0;
72 }
```

30.1.3 *Rezolvare detaliată

30.2 curent

Problema 2 - curent

100 de puncte

Satul Mirşid este format din N case numerotate cu numerele naturale de la 1 la N . În casa numerotată cu 1 se află un generator care alimentează cu curent electric toate casele din sat astfel: casa cu numărul 1 alimentează cu curent un număr de case, fiecare dintre aceste case alimentând la rândul lor alte case, și.a.m.d. Fiecare casă este alimentată de exact o sursă de curent (care poate fi o altă casă alimentată cu curent electric sau generatorul, în cazul casei cu numărul 1).

În fiecare casă există o singură siguranță electrică. În cazul unei defecțiuni la un aparat electric dintr-o casă, siguranța casei se arde. În acest caz, alimentarea cu curent electric a casei se întrerupe, determinând întreruperea curentului în toate casele alimentate de la aceasta, direct sau indirect.

Compania producătoare de siguranțe are un jurnal în care a fost notată fiecare zi în care s-a ars, respectiv s-a înlocuit, o siguranță, împreună cu numărul casei în care s-a petrecut evenimentul. Jurnalul conține în total M evenimente. Pentru a stabili gradul de mulțumire al clientilor, compania vrea să afle numărul total de case care aveau curent electric în anumite zile de interes.

Cerințe

Să se scrie un program care să determine numărul de case care aveau curent electric pentru fiecare dintre zilele de interes stabilite de conducerea companiei.

Date de intrare

Fișierul de intrare **curent.in** are pe prima linie numărul natural N .

A doua linie, conține $N - 1$ numere naturale nenule cel mult egale cu N , separate prin câte un singur spațiu, cel de al k -lea număr ($1 \leq k < N$) reprezentând numărul casei de la care primește curent casa cu numărul $k + 1$.

A treia linie conține un număr natural M reprezentând numărul de evenimente notate în jurnalul companiei.

Fiecare dintre următoarele M linii conțin, câte 3 numere naturale a b c , separate prin câte un spațiu, care descriu astfel un eveniment: dacă c este 0 atunci în ziua a $s-a$ ars siguranța în casa b , dacă c este 1 atunci în ziua a $s-a$ înlocuit siguranța din casa b .

Linia $M + 4$ a fișierului conține un număr natural T reprezentând numărul zilelor pentru care trebuie determinat numărul caselor ce au curent electric, iar linia $M + 5$ a fișierului conține un sir strict crescător de T numere naturale, separate prin câte un spațiu, reprezentând zilele de interes stabilite de conducerea companiei. Zilele sunt numerotate de la 1 la 10^9 .

Date de ieșire

Fișierul de ieșire **curent.out** va contine T linii, fiecare linie k ($1 \leq k \leq T$) va conține un număr natural reprezentând numărul de case care aveau curent electric în a k -a zi de interes (în ordinea din fișierul de intrare).

Restricții și precizări

- $0 < N < 100011$; $0 < M < 100022$; $0 < T < 100033$
- dacă a b c și a' b' c' sunt valorile care descriu, în fișierul de intrare, două evenimente, iar $a = a'$ atunci $b \neq b'$

Exemplu:

curent.in	curent.out	Explicații
11	5	În ziua 2, siguranța din casa 3 se arde, casele 3, 4, 5, 6, 7, 8 nemaifiind alimentate cu curent electric. Siguranța nu a fost înlocuită până în ziua 8, rămânând doar 5 case alimentate (1, 2, 9, 10, 11), valoarea 5 scriindu-se pe prima linie a fișierului curent.out . În zilele 9 și 10 se ard siguranțele din casele 9 și 10. Astfel, în ziua 13 doar 2 case (1 și 2) sunt alimentate cu curent, valoarea 2 scriindu-se pe a doua linie a fișierului.
1 2 3 3 3 5 5	2	În ziua 14 se arde siguranța din casa 5 fapt care nu modifică numărul caselor alimentate cu curent. În ziua 22 se repară siguranța în casa 3, în ziua 23 fiind 5 case alimentate cu curent (1,2,3,4,6), valoarea 5 scriindu-se pe a treia linie a fișierului.
2 1 10	5	În ziua 31 se arde siguranța din casa 1, astfel că în ziua 33 nicio casă nu mai este alimentată cu curent, valoarea 0 scriindu-se pe cea de-a patra linie a fișierului.
6	0	
9 9 0		
2 3 0		
10 10 0		
22 3 1		
14 5 0		
31 1 0		
4		
8 13 23 33		

Timp maxim de executare/test: **0.7** secunde

Memorie: total **16 MB** din care pentru stivă **4 MB**

30.2.1 Indicații de rezolvare

Ionuț Fechete

Problema se poate rezolva făcând o *parcursere în adâncime* pentru fiecare zi de interes pentru a afla numărul de noduri accesibile din nodul 1 la acel moment. Astfel se răspunde la o întrebare în complexitatea $O(N)$ și actualizarea se face în $O(1)$ marcând nodul ca fiind cu siguranță arsă. O astfel de rezolvare obține 25-30 de puncte.

O altă abordare presupune stocarea în memorie a numărului de noduri accesibile pentru fiecare nod al arborelui; inițial acest număr este egal cu numărul de noduri din subarborele a căruia rădăcina este nodul. Pentru a se actualiza acest vector trebuie parcurs arborele plecând de la nodul care trebuie actualizat, mergând din tată în tată până se ajunge la un nod cu siguranță arsă, deci complexitatea este $O(N)$ în cel mai rau caz, iar de răspuns se răspunde în $O(1)$. Această rezolvare obține 35-40 de puncte.

Soluția oficială folosește *arbori de intervale și parcurserea Euler a arborelui* pentru a face o actualizare în $O(\log N)$ și pentru a răspunde în $O(1)$.

Folosind parcurserea Euler a arborelui obținem nodurile într-o anumită ordine astfel încât toate nodurile dintr-un anumit subarbore se află într-un interval compact din secvență.

O parcursere Euler în care scriem un nod doar o dată pentru arborele din figură generează secvența:

1 2 3 4 5 7 8 6 9 10 11

Subsecvența determinată de pozițiile a 2-a și a 9-a din această secvență conține toate nodurile din subarborele al căruia nod radăcină este 2.

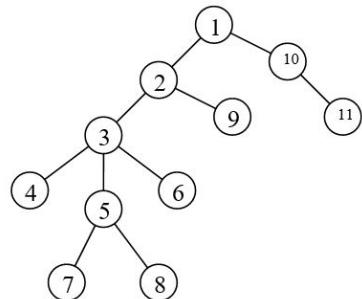
Subsecvența determinată de pozițiile a 5-a și a 7-a din această secvență conține toate nodurile din subarborele al căruia nod radăcină este 5.

O altă proprietate a acestei secvențe este că oricare ar fi două intervalele pentru care se fac actualizări acestea fie sunt disjuncte, fie unul dintre ele este conținut de celălalt.

După ce am obținut această secvență putem folosi *arbori de intervale* pentru a rezolva problema în felul următor: pentru fiecare interval reținem numărul de noduri accesibile din nodul rădăcina corespunzător intervalului $S[nod]$ (după cum am precizat mai sus un interval din secvență este echivalent cu un subarbore din arborele dat la intrare) și numărul de strămoși cu siguranțele arse ale nodului $Arsuri[nod]$.

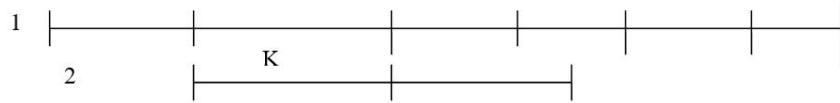
Initial toate nodurile au siguranțele bune deci pentru fiecare interval numărul de noduri accesibile este egal cu lungimea intervalului.

Pentru o actualizare mergem inițial în jos în arborele de intervale ca la o actualizare normală, obținem o împărțire a intervalului ce trebuie inițializat în k intervale $A_1 A_2 A_3 \dots A_K$,



pentru fiecare din aceste intervale incrementăm $Arsuri[A_i]$ dacă actualizarea este o arsură sau decrementăm dacă actualizarea este o înlocuire.

Apoi mergem în sus pe arbore acutalizând $S[nod]$ pentru nodurile strămoși ca la o actualizare normală doar că $S[nod] = S[left] + S[right]$ numai atunci când $Arsuri[nod]$ este 0.



De exemplu dacă avem de actualizat intervalul de sus (1) vom update $Arsuri[x]$ pentru fiecare interval din împărțirea făcută de arborele de intervale (pentru a evita actualizarea pentru fiecare nod). Când vom actualiza intervalul de jos (2), intervalul $Arsuri[K]$ va fi incrementat a două oară, nodurile din acel interval având doi strămoși cu siguranțele arse.

Pentru a răspunde la o întrebare folosim $S[1]$, numărul de noduri accesibile din 1.

O descriere a structurii de date "arbore de intervale" poate fi gasită la următorul link:

<http://infoarena.ro/arbori-de-intervale>

iar o descriere a parcurgerii Euler a unui arbore aici:

http://www.ginfo.ro/revista/11_4/probleme.pdf

30.2.2 Cod sursă

https://www.infoarena.ro/job_detail/2601471?action=view-source

Listing 30.2.1: curent.cpp

```

1 // https://www.infoarena.ro/job_detail/2601471?action=view-source
2 // Lazar Laurentiu - Lazar_Laurentiu
3 #include <algorithm>
4 #include <iostream>
5 #include <fstream>
6 #include <vector>
7
8 #define MAX 100060
9 #define AMAX 300000
10
11 using namespace std;
12
13 struct operatie
14 {
15     int ta,na,op;
16
17 };
18
19 int n,p,m,sz,t,ta,na,op;
20
21 operatie opr[MAX];
22
23 int acc[AMAX],ars[AMAX];
24 int vn[MAX],n1[MAX],n2[MAX],ans[MAX];
25
26 vector<int> nd[MAX];
27
28 int rez(int nod)
29 {
30     vn[++sz]=nod;
31     n1[nod]=sz;
32     int nmax=sz;
33
34     for(auto i:nd[nod])
35         nmax=max(nmax,rez(i));
36
37     n2[nod]=nmax;
38     return nmax;
39 }
40
41 void init(int st,int dr,int ind)
42 {
43     acc[ind]=dr-st+1;
44     if(st==dr) return;

```

```

45     int mij=(st+dr)/2;
46     init(st,mij,ind*2);
47     init(mij+1,dr,ind*2+1);
48 }
49
50 bool qin(int qst,int qdr,int st,int dr)
51 {
52     return (st<=qst&&qdr<=dr);
53 }
54
55 bool qout(int qst,int qdr,int st,int dr)
56 {
57     return (qdr<st||qst>dr);
58 }
59
60 //int indmax;
61
62 void actualizare(int st,int dr,int ind,int qst,int qdr,int qac)
63 {
64 //  if(ind>=AMAX) cout<<"u fucced up\n";
65 //  indmax=max(indmax,ind);
66     if(qout(st,dr,qst,qdr)) return;
67     if(qin(st,dr,qst,qdr))
68     {
69         ars[ind]+=qac;
70         if(ars[ind]==0)
71         {
72             if(st==dr) acc[ind]=1;
73             else acc[ind]=acc[ind*2]+acc[ind*2+1];
74         }
75         else acc[ind]=0;
76     }
77     return;
78 }
79
80     int mij=(st+dr)/2;
81
82     actualizare(st,mij,ind*2      ,qst,qdr,qac);
83     actualizare(mij+1,dr,ind*2+1,qst,qdr,qac);
84     acc[ind]=(acc[ind*2]+acc[ind*2+1])* (ars[ind]==0);
85 }
86
87 bool cmp(operatie o1,operatie o2)
88 {
89     return o1.ta<o2.ta;
90 }
91
92 int main()
93 {
94     ifstream f ("curent.in");
95     ofstream g ("curent.out");
96
97     f>>n;
98     for(int i=2;i<=n;i++)
99         f>>p,
100         nd[p].push_back(i);
101
102     rez(1);
103
104     init(1,n,1);
105
106     f>>m;
107     for(int i=1;i<=m;i++)
108     {
109         f>>ta>>na>>op;
110         if(op==0) op--;
111         opr[i]={ta,na,op};
112     }
113
114     sort(opr+1,opr+m+1,cmp);
115
116     for(int i=1;i<=m;i++)
117     {
118         actualizare(1,n,1,n1[opr[i].na],n2[opr[i].na],opr[i].op);
119         //   cout<<opr[i].ta<<": "<<opr[i].na<<" se "<<
120         //           ((opr[i].op===-1)??"stinge":?"aprinde")<<'\\n';

```

```

121      // cout<<"raman "<<acc[1]<<"\n\n";
122
123     ans[i]=acc[1];
124 }
125
126 ans[0]=n; ans[m+1]=ans[m];
127
128 f>>t;
129
130 // cout<<l;
131 while(t--)
132 {
133     f>>ta;
134     int st=0, dr=m, mij;
135     while(st<dr)
136     {
137         // cout<<"ffs\n";
138         int mij=(st+dr+1)/2;
139         if(opr[mij].ta<=ta) st=mij;
140         else dr=mij-1;
141     }
142
143     g<<ans[st]<<' \n';
144 }
145
146 // cout<<indmax<<' \n';
147 f.close ();
148 g.close ();
149
150 return 0;
151 }
```

30.2.3 *Rezolvare detaliată

30.3 pviz

Problema 3 - pviz

100 de puncte

Fie N un număr natural nenul și P o permutare de lungime N a numerelor din multimea $\{1, 2, \dots, N\}$. Definim un element *vizibil* în permutarea P ca fiind un număr P_i care are proprietatea că $P_j < P_i$, $\forall 1 \leq j < i$ sau $i = 1$.

Cerințe

Determinați numărul X de permutări de lungime N care au ca elemente vizibile exact M elemente date.

Date de intrare

Fișierul de intrare **pviz.in** conține pe prima linie două numere naturale N și M , cu semnificația din enunț, separate printr-un spațiu. A doua linie a fișierului conține M numere naturale distințe, ordonate crescător, separate prin câte un spațiu, reprezentând elementele vizibile.

Date de ieșire

Fișierul de ieșire **pviz.out** va conține o singură linie pe care va fi scris un număr natural reprezentând restul împărțirii numărului X la 10007.

Restricții și precizări

- $1 \leq N \leq 2\ 000$
- $1 \leq M \leq N$
- Elementele vizibile sunt scrise în fișierul de intrare în ordine crescătoare.
- Pentru 10% din teste $N \leq 10$
- Pentru 20% din teste $N \leq 14$
- Pentru 60% din teste $N \leq 375$

Exemplu:

pviz.in	pviz.out	Explicații
4 2 2 4	3	Sunt 3 permutări, de lungime 4, care au pe 2 și 4 ca elemente vizibile: 2 4 3 1 2 4 1 3 2 1 4 3 Permutarea 2 3 4 1 nu corespunde cerinței deoarece are ca elemente vizibile atât pe 2 și 4 cât și pe 3.

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

Memorie: total **64 MB** din care pentru stivă **1 MB**

30.3.1 Indicații de rezolvare

Adrian Diaconu

Soluție 1 (Adrian Diaconu)

Problema se poate rezolva folosind *metoda programării dinamice*. Se va construi o matrice H în care $H[i][j] =$ numărul permutărilor de lungime i care conțin primele j elemente vizibile.

Relația de recurență este:

$$H[i][j] = H[i - 1][j - 1] + (X[j] - i + 1) * H[i - 1][j]$$

unde $X[i]$ reprezintă al i -lea element vizibil. Primul termen reprezintă cazul în care pe poziția i se pune cel de-al j -lea element vizibil. Al doilea termen reprezintă faptul că pe poziția i se plasează un element ce nu este vizibil, acesta putând fi ales din numerele mai mici decât $X[j]$, care nu au fost puse încă în permutare. Rezultatul final este memorat în $H[n][m]$.

Complexitate $O(n^2)$. Memorie $O(n)$ deoarece sunt necesare doar ultimele două linii ale matricii.

La aceeași soluție se poate ajunge și pornind de la o recurență de forma:

$$H[i][j] = \sum_{k=0}^i H[k][j - 1] * A_{X[j-1]-k}^{i-k}$$

care poate fi prelucrată pentru a se ajunge la recurența prezentată anterior.

O soluție în $O(n^3)$ ar fi obținut aproximativ 60 de puncte.

Soluție 2 (Csaba Pătcăș)

Se poate aplica *metoda programării dinamice* și în felul urmator:

$H[i][j] =$ numărul de permutări de lungime j , în care pe ultima poziție avem fixat elementul i din lista elementelor vizibile pe poziția j , și primele $i - 1$ sunt deja vizibile.

Aici, formula de recurență pentru soluția în $O(n^3)$ se calculează fixând al $i - 1$ -lea element vizibil pe poziția k și considerând toate elementele care pot apărea între poziția k și poziția j , obținându-se

$$H[i][j] = \sum_{k=1}^i H[i - 1][k] * A_{X[i-1]-k}^{j-k-1}$$

Această soluție se poate reduce la complexitatea $O(n^2)$ prin aplicarea recurenței:

$$H[i][j] = H[i][j - 1] * (X[i - 1] - j + 2) + H[i - 1][j - 1]$$

Dacă folosim această recurență, nu trebuie să uităm de posibilitățile de a alege elementele după ultima poziție vizibilă.

Așadar soluția finală se obține din suma:

$$\sum_{i=1}^n H[m][i] * A_{X[m]-i}^{n-i}$$

30.3.2 Cod sursă

https://www.infoarena.ro/job_detail/2602605?action=view-source

Listing 30.3.1: pviz.cpp

```

1 // https://www.infoarena.ro/job_detail/2602605?action=view-source
2 // Lazar Laurentiu - Lazar_Laurentiu
3 #include <iostream>
4 #include <fstream>
5
6 #define MAX 2010
7 #define MOD 10007
8
9 using namespace std;
10
11 int n,m,ansf;
12 int a[MAX],ans[MAX][MAX],fact[MAX];
13
14 int pw(int nr,int expn)
15 {
16
17     if(expn==0) return 1;
18     if(expn==1) return nr%MOD;
19     if(expn%2)
20     {
21         int aux=pw(nr,expn-1);
22         return (aux*nr)%MOD;
23     }
24
25     int aux=pw(nr,expn/2);
26     return (aux*aux)%MOD;
27 }
28
29 int aranj(int na,int ka)
30 {
31     return fact[na]*pw(fact[na-ka],MOD-2)%MOD;
32 }
33
34 int main()
35 {
36     ifstream f ("pviz.in");
37     ofstream g ("pviz.out");
38
39     f>>n>>m;
40     for(int i=1;i<=m;i++) f>>a[i];
41
42     if(a[m]!=n) g<<"0\n";
43     else
44     {
45         ans[1][1]=1;
46         for(int i=2;i<=m;i++)
47             for(int j=i;j<=a[i];j++)
48                 ans[i][j]=ans[i-1][j-1]+ans[i][j-1]*(a[i-1]-(j-1)+1),
49                 ans[i][j]%=MOD;
50
51         fact[0]=1;
52         for(int i=1;i<=n;i++)
53             fact[i]=(fact[i-1]*i)%MOD;
54
55         for(int i=m;i<=n;i++)
56             ansf=ansf+ans[m][i]*fact[n-i],
57             ansf%=MOD;
58
59         g<<ansf<<'\n';
60     }
61
62     f.close ();
63     g.close ();
64
65     return 0;
66 }
```

30.3.3 *Rezolvare detaliată

30.4 atac

Problema 4 - atac

100 de puncte

Costel, mare pasionat de jocuri de strategie, a descoperit de curând un joc nou. Acesta se joacă pe o hartă compusă din n orașe, numerotate de la 1 la n , unele dintre acestea fiind conectate prin drumuri directe. Este posibilă deplasarea între oricare două orașe utilizând drumurile existente. Costel a ajuns înaintea bătăliei finale, când adversarul său mai stăpânește un singur oraș. Notăm acest oraș cu x . Costel dispune de u unități de armată, fiecare fiind cantonată într-un oraș oarecare, diferit de orașul x . Pentru a câștiga, Costel trebuie să depleteze căte o unitate de armată în fiecare dintre orașele învecinate cu orașul x . Victoria nu este deloc sigură în situația în care timpul total necesar transportului unităților nu este minim. Datorită tehnologiei avansate, drumul direct dintre oricare două orașe poate fi parcurs în timp constant egal cu o unitate de timp.

Cerințe

Scrieți un program care să determine timpul total minim de deplasare a tuturor unităților necesare în orașele învecinate cu orașul x .

Date de intrare

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

- pe prima linie, 4 numere naturale n , m , u , x , separate prin câte un spațiu, n reprezentând numărul orașelor, m reprezentând numărul drumurilor directe existente între orașe, u reprezentând numărul unităților de armată de care dispune Costel, x reprezentând orașul în care se află armata adversarului.

- pe a doua linie, u numere naturale, separate prin câte un spațiu, reprezentând orașele în care se află cantonate cele u unități de armată.

- fiecare din următoarele m linii conține căte două numere naturale a și b , separate printr-un spațiu, cu semnificația că orașele a și b sunt legate printr-un drum direct.

Date de ieșire

Fișierul de ieșire **atac.out** conține un singur număr natural reprezentând timpul total minim cerut.

Restricții și precizări

- $1 < n \leq 10\ 000$
- $1 < m \leq 100\ 000$
- $0 < u \leq 70$
- Numărul orașelor învecinate cu orașul x este cel mult egal cu numărul unităților de armată
- Pentru 60% din teste $u \leq 8$

Exemplu:

atac.in	atac.out	Explicații
7 11 3 7 6 1 3 1 4 2 5 3 1 3 5 4 5 5 1 6 2 6 3 6 4 7 4 7 5	2	Orașul 7 se învecinează cu orașele 4 și 5, deci va fi nevoie să deplasăm doar 2 unități dintre cele 3 disponibile. Timpul minim necesar este 2. Există mai multe variante pentru mutarea unităților, de exemplu: - mutăm unitatea din orașul 6 în orașul 4 (1 unitate de timp) și cea din orașul 3 în orașul 5 (1 unitate de timp) - mutăm unitatea din orașul 1 în orașul 4 (1 unitate de timp) și cea din orașul 3 în orașul 5 (1 unitate de timp)

Timp maxim de executare/test: **0.6** secunde

30.4.1 Indicații de rezolvare

Asociem hărții un graf neorientat cu n vârfuri și m muchii, fiecăruia oraș corespunzându-i un vârf și fiecăruia drum direct corespunzându-i o muchie. Notăm cu k numărul de noduri care sunt legate printr-o muchie de nodul X . Fie aceste noduri X_1, X_2, \dots, X_k .

Din fiecare nod U_i care conține o unitate, vom face o căutare în lățime pentru a afla lungimile minime ale drumurilor de la acest nod la toate nodurile X_j , $1 \leq j \leq k$.

Soluția 1. (100 puncte) - Tiberiu Daneț

Problema se reduce la a realiza un *cuplaj maxim de cost minim* în *graful bipartit* în care una dintre mulțimi este formată din nodurile ce conțin unități, iar cealaltă mulțime de nodurile X_1, X_2, \dots, X_k (fiecare unitate trebuie dusă într-un nod X_j și nu are rost să ducem mai multe unități în același nod X_j). Costul de pe muchia ce leagă nodul U_i de nodul X_j este lungimea minimă a drumului de la U_i la X_j găsită anterior.

Complexitatea cuplajului maxim de cost minim în graful bipartit complet cu $k + U$ noduri se face în complexitate $O(k * k * k * U)$ (un *drum alternant de creștere* se poate obține cu *algoritmul Bellman-Ford* și sunt k drumuri alternante de creștere).

Soluția 2 (60 puncte) - Alin Burța

Dacă numărul unităților disponibile este suficient de mic (pentru 60% din teste avem $u \leq 8$), determinarea modului în care cuplăm cele u unități cu cele k noduri vecine cu X se poate realiza cu un algoritm de tip *backtracking*.

30.4.2 Cod sursă

https://www.infoarena.ro/job_detail/2597053?action=view-source

Listing 30.4.1: atac.cpp

```

1 // https://www.infoarena.ro/job_detail/2597053?action=view-source
2 // Pavel Alexandru - alex2209alex
3 #include <bits/stdc++.h>
4
5 using namespace std;
6
7 ifstream f("atac2.in");
8 ofstream g("atac2.out");
9
10 //-----
11
12 ///Globale
13
14 const short UMAX = 71, NMAX = 10001, INF = 10001, MAX = 142;
15 short n, u, x, soldat[UMAX], dist[NMAX], vecini, cap[MAX][MAX],
16         cost[MAX][MAX], sursa, destinatie, flux[MAX][MAX], par[MAX];
17 vector<short> muchii[NMAX], muchii2[MAX];
18 bitset<NMAX> ver;
19 bitset<MAX> ap;
20 int raspuns;
21
22 //-----
23
24 ///Functii
25
26 void citire();
27 void rezolvare();
28 void afisare();
29
30 //-----
31
32 int main()
33 {
34     citire();
35     rezolvare();
36     afisare();
37     return 0;
38 }
39

```

```

40 //-----
41
42 void afisare()
43 {
44     g << raspuns;
45 }
46
47 //-----
48
49 bool Bellman_Ford()
50 {
51     for(int i = sursa; i <= destinatie; ++i)
52         dist[i] = INF;
53     dist[sursa] = 0;
54     queue<short>q;
55     q.push(sursa);
56     ap[sursa] = 1;
57     while(!q.empty())
58     {
59         int nod = q.front();
60         q.pop();
61         ap[nod] = 0;
62
63         for(auto it : muchii2[nod])
64             if(flux[nod][it] < cap[nod][it] &&
65                 dist[nod] + cost[nod][it] < dist[it])
66             {
67                 dist[it] = dist[nod] + cost[nod][it];
68                 par[it] = nod;
69                 if(!ap[it])
70                 {
71                     q.push(it);
72                     ap[it] = 1;
73                 }
74             }
75     }
76
77     if(dist[destinatie] != INF)
78         return 1;
79
80     return 0;
81 }
82
83 //-----
84
85 void bfs(int nod)
86 {
87     for(int i = 1; i <= n; ++i)
88         dist[i] = INF;
89
90     dist[nod] = 0;
91     queue<short>q;
92     q.push(nod);
93     while(!q.empty())
94     {
95         int nod = q.front();
96         q.pop();
97         for(auto it : muchii[nod])
98             if(dist[it] == INF)
99             {
100                 dist[it] = dist[nod] + 1;
101                 q.push(it);
102             }
103     }
104
105     vecini++;
106
107     for(int i = 1; i <= u; ++i)
108     {
109         cap[i][u + vecini] = 1;
110         muchii2[i].push_back(u + vecini);
111         muchii2[u + vecini].push_back(i);
112         cost[i][u + vecini] = dist[soldat[i]];
113         cost[u + vecini][i] = -dist[soldat[i]];
114     }
115 }
```

```

116
117 //-----
118
119 void rezolvare()
120 {
121     for(auto it : muchii[x])
122         if(!ver[it])
123         {
124             ver[it] = 1;
125             bfs(it);
126         }
127
128     sursa = 0;
129     for(int i = 1; i <= u; ++i)
130     {
131         cap[sursa][i] = 1;
132         muchii2[sursa].push_back(i);
133     }
134
135     destinatie = u + vecini + 1;
136     for(int i = 1; i <= vecini; ++i)
137     {
138         cap[u + i][destinatie] = 1;
139         muchii2[u + i].push_back(destinatie);
140     }
141
142     while(Bellman_Ford())
143     {
144         int nod = destinatie;
145         while(nod != sursa)
146         {
147             int nod2 = par[nod];
148             flux[nod2][nod]++;
149             flux[nod][nod2]--;
150             nod = nod2;
151         }
152
153         raspuns += dist[destinatie];
154     }
155 }
156 //-----
157
158 void citire()
159 {
160     int m;
161     f >> n >> m >> u >> x;
162
163     for(short i = 1; i <= u; ++i)
164         f >> soldat[i];
165
166     while(m--)
167     {
168         short a,b;
169         f >> a >> b;
170
171         muchii[a].push_back(b);
172         muchii[b].push_back(a);
173     }
174 }
175

```

30.4.3 *Rezolvare detaliată

30.5 drum

Problema 5 - drum

100 de puncte

Fie P , de coordonate carteziene (a, b, c) , și Q , de coordonate carteziene (x, y, z) , două puncte distincte din spațiul tridimensional. Pe mulțimea punctelor din spațiu se definește relația de ordine

'<' astfel: un punct P este mai mic decât un punct Q (adică $P < Q$) dacă este satisfăcută una dintre relațiile:

$$1) a < x; \quad 2) a = x \text{ și } b < y; \quad 3) a = x, b = y \text{ și } c < z.$$

Fie n un număr natural nenul și M mulțimea ordonată crescător, pe baza relației de ordine '<', a tuturor punctelor din spațiu ale căror coordonate (k, i, j) sunt numere naturale și satisfac condițiile: $1 \leq k \leq n$, $1 \leq i \leq k$, $1 \leq j \leq k$. Numărul punctelor din mulțimea M este $m = 1 + 4 + 9 + \dots + n^2$. Punctele din mulțimea ordonată M se numerotează cu numerele distințe $1, 2, \dots, m$ în ordinea în care apar în aceasta.

Fiecare punct din mulțimea ordonată M î se asociază câte o valoare naturală nenulă. Astfel, primului punct $P_1 \in M$ î se asociază valoarea c_1 , celui de-al doilea punct $P_2 \in M$ î se asociază valoarea c_2 , ..., celui de-al m -lea punct $P_m \in M$ î se asociază valoarea c_m , iar $P_1 < P_2 < \dots < P_m$.

Pornind de la punctul P_1 de coordonate $(1, 1, 1)$, se contruiesc drumuri astfel încât succesorul unui punct de pe drum, de coordonate carteziene (k, i, j) , poate fi unul dintre cele 3 puncte din M ale căror coordonate sunt: $(k+1, i, j+1)$, $(k+1, i+1, j)$, $(k+1, i+1, j+1)$, pentru $1 \leq k < n$. De exemplu, dacă $n > 3$ succesorul punctului de coordonate $(3, 1, 2)$ poate fi oricare din punctele de coordonate: $(4, 1, 3)$, $(4, 2, 2)$, $(4, 2, 3)$. Dacă $n = 3$ atunci punctul de coordonate $(3, 1, 2)$ nu are succesor.

Drumul $A_1, A_2, A_3, \dots, A_n$ precede lexicografic drumul $B_1, B_2, B_3, \dots, B_n$ dacă există un indice j ($1 \leq j \leq n$) astfel încât $A_i = B_i$ ($1 \leq i < j$) și $A_j < B_j$.

Cerințe

Scrieți un program care să citească numărul natural nenul n și cele m numere naturale nenule c_1, c_2, \dots, c_m și apoi să determine și să afișeze suma maximă S care se poate obține însumând toate valorile asociate punctelor de pe un drum construit în modul descris în enunț, precum și drumul pentru care se obține suma maximă. Dacă există mai multe drumuri pentru care se obține suma maximă, se va afișa primul drum din punct de vedere lexicografic.

Date de intrare

Fișierul de intrare **drum.in** conține pe prima linie un număr natural nenul n . A doua linie conține m numere naturale nenule c_1, c_2, \dots, c_m , separate prin câte un spațiu, reprezentând valorile asociate punctelor din mulțimea ordonată M .

Date de ieșire

Fișierul de ieșire **drum.out** va conține pe prima linie un număr natural reprezentând suma maximă S . A doua linie va conține un drum pentru care se obține suma maximă S , scriindu-se numărul fiecărui punct aflat pe drum, în ordinea parcurgerii acestora, numerele separându-se prin câte un singur spațiu.

Restricții și precizări

- $1 \leq n \leq 30$; $1 \leq c_i < 100$, $\forall 1 \leq i \leq m$
- Punctele de coordonat (n, i, j) nu au succesi (math>1 \leq i \leq n, $1 \leq j \leq n$)
- Pentru suma maximă S corectă se acordă 60% din punctaj; pentru un drum pentru care se obține suma maximă S se acordă 20% din punctaj, iar pentru primul drum din punct de vedere lexicografic pentru care se obține suma maximă S se acordă 40% din punctaj.

Exemple:

drum.in	drum.out	Explicații
3 3 6 5 7 2 4 5 8 7 6 1 7 8 13	18 1 4 13	Sunt 14 puncte în mulțimea M . Suma maximă care se poate obține este 18, valoare ce se va scrie pe prima linie a fișierului drum.out . Sunt 2 drumuri pentru care se obține suma maximă: $(P1, P4, P13)$ și $(P1, P5, P14)$. Primul drum fiind cel mai mic (lexicografic) se vor scrie pe a doua linie a fișierului drum.out numere 1 4 13, obținându-se punctajul maxim.

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

30.5.1 Indicații de rezolvare

prof. Carmen Mincă

O soluție se poate obține prin aplicarea *metoda programării dinamice, metoda înainte*.

Se utilizează două masive tridimensionale A și T .

În masivul A , $A[1][1][1]$ va memora valoarea asociată punctului de coordonate $(1, 1, 1)$, ..., $A[k][i][j]$ va memora valoarea asociată punctului de coordonate (k, i, j) , pentru $1 \leq k \leq n$, $1 \leq i \leq k$, $1 \leq j \leq k$.

Masivul T se va inițializa cu valorile din masivul A și va fi utilizat pentru a reține valorile sumele maxime parțiale calculate pentru drumurile construite.

Pentru un punct de coordonate (k, i, j) , suma maximă care se poate calcula, pentru un drum care pornește din acest punct, se obține pe baza relațiilor:

$$T[n][i][j] = A[n][i][j], \text{ pentru } 1 \leq i \leq n, 1 \leq j \leq n. (k = n)$$

$$T[k][i][j] = A[k][i][j] + \max\{T[k+1][i][j+1], T[k+1][i+1][j], T[k+1][i+1][j+1]\}$$

pentru $k = n - 1, n - 2, \dots, 1$, $1 \leq i \leq k$, $1 \leq j \leq k$.

Suma maximă va fi reținută în $T[1][1][1]$. Pentru a afișa numerelor punctelor de pe un drum de sumă maximă, refacem traseul prin care a fost obținută suma maximă, pornind de la $T[1][1][1]$ sau se memorează coordonatele punctelor în timpul construirii sumelor parțiale maxime.

Pentru exemplul din enunț, se pot reprezenta punctele separat, pe plane:

Planul de ecuație: $k=1$

$P_1(1, 1, 1)$

Planul de ecuație: $k=2$

$P_2(2, 1, 1)$	$P_3(2, 1, 2)$
$P_4(2, 2, 1)$	$P_5(2, 2, 2)$

Planul de ecuație: $k=3$

$P_6(3, 1, 1)$	$P_7(3, 1, 2)$	$P_8(3, 1, 3)$
$P_9(3, 2, 1)$	$P_{10}(3, 2, 2)$	$P_{11}(3, 2, 3)$
$P_{12}(3, 3, 1)$	$P_{13}(3, 3, 2)$	$P_{14}(3, 3, 3)$

Valorile asociate punctelor:

Planul de ecuație: $k=1$

3

Planul de ecuație: $k=2$

6	5
7	2

Planul de ecuație: $k=3$

4	5	8
7	6	1
7	8	13

Se obțin două drumuri: $D_1 = (P_1, P_4, P_{13})$ și $D_2 = (P_1, P_5, P_{14})$ și $D_1 < D_2$. Astfel drumul căutat este $D_1 = (P_1, P_4, P_{13})$ iar suma maximă este $S = 3 + 7 + 8 = 3 + 2 + 13 = 18$.

30.5.2 Cod sursă

https://www.infoarena.ro/job_detail/2563906?action=view-source

Listing 30.5.1: drum.cpp

```

1 // https://www.infoarena.ro/job_detail/2563906?action=view-source
2 // Moga Marius-Ioan - Mariusbloc
3
4 #include <bits/stdc++.h>
5
6 #define MOD 104659
7 #define ull unsigned long long
8
9 using namespace std;
10
11 ifstream fin("drum2.in");
12 ofstream fout("drum2.out");
13
14 pair<int,int> DP[35][35][35];
15
16 int main()
17 {
18     int n,i,j,k,m = 0,x,y,maxim;
19

```

```

20     fin>>n;
21     for(i = 1; i <= n; i++)
22     {
23         for(j = 1; j <= i; j++)
24         {
25             for(k = 1; k <= i; k++)
26             {
27                 m++;
28                 fin>>DP[i][j][k].first;
29                 DP[i][j][k].second = m;
30             }
31         }
32     }
33
34     for(i = n-1; i >= 1; i--)
35     {
36         for(j = 1; j <= i; j++)
37         {
38             for(k = 1; k <= i; k++)
39             {
40                 DP[i][j][k].first = max(DP[i+1][j][k+1].first,
41                                         max(DP[i+1][j+1][k].first,
42                                             DP[i+1][j+1][k+1].first)) +
43                                         DP[i][j][k].first;
44             }
45         }
46     }
47
48     fout<<DP[1][1][1].first<<' \n';
49
50     x = 1;
51     y = 1;
52     for(i = 1; i <= n; i++)
53     {
54         fout<<DP[i][x][y].second<<' ';
55         maxim = max(DP[i+1][x+1][y].first,
56                     max(DP[i+1][x+1][y+1].first, DP[i+1][x][y+1].first));
57         if(maxim == DP[i+1][x][y+1].first)
58         {
59             y++;
60         }
61         else if(maxim == DP[i+1][x+1][y].first)
62         {
63             x++;
64         }
65         else if(maxim == DP[i+1][x+1][y+1].first)
66         {
67             y++;
68             x++;
69         }
70     }
71
72     return 0;
73 }
```

30.5.3 *Rezolvare detaliată

30.6 virus

Problema 6 - virus

100 de puncte

Marian este programator la o firmă ce produce software antivirus și a primit ca sarcină scrierea motorului de căutare al produsului. Firma fiind recent înființată, analiștii îau pus la dispoziție doar un număr mic de virusi cunoscuți. Motorul va fi testat pe un sir de biți extras dintr-un executabil, trebuie să producă o statistică care să conțină numărul de apariții al fiecărui virus în sirul de biți.

Cerințe

Scrieți un program pentru a-l ajuta pe Marian să obțină statistică cerută.

Date de intrare

Prima linie a fișierul de intrare **virus.in** conține două numere naturale L și N , separate printr-un spațiu, L reprezentând mărimea șirului de biți, iar N reprezentând numărul virusilor cunoscuți. A doua linie a fișierului conține un sir de lungime L , format doar din caracterele '0' și '1', reprezentând șirul de biți. Următoarele $2 * N$ linii conțin descrierea virusilor puși la dispoziție de echipa de analiști. Fiecare virus este descris pe două linii consecutive; prima dintre aceste linii conține un număr natural k reprezentând lungimea acestui virus (exprimată în biți), iar a doua linie conține un sir de lungime k , format doar din caracterele '0' și '1', reprezentând descrierea lui.

Date de ieșire

Fișierul de ieșire **virus.out** va conține exact N linii. Pe fiecare linie se va scrie o valoare naturală reprezentând numărul de apariții al fiecărui virus cunoscut, în ordinea dată în fișierul de intrare.

Restricții și precizări

- $1 \leq N \leq 1\,000$
- $1 \leq L \leq 100\,000$
- $1 \leq k \leq 1\,000$
- numărul total de apariții nu va depăși 1 000 000

Exemple:

virus.in	virus.out	Explicații
7 3	0	Sunt 3 virusi.
0110101	3	Primul virus din fișier, 11111, nu apare în șirul de biți, astfel se va scrie valoarea 0 pe prima linie a fișierului virus.out .
5	2	Cel de-al doilea virus din fișier, 0, apare în șirul de biți de 3 ori (pozițiile 1, 4 și 6), astfel se va scrie valoarea 3 a doua linie a fișierului virus.out .
11111		Ultimul virus din fișier, 101, apare în șirul de biți de 2 ori (începând cu pozițiile 3 și 5), astfel se va scrie valoarea 2 pe a treia linie a fișierului virus.out .
1		
0		
3		
101		

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

Memorie: total **96 MB** din care pentru stivă **1 MB**

30.6.1 Indicații de rezolvare

Pătcaș Csaba

Problema se poate rezolva prin mai multe metode, având complexități diferite.

Fie S suma lungimilor virusilor.

O abordare bazată pe algoritmul naiv de căutare al subșirului are complexitatea $O(S * m)$ și poate obține 10 de puncte.

Construirea unui *arbore de sufixe* se poate face în $O(m^2)$, după care numărarea apariției fiecarui sir se poate face în $O(S)$. Deci aceasă soluție are în total complexitatea $O(m^2 + S)$ și obține 30 de puncte.

Aplicarea de n ori a unui algoritm liniar de potrivire a șirurilor (de exemplu *Knuth-Morris-Pratt*, sau *Boyer-Moore*) are complexitatea $O(n * m + S)$ și obține între 30 și 60 de puncte.

Construirea șirului de sufixe al șirului de biți și se poate face prin diferiți algoritmi cu complexități de $O(m * \log^2 m)$, $O(m * \log m)$ sau $O(m)$. După obținerea șirului de sufice, numărul de apariții al fiecarui virus se poate determina prin două *căutări binare*.

Această abordare are în total complexitatea $O(m * \log^2 m + S * \log m)$, $O(m * \log m + S * \log m)$ sau $O(m + S * \log m)$ și poate obține 70-80 de puncte.

Pentru obținerea punctajului maxim este nevoie de o implementare eficientă a *algoritmului Aho-Corasick*.

30.6.2 Cod sursă

https://www.infoarena.ro/job_detail/2541828?action=view-source

Listing 30.6.1: virus.cpp

```

1 // https://www.infoarena.ro/job_detail/2541828?action=view-source
2 // Morar Eusebiu - eusebiu_alexandru
3
4 #include <stdio.h>
5
6 int n,m,k,i,j,t,a[100005],b[1005],q;
7 char s[100010];
8
9 int main()
10 {
11     freopen("virus.in","r",stdin);
12     freopen("virus.out","w",stdout);
13
14     scanf("%d%d\n",&n,&m);
15     fgets(s,100002,stdin);
16
17     for (i=1; i<=n; ++i) a[i]=a[i-1]+s[i-1];
18
19     for (t=1; t<=m; ++t)
20     {
21         scanf("%d\n",&k);
22         fgets(s,1010,stdin);
23
24         int sol=0;
25         for (i=1; i<=k; ++i)
26             b[i]=b[i-1]+s[i-1];
27
28         for (i=k; i<=n; ++i)
29             if (b[k]==a[i]-a[i-k])
30             {
31                 q=1;
32                 for (j=1; j<=k; ++j)
33                     if (b[j]!=a[i-k+j]-a[i-k])
34                     {
35                         q=0;
36                         break;
37                     }
38                 if (q) ++sol;
39             }
40
41             printf("%d\n",sol);
42     }
43
44     return 0;
45 }
```

30.6.3 *Rezolvare detaliată

Capitolul 31

ONI 2007 clasa a XI-a



Figura 31.1: Sigla ONI 2007

31.1 Descompunere

Considerăm trei numere naturale nenule: n , k și x .

Denumim o kx -descompunere a numărului n o posibilitate de a scrie numărul n ca sumă de k numere naturale nenule astfel încât diferența între oricare doi termeni ai sumei este cel puțin egală cu x .

Cerință

Fiind date trei numere naturale n , k și x , să se determine câte kx -descompuneri distințe există. Două kx -descompuneri sunt distințe dacă diferă prin cel puțin un termen.

Date de intrare

Fișierul **desc.in** conține pe prima linie trei valori naturale nenule reprezentând numerele n , k și x .

Date de ieșire

Fișierul **desc.out** va conține o singură valoare reprezentând restul împărțirii numărului de kx -descompuneri distințe la numărul 10007.

Restricții și precizări

- pentru 20% din teste $0 < n \leq 200$;
- pentru celelalte 80% din teste, $200 < n \leq 10000$;
- $1 \leq x, k \leq n$

Exemplu

desc.in	desc.out
20 2 3	8

Explicații

Numărul de kx -descompuneri în acest caz este 8. Acestea sunt formate din numerele 1 și 19; 2 și 18; 3 și 17; 4 și 16; 5 și 15; 6 și 14; 7 și 13; 8 și 12

desc.in	desc.out
2000 19 7	3184

Limită de memorie: 32 Mb, din care 1 Mb pentru stivă.

Timp maxim de execuție/test: 0.1 secunde

31.1.1 Indicații de rezolvare

Stelian Ciurea

Fie o kx -descompunere, adică

$$N = a_1 + a_2 + \dots + a_k$$

În aceasta vom presupune că

$$a_1 < a_2 < a_3 < \dots < a_k$$

sau

$$a_1 \leq a'_2 + x \leq a'_3 + x \leq \dots \leq a'_k + (k-1)x$$

în care $a_1 \leq a'_2 \leq a'_3 \leq \dots \leq a'_k$

Dacă o rescriem, rezultă

$$N = a_1 + a'_2 + a'_3 + \dots + a'_k + x + 2x + \dots + (k-1)x$$

adică

$$N - x(1 + 2 + \dots + (k-1)) = a_1 + a'_2 + a'_3 + \dots + a'_k$$

ceea ce reprezintă o posibilitate de a scrie numărul $N - x * k(k-1)/2$ ca sumă de k numere naturale care pot fi și egale.

Acste transformări sunt bijective, adică oricarei kx -descompuneri a numărului N îi corespunde o descompunere în k numere care pot fi și egale a numărului $N - x * k(k-1)/2$.

Numărul de posibilități de a scrie un număr oarecare - să îl notăm A - ca o sumă de k numere naturale nenule care pot fi și egale este dat de o formulă cunoscută (care apare și în unele manuale mai vechi de clasa a X-a) denumită *formula lui Stirling*:

$$S(A, k) = S(A - k, 1) + S(A - k, 2) + \dots + S(A - k, k)$$

Aplicând această formulă pentru $A = N - x * k(k-1)/2$ obținem rezultatul cerut.

31.1.2 Cod sursă

Listing 31.1.1: desc.c

```

1 /*desc - varianta c pentru gcc* cu backtracking*/
2 #include <stdio.h>
3
4 #define nrprim 10007
5
6 long s[501],n,k,x,sum,kt=0;
7
8 void back(int p)
9 { for (s[p]=s[p-1]+x;s[p]<=n-sum;s[p]++)
10   {
11     sum += s[p];
12     if (sum==n &&p==k)
13       kt = (kt+1) % nrprim;
14     else
15       if (p<k)
16         back(p+1);
17       sum -= s[p];
18   }
19 }
20
21 int main()
22 { FILE * fi,*fo;
23   fi = fopen("desc.in","rt");

```

```

24     fo = fopen("desc.out", "wt");
25     fscanf(fi, "%ld%ld%ld", &n, &k, &x);
26     s[0]=-x+1;
27
28     back(1);
29
30     fprintf(fo, "%ld\n", kt);
31
32 // fcloseall();
33
34     return 0;
35 }
```

31.1.3 *Rezolvare detaliată

31.2 Felinare

A venit timpul Anarhiei în Orașul Trist! Ca revoltă împotriva manifestărilor subculturale, vrei să pui Orașul pe butuci.

În urma unei descinderi ilegale la Primarie, ai "împrumutat" o hartă și îți-ai dat seama că există M străzi unidirectionale între cele N intersecții ale Orașului Trist.

Fiecare intersecție are câte două felinare. Primul luminează o jumătate din fiecare stradă care pleacă din intersecția respectivă, iar al doilea luminează jumătate din fiecare stradă care intră în intersecție. De exemplu, prima jumătate a străzii dintre intersecțiile A și B este luminată de primul felinar din intersecția A , iar cea de-a doua jumătate este luminată de al doilea felinar din intersecția B .

Un felinar stins nu luminează deloc. O stradă este sigură doar atunci când e luminată în totalitate.

Cerință

În primul rând, trebuie să te asiguri că nicio stradă nu va fi complet luminată, astfel încât siguranța cetățenilor să fie redusă la minim. Dar acest obiectiv nu te mulțumește, aşa că în plus îți dorești un număr maxim de felinare aprinse, pentru a da o grea lovitură bugetului Primăriei din Orașul Trist. Odată îndeplinite aceste condiții, Revoluția poate începe.

Date de intrare

Fișierul de intrare **felinare.in** conține pe prima linie două numere naturale strict pozitive N și M , reprezentând numărul de intersecții și numărul de străzi din Orașul Trist. Pe fiecare din următoarele linii se află o pereche de numere naturale A și B cuprinse între 1 și N reprezentând o stradă care pleacă din intersecția A și ajunge în intersecția B .

Date de ieșire

Fișierul de ieșire **felinare.out** va conține pe prima linie un singur număr natural reprezentând numărul maxim de felinare ce pot fi aprinse. Pe următoarele N linii se vor afla numere cuprinse între 0 și 3, cu semnificația următoare:

- 0 : ambele felinare din intersecție sunt stinse;
- 1 : numai primul dintre felinarele din intersecție este aprins;
- 2 : numai al doilea dintre felinarele din intersecție este aprins;
- 3 : ambele felinare din intersecție sunt aprinse.

Restricții și precizări

$1 \leq N \leq 8191$

$1 \leq M \leq 20000$

Nu există străzi care să unească o intersecție cu ea însăși.

Pentru determinarea numărului maxim de felinare se acordă 40% din punctaj.

Exemplu

felinare.in	felinare.out
4 4	6
1 2	2
4 1	3
4 2	3
4 3	2

Limită de memorie: 32 Mb, din care 1 Mb pentru stivă.

Timp maxim de execuție/test: 0.1 secunde

31.2.1 Indicații de rezolvare

Tiberiu Florea

Pornind de la graful inițial, construim un nou graf bipartit, după cum urmează: în stânga N noduri corespunzătoare felinarelor de tipul 1, iar în dreapta N noduri corespunzătoare felinarelor de tipul 2.

O muchie $i \rightarrow - \rightarrow j$ în graful inițial devine o muchie între nodul i din stânga și nodul j din dreapta în noul graf bipartit.

Se observă că o stradă $i \rightarrow - \rightarrow j$ este complet iluminată $\rightarrow - \rightarrow j$ dacă și numai dacă atât felinarul corespunzător nodului i din stânga cât și felinarul corespunzător nodului j din dreapta sunt aprinse.

În concluzie, având dat un graf bipartit vrem să alegem un număr maxim de noduri astfel încât să nu existe nici o muchie cu ambele noduri adiacente alese.

Pentru a rezolva problema, avem nevoie mai întâi de câteva definiții.

Într-un graf bipartit, un "suport minim" reprezintă o mulțime de noduri cu cardinal minim pentru care orice muchie a grafului este adiacentă cu cel puțin unul dintre nodurile mulțimii.

Unul dintre cele mai importante rezultate de teoria grafurilor, Teorema lui König, spune că într-un graf bipartit cuplajul maxim și suportul minim sunt egale. Pe baza acestei teoreme, un suport minim se poate determina relativ ușor pornind de la orice cuplaj maxim. O "mulțime independentă maximală" (ceea ce ne cere, de fapt, problema), este complementul oricărui suport minim.

Suportul minim se poate calcula în modul următor:

```
// presupunem ca a fost calculat deja un cuplaj maxim
// S este suportul minim, initial multime vida
// C(j-dreapta) e nodul i-stanga cu care e cuplat j-dreapta (daca exista)

procedura calculeaza(i-stanga) // doar pt. noduri care nu sunt in suport
    pentru j-dreapta vecin al lui i-stanga
        daca j-dreapta nu e in S
            S <- S + {j-dreapta}
            S <- S - { C(j-dreapta) } // exista mereu C(j-dreapta)
            calculeaza( C(j-dreapta) )

    pentru fiecare i din stanga
        daca i e cuplat
            S <- S + {i}

    pentru fiecare i din stanga
        daca i nu e cuplat
            calculeaza(i)
```

Pentru 40 de puncte este suficient să calculăm un cuplaj maxim și soluția egală cu 2^N -valoarea cuplajului. Pentru celealte 60 de puncte trebuie să determinăm un suport minim și apoi o mulțime independentă maximală pornind de la acest cuplaj. Complexitatea totală a algoritmului este $O(N * M)$.

31.2.2 Cod sursă

Listing 31.2.1: felinare.cpp

```

1  using namespace std;
2
3 #include <cstdio>
4 #include <cstring>
5 #include <cstdlib>
6 #include <vector>
7
8 #include <cassert>
9
10 #define NDEBUG
11
12 #define FIN "felinare.in"
13 #define FOUT "felinare.out"
14
15 #define MAX_N 8192
16
17 int N, M, cnt;
18 int l[MAX_N], r[MAX_N], u[MAX_N], sl[MAX_N], sr[MAX_N];
19 vector <int> G[MAX_N];
20
21 #ifndef NDEBUG
22 int stp;
23#endif
24
25 void read()
26 {
27     int a, b;
28     scanf("%d %d", &N, &M);
29     while (M--)
30     {
31         scanf("%d %d", &a, &b);
32         G[a].push_back(b);
33     }
34 }
35
36 int pairup(int n)
37 {
38     if (u[n])
39         return 0;
40     u[n] = 1;
41     vector <int> :: iterator it;
42     for (it = G[n].begin(); it != G[n].end(); ++it)
43         if (!l[*it])
44         {
45             l[*it] = n;
46             r[n] = *it;
47             //sr[*it] = 1;
48             sl[n] = 1;
49             return 1;
50         }
51
52     for (it = G[n].begin(); it != G[n].end(); ++it)
53         if (pairup(l[*it]))
54         {
55             l[*it] = n;
56             r[n] = *it;
57             sl[n] = 1;
58             return 1;
59         }
60
61     return 0;
62 }
63
64 void support(int n)
65 {
66     vector <int> :: iterator it;
67     for (it = G[n].begin(); it != G[n].end(); ++it)
68         if (!sr[*it])
69         {
70             #ifndef NDEBUG
71                 ++stp;
72            #endif
73             sr[*it] = 1;
74             sl[l[*it]] = 0;
75         }
76 }

```

```

75             support(l[*it]);
76         }
77     }
78
79     void solve()
80     {
81         int i;
82         for (i = 1; i <= N; ++i)
83             if (!pairup(i))
84             {
85                 memset(u, 0, sizeof(u));
86                 cnt += pairup(i);
87             }
88             else
89                 ++cnt;
90
91         for (i = 1; i <= N; ++i)
92             if (!sl[i])
93                 support(i);
94
95 #ifndef NDEBUG
96         fprintf(stderr, "N = %d, flux = %d, pasi = %d\n", N, cnt, stp);
97 #endif
98     }
99
100    void write()
101    {
102        int i;
103        printf("%d\n", 2*N-cnt);
104        for (i = 1; i <= N; ++i)
105            printf("%d\n", 1-sl[i]+2*(1-sr[i]));
106    }
107
108    int main()
109    {
110        freopen(FIN, "r", stdin);
111        freopen(FOU, "w", stdout);
112
113        read();
114        solve();
115        write();
116        return 0;
117    }

```

31.2.3 *Rezolvare detaliată

31.3 Joc

Gică și Petrică locuiesc în Orașul Vesel. Acolo există o rețea stradală care conține doar străzi cu sens unic iar pentru a te deplasa între două intersecții trebuie să plătești o taxă specifică fiecărei intersecții și fiecărei străzi prin care treci (inclusiv intersecția din care pleci și intersecția în care ajungi). Aceste taxe nu deranjează însă pe nimeni, deoarece sunt valori monetare naturale între 0 și 10. Pe cei doi nu îi interesează harta propriu-zisă a Orașului, însă au găsit tabelul sumelor minime pe care trebuie să le plătească pentru a se plimba între oricare două intersecții și s-au gândit să-l folosească pentru a juca un joc interesant (Această matrice conține numai valori finite).

Astfel, având matricea A în care $A[i, j]$ reprezintă costul minim pentru a ajunge din intersecția i în intersecția j ($A[i, i]$ este taxa care trebuie plătită în nodul i), fiecare dintre cei doi mută alternativ, după cum urmează: jucătorul aflat la mutare își alege un număr natural strict pozitiv k , și scade k din toate elementele unei linii sau coloane ale matricei, cu condiția ca toate elementele matricei să rămână nenegative. Gică mută primul, iar jucătorul care, atunci când îi vine rândul, nu mai poate efectua o mutare corectă, pierde. Se consideră că cei doi prieteni joacă perfect, însemnând că dacă unul dintre ei are la un moment dat o strategie de câștig indiferent de mutările adversarului, nu va efectua o mutare care să ducă la pierderea oricărei strategii de câștig.

Cerință

Dându-se matricea A cu semnificația din enunț, aflați care dintre cei doi este câștigătorul jocului.

Date de intrare

Fișierul **joc.in** conține între 1 și 20 de seturi de date de intrare. Pe prima linie a fiecărui set se află un număr natural pozitiv N reprezentând dimensiunile matricei A , iar pe următoarele N linii se află câte N numere naturale, reprezentând elementele matricei A . $N = 0$ marchează sfârșitul seturilor de date ce compun fișierul de intrare.

Date de ieșire

Fișierul de ieșire **joc.out** va conține un număr de linii egal cu numărul de seturi de date. Pe fiecare dintre aceste linii se află una dintre valorile: 1 în cazul în care jocul respectiv este câștigat de Gică, 2 în cazul în care jocul respectiv este câștigat de Petrică.

Restricții și precizări

$1 \leq$ numărul testelor ≤ 20

$2 \leq N \leq 100$

Rezolvarea corectă a testelor ce conțin cel mult 10 seturi de date cu $N \leq 5$ garantează obținerea a 50% din punctaj.

Exemplu

joc.in	joc.out
2	1
8 25	2
25 9	
2	
3 8	
9 3	
0	

Limită de memorie: 32 Mb, din care 1 Mb pentru stivă.

Timp maxim de execuție/test: 0.1 secunde

31.3.1 Indicații de rezolvare

Tiberiu Florea

Se pot obține 50 de puncte cu diverse soluții exponentiale (bazate, eventual, pe arbori de joc), fără să ținem cont de modul în care este obținută matricea.

Pentru punctajul maxim, facem următoarea observație: în graful inițial (care nu este citit), pentru a ajunge de la nodul i la nodul j trebuie să plătim cel puțin taxa din nodul i și taxa din nodul j . Astfel, fiecare matrice A respectă condiția $A[i, j] \geq A[i, i] + A[j, j]$ pentru orice i diferit de j .

Această proprietate se păstrează și în urma operațiilor permise, de scădere a unui număr de pe o linie sau coloană. Un jucător pierde atunci când pe fiecare linie și pe fiecare coloană există cel puțin câte un zero și, ținând cont de proprietatea precedentă, primele elemente ale matricii care devin nule sunt elementele diagonalei principale.

Indiferent dacă la un anumit pas scădem k de pe o linie sau de pe o coloană, un singur element de pe diagonala principală va scădea cu k . Se observă că jocul este echivalent cu NIM în varianta clasică pe elementele diagonalei principale. Complexitate: $O(N^2)$.

31.3.2 Cod sursă

Listing 31.3.1: joc.cpp

```

1 #include <cstdio>
2
3 int main()
4 {
5     int N, ans, i, j, x;
6
7     freopen("joc.in", "r", stdin);

```

```

8     freopen("joc.out", "w", stdout);
9
10    while (scanf("%d", &N) && N != 0)
11    {
12        for (ans = 0, i = 1; i <= N; ++i)
13            for (j = 1; j <= N; ++j)
14            {
15                scanf("%d", &x);
16                if (i == j)
17                    ans ^= x;
18            }
19
20        printf("%d\n", 2-(ans != 0));
21    }
22
23    return 0;
24 }
```

31.3.3 *Rezolvare detaliată

31.4 Logaritmi

Fie expresia:

$$\log_{a_1} b_1 * \log_{a_2} b_2 * \dots * \log_{a_n} b_n$$

Un calculator trebuie să evalueze această expresie aducând-o la forma unui singur număr real. Pentru aceasta, el poate face următoarele calcule:

- Produs = produsul a două numere reale în t_1 unități de timp;
- Reducere = înlocuirea expresiei $\log_a b * \log_b c$ cu $\log_a c$ în t_2 unități de timp;
- Calcul = calculul unui logaritmul rezultatul fiind un număr real; pentru a calcula $\log_a b$ îi sunt necesare $t_3 * (a - b)^2$ unități de timp.

Cerință

Să se determine timpul minim pentru a calcula o expresie dată.

Date de intrare

Fișierul **log.in** conține:

- pe prima linie o valoare numerică naturală n cu semnificația din enunț;
- pe a doua linie trei valori numerice naturale $t_1 \ t_2 \ t_3$ separate prin câte un spațiu, cu semnificația din enunț;
- pe fiecare din următoarele n linii câte două valori numerice naturale $a_i \ b_i$ cu semnificațiile din enunț.

Date de ieșire

Fișierul **log.out** va contine o singură valoare reprezentând numărul de unități de timp necesare evaluării expresiei.

Restricții și precizări

Pentru 70% din teste $0 < n \leq 500$; pentru celelalte 30% din teste $n \leq 10000$;

$1 < a_i, b_i < 100 \ 1 \leq t_1, t_2, t_3 \leq 100$

Factorii expresiei inițiale sau ai oricareia dintre expresiile rezultate pe parcursul evaluării NU pot fi comutați între ei.

Exemplu

log.in	log.out
3	13
2 1 3	
2 3	
3 4	
4 5	

Explicații:

Se calculează fiecare din cei trei logaritmi, rezultă trei numere, fiecare calcul necesită 3 unități de timp; se înmulțesc primele două numere în 2 unități de timp, apoi rezultatul se înmulțește cu al treilea număr tot în 2 unități; în total: $3 + 3 + 3 + 2 + 2 = 13$ unități.

Exemplu

log.in	log.out
4	9
2 1 2	
2 2	
3 4	
4 4	
4 5	

Explicații:

Primul logaritmul se calculează în 0 unități; al doilea și al treilea se reduc la un logaritmul în 1 unitate iar acest logaritmul se calculează în 2 unități; al patrulea se calculează în 2 unități; au rezultat trei numere, care pot fi aduse la unul singur prin două înmulțiri, fiind necesare $1 + 2 + 2 + 2 + 2 = 9$ unități de timp.

Limită de memorie: 32 Mb, din care 1 Mb pentru stivă.

Timp maxim de execuție/test: 1 secundă

31.4.1 Indicații de rezolvare

Stelian Ciurea

Soluția 1 (Stelian Ciurea)

Pentru 70 puncte, problema se poate rezolva în complexitate $O(n^3)$ folosind un algoritm asemănător cu cel pentru înmulțirea optimală a unui sir de matrice. Astfel se calculează o matrice în care fiecare element $o[i, j, 0]$ reține timpul minim necesar pentru a calcula expresia formată din logaritmi și a o aduce la forma unui logaritmul, iar $o[i, j, 1]$ reține timpul minim necesar pentru a calcula expresia formată din logaritmi și a o aduce la forma unui număr real.

Soluția se găsește în $o[1][n][1]$.

Soluția 2

(Tiberiu Florea, Daniel Păsăilă)

Pentru 100 puncte era necesară o soluție în $O(n^2)$. Pentru asta trebuie să folosim un tablou V , unde $V[i]$ reprezintă costul minim pentru a calcula primii i logaritmi. $V[i]$ se calculează ușor în timp liniar, considerând toate secvențele posibile din care ar putea face parte logaritmul i (înainte de a fi transformat într-un real). Relațiile de recurență se deduc ușor.

31.4.2 *Cod sursă

Listing 31.4.1: log.cpp

```

1 #include <cstdio>
2 #include <cstring>
3
4 int N, t1, t2, t3, ans, a[1<<14], b[1<<14];
5 int S[1<<14];
6
7 int main()
8 {
9     int i, j;
10    int t;
11
12    freopen("log.in", "r", stdin);
13    freopen("log.out", "w", stdout);
14
15    scanf("%d\n%d %d %d", &N, &t1, &t2, &t3);
16
17    memset(S, 0x3f3f3f3f, sizeof(S));
18
19    S[0] = N*t2-t1;
20    for (i = 1; i <= N; ++i)

```

```

21      {
22          scanf("%d %d", a+i, b+i);
23          for (j = i; j > 0; --j)
24          {
25              if (j < i && b[j] != a[j+1])
26                  break;
27              if (S[i] > (t = S[j-1]+t3*(a[j]-b[i])* (a[j]-b[i])+t1-t2))
28              {
29                  S[i] = t;
30              }
31          }
32      }
33
34      printf("%d\n", S[N]);
35      return 0;
36  }

```

31.4.3 *Rezolvare detaliată

31.5 Maxq

Johnie a început să se joace cu un vector de numere. El dispune inițial de un vector V cu N numere întregi (numerotate de la 0 la $N - 1$) și poate efectua următoarele operații:

- schimbarea elementului de pe poziția p cu un alt număr întreg;
- aflarea subsecvenței de sumă maximă din V inclusă între indicii a și b ;

Cerință

Ajutați-l pe Johnie să efectueze repede operațiile de mai sus.

Date de intrare

Fisierul **maxq.in** conține pe prima linie numărul N reprezentând dimensiunea vectorului.

Pe următoarea linie se găsesc N elemente reprezentând valorile inițiale ale vectorului. Următoarea linie conține M , reprezentând numărul de operații. Pe fiecare dintre următoarele M linii sunt descrise cele M operații în forma următoare:

- 0 i p : numărul 0 de la început codifică faptul că operația curentă este una de schimbare; astfel elementul de pe poziția i a vectorului se înlocuiește cu numărul întreg p ;
- 1 a b : numărul 1 de la început codifică faptul că operația curentă este o întrebare; astfel se cere să se afle subsecvența de sumă maximă din vector inclusă între indicii a și b ($a \leq b$);

Date de ieșire

Fisierul **maxq.out** trebuie să conțină un număr de linii egal cu numărul de întrebări din fisierul de intrare. Pe linia i se cere să se afișeze un singur număr reprezentând suma maximă ce se poate obține în contextul întrebării i din fisierul de intrare ($i = 1, 2, \dots$); în cazul în care vor exista doar subsecvențe de sumă negativă se va afișa 0.

Restricții și precizări

- $1 \leq N \leq 200000$;
- $1 \leq M \leq 200000$;
- toate elementele vectorului apar?in intervalului $[-100000, 100000]$;
- definim o subsecvență ca fiind un sir de termeni consecutivi din vector, iar suma ei se obține adunând elementele ce o compun;
- există cel puțin o întrebare.
- pentru 20% din teste se garantează $N \leq 5000$

Exemplu

maxq.in	maxq.out
5	4
1 10 4 1 9	16
4	12
1 0 3	
0 1 1	
1 0 3	
1 2 4	

Explicație

Pentru prima întrebare se alege subsecvența formată de elementul pe poziția 2 din vector. Pentru a 2-a întrebare se aleg primele 3 elemente din vector (elementul de pe poziția 1 a fost schimbat). Pentru a 3-a întrebare se aleg toate elementele din intervalul cerut.

Limită de memorie: 32 Mb, din care 1 Mb pentru stivă.

Timp maxim de execuție/test: 1.2 secunde

31.5.1 Indicații de rezolvare

Daniel Păsăilă

Problema se poate rezolva în mai multe moduri. Prima posibilitate ar fi folosirea unui arbore de intervale. Astfel, în fiecare nod al arborelui se țin următoarele valori:

- A[nod] - valoarea subsecvenței de sumă maximă situată la începutul intervalului curent
- B[nod] - valoarea subsecvenței de sumă maximă situată la sfârșitul intervalului curent
- C[nod] - valoarea subsecvenței de sumă maximă situată oriunde în intervalul curent
- D[nod] - suma elementelor din intervalul current

Astfel, fiecare update se poate efectua în $O(\log N)$ (se apelează procedura pentru fiii nodului curent, apoi se calculează valorile pentru intervalul curent). Query-ul se efectuează parcursând recursiv intervalele consecutive din arbore ce sunt incluse în intervalul curent (complexitate $O(\log N)$). Soluția aceasta obține 100 puncte.

O altă modalitate de rezolvare a problemei ar fi împărțirea sirului de la intrare în \sqrt{N} secvențe. Pentru fiecare secvență trebuie ținute aceleași valori ca și în cazul arborelui de intervale, iar query-ul se efectuează asemănător. Soluția poate lua de la 60 puncte la 100 puncte, în funcție de implementare.

31.5.2 Cod sursă

Listing 31.5.1: maxq.cpp

```

1 //arbore de intervale -> 100p
2 #include <iostream>
3 #include <algorithm>
4
5 using namespace std;
6
7 #define FIN "maxq.in"
8 #define FOUT "maxq.out"
9
10 #define MAX_N (1<<18)
11 #define max(a, b) ((a) > (b) ? (a) : (b))
12
13 int N, M;
14 int v[MAX_N];
15 long long A[MAX_N<<1], B[MAX_N<<1], C[MAX_N<<1], Sum[MAX_N<<1];
16
17 #define lf (n<<1)
18 #define rt (lf+1)
19 #define md ((lf+r)>>1)
20
21 void build(int n, int l, int r)
22 {
23     if (l == r)
24     {
25         A[n] = B[n] = C[n] = max(v[l], 0);
26     }
27     else
28     {
29         int m = md;
30         build(n, l, m);
31         build(n, m, r);
32         A[n] = max(A[n], A[m]);
33         B[n] = max(B[n], B[m]);
34         C[n] = max(C[n], C[m]);
35         for (int i = l; i <= r; i++)
36             Sum[n] += v[i];
37     }
38 }
39
40 int main()
41 {
42     ifstream fin(FIN);
43     ofstream fout(FOUT);
44     int n, l, r, q;
45     fin >> n >> l >> r >> q;
46     build(0, l, r);
47     while (q--)
48     {
49         fin >> l >> r;
50         cout << max(A[0], A[r]) << endl;
51     }
52 }
```

```

26         Sum[n] = v[1];
27     }
28     else
29     {
30         build(lf, l, md);
31         build(rt, md+1, r);
32
33         A[n] = max(A[lf], Sum[lf]+A[rt]);
34         B[n] = max(B[lf]+Sum[rt], B[rt]);
35         C[n] = max(max(C[lf], C[rt]), B[lf]+A[rt]);
36
37         Sum[n] = Sum[lf]+Sum[rt];
38     }
39 }
40
41 int p, x;
42
43 void update(int n, int l, int r) // p, x
44 {
45     if (l == r)
46     {
47         A[n] = B[n] = C[n] = max(x, 0);
48         Sum[n] = x;
49     }
50     else
51     {
52         if (p <= md)
53             update(lf, l, md);
54         else
55             update(rt, md+1, r);
56
57         A[n] = max(A[lf], Sum[lf]+A[rt]);
58         B[n] = max(B[lf]+Sum[rt], B[rt]);
59         C[n] = max(max(C[lf], C[rt]), B[lf]+A[rt]);
60
61         Sum[n] = Sum[lf]+Sum[rt];
62     }
63 }
64
65 int a, b;
66 long long ans, S;
67
68 void query(int n, int l, int r) // a, b
69 {
70     if (a <= l && r <= b)
71     {
72         if (S < 0) S = 0;
73         ans = max(ans, max(S+A[n], C[n]));
74         S = max(S+Sum[n], B[n]);
75     }
76     else
77     {
78         if (a <= md) query(lf, l, md);
79         if (b > md) query(rt, md+1, r);
80     }
81 }
82
83 int main()
84 {
85     int i, t, v1, v2;
86
87     freopen(FIN, "r", stdin);
88     freopen(FOUT, "w", stdout);
89
90     scanf("%d", &N);
91     for (i = 1; i <= N; ++i)
92         scanf("%d", v+i);
93
94     build(1, 1, N);
95
96     scanf("%d", &M);
97     while (M--)
98     {
99         scanf("%d %d %d", &t, &v1, &v2);
100        if (t == 0)
101            { // update

```

```

102         p = v1+1;
103         x = v2;
104         update(1, 1, N);
105     }
106     else
107     { // query
108         S = ans = 0;
109         a = v1+1;
110         b = v2+1;
111         query(1, 1, N);
112         printf("%lld\n", ans);
113     }
114 }
115
116 return 0;
117 }
```

31.5.3 *Rezolvare detaliată

31.6 Tric

Din cei n 参 品 計 參 ン la olimpiada de informatică se pot distinge m perechi de prieteni. Aceste perechi au câteva proprietăți interesante:

- dacă A este prieten cu B , atunci B este prieten cu A ;
- dacă A_1 este prieten cu A_2 , A_2 cu A_3 , ..., A_{k-1} cu A_k și A_k cu A_1 , atunci există cel puțin o pereche (i, j) , $1 \leq i, j \leq k$, astfel încât:
 - A_i și A_j sunt prieteni și
 - $(i \bmod k) + 1 \neq j$ și $(j \bmod k) + 1 \neq i$

Se numește triunghi de prieteni un set de 3 prieteni A , B și C , cu proprietatea că A este prieten cu B , B cu C și C cu A .

Cerință

Aflați câte triunghiuri de prieteni există.

Date de intrare

Pe prima linie a fișierului de intrare **tric.in** se găsesc, separate prin spații, numerele naturale n și m . Pe următoarele m linii se găsesc perechi de numere A B , între 0 și $n - 1$, cu semnificația că A este prieten cu B .

Date de ieșire

Pe singura linie a fișierului de ieșire **tric.out** afișați numărul de triunghiuri de prieteni.

Restricții și precizări

- $2 \leq n \leq 100000$
- $1 \leq m \leq 100000$
- pentru 20% din teste, $n \leq 300$
- pentru 50% din teste, $n \leq 1000$

Exemplu

tric.in	tric.out
4 4	1
0 1	
1 2	
2 0	
2 3	

Limită de memorie: 64 Mb, din care 1 Mb pentru stivă.

Timp maxim de execuție/test: 0.4 secunde

31.6.1 Indicații de rezolvare

Stefan Ciobâcă

Problema cere găsirea tuturor ciclurilor de grad 3 din graful dat la intrare. Graful acesta este cordal (fiecare ciclu de lungime 4 trebuie să aibă cel puțin o coardă), fapt care ne permite să numărăm triunghiurile în timp $O(n + m)$.

Se realizează o ordonare a nodurilor v_1, v_2, \dots, v_n , cu proprietatea că v_i , împreună cu toții vecinii săi care se găsesc la dreapta lui în ordonare, formează o clică. Deoarece graful este cordal, această ordonare se poate realiza întotdeauna. Odată stabilită această ordine, pentru orice i , putem număra triunghiurile formate de nodul v_i împreună cu nodurile $v_{i+1}, v_{i+2}, \dots, v_n$ în timp $O(1)$ (dacă t e numărul de vecini ai lui v_i aflați la dreapta acestuia, sunt $t * (t - 1)/2$ astfel de triunghiuri).

Pentru a găsi ordonarea, există doi algoritmi cunoscuți: *LexBFS* și *MCS*. Cel mai simplu este *MCS* și funcționează astfel:

Alegem nodurile din ordonare pe rând, de la v_n la v_1 .

- asociem fiecărui nod numărul de vecini aflați la stânga sa, initializând aceste valori cu 0
- presupunând că am ales $v_{i+1}, v_{i+2}, \dots, v_n$, alegem v_i ca fiind nodul cu valoarea asociată cea mai mare
- incrementăm valoarea asociată tuturor vecinilor lui v_i

Algoritmul se poate implementa în $O(n + m)$, dacă se țin toate nodurile care au aceeași valoare asociată într-un bucket.

Se poate demonstra că acest algoritm găsește o ordonare cu proprietatea de mai sus (vezi Tarjan și Yannakakis, "Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs").

31.6.2 Cod sursă

Listing 31.6.1: tric.cpp

```

1 // #include <iostream>
2 // #include <fstream>
3 #include <cstdio>
4 #include <cassert>
5 #include <ctime>
6 #include <cstring>
7
8 using namespace std;
9
10 const int MAX_NODES = 100000;
11
12 int adj[MAX_NODES];
13 int used[MAX_NODES];
14 int *adjNodes[MAX_NODES];
15 int adjNodeCount[MAX_NODES];
16
17 int peo[MAX_NODES];
18
19 struct dd
20 {
21     int x;
22     struct dd *next;
23     struct dd *prev;
24
25     dd(int info) : x(info) {}
26 };
27
28 struct dd *buckets[MAX_NODES]; // buckets[i]=linked list all nodes with degree i
29 struct dd *where[MAX_NODES]; // node i is here
30
31 int degree[MAX_NODES]; // what degree is node i
32 int oldbest = 0;
33
34 bool isEdge(int a, int b)
35 {
36     for (int i = 0; i < adjNodeCount[a]; ++i)
37     {

```

```

38     if (adjNodes[a][i] == b)
39     {
40         return true;
41     }
42 }
43 return false;
44 }
45
46 void init(int n)
47 {
48     dd *old = 0;
49     for (int i = 0; i < n; ++i)
50     {
51         degree[i] = 0;
52         dd *t = new dd(i);
53         where[i] = t;
54         t->next = old;
55         t->prev = 0;
56         if (old)
57         {
58             old->prev = t;
59         }
60         old = t;
61     }
62
63     buckets[0] = old;
64 }
65
66 void delete_stuff(dd **list, dd *cell)
67 {
68     if (cell->prev == 0)
69     {
70         *list = cell->next;
71         if (cell->next)
72         {
73             cell->next->prev = 0;
74         }
75     }
76     else
77     {
78         cell->prev->next = cell->next;
79         if (cell->next)
80         {
81             cell->next->prev = cell->prev;
82         }
83     }
84 }
85
86 void add(int key)
87 {
88     dd *d = new dd(key);
89     where[key] = d;
90     d->next = buckets[degree[key]];
91     d->prev = 0;
92     if (buckets[degree[key]])
93     {
94         buckets[degree[key]]->prev = d;
95         buckets[degree[key]] = d;
96     }
97     else
98     {
99         buckets[degree[key]] = d;
100    }
101 }
102
103 void increase(int key)
104 {
105     delete_stuff(&buckets[degree[key]], where[key]);
106     degree[key]++;
107     if (degree[key] > oldbest)
108     {
109         oldbest = degree[key];
110     }
111
112     add(key);
113 }
```

```

114
115     int best()
116    {
117        while (!buckets[oldbest])
118        {
119            oldbest--;
120        }
121        int result = buckets[oldbest]->x;
122        delete_stuff(&buckets[oldbest], buckets[oldbest]);
123        return result;
124    }
125
126    int nodeCount, edgeCount;
127
128    int main()
129    {
130        FILE *fi = fopen("tric.in", "r");
131        FILE *fo = fopen("tric.out", "w");
132
133        memset(adjNodeCount, 0, sizeof(adjNodeCount));
134
135        fscanf(fi, "%d %d", &nodeCount, &edgeCount);
136        for (int i = 0; i < edgeCount; ++i)
137        {
138            int a, b;
139            fscanf(fi, "%d %d", &a, &b);
140            adjNodeCount[a]++;
141            adjNodeCount[b]++;
142        }
143
144        for (int i = 0; i < nodeCount; ++i)
145        {
146            adjNodes[i] = new int [adjNodeCount[i]];
147        }
148
149        memset(adjNodeCount, 0, sizeof(adjNodeCount));
150
151        rewind(fi);
152        fscanf(fi, "%d %d", &nodeCount, &edgeCount);
153        for (int i = 0; i < edgeCount; ++i)
154        {
155            int a, b;
156            fscanf(fi, "%d %d", &a, &b);
157            adjNodes[a][adjNodeCount[a]++] = b;
158            adjNodes[b][adjNodeCount[b]++] = a;
159        }
160
161        init(nodeCount);
162        memset(adj, 0, sizeof(adj));
163        memset(used, 0, sizeof(used));
164
165        long long result = 0;
166        for (int i = nodeCount - 1; i >= 0; --i)
167        {
168            int bestNode = best();
169            peo[i] = bestNode;
170            used[bestNode] = 1;
171
172            int count = 0;
173            for (int j = 0; j < adjNodeCount[bestNode]; ++j)
174            {
175                int toInc = adjNodes[bestNode][j];
176                if (!used[toInc])
177                {
178                    increase(toInc);
179                }
180                else
181                {
182                    count++;
183                }
184            }
185            result += (count - 1) * count / 2;
186        }
187    }
188
189    fprintf(fo, "%lld\n", result);

```

```
190  
191     fclose(fi);  
192     fclose(fo);  
193 }
```

31.6.3 *Rezolvare detaliată

Capitolul 32

ONI 2006 clasa a XI-a



Figura 32.1: Sigla ONI 2006

32.1 borg

Oricine a urmărit serialul Star Trek își aduce aminte de borgi și de nava lor spațială în formă de cub. Una dintre problemele pe care și-au pus-o înainte de a construi nava a fost următoarea.

Nava borgilor are forma unui paralelipiped dreptunghic de dimensiuni $N \times M \times H$, împărțit în camere de dimensiune $1 \times 1 \times 1$. Pentru ca nava să poată funcționa, în aceste camere trebuie plasate K motoare de propulsie, în fiecare cameră putându-se plasa cel mult un motor.

O cameră poate fi identificată printr-un triplet (a, b, c) , unde $1 \leq a \leq N$, $1 \leq b \leq M$, $1 \leq c \leq H$, reprezentând coordonatele sale.

Un plan al paralelipipedului este o mulțime de camere de unul dintre următoarele 3 tipuri:

$\{(a, b, c) | a$ fixat, $1 \leq b \leq M$, $1 \leq c \leq H\}$ - în total sunt N plane de acest tip;

$\{(a, b, c) | b$ fixat, $1 \leq a \leq N$, $1 \leq c \leq H\}$ - în total sunt M plane de acest tip;

$\{(a, b, c) | c$ fixat, $1 \leq a \leq N$, $1 \leq b \leq M\}$ - în total sunt H plane de acest tip.

Cerință

Se cere să se găsească R , numărul de moduri în care se pot plasa cele K motoare, astfel încât orice plan al paralelipipedului să conțină cel puțin o cameră ocupată de un motor.

Deoarece numărul cerut poate fi foarte mare, este suficient să aflați restul împărțirii lui R la 30103.

Date de intrare

Fișierul de intrare **borg.in** va conține o singură linie pe care sunt scrise numerele naturale N , M , H și K separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire **borg.out** va conține o singură linie pe care va fi scris un singur număr natural reprezentând restul împărțirii lui R la 30103.

Restricții și precizări

- $1 \leq N, M, H \leq 20$
- $1 \leq K \leq N * M * H$
- 80% din teste vor avea $K \leq 2000$

Exemplu

borg.in	borg.out	borg.in	borg.out
3 1 2 4	12	3 1 2 2	0

Timp maxim de execuție/test: 0.7 secunde

32.1.1 Indicații de rezolvare

Soluția oficială

Numărul total de moduri în care se pot plasa K motoare în paralelipiped, fără restricția ca fiecare plan să conțină cel puțin un motor, este $C_{N \cdot M \cdot H}^K$. Din acestea, trebuie să le scădem pe cele care nu sunt bune.

Dacă notăm cu X_i ($1 \leq i \leq N$) numărul de moduri în care se pot plasa cele K motoare astfel încât planul i (de tip 1) să fie gol, cu Y_i ($1 \leq i \leq M$) numărul de moduri astfel încât planul i (de tip 2) să fie gol și cu Z_i ($1 \leq i \leq H$) numărul de moduri astfel încât planul i (de tip 3) să fie gol, atunci numărul căutat va fi

$$C_{N \cdot M \cdot H}^K - \left| \bigcup_{1 \leq i \leq N} X_i \cup \bigcup_{1 \leq i \leq M} Y_i \cup \bigcup_{1 \leq i \leq H} Z_i \cup \right|.$$

Cardinalul reuniunii se poate dezvolta cu *principiul includerii și excluderii*, fiecare mulțime rezultată conținând o intersecție de a mulțimi X , b mulțimi Y și c mulțimi Z .

Cardinalul unei astfel de mulțimi ar reprezenta numărul de moduri în care se pot pune K motoare, astfel încât a dintre planele de tip 1 să fie goale, b dintre planele de tip 2 să fie goale și c dintre planele de tip 3 să fie goale, adică ar fi $C_{(N-a) \cdot (M-b) \cdot (H-c)}^K$. În dezvoltarea produsă de principiul includerii și excluderii vor fi $C_N^a \cdot C_M^b \cdot C_H^c$ astfel de mulțimi.

Așadar numărul total va fi

$$\sum_{a=1}^N \sum_{b=1}^M \sum_{c=1}^H (-1)^{a+b+c} C_N^a \cdot C_M^b \cdot C_H^c \cdot C_{(N-a) \cdot (M-b) \cdot (H-c)}^K$$

Se observă că este nevoie de toate combinările până la maximul dintre N , M și H , dar numai de cele care au K sus pentru restul. Acestea se pot calcula în complexitatea $O(N \cdot M \cdot H \cdot K)$ folosind regula generală $C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$ ($k \leq K$ și $n \leq N * M * H$). Pentru a reduce complexitatea la $O(N \cdot M \cdot H)$, putem calcula C_n^K direct din C_{n-1}^K , folosindu-ne de faptul că numărul 30103 este prim și deci orice număr are un *invers modular* (care înmulțit cu el dă restul 1 la împărțirea la 30103).

32.1.2 Cod sursă

Listing 32.1.1: borg.cpp

```

1  /* Tiberiu Danet, 2006 */
2  #include <cstdio>
3  #include <cstring>
4
5  #define N_MAX 32
6  #define MAGIC 30103
7
8  int N, M, H, K, Res;
9
10 // combinările care au N, M, H jos
11 int combN[N_MAX], combM[N_MAX], combH[N_MAX];
12
13 // combinările care au K sus
14 int combK[N_MAX * N_MAX * N_MAX];
15
16 int comb[N_MAX * N_MAX * N_MAX];
17
18 void compute_comb ()
19 {
20     int i, j, start;
21
22     comb[0] = 1;

```

```

23     for (i = 1; i <= N * M * H; i++)
24     {
25         start = (i > N && i > M && i > H && K < i) ? K : i;
26
27         for (j = start; j >= 1; j--)
28             comb[j] = (comb[j] + comb[j - 1]) % MAGIC;
29
30         combK[i] = comb[K];
31
32         if (i == N)
33             memccpy(combN, comb, -1, (N + 1) * sizeof(int));
34
35         if (i == M)
36             memccpy(combM, comb, -1, (M + 1) * sizeof(int));
37
38         if (i == H)
39             memccpy(combH, comb, -1, (H + 1) * sizeof(int));
40     }
41 }
42
43 void solve ()
44 {
45     int i, j, k, r, sgn;
46
47     if (K > N * M * H)
48         return;
49
50     for (i = 0; i <= N; i++)
51         for (j = 0; j <= M; j++)
52             for (k = 0; k <= H; k++)
53             {
54                 sgn = ((i + j + k) % 2 == 0) ? 1 : -1;
55                 r = (combN[i] * combM[j]) % MAGIC;
56                 r = (r * combH[k]) % MAGIC;
57                 r = (r * combK[(N - i) * (M - j) * (H - k)]) % MAGIC;
58                 r = (MAGIC + sgn * r) % MAGIC;
59                 Res = (Res + r) % MAGIC;
60             }
61     }
62
63 void read_solve ()
64 {
65     freopen("borg.in", "r", stdin);
66     freopen("borg.out", "w", stdout);
67
68     scanf("%d%d%d%d", &N, &M, &H, &K);
69
70     compute_comb();
71     solve();
72
73     printf("%d\n", Res);
74 }
75
76 int main ()
77 {
78     read_solve();
79     return 0;
80 }
```

Listing 32.1.2: borg2.cpp

```

1  /* Emilian Miron */
2  #include <cstdio>
3  #include <cassert>
4  #include <algorithm>    // std::max
5
6  using namespace std;
7
8  #define maxn 21
9  #define prime 30103
10
11 int invmod (int x)
12 {
13     int p = prime - 2, rv = 1, oldx = x;
14     for (; p; p /= 2)
```

```

15      {
16          rv = p & 1 ? x * rv % prime : rv;
17          x = x*x % prime;
18      }
19
20      assert ((rv * oldx % prime) == 1);
21      return rv;
22 }
23
24 int C[maxn][maxn], N, M, H, K, CM[maxn*maxn*maxn];
25
26 int main()
27 {
28     int i, j, k, MAX, sol;
29
30     freopen ("borg.in", "rt", stdin);
31     freopen ("borg.out", "wt", stdout);
32
33     scanf ("%d%d%d%d", &N, &M, &H, &K);
34
35 //MAX = N >? M >? H;
36 MAX = max(N,M), H;
37
38     assert (MAX < maxn && K <= N*M*H);
39
40     C[0][0] = 1;
41
42     for (i = 1; i <= MAX; i++)
43     {
44         C[i][0] = 1;
45         for (j = 1; j <= MAX; j++)
46             C[i][j] = (C[i-1][j-1] + C[i-1][j]) % prime;
47     }
48
49     CM[K] = 1;
50     for (k = K + 1; k <= N*M*H; k++)
51         CM[k] = CM[k - 1]*k%prime*invmod(k - K)%prime;
52
53     for (sol = 0, i = 0; i < N; i++)
54     for (j = 0; j < M; j++)
55     for (k = 0; k < H; k++)
56     {
57         sol = sol + C[N][i]* C[M][j]%prime*C[H][k]%prime
58             * CM[(N-i)*(M-j)*(H-k)] % prime *
59             (((i+j+k)%2) ? -1 : 1);
60         for (; sol < 0; sol += prime);
61         sol %= prime;
62     }
63
64     fprintf (stderr, "K, NMH: %d %d\n", K, N*M*H);
65     printf ("%d\n", sol);
66
67     return 0;
68 }

```

32.1.3 *Rezolvare detaliată

32.2 diamant

O firmă produce un tip nou de diamante de formă dreptunghiulară și de calități diferite. Pentru a calcula calitatea unui diamant firma împarte diamantul în $N \times M$ pătrățele formând o matrice cu N linii numerotate de la 1 la N și M coloane numerotate de la 1 la M . Pătrățelul de pe linia i și coloana j poate influența calitatea diamantului în felul următor ($1 \leq i \leq N, 1 \leq j \leq M$):

- dacă pătrățelul conține impurități este marcat cu -1 și va diminua calitatea diamantului cu $i * j$
- dacă pătrățelul este simplu este marcat cu 0 și nu schimbă calitatea diamantului
- dacă pătrățelul conține aur este marcat cu $+1$ și va mări calitatea diamantului cu $i * j$.

Fiecare pătrățel va fi marcat cu unul dintre cele trei numere $(-1, 0, +1)$.

Un client bogat vrea să cumpere cât mai multe diamante diferite, de aceeași calitate X . Două diamante sunt diferite dacă există cel puțin un pătrățel de pe o linie i și coloană j marcat diferit în cele două diamante.

Cerință

Ajuatați firma să poată răspunde la astfel de cereri scriind un program care pentru un anumit X găsește numărul de diamante diferite de calitate X .

Date de intrare

Pe prima linie a fișierului de intrare **diamant.in** sunt scrise trei numere întregi $N \ M \ X$ separate prin câte un spațiu reprezentând numărul de linii, numărul de coloane ale unui diamant, și respectiv calitatea cerută.

Date de ieșire

Pe prima linie din fișierul de ieșire **diamant.out** se va afla numărul de diamante diferite cu calitatea cerută, modulo 10000.

Restricții și precizări

- $0 < N < 21$
- $0 < M < 21$
- $-2^{31} < X < 2^{31}$

Exemplu

diamant.in	diamant.out	Explicatii
2 2 7	3	Matricile corespunzătoare celor 3 diamante sunt: $\begin{matrix} -1 & +1 & +1 & 0 & +1 & +1 \\ +1 & +1 & +1 & +1 & 0 & +1 \end{matrix}$

Timp maxim de execuție/test: 2 secunde

32.2.1 Indicații de rezolvare

Soluția oficială

Soluția se bazează pe faptul ca valoarea maximă în modul a lui X este

$$1 * 1 + 1 * 2 + \dots + 1 * 19 + 1 * 20 + 2 * 1 + 2 * 2 + \dots 2 * 20 + 3 * 1 + \dots + 19 * 20 + 20 * 20$$

în cazul în care valorile lui N și M sunt maxime adică 20 și avem $+1$ (sau -1) în toată matricea. Suma are valoarea 44100.

Odată observat acest fapt se poate deduce că o soluție asemănătoare cu cea a *problemei rucsacului* se încadrează în limita de timp. Capacitatea rucsacului ar fi X iar obiectele ar fi pătrățelele, greutatea fiind $i * j$ pentru pătrățelul de pe linia i coloana j .

32.2.2 Cod sursă

Listing 32.2.1: diamant.cpp

```

1 #include <cstdio>
2 #include <cstring>
3 #include <ctime>
4
5 using namespace std;
6
7 int An[1<<16];
8 int Ap[1<<16];
9
10 int Bn[1<<16];
11 int Bp[1<<16];
12
13 int i,j,n,m,k;
14
15 #define A(i) (((i)<0)?(An[-(i)]):(Ap[(i)]))
16 #define B(i) (((i)<0)?(Bn[-(i)]):(Bp[(i)]))
17
18 #define li -50000
19 #define ls 50000
20
21 int main()

```

```

22 {
23     long t1=clock();
24
25     freopen ("diamant.in", "r", stdin);
26     freopen ("diamant.out", "w", stdout);
27
28     scanf ("%d%d%d", &n, &m, &k);
29
30     if (k>ls)
31     {
32         printf ("0\n");
33         return 0;
34     }
35
36     A(0)=1;
37
38     for (i=1; i<=n; i++)
39         for (j=1; j<=m; j++)
40     {
41         memcpy (Bn, An, sizeof (An));
42         memcpy (Bp, Ap, sizeof (Bp));
43         //+
44         for (int t=ls; t>=li; t--)
45         {
46             A(t)=B(t-i*j)+B(t)+B(t+i*j);
47             A(t)%=10000;
48         }
49     }
50
51     printf ("%d\n", A(k));
52     return 0;
53 }
54 }
```

32.2.3 *Rezolvare detaliată

32.3 matrice

Fie A o matrice dreptunghiulară de numere întregi cu N linii numerotate de la 1 la N și M coloane numerotate de la 1 la M . În matricea A oricare două elemente consecutive de pe aceeași linie sunt distincte.

Se definește un sir valid de numere întregi ca fiind fie un sir crescător, fie un sir descrescător, fie un sir crescător concatenat cu un sir descrescător, fie un sir descrescător concatenat cu unul crescător. Exemple de siruri valide sunt: 1 2 3 7, 8 5 2 1, 3 5 6 2, 4 1 5 6.

Se definește o submatrice a lui A de coordonate (l_1, c_1, l_2, c_2) ca fiind matricea formată din toate elementele $A(i, j)$, cu $l_1 \leq i \leq l_2$ și $c_1 \leq j \leq c_2$.

O submatrice a lui A este validă dacă liniile sale sunt siruri valide.

Atenție! O submatrice validă poate avea pe o linie un sir crescător de numere, pe a doua un sir descrescător, pe a treia un sir crescător concatenat cu unul descrescător etc. Deci, liniile unei submatrice valide nu trebuie să fie neapărat siruri de același tip.

Aria unei submatrice este egală cu numărul de elemente din care este formată submatricea.

Cerință

Se cere să se găsească o submatrice validă a lui A de arie maximă.

Date de intrare

Pe prima linie a fișierului de intrare **matrice.in** se află numerele N și M , separate prin spațiu.

Pe fiecare dintre următoarele N linii se află câte M numere întregi separate prin câte un spațiu, reprezentând elementele matricei A .

Date de ieșire

Fișierul de ieșire **matrice.out** va conține o singură linie pe care vor fi scrise coordonatele l_1 , c_1 , l_2 , c_2 (în această ordine și separate prin câte un spațiu) ale unei submatrice valide de arie maximă. În cazul în care există mai multe soluții cu arie maximă, se va afișa oricare dintre ele.

Restricții și precizări

- $1 \leq N, M \leq 1000$
- 70% din teste vor avea $N, M \leq 600$

- Elementele matricei A sunt numere întregi din intervalul $[-30000, 30000]$.

Exemplu

matrice.in	matrice.out	Explicatii
2 6	1 1 2 3	Aria maximă este 6.
1 2 5 7 9 10		O altă soluție de arie maximă ar putea fi
3 4 3 5 1 10		1 1 1 6 sau 1 2 2 4 sau 1 3 2 5 sau 1 4 2 6

Timp maxim de execuție/test: 1 secundă

32.3.1 Indicații de rezolvare

Soluția oficială

Se parcurg în ordine coloanele matricei. Pentru o coloană j se calculează pentru fiecare linie i o valoare $Left[i][j]$ ca fiind lungimea maximă a unui sir valid de pe linia i care are ultimul element pe coloana j . $Left[i][j]$ se poate determina din $Left[i][j - 1]$ memorând ultima poziție (coloană) de pe fiecare linie unde sirul și-a schimbat monotonia (deoarce două elemente consecutive de pe o linie sunt distințe, sirurile nu vor putea fi decât strict crescătoare/descrescătoare). Ca o observație, este suficient să memorăm doar ultima coloană a acestei matrice $Left$.

Problema revine la a afla submatricea de arie maximă care are coloana din dreapta j , știind pentru fiecare linie lungimea maximă a unui sir valid de pe linia i care se termină pe coloana j ($Left[i][j]$).

Acest lucru se poate face în $O(N)$ pentru fiecare coloană, ținând o stivă de elemente crescătoare. De exemplu, pentru o matrice cu 4 linii considerăm că pentru o coloană j am calculat următoarele valori $Left[i][j] : 3, 4, 1, 2$. Introducem 3 și 4 în stivă, iar când introducem 1 va trebui să scoatem din stivă pe 4 și apoi pe 3. De fiecare dată când scoatem un număr din stivă vedem ce dreptunghiuri pot apărea care să îl conțină.

Această soluție are complexitatea $O(M * N)$ și obține 100 puncte.

O soluție care pentru fiecare coloană, având calculate valorile $Left[i][j]$, rezolvă problema în $O(N^2)$ (deci are complexitate totală $O(M * N^2)$) va obține 70 puncte.

O soluție care pentru fiecare coloană calculează de fiecare dată valorile $Left[i][j]$ (mergând în stânga atât cât este nevoie), deci având complexitatea totală $O(N^2 * M^2)$, va obține 40 puncte.

32.3.2 Cod sursă

Listing 32.3.1: matrice.cpp

```

1 #include <cstdio>
2
3 #define N_MAX          1024
4 #define SEQ_UNKNOWN     0
5 #define SEQ_INCREASING 1
6 #define SEQ_DECREASING 2
7
8 int N, M;
9 short A[N_MAX][N_MAX];
10
11 int Height[N_MAX], Type[N_MAX], Start[N_MAX];
12 int Count, Stack[N_MAX], Top[N_MAX];
13
14 int Best, Best1l, Best1c, Best2l, Best2c;
15
16 void solve ()
17 {
18     int i, j, top;
19
20     for (j = 0; j < M; j++)
21     {
22         for (i = 0; i <= N; i++)
23         {
24             // Get the height of the column
25             if (i == N)
26                 Height[i] = 0;
27             else if (j == 0)
28                 Height[i] = 1;
29             else if (Type[i] == SEQ_UNKNOWN)

```

```

30         {
31             Type[i]=(A[i][j]>A[i][j-1]) ? SEQ_INCREASING : SEQ_DECREASING;
32             Height[i]++;
33         }
34     else if (Type[i] == SEQ_INCREASING)
35     {
36         if (A[i][j] < A[i][j - 1])
37         {
38             Height[i] = j - Start[i] + 1;
39             Start[i] = j - 1;
40             Type[i] = SEQ_DECREASING;
41         }
42     else
43         Height[i]++;
44     }
45     else if (Type[i] == SEQ_DECREASING)
46     {
47         if (A[i][j] > A[i][j - 1])
48         {
49             Height[i] = j - Start[i] + 1;
50             Start[i] = j - 1;
51             Type[i] = SEQ_INCREASING;
52         }
53     else
54         Height[i]++;
55     }
56
57 // Do the stack thing
58 top = i;
59 while (Count > 0 && Height[i] <= Stack[Count - 1])
60 {
61     top = Top[Count - 1];
62     if (Stack[Count - 1] * (i - top) > Best)
63     {
64         Best = Stack[Count - 1] * (i - top);
65         Best1l = top;
66         Best2l = i - 1;
67         Best1c = j - Stack[Count - 1] + 1;
68         Best2c = j;
69     }
70     Count--;
71 }
72
73 if (Height[i] > 0)
74 {
75     Stack[Count] = Height[i];
76     Top[Count++] = top;
77 }
78 }
79 }
80 }
81 printf("%d %d %d %d\n", Best1l + 1, Best1c + 1, Best2l + 1, Best2c + 1);
82 }
83
84 void read_solve ()
85 {
86     int i, j;
87
88     freopen("matrice.in", "r", stdin);
89     freopen("matrice.out", "w", stdout);
90
91     scanf("%d%d", &N, &M);
92     for (i = 0; i < N; i++)
93         for (j = 0; j < M; j++)
94             scanf("%d", &A[i][j]);
95
96     solve();
97 }
98
99
100 int main ()
101 {
102     read_solve();
103     return 0;
104 }
```

32.3.3 *Rezolvare detaliată

32.4 Petrom

Firma Petrom are n benzinării amplasate de-a lungul autostrăzii Soarelui. Benzinăriile sunt numerotate de la 1 la n în ordinea amplasării pe autostradă.

Pozиїile benzinăriilor sunt cunoscute, fiind specificate prin distanțele d_1, d_2, \dots, d_n (d_i reprezintă distanța de la sediul firmei Petrom până la benzinăria i).

Sediul firmei Petrom este amplasat la intrarea pe autostrada Soarelui.

În k dintre aceste benzinării firma dorește să amenajeze depozite de combustibil, care să alimenteze benzinăria respectivă, dar și alte benzinării învecinate. Fiecare benzinărie va fi alimentată de la depozitul cel mai apropiat.

Costul de transport pentru o amplasare a depozitelor dată va fi egal cu suma distanțelor de la fiecare benzinărie la depozitul de la care se alimentează.

Cerință

Scriți un program care să determine benzinăriile în care trebuie să construite depozite, astfel încât costul de transport să fie minim.

Date de intrare

Fișierul de intrare **petrom.in** conține pe prima linie două numere naturale separate prin spațiu n și k , reprezentând numărul de benzinării și respectiv numărul de depozite care trebuie construite.

Pe următoarele n linii sunt scrise n numere naturale d_1, d_2, \dots, d_n , câte un număr pe o linie, reprezentând distanțele de la sediul Petrom la cele n benzinării.

Date de ieșire

Fișierul de ieșire **petrom.out** va conține pe prima linie costul de transport (minim posibil).

Pe următoarele k linii sunt scrise benzinăriile în care vor fi construite depozite pentru a obține costul minim, câte o benzinărie pe o linie.

Restricții și precizări

$$1 \leq n \leq 400$$

$$1 \leq k \leq 300$$

$$k \leq n$$

$$0 < d_1 < d_2 < \dots < d_n \leq 30000$$

Dacă există mai multe soluții potrivite determină oricare dintre acestea.

Pentru fiecare test, se acordă 40% din punctaj pentru determinarea costului minim și 100% pentru rezolvarea integrală.

Exemplu

petrom.in	petrom.out	Explicatii
6 3	8	Depozitul 1 va fi construit în benzinăria 2 și alimentează benzinăriile 1, 2, 3. (Cost: $1+0+6=7$)
5	2	
6	4	
12	6	Depozitul 2 va fi construit în benzinăria 4 și alimentează benzinăriile 4, 5. (Cost: $0+1=1$)
19		
20		Depozitul 3 va fi construit în benzinăria 6 și alimentează benzinăriile 6. (Cost: 0)
27		

Timp maxim de execuție/test: 0.2

32.4.1 Indicații de rezolvare

Soluția oficială

Vom precalcula mai întâi o matrice cost:

$cost[a][b] =$ costul de transport pentru cazul în care un depozit alimentează benzinăriile din intervalul $[a, b]$, $a < b$

$$cost[a][a] = 0$$

Costul optim se obține amplasând un depozit în benzinăria situată în mijlocul intervalului.

Soluția problemei se obține prin programare dinamică.

Subprobleme:

Să se determine o amplasare optimă a i depozite în benzinăriile $1, \dots, j$

Vom memora soluțiile subproblemelor în matricea c_{min} :

$cmin[i][j]$ = costul minim de transport în condițiile în care construim i depozite în benzinăriile $1, \dots, j$

Relația de recurență:

$cmin[1][j] = cost[1][j]$, pentru $j = 1, \dots, n$

$cmin[i][j] = \min\{cmin[i-1][p-1] + cost[p][j] | p = i, i+1, \dots, j\}$

Pentru reconstituirea soluției vom utiliza o matrice *back*

$back[i][j]$ = poziția p pentru care se obține costul minim $cmin[i][j]$

32.4.2 Cod sursă

Listing 32.4.1: petrom.c

```

1 #include <stdio.h>
2 #include <math.h>
3
4 #define InFile "petrom.in"
5 #define OutFile "petrom.out"
6 #define INFINITE 30000
7 #define NMAX 1001
8 #define KMAX 301
9
10 int d[NMAX];
11 int cmin[KMAX][NMAX];
12 int cost[NMAX][NMAX];
13 int k, n;
14 int sol[KMAX+1];
15
16 void det_cost(void);
17 void rezolvare(void);
18 void citire(void);
19 void afisare(void);
20
21 int main()
22 {
23     citire();
24     rezolvare();
25     afisare();
26     return 0;
27 }
28
29 void det_cost()
30 {
31     int i, j, l, med, err;
32     /* cost[a][b] = costul de transport pentru cazul in care un depozit
33        alimenteaza benzinariile din intervalul [a,b], a<b
34        cost[a][a]=0
35        costul optim se obtine amplasand un depozit in benzinaria
36        situata in mijlocul intervalului */
37
38     for (i=1; i<=n; i++)
39         for (j=i+1; j<=n; j++)
40     {
41         med = d[(i+j)/2];
42         err = 0;
43         for (l=i; l<=j; l++)
44             err += abs(d[l] - med);
45         cost[i][j] = err;
46     }
47 }
48
49 void rezolvare()
50 {
51     int i, j, p;
52     short back[KMAX][NMAX];
53     det_cost();
54
55     /* Metoda: programare dinamica.
56     Subproblema:
57     Sa se determine o amplasare optima a i depozite in benzinariile 1, ..., j
58
59     cmin[i][j]= costul minim de transport in conditiile in care construim
60     i depozite in benzinariile 1, ..., j

```

```

61     Relatia de recurenta:
62     cmin[1][j]=cost[1][j], pentru j=1, ..., n
63     cmin[i][j]=min {cmin[i-1][p-1]+cost[p][j] | p=i, i+1, ..., j}
64
65     Pentru reconstituirea solutiei vom utiliza o matricea back
66     back[i][j]= pozitia p pentru care se obtine costul minim cmin[i][j]
67
68     Solutia problemei initiale se afla in cmin[k][n]
69
70     */
71
72     for (j=1; j<=n; j++)
73     {
74         cmin[1][j] = cost[1][j];
75         back[1][j] = 1;
76     }
77
78
79     for (i=2; i<=k; i++)
80         for (j=1; j <= n; j++)
81         {
82             cmin[i][j] = INFINITE;
83             for (p=i; p<=j; p++)
84                 if (cost[p][j]+cmin[i-1][p-1] < cmin[i][j])
85                 {
86                     cmin[i][j] = cost[p][j] + cmin[i-1][p-1];
87                     back[i][j] = p;
88                 }
89         }
90
91     sol[k+1]=n+1;
92     for (i=k, j=n; i>0; i--)
93     {
94         sol[i] = back[i][j];
95         j=sol[i]-1;
96     }
97 }
98 }
99
100 void citire()
101 {
102     int i;
103     FILE * fin = fopen( InFile, "r" );
104     fscanf(fin, "%d %d", &n, &k);
105     for(i=1;i<=n;i++)
106         fscanf(fin, "%d", d+i);
107     fclose(fin);
108 }
109
110 void afisare()
111 {
112     int i, last;
113     FILE * fout=fopen(OutFile, "w");
114     fprintf(fout, "%d\n", cmin[k][n]);
115     i=k;
116     for (i=1; i<=k; i++)
117     {
118         last=sol[i+1]-1;
119         fprintf(fout, "%d\n", (sol[i]+last)/2);
120     }
121     fclose(fout);
122 }
123 }
```

32.4.3 *Rezolvare detaliată

32.5 ratina

Limba ratină are doar N cuvinte, numerotate de la 1 la N . Două sau mai multe cuvinte se numesc k -asemenea dacă au primele k litere identice.

Gradul de asemănare între t cuvinte este k dacă cele t cuvinte sunt k -asemenea, dar nu sunt $(k + 1)$ -asemenea.

Cerință

Scrieți un program care pentru un set de t cuvinte dat, răspunde la interogări de genul: "Care este gradul de asemănare între cuvintele $x_1x_2\dots x_t$ " ?

Date de intrare

Fișierul de intrare **ratina.in** va conține pe prima linie două numere naturale $N M$, separate printr-un spațiu, reprezentând numărul de cuvinte din limba ratină, respectiv numărul de interogări.

Următoarele N linii vor conține cuvintele din limba ratină, câte un cuvânt pe o linie. Mai exact, pe linia $i + 1$ este scris cuvântul cu numărul de ordine i . Cuvintele sunt formate din litere mici din alfabetul englez.

Urmează M linii, fiecare linie reprezentând câte o interogare exprimată astfel: primul număr de pe linie este un număr natural t cuprins între 2 și 10 reprezentând numărul de cuvinte din interogare, apoi vor urma cele t numere de ordine ale cuvintelor din interogare, separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire **ratina.out** va conține M linii, câte una pentru fiecare interogare din fișierul de intrare. Pe linia i va fi scris gradul de asemănare al cuvintelor din interogarea i .

Restricții și precizări

$$0 < N < 10001$$

$$0 < \text{lungimea maximă a unui cuvânt} < 2000$$

$$0 < \text{suma lungimilor tuturor cuvintelor} < 200000$$

$$1 < M < 100001$$

Pentru teste cu $N \leq 200$, se acordă 50 de puncte din care 30 de puncte pentru teste și cu $M \leq 10000$.

Exemplu

ratina.in	ratina.out	Explicatii
6 5	3	Prima interogare cere gradul de asemănare între cuvintele asdf și asdeffff, care este 3 (deoarece cele două cuvinte au primele 3 litere identice, dar nu și primele 4 litere).
asdf	2	
asdeffff	3	
gata	0	
gara	4	Cea de a doua interogare cere gradul de asemănare între cuvintele gata și gara, care este 2.
pesistem		
pestesistem		Cea de a treia interogare cere gradul de asemănare între cuvintele pesistem și pestesistem care este 3.
2 1 2		
2 3 4		Cea de a patra interogare cere gradul de asemănare între cuvintele asdf, gata și pesistem care este 0.
2 5 6		
3 1 3 5		Ultima interogare este evidentă: un cuvânt de lungime k este k -asemenea cu el însuși.
2 1 1		

Timp maxim de execuție/test: 1 secundă

32.5.1 Indicații de rezolvare

Soluția oficială

Problema are mai multe solutii.

Dacă răspundem la fiecare întrebare parcurgând cuvintele pe toata lungimea lor până la găsirea unei diferențe (soluție de complexitate $O(M * L * C)$ (L =lungimea maximă, C numărul de cuvinte din întrebare)) obținem 30 de puncte.

Dacă preprocesăm pentru oricare 2 cuvinte gradul de asemănare, obținem 50 de puncte.

Pentru a obține punctajul maxim este necesar să răspundem mai repede la o astfel de întrebare.

Soluția comisiei folosește un *arbore de prefixe* pentru a reține dicționarul și totodată pentru fiecare caracter din cuvânt ce nod din arbore îi corespunde. Având aceste date preprocesate putem răspunde la o întrebare în $O(C * \log L)$ folosind *căutarea binară*. Mai multe cuvinte au prefixul de lungime k în cazul în care cuvintelor le corespunde același nod din arbore pentru acea poziție.

Un arbore de prefixe este un arbore în care fiecare nod are S fii unde S este dimensiunea alfabetului, în cazul nostru 26. Nodul rădăcină este special, îl putem nota cu caracterul $\#$, acesta are 26 de fii notați cu 'a', 'b', ..., 'z' acești fii având la rândul lor alti 26 de fii. Când introducem un cuvânt în arbore începem cu rădăcina și mergem în fiu care corespunde primului caracter din

acesta în fiul care corespunde celui de-al doilea caracter s.a.m.d în acest mod găsim ce nod în arbore îi corespunde unei anumite poziții dintr-un cuvânt.

32.5.2 Cod sursă

Listing 32.5.1: ratina.cpp

```

1 #include <iostream>
2 #include <stdlib.h>
3 #include <vector>
4 #include <cstring>
5 #include <ctime>
6
7 using namespace std;
8
9 struct node
10 {
11     node *L[26];
12 };
13
14 typedef node *trie;
15
16 trie T;
17
18 vector<node*> X[10024];
19
20 int i, j, n, m, k;
21
22 void baga(trie T, char *S, int x)
23 {
24     if (x == X[i][x-1]) return;
25     if (!S[x]) return;
26     if (!T->L[S[x]-'a'])
27     {
28         T->L[S[x]-'a'] = new (node);
29         memset(T->L[S[x]-'a']->L, 0, sizeof(T->L));
30     }
31     baga(T->L[S[x]-'a'], S, x+1);
32 }
33
34 void readdictionary()
35 {
36     scanf ("%d%d", &n, &m);
37     char S[5012];
38     T = new (node);
39     memset(T->L, 0, sizeof(T->L));
40
41     for (i=1; i<=n; i++)
42     {
43         scanf ("%s", &S);
44         X[i].resize(strlen(S));
45         baga(T, S, 0);
46     }
47 }
48
49
50 int sim(int *T, int n)
51 {
52     int cnt=4096;
53     int i=-1;
54
55     for (;cnt;cnt/=2)
56     {
57         if (X[T[1]].size()<=i+cnt) continue;
58
59         for (j=2; j<=n; j++)
60             if (X[T[j]].size()<=i+cnt || X[T[j]][i+cnt] != X[T[1]][i+cnt])
61                 break;
62         if (j<=n) continue;
63         i+=cnt;
64     }
65
66     return i+1;

```

```

67 }
68
69 void solve()
70 {
71     for (i=1; i<=m; i++)
72     {
73         int n;
74         int T[16];
75
76         scanf ("%d", &n);
77         for (j=1; j<=n; j++)
78             scanf ("%d", &T[j]);
79         printf ("%d\n", sim(T, n));
80     }
81 }
82 int main()
83 {
84     freopen ("ratina.in", "r", stdin);
85     freopen ("ratina.out", "w", stdout);
86
87     readdictionary();
88     solve();
89
90     return 0;
91 }
```

32.5.3 *Rezolvare detaliată

32.6 vitale

În orașul Etsivograt există n intersecții numerotate de la 1 la n . Între unele perechi de intersecții există străzi. În total sunt m străzi, iar rețeaua stradală formată din acestea a fost concepută astfel încât între oricare două intersecții există cel puțin o legătură care poate fi directă sau poate trece prin alte intersecții.

După trecerea anotimpului rece, starea străzilor a devenit deplorabilă, așa că se decide asfaltarea acestora. Asfaltarea fiecărei străzi are un cost cunoscut.

Având resurse limitate, primarul hotărăște să asfalteze câteva străzi care să asigure cel puțin o legătură (directă sau indirectă, adică trecând prin alte intersecții) între oricare două intersecții, iar suma costurilor asfaltării acestor câteva străzi să fie minimă. Consilierii lui îi prezintă lista tuturor posibilităților de asfaltare ce asigură legături (directe sau indirecte) între toate intersecțiile și pentru care suma costurilor este minimă. Primarul observă că există străzi care apar în toate aceste posibilități de asfaltare. El consideră aceste străzi ca fiind "vitale".

Cerință

Scrieți un program care determină numărul de străzi vitale care se află în rețeaua stradală a orașului Etsivograt, precum și care sunt aceste străzi vitale.

Date de intrare

Fișierul de intrare **vitale.in** conține pe prima linie două numere naturale n m separate printr-un spațiu reprezentând numărul de intersecții, respectiv numărul de străzi din oraș. Pe următoarele m linii se află câte trei numere naturale a b c separate prin câte un spațiu; a și b reprezintă numerele de ordine ale două intersecții distincte din oraș între care există o stradă, iar c reprezintă costul asfaltării străzii care unește intersecțiile a și b .

Date de ieșire

Fișierul de ieșire **vitale.out** va conține pe prima linie un număr natural x , reprezentând numărul de străzi vitale din oraș. Pe fiecare dintre următoarele x linii, vor fi scrise câte două numere naturale reprezentând numerele de ordine ale intersecțiilor care delimită o stradă vitală iar primul număr va fi strict mai mic decât al doilea. Aceste x linii vor fi sortate în ordine lexicografică, cu alte cuvinte notând cu a_i și b_i cele două numere de pe linia $i + 1$ și cu a_j și b_j cele două numere de pe linia $j + 1$, dacă $i < j$ atunci $a_i < a_j$ sau $a_i = a_j$ și $b_i < b_j$.

Restricții și precizări

$1 \leq n \leq 50000$

$1 \leq m \leq 100000$

$1 \leq c \leq 1000000000$

Între oricare două intersecții poate exista cel mult o stradă. Străzile sunt bidirectionale.

Exemplu

vitale.in	vitale.out
4 5	2
1 2 1	1 2
2 3 2	3 4
4 3 1	
1 4 2	
1 3 6	

Explicații

Intersecțiile și străzile din exemplu corespund configurației din figură.

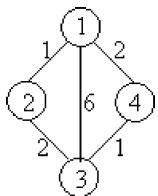


Figura 32.2: Vitale

Posibilitățile de asfaltare care corespund cerințelor sunt formate din străzile: 1-2, 1-4, 3-4 și 1-2, 2-3, 3-4. Sunt două muchii vitale (care apar în ambele posibilități) și anume 1-2 și 3-4.

Timp maxim de execuție/test: 1.5 secunde

32.6.1 Indicații de rezolvare

Soluția oficială

Există mai multe soluții de rezolvare în complexitate $O(m \log m)$.

Dăm una dintre ele.

Ideea pornește de la *algoritmul lui Kruskal* pentru arbore de cost minim care adaugă muchii în graf în ordinea crescătoare a costurilor.

Pentru toate muchiile de cost fixat x ne uităm la componentele conexe ale grafului ce conține muchiile de cost mai mic decât x , acest graf va conține niște componente conexe. Putem să contractăm acele componente conexe în noduri și să ignorăm muchiile de cost mai mare sau egal cu x care evident nu sunt muchii vitale.

Acum observăm că muchiile vitale de cost x sunt muchiile critice în graful care conține nodurile menționate anterior și muchiile de cost x .

Contractarea nodurilor în *componente conexe* se realizează folosind structuri de mulțimi disjuncte, iar determinarea muchiilor critice se face cu binecunoscutul algoritm pentru aflarea *componentelor biconexe* de complexitate $O(m)$. Acest algoritm are complexitate $O(m \log m)$ în faza de sortare, fazele de contracție ale grafului vor avea complexitate totală maxim $O(m \log *n)$ folosind structuri de mulțimi disjuncte, iar fazele de determinare ale muchiilor critice durează $O(m)$ în total, pentru că la pasul de cost x determinarea componentelor biconexe are complexitate $O(mx)$ unde mx este numărul de muchii de cost x , cum în total numărul de muchii este m , obținem complexitatea $O(m)$. Astfel complexitatea finală este $O(m \log m)$.

Algoritmi neoptimi la care ne-am gândit au fost generarea unui arbore parțial de cost minim, iar apoi eliminarea câte unei muchii din graf după care determinăm arborele de cost minim pe graful obținut. Dacă acest arbore are cost diferit de cel original, evident muchia eliminată a fost una critică. Acest algoritm are complexitatea $O(n * m)$ pentru că avem $n - 1$ muchii într-un arbore parțial, iar după ce muchiile grafului sunt sortate algoritmul Kruskal are complexitate $O(m)$. Aceasta ar fi luat în jur de 30-40 de puncte.

Alt algoritm ar fi determinarea randomizată a mai multor arbori parțiali și returnarea muchiilor comune tuturor arborilor. Aceasta ar fi luat în jur de 40 - 50 de puncte.

32.6.2 Cod sursă

Listing 32.6.1: vitale.pas

```

1 program vitale;
2 uses sysutils;
3
4 const VMAX = 100000;
5     EMAX = 200000;
6     inputfile = 'vitale.in';
7     outputfile = 'vitale.out';
8
9 type node = ^element;
10    element=record
11        vertex, index:longint;
12        next:node;
13    end;
14
15 var a:array[1..VMAX] of node;
16     x, y, w:array[1..EMAX + 1] of longint;
17     n, m:longint;
18
19     father, weight, dfs_num, low:array[1..VMAX] of longint;
20     dfs_nr:longint;
21
22     critical: array[1..EMAX] of boolean;
23
24 procedure read_data;
25 var fil:text;
26     i:longint;
27 begin
28     assign(fil, inputfile);reset(fil);
29     readln(fil, n, m);
30     for i := 1 to m do readln(fil, x[i], y[i], w[i]);
31     close(fil);
32 end;
33
34 var aux, mid:longint;
35 procedure qsort(lo, hi:longint);
36 var i, j:longint;
37 begin
38     i := lo; j := hi; mid := w[(i + j) shr 1];
39     repeat
40         while w[i] < mid do inc(i);
41         while w[j] > mid do dec(j);
42         if i <= j then begin
43             aux := x[i]; x[i] := x[j]; x[j] := aux;
44             aux := y[i]; y[i] := y[j]; y[j] := aux;
45             aux := w[i]; w[i] := w[j]; w[j] := aux;
46             inc(i); dec(j);
47         end;
48     until i > j;
49     if lo < j then qsort(lo, j);
50     if i < hi then qsort(i, hi);
51 end;
52
53 function root(x :longint):longint;
54 begin
55     if father[x] = 0 then root := x
56     else begin
57         father[x] := root(father[x]);
58         root := father[x];
59     end;
60 end;
61
62 procedure join(x, y:longint);
63 begin
64     if weight[x] > weight[y] then begin
65         father[y] := x;
66     end else if weight[x] < weight[y] then begin
67         father[x] := y;
68     end else begin
69         father[y] := x;
70         inc(weight[x]);
71     end;
72 end;
73
74 procedure dfs(xx, edge_index:longint);

```

```

75  var nd:node;
76      yy:longint;
77 begin
78     inc(dfs_nr);
79     dfs_num[xx] := dfs_nr;
80     low[xx] := dfs_nr;
81     nd := a[xx];
82     while nd <> nil do begin
83         yy := root(nd^.vertex);
84         if dfs_num[yy] = 0 then begin
85             dfs(yy, nd^.index);
86             if low[yy] = dfs_num[yy] then begin
87                 critical[nd^.index] := true;
88             end;
89             if low[yy] < low[xx] then low[xx] := low[yy];
90         end else if nd^.index <> edge_index then
91             if dfs_num[yy] < low[xx] then low[xx] := dfs_num[yy];
92         nd := nd^.next;
93     end;
94 end;
95
96 procedure kruskal;
97 var i, j, k:longint;
98     nd:node;
99 begin
100    qsort(1, m);
101    w[m + 1] := w[m] + 1;
102
103    fillchar(father, sizeof(father), 0);
104    fillchar(weighth, sizeof(weighth), 0);
105    fillchar(a, sizeof(a), 0);
106    fillchar(critical, sizeof(critical), false);
107
108    j:=1;
109    for i := 1 to m + 1 do begin
110        if w[j] <> w[i] then begin
111
112            for k := j to i - 1 do begin
113                a[root(x[k])] := nil;
114                a[root(y[k])] := nil;
115                dfs_num[root(x[k])] := 0;
116                low[root(x[k])] := 0;
117                dfs_num[root(y[k])] := 0;
118                low[root(y[k])] := 0;
119            end;
120
121            for k := j to i - 1 do begin
122                new(nd);
123                nd^.vertex := root(y[k]);
124                nd^.index := k;
125                nd^.next := a[root(x[k])];
126                a[root(x[k])] := nd;
127
128                new(nd);
129                nd^.vertex := root(x[k]);
130                nd^.index := k;
131                nd^.next := a[root(y[k])];
132                a[root(y[k])] := nd;
133            end;
134
135            dfs_nr := 0;
136            for k := j to i - 1 do begin
137                if dfs_num[root(x[k])] = 0 then dfs(root(x[k]), 0);
138            end;
139
140            for k := j to i - 1 do begin
141                if root(x[k]) <> root(y[k]) then begin
142                    join(root(x[k]), root(y[k]));
143                end;
144            end;
145            j := i;
146        end;
147    end;
148 end;
149 end;
150

```

```

151 var midx, midy:longint;
152 procedure qsort1(lo, hi:longint);
153 var i, j:longint;
154 begin
155   i := lo; j := hi;
156   midx := x[(lo + hi) shr 1];
157   midy := y[(lo + hi) shr 1];
158   repeat
159     while (x[i] < midx)
160       or ((x[i] = midx) and (y[i] < midy)) do inc(i);
161     while (x[j] > midx)
162       or ((x[j] = midx) and (y[j] > midy)) do dec(j);
163     if i <= j then begin
164       aux := x[i]; x[i] := x[j]; x[j] := aux;
165       aux := y[i]; y[i] := y[j]; y[j] := aux;
166       inc(i); dec(j);
167     end;
168   until i > j;
169   if lo < j then qsort1(lo, j);
170   if i < hi then qsort1(i, hi);
171 end;
172
173 procedure write_data;
174 var fil:text;
175   i, nr, xx, yy:longint;
176 begin
177 NR:=0;
178   for i := 1 to m do
179     if critical[i] then begin
180       inc(nr);
181       if (x[i] <= y[i]) then begin
182         x[nr] := x[i];
183         y[nr] := y[i];
184       end else begin
185         xx:= x[i];
186         yy:=y[i];
187         x[nr] := yy;
188         y[nr] := xx;
189       end;
190     end;
191   qsort1(1, nr);
192
193   assign(fil, outputfile); rewrite(fil);
194   writeln(fil, nr);
195   for i := 1 to nr do
196     writeln(fil, x[i], ' ', y[i]);
197   close(fil);
198 end;
199
200 begin
201   read_data;
202   kruskal;
203   write_data;
204 end.

```

32.6.3 *Rezolvare detaliată

Capitolul 33

ONI 2005 clasa a XI-a

33.1 Mașina

Se știe că Balaurul este un împătimit al volanului. Ieri a ajuns la o intersecție (dacă îi putem zice așa) foarte ciudată, ca în figura alăturată ...

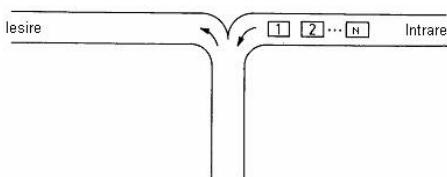


Figura 33.1: Masina

La această intersecție au ajuns N mașini (numerotate de la 1 la N). Balaurul se află în mașina X . În momentul acela se întreceau cu mașina Y (X diferit de Y). Cele 3 drumuri sunt foarte înguste, așa încât doar o mașină poate să încapă, deci depășirea este imposibilă. Totuși, datorită configurației drumurilor, mașinile își pot schimba poziția la ieșire.

De exemplu, pentru $N = 3$, la final avem 5 posibilități de ordonare a celor 3 mașini:

- 1) 1 2 3 : intră mașina 1 pe drumul din mijloc, și ieșe 1, intră 2 și ieșe 2, intră 3 și ieșe 3
- 2) 1 3 2 : intră 1 și ieșe 1, intră 2, intră 3, ieșe 3, ieșe 2
- 3) 2 1 3 : intră 1, intră 2, ieșe 2, ieșe 1, intră 3, ieșe 3
- 4) 2 3 1 : intră 1, intră 2, ieșe 2, intră 3, ieșe 3, ieșe 1
- 5) 3 2 1 : intră 1, intră 2, intră 3, ieșe 3, ieșe 2, ieșe 1

Oricare din cele M (în cazul acesta $N = 3, M = 5$) configurații posibile are şanse egale de a se întâmpla.

Balaurul vrea să știe care sunt şansele (în procente) ca la final să iasă în fața mașinii cu care se întrecea.

Cerință

Ajutați-l pe Balaur să determine şansele de a câștiga, deci de a ieși în fața mașinii Y .

Date de intrare

Pe prima linie a fișierului **masina.in** se află 3 numere naturale N, X și Y , separate prin câte un spațiu, reprezentând numărul de mașini, mașina Balaurului și respectiv mașina concurrentului.

Date de ieșire

Fișierul **masina.out** va conține pe singura sa linie un singur număr real cu primele 2 zecimale exacte (obținute prin trunchiere), și anume şansele (în procente) ca Balaurul să iasă la final în fața concurrentului.

Restricții și precizări

$$1 < N < 101$$

$$0 < X, Y < N + 1$$

pentru 50% din teste $X = 1$

trunchierea la două zecimale exacte a numărului real 60.5673 este 60.56

trunchierea la două zecimale exacte a numărului real 60.5628 este 60.56

trunchierea la două zecimale exacte a numărului real 60.5655 este 60.56

Exemplu

masina.in	masina.out	Explicație
3 1 3	60.00	Din cele 5 configurații în total, în 3 dintre ele mașina 1 ieșe în fața mașinii 3, deci şansele sunt de 60%

Timp maxim de execuție/test: 0.1 secunde sub Linux și 0.1 secunde sub Windows

33.1.1 Indicații de rezolvare

Problema este de programare dinamică. Există mai multe feluri de rezolvare, iar acesta este unul dintre ele:

Asemănăm acel drum din mijloc cu o stivă, iar operațiile sunt paranteze deschise (atunci când vine o mașină în stivă) și paranteze închise (când pleacă o mașină din stivă).

Construim:

$A[i][j]$ = numărul de posibilități de a ajunge în poziția cu i paranteze deschise și j închise

$B[i][j]$ = numărul de posibilități de a încheia situația (N deschise, N închise) din poziția (i, j) , adică i deschise, j închise.

Presupunem $X < Y$, altfel inversăm și scădem la sfârșit.

Variem momentul în care apare X în stivă (baza), și anume poziția pe care se găsește în stivă când apare.

Construim:

$Q[i][j]$ = numărul de posibilități de procesare a i mașini având j în stivă, ținem i între x și y și j între $baza$ și $baza + diferența$ între x și y .

Momentul în care se ia o decizie (dacă x e în fața lui y sau invers), este atunci când stiva e la nivelul $baza$ sau când urmează să procesăm pe y . Adunăm aceste momente, înmulțindu-le cu B -ul corespunzător (adică câte posibilități sunt până la sfârșit) și cu A -ul corespunzător (câte posibilități să ajungem la $baza$).

Întrucât precizia este doar de două zecimale, putem folosi tipul *double* pentru a calcula toate posibilitățile.

33.1.2 *Cod sursă

33.1.3 *Rezolvare detaliată

33.2 Matrice

Se dă o matrice cu N linii și 2 coloane de numere întregi. Liniile sunt numerotate cu numere de la 1 la N , iar coloanele cu 1 și 2.

Există patru operații care pot fi executate asupra matricei: S_3 , J_3 , S_4 , J_4 .

La o operație J_k se aleg k linii (dintincte) ale matricei și se permute circular în jos, iar la o operație S_k se aleg k linii (dintincte) și se permute circular în sus ($k = 3, 4$).

...	
x_1	x_2		z_1	z_2		x_1	x_2		y_1	y_2
...	...	→
y_1	y_2	→	x_1	x_2		y_1	y_2	→	z_1	z_2
...
z_1	z_2		y_1	y_2		z_1	z_2		x_1	x_2
...
Operație J_3					Operație S_3					

Operațiile J_4 și S_4 sunt similare, numai că în loc de 3 linii se aleg 4.

Cerință

Scriți un program care prin efectuarea a cel mult $2 * N$ operații de tip S_3 , J_3 , S_4 , J_4 asupra matricei date să o transforme astfel încât nici una dintre coloanele ei să nu conțină un subșir strict

crescător de lungime mai mare decât $\lceil \sqrt{N} \rceil$ (adică cel mai mic întreg mai mare sau egal decât \sqrt{N}).

Date de intrare

Fișierul de intrare **matrice.in** conține pe prima linie numărul natural N . Pe fiecare din următoarele N linii sunt câte 2 numere întregi separate printr-un spațiu, reprezentând elementele matricei.

Date de ieșire

Fișierul de ieșire **matrice.out** va conține câte o operație pe linie. O operație este identificată prin tipul ei (poate fi J_3 , S_3 , J_4 sau S_4) și 3 numere (pentru J_3 și S_3) sau 4 numere (pentru J_4 și S_4) în ordine strict crescătoare, reprezentând rândurile asupra cărora este executată operația. Tipul operației și celealte numere trebuie să fie separate prin exact un spațiu.

Atenție! Tipul operației este format din două caractere scrise fără spațiu între ele.

Restricții și precizări

$6 \leq N \leq 30000$

Un subșir strict crescător al unui sir a_1, a_2, \dots, a_N este un sir $a_{i1}, a_{i2}, \dots, a_{ik}$, unde $i_1 < i_2 < \dots < i_k$ și $a_{i1} < a_{i2} < \dots < a_{ik}$.

Elementele matricei sunt numere întregi mai mari sau egale cu 0 și mai mici sau egale cu 65000.

Exemplu

matrice.in	matrice.out
6	
1 2	S4 1 3 4 5
3 1	J3 4 5 6
4 4	
6 3	
5 5	
2 6	

Explicație

După efectuarea operațiilor, matricea va fi

4 4
3 1
6 3
2 6
5 5
1 2

Lungimea celui mai lung subșir strict crescător de pe prima coloană este 2, iar de pe cea de-a doua este 3 (2 și 3 sunt mai mici sau egale decât $\lceil \sqrt{6} \rceil = \lceil \sqrt{6} \rceil = 3$).

Timp maxim de execuție/test: 0.2 secunde sub Linux și 0.5 secunde sub Windows.

33.2.1 Indicații de rezolvare

Ideea este de a permuta rândurile matricii astfel încât să respecte proprietatea cerută și anume aceea ca oricare coloană să nu conțină un subșir strict crescător de lungime mai mare decât \sqrt{N} și de a scrie permutarea respectivă folosind operațiile permise.

Permutarea cerută se obține astfel: mai întâi se sortează rândurile matricei descrescător după valorile primei coloane. Apoi pentru grupe de câte \sqrt{N} rânduri, se sortează descreșător după valorile celei de-a doua coloane. Cel mai lung subșir crescător în prima coloană poate fi găsit numai în cadrul grupelor și deci are lungimea maximă \sqrt{N} . În a doua coloană, cel mai lung subșir crescător poate fi obținut alegând căt mai convenabil un element din fiecare grupă (cele din cadrul grupelor sunt sortate descreșător) și deci are lungimea maximă posibilă egală cu numărul grupelor, adică \sqrt{N} .

Pentru a obține o permutare cu ajutorul operațiilor S_3, J_3, S_4, J_4 , observăm că orice transpoziție se poate scrie în funcție de aceste operații. Orice permutare se poate descompune în cicluri (suma lungimilor lor fiind egală cu N). Cum orice ciclu de lungime L se poate descompune în L transpoziții și (vom demonstra că) orice transpoziție se poate obține prin efectuarea a 2 dintre operațiile permise, rezultă că orice permutare se poate obține prin efectuarea a cel mult 2^*N operațiilor permise.

Pentru a arăta că orice transpoziție se poate obține din operațiile S_3, J_3, S_4, J_4 , considerăm cazurile

$$\begin{array}{c} \left[\begin{array}{c} a \\ b \\ c \\ d \end{array} \right] \xrightarrow{S_4} \left[\begin{array}{c} b \\ c \\ d \\ a \end{array} \right] \xrightarrow{J_3 \text{ (pentru ultimele 3)}} \left[\begin{array}{c} b \\ a \\ c \\ d \end{array} \right] \\ \Rightarrow \begin{array}{l} \text{putem interschimba rândul } a \text{ cu } b \text{ dacă} \\ \text{după ele mai sunt cel puțin 2 rânduri} \end{array} \end{array}$$

$$\begin{array}{c} \left[\begin{array}{c} a \\ b \\ c \\ d \end{array} \right] \xrightarrow{J_4} \left[\begin{array}{c} d \\ a \\ b \\ c \end{array} \right] \xrightarrow{S_3 \text{ (pentru primele 3)}} \left[\begin{array}{c} a \\ b \\ d \\ c \end{array} \right] \\ \Rightarrow \begin{array}{l} \text{putem interschimba rândul } c \text{ cu } d \text{ dacă} \\ \text{înaintea lor mai sunt cel puțin 2 rânduri} \end{array} \end{array}$$

$$\begin{array}{c} \left[\begin{array}{c} a \\ b \\ c \\ d \end{array} \right] \xrightarrow{S_4} \left[\begin{array}{c} b \\ c \\ d \\ a \end{array} \right] \xrightarrow{J_3 \text{ (pentru primele 3)}} \left[\begin{array}{c} d \\ b \\ c \\ a \end{array} \right] \\ \Rightarrow \begin{array}{l} \text{putem interschimba rândul } a \text{ cu } d \text{ dacă} \\ \text{între ele mai sunt cel puțin 2 rânduri} \end{array} \end{array}$$

Pentru două rânduri a și b, cel puțin unul dintre cazuri este adevărat.

33.2.2 *Cod sursă

33.2.3 *Rezolvare detaliată

33.3 Ziduri

Un ziar are redacția la etajul unei clădiri. Acest etaj de formă pătratică este alcătuit numai din camere de aceeași dimensiune și de formă pătratică. Pentru un etaj cu 4×4 camere avem configurația:

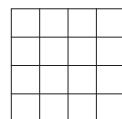


Figura 33.2: Ziduri1

Unele dintre zidurile camerelor lipsesc. Directorul și redactorul șef au fiecare biroul în camere separate. Directorul are biroul în camera de pe linia 1 și coloana 1, iar redactorul șef în camera de pe ultima linie și ultima coloană. Deplasarea între două camere vecine se poate face numai dacă ele nu au zid despărțitor. Pentru a mări viteza de deplasare între birourile directorului și redactorului șef se ia decizia ca unele ziduri să fie desființate. Un studiu făcut de departamentul administrativ arată că deplasarea între două camere fără zid conduce la un cost de o unitate monetară, iar deplasarea între două camere care au avut zid și a fost dărâmat conduce la un cost de p unități monetare.

Cerință

Determinați costul minim al unei deplasări de la camera directorului la camera redactorului șef. Dintre toate deplasările de cost minim, determinați numărul minim de ziduri ce trebuie dărâmate într-o astfel de deplasare.

Date de intrare

Fișierul de intrare **ziduri.in** conține pe prima linie n (numărul de camere de pe o linie, respectiv coloană) și p , costul trecerii de la o cameră la alta între care s-a dărâmat zidul despărțitor; cele două numere fiind separate printr-un spațiu. Pe următoarele n linii se află câte n numere naturale din mulțimea $\{0, 1, \dots, 15\}$ separate prin câte un spațiu. Aceste numere naturale transformate în baza 2 (pe 4 biți) ne dau informații despre existența zidurilor în jurul camerii (1 pentru zid și 0 în caz contrar). De exemplu dacă un astfel de număr are reprezentarea $abcd$ în baza 2, atunci a este pentru zidul din strelă vest, b pentru cel din nord, c pentru cel din est, iar d pentru cel din sud.

Date de ieșire

Fișierul de ieșire **ziduri.out** va conține pe prima linie costul minim al deplasării de la director la redactorul șef, iar pe linia a doua numărul minim de ziduri dărmăte.

Restricții și precizări

$$1 < n < 101$$

$$0 < p < 101$$

Nu se acordă punctaje parțiale.

Exemplu

ziduri.in	ziduri.out
4 3	8
9 1 1 0	1
12 5 5 1	
1 5 5 4	
4 6 12 0	

Explicație

Configurația birourilor redacției (zidurile sunt marcate cu linie îngroșată)

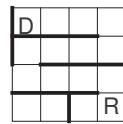


Figura 33.3: Ziduri2

Timp maxim de execuție/test: 0.1 secunde sub Linux și 0.1 secunde sub Windows

33.3.1 Indicații de rezolvare

Soluția 1

Se consideră graful format astfel: pentru fiecare cameră se consideră un nod, iar între două camere vecine pe orizontală/verticală (deci două noduri) o muchie de cost 1 dacă nu există zid între cele două camere, respectiv p dacă există zid.

Pe acest graf se calculează cu *algoritmul lui Dijkstra* costurile minime de la camera stânga-sus la toate celelalte camere.

Apoi se construiește *graful orientat aciclic* al tuturor drumurilor minime de la camera stânga-sus la camera dreapta jos.

Pe acest graf aciclic se calculează cu o parcurgere drumul dintre cele două camere care folosește număr minim de muchii formate din ziduri.

Pentru o complexitate mică, algoritmul lui Dijkstra trebuie implementat cu *heapuri*. Astfel, complexitatea totală e de $N^2 \log N$.

Soluția 2

Pe același graf, putem afla aceasta distanță folosind *algoritmul Bellman-Ford-Moore*. Deși acesta poate conduce la o complexitate de $O(N^4)$, ne putem baza pe faptul că p este destul de mic (mai mic decât 100 (adică maximul lui N)) și acesta reduce la $O(N^3)$ complexitatea, cu care putem obține punctaj maxim.

33.3.2 *Cod sursă

33.3.3 *Rezolvare detaliată

33.4 Lsort

Se consideră o listă simplu înlănțuită (L_1) ce conține numerele întregi de la 1 la N (într-o ordine oarecare). Se dorește construirea unei alte liste simplu înlănțuite (L_2) care să conțină numerele din lista L_1 în ordine crescătoare. Pentru a obține lista L_2 , se vor șterge elemente din L_1 și se vor adăuga în L_2 , conform procedeului descris în continuare: inițial lista L_2 e vidă. La primul pas putem șterge orice element din L_1 și acesta se adaugă în L_2 . Apoi, în următorii $N - 1$ pași, se poate șterge orice element din L_1 , dar acesta poate fi adăugat numai la începutul sau la sfârșitul lui L_2 . La final, L_1 nu va conține nici un element, iar L_2 trebuie să conțină numerele de la 1 la N în ordine crescătoare. Întrucât există mai multe posibilități de a construi L_2 , vom defini în continuare costul construcției lui L_2 și vom dori să determinăm o posibilitate de construcție a lui L_2 care are costul minim.

Algoritmul de construcție al lui L_2 constă din N pași, la fiecare pas ștergându-se un element din L_1 și adăugând acest element în L_2 (conform restricțiilor precizate). La pasul i ($1 \leq i \leq N$), lista L_1 conține $N - i + 1$ elemente și lista L_2 conține $i - 1$ elemente. Dacă se mută elementul de pe poziția k din L_1 în L_2 la pasul i ($1 \leq k \leq N - i + 1$), costul acestei mutări este egal cu $k * i$. După mutarea elementului, lista L_1 va avea cu un element mai puțin (toate elementele de pe poziții mai mari decât k vor ajunge cu o poziție mai în față) și lista L_2 va avea cu un element mai mult. Costul total al construcției lui L_2 este egal cu suma costurilor mutărilor efectuate la fiecare dintre cei N pași.

Date de intrare

Pe prima linie a fișierului **lsort.in** se află numărul N , reprezentând numărul de elemente din L_1 . Următoarea linie conține numerele întregi de la 1 la N , separate prin cel puțin un spațiu, în ordinea în care se află acestea pozitionate în lista L_1 (primul număr se află pe poziția 1 în L_1 , al doilea număr pe poziția 2 și.m.d.).

Date de ieșire

Pe prima linie a fișierului de ieșire **lsort.out** se va afișa costul minim de construcție a lui L_2 . Pe următoarea linie se va afișa modul de construcție al acesteia. Pe această linie se va afișa ordinea în care sunt mutate elementele din lista L_1 în lista L_2 . Primul număr afișat va reprezenta numărul mutat din L_1 în L_2 la primul pas; al doilea număr va reprezenta numărul mutat la pasul 2 și.m.d. Numerele afișate trebuie separate prin cel puțin un spațiu. În cazul în care există mai multe posibilități de construcție a listei L_2 având costul minim, se poate afișa oricare dintre ele.

Restricții și precizări

$$1 \leq N \leq 1000$$

Exemple

lsort.in	lsort.out	lsort.in	lsort.out
4	15	7	43
4 1 3 2	3 4 2 1	6 3 5 4 1 7 2	6 5 4 3 2 1 7

Explicație pentru primul exemplu

La pasul 1, $L_1 = [4, 1, 3, 2]$ și $L_2 = []$. Se șterge din L_1 elementul 3 (care se află pe poziția 3) și se introduce în L_2 . Costul acestei mutări este $3 * 1 = 3$.

La pasul 2, $L_1 = [4, 1, 2]$ și $L_2 = [3]$. Se șterge din L_1 elementul 4 (care se află pe poziția 1) și se introduce în L_2 (este evident că acest element trebuie adăugat la sfârșitul listei L_2 și nu la începutul ei; în caz contrar, lista L_2 nu ar mai fi sortată). Costul acestei mutări este $1 * 2 = 2$.

La pasul 3, $L_1 = [1, 2]$ și $L_2 = [3, 4]$. Se șterge din L_1 elementul 2 (care se află pe poziția 2) și se introduce în L_2 (din nou, poziția unde trebuie adăugat elementul este evidentă: la începutul listei L_2). Costul acestei mutări este $2 * 3 = 6$.

La pasul 4, $L_1 = [1]$ și $L_2 = [2, 3, 4]$. Se șterge din L_1 elementul 1 (care se află pe poziția 1) și se introduce în L_2 . Costul acestei mutări este $1 * 4 = 4$.

Costul total al construcției listei L_2 este $3 + 2 + 6 + 4 = 15$.

Timp maxim de execuție/test: 0.3 secunde sub Linux și 1.0 secunde sub Windows

33.4.1 Indicații de rezolvare

Problema se rezolvă folosind programare dinamică. Vom calcula o matrice

$cmin[i][j] =$ costul minim al unor operații în urma cărora, în L_2 , am obținut intervalul de numere $[i, j]$ ($i, i + 1, \dots, j - 1, j$).

Datorită modului de construcție a lui L_2 , pentru a obține intervalul $[i, j]$, ori am adăugat la ultima mutare numărul i și înainte am obținut intervalul $[i + 1, j]$, ori am adăugat numărul j și înainte am obținut intervalul $[i, j - 1]$.

Vom încerca ambele variante (să îl mutăm pe i ultimul sau să îl mutăm pe j ultimul) și vom alege costul minim.

Pentru a muta oricare din cele 2 numere (i sau j) trebuie să cunoaștem poziția lui curentă în lista $L1$. Această poziție se poate obține parcurgând toate elementele din intervalul $[i, j]$ și determinând câte elemente se aflau, inițial, pe o poziție mai mică în $L1$ (deoarece aceste elemente au fost deja introduse în $L2$ și, deci, nu se mai află în $L1$).

Procedând astfel, obținem o complexitate $O(N^3)$. Putem, însă, calcula poziția curentă a numărului i sau j în $O(1)$ (de exemplu, în cazul lui i , poziția curentă în $L1$ este poziția calculată pentru intervalul $[i, j - 1]$, luând, în plus, în considerare și elementul j - care poate sau nu să se afle înaintea lui i în $L1$). Astfel, complexitatea finală este $O(N^2)$.

Pentru compilatoarele Borland (care nu pot memora o matrice de $N * N$ elemente *long*), se observă că, pentru a calcula costul pentru intervalele de lungime L avem nevoie doar de costruile pentru intervalele de lungime $L - 1$.

Mai rămâne problema reconstituirii soluției. Pentru aceasta, putem memora 1 bit pentru fiecare pereche $[i, j]$ (am ales pe i sau pe j ca ultim element) $\Rightarrow N^2/8$ octeți.

33.4.2 *Cod sursă

33.4.3 *Rezolvare detaliată

33.5 Pătrat

Se numește *pătrat magic* de ordin N o matrice cu 3 linii și 3 coloane cu elemente întregi nenegative (mai mari sau egale cu 0) cu proprietatea că suma elementelor oricărei linii și oricărei coloane este N .

Cerință

Scrieți un program care să determine numărul pătratelor magice de ordin N modulo 30103.

Date de intrare

Fișierul de intrare **patrat.in** conține pe prima linie numărul N .

Date de ieșire

Fișierul de ieșire **patrat.out** va conține numărul pătratelor magice de ordin N modulo 30103.

Restrictii și precizări

$1 \leq N \leq 1000000$

Exemple

patrat.in	patrat.out	patrat.in	patrat.out
2	21	5	231

Timp maxim de execuție/test: 0.1 secunde pentru Linux și 0.1 secunde pentru Windows.

33.5.1 Indicații de rezolvare

Numărul pătratelor magice de ordin N este $(C_{N+2}^2)^2 - 3C_{N+3}^4$.

Fie $\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$ o matrice care respectă proprietățile date.

Observăm că dacă fixăm numerele întregi $a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23} \geq 0$, astfel încât

$$\begin{cases} a_{11} + a_{12} + a_{13} = N \\ a_{21} + a_{22} + a_{23} = N \end{cases}, \text{ atunci obținem } \begin{cases} a_{31} = N - a_{11} + a_{21} \\ a_{32} = N - a_{12} + a_{22} \\ a_{33} = N - a_{13} + a_{23} \end{cases}.$$

În plus, $a_{31} + a_{32} + a_{33} = N$. Singura condiție care trebuie impusă este ca $a_{31}, a_{32}, a_{33} \geq 0$.

O ecuație de forma $x + y + z = N$ cu $x, y, z \geq 0$ are C_N^2 soluții. (Dacă fixăm x și y , z va rezulta ca $N - x - y$. Pentru un x fixat, există $N - x + 1$ posibilități pentru y . Deci ecuația are $\sum_{x=0}^N (N - x + 1) = (N + 1) + N + \dots + 1 = \frac{(N+1)*(N+2)}{2}$ soluții.)

Ecuațiile $a_{11} + a_{12} + a_{13} = N$ și $a_{21} + a_{22} + a_{23} = N$ au deci $(C_{N+2}^2)^2$ soluții.

Dintre acestea trebuie să le scădem pe cele de unde unul dintre a_{31}, a_{32}, a_{33} rezultă negativ.

a_{31} rezultă negativ când $a_{11} + a_{21} \geq N + 1$.

Pentru a_{11} fixat, există $N - a_{11} + 1$ posibilități pentru a_{12} . De asemenea, există a_{11} posibilități pentru a_{21} și anume $N - a_{11} + 1 \leq a_{21} \leq N$, iar pentru fiecare din această posibilitate (adică pentru a_{21} fixat) există $N - a_{21} + 1$ posibilități pentru a_{22} .

Însumând după a_{11} , obținem $\sum_{a_{11}=1}^N (N - a_{11} + 1) * (1 + 2 + \dots + a_{11}) = C_{N+3}^4$ posibilități. (Această sumă nu trebuie neapărat calculată; putem să facem un *for* după a_{11} .)

Analog pentru a_{32} și a_{33} .

Mai facem observația că nu se poate ca două dintre a_{31}, a_{32}, a_{33} să rezulte negative deodată. Presupunând că $a_{11} + a_{21} \geq N + 1$ și $a_{12} + a_{22} \geq N + 1$, ar rezulta că $a_{21} + a_{22} \geq 2 * N - (a_{11} + a_{12}) + 2 \geq N + 2$, ceea ce este fals.

Deci numărul cerut este $(C_{N+2}^2)^2 - 3C_{N+3}^4$.

33.5.2 *Cod sursă

33.5.3 *Rezolvare detaliată

33.6 Cșir

Un sir *circular* este un sir format numai din caracterele "A" și "B" care are următoarele proprietăți:

- are lungime $N \geq 1$ (nu poate fi sirul vid);
- se consideră că după ultimul caracter din sir urmează primul caracter din sir.

Această proprietate implică faptul că orice sir circular are N subsecvențe de lungime L ($1 \leq L \leq N$).

O subsecvență de lungime L a unui sir circular S este un sir de caractere (obișnuit, nu circular) format din L caractere aflate pe poziții consecutive în sirul S . De exemplu, sirul circular "ABAAB" are 5 subsecvențe de lungime 3: "ABA", "BAA", "AAB", "ABA" și "BAB" (ele nu sunt distințe ca valoare, însă diferă ca poziție de început în sirul din care fac parte).

Un **cșir** este un sir circular care are în plus următoarea proprietate: pentru orice L ($1 \leq L \leq N$) și oricare două subsecvențe de lungime L (să le numim S_1 și S_2), numărul de caractere "A" din S_1 diferă față de numărul de caractere "A" din S_2 cu cel mult 1 (în valoare absolută).

Să considerăm sirul circular "BBAABAA". Acest sir nu este un **cșir**, deoarece există subsecvențele "BBAAB" și "AABAA" (de lungime 5), care conțin 2, respectiv 4 caractere "A" (diferența dintre numărul de caractere "A" este, astfel, 2). De asemenea, sirul "ABABAABAAAB" nu este un **cșir**, deoarece conține subsecvențe "AABAA" și "BABAB" pentru care diferența dintre numărul de caractere "A" este mai mare decât 1 (în valoare absolută).

Sirurile circulare "ABA" și "AABABAAB" sunt, în schimb, **cșir-uri**, deoarece oricare ar fi două subsecvențe S_1 și S_2 având aceeași lungime, diferența dintre numărul de caractere "A" din S_1 și numărul de caractere "A" din S_2 este mai mică sau egală cu 1 (în valoare absolută).

Cerință

Dându-se mai multe siruri circulare, determinați dacă ele sunt **cșir-uri**.

Date de intrare

Prima linie a fișierului de intrare **csir.in** conține numărul întreg S , reprezentând numărul de siruri conținute în fișier. Pe fiecare dintre următoarele S linii se află câte un sir circular.

Date de ieșire

În fișierul de ieșire **csir.out** se vor scrie S linii. Pe a K -a linie din acest fișier, se va afișa 1, dacă al K -lea sir din fișierul de intrare este un **cșir**, sau 0, în caz contrar.

Restricții și precizări

$$1 \leq S \leq 20$$

Lungimea fiecărui sir circular din fișierul de intrare este cuprinsă între 1 și 50000 (inclusiv). Sirurile conțin numai caracterele "A" și "B" (nu și "a" sau "b").

Nu se acordă punctaje parțiale.

Exemplu

csir.in	csir.out
4	0
BBAABAA	0
ABABAABAAB	1
ABA	1
AABABAAB	

Timp maxim de execuție/test: 0.2 secunde sub Linux și 0.5 secunde sub Windows

33.6.1 Indicații de rezolvare

Vom presupune că sirul conține mai multe A-uri decât B-uri (dacă nu se întâmplă astfel, toate B-urile se pot transforma în A-uri și toate A-urile în B-uri).

Șirul are o structură de forma următoare: $A^{i_1}BA^{i_2}BA^{i_3}B\dots A^{i_k}B$ (unde A^i reprezintă o secvență de i "A"-uri consecutive).

Se poate demonstra ușor că, într-un *csir*, există doar secvențe de forma A^iB sau $A^{i+1}B$.

În continuare, vom considera sirul compactat în care fiecare secvență de forma A^iB este notată cu A și fiecare secvență de forma $A^{i+1}B$ este notată cu B.

Acest sir compactat este tot un sir circular format numai din "A"-uri și "B"-uri, având o lungime mai mică decât sirul inițial. Se poate demonstra (nu chiar la fel de ușor) că sirul inițial este un *csir* dacă și numai dacă sirul compactat este un *csir*. Astfel, se folosește o rezolvare recursivă, care se termină când sirul conține numai "A"-uri (sau numai "B"-uri) sau când nu mai este de forma precizată (nu conține numai secvențe de forma A^iB și $A^{i+1}B$). Complexitatea rezolvării este $O(N)$ pentru fiecare sir.

33.6.2 *Cod sursă

33.6.3 *Rezolvare detaliată

Capitolul 34

ONI 2004

34.1 BASE3

Se dau 3 numere scrise în baza 3 (folosind cifrele 0, 1 și 2). Se dorește găsirea unui număr N în baza 3, care să aibă un număr impar de cifre, iar cifra de pe poziția din mijloc să aibă valoarea 1. Acest număr N trebuie obținut prin concatenarea celor trei numere date; în această concatenare, fiecare din cele 3 numere poate fi folosit de zero sau mai multe ori.

Cerință

Determinați numărul minim de cifre pe care îl poate avea un număr având proprietățile precizate mai sus.

Date de intrare

Fișierul de intrare **base3.in** conține 3 linii. Pe fiecare linie se află scris un număr în baza 3.

Date de ieșire

Fișierul de ieșire **base3.out** va conține numărul minim de cifre pe care îl poate avea un număr N cu proprietățile specificate. Dacă nu se poate obține nici un astfel de număr, afișați în fișier valoarea 0.

Restricții și precizări

Numărul de cifre al fiecărui din cele 3 numere este un număr întreg între 1 și 16000.

Numerele date pot conține zerouri la început; acestea trebuie luate în considerare, dacă numărul respectiv este folosit în concatenare.

Exemplu

base3.in	base3.out
001	13
020	
2020	

Explicație:

Se poate obține numărul 2020001001001.

Timp maxim de executare/test: 0.4 secunde pentru Linux și 0.8 secunde pentru Windows

34.1.1 Indicații de rezolvare

Mugurel Ionuț Andreica

Se calculează matricea $MIN[i, x, j]$, cu i între 1 și 3, j între 1 și 2, iar x între 0 și lungimea numărului i , având următoarea semnificație:

- dacă $j = 1$, $MIN[i, x, j]$ reprezintă lungimea celui mai scurt număr care are la mijloc primele x cifre din al i -lea număr
- dacă $j = 2$, $MIN[i, x, j]$ reprezintă lungimea celui mai scurt număr care are la mijloc ultimele x cifre din al i -lea număr

Calculul acestor valori corespunde unei determinări a numărului din exterior spre centru. Cei 3 indici ai matricei codifică o stare, iar trecerea de la o stare la alta se realizează prin concatenarea unuia din numere în partea stângă sau dreaptă.

Astfel, se poate folosi un algoritm de drum minim (de exemplu, *Dijkstra cu heap-uri*). Conform acestei codificări, numărul cerut corespunde unui drum minim în graful stărilor, iar lungimea acestui drum este limitată superior de $6 * 160002$ (dar, în practică, este mai mică).

Complexitatea algoritmului este $O((L1 + L2 + L3) * \log(L1 + L2 + L3))$.

34.1.2 Cod sursă

Listing 34.1.1: base3bkt1.pas

```

1 Program _BASE3_;
2
3 const filein = 'base3.in';
4     fileout = 'base3.out';
5     MAXLEN = 16000;
6     MAXNUM = 3;
7     infinit = 1000000000;
8     maxlimit = 2000000;
9
10 var num : array [1..MAXNUM, 1..MAXLEN] of byte;
11    len : array [1..MAXNUM] of longint;
12    i, j, k, m, n, t, minlen, actlen, limit : longint;
13    ch : char;
14
15 procedure writesol;
16 begin
17 { afisaza rezultatul }
18 assign(output, fileout);
19 rewrite(output);
20
21 if (minlen = infinit) then
22   writeln(0)
23 else
24   begin
25     writeln(minlen);
26   end;
27
28 close(output);
29 halt(0);
30 end;
31
32 procedure bkt(i, j, t : integer);
33 var nexti, nextj, nextt, k : integer;
34 begin
35
36 if (t = 1) and ((len[i] - j) and 1 = 1) and
37   (num[i, (j + len[i] + 1) shr 1] = 1) then
38   begin
39     minlen := actlen;
40     writeln(minlen);
41     exit;
42   end;
43
44 if (t = 2) and (j and 1 = 1) and (num[i, (j + 1) shr 1] = 1) then
45   begin
46     minlen := actlen;
47     writeln(minlen);
48     exit;
49   end;
50
51 for k := 1 to 3 do
52   if (actlen + len[k] < minlen) and (actlen + len[k] < limit) then
53     begin
54       actlen := actlen + len[k];
55
56       nexti := i;
57       nextt := t;
58       nextj := j;
59
60       if (t = 1) then

```

```

61      begin
62        if (len[k] > len[i] - j) then
63          begin
64            nexti := k;
65            nextj := len[k] - (len[i] - j);
66            nextt := 2;
67          end
68        else
69          if (len[k] < len[i] - j) then
70            begin
71              nexti := i;
72              nextj := j + len[k];
73              nextt := 1;
74            end;
75          end
76        else
77          begin
78            if (len[k] > j) then
79              begin
80                nexti := k;
81                nextj := j;
82                nextt := 1;
83              end
84            else
85              if (len[k] < j) then
86                begin
87                  nexti := i;
88                  nextj := j - len[k];
89                  nextt := 2;
90                end;
91            end;
92          end;
93          bkt(nexti, nextj, nextt);
94
95        actlen := actlen - len[k];
96      end;
97    end;
98
99  begin
100 { read input data }
101 assign(input, filein);
102 reset(input);
103 for i := 1 to 3 do
104   begin
105     len[i] := 0;
106     while (not seekeoln(input)) do
107       begin
108         repeat
109           read(ch);
110           until (ch in ['0'..'2']);
111
112         inc(len[i]);
113         num[i, len[i]] := ord(ch) - 48;
114       end;
115       readln;
116     end;
117   close(input);
118
119 minlen := infinit;
120 actlen := 0;
121
122 limit := 16;
123 while (limit < maxlimit) do
124   begin
125     bkt(1, 0, 2);
126
127     if (minlen < infinit) then
128       writesol;
129
130     limit := limit shl 1;
131   end;
132
133 writesol;
134 end.

```

Listing 34.1.2: base3bkt2.pas

```

1 Program _BASE3_;
2
3 const filein = 'base3.in';
4     fileout = 'base3.out';
5     MAXLEN = 16000;
6     MAXNUM = 3;
7     infinit = 1000000000;
8     maxlimit = 2000000;
9     MAXNIV = 15000;
10
11 var num : array [1..MAXNUM, 1..MAXLEN] of byte;
12    len : array [1..MAXNUM] of longint;
13    stk : array [1..MAXNIV] of byte;
14    i, j, k, m, n, t, maxlen, actlen, limit, needed : longint;
15    found : boolean;
16    ch : char;
17
18 procedure writesol;
19 begin
20 { afisaza rezultatul }
21 assign(output, fileout);
22 rewrite(output);
23
24 if (minlen = infinit) then
25   writeln(0)
26 else
27   begin
28     writeln(minlen);
29   end;
30
31 close(output);
32 halt(0);
33 end;
34
35 procedure bkt(niv : integer);
36 var k : integer;
37 begin
38 if (niv > MAXNIV) then
39   exit;
40
41 for k := 1 to 3 do
42   begin
43     actlen := actlen + len[k];
44
45     if (actlen < maxlen) and (actlen < limit) then
46       begin
47         stk[niv] := k;
48
49         if (actlen and 1 = 1) then
50           begin
51             needed := (actlen + 1) shr 1;
52
53             found := false;
54             for i := 1 to niv do
55               if (len[stk[i]] >= needed) then
56                 begin
57                   if (num[stk[i], needed] = 1) then
58                     found := true;
59
60                   break;
61                 end
62               else
63                 needed := needed - len[stk[i]];
64
65               if (found) then
66                 begin
67                   maxlen := actlen;
68                   writeln(maxlen);
69                 end
70               else
71                 bkt(niv + 1);
72             end
73           else
74             bkt(niv + 1);
75       end;

```

```

76      actlen := actlen - len[k];
77  end;
78 end;
79
80 begin
81 { read input data }
83 assign(input, filein);
84 reset(input);
85 for i := 1 to 3 do
86 begin
87   len[i] := 0;
88   while (not seekeoln(input)) do
89   begin
90     repeat
91       read(ch);
92       until (ch in ['0'..'2']);
93
94     inc(len[i]);
95     num[i, len[i]] := ord(ch) - 48;
96   end;
97   readln;
98 end;
99 close(input);
100
101 maxlen := infinit;
102 actlen := 0;
103
104 limit := 16;
105 while (limit < maxlimit) do
106 begin
107   bkt(1);
108
109   if (minlen < infinit) then
110     writesol;
111
112   limit := limit shl 1;
113 end;
114
115 writesol;
116 end.

```

Listing 34.1.3: base3ok.pas

```

1 Program _BASE3_;
2
3 const filein = 'base3.in';
4   fileout = 'base3.out';
5   MAXLEN = 16000;
6   MAXNUM = 3;
7   infinit = 1000000000;
8
9 type longarray = array [0..MAXLEN] of longint;
10  plongarray = ^longarray;
11  pq = ^q;
12  q = record
13    nu, tip : byte;
14    po : integer;
15    urm : pq;
16  end;
17
18 var min : array [1..MAXNUM, 1..2] of plongarray;
19   num : array [1..MAXNUM, 1..MAXLEN] of byte;
20   len : array [1..MAXNUM] of longint;
21   i, j, k, n, n1, t, nexti, nextj, nextt, newv, minlen : longint;
22   ch : char;
23   cst, cfin, aux : pq;
24
25 begin
26 { read input data }
27 assign(input, filein);
28 reset(input);
29
30 n1 := 0;
31 for i := 1 to 3 do

```

```

32  begin
33    len[i] := 0;
34    while (not seekeoln(input)) and (not seekeof(input)) do
35      begin
36        repeat
37          read(ch);
38          until (ch in ['0'..'2']);
39
40        inc(len[i]);
41        num[i, len[i]] := ord(ch) - 48;
42
43        if (ch = '1') then
44          inc(n1);
45      end;
46      readln;
47    end;
48  close(input);
49
50 { calculeaza }
51 for i := 1 to 3 do
52   for j := 1 to 2 do
53     begin
54       new(min[i, j]);
55
56       for k := 0 to MAXLEN do
57         min[i, j]^[k] := infinit;
58     end;
59
60 new(cst);
61 cst^.nu := 1;
62 cst^.po := 0;
63 cst^.tip := 2;
64 cst^.urm := nil;
65 cfin := cst;
66 min[1, 2]^[0] := 0;
67
68 while (cst <> nil) do
69   begin
70     i := cst^.nu;
71     j := cst^.po;
72     t := cst^.tip;
73
74     for k := 1 to 3 do
75       begin
76         nexti := i;
77         nextj := j;
78         nextt := t;
79         newv := min[i, j]^[j] + len[k];
80
81         if (t = 1) then
82           begin
83             if (len[k] > len[i] - j) then
84               begin
85                 nexti := k;
86                 nextj := len[k] - (len[i] - j);
87                 nextt := 2;
88               end
89             else
90               if (len[k] < len[i] - j) then
91                 begin
92                   nexti := i;
93                   nextj := j + len[k];
94                   nextt := 1;
95                 end;
96               end
97             else
98               begin
99                 if (len[k] > j) then
100                   begin
101                     nexti := k;
102                     nextj := j;
103                     nextt := 1;
104                   end
105                 else
106                   if (len[k] < j) then
107                     begin

```

```

108         nexti := i;
109         nextj := j - len[k];
110         nexttt := 2;
111         end;
112     end;
113
114     if (newv < min[nexti, nexttt]^nextj) then
115     begin
116         min[nexti, nexttt]^nextj := newv;
117
118         new(aux);
119         aux^.nu := nexti;
120         aux^.tip := nexttt;
121         aux^.po := nextj;
122         aux^.urm := nil;
123         cfin^.urm := aux;
124         cfin := aux;
125     end;
126 end;
127
128 aux := cst;
129 cst := cst^.urm;
130 dispose(aux);
131 end;
132
133 { cauta rezultatul }
134 minlen := infinit;
135 for i := 1 to MAXNUM do
136   for j := 0 to len[i] do
137   begin
138     if (min[i, 1]^j < minlen) and
139       ((len[i] - j) and 1 = 1) and
140       (num[i, (j + len[i] + 1) shr 1] = 1) then
141     begin
142       minlen := min[i, 1]^j;
143       nexti := i;
144       nextj := j;
145       nexttt := 1;
146     end;
147
148     if (min[i, 2]^j < minlen) and
149       (j and 1 = 1) and (num[i, (j + 1) shr 1] = 1) then
150     begin
151       minlen := min[i, 2]^j;
152       nexti := i;
153       nextj := j;
154       nexttt := 2;
155     end;
156   end;
157
158 { afisaza rezultatul }
159 writeln(n1, ': ', nextt, ' ', nextj, ' ', nexttt);
160
161 assign(output, fileout);
162 rewrite(output);
163
164 if (minlen = infinit) then
165   writeln(0)
166 else
167   begin
168     writeln(minlen);
169   end;
170
171 close(output);
172 end.

```

Listing 34.1.4: base3ok2.pas

```

1 Program _BASE3_;
2
3 const filein = 'base3.in';
4   fileout = 'base3.out';
5   MAXLEN = 16000;
6   MAXNUM = 3;
7   MAXHEAP = 100000;

```

```

8      infinit = 1000000000;
9
10 var min, where : array [1..MAXNUM, 1..2, 0..MAXLEN] of longint;
11    whoi, whoj, whot : array [1..MAXHEAP] of integer;
12    num : array [1..MAXNUM, 1..MAXLEN] of byte;
13    len : array [1..MAXNUM] of longint;
14    i, j, k, n, n1, t, nexti, nextj, newv, minlen, dimheap : longint;
15    ch : char;
16
17 procedure up(poz : longint);
18 var tata : longint;
19     i, j, t : integer;
20 begin
21   tata := poz shr 1;
22   if (tata = 0) then
23     tata := poz;
24
25   while (min[whoi[tata], whot[tata], whoj[tata]] >
26         min[whoi[poz], whot[poz], whoj[poz]]) do
27     begin
28       i := whoi[tata];
29       t := whot[tata];
30       j := whoj[tata];
31       whoi[tata] := whoi[poz];
32       whot[tata] := whot[poz];
33       whoj[tata] := whoj[poz];
34       whoi[poz] := i;
35       whot[poz] := t;
36       whoj[poz] := j;
37
38       where[whoi[tata], whot[tata], whoj[tata]] := tata;
39       where[whoi[poz], whot[poz], whoj[poz]] := poz;
40
41       poz := tata;
42       tata := poz shr 1;
43       if (tata = 0) then
44         tata := poz;
45     end;
46   end;
47
48 procedure down(poz : longint);
49 var i, j, t : integer;
50   v, v1, v2, p1, p2 : longint;
51   ok : boolean;
52 begin
53   ok := false;
54   while (not ok) do
55     begin
56       ok := true;
57
58       p1 := poz shl 1;
59       if (p1 <= dimheap) then
60         v1 := min[whoi[p1], whot[p1], whoj[p1]]
61       else
62         v1 := infinit;
63
64       p2 := p1 + 1;
65       if (p2 <= dimheap) then
66         v2 := min[whoi[p2], whot[p2], whoj[p2]]
67       else
68         v2 := infinit;
69
70       v := min[whoi[poz], whot[poz], whoj[poz]];
71
72       if (v1 <= v2) and (v1 < v) then
73         begin
74           i := whoi[p1];
75           t := whot[p1];
76           j := whoj[p1];
77           whoi[p1] := whoi[poz];
78           whot[p1] := whot[poz];
79           whoj[p1] := whoj[poz];
80           whoi[poz] := i;
81           whot[poz] := t;
82           whoj[poz] := j;
83

```

```

84      where[whoi[p1], whot[p1], whoj[p1]] := p1;
85      where[whoi[poz], whot[poz], whoj[poz]] := poz;
86
87      poz := p1;
88      ok := false;
89      end
90    else
91      if (v2 <= v1) and (v2 < v) then
92        begin
93          i := whoi[p2];
94          t := whot[p2];
95          j := whoj[p2];
96          whoi[p2] := whoi[poz];
97          whot[p2] := whot[poz];
98          whoj[p2] := whoj[poz];
99          whoi[poz] := i;
100         whot[poz] := t;
101         whoj[poz] := j;
102
103        where[whoi[p2], whot[p2], whoj[p2]] := p2;
104        where[whoi[poz], whot[poz], whoj[poz]] := poz;
105
106        poz := p2;
107        ok := false;
108      end;
109    end;
110  end;
111
112 procedure insheap(i, t, j : integer);
113 begin
114  inc(dimheap);
115  whoi[dimheap] := i;
116  whoj[dimheap] := j;
117  whot[dimheap] := t;
118  where[i, t, j] := dimheap;
119
120  up(dimheap);
121 end;
122
123 procedure delmin;
124 begin
125  whoi[1] := whoi[dimheap];
126  whoj[1] := whoj[dimheap];
127  whot[1] := whot[dimheap];
128  where[whoi[1], whot[1], whoj[1]] := 1;
129
130  dec(dimheap);
131
132  if (dimheap > 0) then
133    down(1);
134  end;
135
136 begin
137  { read input data }
138  assign(input, filein);
139  reset(input);
140
141  n1 := 0;
142  for i := 1 to 3 do
143    begin
144      len[i] := 0;
145      while (not seekeoln(input)) and (not seekeof(input)) do
146        begin
147          repeat
148            read(ch);
149            until (ch in ['0'..'2']);
150
151          inc(len[i]);
152          num[i, len[i]] := ord(ch) - 48;
153
154          if (ch = '1') then
155            inc(n1);
156        end;
157        readln;
158      end;
159  close(input);

```

```

160
161 { calculeaza }
162 for i := 1 to 3 do
163   for j := 1 to 2 do
164     for k := 0 to MAXLEN do
165       begin
166         min[i, j, k] := infinit;
167         where[i, j, k] := 0;
168       end;
169
170 min[1, 2, 0] := 0;
171
172 dimheap := 0;
173
174 insheap(1, 2, 0);
175
176 while (dimheap > 0) do
177   begin
178     i := whoi[1];
179     j := whoj[1];
180     t := whot[1];
181
182     for k := 1 to 3 do
183       begin
184         nexti := i;
185         nextj := j;
186         nextt := t;
187         newv := min[i, t, j] + len[k];
188
189         if (t = 1) then
190           begin
191             if (len[k] > len[i] - j) then
192               begin
193                 nexti := k;
194                 nextj := len[k] - (len[i] - j);
195                 nextt := 2;
196               end
197             else
198               if (len[k] < len[i] - j) then
199                 begin
200                   nexti := i;
201                   nextj := j + len[k];
202                   nextt := 1;
203                 end;
204               end
205             else
206               begin
207                 if (len[k] > j) then
208                   begin
209                     nexti := k;
210                     nextj := j;
211                     nextt := 1;
212                   end
213                 else
214                   if (len[k] < j) then
215                     begin
216                       nexti := i;
217                       nextj := j - len[k];
218                       nextt := 2;
219                     end;
220               end;
221
222             if (newv < min[nexti, nextt, nextj]) then
223               begin
224                 min[nexti, nextt, nextj] := newv;
225
226                 if (where[nexti, nextt, nextj] > 0) then
227                   up(where[nexti, nextt, nextj])
228                 else
229                   insheap(nexti, nextt, nextj);
230               end;
231             end;
232
233             delmin;
234           end;
235

```

```

236 { cauta rezultatul }
237 minlen := infinit;
238 for i := 1 to MAXNUM do
239   for j := 0 to len[i] do
240     begin
241       if (min[i, 1, j] < minlen) and
242         ((len[i] - j) and l = 1) and
243         (num[i, (j + len[i] + 1) shr 1] = 1) then
244       begin
245         minlen := min[i, 1, j];
246         nexti := i;
247         nextj := j;
248         nextt := 1;
249       end;
250
251       if (min[i, 2, j] < minlen) and
252         (j and l = 1) and (num[i, (j + 1) shr 1] = 1) then
253       begin
254         minlen := min[i, 2, j];
255         nexti := i;
256         nextj := j;
257         nextt := 2;
258       end;
259     end;
260
261 { afisaza rezultatul }
262 writeln(n1, ': ', nextt, ' ', nextj, ' ', nextt);
263
264 assign(output, fileout);
265 rewrite(output);
266
267 if (minlen = infinit) then
268   writeln(0)
269 else
270   begin
271     writeln(minlen);
272   end;
273
274 close(output);
275 end.

```

34.1.3 *Rezolvare detaliată

34.2 Coach

Sunteți antrenorul ciclistului Adirem Onamihs. În curând va avea loc un eveniment sportiv, iar pentru organizarea acestuia s-au amenajat N intersecții și M drumuri bidirectionale între aceste intersecții. Pentru fiecare drum se cunoaște numărul de minute necesare pentru parcurgerea lui. La fiecare intersecție ciclistul care trece pe acolo este obligat să servească o băutură energizantă și răcoritoare. Băutura diferă de la intersecție la intersecție și se cunoaște deja numărul de calorii ale fiecărei băuturi.

Ca mare antrenor, urmează să întocmiți un plan special pentru a-l antrena pe Adirem. Doriți ca durata traseului pe care îl alege Adirem să aibă exact T minute, însă nu vreți să-i plănuiti întregul traseu (Adirem trebuie să își antreneze și mintea, nu numai corpul). Îi veți preciza lui Adirem intersecția de unde își începe traseul și intersecția unde îl termină. Adirem învață repede - el știe întotdeauna să aleagă traseul optim (drumul cel mai scurt dintre cele două intersecții). Pentru a-l face să meargă exact T minute îi veți interzice trecerea prin anumite intersecții, sub pretextul că valoarea calorică a băuturii servite în intersecția respectivă nu este benefică pentru antrenamentul lui. Astfel, îi veți preciza o limită inferioară și una superioară pentru numărul de calorii ale băuturilor pe care el are voie să le bea. Adirem nu va trece decât prin intersecțiile unde se servește o băutură cu valoare calorică între limitele date.

Cerință

Scrieți un program care să calculeze cele patru variabile din antrenamentul lui Adirem: intersecția de start, intersecția de finish, valoarea calorică minimă pe care poate să o consume și

valoarea calorică maximă, astfel încât drumul cel mai scurt dintre cele două intersecții (care să respecte restricțiile) să dureze exact T minute.

Date de intrare

Prima linie a fișierului **coach.in** conține trei numere întregi N , M și T - numărul de intersecții, numărul de drumuri, respectiv timpul dorit. Următoarele N linii conțin câte un număr - valorile calorice (întregi între 1 și 10000 inclusiv) ale băuturilor din intersecții, în ordine (de la 1 la N). Următoarele M linii conțin câte un triplet de numere: două intersecții (numere distințe între 1 și N) și durata drumului dintre ele (întreg între 1 și 10000 inclusiv).

Date de ieșire

Fișierul **coach.out** va conține o linie pe care se vor afla cele patru valori găsite: nodul de start, nodul de finish, valoarea calorică minimă și valoarea calorică maximă. Nodurile vor fi întregi între 1 și N , iar valorile calorice vor fi întregi între 1 și 10000 (inclusiv).

Restricții și precizări

$1 \leq N \leq 100$,

$1 \leq M \leq 4950$,

$1 \leq T \leq 1000000$

Intersecțiile găsite (de start și de finish) trebuie să respecte și ele restricțiile calorice

O băutură cu valoare calorică x poate fi băută dacă și numai dacă $cmin \leq x \leq cmax$, unde $cmin$ și $cmax$ sunt valorile calorice minime și maxime stabilite de antrenor

Între două intersecții există maximum un drum

Valorile calorice sunt distințe

Există întotdeauna soluție; dacă există mai multe soluții se cere oricare dintre ele

Exemplu

coach.in	coach.out
6 9 11	3 6 20 55
40	
10	
20	
30	
60	
50	
1 2 2	
1 3 2	
1 4 4	
1 6 10	
2 3 3	
2 4 1	
4 5 1	
4 6 5	
5 6 2	

Timp maxim de executare/test: 0.5 secunde pentru Linux și 0.9 secunde pentru Windows

34.2.1 Indicații de rezolvare

Radu Berinde

Se sortează nodurile după valoarea asociată și se renumeștează (nodul 1 are valoarea cea mai mică).

Se fixează un nod de start s (și deci un $cmin$) și se aplică *algoritmul Roy-Floyd*, pentru nodurile din graf care au valoarea asociată mai mare sau egală cu $cmin$ (noduri de la s la N).

Proprietatea algoritmului Roy-Floyd aplicat de la nodul s este următoarea: la pasul k (cu k de la s la N), $A[i][j]$ este drumul cel mai scurt de la i la j care poate trece prin noduri de la s la k .

Datorită acestei proprietăți, complexitatea totală este $O(N^4)$, nefiind necesară și fixarea lui $cmax$ (care ar fi condus la o complexitate $O(N^5)$ și obținerea a 40 puncte), înaintea aplicării algoritmului.

34.2.2 Cod sursă

Listing 34.2.1: coach.c

```

1  /*
2  Coach O(N^4) solution
3  by Berinde Radu
4  */
5
6 #ifdef __BORLANDC__
7 #pragma option -3 -O2
8 #endif
9
10 #include <stdio.h>
11
12 int N;
13 long T;
14
15 int Val[101], Num[101], Renum[101];
16
17 int M[101][101];
18 long A[101][101];
19 #define INF 1006666666L
20 int Ri, Rj, Rs, Re;
21
22 int try_start(int s)
23 {
24     int i, j, k;
25     int cont = 0;
26
27     for (i = 1; i <= N; i++)
28         for (j = 1; j <= N; j++)
29             A[i][j] = (i >= s && j >= s && M[i][j]) ? M[i][j] : INF;
30     for (k = s; k <= N; k++)
31     {
32         for (i = s; i <= N; i++)
33             if (A[i][k] <= INF)
34                 for (j = s; j <= N; j++)
35                     if (A[i][j] > A[i][k] + A[k][j])
36                         A[i][j] = A[i][k] + A[k][j];
37         for (i = s; i <= k; i++)
38             for (j = s; j <= k; j++)
39             {
40                 if (A[i][j] < T)
41                     cont = 1;
42                 if (A[i][j] == T)
43                 {
44                     Ri = i, Rj = j, Rs = Val[s], Re = Val[k];
45                     return 0;
46                 }
47             }
48     }
49     return cont ? 1 : -1;
50 }
51
52 void solve()
53 {
54     int i;
55     for (i = 1; i < N; i++)
56         if (try_start(i) != 1) break;
57 }
58
59 void read_data()
60 {
61     int i, j, t, m;
62     FILE *fi = fopen("coach.in", "rt");
63     fscanf(fi, "%d %d %ld", &N, &m, &T);
64     for (i = 1; i <= N; i++)
65     {
66         fscanf(fi, "%d", Val + i);
67         Renum[i] = Num[i] = i;
68     }
69     for (i = 1; i < N; i++)
70         for (j = i+1; j <= N; j++)

```

```

71     if (Val[i] > Val[j])
72     {
73         Renum[Num[i]] = j, Renum[Num[j]] = i;
74         t = Val[i], Val[i] = Val[j], Val[j] = t;
75         t = Num[i], Num[i] = Num[j], Num[j] = t;
76     }
77     while (m--)
78     {
79         fscanf(fi, "%d %d %d", &i, &j, &t);
80         M[Renum[i]][Renum[j]] = M[Renum[j]][Renum[i]] = t;
81     }
82     fclose(fi);
83 }
84
85 int main()
86 {
87     read_data();
88     solve();
89     fprintf(fopen("coach.out", "wt"), "%d %d %d %d\n", Num[Ri], Num[Rj], Rs, Re);
90     return 0;
91 }
```

Listing 34.2.2: coachsl.c

```

1 /*
2 Coach O(N^5) solution
3 Should get about 40 points
4 by Berinde Radu
5 */
6
7 #ifdef __BORLANDC__
8 #pragma option -3 -O2
9 #endif
10
11 #include <stdio.h>
12
13 int N;
14 long T;
15
16 int Val[101], Num[101], Renum[101];
17
18 int M[101][101];
19 long A[101][101];
20 #define INF 1006666666L
21 int Ri, Rj, Rs, Re;
22
23 int try_start(int s, int e)
24 {
25     int i, j, k;
26
27     for (i = s; i <= e; i++)
28     {
29         for (j = s; j <= e; j++)
30             A[i][j] = M[i][j] ? M[i][j] : INF;
31         A[i][i] = 0;
32     }
33
34     for (k = s; k <= e; k++)
35         for (i = s; i <= e; i++)
36             if (A[i][k] <= INF)
37                 for (j = s; j <= e; j++)
38                     if (A[i][j] > A[i][k] + A[k][j])
39                         A[i][j] = A[i][k] + A[k][j];
40         for (i = s; i <= e; i++)
41             for (j = s; j <= e; j++)
42                 if (A[i][j] == T)
43                 {
44                     Ri = i, Rj = j, Rs = Val[s], Re = Val[e];
45                     return 0;
46                 }
47     return 1;
48 }
49
50 void solve(void)
51 {
```

```

52     int i, j;
53     for (i = 1; i < N; i++)
54         for (j = i+1; j <= N; j++)
55             if (try_start(i, j) != 1)
56             {
57             /*
58             printf("7 : %d %d\n", i, j);
59             */
60             return;
61         }
62     }
63
64 void read_data(void)
65 {
66     int i, j, t, m;
67     FILE *fi = fopen("coach.in", "rt");
68     fscanf(fi, "%d %d %d", &N, &m, &t);
69     for (i = 1; i <= N; i++)
70     {
71         fscanf(fi, "%d", Val + i);
72         Renum[i] = Num[i] = i;
73     }
74     for (i = 1; i < N; i++)
75         for (j = i+1; j <= N; j++)
76             if (Val[i] > Val[j])
77             {
78                 Renum[Num[i]] = j, Renum[Num[j]] = i;
79                 t = Val[i], Val[i] = Val[j], Val[j] = t;
80                 t = Num[i], Num[i] = Num[j], Num[j] = t;
81             }
82     while (m--)
83     {
84         fscanf(fi, "%d %d %d", &i, &j, &t);
85         M[Renum[i]][Renum[j]] = M[Renum[j]][Renum[i]] = t;
86     }
87     fclose(fi);
88 }
89
90 int main()
91 {
92     read_data();
93     solve();
94     fprintf(fopen("coach.out", "wt"), "%d %d %d %d\n", Num[Ri], Num[Rj], Rs, Re);
95     return 0;
96 }
```

34.2.3 *Rezolvare detaliată

34.3 Color

Ion și Vasile joacă un joc. Ei au la dispoziție un arbore binar strict (adică fiecare nod are 0 sau 2 fi) cu N noduri, numerotate de la 1 la N (nodul numerotat cu 1 este rădăcina arborelui). Inițial, toate nodurile sunt colorate în alb. Jucătorii vor efectua mutări alternativ, iar jucătorul aflat la mutare va colora în negru un nod colorat în alb. Ion efectuează prima mutare și poate colora în negru un singur nod (oricare) al arborelui. Considerând că ultimul nod colorat de unul dintre jucători este P , jucătorul care urmează la mutare poate colora în negru unul din următoarele noduri (dacă nu au fost deja colorate în negru):

- unul din cei 2 fi ai lui P (dacă P nu este frunză în arbore)
- tatăl lui P (dacă P nu este rădăcina arborelui)

Jocul continuă până când unul dintre jucători nu mai poate efectua nici o mutare. Atunci, jucătorul care a efectuat ultima mutare este considerat câștigător. Nodul ales de Ion la prima mutare este numit nod câștigător dacă Ion poate câștiga jocul, indiferent de mutările lui Vasile (adică Ion are strategie sigură de câștig).

Cerință

Considerând că ambii jucători joacă optim, determinați toate nodurile din arbore pe care le poate colora Ion la prima mutare, astfel încât să fie sigur de victorie (nodurile câștigătoare).

Date de intrare

Prima linie a fișierului **color.in** conține numărul întreg N , reprezentând numărul de noduri din arbore. Următoarele $N - 1$ linii conțin câte două numere întregi separate printr-un spațiu, a și b , având semnificația că a este tatăl lui b .

Date de ieșire

Pe prima linie a fișierului **color.out** veți afișa numărul întreg M , reprezentând numărul de noduri câștigătoare. Pe următoarea linie veți afișa numerele acestor noduri, în ordine crescătoare.

Restricții și precizări

$1 \leq N \leq 16000$, N impar

40% din teste vor avea $N \leq 1000$

Exemplu

color.in	color.out
9	6
1 2	1 5 6 7 8 9
1 3	
2 4	
2 5	
4 6	
4 7	
3 8	
3 9	

Timp maxim de executare/test: 0.2 secunde pentru Linux și 0.3 secunde pentru Windows

34.3.1 Indicații de rezolvare

Mugurel Ionuț Andreica

Pentru fiecare nod din arbore se calculează două valori : $WINJOS[i] = 1$, dacă jucătorul care începe are strategie sigură de câștig, în cazul în care el colorează întâi nodul i , iar al doilea jucător colorează, în continuare, unul din fiii lui i (și 0 în caz contrar), respectiv $WINSUS[i] = 1$, dacă jucătorul care începe are strategie sigură de câștig, în cazul în care el colorează întâi nodul i , iar al doilea jucător colorează, în continuare, tatăl lui i .

$WINJOS[i]$ se calculează pe baza valorilor fililor lui i , iar

$WINSUS[i]$, pe baza lui $WINSUS[tata[i]]$ și $WINJOS[frate[i]]$, unde $frate[i]$ este nodul care are același tată ca și nodul i .

Ambele valori se calculează în timp liniar.

34.3.2 Cod sursă

Listing 34.3.1: color_ok.pas

```

1 { Time Complexity : O(N) }
2
3 Program _color_;
4
5 const filein = 'color.in';
6     fileout = 'color.out';
7     MAXN = 16000;
8
9 type bytearray = array [1..MAXN] of byte;
10    intarray = array [1..MAXN] of integer;
11    pintarray = ^intarray;
12
13 var i, j, k, n, m : integer;
14     winup, window : bytearray;
15     nfii, tata, frate : pintarray;
16     fiu : array [1..2] of pintarray;
17     ok : boolean;
18
19 procedure dfdown(nod : integer);
20 begin
21   if (nfii^[nod] = 0) then

```

```

22     window[nod] := 1
23 else
24 begin
25     dfdown(fiu[1]^nod);
26     dfdown(fiu[2]^nod);
27
28     window[nod] := (1 - window[fiu[1]^nod]) and
29                     (1 - window[fiu[2]^nod]);
30 end;
31 end;
32
33 procedure dfup(nod : integer);
34 begin
35 if (nod = 1) then
36     winup[1] := 1
37 else
38     winup[nod] := window[frate^nod] or
39                     (1 - winup[tata^nod]);
40
41 if (nfii^nod > 0) then
42 begin
43     dfup(fiu[1]^nod);
44     dfup(fiu[2]^nod);
45 end;
46 end;
47
48 begin
49 assign(input, filein);
50 reset(input);
51 read(n);
52
53 new(nfii);
54 new(tata);
55 new(frate);
56 new(fiu[1]);
57 new(fiu[2]);
58
59 fillchar(nfii^, sizeof(intarray), 0);
60 fillchar(tata^, sizeof(intarray), 0);
61
62 for k := 1 to n-1 do
63 begin
64     read(i, j);
65
66     tata^[j] := i;
67     inc(nfii^[i]);
68     fiu[nfii^[i]]^i := j;
69 end;
70
71 close(input);
72
73 for i := 1 to n do
74 if (nfii^[i] = 2) then
75 begin
76     frate^[fiu[1]^i] := fiu[2]^i;
77     frate^[fiu[2]^i] := fiu[1]^i;
78 end;
79
80 dfdown(1);
81 dfup(1);
82
83 m := 0;
84 for i := 1 to n do
85 if (winup[i] and window[i] = 1) then
86     inc(m);
87
88
89 ok := false;
90 for i := 2 to n do
91 if (winup[i] and window[i] = 1) and (nfii^[i] > 0) then
92     ok := true;
93
94 if (ok) then
95     writeln('OK')
96 else
97     writeln('not OK');

```

```

98
99 assign(output, fileout);
100 rewrite(output);
101 writeln(m);
102 for i := 1 to n do
103   if (winup[i] and window[i] = 1) then
104     write(i, ' ');
105 writeln;
106 close(output);
107 end.
```

34.3.3 *Rezolvare detaliată

34.4 Magic

Misopan și Trofonaced sunt doi eroi care vor să-și unească forțele în lupta împotriva răului. Regatul este reprezentat printr-o matrice dreptunghiulară de N linii și M coloane. Fiecare element al matricei corespunde unei bucăți de teren uscat sau mlăștinos. Cei doi eroi nu se vor aventura în părțile mlăștinoase ale regatului - se vor deplasa numai pe uscat. Ei se pot muta dintr-o poziție a matricei în una din cele 4 poziții vecine pe orizontală sau pe verticală, dacă această poziție corespunde unei zone de uscat. Unele poziții de uscat pot fi transformate prin vrajă în mlaștină.

Cerință

Ajutați un vrăjitor malefic să aleagă un număr minim de poziții "transformabile", prin schimbarea cărora cei doi eroi să nu se poată întâlni (să nu existe un drum pe uscat între cei doi).

Date de intrare

Prima linie a fișierului **magic.in** conține două numere întregi N și M reprezentând numărul de linii, respectiv de coloane ale matricei. Următoarele N linii conțin câte M caractere cu următoarea semnificație:

- pentru o poziție uscată
- x (mic) pentru una mlăștinoasă
- * pentru una uscată "transformabilă" în una mlăștinoasă de către vrăjitor
- M pentru poziția eroului Misopan
- T pentru poziția eroului Trofonaced

Date de ieșire

Pe prima linie a fișierului **magic.out** se scrie numărul întreg R , reprezentând numărul minim de poziții care trebuie transformate. Pe următoarele R linii vor apărea câte 2 numere, reprezentând pozițiile alese. Primul număr va fi linia (între 1 și N), iar al doilea va fi coloana (între 1 și M).

Restricții și precizări

$1 \leq N, M \leq 50$

R (rezultatul afișat) poate fi 0

Pe teste date va exista întotdeauna soluție

Se garantează că în toată matricea caracterele M , respectiv T vor apărea fiecare exact o dată
Pozițiile eroilor sunt implicit zone de uscat care nu pot fi transformate de vrăjitor

Exemplu

magic.in	magic.out
4 4	1
MxxT	3 3
. x *	
. * *	
* * x .	

Timp maxim de executare/test: 0.6 secunde pentru Linux și 1.8 secunde pentru Windows

34.4.1 Indicații de rezolvare

Matricei date i se poate asocia un graf, construit astfel:

- pozițiilor uscate li se vor asocia câte un nod
- pozițiilor transformabile li se vor asocia câte două noduri (un nod dedublat) - unul pentru muchiile care "intra" în poziția respectiva, iar celălalt pentru cele care "ies" din poziție; se va pune o muchie de capacitate 1 între aceste două noduri.
- se vor pune muchii cu capacitate infinită între două noduri vecine în matrice (fiecare poziție are maxim 4 vecini)

Este destul de evident că problema inițială este echivalentă cu aflarea unei *tăieturi minime* în graful astfel creat. Tăietura minimă se determină folosind un *algoritm de flux maxim*.

Complexitate: $O(N^4)$. De observat că un algoritm care creează explicit graful folosește destul de multă memorie și ar putea să nu intre în timp pe toate testele. Este de preferat o implementare unde graful este implicit, fluxurile fiind ținute într-o matrice (câte 4 fluxuri pentru fiecare poziție + un flux intern pentru pozițiile deduplicate).

34.4.2 Cod sursă

Listing 34.4.1: magic.cpp

```

1 /*
2 Magic Flow Solution
3 O(N^2*M^2).
4 Optimized version (i.e. implicit graph)
5 by Radu Berinde
6 */
7
8 #ifdef __BORLANDC__
9 #pragma option -3 -O2
10 #include <alloc.h>
11 #else
12 #define far
13 #endif
14
15 #include <stdio.h>
16 #include <stdlib.h>
17 #include <string.h>
18
19 char Type[52][52];
20 short far Val[52][52];
21
22 int N, M;
23 int L1, C1, L2, C2;
24
25 int Nr;
26
27 int vi[4] = {0, -1, 1, 0};
28 int vj[4] = {-1, 0, 0, 1};
29 #define ni (i + vi[v])
30 #define nj (j + vj[v])
31 #define vinv (3-(v))
32
33 short Flow[52][52][4];
34 short F[52][52], C[52][52];
35 char tv[52][52][2];
36 char far ci[52*52*2];
37 char far cj[52*52*2];
38 char far cd[52*52*2];
39 int cl, cr;
40 int TFlow;
41
42 #define INF 32666
43
44 int bf()
45 {
46     int i, j, d, v;
47     memset(tv, -1, sizeof(tv));
48     for (i = 0; i <= M; i++)
49         tv[0][i][0] = tv[0][i][1] = tv[N+1][i][0] = tv[N+1][i][1] = 13;
50     for (i = 0; i <= N; i++)
51         tv[i][0][0] = tv[i][0][1] = tv[i][M+1][0] = tv[i][M+1][1] = 13;
52 }
```

```

53     ci[0] = L1, cj[0] = C1, cd[0] = 1;
54     tv[L1][C1][1] = 5;
55
56     for (cl = cr = 0; cl <= cr && tv[L2][C2][0] == -1; cl++)
57     {
58         i = ci[cl], j = cj[cl], d = cd[cl];
59
60         if (tv[i][j][!d] == -1 &&
61             ((!d && F[i][j] < C[i][j]) || (d & F[i][j])))
62             tv[i][j][!d] = 4, ci[++cr] = i, cj[cr] = j, cd[cr] = !d;
63
64         for (v = 0; v < 4; v++)
65             if (tv[ni][nj][!d] == -1 && (d || Flow[ni][nj][vinv]))
66                 tv[ni][nj][!d] = vinv, ci[++cr] = ni, cj[cr] = nj, cd[cr] = !d;
67     }
68     if (tv[L2][C2][0] == -1) return 0;
69     for (i = L2, j = C2, d = 0; i != L1 || j != C1;)
70     {
71         v = tv[i][j][d];
72         if (v == 4)
73         {
74             F[i][j] += d ? +1 : -1;
75             d = !d;
76             continue;
77         }
78         if (d)
79             Flow[i][j][v]--;
80         else
81             Flow[ni][nj][vinv]++;
82         i = ni, j = nj, d = !d;
83     }
84     return 1;
85 }
86
87 void solve()
88 {
89     while (bf())
90         TFlow++;
91 }
92
93 void print()
94 {
95     int i, j;
96     FILE *fo = fopen("magic.out", "wt");
97     fprintf(fo, "%d\n", TFlow);
98     for (i = 1; i <= N; i++)
99         for (j = 1; j <= M; j++)
100            if (Type[i][j] == 2 && tv[i][j][0] != -1 && tv[i][j][1] == -1)
101                fprintf(fo, "%d %d\n", i, j);
102     fclose(fo);
103 }
104
105 void read_data()
106 {
107     int i, j;
108     char c;
109     FILE *fi = fopen("magic.in", "rt");
110     fscanf(fi, "%d %d", &N, &M);
111     for (i = 1; i <= N; i++)
112         for (j = 1; j <= M; j++)
113         {
114             fscanf(fi, " %c", &c);
115             Type[i][j] = (c != 'x' && c != 'X') + (c == '*');
116             if (c == 'M') L1 = i, C1 = j, Type[i][j] = -1;
117             if (c == 'T') L2 = i, C2 = j, Type[i][j] = -2;
118             C[i][j] = (Type[i][j] == 0) ? 0 : (Type[i][j] == 2) ? 1 : INF;
119         }
120     fclose(fi);
121 }
122
123 int main()
124 {
125     read_data();
126     solve();
127     print();
128     return 0;

```

129 }

Listing 34.4.2: magicsl.cpp

```

1 /*
2 Magic Flow Solution
3 O(N^2*M^2).
4 Un-optimized version (i.e. explicit graph)
5 Should NOT earn maximum points.
6 Uses more memory than borland can handle.
7 by Radu Berinde
8 */
9
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <string.h>
13
14 char Type[52][52];
15 short Val[52][52];
16
17 int N, M;
18 int L1, C1, L2, C2;
19
20 int Nr;
21
22 typedef struct edge
23 {
24     unsigned short v;
25     short c, f;
26     struct edge *next, *rev;
27 } edge;
28
29 edge * V[52*52*2];
30 short tata[52*52*2];
31 edge * tedge[52*52*2];
32 short coada[52*52*2];
33 int cl, cr;
34 int Flow;
35
36 #define INF 32666
37
38 int bf()
39 {
40     edge *e;
41     int i;
42     coada[0] = 1;
43     memset(tata, 0, sizeof(tata));
44
45     for (cl = cr = 0; cl <= cr && !tata[2]; cl++)
46         for (e = V[coada[cl]]; e; e = e->next)
47             if (e->f < e->c && !tata[e->v])
48             {
49                 coada[++cr] = e->v;
50                 tata[e->v] = coada[cl];
51                 tedge[e->v] = e;
52             }
53     if (!tata[2]) return 0;
54
55     for (i = 2; i != 1; i = tata[i])
56         tedge[i]->f++, tedge[i]->rev->f--;
57     return 1;
58 }
59
60 void solve()
61 {
62     while (bf())
63         Flow++;
64 }
65
66 void print()
67 {
68     int i, j;
69     FILE *fo = fopen("magic.out", "wt");
70     fprintf(fo, "%d\n", Flow);
71     for (i = 1; i <= N; i++)

```

```

72     for (j = 1; j <= M; j++)
73         if (Type[i][j] == 2 && tata[Val[i][j]] && !tata[Val[i][j]+1])
74             fprintf fo, "%d %d\n", i, j);
75         fclose(fo);
76     }
77 }
78 void add_edge(int i, int j, int c)
79 {
80     edge *e1, *e2;
81     e1 = (edge *) malloc(sizeof(edge));
82     e2 = (edge *) malloc(sizeof(edge));
83     e1->rev = e2, e2->rev = e1;
84     e1->c = c, e2->c = 0;
85     e1->f = 0, e2->f = 0;
86     e1->v = j, e2->v = i;
87     e1->next = V[i], e2->next = V[j];
88     V[i] = e1, V[j] = e2;
89 }
90
91 int vi[4] = {0, -1, 0, 1};
92 int vj[4] = {-1, 0, 1, 0};
93 #define ni (i + vi[v])
94 #define nj (j + vj[v])
95
96 void create_graph()
97 {
98     int i, j, v;
99     Val[L1][C1] = 1;
100    Val[L2][C2] = 2;
101    Nr = 2;
102
103    for (i = 1; i <= N; i++)
104        for (j = 1; j <= M; j++)
105            if (Type[i][j] > 0)
106            {
107                Val[i][j] = Nr+1;
108                if (Type[i][j] == 2)
109                    add_edge(Nr+1, Nr+2, 1);
110                Nr += Type[i][j];
111            }
112
113    for (i = 1; i <= N; i++)
114        for (j = 1; j <= M; j++)
115            if (Type[i][j] > 0 || Type[i][j] == -1)
116            {
117                for (v = 0; v < 4; v++)
118                    if (Val[ni][nj])
119                        add_edge(Val[i][j] + (Type[i][j] == 2), Val[ni][nj], INF);
120            }
121 }
122
123 void read_data()
124 {
125     int i, j;
126     char c;
127     FILE *fi = fopen("magic.in", "rt");
128     fscanf(fi, "%d %d", &N, &M);
129     for (i = 1; i <= N; i++)
130         for (j = 1; j <= M; j++)
131         {
132             fscanf(fi, " %c", &c);
133             Type[i][j] = (c != 'x' && c != 'X') + (c == '*');
134             if (c == 'M') L1 = i, C1 = j, Type[i][j] = -1;
135             if (c == 'T') L2 = i, C2 = j, Type[i][j] = -2;
136         }
137     fclose(fi);
138 }
139
140 int main()
141 {
142     read_data();
143     create_graph();
144     solve();
145     print();
146     return 0;
147 }
```

34.4.3 *Rezolvare detaliată

34.5 Pătrate

Ovi este un băiețel foarte isteț căruia îi place să scrie pe asfalt cu creta și să șopăie. El desenează cu cretă roșie un dreptunghi de lățime exact 2 metri și lungime N metri, pe care îl împarte în pătrate egale de latură 1 metru, unele laturi interioare fiind desenate cu cretă roșie, iar restul laturilor interioare cu cretă albă. Ovi pornește din pătratul aflat în colțul stânga sus al dreptunghiului, sărind dintr-un pătrat în altul vecin pe linie sau coloană, cu condiția ca latura care desparte cele două pătrate să nu fie colorată în roșu. El își dorește ca prin sărituri succesive să ajungă în toate pătratele dreptunghiului, dar a observat că numai pentru anumite variante de colorare a laturilor pătratelor reușește acest lucru.

În exemplele de mai jos (cu $N = 2$) liniile interioare îngroșate sunt colorate cu roșu, iar cele punctate sunt colorate cu alb. La exemplul din fig. 1, pornind din colțul stânga sus se poate ajunge în oricare alt pătrat, dar în exemplul din fig. 2 nu se poate ajunge la pătratele din partea dreaptă.

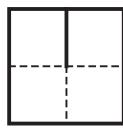


fig. 1

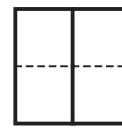


fig. 2

Figura 34.1: Patrate1

Cerință

Ajutați-l pe Ovi să numere câte posibilități de colorare în roșu a unor laturi interioare ale pătratelor sunt astfel încât plecând din colțul stânga sus să poată ajunge prin sărituri în oricare alt pătrat.

Date de intrare

În fișierul **patrate.in** se află un singur număr natural N ce reprezintă lungimea în metri a dreptunghiului.

Date de ieșire

În fișierul **patrate.out** veți scrie un singur număr natural (urmat de caracterul de sfârșit de linie) ce reprezintă numărul de posibilități cerut.

Restricții și precizări

$$2 \leq N \leq 1000$$

Exemplu

patrate.in	patrate.out
2	5

Explicație:

Cele 5 posibilități sunt:

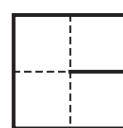
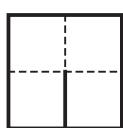
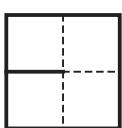
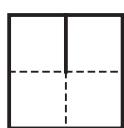
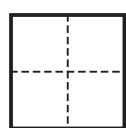


Figura 34.2: Patrate2

Timp maxim de executare/test: 0.2 secunde pentru Linux și 0.4 secunde pentru Windows

34.5.1 Indicații de rezolvare

Dan Pracsu

Să notăm numărul căutat cu X_N (numărul posibilităților pentru matricea $2 \times N$). Relația de recurență este

$$X_N = 5 * X_{N-1} - 2 * X_{N-2}$$

Este vorba de o relație de conexitate în matrice. La trecerea de la o matrice cu N linii la una cu $N + 1$ linii trebuie avut grijă la faptul că dintr-o matrice de lungime N neconexă se poate obține o matrice conexă (dacă la ultima coloană nu așez nicio linie roșie).

Pentru $n \leq 15$ se obține un rezultat care se încadrează în *long*. În rest se lucrează cu *numere mari*. Era suficientă pentru încadrarea în timp doar a operației de adunare.

Aplicarea unui algoritm *backtracking* ar fi obținut 20 sau 30 puncte.

Complexitatea: $O(n^2)$

34.5.2 Cod sursă

Listing 34.5.1: patrate.c

```

1  /*
2  Patrate O(N^2) solution.
3  by Radu Berinde
4  */
5
6 #ifdef __BORLANDC__
7 #pragma option -3 -O2
8 #endif
9
10 #include <stdio.h>
11 #include <string.h>
12
13 #define MAXC 1000
14
15 void add(int *A, int *B)
16 {
17     int i, t = 0;
18     for (i = 1; i <= A[0] || i <= B[0] || t; A[i++] %= 10)
19         t = (A[i] += B[i] + t) / 10;
20     A[0] = i-1;
21 }
22
23 int N;
24
25 int A[MAXC], L[MAXC], T[MAXC], S1[MAXC];
26
27 int main()
28 {
29     int i;
30     freopen("patrate.ok", "wt", stdout);
31     fscanf(fopen("patrate.in", "rt"), "%d", &N);
32     A[0] = A[1] = 1;
33     S1[0] = S1[1] = 1;
34     for (i = 2; i <= N; i++)
35     {
36         memcpy(L, A, sizeof(A));
37         add(A, L), add(A, L), add(A, L), add(A, T);
38         add(A, S1);
39
40         add(T, L), add(T, L);
41     }
42
43     for (i = A[0]; i; i--)
44         printf("%d", A[i]);
45     printf("\n");
46     return 0;
47 }
```

Listing 34.5.2: psmall.c

```

1  /*
2  Patrate small number solution
3  by Radu Berinde
4  */
5
6 #include <stdio.h>
7
8 long A[1002], T;
9 int N;
10
11 int main()
12 {
13     int i;
14     fscanf(fopen("patrate.in", "rt"), "%d", &N);
15     A[1] = 1, T = 0;
16     for (i = 2; i <= N; i++)
17     {
18         A[i] = 4 * A[i-1] + T + 1;
19         T += 2 * A[i-1];
20     }
21
22     fprintf(fopen("patrate.out", "wt"), "%ld", A[N]);
23     return 0;
24 }
```

34.5.3 *Rezolvare detaliată

34.6 Turnuri

Renumitul arhitect Prăbușilă dorește să construiască unul din cele mai interesante turnuri de pe planetă. Acest turn, în mod cu totul deosebit, va avea etaje de diverse lățimi, între 1 și 100, numere întregi.

Prăbușilă s-a hotărât deja ce dimensiune va avea fiecare din etajele turnului, dar nu și cum să le așeze pe orizontală. El ar dori mai întâi să știe câte variante are.

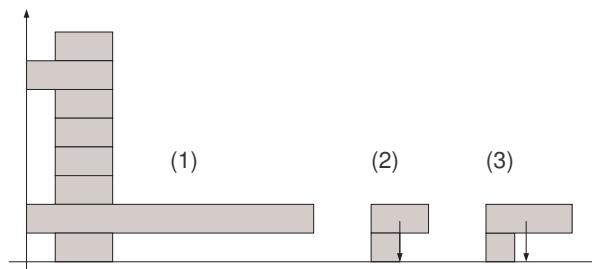


Figura 34.3: Turnuri1

Etajele pot fi așezate la coordonate întregi și va trebui ca un astfel de turn să nu se dărâme.

Condiția pentru ca un turn să fie stabil este ca la fiecare etaj perpendiculara coborâtă din centrul de greutate al grupului etajelor superioare să cadă strict în interiorul acelui etaj (nu are voie să fie pe margini sau în afară - de ex. turnurile 2 și 3 sunt instabile).

Centrul de greutate al unui etaj se află la mijlocul etajului respectiv.

Centrul de greutate al unui grup de etaje are drept coordonată x (orizontală) media coordonatelor x ale centrelor de greutate ale etajelor componente. (Etajele au mase egale, indiferent de cât de late sunt).

În exemplul 1, etajul din vîrf are coordonata x a centrului de greutate 2, iar grupul celor 2 etaje din vîrf are centrul de greutate la coordonata $x = 1.75$ (media aritmetică între 2 și 1.5).

Se observă în figura 1 că, deși perpendiculara din centrul de greutate al etajului 2 cade în afară etajului 1, totuși turnul este stabil, deoarece perpendiculara din centrul de greutate al grupului format din etajele 2 – 8 cade strict în interiorul etajului 1.

Cerință

Să se afle câte turnuri stabile există.

Date de intrare

Fișierul de intrare **turnuri.in** conține pe o singură linie lista de numere naturale separate prin câte un spațiu, numere reprezentând lățimile etajelor turnului, începând cu cel mai de sus. Lista se termină cu un 0.

Date de ieșire

Fișierul de ieșire **turnuri.out** conține numărul de turnuri.

Restricții și precizări:

- numărul maxim de turnuri nu va depăși 2 miliarde
- numărul maxim de etaje ale unui turn este 200
- lățimea maximă a unui etaj este 100

Exemplu:

turnuri.in	turnuri.out
1 3 4 1 0	6

Cele 6 variante sunt:

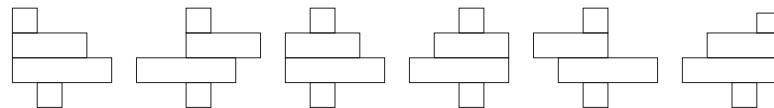


Figura 34.4: Turnuri2

Timp maxim de executare/test: 0.2 secunde pentru Linux și 0.2 secunde pentru Windows

34.6.1 Indicații de rezolvare

Radu Andrei Ștefan

Programare dinamică.

Construim soluția de sus în jos.

Pentru fiecare nivel "n", se observă că partea fracționară a coordonatei centrului de greutate poate fi $k/(2 \cdot \text{numărul de etaje din care facem media}=n)$ $k \in [0, 2n - 1]$.

Vom reține într-un vector câte etaje există pentru fiecare fracție.

Astfel, pentru etajul cel mai de sus: numărul de turnuri formate numai din acest etaj cu centrul de greutate cu partea fracționară $1/2$ este 1 iar numărul de turnuri cu centrul de greutate cu parte fracționară 0 este 0 (dacă etajul are lungime impară).

Construim posibilitățile de la nivelul $n + 1$ prin sumarea elementelor de la nivelul n (etajele sunt numărate de sus în jos), pentru toate pozițiile pe orizontală etajului $n+1$.

Atenție, fracțiile se schimbă de la un etaj la altul din $k/(2n)$ în $k/(2n+2)$.

Complexitate: $O(\text{înălțime} * \text{suma lungimilor})$.

34.6.2 Cod sursă

Listing 34.6.1: turnuri.cpp

```

1 #include <stdio.h>
2
3 FILE *fin=fopen("turnuri.in","r");
4 FILE *fout=fopen("turnuri.out","w");
5
6 int n;
7 int a[300],r[300];
8 long double res=0;
9
10 void read()
11 {
12     int i;
13     n=0;
```

```

14     while (1)
15     {
16         fscanf(fin,"%d",&i);
17         if (!i) break;
18         a[n++]=i;
19     }
20 }
21
22 int modn(int x, int n)
23 {
24     while (x<0) x+=n;
25     return x%n;
26 }
27
28 void solve()
29 {
30     int i,j,k,l;
31     long double count[500],c2[500];
32     for (i=0; i<500; i++) count[i]=0;
33     if (a[0]%2==0) count[0]=1; else count[1]=1;
34     for (l=1; l<n; l++)
35     {
36         for (j=0; j<n*2; j++) c2[j]=0;
37         k=a[l];
38         for (i=0; i<k; i++) for (j=0; j<2*l; j++) if (i!=0 || j!=0)
39             c2[modn(j+k-2*i,2*(l+1))]+=count[j];
40         res=0;
41         for (j=0; j<n*2; j++) res+=count[j]-c2[j];
42     }
43 }
44
45 void write()
46 {
47     fprintf(fou,"%.0lf\n", (double)res);
48 }
49
50 int main(int argc, char *argv[])
51 {
52     read();
53     solve();
54     write();
55     return 0;
56 }
```

34.6.3 *Rezolvare detaliată

Capitolul 35

ONI 2003

35.1 Asmin

Se consideră un arbore (graf conex aciclic) cu N vârfuri, fără rădăcină fixată. Drept rădăcină, poate fi ales oricare dintre vârfuri. Să presupunem că a fost ales vârful cu numărul T . Între oricare vârf și T există un drum unic care conține fiecare vârf al arborelui cel mult o singură dată (un drum între vârfurile i și j este o secvență de vârfuri, care începe cu i , se termină cu j , iar între oricare două vârfuri consecutive există o muchie în arbore). Fiecare vârf i (inclusiv T) trebuie să i se asocieze o valoare V_i , mai mare sau egală cu 0, astfel încât suma valorilor vârfurilor de pe drumul dintre i și rădăcina T , împărțită la K , să dea restul R_i . Se definește costul arborelui cu rădăcina fixată în T , C_T , ca fiind suma valorilor asociate fiecărui nod. Dintre toate posibilitățile de alegere a valorilor V_i care respectă condiția precizată anterior, se va alege aceea pentru care C_T este minim.

Se constată ușor că alegând alt vârf drept rădăcină, de exemplu, vârful S (diferit de T), C_S nu este neapărat egal cu C_T .

Cerință

Dându-se un arbore cu N vârfuri, un număr întreg K și valorile R_i , $i = 1, 2, \dots, N$, corespunzătoare fiecărui vârf, determinați acele vârfuri T care pot fi alese drept rădăcină, pentru care costul C_T este minim (adică $C_T \leq C_S$, oricare ar fi S diferit de T), precum și costul respectiv.

Date de intrare

Pe prima linie a fișierului de intrare **asmin.in** se află 2 valori întregi: N și K . Pe următoarele $N - 1$ linii se află câte două numere întregi a și b , separate printr-un spațiu, având semnificația că există muchie între vârfurile a și b . Vâfurile sunt numerotate de la 1 la N . Pe următoarea linie se află N numere întregi, reprezentând valorile R_i , $i = 1, 2, \dots, N$.

Date de ieșire

Pe prima linie a fișierului de ieșire **asmin.out** se vor afișa două valori întregi: C și M . C reprezintă costul minim posibil al arborelui. M reprezintă numărul de vârfuri care pot fi alese drept rădăcină și pentru care se obține costul C . Pe a doua linie se află M numere întregi separate prin câte un spațiu, scrise în ordine crescătoare, reprezentând numerele vârfurilor ce pot fi alese ca rădăcină astfel încât să se obțină costul C .

Restricții și precizări

$2 \leq N \leq 16000$

$2 \leq K \leq 1000$

$0 \leq R_i \leq K - 1$

Cel puțin 40% din testele folosite la evaluare vor avea $N \leq 1000$

Exemplu

asmin.in	asmin.out
5 3	5 2
1 2	1 5
1 3	
2 4	
2 5	
0 1 2 1 0	

Cei doi arbori obținuți (împreună cu valorile asociate vârfurilor) sunt următorii:

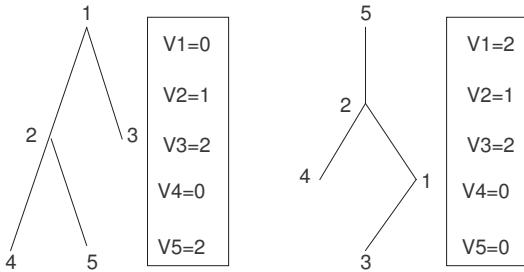


Figura 35.1: Asmin

Timp maxim de execuție: 0.2 secunde/test

35.1.1 Indicații de rezolvare

Există o soluție simplă, cu complexitatea $O(N^2)$.

Această soluție încearcă să aleagă fiecare nod drept rădăcină și pentru rădăcina astfel fixată, calculează în $O(N)$ valorile asociate nodurilor. Dacă rădăcina este nodul i , atunci valoarea rădăcinii este R_i .

Pentru fiecare alt nod, valoarea asociată lui este cea mai mică valoare astfel încât suma valorilor până la tatăl lui în arborele cu rădăcina fixată plus această valoare să dea restul cerut, la împărțirea cu numărul K .

Valorile sunt atribuite nodurilor de la rădăcină către frunze. Astfel, o simplă *parcursere DF* este de ajuns, odată ce rădăcina a fost fixată.

Soluția optimă are complexitatea $O(N)$.

Se fixează întâi ca rădăcină nodul 1 și se calculează valorile asociate nodurilor arborelui având rădăcina în nodul 1, conform algoritmului descris mai sus. Astfel se obține

$C[1]$ = suma valorilor nodurilor arborelui când nodul 1 este rădăcină.

Pentru a calcula $C[i]$ (i diferit de 1) nu este nevoie să repetăm parcurserea DF pentru nodul i considerat rădăcină. În schimb, încă o parcursere DF din nodul 1 este suficientă pentru a calcula costurile fiecărui nod.

Să presupunem că vrem să calculăm $C[i]$ și am calculat deja $C[j]$, unde j este tatăl lui i , în arborele având rădăcina fixată în nodul 1.

Atunci,

$$C[i] = C[j] - R[j] - \min(R[j], R[i]) + R[i] + \min(R[i], R[j]),$$

unde $\min(a, b)$ (cu $0 \leq a, b \leq K - 1$) calculează valoarea x minima ($x \geq 0$), astfel încât $a + x = b$ ($\bmod K$).

Se alege minimul din vectorul C și se tipăresc nodurile i având valoarea $C[i]$ egală cu minimul.

35.1.2 Cod sursă

Listing 35.1.1: asmin.cpp

```

1 /* Mugurel Ionut Andreica - Bucuresti, ROMANIA */
2 /* Time Complexity: O(N) */
3 #include <stdio.h>

```

```

4 #include <string.h>
5
6 #define maxn 1001
7 #define filein "asmin.in"
8 #define fileout "asmin.out"
9
10 typedef struct nod { long x; nod *urm; } *pnod;
11
12 pnod edge[maxn],aux;
13 long v[maxn],marc[maxn],minim[maxn],tata[maxn];
14 long i,j,k,n,root,sum,needed,min,nsol=0,modulo;
15
16 void read();
17 void compute();
18 void print();
19 void df(long);
20 void df2(long);
21 long extra(long,long);
22
23 int main()
24 {
25     read();
26     compute();
27     print();
28     return 0;
29 }
30
31 void read()
32 {
33     freopen(filein,"r",stdin);
34     scanf("%ld %ld",&n,&modulo);
35
36     for (i=1;i<=n;i++)
37         edge[i]=NULL;
38
39     for (k=1;k<n;k++)
40     {
41         scanf("%ld %ld",&i,&j);
42
43         aux=new nod;
44         aux->x=j;
45         aux->urm=edge[i];
46         edge[i]=aux;
47
48         aux=new nod;
49         aux->x=i;
50         aux->urm=edge[j];
51         edge[j]=aux;
52     }
53
54     for (i=1;i<=n;i++)
55         scanf("%ld",&v[i]);
56 }
57
58 void compute()
59 {
60     memset(marc,0,sizeof(marc));
61     df(1);
62
63     minim[1]=min=needed;
64
65     memset(marc,0,sizeof(marc));
66     df2(1);
67 }
68
69 void print()
70 {
71     freopen(fileout,"w",stdout);
72
73     for (i=1,nsol=0;i<=n;i++)
74         if (minim[i]==min)
75             nsol++;
76
77     printf("%ld %ld\n",min,nsol);
78
79     k=1;

```

```

80  for (i=1;i<=n;i++)
81    if (minim[i]==min)
82    {
83      if (!k) printf(" ");
84      printf("%ld",i);
85      k=0;
86    }
87
88  printf("\n");
89 }
90
91 void df(long nod)
92 {
93  pnod aa=edge[nod];
94  long i,suminit=sum;
95
96  marc[nod]=1;
97
98  needed+=extra(sum,v[nod]);
99  sum=v[nod];
100
101 aa=edge[nod];
102 while (aa!=NULL)
103 {
104   i=aa->x;
105
106   if (!marc[i])
107     df(i);
108
109   aa=aa->urm;
110 }
111
112 sum=suminit;
113 }
114
115 void df2(long nod)
116 {
117  pnod aa=edge[nod];
118  long i,suminit=sum;
119
120  marc[nod]=1;
121  aa=edge[nod];
122  while (aa!=NULL)
123  {
124    i=aa->x;
125
126    if (!marc[i])
127    {
128      minim[i]=minim[nod]-v[nod]-extra(v[nod],v[i])
129                  +v[i]+extra(v[i],v[nod]);
130
131      if (minim[i]<min)
132        min=minim[i];
133
134      df2(i);
135    }
136
137    aa=aa->urm;
138  }
139 }
140
141 long extra(long v1,long v2)
142 {
143  if (v1<=v2)
144    return (v2-v1);
145  else
146    return (modulo-v1+v2);
147 }

```

35.1.3 *Rezolvare detaliată

35.2 Căutare

Stim cu toții ce este un arbore binar de căutare. Este acel arbore binar în care informația din orice nod este mai mare decât informațiile nodurilor din subarborele stâng al nodului respectiv și mai mică decât cele din subarborele drept.

Când se caută o informație într-un arbore binar de căutare, începem din rădăcina arborelui și comparăm cu informația din rădăcină. Dacă informația căutată e mai mică decât informația din rădăcină, se continuă căutarea în subarborele stâng, iar dacă e mai mare în subarborele drept. Dacă cele două informații sunt egale căutarea se termină cu succes. Dacă în direcția în care continuăm căutarea (stânga sau dreapta) subarborele nu mai are noduri înseamnă că informația căutată nu se găsește în arbore. Evident, numărul de comparații efectuate la o căutare depinde de distanța dintre rădăcină și nodul în care se găsește informația, respectiv cel la care putem decide că informația nu se află în arbore.

Ce s-ar întâmpla dacă înainte de a construi arborele binar de căutare am ști ce informații urmează să fie căutate în el? Nu cumva am putea construi arborele în aşa fel încât să minimizăm numărul de comparații efectuate?

De exemplu cu informațiile 1, 2 și 3 putem construi un arbore binar de căutare în următoarele 3 moduri (din totalul de 5 posibile):

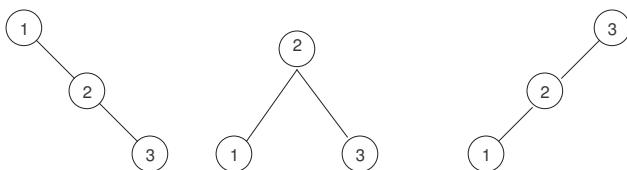


Figura 35.2: Căutare

Dacă vom căuta informațiile (1, 1, 3, 1, 2), atunci timpul total de căutare este pentru arborele din stânga $1 + 1 + 3 + 1 + 2 = 8$, pentru arborele din centru $2 + 2 + 2 + 2 + 1 = 9$, iar pentru arborele din dreapta $3 + 3 + 1 + 3 + 2 = 12$. Deci arborele din stânga este cel adecvat pentru căutările noastre, iar timpul total de căutare minim este 8.

De remarcat că dacă se caută o informație care nu există în arbore atunci numărul de comparații efectuate la căutare este egal cu nivelul ultimului nod interogat. De exemplu, dacă se caută informația 4 pe cei trei arbori atunci timpii vor fi: 3, 2, respectiv 1 (de la stânga la dreapta).

Cerință

Scrieți un program care, pentru anumite informații căutate, determină timpul total de căutare minim.

Date de intrare

Din fișierul **cautare.in** se citește de pe prima linie numărul T de teste. În fișier urmează cele T teste. Pentru fiecare test în fișier sunt scrise următoarele linii:

- prima linie conține N , numărul de noduri ale arborelui de căutare și M numărul de interogări, separate prin spațiu;
- pe următoarea linie urmează N numere întregi distincte, separate prin câte un spațiu, reprezentând informațiile din nodurile arborelui
- pe fiecare dintre următoarele M linii sunt câte 2 numere întregi separate printr-un spațiu, reprezentând un număr căutat și respectiv de câte ori a fost căutat (între 1 și 100).

Date de ieșire

În fișierul **cautare.out** se va scrie, pentru fiecare test câte o linie care conține timpul total minim de căutare pentru interogările din testul respectiv.

Restricții

- $0 < T, N < 101; 0 < M < 1001$
- informațiile nodurilor arborelui sunt numere întregi din intervalul $[-10000, 10000]$
- informațiile căutate sunt numere întregi din intervalul $[-1000000, 1000000]$ și pot fi căutate de un număr de ori cuprins între 1 și 100.

Exemplu

cautare.in	cautare.out	Explicații
3	8	Primul test este cel discutat.
3 3	251	În al doilea test se interoghează 1 de 50 de ori, 2 de 49 de ori și 3 de 51 de ori.
1 2 3	22	Cel mai bun rezultat se obține în cazul arborelui din centru, și anume 251 ($2*50+49+2*51$).
1 3		Al treilea test conține o interogare a lui 4 (care nu se află în arbore) de 20 de ori.
2 1		Evident că cel mai bun e arborele din dreapta, pe care ne dăm seama repede că 4 nu se află în arbore. Timpul total este ($1*20+2$).
3 1		
3 3		
1 2 3		
1 50		
2 49		
3 51		
3 2		
1 2 3		
2 1		
4 20		

Observații:

Pentru un fișier se ia punctajul maxim dacă toate testelete din fișier sunt corecte, altfel 0 puncte
Pentru 5 fișiere de test (din 10) $T < 4$.

Timpul maxim de execuție pentru un fișier de test: 1.5 secunde/Windows respectiv 0.3 secunde/Linux.

35.2.1 Indicații de rezolvare

Se citesc nodurile, se sortează.

Se citesc interogările și (prin *căutări binare*) se adaugă la nodul respectiv sau la intervalul dintre nodurile respective. Astfel se obțin niște ponderi ale nodurilor și intervalelor dintre ele.

Dinamica: se folosește o matricea $a[i, j] =$ costul minim pentru a forma un *arbore de căutare* cu nodurile de la i la j . Pentru aceasta se încearcă fixarea rădăcinii în punctul k (între i și j), iar costul arborelui se determină pe baza lui $a[i, k]$ și $a[k + 1, j]$.

Se află matricea, și se afisează $a[1, n]$, dar $O(n^3)$.

Pentru a reduce complexitatea la $O(n^2)$ trebuie să ne dăm seama că:

rădăcina arborelui de la i la $j - 1 \leq$ radacina arborelui de la i la $j \leq$ radacina arborelui de la $i + 1$ la j .

35.2.2 Cod sursă

Listing 35.2.1: cautare.pas

```

1 var p1,p2,mij,k,d,nt,n,m,b,t,nr,freq,i,j:integer;
2   f:text;
3   p,q:array[0..100] of longint;
4   vf:array[1..100] of longint;
5   a:array[0..101,0..101] of longint;
6   r:array[1..100,1..100] of byte;
7
8 procedure rezolva;
9 var w:array[1..100,1..100] of longint;
10 begin
11   for i:=1 to n do w[i,i]:=p[i]+q[i-1]+q[i];
12   for i:=1 to n-1 do for j:=i+1 to n do w[i,j]:=w[i,j-1]+p[j]+q[j];
13   for i:=1 to n do a[i,i]:=w[i,i];
14   for i:=1 to n do r[i,i]:=i;
15   for d:=1 to n-1 do for i:=1 to n-d do begin
16     a[i,i+d]:=maxlongint;
17     for k:=r[i,i+d-1] to r[i+1,i+d] do begin
18       {for k:=i to i+d do begin}
19         if a[i,i+d]>a[i,k-1]+a[k+1,i+d]+w[i,i+d] then begin
20           a[i,i+d]:=a[i,k-1]+a[k+1,i+d]+w[i,i+d];
21           r[i,i+d]:=k;
22         end;
23     end;

```

```

24 end;
25 end;
26
27 begin
28 for i:=0 to 101 do for j:=0 to 101 do a[i,j]:=0;
29 assign(output,'cautare.out');rewrite(output);
30 assign(f,'cautare.in');reset(f);
31 read(f,nt);
32 for t:=1 to nt do begin
33 read(t,n);
34 read(f,m);
35 for i:=1 to n do read(f,vf[i]);
36 for i:=1 to n-1 do for j:=i+1 to n do if vf[i]>vf[j] then begin
37     b:=vf[i];vf[i]:=vf[j];vf[j]:=b;
38 end;
39 for i:=1 to n do p[i]:=0;
40 for i:=0 to n do q[i]:=0;
41 for j:=1 to m do begin
42     read(f,nr,freq);
43 {   for i:=1 to n do if nr=vf[i] then p[i]:=p[i]+freq;
44     for i:=1 to n-1 do if (nr>vf[i]) and (nr<vf[i+1]) then q[i]:=q[i]+freq;
45     if nr<vf[1] then q[0]:=q[0]+freq;
46     if nr>vf[n] then q[n]:=q[n]+freq; }
47 p1:=1;p2:=n;
48 while p1<=p2 do begin
49     mij:=(p1+p2) div 2;
50     if vf[mij]=nr then break;
51     if vf[mij]<nr then p1:=mij+1;
52     if vf[mij]>nr then p2:=mij-1;
53 end;
54 if vf[mij]=nr then begin p[mij]:=p[mij]+freq;continue;end;
55 if nr<vf[1] then begin q[0]:=q[0]+freq;continue;end;
56 if nr>vf[n] then begin q[n]:=q[n]+freq;continue;end;
57
58 p1:=1;p2:=n-1;
59 while p1<=p2 do begin
60     mij:=(p1+p2) div 2;
61     if (vf[mij]<nr) and (vf[mij+1]>nr) then break;
62     if vf[mij]<nr then p1:=mij+1;
63     if vf[mij]>nr then p2:=mij-1;
64 end;
65 if (vf[mij]<nr) and (vf[mij+1]>nr) then begin
66     q[mij]:=q[mij]+freq;
67     continue;
68 end;
69 writeln('NASPA');
70 end;
71 rezolva;
72 writeln(a[1,n]);
73 end;
74 close(f);
75 close(output);
76 end.

```

35.2.3 *Rezolvare detaliată

35.3 A007

Agentul **007** are de distrus o tabără de teroriști. Tabăra de teroriști este formată din mai multe obiective (depozite de muniție, pavilioane pentru teroriști, etc.), considerate punctiforme în plan. Agentul **007** primește de la serviciul de informații o hartă cu n obiective din tabăra teroriștilor, date prin coordonatele carteziene. Pe lângă hartă, agentul **007** mai primește și o armă specială (construită pentru această misiune). Arma primită are două țevi și permite tragerea simultană pe aceeași direcție (rectilinie), dar în sens invers a două rachete cu aceeași viteză. După ce se trage cu arma, odată cu atingerea unei ținte explodează și cealaltă rachetă (chiar dacă aceasta din urmă nu și-a atins țintă).

Cerință

Agentul **007** vrea să distrugă tabăra cât mai repede și cu cât mai puține rachete, pentru acest lucru el studiază posibilitatea să se așeze într-un punct din tabără (diferit de obiective) care să permită trageri eficace, adică la fiecare tragere să distrugă câte două obiective simultan.

Determinați dacă este posibil să se găsească un astfel de punct.

Date de intrare

În fișierul **a007.in** pe prima linie se află numărul de teste k , după care urmează date pentru fiecare test. Pentru fiecare test pe o linie se află n , iar pe următoarele n linii sunt coordonatele obiectivelor din tabăra teroriștilor (separate printr-un spațiu în ordinea abscisă ordonată).

Date de ieșire

În fișierul **a007.out** se vor scrie k linii, pe fiecare linie se va scrie 1, dacă există soluție și 0 dacă nu există soluție. În cazul în care există soluție se va scrie înapoi continuare pe aceeași linie, separate printr-un spațiu, coordonatele punctului cerut (numere reale trunchiate la 4 zecimale, în ordinea abscisă ordonată).

Restrictions

$$0 \leq n \leq 10000$$

$$1 \leq k \leq 3$$

coordonatele punctelor sunt întregi din intervalul $[-10000, 10000]$

Observație

Un obiectiv este distrus dacă racheta explodează exact în punctul corespunzător lui.

Exemplu

a007.in	a007.out
2	1 5.0000 5.0000
4	0
10 0	
10 10	
0 10	
0 0	
6	
0 0	
10 0	
2 10	
12 0	
5 0	
7 0	

Timp maxim de execuție/test: 0.2 secunde (pentru Windows și Linux)

35.3.1 Indicații de rezolvare

Se observă că problema se reduce la găsirea unui *centru de simetrie* într-o mulțime de n puncte, cu coordonate întregi.

Punctul de simetrie pentru mulțimea de puncte are proprietatea că simetricul fiecărui punct din mulțime față de centrul de simetrie este tot un punct din mulțimea de puncte. De aici obținem că pentru a exista centru de simetrie este necesar ca n să fie par.

Acum dacă n este par, observăm că centrul de simetrie este mijlocul a $n/2$ segmente cu capete în aceste puncte și fiecare punct din mulțime este capăt pentru un singur segment.

Notăm cu x și y vectorii ce rețin coordonatele celor n puncte.

Ordonăm punctele crescător după abscise punctele, la abscise egale vom ordona crescător după ordonată.

Pentru un segment cu capetele de coordonate (x_1, y_1) și (x_2, y_2) mijlocul segmentului are coordonatele $x = (x_1 + x_2)/2$, $y = (y_1 + y_2)/2$.

Dacă există centru de simetrie, atunci acesta este cu centrul în mijlocul segmentului cu capetele în primul punct și celălalt în ultimul punct (adică cu un capăt în cel mai din stânga punct și celălalt în cel mai din dreapta).

Notăm acest punct cu M . Pentru ca să existe centru de simetrie trebuie ca segmentele cu coordonatele (x_i, y_i) și (x_{n-i+1}, y_{n-i+1}) să aibă centrul în M , $i = 1, 2, \dots, n/2$.

Complexitate algoritm: $O(n \log n)$.

35.3.2 Cod sursă

Listing 35.3.1: a007.pas

```

1 program dpa;
2 type punct=record
3     x,y:integer;
4     end;
5     vector=array[1..10001]of punct;
6     list=array[1..10001]of integer;
7
8 var p:vector;
9     a:list;
10    n,k,i,n1:integer;
11    f,g:text;
12    n2,n3:real;
13    t:longint absolute 0:\$46c;
14    t0:longint;
15
16 procedure QuickSort(Lo, Hi: Integer);
17
18 procedure Sort(l, r: Integer);
19 var
20     i, j: integer;
21     xx,yy: punct;
22 begin
23     i := l; j := r; xx := p[(l+r) DIV 2];
24     repeat
25         while (p[i].x < xx.x)or((p[i].x==xx.x)and(p[i].y<xx.y)) do i := i + 1;
26         while (xx.x < p[j].x)or((xx.x=p[j].x)and(xx.y<p[j].y)) do j := j - 1;
27         if i <= j then
28             begin
29                 yy := p[i]; p[i] := p[j]; p[j] := yy;
30                 i := i + 1; j := j - 1;
31             end;
32         until i > j;
33         if l < j then Sort(l, j);
34         if i < r then Sort(i, r);
35     end;
36
37 begin {QuickSort};
38     Sort(Lo,Hi);
39 end;
40
41
42
43
44 procedure solutie(var n1:integer;var n2,n3:real);
45 var M,T:punct;nn2,nn3:real;
46 begin
47     if n mod 2 =1 then
48         begin
49             n1:=0;
50             n2:=0;
51             n3:=0;
52             exit;
53         end
54         else
55         begin
56             n1:=1;
57             n2:=(p[1].x+p[n].x)/ 2;
58             n3:=(p[1].y+p[n].y)/ 2;
59             for i:=2 to n div 2 do
60                 begin
61                     nn2:=(p[i].x+p[n-i+1].x)/ 2;
62                     nn3:=(p[i].y+p[n-i+1].y)/ 2;
63                     if (abs(n2-nn2)>0.0001)or(abs(n3-nn3)>0.0001) then
64                         begin
65                             n1:=0;
66                             exit;
67                         end;
68                     end;
69                 end;
70             end;

```

```

71
72 procedure rezolva;
73 var nr:integer;
74 begin
75 assign(f,'a007.in');
76 reset(f);
77 assign(g,'a007.out');
78 rewrite(g);
79 readln(f,k);
80 for nr:=1 to k do
81 begin
82   readln(f,n);
83   for i:=1 to n do
84     readln(f,p[i].x,p[i].y);
85   quicksort(1,n);
86   solutie(n1,n2,n3);
87   if n1=0 then
88     writeln(g,0)
89   else
90     writeln(g,1,' ',n2:0:4,' ',n3:0:4);
91 end;
92 close(f);
93 close(g);
94 end;
95
96 begin {p.p.}
97 t0:=t;
98 rezolva;
99 writeln(t-t0);
end.
```

35.3.3 *Rezolvare detaliată

35.4 Inter

În țara *Smar* sunt N autostrăzi, sub forma unor drepte în plan. Se știe că la intersecții de drumuri (care includ și autostrăzi) există un risc ridicat de accidente. De aceea polițiștii din această țară au hotărât stabilirea unei zone compacte care să includă toate intersecțiile și în care să se supravegheze atent circulația. Din motive financiare zona trebuie să fie de perimetru minim.

Cerință

Scrieți un program care să determine aria zonei de supraveghere alese.

Date de intrare

Din fișierul **inter.in** se va citi de pe prima linie numărul de autostrăzi, iar de pe fiecare dintre următoarele N linii câte patru numere reale, separate prin câte un spațiu, reprezentând coordonatele a două puncte distincte ce determină câte o dreaptă. Ele sunt date în ordinea $X_1\ Y_1\ X_2\ Y_2$, adică abscisa și ordonata punctului 1, apoi abscisa și ordonata punctului 2.

Date de ieșire

În fișierul **inter.out** se va scrie pe prima linie un singur număr real, cu două zecimale exacte (cu trunchiere), reprezentând aria zonei alese pentru supraveghere.

Restricții și precizări

Între oricare două autostrăzi există fix o intersecție.

Aria suprafeței de supraveghere este strict pozitivă pentru datele de test.

5 teste din 10 vor avea $N < 501$.

$2 < N < 5001$

Exemplu

inter.in	inter.out
4	3.00
0 0 1 0	
0 0 0 2	
0 2 1 0	
-2 0 0 1	

Timp maxim de execuție/test: 0.2 secunde pentru Linux, 1.5 secunde pentru Windows.

35.4.1 Indicații de rezolvare

Sunt $N * (N - 1)/2$ puncte de intersecție. Dar din acestea maxim $2 * N$ ne interesează (extremitățile de pe drepte), dar de fapt sunt maxim N . Pentru aflarea acestor puncte, care pot fi pe *înfășurătoarea convexă* a celor $N * (N - 1)/2$ puncte, se *sortează* dreptele după pantă.

Punctele de interes sunt intersecția a două drepte consecutive în sortare, și în plus prima cu ultima.

Pe aceste N puncte se face *înfășurătoarea convexă*.

Se calculează aria zonei rezultate ...

Complexitate :

- 1 - sortare după pantă - $O(N \log N)$
 - 2 - calcularea punctelor candidate de a fi pe *înfășurătoare* - $O(N)$
 - 3 - *înfășurătoarea convexă* - $O(N \log N)$
 - 4 - calcularea ariei - $O(N)$
- În total: $O(N \log N)$

35.4.2 Cod sursă

Listing 35.4.1: inter.pas

```

1 const dim=5000;
2 type tip=array[1..dim] of double;
3     pt=record x,y:double;end;
4     tip2=array[1..dim] of ^pt;
5 type punct=record
6     x,y:double;
7     end;
8     pozpseg=(stanga,dreapta,fata,spate,intre,orig,dest);
9     pozppol=(interior,exterior,frontiera);
10    intersectie=(secant,paralel,coliniar);
11 var poz,i,j,n:integer;
12 m,x1,x2,y1,y2:^tip;
13 f:text;
14 max:double;
15 o:array[1..dim] of integer;
16 p:tip2;
17 t:double;
18 transx,transy:double;
19
20 const eps:double=0.0001;
21
22
23 procedure Sort(l, r: Integer);
24 var
25   i, j, x, y: integer;
26 begin
27   i := l; j := r; x := l;
28   repeat
29     while p[o[i]]^ .y*p[o[x]]^ .x < p[o[i]]^ .x*p[o[x]]^ .y do i := i + 1;
30     while p[o[j]]^ .y*p[o[x]]^ .x > p[o[j]]^ .x*p[o[x]]^ .y do j := j - 1;
31     if i <= j then
32       begin
33         y:=o[i];o[i]:=o[j];o[j]:=y;
34         i := i + 1; j := j - 1;
35       end;
36     until i > j;
37     if l < j then Sort(l, j);
38     if i < r then Sort(i, r);
39   end;
40
41
42 var x:double;
43
44 procedure Sortm(l, r: Integer);
45 var
46   i, j, y: integer;

```

```

47 begin
48   i := 1; j := r; x := m^[o[1]];
49   repeat
50     while m^[o[i]] < x do i := i + 1;
51     while x < m^[o[j]] do j := j - 1;
52     if i <= j then
53       begin
54         y := o[i]; o[i] := o[j]; o[j] := y;
55         i := i + 1; j := j - 1;
56       end;
57     until i > j;
58     if l < j then Sortm(l, j);
59     if i < r then Sortm(i, r);
60   end;
61
62
63 {consideram segmentul orientat p1 -> p2 si punctul p}
64 function clasif_p_seg(p1,p2:punct):pozpsseg;
65 var a,b:punct;
66   s:double;
67 begin
68   a.x:=p2.x-p1.x;
69   a.y:=p2.y-p1.y;
70   b.x:=p.x-p1.x;
71   b.y:=p.y-p1.y;
72
73   s:=a.x*b.y-b.x*a.y;
74   if (s<0) then begin clasif_p_seg:=stanga;exit;end;
75   if (s>0) then begin clasif_p_seg:=dreapta;exit;end;
76   if (a.x*b.x<0) or (a.y*b.y<0) then
77     begin clasif_p_seg:=spate;exit;end;
78   if (a.x*a.x+a.y*a.y<b.x*b.x+b.y*b.y) then
79     begin clasif_p_seg:=fata;exit;end;
80   if ((p1.x-p.x)<eps) and ((p1.y-p.y)<eps) then
81     begin clasif_p_seg:=orig;exit;end;
82   if ((p2.x-p.x)<eps) and ((p2.y-p.y)<eps) then
83     begin clasif_p_seg:=dest;exit;end;
84   clasif_p_seg:=intre;
85 end;
86
87
88 {intersectia a doua drepte
89 dreapta 1 este m1 (definita de a si b), evident cine e a doua
90 t e parametru din a+t(b-a)
91 }
92
93 function dr_inters(m1a,m1b,m2c,m2d:punct;var t:double):intersectie;
94 var normal,da,db:punct;
95   numerator, numitor:double;
96   clasa_m1:pozpsseg;
97 begin
98   normal.x:=m2d.y-m2c.y;
99   normal.y:=m2c.x-m2d.x;
100  da.x:=m1a.x-m1b.x;
101  da.y:=m1a.y-m1b.y;
102  numerator:=normal.x*da.x+normal.y*da.y;{produs scalar - normal,da}
103  if numitor=0 then begin
104    clasa_m1:=clasif_p_seg(m1a,m2c,m2d);
105    if (clasa_m1=stanga) or (clasa_m1=dreapta)
106      then dr_inters:=paralel
107      else dr_inters:=coliniar;
108    exit;
109  end;
110  db.x:=m1a.x-m2c.x;
111  db.y:=m1a.y-m2c.y;
112  numerator:=normal.x*db.x+normal.y*db.y;
113  t:=numerator/numitor;
114  dr_inters:=secant;
115 end;
116
117 var p1,p2,p3,p4:punct;
118
119 procedure rezolva;
120 var st:array[1..dim+1] of integer;
121   k,il:integer;
122   s:double;

```

```

123 begin
124   transx:=p[1]^x;
125   transy:=p[1]^y;
126   poz:=1;
127   for i:=1 to n do begin
128     if transy>p[i]^y then if transx>p[i]^x then begin
129       transx:=p[i]^x;
130       transy:=p[i]^y;
131       poz:=i;
132       end;
133     if transy>p[i]^y then begin
134       transx:=p[i]^x;
135       transy:=p[i]^y;
136       poz:=i;
137       end;
138     end;
139   for i:=1 to n do begin
140     p[i]^x:=p[i]^x-transx;
141     p[i]^y:=p[i]^y-transy;
142     end;
143   for i:=1 to n do o[i]:=i;
144   o[poz]:=1;
145   o[1]:=poz;
146   sort(2,n);
147   st[1]:=o[1];
148   st[2]:=o[2];
149   k:=2;
150   for i1:=3 to n do begin
151     k:=k+1;
152     st[k]:=o[i1];
153   while (k<>2) and ((p[st[k-2]]^x-p[st[k-1]]^x)*(p[st[k-1]]^y-p[st[k]]^y)-
154     (p[st[k-1]]^x-p[st[k]]^x)*(p[st[k-2]]^y-p[st[k-1]]^y)<=eps)
155   do
156     begin
157       st[k-1]:=st[k];
158       k:=k-1;
159     end;
160   end;
161
162   st[k+1]:=st[1];
163   s:=0;
164   for i:=1 to k do
165   begin
166     s:=s+p[st[i]]^x*p[st[i+1]]^y-p[st[i+1]]^x*p[st[i]]^y;
167   end;
168   s:=s/2;
169
170   assign(f,'inter.out');rewrite(f);
171   writeln(f,s:0:2);
172   close(f);
173
174
175 end;
176
177 begin
178   new(x1);new(x2);new(y1);new(y2);
179   new(m);
180   for i:=1 to dim do new(p[i]);
181   assign(f,'inter.in');reset(f);
182   read(f,n);
183   for i:=1 to n do begin
184     read(f,x1^[i],y1^[i],x2^[i],y2^[i]);
185   end;
186   close(f);
187   max:=0;
188   for i:=1 to n do begin
189     if (x2^[i]-x1^[i]<>0) then begin
190       m^[i]:=(y2^[i]-y1^[i])/(x2^[i]-x1^[i]);
191     end else m^[i]:=0;
192     if max<m^[i] then max:=m^[i];
193   end;
194   for i:=1 to n do begin
195     if (x2^[i]-x1^[i]=0) then m^[i]:=max+1;
196   end;
197
198

```

```

199  for i:=1 to n do o[i]:=i;
200  Sortm(1,n);
201
202  for i:=1 to n-1 do if m^o[i]=m^o[i+1] then begin
203    writeln('Pante egale!!!!');
204    exit;
205  end;
206
207  for i:=1 to n-1 do begin
208    p1.x:=x1^o[i];
209    p1.y:=y1^o[i];
210    p2.x:=x2^o[i];
211    p2.y:=y2^o[i];
212    p3.x:=x1^o[i+1];
213    p3.y:=y1^o[i+1];
214    p4.x:=x2^o[i+1];
215    p4.y:=y2^o[i+1];
216    dr_inters(p1,p2,p3,p4,t);
217    p[i]^x:=p1.x+t*(p2.x-p1.x);
218    p[i]^y:=p1.y+t*(p2.y-p1.y);
219  end;
220
221  p1.x:=x1^o[1];
222  p1.y:=y1^o[1];
223  p2.x:=x2^o[1];
224  p2.y:=y2^o[1];
225  p3.x:=x1^o[n];
226  p3.y:=y1^o[n];
227  p4.x:=x2^o[n];
228  p4.y:=y2^o[n];
229  dr_inters(p1,p2,p3,p4,t);
230  p[n]^x:=p1.x+t*(p2.x-p1.x);
231  p[n]^y:=p1.y+t*(p2.y-p1.y);
232
233  rezolva;
234
235 end.

```

35.4.3 *Rezolvare detaliată

35.5 Nr

Fie x un număr natural cu exact n cifre scris în baza 10.

Cerință

Scrieți un program care să determine cel mai mic număr natural strict mai mare decât x , care are aceleași cifre ca și numărul x și care este palindrom.

Date de intrare

Fișierul de intrare **nr.in** conține două linii. Pe prima linie este scris n , numărul de cifre ale numărului x . Pe cea de a doua linie sunt scrise cele n cifre ale lui x .

Date de ieșire

Fișierul de ieșire **nr.out** conține o singură linie pe care se află cel mai mic număr natural strict mai mare decât x , care are aceleași cifre ca și numărul x și care este palindrom. Dacă nu există soluție pe prima linie a fișierului de ieșire va fi scrisă valoarea 0.

Restricții

$$2 \leq n \leq 1000$$

Numim palindrom un număr care citit de la stânga la dreapta, cât și de la dreapta la stânga este același (de exemplu 1331, 12321, etc).

Prima cifra a unui număr trebuie să fie nenulă.

Prin aceleași cifre se înțelege că fiecare cifră de la 0 la 9 apare în rezultat de același număr de ori ca și în numărul x .

Exemple

nr.in	nr.out	nr.in	nr.out
5	0	5	20102
12022		12200	

Timp maxim de execuție: 0.1 secunde/test (atât sub Windows cât și sub Linux).

35.5.1 Indicații de rezolvare

1. Determinam numărul de apariții pentru fiecare cifră (0..9) în numărul dat (x). Acest număr trebuie să fie par (cu excepția eventuală a unei singure cifre; în caz că o astfel de cifră există trebuie să fie plasată la mijlocul palindromului), altfel nu există soluție.

2. Determinăm numărul de apariții ale fiecărei cifre (0..9) în prima jumătate a numărului x , precum și numărul de apariții necesar (pentru nr cerut). Dacă pentru o cifră numărul de apariții existent în prima jumătate a numărului dat este prea mare, elimin aparițiile suplimentare, începând de la dreapta spre stânga.

3. Încerc să distribui cifrele care mai sunt necesare în prima jumătate a numărului astfel încât să obțin cel mai mic număr posibil $\geq x$ (cu o eventuală rearanjare a cifrelor existente). Distribuirea cifrelor de face căutând de la stânga la dreapta să completăm pozițiile cu cifre egale cu cele din numărul dat. Când nu mai este posibil:

- am obținut o jumătate mai mică sau egală cu cea din numărul dat (cea mai mare cu această proprietate, determin anagrama imediat următoare din pdv lexicografic lui x - dacă există;

- am găsit o cea mai mică cifră mai mare decât cea de pe poziția curentă, o plasez, ordonez crescător cifrele de după \Rightarrow cea mai mică jumătate $> x$.

35.5.2 Cod sursă

Listing 35.5.1: Nr.cpp

```

1 #include <iostream>
2 #include <iostream>
3 #include <cstring>
4
5 using namespace std;
6
7 #define InFile "nr.in"
8 #define OutFile "nr.out"
9 #define NMax 1001
10 #define NAN 100
11
12 char x[NMax], xi[NMax];
13 char sol[NMax];
14 int m[10], uz[10], n, L, Lg, cifra;
15 ofstream fout(OutFile);
16
17 void citire(void);
18 int ok (void);
19 void afiseaza(int );
20 int distribuie();
21 int succesor();
22 int total (void);
23
24 int main()
25 { citire();
26   cifra=ok();
27   if (cifra==NAN) {afiseaza(0); return 0;}
28   int gata=distribuie();
29   if (gata==1) {afiseaza(1); return 0;}
30   if (gata==NAN) {afiseaza(0); return 0;}
31   if (succesor()) afiseaza(1);
32   else afiseaza(0);
33   return 0;
34 }
35
36 void citire(void)
37 { ifstream fin(InFile);
38   fin>>n; fin.get(); fin.getline(x,NMax);
39   fin.close();

```

```

40     strcpy(xi,x);
41     for (int i=0; i<n; i++)
42         {if (!(x[i]>='0'&&x[i]<='9'))
43          {cout<<"fisier de intrare eronat!";return;}
44          uz[x[i]-'0']++;
45     L=n/2-1;
46   }
47
48 int ok(void)
49 { int i, nrímpar=0, cine=10;
50   for (i=0; i<10; i++)
51     {if (uz[i]%2) {nrímpar++; cine=i;}
52      m[i]=uz[i]/2; // m[i]=numarul de aparitii necesare
53      // ale cifrei i intr-o jumataate de palindrom
54      uz[i]=0;}
55   for (i=0; i<=L+n%2;i++) uz[x[i]-'0']++;
56   //uz[i]=numarul de aparitii existente in prima jumataate ale cifrei i
57   if (cine<10) m[cine]++;
58   if (nrímpar<2) return cine; //cifra din mijloc
59   return NAN; //nu exista solutie
60 }
61
62 int succesor()
63 { //determina anagrama urmatoare>x, daca este posibil
64   int i, j, aux;
65   char fostx=x[L+1], fosta=sol[L+1];
66   x[L+1]=sol[L+1]=NULL;
67   if (strcmp(x,sol)==0 && fostx<fosta) {sol[L+1]=fosta;return 1;}
68   while (strcmp(x,sol)>=0)
69     { if (strcmp(x,sol)==0)
70       if (total()) {sol[L+1]=cifra+'0'; sol[L+2]=NULL;return 1;}
71       for (i=L-1; i>=0 && sol[i]>=sol[i+1]; i--)
72         if (i<0)
73           {if (strcmp(x,sol)>0 || strcmp(x,sol)==0 && fostx>fosta)
74             return 0; //nu exista succesor
75             return 1;}
76         for (j=L; sol[j]<=sol[i]; j--);
77         aux=sol[j]; sol[j]=sol[i]; sol[i]=aux;
78         for (j=L, i++; j>i; j--, i++)
79           {aux=sol[i]; sol[i]=sol[j]; sol[j]=aux;}
80       }
81   sol[L+1]=fosta;
82   return 1;
83 }
84
85 void afiseaza(int k)
86 {int i;
87 if (k)
88   {fout<<sol;
89    for (i=L; i>=0; i--) fout<<sol[i];}
90   else fout<<0;
91 fout<<endl;
92 fout.close();}
93
94 int distribuie ()
95 {int i, j, aux=10, min, poz, k, sch;
96   int pus[10];
97   if (cifra<10) //completez separat mijlocul, in caz de lg impara
98     {sol[L+1]=cifra+'0'; m[cifra]--; aux=x[L+1]-'0';}
99   //pus[i]= numarul de cifre i pe care trebuie sa le mai pun
100  for (i=0; i<10; i++)
101    if (m[i]>uz[i]) pus[i]=m[i]-uz[i];
102    else pus[i]=0;
103  if (aux<10) pus[aux]++;
104  for (i=L; i>=0; i--)
105    {sol[i]=x[i];
106     if (m[x[i]-'0']<uz[x[i]-'0'])//sterge o aparitie a cifrei i
107     {uz[x[i]-'0']--;
108      sol[i]=NAN;}
109    }
110  if (sol[0]==NAN)
111    {for (i=x[0]-'0'; i<10 && !pus[i]; i++);
112     if (i==10) return NAN;
113     sol[0]=i+'0'; pus[i]--;
114     j=1;
115     for (i=0; i<10;)

```

```

116     if (pus[i])
117         {if (sol[j]==NAN) {sol[j]=i+'0'; pus[i]--;}
118         j++;}
119     else i++;
120 if (!strcmp(x,sol)) if (total()) return 1;
121 i=0;
122 while (1)
123     { for (; i<=L && x[i]==sol[i];i++);
124     if (i>L) return 0;
125 //caut cea mai mica valoare >=de dupa
126 min='9'+1;
127     for (j=i; j<=L; j++)
128         if (sol[j]>=x[i] && min>sol[j]) {min=sol[j]; poz=j;}
129     if (min!='9'+1) //am gasit
130         {sol[poz]=sol[i]; sol[i]=min;
131         if (min>x[i])
132             {//ordonez crescator restul si gata
133             for (k=0; k<10; k++) pus[k]=0;
134             for (k=i+1; k<=L; k++) pus[sol[k]-'0']++;
135             j=i+1;
136             for (i=0; i<10;)
137                 if (pus[i]) {sol[j]=i+'0'; pus[i]--; j++;}
138                 else i++;
139             return 1;
140         }
141     }
142     else
143     {
144 //ordonez descrescator restul
145     for (k=0; k<10; k++) pus[k]=0;
146     for (k=i; k<=L; k++) pus[sol[k]-'0']++;
147     j=i;
148     for (i=9; i>=0;)
149         if (pus[i]) {sol[j]=i+'0'; pus[i]--; j++;}
150         else i--;
151     return 0;
152     }
153 }
154 }
155
156 int total (void)
157 {
158 char aux[NMax];
159 int i, j;
160 strcpy(aux,sol);
161 j=strlen(aux);
162 if (cifra>=0&&cifra<=9)
163     aux[j++]=cifra+'0';
164 for (i=strlen(sol)-1; i>=0; i--) aux[j++]=sol[i];
165 aux[j]=NULL;
166 if (strcmp(xi,aux)<0) return 1;
167 return 0;
168 }
```

35.5.3 *Rezolvare detaliată

35.6 Proc

O aplicație ce trebuie executată pe un calculator multi-procesor constă din N fragmente de cod independente, ce pot fi rulate în paralel. Fiecare fragment trebuie executat în totalitate pe un singur procesor. Din dorința de a paraleliza cât mai mult aplicația, fragmentele de cod au dimensiuni mici și, în consecință, tempi de execuție mici. Mai precis, execuția fiecărui fragment durează UNUL sau DOUĂ cicluri de ceas pe un procesor de tipul Pentium IV.

Sistemul pe care urmează să fie executată aplicația constă din P procesoare. Spre deosebire de majoritatea sistemelor de acest fel, însă, cele P procesoare au viteze de execuție diferite. Primul procesor este un Pentium IV și este cel mai rapid. Al doilea procesor este de două ori mai lent decât primul, al treilea de trei ori mai lent ... al i -lea procesor este de i ori mai încet decât primul. În aceste condiții, timpul de execuție al fiecărui fragment de cod diferă, în funcție de procesorul

pe care va fi executat. Să presupunem că un segment de cod are timpul de execuție T (unde T este 1 sau 2) pe primul procesor. Atunci pe un procesor i , timpul său de execuție va fi $i * T$.

Cerință

Știind că fragmentele de cod pot fi executate în orice ordine și pe orice procesor și că orice procesor poate executa, la un moment dat, un singur fragment de cod, determinați timpul minim după care se va termina execuția aplicației (adică a tuturor fragmentelor de cod). Timpul după care se termină aplicația este egal cu maximul dintre timpii după care fiecare procesor redevine disponibil. Timpul după care un procesor redevine disponibil este egal cu suma timpilor de execuție a fragmentelor de cod rulate pe procesorul respectiv.

Date de intrare

Prima (și singura) linie a fișierului de intrare **proc.in** conține trei numere întregi, separate prin spații: N - numărul de fragmente de cod, K - numărul de fragmente de cod care au timpul de execuție pe un Pentium IV egal cu 1 (implicit, $N-K$ au timpul de execuție egal cu 2) și P - numărul de procesoare ale sistemului.

Date de ieșire

Fișierul **proc.out** va conține o singură linie pe care se află timpul minim după care se termină de executat aplicația.

Restricții și precizări

$$0 \leq K \leq N \leq 1000000000$$

$$1 \leq P \leq 65535$$

Cel puțin 40% din teste vor avea $N \leq 2000$ și $P \leq 2000$.

Cel puțin 70% din teste vor avea $N \leq 65535$ și $P \leq 16383$.

Exemplu

proc.in	proc.out
4 3 2	4

Explicație

Pe primul procesor se execută un fragment de cod cu timpul de execuție (calculat pe un Pentium IV) egal cu 1 și un fragment de cod cu timpul de execuție egal cu 2 \Rightarrow timpul după care acest procesor devine disponibil este $1 * 1 + 1 * 2 = 3$. Pe al doilea procesor se execută două fragmente de cod cu timpul de execuție (calculat pe un Pentium IV) egal cu 1 \Rightarrow timpul după care acest procesor devine disponibil este $2 [\text{numărul de fragmente}] * (2 * 1) [\text{timpul de execuție al fiecarui fragment pe procesorul 2}] = 4$.

Timp maxim de execuție: 0.2 secunde/test pentru Linux și 1.5 secunde/test pentru Windows.

35.6.1 Indicații de rezolvare

Testele au fost structurate astfel încât diverse abordări ale problemei să obțină un număr diferit de puncte, astfel:

- 20% din punctaj, pentru o soluție cu complexitate exponențială
- 40% din punctaj pentru o soluție greedy cu complexitatea $O(N * P)$
- 70% din punctaj pentru o soluție greedy cu complexitatea $O(N * \log P)$
- 100% din punctaj pentru o soluție cu complexitatea $O(P * \log N)$

Soluția exponențială

Există mai multe posibilități de a implementa o soluție care foloseste *backtracking*. Se poate varia pentru fiecare fragment de cod procesorul pe care va fi executat - $O(NP)$ - sau se poate stabili pentru fiecare procesor câte fragmente de cod de tipul 1, respectiv 2 vor fi executate pe el. O optimizare a celei de-a doua variante ar fi că timpul după care un procesor i devine disponibil, notat prin T_i , trebuie să respecte relația $T_i/i \leq T_j/j$, unde j este numărul unui procesor mai rapid decât o (adică $j < i$).

Soluția greedy cu complexitate $O(N * P)$

Se inițializează timpii după care fiecare procesor redevine disponibil cu 0. Se stabiliește procesorul pe care va fi executat fiecare fragment de cod cu timpul 2. Apoi se stabilește același lucru (adică procesorul pe care va fi executat) pentru fiecare fragment de cod cu timpul 1. Pentru a

stabilii procesorul pe care va fi executat un fragment de cod, se încearcă executarea sa pe orice procesor de la 1 la P și se alege acel procesor pentru care se găsește minimul dintre timpii după care procesoarele redevin disponibile.

Altfel spus, să presupunem că până la adăugarea fragmentului de cod curent, timpul de revenire al fiecarui procesor i este T_i și să presupunem că fragmentul de cod actual are timpul de execuție Q (Q este 1 sau 2). Atunci se alege procesorul i , pentru care $T_i + Q * i$ este minim.

Soluția greedy cu complexitate $O(N * \log P)$

Ideea acestei soluții este la fel ca cea descrisă mai sus, numai că determinarea procesorului pentru care se obține timpul de revenire minim se face logaritmic, folosind o structură de *heap*.

Soluția (comisiei) cu complexitatea $O(P * \log N)$

Se observă că timpul minim după care se va termina aplicația este între 0 și $2 * N$ (în cazul când toate fragmentele de cod au timpul 2 și sunt executate pe primul procesor).

Se va folosi o *căutare binară* pentru determinarea acestui timp.

La un moment dat, în cadrul căutării, se va testa dacă timpul minim este mai mic sau egal decât timpul T . Dacă da, atunci se încearcă un T mai mic. Altfel, se mărește T -ul.

Funcția care decide dacă timpul minim este mai mic sau egal cu T are complexitatea $O(P)$.

Se observă că dacă timpul minim este cel mult T , atunci suma timpilor elementari pe fiecare procesor k trebuie să fie cel mult $[T/k]$. Se calculează

$$N_2 = \sum_{k=1}^P \left\lceil \frac{\left\lceil \frac{T}{k} \right\rceil}{2} \right\rceil$$

având semnificația: numărul maxim de fragmente de cod cu timpul 2 ce pot fi executate dacă timpul minim este cel mult T .

Dacă N_2 este mai mare sau egal cu numărul de fragmente de cod cu timpul 2 existente (să notăm acest număr cu *DOI*), se verifică dacă se pot executa destule fragmente de cod cu timpul 1.

Pentru aceasta se calculează

$$N_1 = \sum_{k=1}^P \left(\left\lceil \frac{T}{k} \right\rceil \mod 2 \right).$$

Dacă $N_1 + 2 * (N_2 - DOI)$ este mai mare sau egal cu numărul de fragmente de cod cu timpul 1, atunci timpul minim după care se termină aplicația este mai mic sau egal cu T . Altfel, nu.

35.6.2 Cod sursă

Listing 35.6.1: proc.pas

```

1 { Mugurel Ionut Andreica - Bucuresti, ROMANIA }
2 {
3   Time Complexity : O(P*logN)
4   Memory          : O(1)
5 }
6
7 Program _proc_OK;
8
9 const filein='proc.in';
10    fileout='proc.out';
11
12 var i,n,p,unu,doi,li,ls,mid,sol,sum,n1,n2:longint;
13
14 function ok(t:longint):boolean;
15 var i,s:longint;
16   n1,n2:extended;
17 begin
18 n1:=0; n2:=0;
19 for i:=1 to p do
20 begin
21   s:=t div i;
22   n2:=n2+(s div 2);
23   n1:=n1+(s mod 2);

```

```

24   end;
25
26 ok:=( (n2>=doi) and ((n1+2.0*(n2-doi))>=unu));
27 end;
28
29 begin
30 { Read input data }
31 assign(input,filein);
32 reset(input);
33 read(n,unu,p);
34 doi:=n-unu;
35 close(input);
36
37 { Search the minimum completion time binary }
38 li:=0; ls:=2*n;
39 sol:=ls;
40
41 while (li<=ls) do
42   begin
43     mid:=(li+ls) shr 1;
44
45     if (ok(mid)) then
46       begin
47         sol:=mid;
48         ls:=mid-1;
49       end
50     else
51       li:=mid+1;
52   end;
53
54 { Write output data }
55 assign(output,fileout);
56 rewrite(output);
57 writeln(sol);
58 close(output);
59 end.

```

Listing 35.6.2: PROCBKT1.pas

```

1 Program _PROC_BKT;
2
3 const MAXP=1000;
4   filein='proc.in';
5   fileout='proc.out';
6   infinit=1000000;
7
8 var nf,sol:array[0..MAXP,1..2] of longint;
9   tmax,tact,taux,i,j,k,m,n,p,unu,doi:longint;
10
11 procedure print;
12 begin
13   assign(output,fileout);
14   rewrite(output);
15   writeln(tmax);
16   close(output);
17   halt(0);
18 end;
19
20 procedure bkt(niv:integer);
21 var told:longint;
22   s1,s2:longint;
23 begin
24   if (niv=p+1) and (unu=0) and (doi=0) then
25     begin
26       tmax:=tact;
27       for i:=1 to p do
28         for j:=1 to 2 do
29           sol[i,j]:=nf[i,j];
30     end
31   else
32   if (niv<=p) then
33     begin
34       for s1:=0 to unu do
35         for s2:=0 to doi do
36           if (s1+2*s2<=(nf[niv-1,1]+2*nf[niv-1,2])) then

```

```

37      begin
38          told:=tact;
39          unu:=unu-s1;
40          doi:=doi-s2;
41
42          nf[niv,1]:=s1;
43          nf[niv,2]:=s2;
44
45          taux:=niv*(s1+2*s2);
46          if (taux>tact) then
47              tact:=taux;
48
49          if (tact<tmax) then
50              bkt(niv+1);
51
52          unu:=unu+s1;
53          doi:=doi+s2;
54          tact:=told;
55      end;
56  end;
57
58 begin
59 assign(input,filein);
60 reset(input);
61 read(n,k,p);
62 close(input);
63
64 tmax:=infinit;
65 tact:=0;
66 nf[0,1]:=infinit;
67 nf[0,2]:=infinit;
68
69 unu:=k; doi:=n-k;
70 bkt(1);
71
72 print;
73 end.

```

Listing 35.6.3: PROCGRE1.pas

```

1 Program _PROC_GRE;
2
3 const MAXP=3000;
4     filein='proc.in';
5     fileout='proc.out';
6     infinit=1000000000;
7
8 var nf,sol:array[0..MAXP,1..2] of longint;
9     t:array[1..MAXP] of longint;
10    tmax,tact,taux,i,j,k,m,n,p,unu,doi:longint;
11
12 procedure add(time:longint);
13 var i,index,mint:longint;
14 begin
15     mint:=infinit;
16
17     for i:=1 to p do
18         if (t[i]+i*time<mint) then
19             begin
20                 mint:=t[i]+i*time;
21                 index:=i;
22             end;
23
24     t[index]:=t[index]+index*time;
25     inc(sol[index,time]);
26 end;
27
28 procedure print;
29 begin
30     assign(output,fileout);
31     rewrite(output);
32     writeln(tmax);
33     close(output);
34     halt(0);

```

```

35 end;
36
37 begin
38 assign(input,filein);
39 reset(input);
40 read(n,k,p);
41 close(input);
42
43 fillchar(t,sizeof(t),0);
44
45 for m:=1 to n-k do
46   add(2);
47 for m:=1 to k do
48   add(1);
49
50 tmax:=0;
51 for i:=1 to p do
52   if (t[i]>tmax) then
53     tmax:=t[i];
54
55 print;
56 end.

```

Listing 35.6.4: procok.pas

```

1 { Mugurel Ionut Andreica - Bucuresti, ROMANIA }
2 {
3   Time Complexity : O(P*logN)
4   Memory          : O(1)
5 }
6
7 Program _proc_OK;
8
9 const filein='proc.in';
10    fileout='proc.out';
11
12 var i,n,p,unu,doi,li,ls,mid,sol,sum,n1,n2:longint;
13
14 function ok(t:longint):boolean;
15 var i,s:longint;
16   n1,n2:extended;
17 begin
18   n1:=0; n2:=0;
19   for i:=1 to p do
20     begin
21       s:=t div i;
22       n2:=n2+(s div 2);
23       n1:=n1+(s mod 2);
24     end;
25
26   ok:=( (n2>=doi) and ((n1+2.0*(n2-doi))>=unu));
27 end;
28
29 begin
30   { Read input data }
31   assign(input,filein);
32   reset(input);
33   read(n,unu,p);
34   doi:=n-unu;
35   close(input);
36
37   { Search the minimum completion time binary }
38   li:=0; ls:=2*n;
39   sol:=ls;
40
41   while (li<=ls) do
42     begin
43       mid:=(li+ls) shr 1;
44
45       if (ok(mid)) then
46         begin
47           sol:=mid;
48           ls:=mid-1;
49         end
50       else

```

```
51      li:=mid+1;
52  end;
53
54 { Write output data }
55 assign(output,fileout);
56 rewrite(output);
57 writeln(sol);
58 close(output);
59 end.
```

35.6.3 *Rezolvare detaliată

Capitolul 36

ONI 2002

36.1 Arbore

Să considerăm un arbore cu N vârfuri, numerotate de la 1 la N .

Cerință

Scriți un program care să adauge, dacă este posibil, un număr minim de muchii astfel încât fiecare vârf să aparțină exact unui singur ciclu.

Date de intrare

Fișierul de intrare ARBORE.IN conține:

ARBORE.IN	Semnificație
N	numărul de vârfuri din arbore
$x_1 \ y_1$	
$x_2 \ y_2$	
...	
$x_{N-1} \ y_{N-1}$	x_i și y_i sunt extremitățile muchiei i

Date de ieșire

Fișierul de ieșire ARBORE.OUT va conține pe prima linie valoarea -1 dacă problema nu admite soluție, respectiv numărul de muchii adăugate, dacă problema admite soluție. Dacă problema admite soluție, pe fiecare dintre următoarele linii se vor scrie extremitățile unei muchii adăugate, separate printr-un spațiu, sub forma:

ARBORE.OUT	Semnificație
Nr	numărul de muchii adăugate
$a_1 \ b_1$	
$a_2 \ b_2$	
...	
$a_{Nr} \ b_{Nr}$	a_i și b_i sunt extremitățile unei muchii adăugate

Restricții

$$3 \leq N \leq 100$$

x_i, y_i sunt numere întregi din intervalul $[1, N]$.

Exemple

ARBORE.IN	ARBORE.OUT	ARBORE.IN	ARBORE.OUT
4	-1	7	2
1 2		1 2	6 7
2 3		1 3	4 2
2 4		3 5	
		3 4	
		5 6	
		5 7	

Timp maxim de executare: 1 secundă/test

Observație. Punctajul pentru teste care nu admit soluție se va acorda dacă și numai dacă la un test care admite soluție s-a răspuns corect.

36.1.1 Indicații de rezolvare

Emanuela Cerchez

1. Vom selecta un vârf neterminat și vom considera acest vârf rădăcina arborelui.
2. Un vârf este considerat potențial "rezolvabil" dacă și numai dacă are ca subarbore numai vârfuri terminale sau "fire" (subarbore care este *lanț*).
3. Cât timp nu am rezolvat toate vârfurile (există vârfuri care nu aparțin unui ciclu) și nici nu am depistat o situație nerezolvabilă execut:
 - identific vârfurile potențial rezolvabile; fie x un astfel de vârf
 - analizez cazurile următoare (ceva mai rafinat decât explic)
 I. x are mai mult de 2 fi terminali \rightarrow nu există soluție
 II. x are exact 2 fi și aceștia sunt terminali \rightarrow unesc cei doi fi printr-o muchie; am obținut un ciclu cu 3 vârfuri care îl "rezolvă" pe x și cei doi fi ai săi;
 III. x are 2 fi terminali dar și alte "fire": unesc cei doi fi terminali (rezultă un ciclu de lungime 3 în care intră x și cei doi terminali), apoi rezolv firele (unesc printr-o muchie primul și ultimul vârf de pe "fir"; evident acest lucru este posibil dacă toate firele au lungimea mai mare decât 2);
 IV. x are 1 fiu terminal și evident alte "fire": unesc vârful terminal cu extremitatea finală a unuia dintre fire (cel de lungime 2 dacă există, oricare altul dacă nu există), apoi rezolv firele (acestea trebuie să aibă lungime mai mare decât 2)
 V. x nu are fiu terminali, numai fire; rezolv firele (dacă există mai mult de 2 fire de lungime 2 nu există soluție), unind între ele extremitățile finale a două fire (cele de lungime 2 dacă există, sau oricare altele), iar restul le rezolv independent (unind extremitatea lor initială cu extremitatea lor finală).

36.1.2 Cod sursă

Listing 36.1.1: ARB_MIA.pas

```

1 { Mugurel Ionut Andreica - Bucuresti, ROMANIA }
2
3 {$S-,I-,R-,Q-,N+}
4 {$M 15000,0,655360}
5 Program ARBORE;
6
7 const filein='ARBORE.IN';
8     fileout='ARBORE.OUT';
9     MAXN=100;
10    MARE=10000;
11
12 type plist^list;
13     list=record
14         fiu:longint;
15         urm:plist;
16     end;
17
18 var min:array[1..MAXN,0..1] of longint;
19 next:array[1..MAXN] of plist;
20 deg,tata,nfii:array[1..MAXN] of longint;
21 muc:array[1..MAXN,1..2] of longint;
22 i,j,k,m,n,r,sum,nodstart,f1,f2:longint;
23 aux,aa1,aa2:plist;
24
25 procedure maketree(nod:longint);
26 var aa:plist;
27 begin
28 aa:=next[nod];
29 while (aa<>nil) do
30 begin
31   if ((tata[aa^.fiu]=0) and (aa^.fiu<>tata[nod])) then
32     begin
33       inc(nfii[nod]);
34       tata[aa^.fiu]:=nod;
35       maketree(aa^.fiu);
36     end;
37   aa:=aa^.urm;
38 end;
39 end;

```

```

40
41 procedure df(nod:longint);
42 var aa:plist;
43 begin
44 sum:=sum+min[nod,0];
45
46 if (tata[nod]<>nodstart) and ((sum+1)<min[nodstart,1]) then
47 begin
48 min[nodstart,1]:=sum+1;
49 muc[nodstart,1]:=nodstart;
50 muc[nodstart,2]:=nod;
51 end;
52
53 aa:=next[nod];
54 while (aa<>nil) do
55 begin
56 if (aa^.fiu<>tata[nod]) then
57 begin
58 sum:=sum-min[aa^.fiu,1];
59 df(aa^.fiu);
60 sum:=sum+min[aa^.fiu,1];
61 end;
62
63 aa:=aa^.urm;
64 end;
65
66 sum:=sum-min[nod,0];
67 end;
68
69 procedure go2(nod:longint);
70 var aa:plist;
71 begin
72 sum:=sum+min[nod,0];
73
74 if ((sum+1)<min[nodstart,1]) then
75 begin
76 min[nodstart,1]:=sum+1;
77 muc[nodstart,1]:=f1;
78 muc[nodstart,2]:=nod;
79 end;
80
81 aa:=next[nod];
82 while (aa<>nil) do
83 begin
84 if (aa^.fiu<>tata[nod]) then
85 begin
86 sum:=sum-min[aa^.fiu,1];
87
88 go2(aa^.fiu);
89
90 sum:=sum+min[aa^.fiu,1];
91 end;
92
93 aa:=aa^.urm;
94 end;
95
96 sum:=sum-min[nod,0];
97 end;
98
99 procedure go1(nod:longint);
100 var aa:plist;
101 begin
102 sum:=sum+min[nod,0];
103
104 go2(f2);
105
106 aa:=next[nod];
107 while (aa<>nil) do
108 begin
109 if (aa^.fiu<>tata[nod]) then
110 begin
111 sum:=sum-min[aa^.fiu,1];
112 f1:=aa^.fiu;
113 go1(f1);
114
115

```

```

116         sum:=sum+min[aa^.fiu,1];
117     end;
118
119     aa:=aa^.urm;
120   end;
121
122 sum:=sum-min[nod,0];
123 end;
124
125 procedure compute(nod:longint);
126 var aa:plist;
127     summ:longint;
128 begin
129 if (deg[nod]=1) then
130   begin
131     min[nod,0]:=0;
132     min[nod,1]:=MARE;
133   end
134 else
135   begin
136     summ:=0;
137     aa:=next[nod];
138
139     while (aa<>nil) do
140       begin
141         if (aa^.fiu<>tata[nod]) then
142           begin
143             compute(aa^.fiu);
144             summ:=summ+min[aa^.fiu,1];
145           end;
146           aa:=aa^.urm;
147         end;
148
149     min[nod,0]:=summ;
150     min[nod,1]:=MARE;
151
152 { caz 1: se uneste nod cu unul din subarborele sau }
153 nodstart:=nod;
154 sum:=min[nod,0];
155
156 aa:=next[nod];
157 while (aa<>nil) do
158   begin
159     if (aa^.fiu<>tata[nod]) then
160       begin
161         sum:=sum-min[aa^.fiu,1];
162         df(aa^.fiu);
163         sum:=sum+min[aa^.fiu,1];
164       end;
165       aa:=aa^.urm;
166     end;
167
168 { caz 2: se unesc 2 noduri din subarborele lui nod }
169 nodstart:=nod;
170 sum:=min[nod,0];
171
172 aal:=next[nod];
173 while (aal<>nil) do
174   begin
175     if (aal^.fiu<>tata[nod]) then
176       begin
177         sum:=sum-min[aal^.fiu,1];
178
179         aa2:=aal^.urm;
180         while (aa2<>nil) do
181           begin
182             if (aa2^.fiu<>tata[nod]) then
183               begin
184                 sum:=sum-min[aa2^.fiu,1];
185                 f2:=aa2^.fiu;
186                 f1:=aal^.fiu;
187
188                 go1(f1);
189
190                 sum:=sum+min[aa2^.fiu,1];
191               end;

```

```

192           aa2:=aa2^.urm;
193       end;
194
195           sum:=sum+min[aa1^.fiu,1];
196       end;
197
198           aa1:=aa1^.urm;
199       end;
200   end;
201 end;
202
203 procedure build(nod:longint);
204 var aa:plist;
205     i,j,f1,f2,lf1,lf2:longint;
206 begin
207     i:=muc[nod,1];
208     j:=muc[nod,2];
209     writeln(i, ' ', j);
210
211     f2:=j; lf2:=0;
212     while (tata[f2]<>nod) do
213         begin
214             aa:=next[f2];
215             while (aa<>nil) do
216                 begin
217                     if (aa^.fiu<>tata[f2]) and (aa^.fiu<>lf2) then
218                         build(aa^.fiu);
219
220                     aa:=aa^.urm;
221                 end;
222
223             lf2:=f2;
224             f2:=tata[f2];
225         end;
226
227     aa:=next[f2];
228     while (aa<>nil) do
229         begin
230             if (aa^.fiu<>tata[f2]) and (aa^.fiu<>lf2) then
231                 build(aa^.fiu);
232
233             aa:=aa^.urm;
234         end;
235
236     if (i<>nod) then
237         begin
238             f1:=i; lf1:=0;
239             while (tata[f1]<>nod) do
240                 begin
241                     aa:=next[f1];
242                     while (aa<>nil) do
243                         begin
244                             if (aa^.fiu<>tata[f1]) and (aa^.fiu<>lf1) then
245                                 build(aa^.fiu);
246
247                             aa:=aa^.urm;
248                         end;
249
250                     lf1:=f1;
251                     f1:=tata[f1];
252                 end;
253
254             aa:=next[f1];
255             while (aa<>nil) do
256                 begin
257                     if (aa^.fiu<>tata[f1]) and (aa^.fiu<>lf1) then
258                         build(aa^.fiu);
259
260                     aa:=aa^.urm;
261                 end;
262             end;
263         end;
264     else
265         begin
266             f1:=i;
267         end;

```

```

268
269 aa:=next[nod];
270 while (aa<>nil) do
271   begin
272     if (aa^.fiu<>tata[nod]) and (aa^.fiu<>f1) and (aa^.fiu<>f2) then
273       build(aa^.fiu);
274
275     aa:=aa^.urm;
276   end;
277 end;
278
279 begin
280 assign(input,filein);
281 reset(input);
282 readln(n);
283
284 fillchar(deg,sizeof(deg),0);
285 fillchar(next,sizeof(next),0);
286 for k:=1 to n-1 do
287   begin
288     readln(i,j);
289     inc(deg[i]);
290     inc(deg[j]);
291
292     new(aux);
293     aux^.fiu:=i;
294     aux^.urm:=next[j];
295     next[j]:=aux;
296
297     new(aux);
298     aux^.fiu:=j;
299     aux^.urm:=next[i];
300     next[i]:=aux;
301   end;
302 close(input);
303
304 for i:=1 to n do
305   if (deg[i]>1) then
306     begin
307       r:=i;
308       break;
309     end;
310
311 fillchar(tata,sizeof(tata),0);
312 fillchar(nfii,sizeof(nfii),0);
313 maketree(r);
314
315 fillchar(muc,sizeof(muc),0);
316 compute(r);
317
318 assign(output,fileout);
319 rewrite(output);
320
321 if (min[r,1]>=MARE) then
322   writeln(-1)
323 else
324   begin
325     writeln(min[r,1]);
326     build(r);
327   end;
328 close(output);
329 end.

```

Listing 36.1.2: arbor.pas

```

1 program Arbore;
2 const NMaxVf = 100;
3   nfin='arbore.in';
4   nfout='arbore.out';
5 type Vf = -1 .. NMaxVf;
6   Muchie = record x,y: Vf; end;
7   Graf=array[Vf, Vf] of Vf;
8 var n, rad, nr, nrezolv, i, j, nrterminali, t1, t2,
9   lg, k, v, nrfire2, nrfire: Vf;
10  MC: array[Vf] of Muchie;

```

```

11      tata, d, sus1, sus2 : array[Vf] of Vf;
12      ok, t_unit: boolean;
13      G: Graf;
14
15  procedure DFS (x, tx: Vf);
16  var   i: Vf;
17  begin
18  if tata[x]=-1 then
19    begin
20      tata[x]:=tx;
21      for i:=1 to G[x][0] do
22        if (tx<>G[x][i]) then DFS(G[x,i],x)
23    end
24
25  end;
26
27  procedure Initializare;
28  var fin: text;
29  i, x, y: Vf;
30  begin
31  assign(fin, nfin); reset(fin);
32  readln(fin, n);
33  for i:= 1 to n-1 do
34    begin
35      readln(fin, x, y); inc(d[x]); inc(d[y]);
36      inc (G[x,0]); G[x][G[x,0]]:=y;
37      inc (G[y,0]); G[y][G[y,0]]:=x;
38    end;
39  close(fin);
40  rad:=1; while d[rad]=1 do inc(rad);
41  for i := 1 to n do tata[i]:=-1;
42  DFS (rad, 0);
43  for i := 1 to n do dec(d[i]); inc(d[rad]);
44  end;
45
46  procedure Afisare;
47  var fout: text;
48  i: Vf;
49  begin
50  assign(fout, nfout); rewrite(fout);
51  if not ok or (nrezolv<>n) then writeln(fout, -1)
52  else
53    begin
54      writeln(fout, nr);
55      for i:=1 to nr do writeln(fout, MC[i].x, ' ',MC[i].y);
56    end;
57  close(fout);
58  end;
59
60  procedure DeterminSus;
61  var  i: Vf;
62  begin
63  for i:=1 to n do begin sus1[i]:=-1; sus2[i]:=-1; end;
64  for i:=1 to n do
65    if d[i]=0 then {varf terminal}
66      begin
67        sus1[i]:=i; sus2[i]:=tata[i];
68        while (sus2[i]<>0) and (d[sus2[i]]=1) do
69          begin
70            j:=sus2[i]; sus1[j]:=j;
71            sus1[i]:=sus2[i]; sus2[i]:=tata[sus2[i]];
72            sus2[j]:=sus2[i];
73          end;
74      end;
75  end;
76
77  function Rezolvabil(x: Vf): boolean;
78  var  i: Vf;
79  begin
80  Rezolvabil:=false;
81  if (d[x]>1) or (d[x]=1) and (x=rad) then
82    begin
83      Rezolvabil:=true;
84      for i:=1 to G[x,0] do
85        if G[x,i]<>tata[x] then
86          if (d[G[x,i]]<>-1) and (sus2[G[x,i]]=-1) then

```

```

87 begin Rezolvabil:=false; break; end;
88 end;
89 end;
90
91 begin {program principal}
92 Initializare;
93 ok:=true; nr:=0; nrezolv:=0;
94 repeat
95 DeterminSus;
96 for i:= 1 to n do
97 if Rezolvabil(i) then
98 begin
99 nrterminali:=0; t1:=-1; t2:=-1;
100 for j:=1 to G[i,0] do
101 if G[i,j]<>tata[i] then
102 if( (d[G[i,j]]<-1) and (sus1[G[i,j]]=G[i,j]) ) d[G[i,j]]=0 then
103 begin
104 inc(nrterminali);
105 t1:=t2; t2:=G[i,j];
106 end;
107 if (nrterminali > 2) then ok:=false
108 else
109 if (nrterminali=2) then
110 begin
111 inc(nr); MC[nr].x:=t1; MC[nr].y:=t2; {unesc terminalii}
112 d[t1]:=-1; d[t2]:=-1; nrezolv:=nrezolv+3;
113 if tata[i]<>0 then dec(d[tata[i]]);
114 if d[i]>2 then {rezolv firele}
115 begin
116 for j:=1 to G[i,0] do
117 if G[i,j]<>tata[i] then
118 if d[G[i,j]]<-1 then
119 begin
120 inc(nr); MC[nr].x:=G[i,j];
121 lg:=1; k:=G[i,j];
122 while d[k]>0 do
123 for v:=1 to G[k,0] do
124 if G[k,v]<>tata[k] then
125 if d[G[k,v]]<-1 then
126 begin
127 d[k]:=-1;
128 k:=G[k,v];
129 inc(lg);
130 break;
131 end;
132 if lg=2 then ok:=false
133 else
134 begin MC[nr].y:=k; nrezolv:=nrezolv+lg; end;
135 end;
136 end;
137 d[i]:=-1;
138 end
139 else
140 if nrterminali = 1 then
141 begin
142 {rezolv firele}
143 t_unit:=false; d[i]:=-1; dec(d[tata[i]]);
144 for j:=1 to G[i,0] do
145 if G[i,j]<>tata[i] then
146 if d[G[i,j]]>0 then
147 begin
148 inc(nr); MC[nr].x:=G[i,j];
149 lg:=1; k:=G[i,j];
150 while d[k]>0 do
151 for v:=1 to G[k,0] do
152 if G[k,v]<>tata[k] then
153 if d[G[k,v]]<-1 then
154 begin
155 d[k]:=-1;
156 k:=G[k,v];
157 inc(lg);
158 break;
159 end;
160 if (lg=2) then
161 if t_unit then ok:=false
162 else

```

```

163           begin
164             MC[nr].x:=t2;
165             MC[nr].y:=k;
166             nrezolv:=nrezolv+lg+2;
167             t_unit:=true;
168           end
169         else
170           begin MC[nr].y:=k; nrezolv:=nrezolv+lg; end;
171         end;
172       if not t_unit then
173         begin MC[nr].x:=t2; inc(nrezolv, 2); end;
174       end
175     else {nu există terminali}
176     begin
177       {rezolv firele}
178       d[i]:=-1; dec(d[tata[i]]);
179       nrfire2:=0; t1:=0; t2:=0; nrfire:=0;
180       for j:=1 to G[i,0] do
181         if G[i,j]<>tata[i] then
182           if d[G[i,j]]>0 then
183             begin
184               lg:=1; k:=G[i,j];
185               while d[k]>0 do
186                 for v:=1 to G[k,0] do
187                   if d[G[k,v]]<>-1 then
188                     begin
189                       d[k]:=-1;
190                       k:=G[k,v];
191                       inc(lg);
192                       break;
193                     end;
194                   inc(nrfire);
195                 if (lg=2) then
196                   begin inc(nrfire2); t1:=t2; t2:=k; end
197                 else
198                   begin
199                     inc(nr); MC[nr].x:=G[i,j];
200                     MC[nr].y:=k; nrezolv:=nrezolv+lg;
201                   end;
202                 end;
203               if nrfire2>2 then ok:=false
204             else
205               if nrfire2=2 then
206                 begin
207                   inc(nr);
208                   MC[nr].x:=t1;
209                   MC[nr].y:=t2;
210                   nrezolv:=nrezolv+5;
211                 end
212               else
213                 if nrfire2=1 then
214                   if nrfire=1 then
215                     begin
216                       inc(nr);
217                       MC[nr].x:=t2;
218                       MC[nr].y:=i;
219                       inc(nrezolv, 3);
220                     end
221                   else
222                     begin MC[nr].x:=t2; inc(nrezolv, 3); end
223                   else
224                     if nrfire>=2 then
225                       begin
226                         MC[nr-1].x:=MC[nr].y;
227                         dec(nr);
228                         inc(nrezolv);
229                       end;
230                     end;
231                   end;
232     until not ok or (nrezolv>=n-2);
233   Afisare;
234 end.

```

36.1.3 *Rezolvare detaliată

36.2 Decod

Serviciul Român de Informații (SRI) a dat de urma unei binecunoscute și periculoase organizații teroiste, care își are sediul pe teritoriul țării noastre. Folosind cei mai pricepuți și mai bine antrenați spioni și ofițeri, SRI a reușit să identifice computerul principal al organizației teroiste. Dacă va reuși să acceseze informațiile din acest computer, SRI îi va putea aresta pe toți membrii organizației și va asigura (în continuare) pacea mondială. Singura problemă este spargerea codului de acces. Tot ceea ce se știe despre acest cod este că el este reprezentat de către o permutare de lungime N . Specialiștii SRI au încercat diverse metode de a descoperi codul, însă tot ceea ce au reușit să obțină este un program care, transmitându-i-se ca parametru o permutare de lungime N , specifică în câte pozitii coincid această permutare și codul de acces. Din păcate, șefii nu cred în utilitatea acestui program.

Cerință

Scriți un program care, folosind programul-ajutător (reprezentat sub forma unui modul), să determine codul de acces în computerul teroștilor.

Date de intrare

Programul dumneavoastră nu va citi date din vreun fișier de intrare. El va apela întâi funcția **GetN** a modulului **PROG**, care îi va returna valoarea N - numărul de elemente ale permutării ce trebuie descoperită.

Apoi va apela numai funcția **Check**, căreia îi va fi transmisă ca parametru, de fiecare dată, o permutare de lungime N . Această funcție va returna numărul de pozitii în care permutarea transmisă ca parametru coincide cu permutarea ce trebuie descoperită. Programul dumneavoastră trebuie ca, după un număr finit de apelări ale funcției **Check**, să descopere permutarea căutată.

Date de ieșire

Programul dumneavoastră va trebui să tipărească în fișierul **DECOD.OUT** o permutare de lungime N . Toate cele N elemente vor fi tipărite pe prima linie a fișierului, fiind separate de către un spațiu.

Restricții:

$$5 \leq N \leq 256$$

Instrucțiuni pentru programatorii în C/C++

Veți avea la dispoziție header-ul **prog.h**, pe care va trebui să-l includeți în sursa dumneavoastră. Acest fișier declară următoarele funcții:

```
int GetN ( void )
int Check ( int p[256] )
```

Funcția **Check** are ca parametru un vector cu elemente întregi, care reprezintă o permutare. Ea va evalua această permutare și va returna una dintre următoarele valori:

-1 dacă primele N elemente întregi (începând cu poziția 0) ale vectorului transmis ca parametru nu constituie o permutare de lungime N

0-N numărul de pozitii în care coincide permutarea de lungime N transmisă ca parametru, cu permutarea ce trebuie descoperită

Instrucțiuni pentru programatorii în PASCAL

Modulul **PROG** este implementat sub forma unit-ului **PROG**. În acest unit sunt definite următoarele tipuri și funcții, care vor fi folosite de către programul dumneavoastră:

tipuri:

```
perm = array [ 1..256 ] of integer;
```

funcții:

```
function GetN : integer;
```

```
function Check ( var permut : perm ) : integer;
```

Funcția **Check** are ca parametru un vector de tipul **perm**. Ea va evalua această permutare și va returna una din următoarele valori:

-1 dacă primele N elemente întregi ale vectorului transmis ca parametru nu constituie o permutare de lungime N

$0-N$ numărul de poziții în care coincide permutarea de lungime N transmisă ca parametru, cu permutarea ce trebuie descoperită

Exemplu de apel

/* C/C++ */	{ Pascal }
int p[256];	var p: perm;
...	...
tmp = Check(p);	tmp := Check(p);

Pentru a să testa programul, creați un fișier numit DECOD.IN. Pe prima linie a acestui fișier scrieți valoarea N . Pe a doua linie scrieți o permutare de lungime N , având elementele separate prin spațiu. Modulul PROG va citi această permutare din fișierul DECOD.IN (din directorul curent) și o va considera permutarea ce trebuie descoperit?.

Exemplu

DECOD.IN
5
2 1 3 4 5

O serie posibilă de apeluri ale funcțiilor **GetN** și **Check** este următoarea:

- Se apelează funcția GetN și se returnează valoarea 5
- Se apelează funcția Check cu permutarea (1 2 3 4 5) și se returnează valoarea 3
- Se apelează funcția Check cu permutarea (3 2 1 4 5) și se returnează valoarea 2
- Se apelează funcția Check cu permutarea (2 1 3 4 5) și se returnează valoarea 5
- Se tipărește în fișierul DECOD.OUT permutarea 2 1 3 4 5.

Fișierul DECOD.OUT va arăta astfel:

2 1 3 4 5 - pe prima linie a fișierului

Timp maxim de executare: 1 secundă/test

Notă: Pe calculatorul dumneavoastră va fi instalat o versiune a modulului PROG în directorul c:\decod\, pe care o puteți folosi pentru testare. Înainte de evaluare, programul dumneavoastră va fi compilat folosind o altă versiune a modulului PROG. Această versiune nu va necesita un timp mai mare pentru executarea funcțiilor puse la dispoziție decât varianta de pe calculatoarele dumneavoastră.

36.2.1 *Indicații de rezolvare

36.2.2 Cod sursă

Listing 36.2.1: decod.pas

```

1 { Mugurel Ionut Andreica - Bucuresti, ROMANIA }
2
3 Program DECOD;
4
5 uses prog;
6
7 const fileout='decod.out';
8
9 var fout:text;
10    myperm,permini:perm;
11    ok:array[1..256] of boolean;
12    i,j,k,m,n,np,npini,np2,tip,poz,last:integer;
13    found:boolean;
14
15 begin
16   n:=GetN;
17   for i:=1 to n do

```

```

19 myperm[i]:=i;
20
21 fillchar(ok,sizeof(ok),false);
22
23 npini:=Check(myperm);
24 tip:=0;
25
26 for i:=2 to n do
27 begin
28   m:=myperm[1];
29   myperm[1]:=myperm[i];
30   myperm[i]:=m;
31
32   np:=Check(myperm);
33
34   if (np=npini+2) then
35     begin
36       tip:=1;
37       ok[1]:=true;
38       ok[i]:=true;
39       npini:=npini+2;
40
41       break;
42     end
43   else
44     if (np=npini-2) then
45       begin
46         tip:=1;
47         m:=myperm[1];
48         myperm[1]:=myperm[i];
49         myperm[i]:=m;
50
51         ok[1]:=true;
52         ok[i]:=true;
53         break;
54       end
55     else
56       if (np=npini-1) or (np=npini+1) then
57         begin
58           tip:=2;
59           poz:=i;
60
61           if (np=npini-1) then
62             begin
63               m:=myperm[1];
64               myperm[1]:=myperm[i];
65               myperm[i]:=m;
66             end;
67             break;
68           end
69         else
70           begin
71               m:=myperm[1];
72               myperm[1]:=myperm[i];
73               myperm[i]:=m;
74             end;
75         end;
76
77 if (tip=1) then
78 begin
79   npini:=Check(myperm);
80
81   for i:=2 to n-1 do
82     if (not ok[i]) then
83       begin
84         if (npini=n) then
85           break;
86
87         found:=false;
88
89         for j:=i+1 to n do
90           begin
91             m:=myperm[i];
92             myperm[i]:=myperm[j];
93             myperm[j]:=m;
94
95         end;
96       end;
97     end;
98   end;
99
100
```

```

95             np:=Check (myperm);
96
97             if (np=npini+2) then
98                 begin
99                     ok[i]:=true;
100                    ok[j]:=true;
101                    npini:=npini+2;
102
103                    found:=true;
104                    break;
105                end
106            else
107                if (np=npini-2) then
108                    begin
109                        ok[i]:=true;
110                        ok[j]:=true;
111
112                        m:=myperm[i];
113                        myperm[i]:=myperm[j];
114                        myperm[j]:=m;
115
116                        found:=true;
117                        break;
118                    end
119            else
120                if (np=npini-1) or (np=npini+1) then
121                    begin
122                        if (np=npini-1) then
123                            begin
124                                np2:=npini;
125                                m:=myperm[i];
126                                myperm[i]:=myperm[j];
127                                myperm[j]:=m;
128                            end
129                        else
130                            np2:=np;
131
132                            m:=myperm[i];
133                            myperm[i]:=myperm[1];
134                            myperm[1]:=m;
135
136                            np:=Check (myperm);
137
138                            if (np=np2-1) then
139                                begin
140                                    m:=myperm[i];
141                                    myperm[i]:=myperm[1];
142                                    myperm[1]:=m;
143
144                                    npini:=np2;
145                                    ok[j]:=true;
146                                end
147                            else
148                                begin
149                                    m:=myperm[i];
150                                    myperm[i]:=myperm[1];
151                                    myperm[1]:=m;
152
153                                    ok[i]:=true;
154                                    found:=true;
155                                    npini:=np2;
156                                    break;
157                                end;
158                            end
159            else
160                begin
161                    m:=myperm[i];
162                    myperm[i]:=myperm[j];
163                    myperm[j]:=m;
164                end;
165            end;
166        end;
167    end
168 else
169 begin
170     move (myperm,permin, sizeof (perm));

```

```

171     ok[1]:=true;
172
173     npini:=Check (myperm) ;
174
175     for i:=2 to n-1 do
176         if (not ok[i]) then
177             begin
178                 if (npini=n) then
179                     break;
180
181             found:=false;
182
183             for j:=i+1 to n do
184                 begin
185                     m:=myperm[i];
186                     myperm[i]:=myperm[j];
187                     myperm[j]:=m;
188
189                     np:=Check (myperm) ;
190
191                     if (np=npini+2) then
192                         begin
193                             ok[i]:=true;
194                             ok[j]:=true;
195                             npini:=npini+2;
196
197                             found:=true;
198                             break;
199                         end
200                     else
201                     if (np=npini-2) then
202                         begin
203                             ok[i]:=true;
204                             ok[j]:=true;
205
206                             m:=myperm[i];
207                             myperm[i]:=myperm[j];
208                             myperm[j]:=m;
209
210                             found:=true;
211                             break;
212                         end
213                     else
214                     if (np=npini-1) or (np=npini+1) then
215                         begin
216                             if (np=npini-1) then
217                                 begin
218                                     np2:=npini;
219                                     m:=myperm[i];
220                                     myperm[i]:=myperm[j];
221                                     myperm[j]:=m;
222
223                                 end
224                             else
225                                 np2:=np;
226
227                                 m:=myperm[i];
228                                 myperm[i]:=myperm[1];
229                                 myperm[1]:=m;
230
231                                 np:=Check (myperm) ;
232
233                                 if (np=np2-1) then
234                                     begin
235                                         m:=myperm[i];
236                                         myperm[i]:=myperm[1];
237                                         myperm[1]:=m;
238
239                                         npini:=np2;
240                                         ok[j]:=true;
241
242                                     end
243                                 begin
244                                     m:=myperm[i];
245                                     myperm[i]:=myperm[1];
246                                     myperm[1]:=m;

```

```

247
248          ok[i]:=true;
249          npini:=np2;
250          found:=true;
251          break;
252      end;
253    end
254  else
255    begin
256      m:=myperm[i];
257      myperm[i]:=myperm[j];
258      myperm[j]:=m;
259    end;
260  end;
261
262  if (not found) then
263    break;
264 end;
265
266 np:=Check (myperm);
267
268 if (np<>n) then
269  begin
270    move (permini,myperm,sizeof (perm));
271    fillchar(ok,sizeof (ok),false);
272    ok[poz]:=true;
273
274    npini:=Check (myperm);
275
276    for i:=1 to n do
277      if (not ok[i]) then
278        begin
279          if (npini=n) then
280            break;
281
282          last:=i-1;
283
284          for j:=last+1 to n do
285            if (j<>i) and (j<>poz) then
286              begin
287                m:=myperm[i];
288                myperm[i]:=myperm[j];
289                myperm[j]:=m;
290
291                np:=Check (myperm);
292
293                if (np=npini+2) then
294                  begin
295                    ok[i]:=true;
296                    ok[j]:=true;
297                    npini:=npini+2;
298                    break;
299                  end;
300                else
301                  if (np=npini-2) then
302                    begin
303                      ok[i]:=true;
304                      ok[j]:=true;
305
306                      m:=myperm[i];
307                      myperm[i]:=myperm[j];
308                      myperm[j]:=m;
309                      break;
310                    end;
311                  else
312                    if (np=npini-1) or (np=npini+1) then
313                      begin
314                        if (np=npini-1) then
315                          begin
316                            np2:=npini;
317                            m:=myperm[i];
318                            myperm[i]:=myperm[j];
319                            myperm[j]:=m;
320                          end;
321                      else
322                        np2:=np;

```

```

323
324         m:=myperm[i];
325         myperm[i]:=myperm[poz];
326         myperm[poz]:=m;
327
328         np:=Check (myperm);
329
330         if (np=np2-1) then
331             begin
332                 m:=myperm[i];
333                 myperm[i]:=myperm[poz];
334                 myperm[poz]:=m;
335
336                 ok[j]:=true;
337                 npini:=np2;
338             end
339         else
340             begin
341                 m:=myperm[i];
342                 myperm[i]:=myperm[poz];
343                 myperm[poz]:=m;
344
345                 ok[i]:=true;
346                 npini:=np2;
347                 break;
348             end;
349         end
350     else
351         begin
352             m:=myperm[i];
353             myperm[i]:=myperm[j];
354             myperm[j]:=m;
355         end;
356     end;
357   end;
358 end;
359 end;
360
361 assign(fout,fileout);
362 rewrite(fout);
363
364 for i:=1 to n do
365   write(fout,myperm[i],' ');
366 writeln(fout);
367
368 close(fout);
369 end.

```

36.2.3 *Rezolvare detaliată

36.3 Seti

Se pare că în sfârșit căutătorii vieții extraterestre au descoperit ceva! În cursul proiectului SETI@home a fost izolată o secvență care ar putea reprezenta un semnal de la alte forme de viață intelligentă. Ca urmare, proiectul SETI@ONI își propune să verifice dacă acel semnal provine într-adevăr de la extratereștri sau doar de la niște puști care beau Fanta.

Cerință

Pentru comoditate, porțiunea de semnal ce trebuie analizată vi se pune la dispoziție sub forma unei succesiuni de litere ale alfabetului latin. Vi se mai pune la dispoziție și un dicționar de cuvinte extraterestre, codificate în același mod. Scopul dumneavoastră este să numărați de câte ori apare fiecare dintre aceste cuvinte în posibilul mesaj extraterestru. Pornind de la aceste date, lingviștii pot să înceapă lucrul la traducerea mesajului.

Date de intrare

Pe prima linie a fișierului de intrare SETI.IN este scris numărul N de linii ale mesajului. Urmează N linii, fiecare conținând exact 64 de litere ale alfabetului latin urmate de marcajul de sfârșit de linie. Prin alipirea acestor bucăți se obține mesajul de analizat, format din $64 * N$ litere.

Pe prima linie a celui de-al doilea fișier de intrare DIC.IN este scris numărul M de cuvinte din dicționar. Urmează apoi M linii, fiecare conținând un cuvânt din dicționar, reprezentat ca o secvență de cel puțin una și cel mult 16 litere. Cuvintele nu sunt neapărat distincte.

Date de ieșire

Fișierul de ieșire SETI.OUT va conține exact M linii. Pe linia cu numărul i va fi scris numărul de apariții în mesajul extraterestru ale cuvântului cu numărul i din dicționar. Numărul de apariții nu va depăși niciodată 65535. Orice apariție a unui cuvânt trebuie numărată, chiar dacă se suprapune peste alte apariții. Se va face diferență între litere mari și litere mici.

Restricții

$0 \leq N < 2048$

$0 \leq M \leq 32000$

Exemplu

SETI.IN	
2	aaBaba
	babaaBaB
DIC.IN	

DIC.IN

2
3
b
bab
b

SETI.OUT

3
2
3

Timp maxim de executare: 1 secundă/test

36.3.1 Indicații de rezolvare

Mihai Pătrașcu

O metodă de a rezolva această problemă este să se ordoneze toate substringurile de lungime 16 din mesajul care trebuie analizat. Apoi, pentru fiecare cuvânt din dicționar se fac două căutări binare.

Sunt necesare însă următoarele două rafinări ale ideii:

- Sortarea tuturor substringurilor este consumatoare de timp. Dacă se folosesc metode precum *quicksort* sau *heapsort*, este probabil ca programul să nu treacă toate testele. Cea mai eficientă metodă de sortare în cazul de față, care se încadrează fără probleme în timp, este *radixsort*.

- Nu se poate ține un tabel cu substringurile după ce au fost sortate. În cel mai bun caz trebuie să reținem o referință căre fiecare substring (de exemplu, un pointer sau un indice), dar fiindcă pot fi peste 100,000 de substringuri, referința trebuie să fie de minim 24 de biți, ceea ce face tabloul de dimensiune inaceptabilă. Soluția constă în a împărti arbitrar mesajul în bucăți de 32,768 de caractere, și de a aplica algoritmul menționat pentru fiecare bucătă. Bineînțeles, trebuie să ne asigurăm că algoritmul ia în calcul și secvențele conținute în două bucăți alăturate.

36.3.2 *Rezolvare detaliată

36.3.3 Cod sursă

Listing 36.3.1: seti.c

```

1 /* IMPORTANT:
2  * special compiler instructions: set Assume SS=DS to Never
3  * (in Options > Compiler > Code generation)

```

```

4   */
5
6 /* seti.c -- ONI 2002 problem
7 * problem and official solution by Mihai Patrascu
8 */
9 #pragma option -3 -r -z -O2 -a -h- -N-
10 #include <stdio.h>
11 #include <string.h>
12 #include <stdlib.h>
13 #include <ctype.h>
14 #include <cmath>
15
16 using namespace std;
17
18 #define MAX_WORD      16
19
20 /* 512 input lines per pool line */
21 char huge datapool[4][32768u];
22
23 char _ds *huge sortpool[4][32768u];
24 char _ds *huge tmp_pool[32768u];
25
26 char buf[32768u];
27 char s[100];
28
29 #if 0
30 #define my_assert(cond)      ((cond) ? (void) 0 : (_DS = _SS, abort()))
31 #else
32 #define my_assert(cond)      ((void) 0)
33 #endif
34
35 #define achr    ((char _ds *) 0)
36 #define aptr    ((char _ds * _es *) 0)
37 #define atmp    ((char _ds * _ds *) 0)
38 #define abuf    ((char _ss *) buf)
39
40 void do_preproc(unsigned n, unsigned ds_achr, unsigned ds_atmp)
41 {
42     unsigned i, j, cnt[256];
43
44     for(i = 0; i < n; i++)
45         aptr[i] = achr + i;
46     for(i = MAX_WORD; i--; ) {
47         /* must prevent BC optimizations; now, ain't this ironical? */
48         *cnt = 0;
49         for(j = 1; j < 256; j++)
50             cnt[j] = cnt[j - 1];
51
52         _DS = ds_achr;
53         for(j = 0; j < n; j++)
54             cnt[abuf[j]] = aptr[j][i]++;
55         my_assert(_DS == ds_achr);
56
57         for(j = 0; j < 255; j++)
58             cnt[j + 1] += cnt[j];
59         for(j = 255; j; j--)
60             cnt[j] = cnt[j - 1];
61         *cnt = 0;
62
63         _DS = ds_atmp;
64         for(j = 0; j < n; j++)
65             atmp[j] = aptr[j];
66         for(j = 0; j < n; j++)
67             aptr[cnt[abuf[j]]++] = atmp[j];
68         my_assert(_DS == ds_atmp);
69     }
70 }
71
72 #undef achr
73 #undef aptr
74 #undef atmp
75 #undef abuf
76
77 void preproc(int line, unsigned n)
78 {
79     union {

```

```

80         void far *p;
81         unsigned b[2];
82     } ds1, ds2, newes;
83
84     ds1.p = datapool[line];
85     ds2.p = tmp_pool;
86     newes.p = sortpool[line];
87     my_assert(!newes.b[0] && !ds1.b[0] && !ds2.b[0]);
88     my_assert(_DS == _SS);
89
90     _ES = newes.b[1];
91     do_preproc(n, ds1.b[1], ds2.b[1]);
92     my_assert(_ES == newes.b[1]);
93     _DS = _SS;
94 }
95
96 #define achr    ((char _ds *) 0)
97 #define aptr    ((char _ds * _es *) 0)
98
99 #define X(i, x) \
100     if(s1[i] == s2[i]) { x } \
101     else return(s1[i] > s2[i]);
102
103 int ge(char _ss *s1, char _ds *s2)
104 {
105     X(0, X(1, X(2, X(3, X(4, X(5, X(6, X(7, X(8, X(9, X(10, X(11,
106         X(12, X(13, X(14, X(15, X(16,
107         my_assert(0);
108         return(0); /* avoid warning */
109         )))))))))))))
110 }
111
112 unsigned do_query(unsigned n)
113 {
114     unsigned i = 0, inc;
115
116     for(inc = 0x8000; inc; inc >= 1)
117         if((i + inc <= n) && ge((char _ss *) s, aptr[i + inc - 1]))
118             i += inc;
119     return(i);
120 }
121
122 #undef achr
123 #undef aptr
124 #undef s
125
126 unsigned query(int line, unsigned n)
127 {
128     union {
129         void far *p;
130         unsigned b[2];
131     } newds, newes;
132     unsigned res;
133
134     newds.p = datapool[line];
135     newes.p = sortpool[line];
136     my_assert(!newes.b[0] && !newds.b[0]);
137     my_assert(_DS == _SS);
138
139     _ES = newes.b[1];
140     _DS = newds.b[1];
141     res = do_query(n);
142     my_assert(_ES == newes.b[1] && _DS == newds.b[1]);
143     _DS = _SS;
144     return(res);
145 }
146
147 int main()
148 {
149     FILE *f;
150     int n, i, m;
151
152     f = fopen("seti.in", "r");
153     setvbuf(f, buf, _IOFBF, 0x4000);
154     fscanf(f, "%d ", &n);
155     for(i = 0; i < n; i++) {

```

```

156     fgets(s, sizeof(s), f);
157     _fmemcpy(datapool[i >> 9] + ((i & 511) << 6), s, 64);
158 }
159 fclose(f);
160
161 _fmemset(datapool[n >> 9] + ((n & 511) << 6), '.', 64);
162 for(i = 0; i < n; i += 512) {
163     if(i + 512 <= n)
164         preproc(i >> 9, 32768u);
165     else preproc(i >> 9, (n - i) << 6);
166 }
167
168 freopen("dic.in", "r", stdin);
169 freopen("seti.out", "w", stdout);
170 setvbuf(stdin, buf, _IOFBF, 0x4000);
171 setvbuf(stdout, buf + 0x4000, _IOFBF, 0x4000);
172 for(scanf("%d ", &m); m--; ) {
173     long res = 0;
174     int l;
175
176     gets(s);
177     l = strlen(s);
178
179     for(i = l; i <= MAX_WORD; s[i++] = 0) ;
180     for(i = 0; i < n; i += 512) {
181         if(i + 512 <= n)
182             res -= query(i >> 9, 32768u);
183         else res -= query(i >> 9, (n - i) << 6);
184     }
185
186     for(i = l; i <= MAX_WORD; s[i++] = 127) ;
187     for(i = 0; i < n; i += 512) {
188         if(i + 512 <= n)
189             res += query(i >> 9, 32768u);
190         else res += query(i >> 9, (n - i) << 6);
191     }
192     printf("%ld\n", res);
193 }
194 return(0);
195 }
```

36.4 Suma divizorilor

Se consideră două numere naturale A și B . Fie S suma tuturor divizorilor naturali ai lui A^B (A la puterea B).

Cerintă

Sa se afișeze restul împărțirii lui S la 9901.

Date de intrare

Pe prima linie a fișierului de intrare SUMDIV.IN sunt scrise cele două numere A și B , separate prin cel puțin un spațiu.

Date de ieșire

Prima linie a fișierului SUMDIV.OUT va conține restul împărțirii lui S la 9901.

Restricții

$0 \leq A, B \leq 50000000$ (cincizeci de milioane)

Exemplu

SUMDIV.IN	SUMDIV.OUT
2 3	15

Explicație

$$2^3 = 8.$$

Divizorii naturali ai lui 8 sunt: 1, 2, 4, 8. Suma lor este 15.

Restul împărțirii lui 15 la 9901 este 15 (care trebuie să apară în fișierul de ieșire).

Timp maxim de executare: 0.5 secunde (500 de milisecunde)/test

36.4.1 Indicații de rezolvare

Mihai Pătrașcu

Să presupunem că avem $A = 12$ și $B = 3$.

$$\text{Evident, } 12^3 = (2^2 \cdot 3)^3 = 2^6 \cdot 3^3$$

Un divizor al acestui număr are forma: $2^x \cdot 3^y$, $0 \leq x \leq 6$, $0 \leq y \leq 3$

Deci suma divizorilor este: $\sum_{\substack{x=0,6 \\ y=0,3}} 2^x \cdot 3^y$

Această sumă dublă se poate scrie ca: $(1 + 2 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6) \cdot (1 + 3 + 3^2 + 3^3)$

Există mai multe metode de a evalua în timp logaritmic progresiile geometrice care apar. Una dintre ele folosește în mod esențial faptul ca 9901 este prim.

Testarea

Test	A	B	S % 9901
0	1	45,000,963	1
1	34,002,980	0	1
2	9,901	999	1
3	1,576	1	2970
4	$2^{25} = 33,554,432$	$2^{25} = 33,554,432$	3612
5	$24,999,983 * 2 = 49,999,966$	$2^{25} - 1 = 33,554,431$	1739
6	$2^2 * 3 * 5 * 7 * 59,407 = 12,475,470$	90,000	9879
7	$2^2 * 3 * 3,980,203 = 47,762,436$	137	3575
8	$3^3 * 5^2 * 7^3 * 11 * 17 = 43,295,175$	25,983,137	3816
9	$3^3 * 5 * 7 * 11 * 13 * 17 * 19 = 43,648,605$	49,999,801	6407

Lămuriri:

- fie p cel mai mare factor prim al lui A ; factorizarea trebuie să fie $O(\sqrt{p})$; dacă este $O(p)$, testele 5, 7 și poate 1, nu vor fi trecute;
- dacă folosiți o metodă în timp liniar pentru a evalua progresia geometrică, testele 4, 5, 8, 9 și poate 6 nu vor fi trecute;
- un caz particular important pentru progresia geometrică este $q = 1$; dacă ati omis acest caz, nu veti obține punctaj pentru testele 6 și 7 (59407 % 9901 == 1).

36.4.2 Cod sursă

Listing 36.4.1: sumdiv.c

```

1 #pragma option -3 -r -Z -O2 -a
2 #include <stdio.h>
3
4 const int revs[9901] = {0, 1, 4951, 6601, 7426, 7921, 8251}; // ... toate ... !!!
5
6 #define PROD(a, b) ((int) (((long) (a) * (b)) % 9901))
7
8 int pow_mod(int q, long n)
9 {
10     int tmp;
11
12     if (!n)
13         return (1);
14     tmp = pow_mod(q, n >> 1);
15     tmp = PROD(tmp, tmp);
16     return ((n & 1) ? PROD(tmp, q) : tmp);
17 }
18
19 int calc_prog(int q, long n)
20 {
21     // ATTN: q = 1, esp. n > 9901
22     if (q == 1)
23         return ((int) ((n + 1) % 9901));
24     return (PROD(pow_mod(q, n + 1) + 9900, revs[q ? q - 1 : 9900]));
25 }
26
27 int main()

```

```

28  {
29      long a, b;
30      int res, i, cnt;
31
32      fscanf(fopen("sumdiv.in", "r"), "%ld%ld", &a, &b);
33
34      // ATTN: (2**k)*N
35      for(cnt = 0; !((int) a & 1); a >= 1, cnt++) ;
36      res = calc_prog(2, cnt * b);
37
38      for(i = 3; (long) i * i <= a; i += 2) {
39          if(a % i)
40              continue;
41          for(cnt = 1; !(a /= i) % i; cnt++) ;
42          res = PROD(res, calc_prog(i % 9901, cnt * b));
43      }
44
45      // ATTN: large primes
46      if(a > 1)
47          res = PROD(res, calc_prog((int) (a % 9901), b));
48
49      fprintf(fopen("sumdiv.out", "w"), "%d\n", res);
50
51  }

```

36.4.3 *Rezolvare detaliată

36.5 Sistem

Județul în care are loc Olimpiada Națională de Informatică de anul acesta este puțin ciudat. În județ există N orașe, numerotate de la 1 la N . Fiecare dintre cele N orașe ale județului este legat de EXACT alte 2 orașe, prin străzi bidirectionale. și mai ciudat este faptul că, în cadrul acestui sistem stradal, nu este întotdeauna posibil să ajungi din orice oraș în oricare alt oraș mergând pe străzi. Oricum, locuitorii județului sunt mândri de acest sistem al lor și sunt de părere că nu mai există altul la fel. Dumneavoastră vreți să le demonstrați contrariul și pentru aceasta vreți să calculați câte sisteme stradale distincte cu proprietatea de mai sus există. Două sisteme sunt considerate distincte dacă există cel puțin o stradă între o pereche de orașe i și j în cadrul primului sistem, care nu există în cadrul celui de-al doilea.

Cerință

Scrieți un program care să calculeze câte sisteme stradale distincte există.

Date de intrare

Din fișierul SISTEM.IN veți citi valoarea întreagă N , reprezentând numărul de orașe ale județului.

Date de ieșire

În fișierul SISTEM.OUT veți afișa o valoare întreagă, reprezentând numărul de sisteme stradale distincte, cu proprietatea că orice oraș este legat prin străzi directe de exact alte 2 orașe.

Restricții

$3 \leq N \leq 100$

Exemplu

SISTEM.IN	SISTEM.OUT
4	3

Cele 3 soluții sunt următoarele:

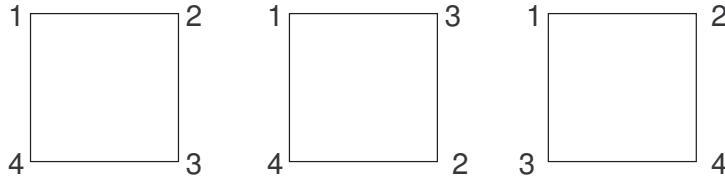


Figura 36.1: Sistem

SISTEM.IN	SISTEM.OUT
6	70

Timp maxim de executare: 1 secundă/test

36.5.1 Indicații de rezolvare

Mugurel Ionuț Andreica

Notăm cu G_i numărul de grafuri 2-regulate (în care fiecare nod are gradul 2). Avem:

$$G_0 = 1; G_1 = 0; G_2 = 0; G_3 = 1$$

Pentru $i > 3$, formula de calcul este următoarea:

$$G_i = \sum G_{i-k} * C(i-1, k-1) * (k-1)!/2, \quad k = 3, \dots, i$$

unde $C(i, j)$ reprezintă combinări de i luate câte j .

36.5.2 *Rezolvare detaliată

36.5.3 Cod sursă

Listing 36.5.1: Sist_ok.pas

```

1 { Mugurel Ionut Andreica - Bucuresti, ROMANIA }
2
3 Program SISTEM;
4
5 const filein='SISTEM.IN';
6   fileout='SISTEM.OUT';
7   MAXN=100;
8   NCIFRE=55;
9   baza=10000;
10  vmax=9999;
11  avans=2;
12
13 type NMARE=array[0..NCIFRE] of integer;
14   pNMARE^NMARE;
15   NMAREvect=array[0..MAXN] of NMARE;
16
17 var fin,fout:text;
18   n2g,fact:array[0..MAXN] of NMARE;
19   comb:array[0..MAXN] of ^NMAREvect;
20   i,j,k,m,n,p:integer;
21   act,naux,naux2:NMARE;
22
23 procedure adunam(n1,n2:pNMARE;rez:pNMARE);
24 var r,co,tt:integer;
25 begin
26 if (n1^[0]>n2^[0]) then
27   r:=n1^[0]
28 else
29   r:=n2^[0];
30 fillchar(rez^,sizeof(NMARE),0);
31 tt:=0;
```

```

33  for co:=1 to r+avans do
34    begin
35      rez^ [co]:=rez^ [co]+n1^ [co]+n2^ [co];
36      tt:=rez^ [co] div baza;
37      rez^ [co]:=rez^ [co] mod baza;
38      rez^ [co+1]:=rez^ [co+1]+tt;
39    end;
40
41  for co:=r+avans downto 1 do
42    if (rez^ [co]>0) then
43      begin
44        rez^ [0]:=co;
45        break;
46      end;
47  end;
48
49  procedure inmul(n1:pNMARE;x:longint;rez:pNMARE);
50  var r,co:integer;
51  tt:longint;
52  begin
53  fillchar(rez^ ,sizeof(NMARE),0);
54  for co:=1 to n1^ [0]+avans do
55    begin
56      tt:=n1^ [co];
57      tt:=tt*x;
58      rez^ [co]:=rez^ [co]+(tt mod baza);
59      rez^ [co+1]:=tt div baza;
60
61      tt:=rez^ [co] div baza;
62      rez^ [co]:=rez^ [co] mod baza;
63      rez^ [co+1]:=rez^ [co+1]+tt;
64    end;
65
66  for co:=n1^ [0]+avans downto 1 do
67    if (rez^ [co]>0) then
68      begin
69        rez^ [0]:=co;
70        break;
71      end;
72  end;
73
74  procedure imparie2(n1:pNMARE;rez:pNMARE);
75  var r,cif,ncf:integer;
76  begin
77  fillchar(rez^ ,sizeof(NMARE),0);
78  cif:=n1^ [0]+1;
79  ncf:=0;
80
81  r:=0;
82  while (r<2) do
83    begin
84      dec(cif);
85      r:=r*baza+n1^ [cif];
86    end;
87
88  inc(ncf);
89  rez^ [ncf]:=r shr 1;
90  r:=r and 1;
91
92  for cif:=cif-1 downto 1 do
93    begin
94      inc(ncf);
95      r:=r*baza+n1^ [cif];
96      rez^ [ncf]:=r shr 1;
97      r:=r and 1;
98    end;
99
100 rez^ [0]:=ncf;
101 for r:=1 to (ncf div 2) do
102  begin
103    cif:=rez^ [r];
104    rez^ [r]:=rez^ [ncf-r+1];
105    rez^ [ncf-r+1]:=cif;
106  end;
107 end;
108

```

```

109 procedure inmulnm(nn1,nn2:pNMARE;rrez:pNMARE);
110 var rr,cco:integer;
111   nnaux,nnaux2:NMARE;
112 begin
113   fillchar(rrez^,sizeof(NMARE),0);
114   for rr:=1 to nn1^[0] do
115     begin
116       inmul(nn2,nn1^[rr],@nnaux);
117
118       for cco:=nnaux[0]+rr-1 downto rr do
119         nnaux[cco]:=nnaux[cco-rr+1];
120       for cco:=1 to rr-1 do
121         nnaux[cco]:=0;
122       nnaux[0]:=nnaux[0]+rr-1;
123
124       adunanm(@nnaux,rrez,@nnaux2);
125       move(nnaux2,rrez^,sizeof(NMARE));
126     end;
127   end;
128
129 begin
130   assign(fin,filein);
131   reset(fin);
132   readln(fin,n);
133   close(fin);
134
135 { FACTORIALS }
136 fillchar(fact[0],sizeof(NMARE),0);
137 fact[0][0]:=1;
138 fact[0][1]:=1;
139
140 for i:=1 to n do
141   inmul(@fact[i-1],i,@fact[i]);
142 { END FATORIALS }
143
144 { COMBINATIONS }
145 new(comb[2]);
146 fillchar(comb[2]^,sizeof(NMAREvect),0);
147 for j:=0 to 2 do
148   begin
149     comb[2]^*[j][0]:=1;
150
151     if (j=1) then
152       comb[2]^*[j][1]:=2
153     else
154       comb[2]^*[j][1]:=1;
155   end;
156 { END COMBINATIONS }
157
158 fillchar(n2g[0],sizeof(NMARE),0);
159 n2g[0][0]:=1;
160 n2g[0][1]:=1;
161
162 for i:=3 to n do
163   begin
164
165     new(comb[i]);
166     fillchar(comb[i]^,sizeof(NMAREvect),0);
167
168     for j:=0 to i do
169       begin
170         if (j=0) or (j=i) then
171           begin
172             comb[i]^*[j][0]:=1;
173             comb[i]^*[j][1]:=1;
174           end
175         else
176           begin
177             adunanm(@comb[i-1]^*[j],@comb[i-1]^*[j-1],@comb[i]^*[j]);
178             {
179               comb[i,j]:=comb[i-1,j]+comb[i-1,j-1];
180             }
181           end;
182       end;
183
184     fillchar(n2g[i],sizeof(NMARE),0);

```

```

185
186     for j:=2 to i-1 do
187         if (i-j-1<>1) and (i-j-1<>2)
188         then
189             begin
190             {
191                 act:=(comb[i-1,j]*fact[j]*n2g[i-j-1])/2;
192                 n2g[i]:=n2g[i]+act;
193             }
194             inmulnm(@comb[i-1]^[j],@fact[j],@naux);
195             inmulnm(@naux,@n2g[i-j-1],@naux2);
196             imparate2(@naux2,@act);
197
198             adunanm(@act,@n2g[i],@naux);
199             move(naux,n2g[i],sizeof(NMARE));
200         end;
201
202         dispose(comb[i-1]);
203     end;
204
205 assign(fout,fileout);
206 rewrite(fout);
207
208 write(fout,n2g[n][n2g[n][0]]);
209 for i:=n2g[n][0]-1 downto 1 do
210     begin
211         p:=baza div 10;
212         while (p>0) do
213             begin
214                 write(fout,(n2g[n][i] div p) mod 10);
215                 p:=p div 10;
216             end;
217     end;
218 writeln(fout);
219 close(fout);
220 end.

```

36.6 Comitat

Toate semințiile conviețuitoare pe Terra au hotărât ca hobbitii, păstrătorii Inelului Puterii, să fie izolați într-o zonă a pământului numită Comitat. Hotarele Comitatului trebuie să fie reprezentate de un poligon convex cu câte un turn de pază în fiecare vârf.

Se cunosc pozițiile tuturor turnurilor din regiune (două numere naturale raportate la un sistem de axe rectangulare). Un paznic pe cal alb veghează hotarele comitatului parcugând, pe rând, toate distanțele dintre două turnuri succesive mergând pe drum minim, numai pe cărări paralele cu axele sistemului de axe.

Se cunoaște lungimea maximă a drumului pe care-l poate parurge paznicul la un tur complet al hotarelor comitatului și se cere să se determine un poligon cu un număr maxim de turnuri pe contur, poligon ce poate constitui hotarul comitatului. În plus, hotarul trebuie să conțină turnul din Mordor (de coordonate 0 și 0) într-un vârf, în fiecare dintre celelalte vârfuri aflându-se obligatoriu unul din turnurile existente.

De exemplu, pentru amplasamentul turnurilor ilustrat alăturat și pentru limita de 25 Km a unui tur efectuat de paznic, hotarul comitatului poate fi format, în această ordine, din turnurile de coordonate (0,0), (4,1), (8,3), (4,4), (1,4), (0,0). Se observă că poligonul determinat de aceste turnuri este un poligon convex cu 5 turnuri pe contur.

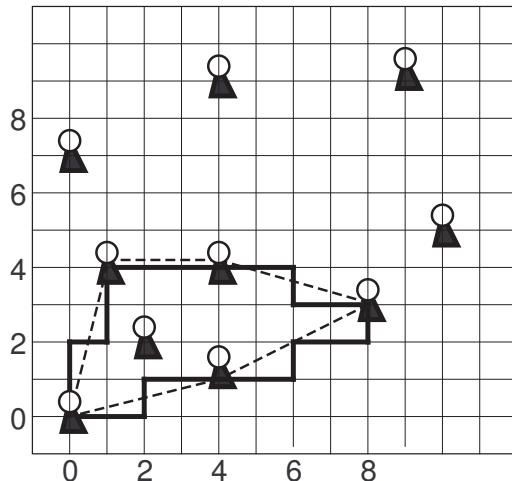


Figura 36.2: Comitat

Poligonul cu vârfurile $(0,0)$, $(4,1)$, $(4,12)$, $(0,7)$, $(0,0)$ are tot 5 turnuri pe contur, dar un tur complet al acestui poligon depășește 25 Km.

Date de intrare

Din fișierul COMITAT.IN se citesc, în ordine:

COMITAT.IN	Semnificație
n	numărul de turnuri din ținut
$x_1 y_1$	(cu excepția turnului implicit - Mordor)
$x_2 y_2$	coordonatele (abscisă ordonată)
...	ale fiecărui dintre cele n turnuri
$x_n y_n$	numărul reprezentând lungimea maximă a unui tur complet al poligonului
L	

Date de ieșire

În fișierul COMITAT.OUT se scriu:

COMITAT.IN	Semnificație
v	numărul de turnuri de pe conturul poligonului
$t_1 t_2 \dots t_{v-1}$	numerele de ordine (în ordinea din fișierul de intrare) ale turnurilor de pe contur, pornind de la turnul Mordor (care este implicit) și respectând succesiunea în sens trigonometric sau în sensul acelor de ceasornic a turnurilor de pe contur

Observații

- Pot exista turnuri strict în interiorul poligonului, dar acestea nu sunt luate în considerare pentru criteriul de maxim.
- Se consideră soluții și poligonul degenerat format dintr-un singur vârf (Mordor) sau din două vârfuri (Mordor și un alt turn) sau din mai multe vârfuri coliniare.
- Pot exista turnuri coliniare pe conturul poligonului determinat.
- Dacă există mai multe soluții ce respectă condițiile din enunț, se va furniza doar una dintre acestea.
- În fișierul de intrare nu există două turnuri ale căror poziții să coincidă și nu există un turn în poziția $(0,0)$.

Restricții

$$0 < n \leq 50$$

$$0 \leq x_i, y_i \leq 200$$

$$0 < L < 1000$$

Exemplu

COMITAT.IN (conform figurii)	COMITAT.OUT (o soluție posibilă)
9	5
0 7	4 7 5 2
1 4	
2 2	
4 1	
4 4	
4 9	
8 3	
9 9	
10 5	
25	

Timp maxim de executare: 1 secundă/test.

36.6.1 Indicații de rezolvare

Mugurel Ionuț Andreica și Rodica Pintea

O rezolvare de *programare dinamică* presupune construirea unui tablou alocat dinamic ce reține pentru fiecare triplet i, j, k :

- lungimea liniei convexe cu k turnuri pe contur, linie ce pornește din turnul de ordin 0 (Mordor) și are ultima latură determinată de turnurile de ordin i și j

$$lmin[i, j, k] = \min(lmin(j, p, k - 1) + distM(i, j)) \text{ astfel încât } (p, j, i) = \text{ colț convex}$$

36.6.2 Cod sursă

Listing 36.6.1: COMI.pas

```

1 Program COMITAT;
2
3 const filein='comitat.in';
4     fileout='comitat.out';
5     MAXN=51;
6     MARE=1001;
7     eps:double=1e-8;
8
9 type linie=array[0..MAXN] of integer;
10    plinie^linie;
11    lin2=array[0..MAXN] of byte;
12    plin2^lin2;
13
14 var fin,fout:text;
15     lmin:array[0..MAXN,0..MAXN] of plinie;
16     pred:array[0..MAXN,0..MAXN] of plin2;
17     x,y,ind:array[0..MAXN] of longint;
18     ung:array[0..MAXN] of double;
19     i,j,ii,jj,k,kk,n,p,pp,l,plus,a,b,c,max,li,lj:longint;
20     ok:boolean;
21
22 begin
23 assign(fin,filein);
24 reset(fin);
25 readln(fin,n);
26 for i:=1 to n do
27 begin
28   readln(fin,x[i],y[i]);
29   ind[i]:=i;
30 end;
31 readln(fin,l);
32 close(fin);
33
34 x[0]:=0;
35 y[0]:=0;
36 ind[0]:=0;
37
38 for i:=1 to n do

```

```

39   begin
40     if (x[i]=0) then
41       ung[i]:=pi/2
42     else
43       ung[i]:=arctan(y[i]/x[i]);
44   end;
45
46 for i:=0 to n-1 do
47   for j:=i+1 to n do
48     if (ung[ind[i]]>ung[ind[j]]) or
49       ((abs(ung[ind[i]]-ung[ind[j]])<eps) and
50       ((y[ind[i]]<y[ind[j]]) or (x[ind[i]]<x[ind[j]]))) then
51     begin
52       k:=ind[i];
53       ind[i]:=ind[j];
54       ind[j]:=k;
55     end;
56
57 for i:=0 to n do
58   for j:=0 to n do
59     begin
60       new(lmin[i,j]);
61       for k:=0 to n+1 do
62         lmin[i,j]^[k]:=MARE;
63
64       new(pred[i,j]);
65       fillchar(pred[i,j]^,sizeof(linie),0);
66     end;
67
68 p:=-1;
69 repeat
70 inc(p);
71 pp:=p;
72
73 while (abs(ung[ind[pp]]-ung[ind[p]])<eps) and (pp<n) do
74   inc(pp);
75
76 if (abs(ung[ind[pp]]-ung[ind[p]])>eps) then
77   dec(pp);
78
79 if (pp-p>0) then
80   begin
81     i:=ind[pp];
82     j:=ind[pp-1];
83
84     if (i<>0) then
85       plus:=1
86     else
87       plus:=0;
88
89     lmin[j,i]^[2+plus]:=abs(x[i]-x[j])+abs(y[i]-y[j])+
90                           abs(x[i]-x[0])+abs(y[i]-y[0]);
91     pred[j,i]^[2+plus]:=0;
92
93     for ii:=pp-2 downto p do
94       begin
95         i:=ind[ii+1];
96         j:=ind[ii];
97
98         lmin[j,i]^[pp-ii+1+plus]:=lmin[i,ind[ii+2]]^[pp-ii+plus]+
99                           abs(x[i]-x[j])+abs(y[i]-y[j]);
100        pred[j,i]^[pp-ii+1+plus]:=ind[ii+2];
101      end;
102    end;
103
104 p:=pp;
105 until (p=n);
106
107 for i:=1 to n do
108   lmin[i,0]^[2]:=x[i]+y[i];
109
110 for ii:=1 to n do
111   begin
112     i:=ind[ii];
113
114     for jj:=0 to ii-1 do

```

```

115      begin
116        j:=ind[jj];
117        for k:=3 to ii+1 do
118          begin
119            for pp:=0 to n do
120              if (ind[pp]<>j) and (ind[pp]<>i) and
121                (
122                  (abs(ung[ind[ii]]-ung[ind[jj]])>eps) or
123                  (
124                    (ung[ind[pp]]<ung[ind[jj]]) or
125                    (
126                      (pp<jj) and (pp<ii)
127                    )
128                  )
129                )
130              )
131            )
132          then
133            begin
134              p:=ind[pp];
135
136              a:=y[p]-y[j];
137              b:=x[j]-x[p];
138              c:=x[p]*y[j]-x[j]*y[p];
139
140              plus:=lmin[j,p]^(k-1)+abs(x[i]-x[j])+abs(y[i]-y[j]);
141              if (a*x[i]+b*y[i]+c>=0) and (plus<lmin[i,j]^k) then
142                begin
143                  lmin[i,j]^k:=plus;
144                  pred[i,j]^k:=p;
145                end;
146              end;
147            end;
148          end;
149        end;
150
151 max:=0;
152 for ii:=0 to n do
153  begin
154    i:=ind[ii];
155    for jj:=0 to n do
156      if (ii<>jj) then
157        begin
158          j:=ind[jj];
159
160          {
161            a:=y[j]-y[i];
162            b:=x[i]-x[j];
163          }
164          c:=x[j]*y[i]-x[i]*y[j];
165
166          if (c>=0) then
167            begin
168              for k:=2 to n+1 do
169                if (k>max) then
170                  begin
171                    plus:=lmin[i,j]^k+x[i]+y[i];
172
173                    if (plus<=l) then
174                      begin
175                        max:=k;
176                        li:=i;
177                        lj:=j;
178                      end;
179                  end;
180                end;
181            end;
182          end;
183        end;
184 assign(fout,fileout);
185 rewrite(fout);
186 writeln(fout,max);
187
188 write(fout,li);
189 while (lj>0) do
190   begin

```

```

191     i:=pred[li,lj]^*[max];
192     li:=lj;
193     lj:=i;
194     max:=max-1;
195
196     write(fout,' ',li);
197   end;
198
199 writeln(fout);
200 close(fout);
201 end.
```

Listing 36.6.2: COMIBCK.pas

```

1 Program COMITAT_BACK;
2
3 const filein='comitat.in';
4     fileout='comitat.out';
5     MAXN=51;
6     maxticks=17;
7
8 var x,y,sol,st:array[0..51] of integer;
9     used:array[0..MAXN] of boolean;
10    i,j,k,n,l,lmax,max:integer;
11
12   t2:longint absolute 0:$046c;
13   t1:longint;
14
15 procedure print;
16 begin
17   assign(output,fileout);
18   rewrite(output);
19   writeln(max);
20   for i:=1 to max-1 do
21     write(sol[i],' ');
22   writeln;
23   close(output);
24   halt(0);
25 end;
26
27 function convex(p1,p2,p3:integer):boolean;
28 var a,b,c,rez:longint;
29 begin
30   a:=y[p1]-y[p2];
31   b:=x[p2]-x[p1];
32   c:=x[p1]*y[p2]-x[p2]*y[p1];
33
34   rez:=a*x[p3]+b*y[p3]+c;
35
36   if (rez>0) then
37     convex:=true
38   else
39     if (rez<0) then
40       convex:=false
41     else
42       begin
43         if (y[p2]=y[p1]) then
44           begin
45             if (abs(y[p3]-y[p2])<abs(y[p3]-y[p1])) and
46               (abs(y[p3]-y[p1])>abs(y[p2]-y[p1])) then
47                 convex:=true
48             else
49               convex:=false;
50           end
51         else
52           begin
53             if (abs(x[p3]-x[p2])<abs(x[p3]-x[p1])) and
54               (abs(x[p3]-x[p1])>abs(x[p2]-x[p1])) then
55                 convex:=true
56             else
57               convex:=false;
58           end;
59       end;
60   end;
```

```

62  function colin0(p1,p2:integer):boolean;
63  begin
64  colin0:=(x[p1]*y[p2]-x[p2]*y[p1]=0);
65  end;
66
67  procedure back(niv:integer);
68  var s:integer;
69  begin
70  if (t2-t1>maxticks) then
71    print;
72
73  if (niv>max) and
74    ( (convex(st[niv-2],st[niv-1],0)) or
75      (colin0(st[niv-2],st[niv-1]))) and
76      (l+x[st[niv-1]]+y[st[niv-1]]<=lmax)
77  then
78    begin
79      max:=niv;
80      move(st,sol,sizeof(st));
81    end;
82
83  for s:=1 to n do
84    if (not used[s]) and (convex(st[niv-2],st[niv-1],s)) and
85    ( (convex(st[niv-1],s,0)) or (colin0(st[niv-1],s)) ) and
86    (l+abs(x[s]-x[st[niv-1]])+abs(y[s]-y[st[niv-1]])<=lmax)
87  then
88    begin
89      used[s]:=true;
90      st[niv]:=s;
91      l:=l+(abs(x[s]-x[st[niv-1]])+abs(y[s]-y[st[niv-1]]));
92
93      back(niv+1);
94
95      l:=l-(abs(x[s]-x[st[niv-1]])+abs(y[s]-y[st[niv-1]]));
96      st[niv]:=0;
97      used[s]:=false;
98    end;
99  end;
100
101 begin
102 t1:=t2;
103 assign(input,filein);
104 reset(input);
105 readln(n);
106
107 x[0]:=0; y[0]:=0;
108 for i:=1 to n do
109   readln(x[i],y[i]);
110 readln(lmax);
111 close(input);
112
113 max:=1;
114 sol[0]:=0;
115
116 fillchar(used,sizeof(used),false);
117 used[0]:=true;
118 for i:=1 to n do
119 begin
120   used[i]:=true;
121   st[1]:=i;
122   l:=x[i]+y[i];
123
124   if (2*l<=lmax) then
125     begin
126       if (max<2) then
127         begin
128           max:=2;
129           sol[1]:=i;
130         end;
131
132       back(2);
133     end;
134
135   st[1]:=0;
136   used[i]:=false;
137 end;

```

138

```
139 print;
140 end.
```

Listing 36.6.3: COMITATB.pas

```

1 program comitat;
2 const fi='comitat.in';fo='comitat.out';
3     nmax=500;dmax=200;
4 type turn=record x,y:byte end;
5     turnuri=array[0..nmax]of turn;
6 var n,nb,cmax,imax,jmax,l:word;
7     t,tb:turnuri;
8     ales:array[0..nmax]of boolean;
9     st:array[1..nmax]of word;
10    m:array[0..dmax,0..dmax]of boolean;
11 procedure citire;
12 var f:text; i,j:word;
13 begin
14     assign(f,fi);reset(f);
15     readln(f,n);
16     for i:=1 to n do
17         with t[i] do readln(f,x,y);
18     readln(f,l);l:=l div 2;
19     if l>dmax then l:=dmax;
20     close(f);
21 end;
22 procedure sort;
23 var aux:turn;i,j:word;
24 begin
25     for i:=1 to nb-1 do
26         for j:=i+1 to nb do
27             if (tb[i].y>tb[j].y) or
28                 (tb[i].y=tb[j].y)and(tb[i].x>tb[j].x) then begin
29                 aux:=tb[i];tb[i]:=tb[j];tb[j]:=aux
30             end
31     end;
32 function minus(t1,t2,t3:turn):boolean;
33 var a,b,c:real;
34 begin
35     a:=t1.y-t2.y;b:=t2.x-t1.x;
36     c:=t1.x*t2.y-t2.x*t1.y;
37     minus:=a*t3.x+b*t3.y+c<0;
38 end;
39 function conv:byte;
40 var vf:word;i,pas:integer;
41 begin
42     for i:=0 to nb do ales[i]:=false;
43     sort;
44     pas:=1;st[1]:=0;st[2]:=1;
45     ales[1]:=true;vf:=2;i:=1;
46     if nb=1 then begin conv:=2;exit end;
47     while i>0 do begin
48         while ales[i] do begin
49             i:=i+pas;
50             if i=nb then pas:=-1
51         end;
52         if i=-1 then break;
53         while (vf>1)and minus(tb[st[vf]],tb[st[vf-1]],tb[i]) do begin
54             ales[st[vf]]:=false;dec(vf)
55         end;
56         inc(vf);st[vf]:=i;ales[i]:=true
57     end;
58     conv:=vf-1
59 end;
60 function calc(x,y:word):word;
61 var i:word;
62 begin
63     nb:=0;
64     for i:=1 to n do
65         if (t[i].x<=x) and (t[i].y<=y) then begin
66             inc(nb);tb[nb]:=t[i]
67         end;
68         if nb>cmax then calc:=conv else calc:=0
69 end;

```

```

70  function e(x,y:word):boolean;
71  var i:word;ex,ey:boolean;
72  begin
73      ex:=false;ey:=false;
74      for i:=1 to n do
75          if (t[i].x<=x) and (t[i].y<=y) then begin
76              if t[i].x=x then ex:=true;
77              if t[i].y=y then ey:=true
78          end;
79      e:=ex and ey;
80  end;
81  procedure mat;
82  var i,j,cate:word;
83  begin
84      for i:=0 to l do
85          for j:=0 to l-i do
86              if e(j,i) then begin
87                  cate:=calc(j,i);
88                  if cate>cmax then begin
89                      cmax:=cate;imax:=i;jmax:=j
90                  end
91              end
92  end;
93  procedure scrie;
94  var f:text;var i,j:word;
95  begin
96      assign(f,fo);rewrite(f);
97      writeln(f,cmax);
98      calc(jmax,imax);
99      tb:=t;
100     for i:=2 to cmax do
101         for j:=1 to n do
102             if (t[j].x=tb[st[i]].x) and (t[j].y=tb[st[i]].y)
103                 then write(f,j,' ');
104     close(f)
105  end;
106  begin
107      citire;
108      cmax:=1;
109      mat;
110      scrie
111  end.

```

36.6.3 *Rezolvare detaliată

Capitolul 37

ONI 2001

37.1 Comparări

Bogdan Dumitru, Bucureşti

Fie un şir de N numere întregi. Numim **al doilea maxim** al şirului cel mai mare număr din şir, cu excepția maximului. Pentru a afla maximul și al doilea maxim, veți efectua comparări între elementele şirului.

Având la dispoziție numai $N + \lceil \log_2 N \rceil - 2$ comparări, aflați pozițiile pe care le ocupă maximul și al doilea maxim. Prin $\lceil \log_2 N \rceil$ s-a notat partea întreagă superioară a lui "logaritm în baza 2 din N ".

Cerință

Aflați cele două numere, efectuând cel mult $N + \lceil \log_2 N \rceil - 2$ comparări. Pentru a putea efectua comparările, comisia vă pune la dispoziție funcția:

int Compara(int i, int j) (pentru cei care programează în C sau C++), respectiv funcția

Compara(i,j:Integer):Integer; (pentru programatorii în Pascal), care compară numărul de pe poziția i cu cel de pe poziția j și vă returnează 1, dacă elementul aflat pe poziția i este mai mare sau egal cu cel de pe poziția j , respectiv -1, în caz contrar.

Programul trebuie să apeleze la început funcția **GetN** care returnează numărul de elemente din şir. Apelul va fi de forma **N=GetN()**, pentru programatorii în C (și C++), respectiv **N:=GetN**, pentru programatorii în Pascal, unde am presupus că N este variabilă de tip **int**, respectiv **Integer**, în care este memorată lungimea şirului. Este esențial ca apelul funcției **GetN** să preceadă apelurile funcției **Compara** și în plus, să existe un singur apel al funcției **GetN** (în caz contrar programul va primi 0 puncte pe acel test).

În directorul C:\OLIMP programatorii în C vor găsi fișierul **Compar.h**, iar cei care lucrează în Pascal vor găsi fișierul **Compar.pas** reprezentând sursele unit-ului comisiei.

În fișierul **Compar.out** veți scrie pozițiile pe care se află maximul şirului și al doilea maxim, separate printr-un blanc.

Pentru a folosi unitul, trebuie să creați fișierul **Compar.in**, în care veți scrie pe prima linie numărul N , iar pe a doua linie elementele şirului, separate prin câte un spațiu.

Versiunea Pascal a unit-ului este:

```
Unit compar;

interface

const my_nmax=1000;
var my_a:array[1..my_nmax+20] of Integer;
     my_apelat, my_n, my_napel, my_apel : Integer;
function GetN:Integer;
function Compara(i,j:Integer):Integer;

implementation

function GetN:Integer;
var f:Text;
    i,log,tmp:Integer;
```

```

begin
  my_apelat:=1;
  Assign(f,'COMPAR.IN');
  Reset(f);
  Readln(f,my_n);
  for i:=1 to my_n do
    Read(f,my_a[i]);
    Close(f);
    my_apel:=0;
    tmp:=1;
    log:=0;
    while tmp<my_n do
      begin
        tmp:=tmp*2;
        log:=log+1
      end;
      my_napel:=my_n+log-2;
      getn:=my_n
    end;

function compara(i,j:Integer):Integer;
var f:Text;
begin
  if my_apelat=0 then
    begin
      Writeln('Nu ati apelat functia GetN!');
      Halt
    end;
    my_apel:=my_apel+1;
    if my_apel>my_napel then
      begin
        Writeln('Prea multe apeluri!');
        Halt
      end;
    if (i<1) or (i>my_n) or (j<1) or (j>my_n) then
      begin
        Writeln('Apel ilegal!');
        Halt
      end;
    if my_a[i]>=my_a[j] then
      compara:=1
    else compara:=-1
  end;

End.

```

Implementarea C/C++ a acelorași funcții este următoarea:

```

#include<stdio.h>
#include<math.h>
#include<process.h>
#define my_nmax 1000

int my_a[my_nmax+20];
int my_apelat, my_n, my_napel, my_apel;

extern int GetN(void)
{
  FILE *f;
  int i,log,tmp;
  my_apelat=1;
  f=fopen("COMPAR.IN","r");

```

```

fscanf(f,"%d",&my_n);
for (i=1;i<=my_n;i++)
    fscanf(f,"%d",&my_a[i]);
fclose(f);
my_apel=0;
tmp=1;
log=0;
while(tmp<my_n)
{
    tmp=tmp*2;
    log++;
}
my_napel=my_n+log-2;
return my_n;
}

extern int Compara(int i, int j)
{
    FILE *f;
    if (!my_apelat)
    {
        puts("Nu ati apelat functia GetN()!");
        exit(0);
    }
    my_apel=my_apel+1;
    if(my_apel>my_napel)
    {
        puts("Prea multe apeluri!");
        exit(0);
    }
    if(i<1 || i>my_n || j<1 || j>my_n)
    {
        puts("Apel ilegal!");
        exit(0);
    };
    if(my_a[i]>=my_a[j])
        return 1;
    else return -1;
}

```

Restricții $2 \leq N \leq 1000$ **Exemplul 1**

Dacă sirul considerat este (5, 3, 4), atunci o executare corectă a programului este următoarea:

...;
 N = GetN(), obțineți $N = 3$;
 Compara(1, 2), se returnează 1;
 Compara(1, 3), se returnează 1;
 Compara(2, 3), se returnează -1;
 ...;

În acest moment ați epuizat numărul de comparări permise ($3 + 2 - 2 = 3$ comparări) și în fișierul **Compar.out** veți scrie numerele 1 și 3, separate printr-un spațiu, reprezentând pozițiile pe care se află maximul sirului (5), respectiv al doilea maxim (4).

Exemplul 2

Dacă sirul considerat este (5, 7, 7), o executare corectă a programului este următoarea:

...;
 N = GetN(), obțineți $N = 3$;
 Compara(1, 2), se returnează -1;
 Compara(2, 3), returnează 1;
 Compara(1, 3), se returnează -1;

...;

În acest moment ați epuizat numărul de comparări permise ($3 + 2 - 2 = 3$ comparări) și în fișierul **Compar.out** veți scrie numerele 2 și 3, separate printr-un spațiu, reprezentând pozițiile pe care se află maximul sirului (7), respectiv al doilea maxim (7).

Observație:

Pentru acest exemplu, mai există o soluție corectă, cea în care considerați maximul (7) pe poziția 3 și al doilea maxim (7) pe poziția 2.

Timp maxim de executare/test: 1 secundă.

Se garantează că 1008 apeluri ale funcției **Compara**, împreună cu un apel al funcției **GetN** nu durează mai mult de 0.1 secunde.

În timpul concursului, pentru a vă putea testa programele, veți avea la dispoziție module *echivalente* cu cele pe care le va utiliza comisia în timpul evaluării programelor voastre.

37.1.1 *Indicații de rezolvare

37.1.2 Cod sursă

Listing 37.1.1: Comp_MIA.pas

```

1 { Mugurel Ionut Andreica - Bucuresti, ROMANIA }
2 Program CompArator;
3 uses compar;
4
5 type longvect=array[1..1000,1..15] of longint;
6
7 var fout:text;
8     i,j,k,n,p,pant,pmax,pmax2,lim,nco,s:integer;
9     nint:array[1..100] of integer;
10    v,prov:^longvect;
11
12 begin
13
14 n:=GetN;
15
16 new(v);
17 fillchar(v^,sizeof(v^),0);
18 new(prov);
19 fillchar(prov^,sizeof(prov^),0);
20
21 nint[1]:=n;
22
23 for i:=1 to n do
24   v^[i,1]:=i;
25 k:=1;
26
27 while (nint[k]>1) do
28   begin
29     inc(k);
30     for i:=1 to nint[k-1] do
31       if (i mod 2=1) then
32         begin
33           if (i+1>nint[k-1]) then
34             begin
35               v^[(i+1) div 2,k]:=v^[i,k-1];
36               prov^[(i+1) div 2,k]:=i;
37             end
38           else
39             begin
40               if (compara(v^[i,k-1],v^[i+1,k-1])=1) then
41                 begin
42                   v^[(i+1) div 2,k]:=v^[i,k-1];
43                   prov^[(i+1) div 2,k]:=i;
44                 end
45               else
46                 begin
47                   v^[(i+1) div 2,k]:=v^[i+1,k-1];
48                 end;
49             end;
50           end;
51         end;
52       end;
53     end;
54   end;
55
56   begin
57     write(fout,'');
58     for i:=1 to nint[k] do
59       write(fout,v^[i,k]);
60     writeln(fout);
61   end;
62
63   for i:=1 to nint[k] do
64     v^[i,k]:=0;
65   end;
66
67   begin
68     write(fout,'');
69     for i:=1 to nint[k] do
70       write(fout,prov^[i,k]);
71     writeln(fout);
72   end;
73
74   for i:=1 to nint[k] do
75     prov^[i,k]:=0;
76   end;
77
78   begin
79     write(fout,'');
80     for i:=1 to nint[k] do
81       write(fout,nint[i]);
82     writeln(fout);
83   end;
84
85   for i:=1 to nint[k] do
86     nint[i]:=0;
87   end;
88
89   begin
90     write(fout,'');
91     for i:=1 to nint[k] do
92       write(fout,v^[i,k]);
93     writeln(fout);
94   end;
95
96   for i:=1 to nint[k] do
97     v^[i,k]:=0;
98   end;
99
100  begin
101    write(fout,'');
102    for i:=1 to nint[k] do
103      write(fout,prov^[i,k]);
104    writeln(fout);
105  end;
106
107  for i:=1 to nint[k] do
108    prov^[i,k]:=0;
109  end;
110
111  begin
112    write(fout,'');
113    for i:=1 to nint[k] do
114      write(fout,nint[i]);
115    writeln(fout);
116  end;
117
118  for i:=1 to nint[k] do
119    nint[i]:=0;
120  end;
121
122  begin
123    write(fout,'');
124    for i:=1 to nint[k] do
125      write(fout,v^[i,k]);
126    writeln(fout);
127  end;
128
129  for i:=1 to nint[k] do
130    v^[i,k]:=0;
131  end;
132
133  begin
134    write(fout,'');
135    for i:=1 to nint[k] do
136      write(fout,prov^[i,k]);
137    writeln(fout);
138  end;
139
140  for i:=1 to nint[k] do
141    prov^[i,k]:=0;
142  end;
143
144  begin
145    write(fout,'');
146    for i:=1 to nint[k] do
147      write(fout,nint[i]);
148    writeln(fout);
149  end;
150
151  for i:=1 to nint[k] do
152    nint[i]:=0;
153  end;
154
155  begin
156    write(fout,'');
157    for i:=1 to nint[k] do
158      write(fout,v^[i,k]);
159    writeln(fout);
160  end;
161
162  for i:=1 to nint[k] do
163    v^[i,k]:=0;
164  end;
165
166  begin
167    write(fout,'');
168    for i:=1 to nint[k] do
169      write(fout,prov^[i,k]);
170    writeln(fout);
171  end;
172
173  for i:=1 to nint[k] do
174    prov^[i,k]:=0;
175  end;
176
177  begin
178    write(fout,'');
179    for i:=1 to nint[k] do
180      write(fout,nint[i]);
181    writeln(fout);
182  end;
183
184  for i:=1 to nint[k] do
185    nint[i]:=0;
186  end;
187
188  begin
189    write(fout,'');
190    for i:=1 to nint[k] do
191      write(fout,v^[i,k]);
192    writeln(fout);
193  end;
194
195  for i:=1 to nint[k] do
196    v^[i,k]:=0;
197  end;
198
199  begin
200    write(fout,'');
201    for i:=1 to nint[k] do
202      write(fout,prov^[i,k]);
203    writeln(fout);
204  end;
205
206  for i:=1 to nint[k] do
207    prov^[i,k]:=0;
208  end;
209
210  begin
211    write(fout,'');
212    for i:=1 to nint[k] do
213      write(fout,nint[i]);
214    writeln(fout);
215  end;
216
217  for i:=1 to nint[k] do
218    nint[i]:=0;
219  end;
220
221  begin
222    write(fout,'');
223    for i:=1 to nint[k] do
224      write(fout,v^[i,k]);
225    writeln(fout);
226  end;
227
228  for i:=1 to nint[k] do
229    v^[i,k]:=0;
230  end;
231
232  begin
233    write(fout,'');
234    for i:=1 to nint[k] do
235      write(fout,prov^[i,k]);
236    writeln(fout);
237  end;
238
239  for i:=1 to nint[k] do
240    prov^[i,k]:=0;
241  end;
242
243  begin
244    write(fout,'');
245    for i:=1 to nint[k] do
246      write(fout,nint[i]);
247    writeln(fout);
248  end;
249
250  for i:=1 to nint[k] do
251    nint[i]:=0;
252  end;
253
254  begin
255    write(fout,'');
256    for i:=1 to nint[k] do
257      write(fout,v^[i,k]);
258    writeln(fout);
259  end;
260
261  for i:=1 to nint[k] do
262    v^[i,k]:=0;
263  end;
264
265  begin
266    write(fout,'');
267    for i:=1 to nint[k] do
268      write(fout,prov^[i,k]);
269    writeln(fout);
270  end;
271
272  for i:=1 to nint[k] do
273    prov^[i,k]:=0;
274  end;
275
276  begin
277    write(fout,'');
278    for i:=1 to nint[k] do
279      write(fout,nint[i]);
280    writeln(fout);
281  end;
282
283  for i:=1 to nint[k] do
284    nint[i]:=0;
285  end;
286
287  begin
288    write(fout,'');
289    for i:=1 to nint[k] do
290      write(fout,v^[i,k]);
291    writeln(fout);
292  end;
293
294  for i:=1 to nint[k] do
295    v^[i,k]:=0;
296  end;
297
298  begin
299    write(fout,'');
300    for i:=1 to nint[k] do
301      write(fout,prov^[i,k]);
302    writeln(fout);
303  end;
304
305  for i:=1 to nint[k] do
306    prov^[i,k]:=0;
307  end;
308
309  begin
310    write(fout,'');
311    for i:=1 to nint[k] do
312      write(fout,nint[i]);
313    writeln(fout);
314  end;
315
316  for i:=1 to nint[k] do
317    nint[i]:=0;
318  end;
319
320  begin
321    write(fout,'');
322    for i:=1 to nint[k] do
323      write(fout,v^[i,k]);
324    writeln(fout);
325  end;
326
327  for i:=1 to nint[k] do
328    v^[i,k]:=0;
329  end;
330
331  begin
332    write(fout,'');
333    for i:=1 to nint[k] do
334      write(fout,prov^[i,k]);
335    writeln(fout);
336  end;
337
338  for i:=1 to nint[k] do
339    prov^[i,k]:=0;
340  end;
341
342  begin
343    write(fout,'');
344    for i:=1 to nint[k] do
345      write(fout,nint[i]);
346    writeln(fout);
347  end;
348
349  for i:=1 to nint[k] do
350    nint[i]:=0;
351  end;
352
353  begin
354    write(fout,'');
355    for i:=1 to nint[k] do
356      write(fout,v^[i,k]);
357    writeln(fout);
358  end;
359
360  for i:=1 to nint[k] do
361    v^[i,k]:=0;
362  end;
363
364  begin
365    write(fout,'');
366    for i:=1 to nint[k] do
367      write(fout,prov^[i,k]);
368    writeln(fout);
369  end;
370
371  for i:=1 to nint[k] do
372    prov^[i,k]:=0;
373  end;
374
375  begin
376    write(fout,'');
377    for i:=1 to nint[k] do
378      write(fout,nint[i]);
379    writeln(fout);
380  end;
381
382  for i:=1 to nint[k] do
383    nint[i]:=0;
384  end;
385
386  begin
387    write(fout,'');
388    for i:=1 to nint[k] do
389      write(fout,v^[i,k]);
390    writeln(fout);
391  end;
392
393  for i:=1 to nint[k] do
394    v^[i,k]:=0;
395  end;
396
397  begin
398    write(fout,'');
399    for i:=1 to nint[k] do
400      write(fout,prov^[i,k]);
401    writeln(fout);
402  end;
403
404  for i:=1 to nint[k] do
405    prov^[i,k]:=0;
406  end;
407
408  begin
409    write(fout,'');
410    for i:=1 to nint[k] do
411      write(fout,nint[i]);
412    writeln(fout);
413  end;
414
415  for i:=1 to nint[k] do
416    nint[i]:=0;
417  end;
418
419  begin
420    write(fout,'');
421    for i:=1 to nint[k] do
422      write(fout,v^[i,k]);
423    writeln(fout);
424  end;
425
426  for i:=1 to nint[k] do
427    v^[i,k]:=0;
428  end;
429
430  begin
431    write(fout,'');
432    for i:=1 to nint[k] do
433      write(fout,prov^[i,k]);
434    writeln(fout);
435  end;
436
437  for i:=1 to nint[k] do
438    prov^[i,k]:=0;
439  end;
440
441  begin
442    write(fout,'');
443    for i:=1 to nint[k] do
444      write(fout,nint[i]);
445    writeln(fout);
446  end;
447
448  for i:=1 to nint[k] do
449    nint[i]:=0;
450  end;
451
452  begin
453    write(fout,'');
454    for i:=1 to nint[k] do
455      write(fout,v^[i,k]);
456    writeln(fout);
457  end;
458
459  for i:=1 to nint[k] do
460    v^[i,k]:=0;
461  end;
462
463  begin
464    write(fout,'');
465    for i:=1 to nint[k] do
466      write(fout,prov^[i,k]);
467    writeln(fout);
468  end;
469
470  for i:=1 to nint[k] do
471    prov^[i,k]:=0;
472  end;
473
474  begin
475    write(fout,'');
476    for i:=1 to nint[k] do
477      write(fout,nint[i]);
478    writeln(fout);
479  end;
480
481  for i:=1 to nint[k] do
482    nint[i]:=0;
483  end;
484
485  begin
486    write(fout,'');
487    for i:=1 to nint[k] do
488      write(fout,v^[i,k]);
489    writeln(fout);
490  end;
491
492  for i:=1 to nint[k] do
493    v^[i,k]:=0;
494  end;
495
496  begin
497    write(fout,'');
498    for i:=1 to nint[k] do
499      write(fout,prov^[i,k]);
500    writeln(fout);
501  end;
502
503  for i:=1 to nint[k] do
504    prov^[i,k]:=0;
505  end;
506
507  begin
508    write(fout,'');
509    for i:=1 to nint[k] do
510      write(fout,nint[i]);
511    writeln(fout);
512  end;
513
514  for i:=1 to nint[k] do
515    nint[i]:=0;
516  end;
517
518  begin
519    write(fout,'');
520    for i:=1 to nint[k] do
521      write(fout,v^[i,k]);
522    writeln(fout);
523  end;
524
525  for i:=1 to nint[k] do
526    v^[i,k]:=0;
527  end;
528
529  begin
530    write(fout,'');
531    for i:=1 to nint[k] do
532      write(fout,prov^[i,k]);
533    writeln(fout);
534  end;
535
536  for i:=1 to nint[k] do
537    prov^[i,k]:=0;
538  end;
539
540  begin
541    write(fout,'');
542    for i:=1 to nint[k] do
543      write(fout,nint[i]);
544    writeln(fout);
545  end;
546
547  for i:=1 to nint[k] do
548    nint[i]:=0;
549  end;
550
551  begin
552    write(fout,'');
553    for i:=1 to nint[k] do
554      write(fout,v^[i,k]);
555    writeln(fout);
556  end;
557
558  for i:=1 to nint[k] do
559    v^[i,k]:=0;
560  end;
561
562  begin
563    write(fout,'');
564    for i:=1 to nint[k] do
565      write(fout,prov^[i,k]);
566    writeln(fout);
567  end;
568
569  for i:=1 to nint[k] do
570    prov^[i,k]:=0;
571  end;
572
573  begin
574    write(fout,'');
575    for i:=1 to nint[k] do
576      write(fout,nint[i]);
577    writeln(fout);
578  end;
579
580  for i:=1 to nint[k] do
581    nint[i]:=0;
582  end;
583
584  begin
585    write(fout,'');
586    for i:=1 to nint[k] do
587      write(fout,v^[i,k]);
588    writeln(fout);
589  end;
590
591  for i:=1 to nint[k] do
592    v^[i,k]:=0;
593  end;
594
595  begin
596    write(fout,'');
597    for i:=1 to nint[k] do
598      write(fout,prov^[i,k]);
599    writeln(fout);
600  end;
601
602  for i:=1 to nint[k] do
603    prov^[i,k]:=0;
604  end;
605
606  begin
607    write(fout,'');
608    for i:=1 to nint[k] do
609      write(fout,nint[i]);
610    writeln(fout);
611  end;
612
613  for i:=1 to nint[k] do
614    nint[i]:=0;
615  end;
616
617  begin
618    write(fout,'');
619    for i:=1 to nint[k] do
620      write(fout,v^[i,k]);
621    writeln(fout);
622  end;
623
624  for i:=1 to nint[k] do
625    v^[i,k]:=0;
626  end;
627
628  begin
629    write(fout,'');
630    for i:=1 to nint[k] do
631      write(fout,prov^[i,k]);
632    writeln(fout);
633  end;
634
635  for i:=1 to nint[k] do
636    prov^[i,k]:=0;
637  end;
638
639  begin
640    write(fout,'');
641    for i:=1 to nint[k] do
642      write(fout,nint[i]);
643    writeln(fout);
644  end;
645
646  for i:=1 to nint[k] do
647    nint[i]:=0;
648  end;
649
650  begin
651    write(fout,'');
652    for i:=1 to nint[k] do
653      write(fout,v^[i,k]);
654    writeln(fout);
655  end;
656
657  for i:=1 to nint[k] do
658    v^[i,k]:=0;
659  end;
660
661  begin
662    write(fout,'');
663    for i:=1 to nint[k] do
664      write(fout,prov^[i,k]);
665    writeln(fout);
666  end;
667
668  for i:=1 to nint[k] do
669    prov^[i,k]:=0;
670  end;
671
672  begin
673    write(fout,'');
674    for i:=1 to nint[k] do
675      write(fout,nint[i]);
676    writeln(fout);
677  end;
678
679  for i:=1 to nint[k] do
680    nint[i]:=0;
681  end;
682
683  begin
684    write(fout,'');
685    for i:=1 to nint[k] do
686      write(fout,v^[i,k]);
687    writeln(fout);
688  end;
689
690  for i:=1 to nint[k] do
691    v^[i,k]:=0;
692  end;
693
694  begin
695    write(fout,'');
696    for i:=1 to nint[k] do
697      write(fout,prov^[i,k]);
698    writeln(fout);
699  end;
700
701  for i:=1 to nint[k] do
702    prov^[i,k]:=0;
703  end;
704
705  begin
706    write(fout,'');
707    for i:=1 to nint[k] do
708      write(fout,nint[i]);
709    writeln(fout);
710  end;
711
712  for i:=1 to nint[k] do
713    nint[i]:=0;
714  end;
715
716  begin
717    write(fout,'');
718    for i:=1 to nint[k] do
719      write(fout,v^[i,k]);
720    writeln(fout);
721  end;
722
723  for i:=1 to nint[k] do
724    v^[i,k]:=0;
725  end;
726
727  begin
728    write(fout,'');
729    for i:=1 to nint[k] do
730      write(fout,prov^[i,k]);
731    writeln(fout);
732  end;
733
734  for i:=1 to nint[k] do
735    prov^[i,k]:=0;
736  end;
737
738  begin
739    write(fout,'');
740    for i:=1 to nint[k] do
741      write(fout,nint[i]);
742    writeln(fout);
743  end;
744
745  for i:=1 to nint[k] do
746    nint[i]:=0;
747  end;
748
749  begin
750    write(fout,'');
751    for i:=1 to nint[k] do
752      write(fout,v^[i,k]);
753    writeln(fout);
754  end;
755
756  for i:=1 to nint[k] do
757    v^[i,k]:=0;
758  end;
759
760  begin
761    write(fout,'');
762    for i:=1 to nint[k] do
763      write(fout,prov^[i,k]);
764    writeln(fout);
765  end;
766
767  for i:=1 to nint[k] do
768    prov^[i,k]:=0;
769  end;
770
771  begin
772    write(fout,'');
773    for i:=1 to nint[k] do
774      write(fout,nint[i]);
775    writeln(fout);
776  end;
777
778  for i:=1 to nint[k] do
779    nint[i]:=0;
780  end;
781
782  begin
783    write(fout,'');
784    for i:=1 to nint[k] do
785      write(fout,v^[i,k]);
786    writeln(fout);
787  end;
788
789  for i:=1 to nint[k] do
790    v^[i,k]:=0;
791  end;
792
793  begin
794    write(fout,'');
795    for i:=1 to nint[k] do
796      write(fout,prov^[i,k]);
797    writeln(fout);
798  end;
799
800  for i:=1 to nint[k] do
801    prov^[i,k]:=0;
802  end;
803
804  begin
805    write(fout,'');
806    for i:=1 to nint[k] do
807      write(fout,nint[i]);
808    writeln(fout);
809  end;
810
811  for i:=1 to nint[k] do
812    nint[i]:=0;
813  end;
814
815  begin
816    write(fout,'');
817    for i:=1 to nint[k] do
818      write(fout,v^[i,k]);
819    writeln(fout);
820  end;
821
822  for i:=1 to nint[k] do
823    v^[i,k]:=0;
824  end;
825
826  begin
827    write(fout,'');
828    for i:=1 to nint[k] do
829      write(fout,prov^[i,k]);
830    writeln(fout);
831  end;
832
833  for i:=1 to nint[k] do
834    prov^[i,k]:=0;
835  end;
836
837  begin
838    write(fout,'');
839    for i:=1 to nint[k] do
840      write(fout,nint[i]);
841    writeln(fout);
842  end;
843
844  for i:=1 to nint[k] do
845    nint[i]:=0;
846  end;
847
848  begin
849    write(fout,'');
850    for i:=1 to nint[k] do
851      write(fout,v^[i,k]);
852    writeln(fout);
853  end;
854
855  for i:=1 to nint[k] do
856    v^[i,k]:=0;
857  end;
858
859  begin
860    write(fout,'');
861    for i:=1 to nint[k] do
862      write(fout,prov^[i,k]);
863    writeln(fout);
864  end;
865
866  for i:=1 to nint[k] do
867    prov^[i,k]:=0;
868  end;
869
870  begin
871    write(fout,'');
872    for i:=1 to nint[k] do
873      write(fout,nint[i]);
874    writeln(fout);
875  end;
876
877  for i:=1 to nint[k] do
878    nint[i]:=0;
879  end;
880
881  begin
882    write(fout,'');
883    for i:=1 to nint[k] do
884      write(fout,v^[i,k]);
885    writeln(fout);
886  end;
887
888  for i:=1 to nint[k] do
889    v^[i,k]:=0;
890  end;
891
892  begin
893    write(fout,'');
894    for i:=1 to nint[k] do
895      write(fout,prov^[i,k]);
896    writeln(fout);
897  end;
898
899  for i:=1 to nint[k] do
900    prov^[i,k]:=0;
901  end;
902
903  begin
904    write(fout,'');
905    for i:=1 to nint[k] do
906      write(fout,nint[i]);
907    writeln(fout);
908  end;
909
910  for i:=1 to nint[k] do
911    nint[i]:=0;
912  end;
913
914  begin
915    write(fout,'');
916    for i:=1 to nint[k] do
917      write(fout,v^[i,k]);
918    writeln(fout);
919  end;
920
921  for i:=1 to nint[k] do
922    v^[i,k]:=0;
923  end;
924
925  begin
926    write(fout,'');
927    for i:=1 to nint[k] do
928      write(fout,prov^[i,k]);
929    writeln(fout);
930  end;
931
932  for i:=1 to nint[k] do
933    prov^[i,k]:=0;
934  end;
935
936  begin
937    write(fout,'');
938    for i:=1 to nint[k] do
939      write(fout,nint[i]);
940    writeln(fout);
941  end;
942
943  for i:=1 to nint[k] do
944    nint[i]:=0;
945  end;
946
947  begin
948    write(fout,'');
949    for i:=1 to nint[k] do
950      write(fout,v^[i,k]);
951    writeln(fout);
952  end;
953
954  for i:=1 to nint[k] do
955    v^[i,k]:=0;
956  end;
957
958  begin
959    write(fout,'');
960    for i:=1 to nint[k] do
961      write(fout,prov^[i,k]);
962    writeln(fout);
963  end;
964
965  for i:=1 to nint[k] do
966    prov^[i,k]:=0;
967  end;
968
969  begin
970    write(fout,'');
971    for i:=1 to nint[k] do
972      write(fout,nint[i]);
973    writeln(fout);
974  end;
975
976  for i:=1 to nint[k] do
977    nint[i]:=0;
978  end;
979
980  begin
981    write(fout,'');
982    for i:=1 to nint[k] do
983      write(fout,v^[i,k]);
984    writeln(fout);
985  end;
986
987  for i:=1 to nint[k] do
988    v^[i,k]:=0;
989  end;
990
991  begin
992    write(fout,'');
993    for i:=1 to nint[k] do
994      write(fout,prov^[i,k]);
995    writeln(fout);
996  end;
997
998  for i:=1 to nint[k] do
999    prov^[i,k]:=0;
1000 end;

```

```

48           prov^(i+1) div 2,k]:=i+1;
49           end;
50       end;
51   end;
52
53   nint[k]:=(nint[k-1]+1) div 2;
54   end;
55
56 pmax:=v^(1,k);
57
58 if (prov^(1,k)=1) then pmax2:=v^(2,k-1)
59 else pmax2:=v^(1,k-1);
60
61 pant:=prov^(1,k);
62 k:=k-1;
63 p:=prov^(pant,k);
64 k:=k-1;
65
66 while (k>=1) do
67 begin
68   if (p mod 2=1) and (p+1<=nint[k]) then
69   begin
70     if (compara(pmax2,v^(p+1,k))=-1) then
71     begin
72       pmax2:=v^(p+1,k);
73     end;
74   end
75   else
76   begin
77     if (compara(pmax2,v^(p-1,k))=-1) then
78     begin
79       pmax2:=v^(p-1,k);
80     end;
81   end;
82
83   pant:=p;
84   p:=prov^(p,k);
85   k:=k-1;
86 end;
87
88 assign(fout,'compar.out');
89 rewrite(fout);
90 writeln(fout,pmax,' ',pmax2);
91 close(fout);
92
93 end.
```

Listing 37.1.2: COMPAR.pas

```

1 unit compar;
2
3 interface
4
5 const my_nmax=1000;
6
7 var my_a:array [1..my_nmax+20] of integer;
8   my_apelat,my_n,my_napel,my_apel:integer;
9
10 function getn:integer;
11 function compara(i,j:integer):integer;
12
13 implementation
14
15 function getn:integer;
16 var f:text;
17   i,log,tmp:integer;
18 begin
19   my_apelat:=1;
20   assign(f,'compar.in');reset(f);
21   readln(f,my_n);
22   for i:=1 to my_n do
23     read(f,my_a[i]);
24   close(f);
25   my_apel:=0;
26   tmp:=1;
```

```

27      log:=0;
28      while tmp<my_n do begin
29          tmp:=tmp*2;
30          log:=log+1;
31          end;
32          my_napel:=my_n+log-2;
33          getn:=my_n;
34      end;
35
36 function compara(i,j:integer):integer;
37 var f:text;
38 begin
39     if my_apelat=0 then begin
40         writeln('Nu a fost apelata functia GetN!');
41         halt;
42         end;
43     my_apel:=my_apel+1;
44     if my_apel>my_napel then begin
45         writeln('Depasire a numarului de apeluri permis!');
46         halt;
47         end;
48     if (i<1) or (i>my_n) or (j<1) or (j>my_n) then begin
49         writeln('Apel ilegal!');
50         halt;
51         end;
52     if my_a[i]>=my_a[j] then compara:=1
53     else compara:=-1;
54 end;
55
56 end.

```

37.1.3 *Rezolvare detaliată

37.2 Relee

suf lucrări Stelian Ciurea, Sibiu

Fie date altitudinile a N puncte, situate în linie dreaptă de-a lungul axei Ox , astfel încât ele corespund unor abscise naturale consecutive. Primul punct are abscisa 1. Din acest punct trebuie trimisă o rază laser în ultimul punct (cel de abscisă N). Raza se propagă doar în linie dreaptă. Pentru a "ocoli" punctele având altitudini care împiedică trecerea razei, în anumite puncte se monteză relee care schimbă unghiul sub care se propagă raza, cu scopul ca ea să poată trece de vârfurile care se află în drumul ei. Releele se vor monta în oricare dintre punctele date, mai puțin în primul punct, de unde raza poate porni sub orice unghi și în ultimul punct unde raza poate fi recepționată sub orice unghi.

S-a observat că dacă în anumite puncte releul se montează pe un pilon, numărul releelor necesare se poate micșora. Toți pilonii care se vor monta au aceeași înălțime dată H .

Cerință

Determinați numărul minim de relee pentru ca raza să ajungă din punctul inițial în cel final, precum și punctele în care acestea se vor monta. În cazul în care există mai multe soluții cu același număr minim de relee, se va alege cea cu număr minim de piloni.

Date de intrare

Fișier de intrare: RELEE.IN

Linia 1: $N \ H$ două numere naturale nenule, reprezentând numărul punctelor (N), respectiv înălțimea pilonilor (H);

Linia 2: $A_1 A_2 \dots A_N \ N$ numere întregi, separate prin câte un spațiu, reprezentând altitudinile (înălțimile) punctelor.

Date de ieșire

Fișier de ieșire: RELEE.OUT

Linia 1: NR_{relee} număr natural nenul, reprezentând numărul releeelor care se vor monta, fără să fie înălțate pe piloni;

Linia 2: NR_{piloni} număr natural nenul, reprezentând numărul pilonilor care se vor monta;

Linia 3: $C_1C_2\dots C_{NR_{relee}}$ numere natural nenule, reprezentând numărul de ordine al punctelor unde se vor monta relee, fără să fie înălțate pe piloni;

Linia 4: $D_1D_2\dots D_{NR_{piloni}}$ numere natural nenule, reprezentând numărul de ordine al punctelor unde releele se vor monta pe piloni.

Restricții

$$1 \leq N \leq 200$$

$$1 \leq H \leq 500$$

$$1 \leq A_i \leq 2500, i = 1, 2, \dots, N$$

dacă trei vârfuri sunt coliniare, atunci pe cel din mijloc nu trebuie amplasat un releu.

Exemplu

RELEE.IN	RELEE.OUT
9 2	1
3 2 6 6 4 3 5 3 2	1
	7
	4

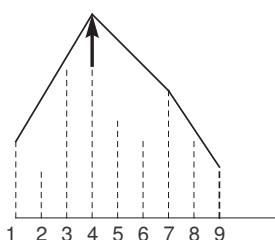


Figura 37.1: Relee

Liniile punctate reprezintă altitudinea punctelor date. Releu fără pilon se va monta în punctul 7, iar pe pilon în punctul 4.

Timp maxim de execuție/test: 1 secundă

37.2.1 *Indicații de rezolvare

37.2.2 Cod sursă

Listing 37.2.1: Relee_MIA.pas

```

1 { Mugurel Ionut Andreica - Bucuresti, ROMANIA }
2
3 Program RELEE;
4
5 type intvect=array[0..1] of integer;
6
7 const infinit=16500;
8
9 var fin,fout:text;
10    rez,min,i,j,k,m,n,p,h,nrp,nrh,npil:longint;
11    a,b,c:longint;
12    hh:array[1..202] of longint;
13    nrel,npil,predp,predh:array[0..200] of ^intvect;
14    rp,rh:array[1..202] of integer;
15    ok:boolean;
16
17 begin
18  assign(fin,'RELEE.IN');
19  reset(fin);
20  readln(fin,N,H);
21
22 for i:=1 to n do
23   read(fin,hh[i]);
24
```

```

25 close(fin);
26
27 for i:=0 to n do
28 begin
29   new(nrel[i]);
30   fillchar(nrel[i]^,sizeof(nrel[i]^),62);
31   new(predp[i]);
32   fillchar(predp[i]^,sizeof(predp[i]^),0);
33   new(predh[i]);
34   fillchar(predh[i]^,sizeof(predh[i]^),0);
35   new(npil[i]);
36   fillchar(npil[i]^,sizeof(npil[i]^),0);
37 end;
38
39 nrel[1]^[0]:=0;
40
41 for i:=2 to n do
42   for k:=0 to 1 do
43     if (i<n) or ((i=n) and (k=0)) then
44     begin
45       min:=infinit;
46       npi:=infinit;
47
48       for j:=i-1 downto 1 do
49         for m:=0 to 1 do
50           if ((j>1) or ((j=1) and (m=0))) then
51             begin
52               if (nrel[j]^*[m]+1<min) or
53                 ((nrel[j]^*[m]+1=min) and (npil[j]^*[m]<npi))
54             then
55               begin
56                 a:=(hh[j]+m*h)-(hh[i]+k*h);
57                 b:=i-j;
58                 c:=j*(hh[i]+k*h)-i*(hh[j]+m*h);
59
60                 ok:=true;
61                 for p:=j+1 to i-1 do
62                   begin
63                     rez:=a*p+b*hh[p]+c;
64                     if (rez>0) then
65                       begin
66                         ok:=false;
67                         break;
68                       end;
69                     end;
70
71                   if (ok) then
72                     begin
73                       min:=nrel[j]^*[m]+1;
74                       npi:=npil[j]^*[m];
75                       predp[i]^*[k]:=j;
76                       predh[i]^*[k]:=m;
77                     end;
78                   end;
79                 end;
80
81                 nrel[i]^*[k]:=min;
82                 npil[i]^*[k]:=npi;
83                 if (k=1) then inc(npil[i]^*[k]);
84               end;
85
86               min:=nrel[n]^*[0]-1; npi:=npil[n]^*[0];
87               i:=predp[n]^*[0]; k:=predh[n]^*[0];
88               nrp:=min-npi; nrh:=npi;
89
90             while (i>1) do
91               begin
92                 if (k=0) then
93                   begin
94                     rp[nrp]:=i;
95                     dec(nrp);
96                   end
97                 else
98                   begin
99                     rh[nrh]:=i;
100                    dec(nrh);

```

```

101      end;
102
103      j:=predp[i]^k;
104      k:=predh[i]^k;
105      i:=j;
106      end;
107
108 assign(fout,'RELEE.OUT');
109 rewrite(fout);
110 writeln(fout,min-npi);
111 writeln(fout,npi);
112
113 nrp:=min-npi; nrh:=npi;
114 if (nrp=0) then
115   writeln(fout)
116 else
117 begin
118   write(fout, rp[1]);
119   for i:=2 to nrp do
120     write(fout, ' ', rp[i]);
121   writeln(fout);
122 end;
123
124 if (nrh=0) then
125   writeln(fout)
126 else
127 begin
128   write(fout, rh[1]);
129   for i:=2 to nrh do
130     write(fout, ' ', rh[i]);
131   writeln(fout);
132 end;
133
134 close(fout);
135 end.

```

37.2.3 *Rezolvare detaliată

37.3 Telecomanda

prof. Emanuela Cerchez și prof. Marinel Șerban, Iași

Cu ocazia olimpiadei, televiziunea locală organizează un nou joc în direct. Organizatorii utilizează un calculator, care generează și afișează pe un monitor două numere de maxim 100 de cifre fiecare (N_1 și N_2).

Fiecare concurent dispune de o telecomandă prevăzută cu un afișaj de o cifră și cu anumite taste, ca în figura alăturată. Telecomanda are și o memorie, în care sunt reținute în ordine cifrele obținute de concurenți.

afisaj				
0	1	2	3	4
5	6	7	8	9
+	-	*	/	#
=				

Figura 37.2: Telecomanda

Cifrele primului număr (N_1) sunt afișate succesiv pe afișajul telecomenții fiecărui concurent, în ordine de la stânga la dreapta. Concurenții trebuie să transforme primul număr, obținând în memoria telecomenții proprii pe cel de al doilea, utilizând tastele pe care le au la dispoziție pe telecomandă. După efectuarea unei operații asupra cifrei curente (cea de pe afișaj), pe afișaj apare automat următoarea cifră din N_1 (dacă mai există).

Efectele apăsării tastelor sunt următoarele:

Taste acționate	Efect
+ urmat de o cifră	Se generează suma dintre cifra de pe afișaj și cifra tastată (operație posibilă doar dacă suma este tot o cifră). Cifra sumă este reținută în memorie.
- urmat de o cifră	Se generează diferența dintre cifra de pe afișaj și cifra tastată (operație posibilă doar dacă se obține tot o cifră). Cifra obținută este reținută în memorie.
* urmat de o cifră	Se reține în memorie valoarea tastei care se acționează după tasta *. Deoarece asupra cifrei curente din N_1 nu se efectuează nici o operație, aceasta nu dispare de pe afișaj.
/	Se șterge cifra curentă din N_1
#	Se sterg din N_1 cifra curentă și toate cifrele care urmează, până la sfârșit.
=	Se copiază în memorie cifra curentă.

Acțiunea se încheie atunci când toate cifrele lui N_1 au fost prelucrate. Am obținut o soluție când în memoria telecomenții se află cifrele numărului N_2 . O soluție este optimă dacă numărul de taste acționate este minim. Câștigătorii jocului sunt acei concurenți care descoperă o soluție optimă.

Cerință

Date fiind N_1 și N_2 , scrieți un program care să determine o soluție optimă de transformare a numărului N_1 în numărul N_2 .

Date de intrare

Fișierul de intrare TELE.IN conține două linii:

N_1

N_2

Date de ieșire

Fișierul de ieșire TELE.OUT conține două linii:

min

$t_1t_2...t_{min}$

unde:

min este un număr natural nenul, reprezentând numărul minim de taste acționate pentru transformarea lui N_1 în N_2 .

$t_1t_2...t_{min}$ este o succesiune de min caractere, care reprezintă tastele acționate; între caractere nu se vor pune separatori.

Exemplu

TELE.IN	TELE.OUT
372	4
78	/=+6

Timp maxim de execuție/test: 1 secundă

37.3.1 *Indicații de rezolvare

37.3.2 Cod sursă

Listing 37.3.1: Tele_MIA.pas

```

1 { Mugurel Ionut Andreica - Bucuresti, ROMANIA }
2
3 Program Telecomanda;
4
5 const infinit=32001;
6
7 type ppivect=array[0..101,0..101] of integer;
```

```

8
9  var fin,fout:text;
10 n1,n2:array[1..110] of integer;
11 nmc:array[0..101,0..101] of integer;
12 predc:array[0..101,0..101] of string[2];
13 pop:array[0..101,0..101] of char;
14 i,j,k,m,n,nc1,nc2,min,mnew:integer;
15 ch:char;
16 ss:string[3];
17 ppi,ppj,popi,popj:^ppivect;
18
19 procedure print(ci,cj:integer);
20 var pi,pj:integer;
21 begin
22 if (ci>0) or (cj>0) then
23 begin
24
25   pi:=ppi^[ci,cj]; pj:=ppj^[ci,cj];
26
27   if (pop[ci,cj]='*') then
28     begin
29       print(popi^[ci,cj],popj^[ci,cj]);
30
31       for i:=popj^[ci,cj]+1 to cj-1 do
32         write(fout,'*',n2[i]);
33
34       write(fout,predc[ci,cj]);
35
36     end
37   else
38     begin
39       print(pi,pj);
40       write(fout,predc[ci,cj]);
41     end;
42   end;
43 end;
44
45 begin
46 assign(fin,'TELE.IN');
47 reset(fin);
48
49 nc1:=0;
50 while (not seekeoln(fin)) do
51 begin
52   inc(nc1);
53   read(fin,ch);
54   val(ch,n1[nc1],j);
55 end;
56 readln(fin);
57
58 nc2:=0;
59 while (not seekeoln(fin)) do
60 begin
61   inc(nc2);
62   read(fin,ch);
63   val(ch,n2[nc2],j);
64 end;
65
66 close(fin);
67
68 fillchar(nmc,sizeof(nmc),0);
69 fillchar(predc,sizeof(predc),0);
70 fillchar(pop,sizeof(pop),0);
71
72 new(ppi); fillchar(ppi^,sizeof(ppi^),0);
73 new(ppj); fillchar(ppj^,sizeof(ppj^),0);
74 new(popi); fillchar(popi^,sizeof(popi^),0);
75 new(popj); fillchar(popj^,sizeof(popj^),0);
76
77 for j:=1 to nc2 do
78 begin
79   nmc[0,j]:=2*j;
80   popi^[0,j]:=0; popj^[0,j]:=0;
81   pop[0,j]:='*';
82   str(n2[j],ss);
83

```

```

84      predc[0,j]:='*' +ss;
85  end;
86
87  for i:=1 to nc1 do
88    for j:=0 to nc2 do
89      begin
90        min:=infinit;
91
92        ss:='';
93        if (n1[i]<=n2[j]) and (j>0) then
94          begin
95            if (min>nmc[i-1,j-1]+2) then
96              begin
97                min:=nmc[i-1,j-1]+2;
98                str(n2[j]-n1[i],ss); ss:='+' +ss;
99                predc[i,j]:=ss;
100               ppi^ [i,j]:=i-1; ppj^ [i,j]:=j-1;
101               pop[i,j]:='a';
102             end;
103
104            ss:='';
105            for k:=j-2 downto 0 do
106              if (nmc[i-1,k]+(j-k-1)*2+2<min) then
107                begin
108                  ss:='';
109                  min:=nmc[i-1,k]+(j-k-1)*2+2;
110                  str(n2[j]-n1[i],ss); ss:='+' +ss;
111                  predc[i,j]:=ss;
112                  ppi^ [i,j]:=i-1; ppj^ [i,j]:=j-1;
113                  pop[i,j]:='';
114                  popi^ [i,j]:=i-1; popj^ [i,j]:=k;
115                end;
116              end;
117
118            ss:='';
119            if (n1[i]>n2[j]) and (j>0) then
120              begin
121
122                if (min>nmc[i-1,j-1]+2) then
123                  begin
124                    ss:='';
125                    min:=nmc[i-1,j-1]+2;
126                    str(n1[i]-n2[j],ss); ss:='-' +ss;
127                    predc[i,j]:=ss;
128                    ppi^ [i,j]:=i-1; ppj^ [i,j]:=j-1;
129                    pop[i,j]:='a';
130                  end;
131
132                ss:='';
133                for k:=j-2 downto 0 do
134                  if (nmc[i-1,k]+(j-k-1)*2+2<min) then
135                    begin
136                      ss:='';
137                      min:=nmc[i-1,k]+(j-k-1)*2+2;
138
139                      if (n2[j]>=n1[i])then
140                        str(n2[j]-n1[i],ss)
141                      else
142                        str(n1[i]-n2[j],ss);
143
144                      ss:='-' +ss;
145                      predc[i,j]:=ss;
146                      ppi^ [i,j]:=i-1; ppj^ [i,j]:=j-1;
147                      pop[i,j]:='*';
148                      popi^ [i,j]:=i-1; popj^ [i,j]:=k;
149                    end;
150                  end;
151
152                ss:='';
153                if (min>nmc[i-1,j]+1) then
154                  begin
155                    min:=nmc[i-1,j]+1;
156                    ss:='/';
157                    predc[i,j]:=ss;
158                    ppi^ [i,j]:=i-1; ppj^ [i,j]:=j;
159                    pop[i,j]:='b';

```

```

160      end;
161
162      ss:='';
163      for k:=j downto 0 do
164          if (nmc[i-1,k]+(j-k)*2+1<min) then
165              begin
166                  min:=nmc[i-1,k]+(j-k)*2+1;
167                  ss:='/';
168                  predc[i,j]:=ss;
169                  ppi^*[i,j]:=i-1; ppj^*[i,j]:=j;
170                  pop[i,j]:='*';
171                  popi^*[i,j]:=i-1; popj^*[i,j]:=k;
172              end;
173
174      ss:='';
175      if (min>nmc[i-1,j-1]+1) and (n1[i]=n2[j]) and (j>0) then
176          begin
177              min:=nmc[i-1,j-1]+1;
178              ss:='=';
179              predc[i,j]:=ss;
180              ppi^*[i,j]:=i-1; ppj^*[i,j]:=j-1;
181              pop[i,j]:='b';
182          end;
183
184      ss:='';
185      if (n1[i]=n2[j]) then
186          for k:=j-2 downto 0 do
187              if (nmc[i-1,k]+(j-k-1)*2+1<min) then
188                  begin
189                      ss:='';
190                      min:=nmc[i-1,k]+(j-k-1)*2+1;
191                      ss:='=';
192                      predc[i,j]:=ss;
193                      ppi^*[i,j]:=i-1; ppj^*[i,j]:=j-1;
194                      pop[i,j]:='*';
195                      popi^*[i,j]:=i-1; popj^*[i,j]:=k;
196                  end;
197
198      nmc[i,j]:=min;
199  end;
200
201 assign(fout,'TELE.OUT');
202 rewrite(fout);
203
204 min:=infinit; k:=0;
205
206 if (nmc[nc1,nc2]<min)then
207     begin
208         min:=nmc[nc1,nc2];
209         k:=nc1;
210     end;
211
212 for i:=1 to nc1-1 do
213     if (nmc[i,nc2]+1<min) then
214         begin
215             min:=nmc[i,nc2]+1;
216             k:=i;
217         end;
218
219 writeln(fout,min);
220
221 print(k,nc2);
222
223 if (k<nc1) then write(fout,'#');
224
225 writeln(fout);
226
227 close(fout);
228 end.
```

37.3.3 *Rezolvare detaliată

37.4 Entries

Dumitru Bogdan, București

Se consideră un graf care inițial este format din P noduri izolate, etichetate de la 1 la P . Se mai consideră N intrări, unde intrare poate însemna:

comandă - o comandă are forma ' $I+J$ ', cu semnificația că în graf se adaugă muchia care unește nodurile I și J (dacă I și J erau deja unite în acel moment, nu se întreprinde nici o acțiune);

întrebare - o întrebare este de forma ' $I?J$ ', adică se întreabă dacă în acel moment I și J sunt în aceeași componentă conexă.

Se pleacă deci de la un graf inițial format din noduri izolate, care pe parcurs se "unifică". Tot pe parcurs sunteți întrebat dacă anumite perechi de noduri sunt sau nu în aceeași componentă conexă.

Din fișierul ENTRIES.IN veți citi de pe prima linie numărul N de intrări. Pe următoarele N linii se găsesc intrările, câte una pe linie. O intrare este codificată prin trei numere separate prin câte un blanc. Primele două numere reprezintă nodurile I și J (numere întregi, cuprinse între 1 și P), iar al treilea este 1 dacă intrarea este o comandă, respectiv 2 dacă intrarea este o întrebare.

La fiecare întrebare, veți scrie pe o linie separată în fișierul ENTRIES.OUT numărul 1 dacă nodurile despre care ati fost întrebat sunt în acel moment în aceeași componentă conexă, respectiv numărul 0 în caz contrar.

Restricții

$1 \leq N \leq 5000$

$1 \leq P \leq 10000000$

Exemplu

ENTRIES.IN	ENTRIES.OUT
9	0
1 2 2	0
1 2 1	1
3 7 2	0
2 3 1	1
1 3 2	0
2 4 2	
1 4 1	
3 4 2	
1 7 2	

Timp maxim de executare/test: 1 secundă

37.4.1 *Indicații de rezolvare

37.4.2 Cod sursă

Listing 37.4.1: Entries_MIA.pas

```

1 { Mugurel Ionut Andreica - Bucuresti, ROMANIA }
2
3 Program d_MAX;
4
5 type longvect=array[1..10010] of longint;
6
7 var fin,fout:text;
8   i,j,pi,pj,k,m,n,nii,lli,lls,last,op,ti,tj:longint;
9   pp:longvect;
10  vaux,tata,nel:^longvect;
11
12 procedure sorti(li,ls:integer);
13 begin
14 if (ls-li=1) then
15 begin
16   if (pp[li]>pp[ls]) then

```

```

17      begin
18          i:=pp[li]; pp[li]:=pp[ls]; pp[ls]:=i;
19      end;
20  end
21 else
22  if (ls-li>1) then
23  begin
24      sorti(li,(li+ls) div 2);
25      sorti((li+ls) div 2+1 ,ls);
26
27      k:=li-1;
28      i:=li;
29      m:=(li+ls) div 2;
30      j:=m+1;
31
32      while (i<=m) and (j<=ls) do
33      begin
34          inc(k);
35          if (pp[i]<pp[j]) then
36          begin
37              vaux^ [k]:=pp[i];
38              inc(i);
39          end
40          else
41          begin
42              vaux^ [k]:=pp[j];
43              inc(j);
44          end;
45      end;
46
47      if (i<=m) then
48      begin
49          for j:=i to m do
50          begin
51              inc(k);
52              vaux^ [k]:=pp[j];
53          end;
54      end
55      else
56      begin
57          for i:=j to ls do
58          begin
59              inc(k);
60              vaux^ [k]:=pp[i];
61          end;
62      end;
63
64      for k:=li to ls do
65          pp[k]:=vaux^ [k];
66  end;
67 end;
68
69
70 procedure cbinar(li,ls:integer;x:longint;var poz:longint);
71 begin
72
73  if (x<pp[(li+ls+1) shr 1]) then
74      cbinar(li,(li+ls) shr 1,x,poz)
75  else
76      if (x>pp[(li+ls+1) shr 1]) then
77          cbinar((li+ls) shr 1+1,ls,x,poz)
78      else
79          poz:=(li+ls+1) shr 1;
80  end;
81
82 begin
83  assign(fin,'ENTRIES.IN');
84  reset(fin);
85  readln(fin,n);
86  for k:=1 to n do
87  begin
88      readln(fin,i,j);
89      pp[2*k-1]:=i; pp[2*k]:=j;
90  end;
91  close(fin);
92

```

```

93 new(vaux);
94 fillchar(vaux^,sizeof(vaux^),0);
95 sorti(1,2*n);
96 move(pp,vaux^,sizeof(pp));
97
98 fillchar(pp,sizeof(pp),0);
99 last:=0; nii:=0;
100 for k:=1 to 2*n do
101 if (vaux^[k]<>last) then
102 begin
103     last:=vaux^[k];
104     inc(nii); pp[nii]:=vaux^[k];
105 end;
106
107 new(tata);
108 fillchar(tata^,sizeof(tata^),0);
109 new(nel);
110 fillchar(nel^,sizeof(nel^),0);
111
112 for i:=1 to nii do
113     nel^ [i]:=1;
114
115 reset(fin);
116 assign(fout,'ENTRIES.OUT');
117 rewrite(fout);
118
119 readln(fin,n);
120 for k:=1 to n do
121 begin
122     begin
123         readln(fin,i,j,op);
124
125         cbinar(1,nii,i,pi);
126         cbinar(1,nii,j,pj);
127     {
128         lli:=1; lls:=nii;
129         while (lli<=lls) do
130             begin
131                 if (i<pp[(lli+lls+1) div 2]) then
132                     lls:=(lli+lls) div 2
133                 else
134                     if (i>pp[(lli+lls+1) div 2]) then
135                         lli:=(lli+lls) div 2+1
136                     else
137                         begin
138                             pi:=(lli+lls+1) div 2;
139                             lli:=1; lls:=0;
140                         end;
141             end;
142
143         lli:=1; lls:=nii;
144         while (lli<=lls) do
145             begin
146                 if (j<pp[(lli+lls+1) div 2]) then
147                     lls:=(lli+lls) div 2
148                 else
149                     if (j>pp[(lli+lls+1) div 2]) then
150                         lli:=(lli+lls) div 2+1
151                     else
152                         begin
153                             pj:=(lli+lls+1) div 2;
154                             lli:=1; lls:=0;
155                         end;
156             end;
157         end;
158     }
159     case op of
160     1:begin
161         ti:=pi;
162         while (tata^[ti]>0) do
163             ti:=tata^[ti];
164
165         tj:=pj;
166         while (tata^[tj]>0) do
167             tj:=tata^[tj];
168     end;
169 end;

```

```

169      if (ti<>tj) then
170      begin
171          if (nel^ [ti]>nel^ [tj]) then
172              begin
173                  tata^ [tj]:=ti;
174              end
175          else
176              if (nel^ [tj]>nel^ [ti]) then
177                  begin
178                      tata^ [ti]:=tj;
179                  end
180              else
181                  begin
182                      tata^ [tj]:=ti;
183                      nel^ [ti]:=nel^ [ti]+1;
184                  end;
185              end;
186          end;
187      end;
188 2:begin
189      ti:=pi;
190      while (tata^ [ti]>0) do
191          ti:=tata^ [ti];
192
193      tj:=pj;
194      while (tata^ [tj]>0) do
195          tj:=tata^ [tj];
196
197      if (ti=tj) then
198          writeln(fout,1)
199      else
200          writeln(fout,0);
201      end;
202  end;
203 end;
204
205 end;
206
207 close(fin);
208 close(fout);
209
210 end.

```

37.4.3 *Rezolvare detaliată

37.5 Robot

prof. Emanuela Cerchez, Iași

Un robot punctiform se poate deplasa, în plan, în linie dreaptă în orice direcție. Robotul se găsește într-o poziție inițială S și trebuie să ajungă într-o poziție finală F , evitând coliziunile cu obstacolele existente în teren. Obstacolele sunt suprafețe poligonale convexe, cu interioarele și frontierele disjuncte. Spunem că robotul a intrat în coliziune cu un obstacol dacă poziția sa devine interioară obstacolului. Prin urmare, dacă robotul se deplasează de-a lungul unui obstacol, nu intră în coliziune cu acesta.

Cerință

Scriți un program care să determine cel mai scurt traseu pe care robotul îl poate urma de la poziția sa inițială S la poziția sa finală F , fără a intra în coliziune cu nici un obstacol.

Traseul va fi precizat prin succesiunea punctelor critice (punctul inițial, punctele în care robotul își schimbă direcția și punctul final). Lungimea traseului este egală cu suma lungimilor segmentelor care îl compun.

Date de intrare:

Fișierul de intrare ROBOT.IN conține:

x_{SYS} - coordonatele poziției inițiale a robotului

x_{FYF} - coordonatele poziției finale a robotului

n - numărul de obstacole

k_1 - numărul de vârfuri ale primului obstacol
 x_1y_1
 x_2y_2
 \dots
 $x_{k_1}y_{k_1}$ - coordonatele vârfurilor primului obstacol
 k_2 - numărul de vârfuri ale celui de-al doilea obstacol
 x_1y_1
 x_2y_2
 \dots
 $x_{k_2}y_{k_2}$ - coordonatele vârfurilor celui de-al doilea obstacol
 \dots
 k_n - numărul de vârfuri ale celui de-al n -lea obstacol
 x_1y_1
 x_2y_2
 \dots
 $x_{k_n}y_{k_n}$ - coordonatele vârfurilor celui de-al n -lea obstacol

Date de ieșire:

Fișierul de ieșire ROBOT.OUT conține traseul robotului codificat ca o succesiune de puncte între care robotul se mișcă în linie dreaptă:

nr - numărul de puncte de pe traseu
 x_{SYS} - coordonatele punctului inițial
 x_1y_1 - coordonatele primului punct critic de pe traseu
 x_2y_2 - coordonatele celui de-al doilea punct critic de pe traseu
 \dots
 x_{FYF} - coordonatele punctului final

Restricții:

n număr natural, $0 \leq n \leq 50$

$k_1 + k_2 + \dots + k_n \leq 200$

x_i, y_i sunt numere reale, $|xi|, |yi| \leq 100000$

punctele S și F nu se află în interiorul nici unui obstacol

coordonatele se vor afișa în fișierul de ieșire cu trei zecimale semnificative.

Exemplu:

ROBOT.IN	ROBOT.OUT
10 5	3
-10 -10	10 5
1	10.5 0
3	-10 -10
0 0	
0 10	
10.5 0	

Observație: Dacă există mai multe trasee de lungime minimă, în fișierul de ieșire se va obține o singură soluție.

Timp maxim de execuție: 1 secundă/test.

37.5.1 *Indicații de rezolvare**37.5.2 Cod sursă**

Listing 37.5.1: Rob2_MIA.pas

```

1 { Mugurel Ionut Andreica - Bucuresti, ROMANIA }
2
3 Program ROBOT;
4
5 type dvect=array[0..210] of double;
6     boolvect=array[0..210] of boolean;

```

```

7      coada=array[0..40900] of byte;
8
9  const eps:double=1e-6;
10
11 var fin,fout:text;
12   i,j,k,m,n,last,vi,vj,act,po:integer;
13   xp,yp:array[0..210] of double;
14   np,st:array[1..52] of integer;
15   xs,ys,xf,yf,xi,yi,xm1,xm2,ym1,ym2,xxm1,xxm2,yyml,yyml2:double;
16   dist:array[0..202] of ^dvect;
17   ppo:array[0..210] of integer;
18   a1,b1,c1,a2,b2,c2,min:double;
19   okyet:boolean;
20   dmin:array[0..202] of double;
21   tata,rec:array[0..202] of integer;
22   sel:array[0..202] of boolean;
23   co:^coada;
24
25 begin
26 assign(fin,'ROBOT.IN');
27 reset(fin);
28
29 readln(fin,xp[0],yp[0]); readln(fin,xf,yf);
30 ppo[0]:=0;
31
32 readln(fin,n);
33
34 st[1]:=1;
35 for k:=1 to n do
36 begin
37   readln(fin,np[k]);
38   for i:=1 to np[k] do
39     begin
40       readln(fin,xp[st[k]+i-1],yp[st[k]+i-1]);
41       ppo[st[k]+i-1]:=k;
42     end;
43
44   st[k+1]:=st[k]+np[k];
45 end;
46
47 close(fin);
48
49 xp[st[k+1]]:=xf; yp[st[k+1]]:=yf;
50 ppo[st[k+1]]:=n+1;
51
52 last:=st[k+1];
53
54 for i:=0 to last do
55 begin
56   new(dist[i]);
57   fillchar(dist[i]^,sizeof(dist[i]^),0);
58 end;
59
60 for i:=0 to last-1 do
61   for j:=i+1 to last do
62     begin
63       dist[i]^ [j]:=sqrt((xp[i]-xp[j])*(xp[i]-xp[j])+
64                           (yp[i]-yp[j])*(yp[i]-yp[j]));
65       dist[j]^ [i]:=dist[i]^ [j];
66     end;
67
68 new(co);
69 fillchar(co^,sizeof(co^),0);
70
71 fillchar(tata,sizeof(tata),0);
72 fillchar(sel,sizeof(sel),false);
73
74 for i:=1 to last do
75   dmin[i]:=1e10;
76
77 dmin[0]:=0; sel[0]:=true; act:=0;
78 tata[0]:=-1;
79 m:=1; po:=1; co^ [1]:=0;
80
81 while (po<=m) and (not sel[last]) do
82   begin

```

```

83
84     i:=co^ [po];
85
86     for j:=1 to last do
87       if (dmin[j]>dmin[i]+dist[i]^ [j]) and (j<>i) then
88       begin
89         if (ppo[i]=ppo[j]) then
90           begin
91             if ((abs(i-j)=1) or (j-i=np[ppo[i]]-1)) then
92               begin
93                 dmin[j]:=dmin[i]+dist[i]^ [j];
94                 tata[j]:=i;
95                 if (not sel[j]) then
96                   begin
97                     inc(m);
98                     co^ [m]:=j;
99                     sel[j]:=true;
100                   end;
101                 end;
102               end
103             else
104               begin
105                 a1:=yp[i]-yp[j];
106                 b1:=xp[j]-xp[i];
107                 c1:=xp[i]*yp[j]-xp[j]*yp[i];
108
109                 if (xp[i]>xp[j]) then
110                   begin
111                     xm1:=xp[j]; xm2:=xp[i];
112                   end
113                 else
114                   begin
115                     xm1:=xp[i]; xm2:=xp[j];
116                   end;
117
118                 if (yp[i]>yp[j]) then
119                   begin
120                     ym1:=yp[j]; ym2:=yp[i];
121                   end
122                 else
123                   begin
124                     ym1:=yp[i]; ym2:=yp[j];
125                   end;
126
127                 okyet:=true;
128
129                 for k:=1 to last-1 do
130                   if (k<>i) and (k<>j) then
131                     begin
132                       vi:=k;
133                       if (ppo[k+1]=ppo[k]) then vj:=k+1
134                       else vj:=st[ppo[k]];
135
136                       if (vj=i) or (vj=j) then continue;
137
138                       if (((xp[vi]<xm1) and (xp[vj]<xm1)) or
139                           ((xp[vi]>xm2) and (xp[vj]>xm2)) or
140                           ((yp[vi]<ym1) and (yp[vj]<ym1)) or
141                           ((yp[vi]>ym2) and (yp[vj]>ym2))) then
142                         continue;
143
144                       a2:=yp[vi]-yp[vj];
145                       b2:=xp[vj]-xp[vi];
146                       c2:=xp[vi]*yp[vj]-xp[vj]*yp[vi];
147
148                       if (abs(a2*b1-a1*b2)>eps) then
149                         begin
150                           xi:=(b2*c1-b1*c2)/(a2*b1-a1*b2);
151                           yi:=(c2*a1-c1*a2)/(a2*b1-a1*b2);
152
153                           if (xp[vi]>xp[vj]) then
154                             begin
155                               xxm1:=xp[vj]; xxm2:=yp[vj];
156                             end
157                           else
158                             begin

```

```

159             xxm1:=xp[vi]; xxm2:=xp[vj];
160         end;
161
162         if (yp[vi]>yp[vj]) then
163             begin
164                 yym1:=yp[vj]; yym2:=yp[vi];
165                 end
166             else
167                 begin
168                     yym1:=yp[vi]; yym2:=yp[vj];
169                 end;
170
171
172         if (
173             ((xi>xm1) or (abs(xi-xm1)<eps)) and
174             ((xi<xm2) or (abs(xi-xm2)<eps)) and
175             ((yi>yml) or (abs(yi-yml)<eps)) and
176             ((yi<ym2) or (abs(yi-ym2)<eps)) and
177             ((xi>xxm1) or (abs(xi-xxm1)<eps)) and
178             ((xi<xxm2) or (abs(xi-xxm2)<eps)) and
179             ((yi>yym1) or (abs(yi-yym1)<eps)) and
180             ((yi<yym2) or (abs(yi-yym2)<eps))
181             )
182         then begin
183             okyet:=false;
184             break;
185         end;
186     end;
187
188
189     if (okyet) then
190         begin
191             tata[j]:=i;
192             dmin[j]:=dmin[i]+dist[i]^ [j];
193             if (not sel[j]) then
194                 begin
195                     inc(m);
196                     co^ [m]:=j;
197                 end;
198             end;
199         end;
200     end;
201
202     inc(po);
203 end;
204
205
206 fillchar(rec,sizeof(rec),0);
207 k:=last; m:=1; rec[1]:=last;
208
209 while (tata[k]>-1) do
210     begin
211         k:=tata[k];
212         inc(m);
213         rec[m]:=k;
214     end;
215
216 assign(fout,'ROBOT.OUT');
217 rewrite(fout);
218 writeln(fout,m);
219 for i:=m downto 1 do
220     writeln(fout,xp[rec[i]]:0:3,' ',yp[rec[i]]:0:3);
221 close(fout);
222 end.

```

Listing 37.5.2: Rob3_MIA.pas

```

1 { Mugurel Ionut Andreica - Bucuresti, ROMANIA }
2
3 Program ROBOT;
4
5 type dvect=array[0..210] of double;
6     boolvect=array[0..210] of boolean;
7     coada=array[0..40900] of byte;
8

```

```

9  const eps:double=1e-6;
10
11 var fin,fout:text;
12   i,j,k,m,n,last,vi,vj,act,po:integer;
13   xp,yp:array[0..210] of double;
14   np,st:array[1..52] of integer;
15   xs,ys,xf,yf,xi,yi,xm1,xm2,ym1,ym2,xxm1,xxm2,yyml,yym2:double;
16   dist:array[0..202] of ^dvect;
17   ppo:array[0..210] of integer;
18   a1,b1,c1,a2,b2,c2,min:double;
19   okyet:boolean;
20   dmin:array[0..202] of double;
21   tata,rec:array[0..202] of integer;
22   sel:array[0..202] of boolean;
23   co:^coada;
24
25 begin
26 assign(fin,'ROBOT.IN');
27 reset(fin);
28
29 readln(fin,xp[0],yp[0]); readln(fin,xf,yf);
30 ppo[0]:=0;
31
32 readln(fin,n);
33
34 st[1]:=1;
35 for k:=1 to n do
36 begin
37   readln(fin,np[k]);
38   for i:=1 to np[k] do
39     begin
40       readln(fin,xp[st[k]+i-1],yp[st[k]+i-1]);
41       ppo[st[k]+i-1]:=k;
42     end;
43
44   st[k+1]:=st[k]+np[k];
45 end;
46
47 close(fin);
48
49 xp[st[k+1]]:=xf; yp[st[k+1]]:=yf;
50 ppo[st[k+1]]:=n+1;
51
52 last:=st[k+1];
53
54 for i:=0 to last do
55 begin
56   new(dist[i]);
57   fillchar(dist[i]^,sizeof(dist[i]^),0);
58 end;
59
60 for i:=0 to last-1 do
61   for j:=i+1 to last do
62     begin
63       dist[i]^ [j]:=sqrt((xp[i]-xp[j])*(xp[i]-xp[j])+
64                           (yp[i]-yp[j])*(yp[i]-yp[j]));
65       dist[j]^ [i]:=dist[i]^ [j];
66     end;
67
68 new(co);
69 fillchar(co^,sizeof(co^),0);
70
71 fillchar(tata,sizeof(tata),0);
72 fillchar(sel,sizeof(sel),false);
73
74 for i:=1 to last do
75   dmin[i]:=1e10;
76
77 dmin[0]:=0; sel[0]:=true; act:=0;
78 tata[0]:=-1;
79 i:=0;
80
81 while (not sel[last]) do
82   begin
83
84   for j:=1 to last do

```

```

85  if (not sel[j]) and (dmin[j]>dmin[i]+dist[i]^j) then
86  begin
87    if (ppo[i]=ppo[j]) then
88      begin
89        if ((abs(i-j)=1) or (j-i=np[ppo[i]]-1)) then
90          begin
91            dmin[j]:=dmin[i]+dist[i]^j;
92            tata[j]:=i;
93          end;
94        end
95      else
96        begin
97          a1:=yp[i]-yp[j];
98          b1:=xp[j]-xp[i];
99          c1:=xp[i]*yp[j]-xp[j]*yp[i];
100
101         if (xp[i]>xp[j]) then
102           begin
103             xm1:=xp[j]; xm2:=xp[i];
104           end
105         else
106           begin
107             xm1:=xp[i]; xm2:=xp[j];
108           end;
109
110         if (yp[i]>yp[j]) then
111           begin
112             ym1:=yp[j]; ym2:=yp[i];
113           end
114         else
115           begin
116             ym1:=yp[i]; ym2:=yp[j];
117           end;
118
119         okyet:=true;
120
121         for k:=1 to last-1 do
122           if (k<>i) and (k<>j) then
123             begin
124               vi:=k;
125               if (ppo[k+1]=ppo[k]) then vj:=k+1
126               else vj:=st[ppo[k]];
127
128               if (vj=i) or (vj=j) then continue;
129
130               if ((xp[vi]<xm1) and (xp[vj]<xm1)) or
131                 ((xp[vi]>xm2) and (xp[vj]>xm2)) or
132                 ((yp[vi]<ym1) and (yp[vj]<ym1)) or
133                 ((yp[vi]>ym2) and (yp[vj]>ym2)) then
134                 continue;
135
136               a2:=yp[vi]-yp[vj];
137               b2:=xp[vj]-xp[vi];
138               c2:=xp[vi]*yp[vj]-xp[vj]*yp[vi];
139
140               if (abs(a2*b1-a1*b2)>eps) then
141                 begin
142                   xi:=(b2*c1-b1*c2)/(a2*b1-a1*b2);
143                   yi:=(c2*a1-c1*a2)/(a2*b1-a1*b2);
144
145                   if (xp[vi]>xp[vj]) then
146                     begin
147                       xxm1:=xp[vj]; xxm2:=yp[vj];
148                     end
149                   else
150                     begin
151                       xxm1:=xp[vi]; xxm2:=xp[vj];
152                     end;
153
154                   if (yp[vi]>yp[vj]) then
155                     begin
156                       yym1:=yp[vj]; yym2:=yp[vi];
157                     end
158                   else
159                     begin
160                       yym1:=yp[vi]; yym2:=yp[vj];

```

```

161           end;
162
163
164     if (
165         ((xi>xm1) or (abs(xi-xm1)<eps)) and
166         ((xi<xm2) or (abs(xi-xm2)<eps)) and
167         ((yi>ym1) or (abs(yi-ym1)<eps)) and
168         ((yi<ym2) or (abs(yi-ym2)<eps)) and
169         ((xi>xxm1) or (abs(xi-xxm1)<eps)) and
170         ((xi<xxm2) or (abs(xi-xxm2)<eps)) and
171         ((yi>yym1) or (abs(yi-yym1)<eps)) and
172         ((yi<yym2) or (abs(yi-yym2)<eps))
173     )
174     then begin
175         okyet:=false;
176         break;
177     end;
178   end;
179 end;
180
181 if (okyet) then
182 begin
183   tata[j]:=i;
184   dmin[j]:=dmin[i]+dist[i]^j;
185 end;
186 end;
187 end;
188
189 min:=1e9;
190 for act:=1 to last do
191   if (not sel[act]) and (min>dmin[act]) then
192   begin
193     i:=act;
194     min:=dmin[act];
195   end;
196
197 sel[i]:=true;
198 end;
199
200
201 fillchar(rec,sizeof(rec),0);
202 k:=last; m:=1; rec[1]:=last;
203
204 while (tata[k]>-1) do
205 begin
206   k:=tata[k];
207   inc(m);
208   rec[m]:=k;
209 end;
210
211 assign(fout,'ROBOT.OUT');
212 rewrite(fout);
213 writeln(fout,m);
214 for i:=m downto 1 do
215   writeln(fout,xp[rec[i]]:0:3,' ',yp[rec[i]]:0:3);
216 close(fout);
217 end.

```

Listing 37.5.3: Robot_MIA.pas

```

1 { Mugurel Ionut Andreica - Bucuresti, ROMANIA }
2
3 Program ROBOT;
4
5 type dvect=array[0..210] of double;
6   boolvect=array[0..210] of boolean;
7
8 const eps:double=1e-6;
9
10 var fin,fout:text;
11   i,j,k,m,n,last,vi,vj,act,po:integer;
12   xp,yp:array[0..210] of double;
13   np,st:array[1..52] of integer;
14   xs,ys,xf,yf,xi,yi,xm1,xm2,ym1,ym2,xxm1,xxm2,yym1,yym2:double;
15   dist:array[0..202] of ^dvect;

```

```

16     ppo:array[0..210] of integer;
17     a1,b1,c1,a2,b2,c2,min:double;
18     okyet:boolean;
19     dmin:array[0..202] of double;
20     tata,rec:array[0..202] of integer;
21     sel:array[0..202] of boolean;
22
23 begin
24 assign(fin,'ROBOT.IN');
25 reset(fin);
26
27 readln(fin,xp[0],yp[0]); readln(fin,xf,yf);
28 ppo[0]:=0;
29
30 readln(fin,n);
31
32 st[1]:=1;
33 for k:=1 to n do
34 begin
35   readln(fin,np[k]);
36   for i:=1 to np[k] do
37     begin
38       readln(fin,xp[st[k]+i-1],yp[st[k]+i-1]);
39       ppo[st[k]+i-1]:=k;
40     end;
41
42   st[k+1]:=st[k]+np[k];
43 end;
44
45 close(fin);
46
47 xp[st[k+1]]:=xf; yp[st[k+1]]:=yf;
48 ppo[st[k+1]]:=n+1;
49
50 last:=st[k+1];
51
52 for i:=0 to last do
53 begin
54   new(dist[i]);
55   fillchar(dist[i]^,sizeof(dist[i]^),0);
56 end;
57
58 for i:=0 to last-1 do
59   for j:=i+1 to last do
60     begin
61       dist[i]^ [j]:=sqrt((xp[i]-xp[j])*(xp[i]-xp[j])+
62                           (yp[i]-yp[j])*(yp[i]-yp[j]));
63       dist[j]^ [i]:=dist[i]^ [j];
64     end;
65
66
67 fillchar(tata,sizeof(tata),0);
68 fillchar(sel,sizeof(sel),false);
69
70 for i:=1 to last do
71   dmin[i]:=1e10;
72
73 dmin[0]:=0; sel[0]:=true; act:=0;
74 tata[0]:=-1;
75 i:=0;
76
77 while (not sel[last]) do
78 begin
79
80   for j:=1 to last do
81     if (not sel[j]) and (dmin[j]>dmin[i]+dist[i]^ [j]) then
82     begin
83       if (ppo[i]=ppo[j]) then
84         begin
85           if ((abs(i-j)=1) or (j-i=np[ppo[i]]-1)) then
86             begin
87               dmin[j]:=dmin[i]+dist[i]^ [j];
88               tata[j]:=i;
89             end;
90         end
91     else

```

```

92      begin
93          a1:=yp[i]-yp[j];
94          b1:=xp[j]-xp[i];
95          c1:=xp[i]*yp[j]-xp[j]*yp[i];
96
97          if (xp[i]>xp[j]) then
98              begin
99                  xm1:=xp[j]; xm2:=xp[i];
100             end
101         else
102             begin
103                 xm1:=xp[i]; xm2:=xp[j];
104             end;
105
106         if (yp[i]>yp[j]) then
107             begin
108                 ym1:=yp[j]; ym2:=yp[i];
109             end
110         else
111             begin
112                 ym1:=yp[i]; ym2:=yp[j];
113             end;
114
115         okyet:=true;
116
117         for k:=1 to last-1 do
118             if (k<>i) and (k<>j) then
119                 begin
120                     vi:=k;
121                     if (ppo[k+1]=ppo[k]) then vj:=k+1
122                     else vj:=st[ppo[k]];
123
124                     if (vj=i) or (vj=j) then continue;
125
126                     if ((xp[vi]<xm1) and (xp[vj]<xm1)) or
127                         ((xp[vi]>xm2) and (xp[vj]>xm2)) or
128                         ((yp[vi]<ym1) and (yp[vj]<ym1)) or
129                         ((yp[vi]>ym2) and (yp[vj]>ym2)) then
130                         continue;
131
132                     a2:=yp[vi]-yp[vj];
133                     b2:=xp[vj]-xp[vi];
134                     c2:=xp[vi]*yp[vj]-xp[vj]*yp[vi];
135
136                     if (abs(a2*b1-a1*b2)>eps) then
137                         begin
138                             xi:=(b2*c1-b1*c2)/(a2*b1-a1*b2);
139                             yi:=(c2*a1-c1*a2)/(a2*b1-a1*b2);
140
141                             if (xp[vi]>xp[vj]) then
142                                 begin
143                                     xxm1:=xp[vj]; xxm2:=xp[vi];
144                                 end
145                             else
146                                 begin
147                                     xxm1:=xp[vi]; xxm2:=xp[vj];
148                                 end;
149
150                             if (yp[vi]>yp[vj]) then
151                                 begin
152                                     yym1:=yp[vj]; yym2:=yp[vi];
153                                 end
154                             else
155                                 begin
156                                     yym1:=yp[vi]; yym2:=yp[vj];
157                                 end;
158
159                             if (
160                                 ((xi>xm1) or (abs(xi-xm1)<eps)) and
161                                 ((xi<xm2) or (abs(xi-xm2)<eps)) and
162                                 ((yi>ym1) or (abs(yi-ym1)<eps)) and
163                                 ((yi<ym2) or (abs(yi-ym2)<eps)) and
164                                 ((xi>xxm1) or (abs(xi-xxm1)<eps)) and
165                                 ((xi<xxm2) or (abs(xi-xxm2)<eps)) and
166                                 ((yi>yym1) or (abs(yi-yym1)<eps)) and
167

```

```

168          ((yi<yy2) or (abs(yi-yy2)<eps))
169          )
170      then begin
171          okyet:=false;
172          break;
173          end;
174      end;
175  end;
176
177  if (okyet) then
178  begin
179      tata[j]:=i;
180      dmin[j]:=dmin[i]+dist[i]^j;
181      end;
182  end;
183 end;
184
185 min:=1e9;
186 for act:=1 to last do
187   if (not sel[act]) and (min>dmin[act]) then
188   begin
189       i:=act;
190       min:=dmin[act];
191   end;
192
193 sel[i]:=true;
194 end;
195
196
197 fillchar(rec,sizeof(rec),0);
198 k:=last; m:=1; rec[1]:=last;
199
200 while (tata[k]>-1) do
201 begin
202     k:=tata[k];
203     inc(m);
204     rec[m]:=k;
205 end;
206
207 assign(fout,'ROBOT.OUT');
208 rewrite(fout);
209 writeln(fout,m);
210 for i:=m downto 1 do
211   writeln(fout,xp[rec[i]]:0:3,' ',yp[rec[i]]:0:3);
212 close(fout);
213 end.
```

37.5.3 *Rezolvare detaliată

37.6 Text mare

Radu Stefan, Brașov

Se dă un sir de caractere, ce reprezintă o frază; între cuvinte nu apar spații de separare. De asemenea se dă un vocabular având cel mult 32000 de cuvinte; fiecare cuvânt este format din cel mult 16 caractere. Fraza poate avea cel mult 32000 caractere, ce sunt litere mici din alfabetul latin. Cuvintele din vocabular nu sunt neapărat diferite și nu apar într-o ordine prestabilită.

Cerință

Despărțiți fraza într-un număr minim de cuvinte; toate aceste cuvinte trebuie să existe în vocabularul dat.

Date de intrare

Fișier de intrare: TEXTMARE.IN

pe prima linie apare fraza care trebuie despărțită în cuvinte, terminată cu un punct;
următoarele linii conțin câte un cuvânt din vocabular;
fișierul se termină cu o linie liberă.

Date de ieșire

Fisier de ieșire: TEXTMARE.OUT

Fisierul este format dintr-o singură linie, pe care apare fraza despărțită în cuvinte, urmată imediat de un punct. Între oricare două cuvinte consecutive va apărea exact câte un blanc.

Restricții și precizări:

cel puțin jumătate din teste vor avea mai puțin de 1000 de caractere în frază și cel mult 1000 de cuvinte în vocabular;

dacă există mai multe soluții, la ieșire va fi produsă una singură;

dacă nu există soluție, în fișierul de ieșire se va scrie doar cifra 0 (zero);

într-o frază un cuvânt poate să apară de mai multe ori, fiecare apariție a sa fiind numărată.

Exemplu

TEXTMARE.IN	TEXTMARE.OUT
acesta este un text.	acea este un text.
text	
acea	
cest	
a	
care	
este	
un	
simplu	

Timp maxim de executare/test: 2 secunde.

37.6.1 *Indicații de rezolvare

37.6.2 Cod sursă

Listing 37.6.1: Textmare_MIA.pas

```

1 { Mugurel Ionut Andreica - Bucuresti, ROMANIA }
2
3 Program DitamaiTextul;
4
5 type ptrie=^trie;
6   trie=record
7     v:array['a'..'z'] of ptrie;
8     finish:boolean;
9   end;
10
11 predvect=array[0..32002] of integer;
12 cuvvect=array[1..1100] of string[16];
13 cuvlung=array[1..1100] of byte;
14
15 var fin,fout:text;
16 s:string[40];
17 p,ls,i,j,k,m,n,n2,maxl,ncuv,lung:longint;
18 cuv:array['a'..'z'] of ptrie;
19 pp:ptrie;
20 nmin:predvect;
21 pred:^predvect;
22 lastsec,actsec:string[20];
23 oklast,okact:array[0..20] of boolean;
24 ch:char;
25 ok:boolean;
26 mult:set of char;
27 cset:^cuvvect;
28 lng:^cuvlung;
29
30 procedure nosol;
31 begin
32   assign(fout,'TEXTMARE.OUT');
33   rewrite(fout);
34   writeln(fout,0);
35   close(fout);

```

```

36  halt(0);
37  end;
38
39  procedure bullshit;
40  begin
41  new(cset);
42  fillchar(cset^, sizeof(cset^), 0);
43  new(lng);
44  fillchar(lng^, sizeof(lng^), 0);
45
46  reset(fin);
47  readln(fin);
48  maxl:=0;
49  for i:=1 to ncuv do
50    begin
51      readln(fin,cset^[i]);
52      lng^[i]:=length(cset^[i]);
53
54      if (lng^[i]>maxl) then
55          maxl:=lng^[i];
56    end;
57  close(fin);
58
59  reset(fin);
60
61  fillchar(nmin,sizeof(nmin),127);
62  nmin[0]:=0;
63  new(pred);
64  fillchar(pred^, sizeof(pred^), 0);
65
66  fillchar(actsec,sizeof(actsec),0);
67  fillchar(oklast,sizeof(oklast),0);
68
69  lastsec:=' ';
70  oklast[1]:=true;
71
72
73  ch:='e';
74  n:=0;n2:=-1;
75
76  while (ch<>'.')
77    begin
78      fillchar(okact,sizeof(okact),false);
79      m:=0;
80      while (m<maxl) and (ch<>'.')
81        begin
82          read(fin,ch);
83          if (ch<>'.')
84            begin
85              actsec:=actsec+ch; m:=m+1;
86            end;
87        end;
88
89        if (m=0) then
90          begin
91              if (oklast[ord(lastsec[0])]) then break
92              else nosol;
93          end;
94
95        s:=lastsec+actsec; ls:=length(s);
96        j:=ord(lastsec[0]);
97
98        for k:=2 to ord(s[0])
99          begin
100            if ((k-1<=j+1) and (oklast[k-1])) or
101                ((k-1>j) and (okact[k-j-1])) then
102              begin
103                for p:=1 to ncuv do
104                  if (k+lng^ [p]-1<=ls) and (k+lng^ [p]-1>j) then
105                    begin
106                      ok:=true;
107                      for i:=1 to lng^ [p] do
108                        if (s[k+i-1]<>cset^ [p][i]) then
109                          begin
110                            ok:=false;
111                            break;

```

```

112           end;
113
114           if (ok) and (nmin[n2+k-1]+1<nmin[n2+k+lng^[p]-1]) then
115               begin
116                   nmin[n2+k+lng^[p]-1]:=nmin[n2+k-1]+1;
117                   pred^[n2+k+lng^[p]-1]:=n2+k-1;
118                   okact[k+lng^[p]-1-j]:=true;
119               end;
120           end;
121       end;
122   end;
123
124   lastsec:=actsec; actsec:='';
125   move(okact,oklast,sizeof(oklast));
126
127   j:=0;
128   for i:=1 to m do
129       if (okact[i]) then
130           begin
131               j:=1;
132               break;
133           end;
134
135   if (j=0) then nosol;
136   n2:=n;
137   n:=n+m;
138   end;
139
140 close(fin);
141
142 assign(fout,'TEXTMARE.OUT');
143 rewrite(fout);
144
145 fillchar(nmin,sizeof(nmin),0);
146 i:=n;
147 while (pred^[i]>0) do
148     begin
149         i:=pred^[i];
150         nmin[i]:=1;
151     end;
152
153 reset(fin);
154 ch:='a'; i:=0;
155 while (ch<>'.' ) do
156     begin
157         read(fin,ch);
158         write(fout,ch);
159         inc(i);
160         if (nmin[i]=1) then
161             write(fout,' ');
162     end;
163 close(fin);
164
165 writeln(fout);
166 close(fout);
167 halt(0);
168 end;
169
170 begin
171 assign(fin,'TEXTMARE.IN');
172 reset(fin);
173
174 mult:=[];
175 ch:='a';
176 lung:=0;
177 while (ch<>'.' ) do
178     begin
179         read(fin,ch); inc(lung);
180         mult:=mult+[ch];
181     end;
182
183 ncuv:=0;
184 while (not seekeof(fin)) do
185     begin
186         inc(ncuv);
187         readln(fin);

```

```

188   end;
189   close(fin);
190
191   reset(fin);
192   readln(fin);
193   maxl:=0;
194   while (not seekeof(fin)) do
195     begin
196       readln(fin,s);
197
198       if (s=='') then break;
199
200       if (s[0]=#1) then
201         begin
202           if (maxl=0) then maxl:=1;
203
204           if not( s[1] in mult) then continue;
205
206           if (cuv[s[1]]=nil) then
207             begin
208               new(cuv[s[1]]);
209               fillchar(cuv[s[1]]^,sizeof(cuv[s[1]]^),0);
210             end;
211             cuv[s[1]]^.finish:=true;
212           end
213         else
214           begin
215             k:=length(s);
216             if (k>maxl) then maxl:=k;
217
218             ok:=true;
219             for j:=1 to k do
220               if not (s[j] in mult) then
221                 begin
222                   ok:=false;
223                   break;
224                 end;
225
226             if (not ok) then continue;
227
228             if (cuv[s[k]]=nil) then
229               begin
230                 new(cuv[s[k]]);
231                 fillchar(cuv[s[k]]^,sizeof(cuv[s[k]]^),0);
232               end;
233               pp:=cuv[s[k]];
234
235             for j:=k-1 downto 1 do
236               begin
237                 if (pp^.v[s[j]]=nil) then
238                   begin
239                     new(pp^.v[s[j]]);
240                     fillchar(pp^.v[s[j]]^,sizeof(pp^.v[s[j]]^),0);
241                   end;
242                   pp:=pp^.v[s[j]];
243                 end;
244
245             pp^.finish:=true;
246           end;
247
248           if (maxavail<84100) and (ncuv<=1100) and (lung<=1100) then break;
249           else if (maxavail<65100) then break;
250         end;
251
252   close(fin);
253
254
255
256   if (ncuv<=1100) and (lung<=1100) and (maxavail<82100) then
257     bullshit;
258
259   reset(fin);
260
261   fillchar(nmin,sizeof(nmin),127);
262   nmin[0]:=0;
263   new(pred);

```

```

264 fillchar(pred^,sizeof(pred^),0);
265 fillchar(actsec,sizeof(actsec),0);
266 fillchar(oklast,sizeof(oklast),0);
267
268 lastsec:=' ';
269 oklast[1]:=true;
270
271
272 ch:='e';
273 n:=0;n2:=-1;
274
275 while (ch<>'.')
276 begin
277   fillchar(okact,sizeof(okact),false);
278   m:=0;
279   while (m<maxl) and (ch<>'.')
280   begin
281     read(fin,ch);
282     if (ch<>'.')
283       begin
284         actsec:=actsec+ch; m:=m+1;
285       end;
286     end;
287   if (m=0) then
288     begin
289       if (oklast[ord(lastsec[0])]) then break
290       else nosol;
291     end;
292   endsec:=actsec;
293   j:=ord(lastsec[0]);
294
295   for k:=2 to ord(s[0])
296   do
297     if (k>j) then
298       begin
299         i:=k; pp:=cuv[s[i]];
300         while (pp<>nil) and (i>1)
301         begin
302           if (pp^.finish) then
303             begin
304               if (i<=j+1) and (oklast[i-1])
305                 begin
306                   okact[k-j]:=true;
307                   if (nmin[n2+i-1]+1<nmin[n2+k]) then
308                     begin
309                       nmin[n2+k]:=nmin[n2+i-1]+1;
310                       pred^[n2+k]:=n2+i-1;
311                     end;
312                   end;
313                 end;
314               end;
315             if (i>j+1) and (okact[i-j-1])
316               begin
317                 okact[k-j]:=true;
318                 if (nmin[n2+i-1]+1<nmin[n2+k]) then
319                   begin
320                     nmin[n2+k]:=nmin[n2+i-1]+1;
321                     pred^[n2+k]:=n2+i-1;
322                   end;
323                 end;
324               end;
325             end;
326             dec(i);
327             pp:=pp^.v[s[i]];
328           end;
329         end;
330       lastsec:=actsec; actsec:='';
331       move(okact,oklast,sizeof(oklast));
332
333       j:=0;
334       for i:=1 to m do
335         if (okact[i]) then
336           begin
337             j:=1;
338             break;
339           end;

```

```
340         end;
341
342     if (j=0) then nosol;
343     n2:=n;
344     n:=n+m;
345   end;
346
347 close(fin);
348
349 assign(fout,'TEXTMARE.OUT');
350 rewrite(fout);
351
352 fillchar(nmin,sizeof(nmin),0);
353 i:=n;
354 while (pred^[i]>0) do
355   begin
356     i:=pred^[i];
357     nmin[i]:=1;
358   end;
359
360 reset(fin);
361 ch:='a'; i:=0;
362 while (ch<>'.' ) do
363   begin
364     read(fin,ch);
365     write(fout,ch);
366     inc(i);
367     if (nmin[i]=1) then
368       write(fout,' ');
369   end;
370 close(fin);
371
372 writeln(fout);
373 close(fout);
374 end.
```

37.6.3 *Rezolvare detaliată

Capitolul 38

ONI 2000

38.1 Arbore

Marius Vlad, student, Universitatea Politehnica, București

Se dă un arbore format din n noduri în care fiecare nod are asociat un număr natural nenul.

Cerință:

Să se selecteze din arborele inițial un subgraf conex pentru care suma valorilor asociate nodurilor este egală cu un număr natural k dat. Un subgraf este graful din care se elimină noduri împreună cu muchiile aferente.

Date de intrare:

Nodurile arborelui sunt numerotate de la 1 la n , rădăcina fiind nodul numerotat cu 1.

Fisierul de intrare ARBORE.IN, are următoarea structură:

pe prima linie sunt scrise numerele n și k ;

pe cea de-a doua linie este scrisă valoarea asociată nodului 1 (rădăcina arborelui);

pe cea de-a treia linie sunt scrise două numere, primul reprezentând nodul părinte al nodului 2, iar al doilea reprezentând valoarea asociată nodului 2;

pe cea de-a patra linie sunt scrise două numere, primul reprezentând nodul părinte al nodului 3, iar al doilea reprezentând valoarea asociată nodului 3;

...

pe linia $n + 1$ sunt scrise două numere, primul reprezentând nodul părinte al nodului n , iar al doilea reprezentând valoarea asociată nodului n .

Date de ieșire:

Ieșirea se va face în fisierul ARBORE.OUT:

dacă nu există soluție se va afișa doar numărul -1 pe prima linie;

dacă există soluție se vor afișa nodurile ce formează un subgraf conex care respectă cerința problemei (câte un singur nod pe fiecare linie).

Restricții:

$1 \leq n \leq 100$

$1 \leq k \leq 1000$

$1 \leq \text{valoarea asociată unui nod} \leq 1000$

Exemplu:

ARBORE.IN	ARBORE.OUT poate conține:
10 29	2
30	4
1 7	5
1 22	6
2 5	9
2 10	10
4 1	
4 2	
5 25	
5 2	
5 4	

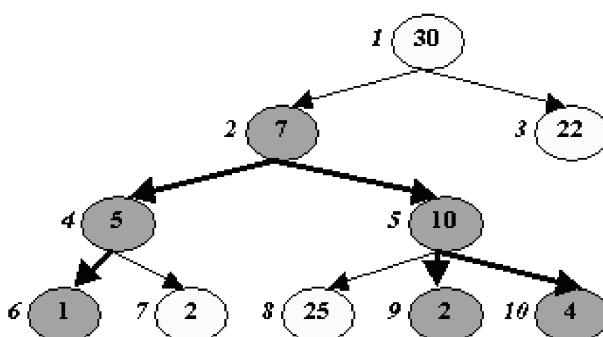


Figura 38.1: Arbore

Timp maxim de execuție pe test: 3 secunde

38.1.1 Indicații de rezolvare

Pentru fiecare nod, se ține un vector de la 1 la 1000, $V[i, p]$ ($1 \leq p \leq 1000$) = TRUE, dacă se poate obține suma p folosind valoarea nodului i , și unele dintre valorile din subarborele determinat de nodul i .

Pentru frunze, $V[i, Val[i]]$ = TRUE, restul fiind FALSE.

Pentru a mări viteza de căutare, se va ține și o listă cu toate sumele obținute, fiecare din ele fiind marcate în tabel (pentru a nu introduce în listă de 2 ori aceeași sumă).

Pentru fiecare j dintre fiii lui i

Pentru C_j de la 1 la Numărul de elemente din lista lui j

Pentru C_i de la 1 la Numarul de elemente din lista lui i ,

Dacă $(List[j, C_j] + List[i, C_i])$ nu a fost marcat în tabelul lui i) se marchează și se adaugă la sfârșitul listei lui i (dar nu se actualizează încă numărul de elemente ale lui i).

- actualizează numărul de elemente ale listei lui i .

38.1.2 Cod sursă

Listing 38.1.1: Arbore_comisie.cpp

```

1 #include <stdio.h>
2 // #include <alloc.h>
3 #include<stdlib.h>
4 #include <process.h>
5
6 using namespace std;
7
8 #define NODS 100UL
9 #define KMAX 1001UL
10#define NIL 255
11
  
```

```

12  typedef struct info_struct
13  {
14      int left;
15      unsigned char down;
16  } info_t;
17
18 FILE *fin,*fout;
19 int n,k;
20
21 info_t *fx[NODS];
22 int tata[NODS];
23 int val[NODS];
24
25 void end()
26 {
27     fclose(fin);
28     fclose(fout);
29     for (int i=0;i<NODS;i++)
30         free(fx[i]);
31     exit(0);
32 }
33
34 info_t *x(int nod,int valoare)
35 {
36     if (nod>=NODS)
37         printf("EROARE!\n");
38     return &fx[nod][valoare];
39 }
40
41 void solution(int nod,int valoare)
42 {
43     int v=valoare;
44     fprintf(fout,"%d\n",nod+1);
45     while (x(nod,v)->down!=NIL)
46     {
47         solution(x(nod,v)->down,v-x(nod,v)->left);
48         v=x(nod,v)->left;
49     }
50 }
51
52 void postord(int q)
53 {
54     int i,j1,j2;
55
56     for (i=0;i<=k;i++)
57     {
58         x(q,i)->left=-1;
59         x(q,i)->down=NIL;
60     }
61     x(q,val[q])->left=0;
62     x(q,val[q])->down=NIL;
63     if (val[q]==k)
64     {
65         solution(q,k);
66         end();
67     }
68     for (i=0;i<n;i++)
69     if (tata[i]==q)
70     {
71         postord(i);
72         for (j1=0;j1<=k;j1++)
73             if (x(q,j1)->left>=0)
74                 for (j2=0;j1+j2<=k;j2++)
75                     if (x(i,j2)->left>=0&&
76                         x(q,j1)->down!=i&&
77                         x(q,j1+j2)->left<=0)
78                 {
79                     x(q,j1+j2)->left=j1;
80                     x(q,j1+j2)->down=i;
81                     if (j1+j2==k)
82                     {
83                         solution(q,k);
84                         end();
85                     }
86                 }
87             }

```

```

88 }
89
90 int main()
91 {
92     unsigned long i,j;
93
94     for (i=0;i<NODS;i++)
95         if ((fx[i]=(info_t *)malloc(KMAX*sizeof(info_t)))==NULL)
96         {
97             for (j=0;j<i;j++)
98                 free(fx[j]);
99             printf("Memorie insuficienta! Rulati din DOS!\n");
100            return -1;
101        }
102    fin=fopen("arbore.in", "r");
103    fout=fopen("arbore.out", "w");
104    fscanf(fin,"%d %d",&n,&k);
105    tata[0]=NIL;
106    fscanf(fin,"%d", &val[0]);
107    for (i=1;i<n;i++)
108    {
109        fscanf(fin,"%d %d",&tata[i],&val[i]);
110        tata[i]--;
111    }
112    postord(0);
113    fprintf(fout, "-1\n");
114    end();
115    return 0;
116 }
```

Listing 38.1.2: Arbore_MIA.pas

```

1 { Andreica Mugurel Ionut }
2 { !!! Sursa de 100 de puncte !!! }
3
4 Program arbore;
5 type linie=array[1..100] of boolean;
6     list=array[1..100] of integer;
7     ant=record
8         nod:byte;
9         {
10         vallst:integer;
11         }
12     end;
13     lst=array[1..100] of ant;
14 var f:text;
15     i,j,k,m,n,p,q:integer;
16     v,tata:array[0..100] of integer;
17     nivel:array[0..100] of integer;
18     ok:array[0..1000] of ^linie;
19     obt:array[0..1000] of ^list;
20     nel:array[1..100] of integer;
21     asoc,aux,asocaux:array[1..100] of integer;
22     par:array[1..1000] of ^lst;
23     suma:integer;
24
25 procedure print(nod:byte;cant:integer);
26 var found:integer;
27 begin
28 if (par[cant]^ [nod].nod=0) then begin
29 writeln(f,nod);
30 end
31 else begin
32 for found:=1 to cant do
33 if (ok[found]^ [nod]) and
34 (ok[cant-found]^ [par[cant]^ [nod].nod])
35 then break;
36
37
38 print(nod,{par[cant]^ [nod].vallst}found);
39 print(par[cant]^ [nod].nod,cant-{par[cant]^ [nod].vallst}found);
40 end;
41
42 end;
```

```

44
45 procedure parc(nod,tat:integer);
46 var s:integer;
47 begin
48 nivel[nod]:=nivel[tat]+1;
49 asoc[nod]:=nod;
50 for s:=1 to n do
51 if tata[s]=nod then parc(s,nod);
52 end;
53
54 procedure sorti(li,ls:integer);
55 begin
56 if (ls-li=1) then begin
57 { interschimba }
58 if (nivel[li]<nivel[ls]) then begin
59 m:=nivel[li];
60 nivel[li]:=nivel[ls];
61 nivel[ls]:=m;
62 m:=asoc[li];
63 asoc[li]:=asoc[ls];
64 asoc[ls]:=m;
65 end;
66 end
67 else
68 if (ls-li>1) then begin
69 sorti(li,(li+ls) div 2);
70 sorti((li+ls) div 2 +1,ls);
71 p:=li-1;
72 i:=li;
73 m:=(li+ls) div 2;
74 j:=m+1;
75
76 while (i<=m) and (j<=ls) do begin
77 inc(p);
78 if nivel[i]>nivel[j] then begin
79 aux[p]:=nivel[i];
80 asocaux[p]:=asoc[i];
81 inc(i);
82 end;
83 else begin
84 aux[p]:=nivel[j];
85 asocaux[p]:=asoc[j];
86 inc(j);
87 end;
88 end;
89 if (i<=m) then begin
90 for j:=i to m do begin
91 inc(p);
92 aux[p]:=nivel[j];
93 asocaux[p]:=asoc[j];
94 end;
95 end;
96 else begin
97 for i:=j to ls do begin
98 inc(p);
99 aux[p]:=nivel[i];
100 asocaux[p]:=asoc[i];
101 end;
102 end;
103 for i:=li to ls do begin
104 nivel[i]:=aux[i];
105 asoc[i]:=asocaux[i];
106 end;
107 end;
108 end;
109
110 begin
111 assign(f,'arbore.in');
112 reset(f);
113 readln(f,n,k);
114 nival[0]:=0;
115 tata[1]:=0;
116 nel[1]:=0;
117 readln(f,v[1]);
118 for i:=2 to n do begin

```

```

120  readln(f,tata[i],v[i]);
121  nel[i]:=0;
122          end;
123  close(f);
124
125  parc(1,0);
126  for i:=0 to k do begin
127
128  new(ok[i]);
129  fillchar(ok[i]^,sizeof(ok[i]^),false);
130
131  new(obt[i]);
132  fillchar(obt[i]^,sizeof(obt[i]^),0);
133  new(par[i]);
134  fillchar(par[i]^,sizeof(par[i]^),0);
135          end;
136
137  assign(f,'arbore.out');
138  rewrite(f);
139
140  sorti(1,n);
141  for i:=1 to n do begin
142  ok[v[asoc[i]]]^*[asoc[i]]:=true;
143  nel[asoc[i]]:=1;
144  obt[1]^*[asoc[i]]:=v[asoc[i]];
145  par[v[asoc[i]]]^*[asoc[i]].nod:=0;
146
147
148  if v[asoc[i]]=k then begin
149  writeln(f,asoc[i]);
150  close(f);
151  halt;
152          end;
153
154  for j:=1 to i-1 do
155  if tata[asoc[j]]=asoc[i] then begin
156  m:=nel[asoc[i]];
157  for p:=1 to nel[asoc[j]] do
158  for q:=1 to m do begin
159  if (not ok[obt[p]^*[asoc[j]]+obt[q]^*[asoc[i]]]^*[asoc[i]])
160  and (obt[p]^*[asoc[j]]+obt[q]^*[asoc[i]]<=k)
161  then begin
162  inc(nel[asoc[i]]);
163  obt[nel[asoc[i]]]^*[asoc[i]]:=obt[p]^*[asoc[j]]+obt[q]^*[asoc[i]];
164  ok[obt[nel[asoc[i]]]^*[asoc[i]]]^*[asoc[i]]:=true;
165  suma:=obt[nel[asoc[i]]]^*[asoc[i]];
166  par[suma]^*[asoc[i]].nod:=asoc[j];
167  {
168  par[suma]^*[asoc[i]].vallst:=obt[p]^*[asoc[j]];
169  }
170  if suma=k then begin
171  print(asoc[i],suma);
172  close(f);
173  halt;
174          end;
175      end;
176          end;
177          end;
178          end;
179  writeln(f,'-1');
180  close(f);
181
182 end.
```

38.1.3 *Rezolvare detaliată

38.2 Moara

prof. Ion Maxim, Inspectoratul Școlar Județean, Suceava

La poarta unei mori se află n saci, etichetați cu numere de la 1 la n , așezăți în linie, într-o ordine dată. Sacul cu eticheta i are greutatea g_i .

Morarul, cere ucenicului să reașeze sacii în ordinea crescătoare a etichetelor. Cum nu este spațiu de manevră pentru a schimba sacii între ei, ucenicul ia un scaun și-l așează lângă poziția k din sir, alege un sac și-l pune pe scaun, ia alt sac și-l mută în locul gol, apoi aduce alt sac în locul eliberat de acesta etc. sau aduce sacul de pe scaun în locul rămas gol. Repetă procedeul până când sacii ajung în ordinea cerută de morar. Scaunul nu se mută. Efortul depus de ucenic pentru mutatul unui sac, indiferent dacă acesta a fost pus sau nu pe scaun, este egal cu produsul dintre greutatea sacului și distanța pe care a fost transportat sacul. Distanța dintre doi saci alăturați este egală cu unitatea.

Cerință:

Se cere să se stabilească poziția amplasării scaunului și ordinea schimbării sacilor între ei, astfel încât ucenicul să facă un număr minim de mutări și efortul depus de acesta să fie minim.

Date de intrare:

Fișierul de intrare MOARA.IN conține pe prima linie numărul de saci n și pe liniile următoare, ordinea așezării sacilor, urmată de sirul greutăților sacilor g_1, g_2, \dots, g_n , unde g_i este greutatea sacului cu eticheta i .

Date de ieșire:

Ieșirea se va face în fișierul MOARA.OUT, care va conține pe prima linie

$p\ k\ e$ - unde p este poziția scaunului, k numărul de mutări și e valoarea efortului depus, și pe fiecare din liniile următoare câte o mutare de forma:

$d\ s$ - unde d este poziția destinație și s este poziția sursă, poziția scaunului se va nota cu 0 (zero!).

Restrictii:

$2 \leq n \leq 10000$

$1 \leq k \leq n$

$1 \leq g_i \leq 255$, pentru $1 \leq i \leq n$

Exemplu:

MOARA.IN	MOARA.OUT poate conține:
5	3 5 25
2 4 3	0 2
5 1	2 1
3 5 1 2 4	1 5 5 4 4 0

Timp maxim de execuție pe test: 1 secundă

38.2.1 *Indicații de rezolvare

38.2.2 Cod sursă

Listing 38.2.1: Moara_MIA.pas

```

1 { Andreica Mugurel Ionut - Bucuresti }
2 { !!! Sursa de +100 de puncte !!! }
3
4 Program Moara_M_I_A;
5
6 type vect=array[1..10000] of integer;
7
8 var f,temp:text;
9   poznr,npepoz:vect;
10  poznrini,npepozini:^vect;
11  g:array[1..10000] of integer;
12  sus,jos,ef,pozsc,ini,next,min,nmut,i,j,k,m,n,psc:longint;
13  e:longint;
14  s:string;

```

```

15      fin:boolean;
16
17 begin
18 assign(f,'moara.in');
19 reset(f);
20 readln(f,n);
21 for i:=1 to n do begin
22   read(f,npepoz[i]);
23   poznr[npepoz[i]]:=i;
24   end;
25 for i:=1 to n do read(f,g[i]);
26 close(f);
27
28
29 new(poznrini);
30 new(npepozini);
31 for i:=1 to n do begin
32   poznrini^[i]:=poznr[i];
33   npepozini^[i]:=npepoz[i];
34   end;
35
36 if n<=1000 then begin
37   pozsc:=0;
38   ef:=maxlongint;
39   for m:=1 to n do begin
40     nmut:=0;
41     psc:=m;
42     e:=0;
43
44     for i:=1 to n do begin
45       poznr[i]:=poznrini^[i];
46       npepoz[i]:=npepozini^[i];
47       end;
48     sus:=psc;
49     jos:=psc;
50     repeat
51       fin:=true;
52     for i:=jos downto 1 do
53       if (npepoz[i]<>i) then begin
54         k:=i;
55         fin:=false;
56         jos:=i;
57         break;
58       end;
59
60     if fin then
61       for i:=sus to n do
62         if (npepoz[i]<>i) then begin
63           k:=i;
64           fin:=false;
65           sus:=i;
66           break;
67         end;
68
69     if not fin then begin
70       e:=e+abs(k-psc)*g[npepoz[k]];
71       inc(nmut);
72       next:=k;
73       ini:=npepoz[k];
74       while (next<>ini) do begin
75         inc(nmut);
76         e:=e+abs(next-poznr[next])*g[next];
77         npepoz[next]:=next;
78         i:=next;
79         next:=poznr[next];
80         poznr[i]:=i;
81         end;
82       inc(nmut);
83       e:=e+abs(psc-ini)*g[ini];
84       poznr[ini]:=ini;
85       npepoz[ini]:=ini;
86       end;
87     until fin;
88
89     if e<ef then begin
90       pozsc:=psc;

```

```

91 ef:=e;
92         end;
93         end;
94 psc:=pozsc;
95 for i:=1 to n do begin
96 poznr[i]:=poznrini^i;
97 npepoz[i]:=npepozini^i;
98         end;
99         end
100 else begin
101 psc:=i;
102 min:=abs(npepoz[1]-1);
103 for i:=2 to n do
104 if abs(npepoz[i]-i)<min then begin
105 min:=abs(npepoz[i]-i);
106 psc:=i;
107         end;
108         end;
109
110
111 assign(temp,'moara.tmp');
112 rewrite(temp);
113 nmut:=0;
114 e:=0;
115 sus:=psc;
116 jos:=psc;
117
118 repeat
119 fin:=true;
120 for i:=jos downto 1 do
121 if (npepoz[i]<>i) then begin
122 k:=i;
123 fin:=false;
124 jos:=i;
125 break;
126         end;
127
128 if fin then
129 for i:=sus to n do
130 if (npepoz[i]<>i) then begin
131 k:=i;
132 fin:=false;
133 sus:=i;
134 break;
135         end;
136
137 if not fin then begin
138 writeln(temp,'0 ',k);
139 e:=e+abs(k-psc)*g[npepoz[k]];
140 inc(nmut);
141 next:=k;
142 ini:=npepoz[k];
143 while (next<>ini) do begin
144 writeln(temp,next,' ',poznr[next]);
145 inc(nmut);
146 e:=e+abs(next-poznr[next])*g[next];
147 npepoz[next]:=next;
148 i:=next;
149 next:=poznr[next];
150 poznr[i]:=i;
151         end;
152 writeln(temp,ini,' ',0);
153 inc(nmut);
154 e:=e+abs(psc-ini)*g[ini];
155 poznr[ini]:=ini;
156 npepoz[ini]:=ini;
157         end;
158 until fin;
159
160 close(temp);
161
162
163 reset(temp);
164 assign(f,'moara.out');
165 rewrite(f);
166 writeln(f,psc,' ',nmut,' ',e);

```

```

167 while not seekeof(temp) do begin
168   readln(temp,s);
169   writeln(f,s);
170           end;
171 close(f);
172 erase(temp);
173
174 end.
```

Listing 38.2.2: Moara_MIA_2.pas

```

1 { Andreica Mugurel Ionut }
2 { !!! Sursa de +100 de puncte !!! }
3 { --- Varianta 2 --- }
4
5 Program Moara_M_I_A;
6 label finish;
7 type vect=array[1..10000] of integer;
8   vdiff=array[1..10000] of longint;
9   mrk=array[1..10000] of boolean;
10
11 const CMAX=500000;
12     timelim=18;
13
14 var f,temp:text;
15   poznr,npepoz:vect;
16   poznrini,npepozini,asoc,asocaux:^vect;
17   dif,aux:^vdiff;
18   g:array[1..10000] of integer;
19   mark:^mrk;
20   sus,jos,ef,pozsc,ini,next,min,nmut,i,j,k,m,n,psc:longint;
21   i1,j1,m1,k1:integer;
22   e:longint;
23   s:string;
24   fin:boolean;
25   t2:longint absolute $0:$046c;
26   t1:longint;
27
28 procedure sorti(li,ls:integer);
29 begin
30   if (ls-li=1) then begin
31     if dif^[li]>dif^[ls] then begin
32       sus:=dif^[li];
33       dif^[li]:=dif^[ls];
34       dif^[ls]:=sus;
35       sus:=asoc^[li];
36       asoc^[li]:=asoc^[ls];
37       asoc^[ls]:=sus;
38           end;
39     end
40   else
41     if (ls-li>1) then begin
42       sorti(li,(li+ls) div 2);
43       sorti((li+ls) div 2+1,ls);
44       k1:=li-1;
45       i1:=li;
46       m1:=(li+ls) div 2;
47       j1:=m1+1;
48
49       while (i1<=m1) and (j1<=ls) do begin
50         inc(k1);
51         if (dif^[i1]<dif^[j1]) then begin
52           aux^[k1]:=dif^[i1];
53           asocaux^[k1]:=asoc^[i1];
54           inc(i1);
55           end
56         else begin
57           aux^[k1]:=dif^[j1];
58           asocaux^[k1]:=asoc^[j1];
59           inc(j1);
60           end;
61         end;
62       if (i1<=m1) then begin
63         for j1:=i1 to m1 do begin
64           inc(k1);
```

```

65 aux^ [k1]:=dif^ [j1];
66 asocaux^ [k1]:=asoc^ [j1];
67           end;
68           end
69 else
70 for ii:=j1 to ls do begin
71 inc(k1);
72 aux^ [k1]:=dif^ [ii];
73 asocaux^ [k1]:=asoc^ [ii];
74           end;
75 for ii:=li to ls do begin
76 dif^ [ii]:=aux^ [ii];
77 asoc^ [ii]:=asocaux^ [ii];
78           end;
79           end;
80 end;
81
82 begin
83 t1:=t2;
84 assign(f,'moara.in');
85 reset(f);
86 readln(f,n);
87 for i:=1 to n do begin
88 read(f,npepoz[i]);
89 poznr[npepoz[i]]:=i;
90           end;
91 for i:=1 to n do read(f,g[i]);
92 close(f);
93
94
95 new(poznrini);
96 new(npepozini);
97 for i:=1 to n do begin
98 poznrini^ [i]:=poznr[i];
99 npepozini^ [i]:=npepoz[i];
100           end;
101
102 new(dif);
103 new(aux);
104 new(asoc);
105 new(asocaux);
106 new(mark);
107
108 if n<=1000 then begin
109 pozsc:=0;
110 ef:=maxlongint;
111 for m:=1 to n do begin
112 nmut:=0;
113 psc:=m;
114 e:=0;
115 for i:=1 to n do begin
116 poznr[i]:=poznrini^ [i];
117 npepoz[i]:=npepozini^ [i];
118           end;
119
120 for i:=1 to n do begin
121 dif^ [i]:=abs(psc-poznr[i])*g[i]+abs(psc-i)*g[i]
122 -abs(poznr[i]-i)*g[i];
123 asoc^ [i]:=i;
124           end;
125           end;
126 sorti(1,n);
127
128 jos:=1;
129
130 repeat
131 fin:=true;
132
133 for i:=jos to n do
134 if (poznr[asoc^ [i]]<>asoc^ [i]) then
135 begin
136 fin:=false;
137 jos:=i+1;
138 k:=poznr[asoc^ [i]];
139 break;
140 end;

```

```

141
142 if not fin then begin
143   e:=e+abs(k-psc)*g[npepoz[k]];
144   inc(nmut);
145   next:=k;
146   ini:=npepoz[k];
147   while (next<>ini) do begin
148     inc(nmut);
149     e:=e+abs(next-poznr[next])*g[next];
150     npepoz[next]:=next;
151     i:=next;
152     next:=poznr[next];
153     poznr[i]:=i;
154     end;
155     inc(nmut);
156     e:=e+abs(psc-ini)*g[ini];
157     poznr[ini]:=ini;
158     npepoz[ini]:=ini;
159     end;
160   until fin;
161
162 if e<ef then begin
163   pozsc:=psc;
164   ef:=e;
165   end;
166 if (t2-t1)>=timelim then goto finish;
167   end;
168 psc:=pozsc;
169   end
170 else begin
171   fillchar(mark^,sizeof(mark^),false);
172   ef:=maxlongint;
173   pozsc:=0;
174
175 for m:=1 to CMAX div n do begin
176   case m of
177     1:begin
178       psc:=1;
179       min:=abs(npepoz[1]-1);
180       for i:=2 to n do
181         if abs(npepoz[i]-i)<min then begin
182           min:=abs(npepoz[i]-i);
183           psc:=i;
184           end;
185           mark^[psc]:=true;
186           end;
187     2:begin
188       psc:=1;
189       min:=abs(npepoz[1]-1);
190       for i:=2 to n do
191         if abs(npepoz[i]-i)>min then begin
192           min:=abs(npepoz[i]-i);
193           psc:=i;
194           end;
195           mark^[psc]:=true;
196           end;
197     3:begin
198       psc:=0;
199       min:=10000;
200       for i:=1 to n do
201         if ((abs(npepoz[i]-i)<min)
202         and (abs(npepoz[i]-i)>0))
203         then begin
204           min:=abs(npepoz[i]-i);
205           psc:=i;
206           end;
207           mark^[psc]:=true;
208           end;
209         else begin
210           repeat
211             psc:=random(n)+1;
212             until (not mark^[psc]);
213             end;
214           end;
215 { -- }
216

```

```

217 nmut:=0;
218 e:=0;
219
220 for i:=1 to n do begin
221 poznr[i]:=poznrini^i;
222 npepoz[i]:=npepozini^i;
223 end;
224
225 for i:=1 to n do begin
226 dif^i:=abs(psc-poznr[i])*g[i]+abs(psc-i)*g[i]
227 -abs(poznr[i]-i)*g[i];
228 asoc^i:=i;
229 end;
230 sorti(1,n);
231
232 jos:=1;
233
234 repeat
235 fin:=true;
236
237 for i:=jos to n do
238 if (poznr[asoc^i]<>asoc^i) then
239 begin
240 fin:=false;
241 jos:=i+1;
242 k:=poznr[asoc^i];
243 break;
244 end;
245
246 if not fin then begin
247 e:=e+abs(k-psc)*g[npepoz[k]];
248 inc(nmut);
249 next:=k;
250 ini:=npepoz[k];
251 while (next<>ini) do begin
252 inc(nmut);
253 e:=e+abs(next-poznr[next])*g[next];
254 npepoz[next]:=next;
255 i:=next;
256 next:=poznr[next];
257 poznr[i]:=i;
258 end;
259 inc(nmut);
260 e:=e+abs(psc-ini)*g[ini];
261 poznr[ini]:=ini;
262 npepoz[ini]:=ini;
263 end;
264 until fin;
265
266 if (e<ef) then begin
267 ef:=e;
268 pozsc:=psc;
269 end;
270 if (t2-t1)>=timelim then goto finish;
271 { -- }
272 end;
273 psc:=pozsc;
274 end;
275
276
277 finish:
278 for i:=1 to n do begin
279 poznr[i]:=poznrini^i;
280 npepoz[i]:=npepozini^i;
281 end;
282
283 assign(temp,'moara.tmp');
284 rewrite(temp);
285 nmut:=0;
286 e:=0;
287 jos:=1;
288
289 for i:=1 to n do begin
290 dif^i:=abs(psc-poznr[i])*g[i]+abs(psc-i)*g[i]
291 -abs(poznr[i]-i)*g[i];
292 asoc^i:=i;

```

```

293           end;
294   sorti(1,n);
295
296   repeat
297     fin:=true;
298
299   for i:=jos to n do
300     if (poznr[asoc^i]<>asoc^i) then
301     begin
302       fin:=false;
303       jos:=i+1;
304       k:=poznr[asoc^i];
305       break;
306     end;
307
308   if not fin then begin
309     writeln(temp,'0 ',k);
310     e:=e+longint(abs(k-psc)*g[npepoz[k]]);
311     inc(nmut);
312     next:=k;
313     ini:=npepoz[k];
314     while (next<>ini) do begin
315       writeln(temp,next,' ',poznr[next]);
316       inc(nmut);
317       e:=e+longint(abs(next-poznr[next])*g[next]);
318       npepoz[next]:=next;
319       i:=next;
320       next:=poznr[next];
321       poznr[i]:=i;
322       end;
323     writeln(temp,ini,' ',0);
324     inc(nmut);
325     e:=e+longint(abs(psc-ini)*g[ini]);
326     poznr[ini]:=ini;
327     npepoz[ini]:=ini;
328     end;
329   until fin;
330
331   close(temp);
332
333
334   reset(temp);
335   assign(f,'moara.out');
336   rewrite(f);
337   writeln(f,psc,' ',nmut,' ',e);
338   while not seeeof(temp) do begin
339     readln(temp,s);
340     writeln(f,s);
341     end;
342   close(f);
343   erase(temp);
344
345   writeln((t2-t1)/18:0:3);
346 end.

```

Listing 38.2.3: Moara-ofic.pas

```

1 { prof. Ioan Maxim - Suceava }
2
3 program moara;
4 var a:array[1..10000] of word;
5   g:array[1..10000] of byte;
6   v:array[1..10000] of boolean;
7   m:array[1..100] of word;
8   i,j,jj,n,k,p,pp,min,max:word;
9   ee:longint;
10  w:byte;
11  f,h:text;
12  function pozmin(k:word):word;
13  var min:byte;
14    i,j:word;
15  begin
16    i:=k;
17    v[i]:=true;
18    min:=g[a[i]];

```

```

19      j:=a[i];
20      repeat
21          if min<g[a[i]] then
22              begin
23                  min:=g[a[i]];
24                  j:=a[i];
25              end;
26          i:=a[i];
27          v[i]:=true;
28      until i=k;
29      pozmin:=j;
30  end;
31  function poz(k:word):word;
32  var i:word;
33  begin
34      i:=1;
35      while a[i]<>k do
36          i:=i+1;
37      poz:=i;
38  end;
39 begin
40     assign(f,'moaral.in');
41     reset(f);
42     readln(f,n);
43     i:=0;
44     repeat
45         if seekeoln(f) then
46             readln(f);
47         i:=i+1;
48         read(f,a[i]);
49     until i=n;
50     i:=0;
51     repeat
52         if seekeoln(f) then
53             readln(f);
54         i:=i+1;
55         read(f,g[i]);
56     until seekeof(f);
57     close(f);
58     p:=0;
59     w:=0;
60     i:=1;
61     repeat
62         while (i<=n) and v[i] do
63             i:=i+1;
64         if (i<=n) and (a[i]<>i) then
65             begin
66                 w:=w+1;
67                 m[w]:=pozmin(i);
68             end;
69         i:=i+1;
70     until i>n;
71     min:=m[1];
72     max:=m[1];
73     for i:=2 to w do
74         begin
75             if m[i]<min then
76                 min:=m[i];
77             if m[i]>max then
78                 max:=m[i];
79         end;
80     pp:=min;
81     ee:=65000;
82     for p:=min to max do
83         begin
84             e:=0;
85             for i:=1 to w do
86                 e:=e+g[a[m[i]]]*(abs(longint(p)-m[i])+abs(longint(p)-a[p]));
87             if e<ee then
88                 begin
89                     ee:=e;
90                     pp:=p;
91                 end;
92             end;
93     assign(f,'moa.out');
94     rewrite(f);

```

```

95      p:=pp;
96      e:=0;
97      k:=0;
98      for i:=1 to w do
99          begin
100             pp:=a[m[i]];
101             j:=m[i];
102             writeln(f,'0 ',j);
103             k:=k+1;
104             e:=e+g[pp]*abs(longint(m[i])-p);
105             repeat
106                 write(f,j,' ');
107                 jj:=poz(j);
108                 a[j]:=a[jj];
109                 writeln(f,jj);
110                 k:=k+1;
111                 e:=e+g[j]*abs(longint(j)-jj);
112                 j:=jj;
113             until j=pp;
114             e:=e+g[pp]*abs(longint(j)-p);
115             a[pp]:=pp;
116             k:=k+1;
117             writeln(f,j,' 0');
118         end;
119         close(f);
120         assign(f,'moa.out');
121         reset(f);
122         assign(h,'moaral.out');
123         rewrite(h);
124         writeln(h,p,' ',k,' ',e);
125         repeat
126             readline(f,i,j);
127             writeln(h,i,' ',j);
128         until seekeof(f);
129         close(f);
130         close(h);
131     end.

```

38.2.3 *Rezolvare detaliată

38.3 Puncte

prof. Doru Anastasiu Popescu, Liceul "Radu Greceanu", Slatina

Se dau n puncte în plan P_1, P_2, \dots, P_n și k un număr natural. Se cere să se verifice dacă se pot construi k segmente cu ambele capete în mulțimea formată din punctele P_1, P_2, \dots, P_n , astfel încât să nu se formeze nici un triunghi cu vârfurile în aceste puncte. Dacă există mai multe soluții se cere una dintre ele.

Date de intrare:

Fișierul de intrare PUNCTE.IN conține pe prima linie numărul n , iar pe a doua linie numărul k .

Date de ieșire:

Ieșirea se va face în fișierul PUNCTE.OUT ce va conține:

- o linie pe care se află caracterul 0, dacă nu există soluție.

- $k + 1$ linii, pe prima linie se află caracterul 1 (cu semnificația că există soluție), iar pe următoarele k linii se află câte două numere separate printr-un spațiu (de forma $i\ j$ cu semnificația că P_iP_j reprezintă un segment), în cazul când există soluție.

Restricții:

$1 \leq n \leq 300$

$1 \leq k \leq 300$

Exemplu:

PUNCTE.IN	PUNCTE.OUT
4	0
5	

PUNCTE.IN	PUNCTE.OUT poate conține:
4	1
2	1 4 1 2

Timp maxim de execuție pe test: 1 secundă

38.3.1 Indicații de rezolvare

Indicații 1:

Se imparte multimea celor N puncte în 2 submultimi:

una cu primele $N/2$ puncte, iar cealalată cu celelalte $N - N/2$ puncte.

Numarul maxim de segmente ce pot fi trasate este: $N/2 * (N - N/2)$ - unind fiecare punct din prima submultime cu fiecare punct din cea de-a două. Astfel se garantează că nu se vor obține triunghiuri.

Din numărul total de segmente, se aleg apoi oricare k .

Indicații 2:

Pentru a determina numărul maxim de segmente ce se pot construi fără să existe triunghiuri se partionează mulțimea punctelor în două submulțimi A și B , prima cu $[n/2]$ elemente și cea de-a două cu $n - [n/2]$ elemente.

Se unesc prin segmente toate punctele din A cu toate punctele din B (ca la grafuri bipartite complete).

Din faptul că nu se unesc puncte din cadrul acelorași multimi A sau B rezultă că nu se formează triunghiuri.

Numărul maxim de segmente căutat este $[n^2/4]$.

O soluție poate fi formată din orice k segmente construite în modul anterior.

Listing 38.3.1: puncte_dpa.pas

```

1 Program puncte_dpa;
2 var n,i,k,h,j:longint;
3   f:text;
4 begin
5 assign(f,'puncte.in');
6 reset(f);
7 readln(f,n);
8 readln(f,k);
9 close(f);
10 assign(f,'puncte.out');
11 rewrite(f);
12 if k>trunc(n*n/4) then
13   write(f,0)
14   else
15 begin
16   writeln(f,1);
17   h:=0;
18   for i:=1 to trunc(n/2) do
19     for j:=trunc(n/2)+1 to n do
20     begin
21       inc(h);
22       if h<=k then
23         writeln(f,i,' ',j);
24       end;
25     end;
26   close(f);
27 end.

```

38.3.2 Cod sursă

38.3.3 *Rezolvare detaliată

38.4 SICN

prof. Emanuela Cerchez, Liceul de Informatică, Iași

Serviciul de Informații al Comisiei Naționale (SICN) este constituit din n agenți oculți ($1 \leq n \leq 150$), cu numere de cod distințe de la 1 la n și un agent șef codificat 0.

Din motive de securitate, nu oricare doi agenți au contacte (informaționale!) directe, dar prin contactele directe existente, oricare doi agenți își pot comunica informații.

Un serviciu este considerat *forte* dacă și numai dacă nu conține nici un agent prin suprimarea căruia se compromite comunicarea (și ca urmare, să existe agenți care să nu își mai poată transmite informații).

Scrieți un program, care verifică dacă SICN este *forte*. În plus, dacă SICN nu este *forte*, programul să determine:

- verigile slabe ale SICN (agenții prin a căror suprimare se compromite comunicarea);
- toate grupurile de agenți care sunt *forte* în cadrul SICN, maximale cu această proprietate.

Date de intrare:

Fișierul de intrare se numește SICN.IN și conține pe prima linie n , numărul de agenți din SICN, (exclusiv șefului), iar pe fiecare din următoarele linii câte o pereche de numere

$x\ y$

unde $x, y \in \{0, 1, 2, \dots, n\}$, sunt separate prin spații și au semnificația "x și y au contact direct".

Date de ieșire:

Fișierul de ieșire, numit SICN.OUT, va conține mesajul "SICN este forte" sau:

- pe prima linie, mesajul "SICN nu este forte".
- pe a doua linie, numerele de cod ale agenților care constituie verigile slabe ale serviciului, în ordine crescătoare, separate prin spații;
- pe a treia linie, m - numărul de grupuri *forte* în cadrul SICN;
- pe fiecare din următoarele m linii, numerele de cod ale agenților fiecărui grup *forte*, în ordine crescătoare, separate prin spații.

Exemplu:

SICN.IN	SICN.OUT
2	SICN este forte
0 2	
0 1	
2 1	

SICN.IN	SICN.OUT
3	SICN nu este forte
0 1	0 1
0 3	3
2 1	0 3
	1 2
	0 1

Timp de execuție: maxim 1 secundă per test.

38.4.1 Indicații de rezolvare

Problema se rezuma la determinarea *punctelor critice* într-un graf, și, în cazul în care acestea există, a tuturor *componentelor biconexe* ale acestuia.

38.4.2 Cod sursă

Listing 38.4.1: Sicn_MIA.pas

```

1 { Andreica Mugurel Ionut }
2 { !!! Sursa de 100 de puncte !!! }
3
4 Program SICN;
5 type muchie=record
6     c1,c2:byte;

```

```

7           end;
8      stmuc=array[1..22900] of muchie;
9      linie=array[0..150] of boolean;
10
11 var f:text;
12   n0,i,j,k,m,n,nst,nbi,maxim:integer;
13   critic:array[0..150] of boolean;
14   maxdf,nrdf,tat:array[-1..150] of integer;
15   a:array[0..150,0..150] of shortint;
16   vizit:array[0..150] of boolean;
17   stiva:^stmuc;
18   instiva:array[0..150,0..150] of boolean;
19   biconex:array[1..151] of ^linie;
20   mcrit:array[0..150] of ^linie;
21   nterm:array[0..150] of boolean;
22   nel:array[1..151] of integer;
23   ok:boolean;
24
25 procedure DF(nod,tata:integer);
26 var s:integer;
27 begin
28   nrdf[nod]:=nrdf[tata]+1;
29   tat[nod]:=tata;
30   maxdf[nod]:=nrdf[nod];
31   vizit[nod]:=true;
32
33   nterm[nod]:=true;
34   for s:=0 to n do
35     if a[nod,s]=1 then begin
36       if (not instiva[nod,s]) then begin
37         inc(nst);
38         stiva^[nst].c1:=nod;
39         stiva^[nst].c2:=s;
40         instiva[nod,s]:=true;
41         instiva[s,nod]:=true;
42           end;
43
44       if (vizit[s]) and (s<>tat[nod]) and (nrdf[s]<maxdf[nod])
45       then maxdf[nod]:=nrdf[s];
46       if (not vizit[s]) then begin
47         nterm[nod]:=false;
48         df(s,nod);
49           end;
50
51       if tat[s]=nod then begin
52         if maxdf[s]>=nrdf[nod] then begin
53           mcrit[s]^[nod]:=true;
54           mcrit[nod]^[s]:=true;
55           critic[nod]:=true;
56           {
57             if (not (nterm[s])) and (maxdf[s]>nrdf[nod])
58             then critic[s]:=true;
59           }
60           end;
61         if (maxdf[s]=nrdf[nod]) or
62         ((maxdf[s]>maxdf[nod]) and (mcrit[nod]^*[s]))
63         then begin
64           inc(nbi);
65           while ((stiva^[nst].c1<>nod) or (stiva^[nst].c2<>s))
66             and ((stiva^[nst].c1<>s) or (stiva^[nst].c2<>nod))
67             do begin
68               biconex[nbi]^*[stiva^[nst].c1]:=true;
69               biconex[nbi]^*[stiva^[nst].c2]:=true;
70               {
71                 instiva[stiva^[nst].c1,stiva^[nst].c2]:=false;
72                 instiva[stiva^[nst].c2,stiva^[nst].c1]:=false;
73               }
74             dec(nst);
75           end;
76         biconex[nbi]^*[s]:=true;
77         biconex[nbi]^*[nod]:=true;
78         dec(nst);
79           end;
80         end;
81       else if maxdf[s]<maxdf[nod] then maxdf[nod]:=maxdf[s];
82       if (nod=0) and (critic[nod]) then begin

```

```

83  ok:=false;
84  n0:=0;
85  for i:=1 to n do
86  if tat[i]=-1 then ok:=true
87  else if tat[i]=0 then inc(n0);
88  if n0>1 then ok:=true;
89  if not ok then critic[nod]:=false;
90          end;
91          end;
92  {
93  if (nterm[nod]) then begin
94  inc(nbi);
95  biconex[nbi]^[nod]:=true;
96          end;
97  }
98  end;
99
100 begin
101 assign(f,'sicn.in');
102 reset(f);
103 readln(f,n);
104 fillchar(a,sizeof(a),0);
105 while not seeeof(f) do begin
106 readln(f,i,j);
107 a[i,j]:=1;
108 a[j,i]:=1;
109          end;
110          end;
111 close(f);
112
113 fillchar(maxdf,sizeof(maxdf),0);
114 fillchar(nrdf,sizeof(nrdf),0);
115 fillchar(vizit,sizeof(vizit),false);
116 for i:=1 to n do tat[i]:=-1;
117
118 fillchar(instiva,sizeof(instiva),false);
119 fillchar(nterm,sizeof(nterm),false);
120 fillchar(critic,sizeof(critic),false);
121
122 new(stiva);
123 for i:=1 to (n+1)*(n+1) do begin
124 stiva^[i].c1:=0;
125 stiva^[i].c2:=0;
126          end;
127 nst:=0;
128 nbi:=0;
129
130 for i:=0 to n do begin
131 new(mcrit[i]);
132 for j:=0 to n do mcrit[i]^[j]:=false;
133          end;
134
135
136 for i:=0 to n do begin
137 new(biconex[i]);
138 for j:=0 to n do biconex[i]^[j]:=false;
139          end;
140
141 nrdf[-1]:=0;
142 df(0,-1);
143
144 assign(f,'sicn.out');
145 rewrite(f);
146 if nbi=1 then writeln(f,'FORTE')
147 else begin
148 writeln(f,'NEFORTE');
149 for i:=0 to n do
150 if critic[i] then begin
151 write(f,i);
152 break;
153          end;
154          end;
155 for i:=i+1 to n do
156 if critic[i] then write(f,' ',i);
157 writeln(f);
158

```

```

159 writeln(f,nbi);
160 nst:=0;
161 for i:=1 to nbi do
162 begin
163 inc(nst);
164 if nst>1 then writeln(f);
165 for j:=0 to n do if biconex[i]^ [j] then begin
166 write(f,j);
167 break;
168 end;
169 for j:=j+1 to n do if biconex[i]^ [j] then begin
170 if biconex[i]^ [j] then write(f,' ',j);
171 end;
172 end;
173 end;
174 close(f);
175 end.

```

Listing 38.4.2: Sicinofic.pas

```

1 program SOICN;
2 const NMaxVf = 150;
3 type Vf = 0 .. NMaxVf;
4 Stiva = ^NodStiva;
5 NodStiva = record
6   f: byte; t: integer;
7   urm: Stiva;
8 end;
9 Lista = ^NodLista;
10 NodLista = record
11   v: Vf;
12   leg: Lista;
13 end;
14 Graf = array[Vf] of Lista;
15 CompBiconexa = set of Vf;
16 var S: Stiva; G: Graf;
17 low, dfn: array[Vf] of integer;
18 nr, n, num, i, nrdesc: Vf;
19 BIC: array[Vf] of CompBiconexa;
20 A: CompBiconexa;
21 fout: text; nout:string;
22
23 procedure Initializare;
24 var x, y: Vf; p: Lista;
25   fin: text; nume: string;
26 begin
27 write ('fisier intrare '); readln (nume);
28 assign(fin, nume); reset(fin);
29 readln(fin, n);
30 while not seekeof (fin) do begin
31   readln (fin, x, y);
32   new(p); p^.v := x; p^.leg:=G[y]; G[y]:=p;
33   new(p); p^.v := y; p^.leg:=G[x]; G[x]:=p;
34 end;
35 for x := 0 to n do
36 begin
37   dfn[x] := -1;
38   low[x]:=-1;
39 end;
40 close(fin);
41 new(S); S^.f := 0; S^.t := -1; S^.urm := nil;
42 end;
43
44 procedure Constructie_Comp_Biconexa(x, u: integer);
45 var p: Stiva; a, b: integer;
46 begin
47 inc(nr);{nr - numarul de componente biconexe}
48 repeat
49   p := S; a:=p^.t; b := p^.f; S := S^.urm;
50   BIC[nr] := BIC[nr] + [a, b];
51   dispose(p);
52 until (a = u) and (b = x);
53 end;
54

```

```

55  function min (a, b: integer): integer;
56  begin
57    if a < b then min := a
58      else min := b
59  end;
60
61  procedure Biconex(u, tu: integer);
62  {calculeaza dfn si low si afiseaza componentele biconexe}
63  var p: Lista;
64    q: Stiva;
65    x: integer;
66  begin
67    dfn[u] := num; low[u] := dfn[u];
68    inc(num);
69    p := G[u];
70    while p <> nil do {parcure lista de adiacenta a lui u}
71      begin
72        x := p^.v;
73        if (x <> tu) and (dfn[x] < dfn[u])
74          then begin {insereaza in stiva S muchia(u,x)}
75            new(q); q^.f := x; q^.t := u;
76            q^.urm := S; S := q;
77            end;
78        if dfn[x] = -1 then {x nu a mai fost vizitat}
79          begin
80            if u=0 then inc(nrdesc);
81            Biconex (x, u);
82            low[u] := min(low[u], low[x]);
83            if low[x] >= dfn[u] then begin
84              {u este un punct de articulatie,
85              am identificat o componenta biconexa,
86              ce contine muchiile din stiva S pana la (u,x)}
87              if u<>0 then
88                A := A + [u];
89                Constructie_Comp_BiconexA(x, u);
90              end;
91              end;
92            else if x <> tu then low[u] := min(low[u], dfn[x]);
93            p := p^.leg
94            end;
95  end;
96
97  procedure AfisMult (var X: CompBiconexa);
98  var v: Vf;
99  begin
100   for v := 0 to n do
101     if v in X then write (fout, v, ' ');
102   writeln (fout);
103  end;
104
105 begin {program principal}
106   Initializare;
107   write('fisier de iesire corect '); readln(nout);
108   assign (fout, {'soicn.out'}nout); rewrite (fout);
109   Biconex(0, -1);
110
111  if nr = 1 then writeln (fout, 'FORTE')
112  else
113    begin
114      writeln (fout, 'NEFORTE');
115      if NrDesc>1 then A:=A+[0];
116      AfisMult (A);
117      writeln (fout, nr);
118      for i := 1 to nr do AfisMult (BIC[i]);
119    end;
120  close (fout);
121 end.

```

38.4.3 *Rezolvare detaliată

38.5 Spioni

prof. Rodica Pintea, Liceul "Grigore Moisil", București

O retea formată din n spioni, numerotăți de la 1 la n , aflați în diferite localități, au o convenție de convocare. Ei vor fi convocați telefonic într-o localitate fixă numită bază. Convenția de transmitere a convocării cuprinde următoarele reguli:

cu excepția unora, fiecare spion știe ce persoane trebuie să-l contacteze telefonic; pentru a considera convocarea validă și a-i da curs, trebuie să primească semnalul de convocare de la toate persoanele de contact;

cei câțiva spioni care nu așteaptă nici o convocare știu că trebuie să pornească toți deodată, la o oră considerată ora 0, spre bază;

fiecare persoană, imediat ce a ajuns la bază, primește lista persoanelor pe care trebuie să le sune, transmițându-le convocarea;

durata unei convorbiri telefonice este neglijabilă;

abia în momentul în care sunt prezente toate cele n persoane la bază poate să înceapă ședința.

Cerință:

Cunoscându-se, pentru fiecare spion, durata deplasării până la bază (exprimată în ore) și lista persoanelor care trebuie să îl contacteze, scrieți un program care stabilește dacă se poate realiza convocarea tuturor persoanelor. Dacă este posibil, să se determine:

a) numărul minim de ore care trec până când poate să înceapă ședință;

b) numărul minim de persoane care trebuie transportate în regim de urgență astfel încât ședința să poată începe cu cel puțin o oră mai devreme; numim transport în regim de urgență un transport în care, în locul timpului dat, transportul durează cu cel puțin o oră mai puțin;

c) numerele de ordine ale persoanelor care vor fi transportate în regim de urgență.

Date de intrare:

Fișierul de intrare SPIONI.IN, conține:

pe prima linie, numărul n de spioni;

pe a doua linie, n numere naturale t_1, t_2, \dots, t_n , despărțite prin câte un spațiu, numere reprezentând durata deplasării fiecărui spion până la bază (în regim normal), durată exprimată în ore;

pe fiecare dintre următoarele n linii, câte o succesiune de minim 0 și maxim n numere naturale despărțite prin câte un spațiu, numerele de pe linia $i + 2$ a fișierului reprezentând lista persoanelor care trebuie să convoace persoana având numărul i , pentru ca aceasta să pornească spre bază.

Datele de ieșire:

Pe prima linie a fișierului SPIONI.OUT se află numărul -1 dacă nu se poate realiza convocarea, sau numărul de ore după care poate să înceapă ședința, în caz contrar.

În cazul în care convocarea este posibilă, fișierul mai conține:

- pe a doua linie, un număr p , reprezentând numărul de persoane care, transportate în regim de urgență, pot devansa ora de începere a ședinței cu cel puțin o oră;

- pe a treia linie, numerele de ordine ale persoanelor care trebuie să fie transportate în regim de urgență, pentru a putea începe ședința mai devreme, numere despărțite prin câte un spațiu.

Restricții:

$1 \leq n \leq 100$

$2 \leq t_i \leq 500$

Exemplu:

SPIONI.IN	SPIONI.OUT poate conține:
5	18
9 8 3 9 1	1
	4
1 3 5	
2 3	

Timp maxim de execuție pe test: 3 secunde

38.5.1 Indicații de rezolvare

Problema nu are soluție dacă graful orientat determinat de cei n spioni conține cicluri. Acest lucru este verificat prin *sortare topologică* - daca la un anumit punct nu există nici un nod de grad 0, și nu au fost introduse toate nodurile în coadă, atunci graful conține un ciclu.

Timpul minim de începere a ședinței este maximul dintre timpul după care poate să pornească fiecare agent i , și timpul care îi ia pentru a ajunge la bază.

Pentru a determina numărul minim de persoane ce trebuie transportate în regim de urgență, se construiește o *rețea de flux*, cu muchii - muchiile critice din graf initial, și noduri - cei n spioni: fiecare nod are capacitatea 1, iar muchiile pot avea orice capacitate.

Valoarea fluxului maxim în această rețea reprezintă numărul minim de spioni ce trebuie transportați în regim de urgență.

Pentru a determina numerele de ordine ale spionilor ce trebuie transportați în regim de urgență, se parcurge rețeaua pe muchiile fără flux, de la destinație (virtuală - în care intră doar muchiile care determină timpul de începere a ședinței), până se ajunge la un nod prin care trece flux. Se marchează acest nod, și se repetă procedeul pentru alt traseu.

În final, se pornește de la fiecare nod marcat și se observă pentru care dintre nodurile "terminale" reduce timpul de start. Nodurile "terminale" care nu au fost "reduse" (prin care, deci, circulă flux), sunt marcate apoi și ele.

Spionii ce trebuie transportați în regim de urgență sunt cei ce au numerele de ordine egale cu nodurile marcate.

38.5.2 Cod sursă

Listing 38.5.1: Spioni_comisie.pas

```

1 type
2   matr=array[0..200,0..200] of byte;
3   vect=array[0..200] of longint;
4
5 var fis,rez:text;
6   a:matr;c,f:^matr;
7   d,tmp,used:vect;
8   n,x,i:byte;
9   viz:set of byte;
10  j,k,ini,fin,fm:word;
11  q,max,t:array[0..200] of word;
12
13 procedure nue;
14 begin
15   writeln(rez,-1);
16   close(rez);halt
17 end;
18
19 function min(m1,m2:word):word;
20 begin
21   if m1<m2 then min:=m1 else min:=m2
22 end;
23
24 procedure add(nod,tata:integer;flux:word);
25 begin
26   viz:=viz+[nod]; inc(fin);
27   q[fin]:=nod; t[nod]:=tata;
28   max[nod]:=flux
29 end;
30
31 procedure fillmin(i:byte);
32 var j:byte;
33 begin
34   used[i]:=1;
35   for j:=1 to 2*n+1 do
36     if a[i,j]=1 then begin
37       if used[j]=1 then nue;
38       if tmp[j]<tmp[i]+d[i] then begin
39         tmp[j]:=tmp[i]+d[i];
40         fillmin(j)
41       end
42     end;
43   used[i]:=0
44 end;

```

```

45
46 procedure fillmax(i:byte);
47 var j:byte;
48 begin
49   for j:=0 to 2*n do
50     if a[j,i]=1 then
51       if not ((d[j]>0) or (tmp[j]=tmp[i])) then a[j,i]:=0
52       else fillmax(j)
53 end;
54
55 function fluxmax:word;
56 begin
57   repeat
58     ini:=1; fin:=1; q[1]:=0;
59     max[0]:=maxint;
60     viz:=[0];
61     while (ini<=fin) do
62     begin
63       k:=q[ini]; inc(ini);
64       for j:=1 to 2*n+1 do if not (j in viz) then
65         if c^[k,j]>f^[k,j] then
66           add(j, k,min(max[k],c^[k,j]-f^[k,j]))
67         else if f^[j,k]>0 then
68           add(j, -k,min(max[k],f^[j,k]))
69       end;
70       i:=2*n+1;
71       if 2*n+1 in viz then
72         while i>0 do
73         begin
74           j:=t[i];
75           if j>=0 then
76             inc(f^[j,i],max[2*n+1]);
77           if j<0 then
78             dec(f^[i,-j],max[2*n+1]);
79           i:=j
80         end
81       until not (2*n+1 in viz);
82       j:=0;
83       for i:=1 to 2*n do j:=j+f^[0,i];
84       fluxmax:=j;
85     end;
86
87   begin
88     assign(fis,'spy.in');reset(fis);
89     assign(rez,'spy.out');rewrite(rez);
90     readln(fis,n);
91     for i:=1 to n do read(fis,d[2*i-1]);
92     readln(fis);
93     d[0]:=0;
94     for i:=1 to n do begin a[0,2*i-1]:=1;a[2*i,2*n+1]:=1;a[2*i-1,2*i]:=1 end;
95     for i:=1 to 2*n+1 do begin tmp[i]:=-1;used[i]:=0 end;
96     for i:=1 to n do begin
97       while not seekeoln(fis) do begin
98         read(fis,x);a[2*x,2*i-1]:=1;
99       end;
100      readln(fis)
101    end;
102    tmp[0]:=0;fillmin(0);
103    writeln(rez,tmp[2*n+1]);
104    for i:=1 to n do
105      for j:=1 to 2*n+1 do
106        if (a[2*i,j]=1)and(tmp[2*i]<>tmp[j]) then a[2*i,j]:=0;
107    new(c);c^:=a;
108    new(f); fillchar(f^,sizeof(matr),0);
109    fm:=fluxmax;writeln(rez,fm);
110    for i:=1 to n do
111      if (2*i-1 in viz) and not (2*i in viz) then write(rez,i,' ');
112    writeln(rez);
113    close(rez);
114  end.

```

38.5.3 *Rezolvare detaliată

38.6 Tezaur

prof. Doru Anastasiu Popescu, Liceul "Radu Greceanu", Slatina

Într-un document arheologic recent descoperit se face referire la un mare tezaur. Datorită faptului că documentul poate fi interpretat în mai multe moduri se face apel la n arheologi care vor studia independent documentul.

La terminarea studiului, fiecare arheolog întocmește o hartă pe care marchează o zonă poligonală închisă și convexă despre care se presupune că este locul unde se află tezaurul.

Deoarece fondurile alocate pentru descoperirea tezaurului sunt reduse, se ia hotărârea să se înceapă cercetările pe teren doar în zona de pe hartă, precizată de toți arheologii.

Cerință:

Cunoscând valoarea n și coordonatele vârfurilor zonelor determinate de arheologi, să se determine aria suprafeței de pe hartă precizată de toți arheologii (intersecția celor n zone).

Date de intrare:

Fișierul de intrare TEZAUR.IN conține:

n - numărul de zone

$m[1]$ - numărul de vârfuri pentru zona primului arheolog

$x_{11}y_{11}\dots x_{1m[1]}y_{1m[1]}$ - coordonatele vârfurilor zonei primului arheolog (date în sens invers acelor de ceasornic)

...

$m[n]$ - numărul de vârfuri pentru zona ultimului arheolog

$x_{n1}y_{n1}\dots x_{nm[n]}y_{nm[n]}$ - coordonatele vârfurilor zonei celui de-al n -lea arheolog (date în sens invers acelor de ceasornic)

Date de ieșire:

Pe prima linie a fișierului TEZAUR.OUT se va scrie aria zonei de pe hartă, precizată de toți arheologii, cu două zecimale. Dacă nu există suprafață comună zonelor celor n arheologi, în fișierul de ieșire se va scrie numărul 0.

Exemplu:

TEZAUR.IN	TEZAUR.OUT
3	400.00
4	
-20 30 -20 -20 40 -20 40 30	
5	
10 80 10 -30 60 -50 100 60 70 80	
4	
20 10 70 10 70 60 20 60	

Restricții:

$1 \leq n \leq 30$

$3 \leq m[i] \leq 20, i \in \{1, 2, \dots, n\}$

Coordonatele vârfurilor zonelor sunt numere întregi din intervalul $[-1000, 1000]$

Numărul din fișierul de ieșire se va scrie cu 2 zecimale.

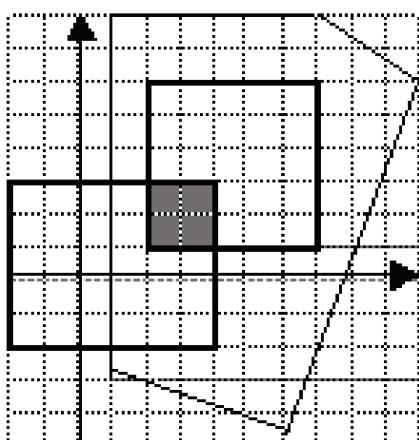


Figura 38.2: Tezaur

Timp maxim de execuție pe test: 1 secundă

38.6.1 Indicații de rezolvare

Se intersectează primele 2 poligoane, și se păstrează intersecția lor (tot un poligon convex). Apoi această intersecție se intersectează cu al treilea poligon, și se păstrează noua intersecție s.a.m.d, până când se obține intersecția celor n poligoane.

38.6.2 Cod sursă

Listing 38.6.1: Tezaur_comisie.pas

```

1 program tezaur;
2 type ve=array [1..100] of real;
3 var
4   f:text;
5   t1,t2,x1,y1,x2,y2,x,y,a:ve;
6   n,i,j,m,k,l,o,p:longint;
7   a1,b1,c1,a2,b2,c2,auxx,auxy,aux,minx,miny,m1,m2:real;
8   sw:boolean;
9   n1,i1:longint;
10  ar:real;
11
12 function min(a,b:real):real;
13 begin
14   if a>b then min:=b else min:=a;
15 end;
16
17 function max(a,b:real):real;
18 begin
19   if a>b then max:=a else max:=b;
20 end;
21
22 function arie_tri(x1,y1,x2,y2,x3,y3:real):real;
23 begin
24 arie_tri:=0.5*abs(x1*y2+x2*y3+x3*y1-x3*y2-x1*y3-x2*y1)
25 end;
26
27 function arie_poli(x,y:ve;d:integer):real;
28 var i:integer;arie:real;
29 begin
30 arie:=0;
31 for i:=2 to d-1 do
32   arie:=arie+arie_tri(x[1],y[1],x[i],y[i],x[i+1],y[i+1]);
33 arie_poli:=arie;
34 end;
35
36 begin
37   {Citirea datelor}
38   assign(f,'tezaur.in');
39   reset(f);
40   readln(f,n1);
41   readln(f,n);
42   for i:=1 to n do read(f,x1[i],y1[i]);
43   readln(f);
44   x1[n+1]:=x1[1];
45   y1[n+1]:=y1[1];
46   x1[n+2]:=x1[2];
47   y1[n+2]:=y1[2];
48
49 for i1:=2 to n1 do
50 begin
51   readln(f,m);
52   for i:=1 to m do read(f,x2[i],y2[i]);
53   readln(f);
54   p:=0;
55   x2[m+1]:=x2[1];
56   y2[m+1]:=y2[1];
57   x2[m+2]:=x2[2];
58   y2[m+2]:=y2[2];
59   for i:=1 to n do

```

```

60         for j:=1 to m do
61             begin
62                 a1:=y1[i+1]-y1[i];
63                 b1:=x1[i]-x1[i+1];
64                 c1:=y1[i]*x1[i+1]-y1[i+1]*x1[i];
65                 if (a1*x2[j]+b1*y2[j]+c1)*(a1*x2[j+1]+b1*y2[j+1]+c1)<=0 then
66                     begin
67                         if x1[i+1]=x1[i] then m1:=0
68                         else if y1[i+1]=y1[i] then m1:=30000000
69                         else m1:=(y1[i+1]-y1[i])/(x1[i+1]-x1[i]);
70                         if x2[j+1]=x2[j] then m2:=0
71                         else if y2[j+1]=y2[j] then m2:=30000000
72                         else m2:=(y2[j+1]-y2[j])/(x2[j+1]-x2[j]);
73                         if abs(m1)=abs(m2) then
74                             begin
75                             end
76                         else
77                             begin
78                             a2:=y2[j+1]-y2[j];
79                             b2:=x2[j]-x2[j+1];
80                             c2:=y2[j]*x2[j+1]-y2[j+1]*x2[j];
81                             if a1=0 then
82                                 begin
83                                     auxy:=-c1/b1;
84                                     auxx:=(-c2-b2*auxy)/a2;
85                                 end
86                             else if a2=0 then
87                                 begin
88                                     auxy:=-c2/b2;
89                                     auxx:=(-c1-b1*auxy)/a1;
90                                 end
91                             else if b1=0 then
92                                 begin
93                                     auxx:=-c1/a1;
94                                     auxy:=(-c2-a2*auxx)/b2;
95                                 end
96                             else if b2=0 then
97                                 begin
98                                     auxx:=-c2/a2;
99                                     auxy:=(-c1-a1*auxx)/b1;
100                                end
101                         else
102                             begin
103                                 auxx:=((b2/b1)*c1-c2)/(a2-a1*(b2/b1));
104                                 auxy:=(-c1-a1*auxx)/b1;
105                             end;
106                             if (auxx>=min(x1[i],x1[i+1])) and
107                                 (auxx<=max(x1[i],x1[i+1])) and
108                                 (auxy>=min(y1[i],y1[i+1])) and
109                                 (auxy<=max(y1[i],y1[i+1])) then
110                                 begin
111                                     p:=p+1;
112                                     x[p]:=auxx;
113                                     y[p]:=auxy;
114                                 end;
115                             end;
116                         end;
117                     end;
118                 for j:=1 to m do
119                     begin
120                         sw:=true;
121                         for i:=1 to n do
122                             if ((y1[i+2]-y1[i])*(x1[i+1]-x1[i])-  

123                                 (x1[i+2]-x1[i])*(y1[i+1]-y1[i]))*  

124                                 ((y2[j]-y1[i])*(x1[i+1]-x1[i])-  

125                                 (x2[j]-x1[i])*(y1[i+1]-y1[i]))<=0 then sw:=false;
126                         if sw then
127                             begin
128                                 p:=p+1;
129                                 x[p]:=x2[j];
130                                 y[p]:=y2[j];
131                             end;
132                         end;
133                     for j:=1 to n do
134                         begin
135                             sw:=true;

```

```

136      for i:=1 to m do
137          if ((y2[i+2]-y2[i])*(x2[i+1]-x2[i])-  

138              (x2[i+2]-x2[i])*(y2[i+1]-y2[i]))*  

139                  ((y1[j]-y2[i])*(x2[i+1]-x2[i])-  

140                      (x1[j]-x2[i])*(y2[i+1]-y2[i]))<=0 then sw:=false;
141      if sw then
142          begin
143              p:=p+1;
144              x[p]:=x1[j];
145              y[p]:=y1[j];
146          end;
147      end;
148  minx:=30000000;
149  for i:=1 to p do if x[i]<minx then
150      begin
151          j:=i;
152          minx:=x[i];
153      end;
154  miny:=30000000;
155  for i:=1 to p do
156      if (x[i]=minx) and (y[i]<miny) then
157          begin
158              j:=i;
159              miny:=y[i];
160          end;
161  for i:=1 to p do
162      if i<>j then
163          begin
164              if x[i]=x[j] then a[i]:=30000000
165                  else a[i]:=(y[i]-y[j])/(x[i]-x[j]);
166          end;
167  a[j]:=-30000000;
168  for i:=1 to p do
169      for j:=1 to p do
170          if (i<>j) and (x[i]=x[j]) and (y[i]=y[j]) then
171              begin
172                  x[j]:=-30000000;
173                  y[j]:=-30000000;
174              end;
175  sw:=true;
176  while sw do
177      begin
178          sw:=false;
179          for i:=1 to p-1 do
180              if a[i]>a[i+1] then
181                  begin
182                      aux:=a[i];
183                      a[i]:=a[i+1];
184                      a[i+1]:=aux;
185                      aux:=x[i];
186                      x[i]:=x[i+1];
187                      x[i+1]:=aux;
188                      aux:=y[i];
189                      y[i]:=y[i+1];
190                      y[i+1]:=aux;
191                      sw:=true;
192                  end;
193          end;
194      {Afisarea}
195      n:=0;
196      for i:=1 to p do
197          if (x[i]<>-30000000) and (y[i]<>-30000000) then
198              begin
199                  inc(n);
200                  x1[n]:=x[i];
201                  y1[n]:=y[i];
202              end;
203          x1[n+1]:=x1[1];
204          y1[n+1]:=y1[1];
205          x1[n+2]:=x1[2];
206          y1[n+2]:=y1[2];
207      end;
208  close(f);
209  assign(f,'tezaur.out');
210  rewrite(f);
211  ar:=arie_poli(x1,y1,n);

```

```

212 write(f,ar:10:2);
213 close(f);
214 end.
```

Listing 38.6.2: Tezaur_MIA.pas

```

1 { Andreica Mugurel Ionut }
2 { !!! Sursa de 100 de puncte !!! }
3
4 Program Tezaur;
5 type punct=record
6   x,y:real;
7   end;
8   poligon=record
9     np:integer;
10    co:array[1..21] of punct;
11    end;
12   poly=record
13     np:integer;
14     co:array[1..1000] of punct;
15     end;
16 var f:text;
17 p,i,j,k,co,m,n,iaux,jaux:integer;
18 pol:array[1..30] of poligon;
19 aux,pint,inters:poly;
20 a,b,c,arie,xii,yii:real;
21 smn,smnl:shortint;
22 inside:boolean;
23 select:array[1..1000] of boolean;
24 pt:punct;
25 st:array[1..1000] of integer;
26
27 function equal(a,b:real):boolean;
28 begin
29   if abs(a-b)<0.0001 then equal:=true
30   else equal:=false;
31 end;
32
33 function semn(a,b,c,x,y:real):shortint;
34 var r:real;
35 begin
36   r:=a*x+b*y+c;
37   if equal(r,0) then semn:=0
38   else if r>0 then semn:=1
39   else semn:=-1;
40 end;
41
42 function intre(x11,x22,xa:real):boolean;
43 var d1,d2,x111,x222:real;
44 begin
45   if (x11>x22) then begin
46     x111:=x22;
47     x222:=x11;
48           end
49   else begin
50     x111:=x11;
51     x222:=x22;
52           end;
53
54   d1:=x111-xa;
55   d2:=x222-xa;
56
57   intre:=((d1<0) or equal(d1,0))
58   and ((d2>0) or equal(d2,0));
59 end;
60
61 function intersect(x1,y1,x2,y2,x3,y3,x4,y4:real;var xi,yi:real):boolean;
62 var a11,b11,c11,a22,b22,c22:real;
63 begin
64   a11:=y1-y2;
65   b11:=x2-x1;
66   c11:=x1*y2-x2*y1;
67   a22:=y3-y4;
68   b22:=x4-x3;
69   c22:=x3*y4-x4*y3;
```

```

70
71 if equal(a22*b11-a11*b22,0) then begin
72 intersect:=false;
73 exit;
74 end
75 else begin
76 xi:=(b22*c11-b11*c22)/(a22*b11-a11*b22);
77 yi:=(a11*c22-a22*c11)/(a22*b11-a11*b22);
78 intersect:=intre(x1,x2,xi) and intre(x3,x4,xi) and
79 intre(y1,y2,yi) and intre(y3,y4,yi);
80 end;
81 end;
82
83
84 procedure sorti(li,ls:integer);
85 begin
86 if (ls-li=1) then begin
87 if (pint.co[li].x>pint.co[ls].x) or
88 (equal(pint.co[li].x,pint.co[ls].x) and (pint.co[li].y>pint.co[ls].y))
89 then begin
90 pt:=pint.co[li];
91 pint.co[li]:=pint.co[ls];
92 pint.co[ls]:=pt;
93 end;
94 end
95 else if (ls-li>1) then begin
96 sorti(li,(li+ls) div 2);
97 sorti((li+ls) div 2+1,ls);
98 i:=li;
99 m:=(li+ls) div 2;
100 j:=m+1;
101 co:=li-1;
102 while (i<=m) and (j<=ls) do begin
103 inc(co);
104 if (pint.co[i].x<pint.co[j].x) or
105 (equal(pint.co[i].x,pint.co[j].x) and (pint.co[i].y<pint.co[j].y))
106 then begin
107 aux.co[co]:=pint.co[i];
108 inc(i);
109 end
110 else begin
111 aux.co[co]:=pint.co[j];
112 inc(j);
113 end;
114 end;
115 end;
116 if (i<=m) then begin
117 for j:=i to m do begin
118 inc(co);
119 aux.co[co]:=pint.co[j];
120 end;
121 end
122 else begin
123 for i:=j to ls do begin
124 inc(co);
125 aux.co[co]:=pint.co[i];
126 end;
127 end;
128 for i:=li to ls do pint.co[i]:=aux.co[i];
129 end;
130 end;
131
132 begin
133 assign(f,'tezaur.in');
134 reset(f);
135 readln(f,n);
136 for k:=1 to n do begin
137 readln(f,pol[k].np);
138 for i:=1 to pol[k].np do read(f,pol[k].co[i].x,pol[k].co[i].y);
139 pol[k].co[pol[k].np+1]:=pol[k].co[1];
140 end;
141 close(f);
142
143
144 inters.np:=pol[1].np;
145 for i:=1 to pol[1].np+1 do

```

```

146 inters.co[i]:=pol[1].co[i];
147
148
149 assign(f,'tezaur.out');
150 rewrite(f);
151
152 for k:=2 to n do begin
153 { ----- }
154 pint.np:=0;
155
156 { 1. Determina punctele de intersectie }
157
158 for i:=1 to pol[k].np do
159 for j:=1 to inters.np do
160 if intersect(pol[k].co[i].x,pol[k].co[i].y,pol[k].co[i+1].x,pol[k].co[i+1].y,
161 inters.co[j].x,inters.co[j].y,inters.co[j+1].x,inters.co[j+1].y,xii,yii)
162 then begin
163 inc(pint.np);
164 pint.co[pint.np].x:=xii;
165 pint.co[pint.np].y:=yii;
166 end;
167
168 { 2. Determina punctele din pol[k], care sunt in interiorul
169   lui inters }
170 for i:=1 to pol[k].np do begin
171 inside:=true;
172 for j:=1 to inters.np do
173 if (inters.co[j].x<>inters.co[j+1].x)
174 or (inters.co[j].y<>inters.co[j+1].y)
175 then begin
176 iaux:=j+1;
177 a:=inters.co[j].y-inters.co[j+1].y;
178 b:=inters.co[j+1].x-inters.co[j].x;
179 c:=inters.co[j].x*inters.co[j+1].y-inters.co[j+1].x*inters.co[j].y;
180 repeat
181 inc(iaux);
182 if iaux>inters.np then iaux:=1;
183 smn:=semn(a,b,c,inters.co[iaux].x,inters.co[iaux].y);
184 until smn<>0;
185
186 smn1:=semn(a,b,c,pol[k].co[i].x,pol[k].co[i].y);
187 if smn1<>smn then begin
188 inside:=false;
189 break;
190 end;
191 end;
192 if inside then begin
193 inc(pint.np);
194 pint.co[pint.np]:=pol[k].co[i];
195 end;
196 end;
197
198 { 3. Determina punctele din inters, care sunt in interiorul
199   lui pol[k] }
200
201 for i:=1 to inters.np do begin
202 inside:=true;
203 for j:=1 to pol[k].np do
204 if (pol[k].co[j].x<>pol[j].co[j+1].x) or
205 (pol[k].co[j].y<>pol[k].co[j+1].y)
206 then begin
207 iaux:=j+1;
208 a:=pol[k].co[j].y-pol[k].co[j+1].y;
209 b:=pol[k].co[j+1].x-pol[k].co[j].x;
210 c:=pol[k].co[j].x*pol[k].co[j+1].y-pol[k].co[j+1].x*pol[k].co[j].y;
211 repeat
212 inc(iaux);
213 if iaux>pol[k].np then iaux:=1;
214 smn:=semn(a,b,c,pol[k].co[iaux].x,pol[k].co[iaux].y);
215 until smn<>0;
216
217 smn1:=semn(a,b,c,inters.co[i].x,inters.co[i].y);
218 if smn1<>smn then begin
219 inside:=false;
220 break;
221 end;

```

```

222                     end;
223 if inside then begin
224 inc(pint.np);
225 pint.co[pint.np]:=inters.co[i];
226         end;
227         end;
228
229 { 4. Realizeaza infasuratoarea convexa a acestor puncte -> rezultatul
230 este trecut in poligonul inters }
231
232 sorti(1,pint.np);
233 fillchar(select,sizeof(select),false);
234 st[1]:=1;
235 st[2]:=2;
236 select[2]:=true;
237 i:=2;
238
239 for p:=3 to pint.np do
240 begin
241 repeat
242 if (i=1) then begin
243 smn:=1;
244         end
245 else begin
246 a:=pint.co[st[i-1]].y-pint.co[st[i]].y;
247 b:=pint.co[st[i]].x-pint.co[st[i-1]].x;
248 c:=pint.co[st[i-1]].x*pint.co[st[i]].y-pint.co[st[i]].x*pint.co[st[i-1]].y;
249 smn:=semn(a,b,c,pint.co[p].x,pint.co[p].y);
250 if smn<0 then begin
251 select[st[i]]:=false;
252 dec(i);
253         end;
254         end;
255 until smn>=0;
256 inc(i);
257 st[i]:=p;
258 select[p]:=true;
259 end;
260
261 for p:=pint.np downto 1 do
262 if not select[p] then
263 begin
264 repeat
265 if (i=1) then begin
266 smn:=1;
267         end
268 else begin
269 a:=pint.co[st[i-1]].y-pint.co[st[i]].y;
270 b:=pint.co[st[i]].x-pint.co[st[i-1]].x;
271 c:=pint.co[st[i-1]].x*pint.co[st[i]].y-pint.co[st[i]].x*pint.co[st[i-1]].y;
272 smn:=semn(a,b,c,pint.co[p].x,pint.co[p].y);
273 if smn<0 then begin
274 select[st[i]]:=false;
275 dec(i);
276         end;
277         end;
278 until smn>=0;
279 inc(i);
280 st[i]:=p;
281 select[p]:=true;
282 end;
283
284 if pint.np=0 then begin
285 writeln(f,0);
286 close(f);
287 halt;
288         end;
289
290 inters.np:=i-1;
291 for p:=1 to i do
292 inters.co[p]:=pint.co[st[p]];
293
294 { ----- }
295         end;
296
297 arie:=0;

```

```
298
299  for i:=1 to inters.np do
300    arie:=arie+inters.co[i].x*inters.co[i+1].y-
301      inters.co[i+1].x*inters.co[i].y;
302
303  arie:=abs(arie)/2;
304
305  writeln(f,arie:0:2);
306  close(f);
307
308 end.
```

38.6.3 *Rezolvare detaliată

Index

- (x«8)|y, 134
#define, 189, 374, 816, 826
 #define, 374
 f first, 189
 mp make_pair, 189
 pb push_back, 189
 pii pair<int,int>, 189
 s second, 189
#include <bits/stdc++.h>, 43
închidem fișierul, 503
 citim din nou, 503
 deschidem din nou, 503
înfășurătoarea convexă, 885
^, 662
şir periodic, 786
şirul sumelor parțiale, 741
stergere, 149
0x3f3f3f3f, 340
1«16, 134
1LL, 155
64 de biți cu semn, 675
parcursere Euler, 789

accumulate, 4
adâncimea arborelui, 21
AIB, 380
algoritm de flux maxim, 866
algoritm determinist, 562
algoritmul Aho-Corasick, 802
algoritmul Bellman-Ford, 117, 796
algoritmul Bellman-Ford-Moore, 843
algoritmul de factorizare Pollard-Rho, 786
algoritmul LexBFS, 817
algoritmul lui Bellman Ford cu coadă, 183
algoritmul lui Dijkstra, 117, 183, 843
algoritmul lui Floyd-Warshall, 183
algoritmul lui Kruskal, 63, 290, 835
algoritmul lui Lee, 132
algoritmul lui Mo, 380
algoritmul lui Prim, 290, 309
algoritmul MCS, 817
algoritmul Roy-Floyd, 859
and, 340
APM, 70, 308
 Arbore Parțial de Cost Minim, 308
APM(), 77
aranjamente cu repetiție, 220
arbore, 19, 225, 438, 577
 rădăcina, 577
 strămoș, 577
arbore AVL, 214
arbore binar, 150
arbore binar echilibrat, 214
arbore de căutare, 880
arbore de intervale, 37, 284, 411, 446, 579, 732, 778
arbore de prefixe, 832
arbore de sufixe, 802
arbore echilibrat, 578, 778
 set, 578
arbore echilibrat de căutare, 578
arbore indexat binar, 446, 578
arbore parțial, 308
arbore parțial de cost minim, 62, 97, 290
arbore de căutare, 225
arbori de intervale, 380, 485, 789, 816
arce, 710
assign, 693, 697
auto, 8, 347, 374
auto &, 304, 332, 374
auto &&, 304

back(), 9, 499
backtracking, 105, 193, 196, 213, 220, 245, 293, 503, 619, 684, 796, 871, 892
baleiere, 45
begin(), 188, 189, 608, 697, 730, 739
Bellman-Ford, 186
BF, 755
BFS, 563
bfs, 8
 lambda, 8
 queue, 8
binary_search, 55
bitset, 66, 70, 77, 186, 188
bool, 84
bool operator, 499, 739
bool operator(), 492
Boyer-Moore, 802
brute-force, 380
buchete, 149
build_ai, 419
build_hash(), 743

căutare în lățime, 796
căutare binară, 11, 149, 166, 213, 473, 485, 568, 578, 675, 707, 747, 769, 802,

- 880, 893, 914
 căutarea binară, 213, 832
 centru de simetrie, 882
 cerr, 347
 ciclu, 143, 451
 ciclu eulerian, 132, 681
 ciclu hamiltonian special, 328
 ciclu simplu, 467, 681
 cin.sync_with_stdio, 398
 cin.sync_with_stdio(false);, 286
 circuit, 143
 citire parsata, 573, 574
 class, 39, 288, 492, 499
 clear(), 9, 316, 326, 497, 563, 728
 clică, 300
 clock(), 421, 757
 CLOCKS_PER_SEC, 421, 757
 coadă, 44, 117, 533, 755, 786
 componentă conexă, 45, 643, 769
 componente biconexe, 300, 835, 982
 componente conexe, 3, 438, 694, 835
 componente tare conexe
 graful aciclic, 609
 componente conexe, 3
 const auto&, 39
 const char, 159, 188, 693
 const int, 398
 INF = 1 « 29, 398
 copy, 42
 count, 347
 cozi de priorități, 309
 cpmponente conexe, 97
 Critical Path Method, 207
 cuplaj, 330, 759
 cuplaj maxim de cost minim, 796
 de la dreapta la stânga, 490
 decltype, 4
 delete, 170
 deque, 121, 126, 747, 763
 pair, 126
 DFS, 20, 66, 70, 77
 dfs, 6, 9, 332, 413, 579
 diametrul grafului, 562
 Dijkstra, 188, 257
 dijkstra, 189
 Dijkstra cu heap-uri, 849
 dijkstra(), 119
 dinamică înainte, 139
 distanța Manhattan, 10
 Divide-Et-Impera, 380
 double, 709
 double-ended queue, 763
 drum, 643, 710
 drum alternant de creștere, 796
 emplace_back, 304
 empty(), 119, 188, 332, 499, 563
 end(), 574
 end(), 115, 188, 608, 697, 729, 730, 739
 enum, 4
 erase, 43, 347, 492, 497, 499, 608
 exit, 585
 exponentierea rapidă, 620
 ExtendedGCD, 627
 false, 697
 fflush, 712, 717
 fflush(stdout);, 662
 fill, 22, 337
 find, 115, 163, 729
 first, 188, 574, 729
 floatfield, 313
 for(;scanf(
 for(d=D;*d;d++), 134
 for(L=0;;), 151
 formă canonica, 451
 formă scară, 451
 formula lui Stirling, 805
 front(), 332, 563
 frunză, 330
 frunză în arbore, 309
 funcție liniară și monotonă, 706
 get(), 4
 graf, 577
 aciclic, 577
 conex, 577
 neorientat, 577
 graf bipartit, 44, 759, 796
 graf complet, 300, 562
 graf conex, 300, 438
 aciclic, 438
 graf dens, 309
 graf eulerian, 681
 graf neorientat, 643, 694, 755
 graf neorientat complet, 290, 308
 graf neorientat conex, 562
 graf orientat, 608, 710
 tare conex, 710
 graf orientat aciclic, 843
 graf parțial, 710
 graful asociat matricii, 769
 graful transpus, 610
 Graph &, 337
 greedy, 193, 205, 225, 485, 629, 892
 grijă la limita de memorie, 786
 heap, 37, 117, 225, 490, 533, 675, 763, 843,
 893
 heapsort, 914
 heavy-path decomposition, 578
 i64, 155
 ifstream cin, 304
 INF = 0x3F3F3F3F, 186
 INF = 0x3f3f3f3f;, 288
 inf = 1 « 30;, 286
 init_hash(), 743
 inline, 159, 189, 413, 421, 492, 728
 inserare, 149
 insert, 9, 43, 115, 347, 492, 497, 574

- int64_t, 16, 35, 39, 288
interclasare, 214, 302, 601, 769
Inv, 627
invers modular, 620, 822
inversul modular, 432

join, 4

KMP, 636
Knapsack, 700
Knuth-Morris-Pratt, 802
Kosaraju's algorithm, 714, 715
Kruskal, 291

lambda, 304, 347
lanț, 3, 19, 20, 233, 299, 309, 329, 899
lanț elementar, 313
lanț simplu, 467, 681
Lee, 769
linearizarea matricei, 105
liniarizare a arborelui, 579
liniarizare arbore, 700
listă de adiacență, 711
listă de priorități, 533
lista de adiacență, 785
lower_bound, 43, 574, 578, 730

măști pe biți, 183
make_pair, 32, 188, 574, 678, 693
make_tuple, 43, 77
map, 40, 115, 348, 419, 497, 728, 729
matricea costurilor, 97
max-heap, 344
max_element, 4, 8
mediana unui sir, 778
memoria disponibilă, 786
memset, 186, 190, 316, 413, 627, 729, 730
merge sort, 303
metodă constructivă, 149
metoda Divide et Impera, 568
metoda Greedy, 205
metoda includerii și excluderii, 18
metoda programării dinamice, 139, 149, 179, 183, 205, 210, 349, 670, 793
 metoda înainte, 800
metoda programării dinamice "înainte", 166
modulo, 21
muchie de întoarcere, 695
multiset, 492

new, 79, 170
next_permutation, 190, 635
nod critic, 300
nod izolat, 643
noduri izolate, 3
normalizare valori, 724
normalize(), 729
nth_element, 513, 784
NULL, 79, 728
numere mari, 871
numerele lui Stirling, 620
 de speță a două, 620

ofstream cout, 304
operator, 39, 625
ordine lexicografică, 105, 149
păduri de mulțimi disjuncte, 63, 533
pair, 32, 39, 186, 188, 348, 693, 697, 709, 729
parcursere în adâncime, 3, 210, 689, 711, 789
parcursere în lățime, 3, 233, 562, 755, 769
parcursere BF, 755
parcursere BFS, 117
parcursere DF, 473, 479, 694, 876
parcursere DFS, 117, 330, 578
parcursere Euleriana, 410
perechi ordonate, 710
permutări, 184
pop_back(), 119, 499
pop_heap, 119
Power, 627
precalculare, 468
precision, 313
prefix, 149, 636
prefix de suma maxima, 579
Prim, 292
principiul includerii și al excluderii, 213
principiul includerii și excluderii, 11, 294, 620, 822
priority_queue, 70, 188, 533, 536, 678
 push, 678
 top(), 678
problema deschisă, 562
problema rucsacului, 700, 825
problema spectacolelor, 205
programare dinamică, 86, 105, 196, 219, 314, 467, 524, 720, 747, 873, 925
progresie geometrică, 110
punct de articulație, 694
puncte critice, 694
puncte critice în graf, 982
puncte de articulatie, 695
push, 188, 563
push_back, 119, 499, 563, 693, 697, 730, 739
push_heap, 119

query_ai, 419
queue, 186, 332, 563, 718, 757
 empty(), 186
 front(), 186
quickSort, 82
quicksort, 303, 914

Rabin-Karp, 636
radix, 164
radix sort, 303
radix-sort, 149
radixsort, 914
rbegin(), 499
rețea de flux, 988
read_int(), 523
recurență, 210
recurență înainte, 20
recurență liniară, 110

resize, 186, 190, 693, 697, 698, 730
ridicarea la putere în timp logaritmic, 110,
 111
RMQ, 411
root, 4, 70

second, 188, 574, 729
SegmentTree, 288
set, 39, 40, 43, 115, 117, 161, 189, 344, 347,
 492, 497, 499, 533, 574, 578, 695,
 778
 clear(), 189
 end(), 739
 erase, 39, 189
 find, 739
 insert, 39, 189, 739
 lower_bound, 578
set_intersection, 608
setf, 313
setprecision, 709
size(), 189, 332, 563, 608, 693, 697
size_t, 155
sizeof, 316, 729, 730
Smenul lui Mars (Marius Andrei), 552
soluție exponentială, 467
sort, 42, 84, 190, 398, 492, 697, 698, 730, 739
sortare, 490
sortare topologică, 207, 609, 988
sortare(), 74, 76
sqrtl, 313
srand, 757
stable_partition, 304
stack, 32, 697
statistici de ordine, 503
stivă, 45, 473, 629, 732
STL, 778
strategie greedy, 490
string, 163
string::npos, 163
struct, 79, 84, 170, 625, 739
structură de date, 149, 210
structuri de date, 284

subșir crescător maximal, 166
subarbore, 700
substr, 163
sufix, 636
sumă de numere impare, 11
sume parțiale, 295, 321, 379
sume partiale, 411
swap, 16, 332, 608, 709
sync_with_stdio, 316

tăietură minimă în graf, 866
tehnica lui Mars, 552
tehnica programării dinamice, 20
teorema Fermat, 112
teorema lui Euler (pentru ciclu eulerian),
 132
tie, 313
timp logaritmic, 149
tri, 313
triunghiul Serpinski, 432
true, 693
tuple, 43, 77
 tie, 77
typedef, 693

u64, 159
unique, 698, 730
unique_ptr, 4
 reset, 4
unordered_map, 413, 419
unsigned, 698
upper_bound, 578

vector
 back(), 190
 front(), 190
vector de frecvențe, 295, 601, 724, 741
vector de predecesori, 210
vector de tați, 636
vertex cover, 344

x»8, 134
x&255, 134

Bibliografie

- [1] Aho, A., Hopcroft, J., Ullman, J.D.; Data strutures and algorithms, Addison Wesley, 1983
- [2] Andreica M.I.; Elemente de algoritmica - 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., Șerban, M.; Informatică - manual pentru clasa a X-a., Ed. Polirom, 2000
- [8] Cerchez, E.; Informatică - Culegere de probleme pentru liceu, Ed. Polirom, 2002
- [9] Cerchez, E., Șerban, M.; Programarea în limbajul C/C++ pentru liceu, Ed. Polirom, 2005
- [10] Cori, R.; Lévy, J.J.; Algorithmes et Programmation, Polycopié, version 1.6; <http://w3.edu.polytechnique.fr/informatique/>
- [11] Cormen, T.H., Leiserson C.E., Rivest, R.L.; Introducere în Algoritmi, Ed Agora, 2000
- [12] Cormen, T.H., Leiserson C.E., Rivest, R.L.; Pseudo-Code Language, 1994
- [13] Cristea, V.; Giumale, C.; Kalisz, E.; Paunoiu, Al.; Limbajul C standard, Ed. Teora, București, 1992
- [14] Erickson J.; Combinatorial Algorithms; <http://www.uiuc.edu/~jeffe/>
- [15] Flanagan, D.; Java in a Nutshell, O'Reilly, 1997.
- [16] Giumale C., Negreanu L., Călinoiu S.; Proiectarea și analiza algoritmilor. Algoritmi de sortare, Ed. All, 1997
- [17] Halim S., Halim F., Competitive programming, 2013
- [18] Knuth, D.E.; Arta programării calculatoarelor, vol. 1: Algoritmi fundamentali, Ed. Teora, 1999.
- [19] Knuth, D.E.; Arta programării calculatoarelor, vol. 2: Algoritmi seminumerici, Ed. Teora, 2000.
- [20] Knuth, D.E.; Arta programării calculatoarelor, vol. 3: Sortare și căutare, Ed. Teora, 2001.
- [21] Knuth, D.E.; The art of computer programming, vol. 4A: Combinatorial algorithms, Part 1, Addison Wesley, 2011.
- [22] Lambert,K. A., Osborne,M.; Java. A Framework for Programming and Problem Solving, PWS Publishing, 1999
- [23] Laaksonen A.; Guide to competitive programming, Springer, 2017
- [24] Livovschi, L.; Georgescu H.; Analiza și sinteza algoritmilor. Ed. Enciclopedică, București, 1986.
- [25] Niemeyer, P., Peck J.; Exploring Java, O'Reilly, 1997.

- [26] Odăgescu, I., Smeureanu, I., Ștefănescu, I.; Programarea avansată a calculatoarelor personale, Ed. Militară, București 1993
- [27] Odăgescu, I.; Metode și tehnici de programare, Ed. Computer Lobris Agora, Cluj, 1998
- [28] Popescu Anastasiu, D.; Puncte de articulație și punți în grafuri, Gazeta de Informatică nr. 5/1993
- [29] Răbâea, A.; https://math.univ-ovidius.ro/Doc/Admitere/CentruPregatire/2007/Info/Lista_probleme_2000-2007.pdf
- [30] Răbâea, A.; https://math.univ-ovidius.ro/Doc/Admitere/CentruPregatire/2007/Info/Rezolvare_C09.pdf
- [31] Răbâea, A.; https://math.univ-ovidius.ro/Doc/Admitere/CentruPregatire/2007/Info/Rezolvare_C10.pdf
- [32] Răbâea, A.; https://math.univ-ovidius.ro/Doc/Admitere/CentruPregatire/2007/Info/Rezolvare_C11.pdf
- [33] Răbâea, A.; https://math.univ-ovidius.ro/Doc/Admitere/CentruPregatire/2007/Info/Rezolvare_Baraj.pdf
- [34] Skiena S.S., Revilla M.A.; Programming challenges - The Programming Contest Training Manual, Springer, 2003
- [35] Tomescu, I.; Probleme de combinatorică și teoria grafurilor, Editura Didactică și Pedagogică, București, 1981
- [36] Tomescu, I.; Leu, A.; Matematică aplicată în tehnica de calcul, Editura Didactică și Pedagogică, București, 1982
- [37] Tudor, S.; Informatică - profilul real intensiv, varianta C++; Editura L&S, București, 2004
- [38] Tudor, S.; Hutanu, V.; Informatică intensiv; Editura L&S, București, 2006
- [39] Văduva, C.M.; Programarea in JAVA. Microinformatica, 1999
- [40] Vișinescu, R.; Vișinescu, V.; Programare dinamică - teorie și aplicații; GInfo nr. 15/4 2005
- [41] Vlada, M.; Conceptul de algoritm - abordare modernă, GInfo, 13/2,3 2003
- [42] Vlada, M.; Grafuri neorientate și aplicații. Gazeta de Informatică, 1993
- [43] Weis, M.A.; Data structures and Algorithm Analysis, Ed. The Benjamin/Cummings Publishing Company. Inc., Redwoods City, California, 1995.
- [44] Winston, P.H., Narasimhan, S.; On to JAVA, Addison-Wesley, 1996
- [45] Wirth N.; Algorithms + Data Structures = Programs, Prentice Hall, Inc 1976
- [46] *** - Gazeta de Informatică, Editura Libris, 1991-2005
- [47] *** - https://github.com/DinuCr/CS/blob/master/Info/stuff%20stuff/Rezolvare_C09.pdf
- [48] *** - https://dokumen.tips/documents/rezolvare_c09.html
- [49] *** - <https://www.scribd.com/doc/266218102/Rezolvare-C09>
- [50] *** - <https://www.scribd.com/document/396362669/Rezolvare-C10>
- [51] *** - <https://www.scribd.com/document/344769195/Rezolvare-C11>
- [52] *** - <https://www.scribd.com/document/364077679/Rezolvare-C11-pdf>
- [53] *** - <https://needoc.net/rezolvare-c11-pdf>
- [54] *** - <https://vdocumente.com/algoritmi-i-structuri-de-date.html>

Lista autorilor

problemelor și indicațiilor

- Adrian Diaconu, 763, 793
Adrian Panaete, 15, 30, 53, 110, 138, 149,
297, 387, 534, 549, 609, 633, 670
Airinei Adrian, 725, 732, 756
Alex Tatomir, 295
Alexandru Cazacu, 609
Alin Burța, 796
Andrei Ciocan, 524, 545
Andrei Costin Constantinescu, 299, 308, 324
Andrei Grigorean, 724, 726, 737, 759, 778
Andrei Heidelbacher, 440, 451, 497
Andrei Pârvu, 503, 578, 596, 601, 636, 637,
675, 681, 700, 702
Banu Denis, 293
Bogdan Ciobanu, 10, 12, 21, 339
Bogdan Cristian Tătăroiu, 514, 563, 574,
598, 602, 619, 642, 658, 675, 700
Cătălin Ștefan Tiseanu, 763, 766, 783
Carmen Mincă, 179, 205, 800
Carmen Popescu, 694
Constantin Gălățan, 196, 201, 205, 747
Constantinescu Andrei Costin, 284
Cosmin Gheorghe, 741
Cosmin Tutunaru, 649, 689, 690
Dan Pracsiu, 166, 871
Daniel Păsăilă, 812, 814
Denis-Gabriel Mită, 439, 445
Doru Anastasiu Popescu, 96, 132, 980
Dragoș Alin Rotaru, 619
Dumitru Bogdan, 945
Duta Vlad, 706
Emanuela Cerchez, 193, 249, 899, 940, 982
Emilian Miron, 786, 823
Eugen Nodea, 368
Eugenie-Daniel Posdărăscu, 329, 344, 372,
379, 410, 485, 490
Filip Buruiana, 180, 188
Florin Manea, 763
Gheorghe Cosmin, 720
Ilie Vieru, 219
Ioanda Popa, 210
Ion Maxim, 970
Ionuț Fechete, 789
Irina Dumitrascu, 273
Lucian Bicsi, 344, 349, 365, 372
Lukacs Sandor, 4, 13, 14, 22
Marinel Șerban, 940
Marius Dragus, 716
Marius Stroe, 148, 183, 568, 601, 695
Marius Vlad, 965
Mihai Calancea, 3, 6, 24, 562
Mihai Ciucu, 387, 476, 485
Mihai Pătrașcu, 914, 918
Mihai Popa, 506, 509, 515, 524
Mircea Pașoi, 755, 776
Mugurel Ionuț Andreica, 601, 645, 675, 848,
863, 876, 893, 896, 920, 925
Mureșan Codruța, 62
Nistor Mot, 194, 198
Onicescu Costin Andrei, 290
Pătcaș Csaba, 198, 207, 764, 782, 793, 802
Paul - Dan Băltescu, 769
Perticas Catalin, 703
Puni Andrei Paul, 711, 748
Răzvan Dan Sălăjan, 344, 396, 479
Radu Ștefan, 958
Radu Andrei Ștefan, 873
Radu Berinde, 859, 865
Radu Muntean, 8, 20, 25, 26
Radu Voroneanu, 485, 533, 539, 552
Radu-Eugen Boriga, 670
Rodica Pintea, 925, 987
Savin Tiberiu, 706, 723
Stefan Ciobăcă, 817
Stelian Ciurea, 210, 214, 220, 256, 805, 812
Szabo Zoltan, 17, 27, 86, 105, 143, 451, 684
Tamio-Vesa Nakajima, 302, 314
Tiberiu Florea, 807, 810, 812
Tiberiu Savin, 662, 675, 681, 682
Vlad Duță, 629
Vlad Gavrilă, 117, 410, 467, 533, 541, 553
Vlad Tudose, 671