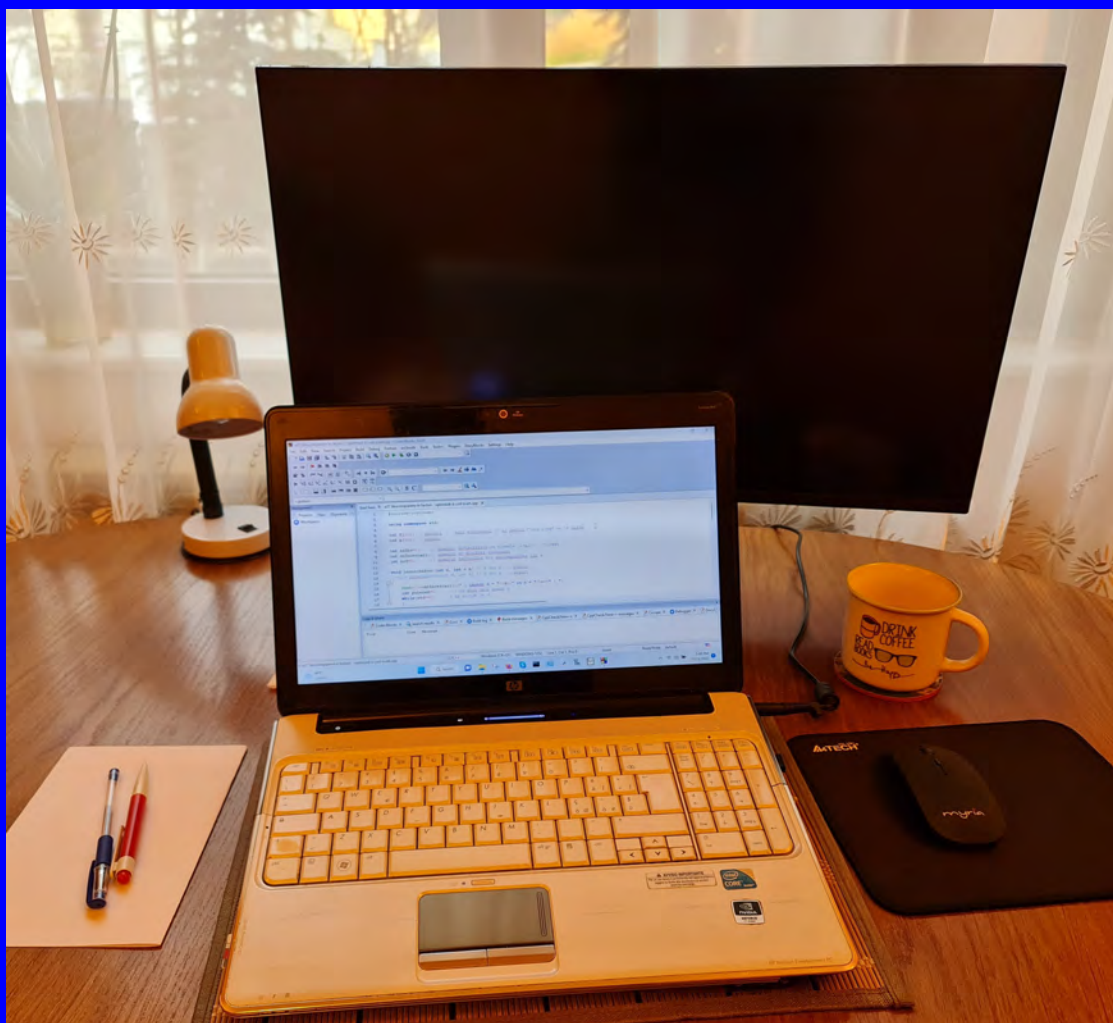


---

# INFORMATICĂ

## *Olimpiada - cls10*

---



# PROBLEME DE INFORMATICĂ

*date la olimpiade*

OJI

*în*

\*\*\*\*\*  
2019 2018 2017 2016 2015  
2014 2013 2012 2011 2010  
2009 2008 2007 2006 2005  
2004 2033 2002 \*\*\*\*\*

... draft (ciornă) ...  
\*\*\* Nobody is perfect \*\*\*



Adrian Răbăea, *Ph.D.*

\*\*\* [https://en.wikipedia.org/wiki/Saint\\_Stephen#Eastern\\_Christianity](https://en.wikipedia.org/wiki/Saint_Stephen#Eastern_Christianity) \*\*\*

2022-12-27



# Dedication

I would like to dedicate this book ...

- "to **myself**" <sup>1</sup> ...

in a time when <sup>2</sup> ...

I will not be able ... "to be" <sup>3</sup>

That is because ...

"When I Die Nobody Will **Remember** Me" <sup>4</sup>

- to **people** impacted by the book

- to my nephew **Adam**.

( 😊 in ascending order! 😊 )

---

<sup>1</sup> <https://www.femalefirst.co.uk/books/carol-lynne-fighter-1034048.html>

<sup>2</sup> <https://otiliaromea.bandcamp.com/track/dor-de-el>

<sup>3</sup> [https://en.wikipedia.org/wiki/To\\_be,\\_or\\_not\\_to\\_be](https://en.wikipedia.org/wiki/To_be,_or_not_to_be)

<sup>4</sup> <https://www.youtube.com/watch?v=eMtcDkSh7fU>



# Prefață

Stilul acestor cărți este ... ca și cum aş vorbi cu nepoții mei (și chiar cu mine însumi!) ... 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 răs!" 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!.

Sunt convins că este important să studiem cu atenție cât mai multe probleme *rezolvate*<sup>5</sup>. Cred că sunt utile și primele versiuni ale acestor cărți ... în care sunt prezentate numai enunțurile și indicațiile "oficiale" de rezolvare. Acestea se găsesc în multe locuri; aici încerc să le pun pe "*toate la un loc*"! Fiecare urcă spre vârf ... cât poate! Sunt multe poteci care duc spre vârf iar această carte este ... una dintre ele!

Limbaajul 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, IOI2020 și IOI2019<sup>6</sup>, IOI2015<sup>7</sup>.

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 "restricțiile" care apar acolo (ele sigur "ascund" ceva interesant din punct de vedere al algoritmului de rezolvare!)<sup>8</sup>.

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" ([48] - [56])! Încerc acum să ajung acolo unde am rămas ... plecând mereu din prezent ... până când nu va mai fi posibil ... așa că, de această dată, anii sunt în ordine ... descrescătoare! :-)

"*Codurile sursă*" sunt cele "oficiale" (publicate pe site-urile olimpiadelor) sau publicate pe alte site-uri (dacă mi s-a părut că sunt utile și se poate învăța câte ceva din ele).

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<sup>9</sup> folosite la olimpiade înainte de 2010 erau ceva mai '*slabe*' și ... restricțiile de memorie, din enunțurile problemelor, par '*ciudate*' acum!). 😊

În perioada 2017-2020 cele mai puternice calculatoare din lume au fost: în noiembrie 2017 în **China**, în noiembrie 2019 în **SUA** și în iunie 2020 în **Japonia**. În iunie 2022<sup>10</sup> "Frontier"

<sup>5</sup> Se poate observa din "*Coduri sursă*" că orice problemă are numeroase soluții, atât ca algoritmi de rezolvare cât și ca stil de programare! Studiind aceste coduri ... avem ce învăța ... deși uneori pare că '*se trage cu tunul*' ...

<sup>6</sup> IOI2019 și IOI2020 au permis utilizarea limbajelor de programare C++ și Java

<sup>7</sup> IOI2015 a permis utilizarea limbajelor de programare C++, Java, Pascal, Python și Rubi (...)

<sup>8</sup> Vezi cele 5 secunde pentru **Timp maxim** de executare/test din problema "avârcolaci" - ONI2014 clasa a 11-a

<sup>9</sup> <https://en.wikipedia.org/wiki/Computer>

<sup>10</sup> <https://www.top500.org/lists/top500/2022/06/>

depășește pragul ”**exascale**”! (1.102 Exaflop/s).<sup>11</sup> ”Peta” a fost depășit în 2008, ”tera” în 1997, ”giga” în 1972.<sup>12</sup> Pentru ce a fost mai înainte, vezi [https://en.wikipedia.org/wiki/List\\_of\\_fastest\\_computers](https://en.wikipedia.org/wiki/List_of_fastest_computers).

O mică observație: în 2017 a fost prima ediție a olimpiadei **EJOI**<sup>13</sup> în Bulgaria și ... tot în Bulgaria a fost și prima ediție a olimpiadei **IOI** în 1989.<sup>14</sup> Dar ... prima ediție a olimpiadei **IMO** (International Mathematical Olympiad) a fost în România în 1959.<sup>15</sup> Tot în România s-au ținut edițiile din anii 1960, 1969, 1978, 1999 și 2018. Prima ediție a olimpiadei **BOI** (Balkan Olympiad in Informatics) a fost în România în 1993 la Constanța. Prima ediție a olimpiadei **CEOI** (Central-European Olympiad in Informatics) a fost în România în 1994 la Cluj-Napoca.

Revenind la ... ”*culegerile noastre*” ... mai departe, probabil, va urma completarea unor informații în ”*Rezolvări detaliate*” ... 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!). Acum, în martie 2022, am început și redactarea unei culegeri de probleme date la bacalaureat în ultimii câțiva ani (câți voi putea!).

Îmi aduc aminte că exista o interesantă *vorba de duh* printre programatorii din generația mea: ”nu se *trage cu tunul* într-o muscă” . 😊 Sensul este: nu se scrie un *cod complicat* dacă se poate scrie un cod simplu și clar! Asta încerc eu în ”*Rezolvări detaliate*”.

Vom încerca, împreună, 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! (asta așa, ca să vedem cum sunt problemele la acest nivel!). Cei care ajung acolo sau vor să ajungă acolo (la IOI) *sigur* nu au nevoie de ajutorul meu! Se descurcă singuri!

”*ALGORITMI utili la olimpiadele de informatică*”, separat pentru gimnaziu și liceu, sper să fie de folos, așa cum cred că sunt [1] - [28], [34] - [47], [57] - [83], ... și multe alte cărți și site-uri!. Ar fi interesant *să descoperim* noi înșine cât mai mulți algoritmi ... în loc *să-i învățăm* pur și simplu!

O altă mică observație: ce am strâns și am scris în aceste cărți se adresează celor *interesați de aceste teme*! Nu cărcotașilor! Sunt evidente *sursele* ”de pe net” (și *locurile* în care au fost folosite) așa că nu sunt necesare ”ghilimele anti-plagiat”, referințe și precizări suplimentare la fiecare pas!

Și un ultim gând: criticile și sfaturile sunt utile dacă au valoare! Dacă sunt numai așa ... cum critică lumea la un meci de fotbal ... sau cum, pe bancă în parc, ”își dă cu părerea” despre rezolvarea problemelor economice ale țării ... atunci ... !!!

”I’m only responsible for what I say, not for what you understand.”<sup>16</sup>

Adrese interesante (rezultatele elevilor români):

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

Adresele acestor cursuri:

<https://www.scribd.com/user/550183580/Adrian-Răbăea>  
<https://www.scribd.com/user/552245048/Adi-Rabaea>  
<https://drive.google.com/drive/folders/1hC5PZuslCdS95s137SW46H-qy59GRDGZ>

Adrese utile (programe școlare):

[http://programe.ise.ro/Portals/1/Curriculum/Progr\\_Lic/TH/Informatica\\_teoretic\\_vocational\\_intensiv\\_clasa%20a%20IX-a.pdf](http://programe.ise.ro/Portals/1/Curriculum/Progr_Lic/TH/Informatica_teoretic_vocational_intensiv_clasa%20a%20IX-a.pdf)  
[http://programe.ise.ro/Portals/1/Curriculum/Progr\\_Lic/TH/Informatica\\_teoretic\\_vocational\\_intensiv\\_clasa%20a%20X-a.pdf](http://programe.ise.ro/Portals/1/Curriculum/Progr_Lic/TH/Informatica_teoretic_vocational_intensiv_clasa%20a%20X-a.pdf)  
[http://programe.ise.ro/Portals/1/Curriculum/Progr\\_Lic/TH/Informatica\\_teoretic\\_vocational\\_intensiv\\_clasa%20a%20XI-a.pdf](http://programe.ise.ro/Portals/1/Curriculum/Progr_Lic/TH/Informatica_teoretic_vocational_intensiv_clasa%20a%20XI-a.pdf)

Bistrița, 27th December 2022

Adrian Răbăea

<sup>11</sup> [https://en.wikipedia.org/wiki/Metric\\_prefix/](https://en.wikipedia.org/wiki/Metric_prefix/)

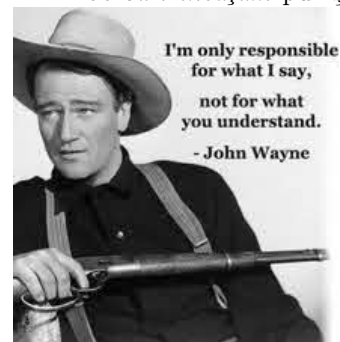
<sup>12</sup> [https://en.wikipedia.org/wiki/Computer\\_performance\\_by\\_orders\\_of\\_magnitude](https://en.wikipedia.org/wiki/Computer_performance_by_orders_of_magnitude)

<sup>13</sup> <https://ejoi.org/about/>

<sup>14</sup> <https://stats.ioinformatics.org/olympiads/>

<sup>15</sup> [https://en.wikipedia.org/wiki/International\\_Mathematical\\_Olympiad](https://en.wikipedia.org/wiki/International_Mathematical_Olympiad)

<sup>16</sup> <https://www.facebook.com/johnwayne/photos/a.156450431041410/2645523435467418/?type=3>



# ”Acknowledgements”

”I want to thank God most of all because without God I wouldn’t be able to do any of this.”<sup>17</sup>

Bistrița, 27th December 2022

Adrian R.

---

<sup>17</sup> I.d.k.: ”I don’t know who the author is.”

# Despre autor



nume: Răbăea Aurel-Adrian, 18.03.1953 - ...<sup>18</sup>

telefon: +40 728 zz ll aa +40 363 xx 25 xx

email: adrian1803@gmail.com

Lector universitar - Universitatea Tehnică din Cluj Napoca - Centrul Universitar Nord din Baia Mare, Facultatea de Științe, Str. Victoriei, nr. 76, Baia Mare, România, (pensionat: 01.10.2018)  
<https://stiinte.utcluj.ro/departamente.html>

Discipline predate (1992-2018):

*Algoritmi și structuri de date*, Algoritmi în teoria opțiunilor financiare, Bazele matematice ale calculatoarelor, Bazele tehnologiei informației, Birotică, Capitole speciale de inteligență artificială, Capitole speciale de teoria algoritmilor, *Calcul paralel*, Informatică economică, Instruire asistată de calculator, Limbaje de programare; Programare orientată pe obiecte, Programare procedurală, Structuri de date.

Studii doctorale în informatică economică - Diplomă de doctor (1997-2002):

Instituția: Academia de Studii Economice, București;

Titlul tezei: *Algoritmi paraleli și aplicații pe mașini virtual paralele*<sup>19</sup>

Conducător științific: Prof. dr. ing. Gheorghe Dodescu<sup>20</sup>

Teme studiate: utilizarea algoritmilor paraleli în teoria opțiunilor financiare

Studii de specializare în informatică - Certificat anul V - 'master' (1978-1979):

Instituția: Facultatea de matematică și informatică, București;

Titlul tezei: *Probleme la limită pentru procese cu creșteri independente și aplicații în teoria așteptării*

Conducător științific: Prof. dr. Constantin Tudor<sup>21</sup>

Studii universitare de licență în informatică - Diplomă de licență (1974-1978):

Instituția: Facultatea de matematică și informatică, București;

Titlul tezei: *Metode de comparație multiplă în analiza dispersională*

Conducător științific: Prof. dr. Ion Văduva<sup>22</sup>

Locuri de muncă: (1979-2018):

- (2018-2009) Universitatea Tehnică din Cluj-Napoca, Centrul Universitar Nord din Baia-Mare, Facultatea de Științe, Departamentul de Matematică-Informatică
- (2009-1992) Universitatea "Ovidius" din Constanța, Facultatea de Matematică și Informatică, Departamentul de Informatică<sup>23</sup>
- (1992-1979) Centrul de Informatică și organizare CINOR, București<sup>24</sup>

Olimpiade: (fiind elev la Liceul Militar "Dimitrie Cantemir" - Breaza, PH)<sup>25</sup>

- 1971: Olimpiada Națională de matematică: participare (fără rezultat notabil)
- 1970: Olimpiada Națională de matematică: participare (fără rezultat notabil)

[https://scholar.google.com/citations?user=-sSE\\_lwAAAAJ&hl=en](https://scholar.google.com/citations?user=-sSE_lwAAAAJ&hl=en)

<https://www.scopus.com/authid/detail.uri?origin=resultslist&authorId=56122389200&zone=>

<http://www.facebook.com/adrian.rabaea>

<sup>18</sup> [https://dmi.cunbm.utcluj.ro/?page\\_id=2](https://dmi.cunbm.utcluj.ro/?page_id=2)

<sup>19</sup> [http://opac.biblioteca.ase.ro/opac/bibliographic\\_view/149021](http://opac.biblioteca.ase.ro/opac/bibliographic_view/149021)

<sup>20</sup> <http://www.ionivan.ro/2015-PERSONALITATI/Dodescu.htm>

<sup>21</sup> [http://old.fmi.unibuc.ro/ro/prezentare/promotii/promotia1978informatica\\_10ani/](http://old.fmi.unibuc.ro/ro/prezentare/promotii/promotia1978informatica_10ani/)

<sup>22</sup> [https://ro.wikipedia.org/wiki/Ion\\_V%C4%83duva](https://ro.wikipedia.org/wiki/Ion_V%C4%83duva)

<sup>23</sup> <https://fmi.univ-ovidius.ro/>

<sup>24</sup> <http://c3.cniv.ro/?q=2021/cinor/>

<sup>25</sup> <https://www.cantemircml.ro/>



# Cuprins

<b>Prefață</b>	<b>iii</b>
<b>Cuprins</b>	<b>vii</b>
<b>Lista figurilor</b>	<b>xii</b>
<b>Lista tabelelor</b>	<b>xiv</b>
<b>Lista programelor</b>	<b>xv</b>
<b>1 OJI 2022</b>	<b>1</b>
1.1 circular . . . . .	1
1.1.1 Indicații de rezolvare . . . . .	2
1.1.2 *Cod sursă . . . . .	3
1.1.3 *Rezolvare detaliată . . . . .	3
1.2 pulsar . . . . .	3
1.2.1 Indicații de rezolvare . . . . .	5
1.2.2 *Cod sursă . . . . .	6
1.2.3 *Rezolvare detaliată . . . . .	6
1.3 transport . . . . .	7
1.3.1 Indicații de rezolvare . . . . .	8
1.3.2 *Cod sursă . . . . .	9
1.3.3 *Rezolvare detaliată . . . . .	10
<b>2 OJI 2021 - OSEPI</b>	<b>11</b>
2.1 Labirint . . . . .	11
2.1.1 Indicații de rezolvare . . . . .	12
2.1.2 Cod sursă . . . . .	12
2.1.3 *Rezolvare detaliată . . . . .	12
2.2 SDistanțe . . . . .	12
2.2.1 Indicații de rezolvare . . . . .	14
2.2.2 Cod sursă . . . . .	16
2.2.3 *Rezolvare detaliată . . . . .	16
2.3 Tort . . . . .	16
2.3.1 Indicații de rezolvare . . . . .	17
2.3.2 Cod sursă . . . . .	18
2.3.3 *Rezolvare detaliată . . . . .	18
<b>3 OJI 2020</b>	<b>19</b>
3.1 alinieri . . . . .	19
3.1.1 Indicații de rezolvare . . . . .	20
3.1.2 Cod sursă . . . . .	21
3.1.3 *Rezolvare detaliată . . . . .	25
3.2 arh . . . . .	25
3.2.1 Indicații de rezolvare . . . . .	26
3.2.2 Cod sursă . . . . .	28
3.2.3 *Rezolvare detaliată . . . . .	35
3.3 leftmax . . . . .	35
3.3.1 Indicații de rezolvare . . . . .	36
3.3.2 Cod sursă . . . . .	36

3.3.3	*Rezolvare detaliată . . . . .	41
<b>4</b>	<b>OJI 2019</b>	<b>42</b>
4.1	pif . . . . .	42
4.1.1	Indicații de rezolvare . . . . .	43
4.1.2	Cod sursă . . . . .	45
4.1.3	*Rezolvare detaliată . . . . .	55
4.2	traseu . . . . .	55
4.2.1	Indicații de rezolvare . . . . .	56
4.2.2	Cod sursă . . . . .	57
4.2.3	*Rezolvare detaliată . . . . .	62
4.3	yinyang . . . . .	62
4.3.1	Indicații de rezolvare . . . . .	63
4.3.2	Cod sursă . . . . .	64
4.3.3	*Rezolvare detaliată . . . . .	67
<b>5</b>	<b>OJI 2018</b>	<b>68</b>
5.1	castel . . . . .	68
5.1.1	Indicații de rezolvare . . . . .	70
5.1.2	Cod sursă . . . . .	70
5.1.3	*Rezolvare detaliată . . . . .	72
5.2	eq4 . . . . .	72
5.2.1	Indicații de rezolvare . . . . .	73
5.2.2	Cod sursă . . . . .	74
5.2.3	*Rezolvare detaliată . . . . .	75
5.3	turnuri . . . . .	75
5.3.1	Indicații de rezolvare . . . . .	77
5.3.2	Cod sursă . . . . .	78
5.3.3	*Rezolvare detaliată . . . . .	79
<b>6</b>	<b>OJI 2017</b>	<b>80</b>
6.1	caps . . . . .	80
6.1.1	Indicații de rezolvare . . . . .	81
6.1.2	Cod sursă . . . . .	82
6.1.3	*Rezolvare detaliată . . . . .	90
6.2	rover . . . . .	90
6.2.1	Indicații de rezolvare . . . . .	91
6.2.2	Cod sursă . . . . .	91
6.2.3	*Rezolvare detaliată . . . . .	98
6.3	sir . . . . .	98
6.3.1	Indicații de rezolvare . . . . .	99
6.3.2	Cod sursă . . . . .	99
6.3.3	*Rezolvare detaliată . . . . .	104
<b>7</b>	<b>OJI 2016</b>	<b>105</b>
7.1	interesant . . . . .	105
7.1.1	Indicații de rezolvare . . . . .	106
7.1.2	Cod sursă . . . . .	106
7.1.3	*Rezolvare detaliată . . . . .	116
7.2	miting . . . . .	117
7.2.1	Indicații de rezolvare . . . . .	118
7.2.2	Cod sursă . . . . .	120
7.2.3	*Rezolvare detaliată . . . . .	132
<b>8</b>	<b>OJI 2015</b>	<b>133</b>
8.1	charlie . . . . .	133
8.1.1	Indicații de rezolvare . . . . .	134
8.1.2	Cod sursă . . . . .	135
8.1.3	*Rezolvare detaliată . . . . .	145
8.2	panda . . . . .	145
8.2.1	Indicații de rezolvare . . . . .	147

8.2.2	Cod sursă . . . . .	147
8.2.3	*Rezolvare detaliată . . . . .	160
<b>9</b>	<b>OJI 2014</b>	<b>161</b>
9.1	ferma . . . . .	161
9.1.1	Indicații de rezolvare . . . . .	162
9.1.2	Cod sursă . . . . .	162
9.1.3	*Rezolvare detaliată . . . . .	172
9.2	triunghi . . . . .	172
9.2.1	Indicații de rezolvare . . . . .	173
9.2.2	Cod sursă . . . . .	173
9.2.3	*Rezolvare detaliată . . . . .	178
<b>10</b>	<b>OJI 2013</b>	<b>179</b>
10.1	calcule . . . . .	179
10.1.1	Indicații de rezolvare . . . . .	180
10.1.2	Cod sursă . . . . .	180
10.1.3	*Rezolvare detaliată . . . . .	182
10.2	zona . . . . .	182
10.2.1	Indicații de rezolvare . . . . .	184
10.2.2	Cod sursă . . . . .	185
10.2.3	*Rezolvare detaliată . . . . .	189
<b>11</b>	<b>OJI 2012</b>	<b>190</b>
11.1	compresie . . . . .	190
11.1.1	Indicații de rezolvare . . . . .	191
11.1.2	Cod sursă . . . . .	191
11.1.3	*Rezolvare detaliată . . . . .	196
11.2	culori . . . . .	196
11.2.1	Indicații de rezolvare . . . . .	197
11.2.2	Cod sursă . . . . .	198
11.2.3	*Rezolvare detaliată . . . . .	205
<b>12</b>	<b>OJI 2011</b>	<b>206</b>
12.1	ai . . . . .	206
12.1.1	Indicații de rezolvare . . . . .	208
12.1.2	Cod sursă . . . . .	208
12.1.3	*Rezolvare detaliată . . . . .	217
12.2	expresie . . . . .	217
12.2.1	Indicații de rezolvare . . . . .	218
12.2.2	Cod sursă . . . . .	220
12.2.3	*Rezolvare detaliată . . . . .	226
<b>13</b>	<b>OJI 2010</b>	<b>227</b>
13.1	Expoziție . . . . .	227
13.1.1	Indicații de rezolvare . . . . .	228
13.1.2	Cod sursă . . . . .	228
13.1.3	*Rezolvare detaliată . . . . .	230
13.2	Text . . . . .	230
13.2.1	Indicații de rezolvare . . . . .	231
13.2.2	Cod sursă . . . . .	232
13.2.3	*Rezolvare detaliată . . . . .	233
<b>14</b>	<b>OJI 2009</b>	<b>234</b>
14.1	Insule . . . . .	234
14.1.1	Indicații de rezolvare . . . . .	235
14.1.2	Cod sursă . . . . .	235
14.1.3	*Rezolvare detaliată . . . . .	238
14.2	Rețetă . . . . .	238
14.2.1	Indicații de rezolvare . . . . .	239
14.2.2	Cod sursă . . . . .	239

14.2.3	*Rezolvare detaliată . . . . .	242
<b>15 OJI 2008</b>		<b>243</b>
15.1	Colaj . . . . .	243
15.1.1	Indicații de rezolvare . . . . .	244
15.1.2	Cod sursă . . . . .	246
15.1.3	*Rezolvare detaliată . . . . .	250
15.2	Piața . . . . .	250
15.2.1	Indicații de rezolvare . . . . .	251
15.2.2	Cod sursă . . . . .	252
15.2.3	*Rezolvare detaliată . . . . .	254
<b>16 OJI 2007</b>		<b>255</b>
16.1	Alee . . . . .	255
16.1.1	Indicații de rezolvare . . . . .	256
16.1.2	Cod sursă . . . . .	256
16.1.3	Rezolvare detaliată . . . . .	258
16.2	Dir . . . . .	260
16.2.1	Indicații de rezolvare . . . . .	261
16.2.2	Cod sursă . . . . .	261
16.2.3	Rezolvare detaliată . . . . .	265
<b>17 OJI 2006</b>		<b>269</b>
17.1	Ecuatii . . . . .	269
17.1.1	Indicații de rezolvare . . . . .	270
17.1.2	Cod sursă . . . . .	270
17.1.3	Rezolvare detaliată . . . . .	271
17.2	Sudest . . . . .	273
17.2.1	Indicații de rezolvare . . . . .	274
17.2.2	Cod sursă . . . . .	274
17.2.3	Rezolvare detaliată . . . . .	276
<b>18 OJI 2005</b>		<b>279</b>
18.1	Lăcusta . . . . .	279
18.1.1	Indicații de rezolvare . . . . .	279
18.1.2	Cod sursă . . . . .	280
18.1.3	Rezolvare detaliată . . . . .	281
18.2	Scara . . . . .	286
18.2.1	Indicații de rezolvare . . . . .	287
18.2.2	Cod sursă . . . . .	287
18.2.3	Rezolvare detaliată . . . . .	288
<b>19 OJI 2004</b>		<b>291</b>
19.1	Perle . . . . .	291
19.1.1	Indicații de rezolvare . . . . .	292
19.1.2	Cod sursă . . . . .	292
19.1.3	Rezolvare detaliată . . . . .	294
19.2	Romeo și Julieta . . . . .	296
19.2.1	Indicații de rezolvare . . . . .	297
19.2.2	Cod sursă . . . . .	297
19.2.3	Rezolvare detaliată . . . . .	299
<b>20 OJI 2003</b>		<b>301</b>
20.1	Spirala . . . . .	301
20.1.1	Indicații de rezolvare . . . . .	302
20.1.2	Cod sursă . . . . .	302
20.1.3	Rezolvare detaliată . . . . .	304
20.2	Taxe . . . . .	310
20.2.1	Indicații de rezolvare . . . . .	310
20.2.2	Cod sursă . . . . .	310
20.2.3	Rezolvare detaliată . . . . .	312

<b>21 OJI 2002</b>	<b>314</b>
21.1 Cod strămoș . . . . .	314
21.1.1 *Indicații de rezolvare . . . . .	314
21.1.2 *Cod sursă . . . . .	314
21.1.3 Rezolvare detaliată . . . . .	314
21.2 Triangulații . . . . .	317
21.2.1 *Indicații de rezolvare . . . . .	318
21.2.2 *Cod sursă . . . . .	318
21.2.3 Rezolvare detaliată . . . . .	318
<b>Appendix A "Instalare" C++</b>	<b>321</b>
A.1 Kit_OJI.2017 . . . . .	321
A.1.1 Code::Blocks . . . . .	322
A.1.2 Folder de lucru . . . . .	323
A.1.3 Utilizare Code::Blocks . . . . .	325
A.1.4 Setări Code::Blocks . . . . .	328
A.1.5 Multe surse în Code Blocks . . . . .	330
A.2 winlibs . . . . .	331
A.2.1 GCC și MinGW-w64 pentru Windows . . . . .	331
A.2.2 PATH . . . . .	331
A.2.3 CodeBlocks . . . . .	334
<b>Appendix B Exponențiere rapidă</b>	<b>344</b>
B.1 Analogie baza 2 cu baza 10 . . . . .	344
B.2 Notatii, relații și formule . . . . .	345
B.3 Pregătire pentru scrierea codului! . . . . .	345
B.4 Codul . . . . .	346
B.5 Chiar este rapidă? . . . . .	349
B.6 Rezumat intuitiv! . . . . .	350
<b>Index</b>	<b>352</b>
<b>Bibliografie</b>	<b>354</b>
<b>Lista autorilor</b>	<b>357</b>



# Lista figurilor

7.1	Miting	118
7.2	Miting	118
7.3	Miting	118
7.4	mitingIR1	119
7.5	mitingIR2	119
9.1	ferma	161
9.2	fermaIR	162
9.3	triunghi	172
10.1	zona	183
11.1	compresie	190
12.1	ai	206
15.1	colaj	244
15.2	colaj	244
15.3	colaj	244
15.4	colaj	245
16.1	Dir	260
20.1	Spirala1	301
20.2	Spirala2	301
20.3	Spirala3	301
21.1	Triang	318
A.1	Fişierele din Kit_OJI_2017	321
A.2	CodeBlocks & C++ Reference	322
A.3	Ce conţine C:\ OJI \	322
A.4	Folder de lucru	323
A.5	New -> Text document	324
A.6	Schimbare nume fişier şi nume extensie fişier	324
A.7	Confirmare schimbare extensie în .cpp	325
A.8	Pregătit pentru Code::Blocks	325
A.9	Pregătit pentru a scrie cod de program C++ în Code::Blocks	325
A.10	Primul cod de program C++ în Code::Blocks	326
A.11	Build - compilare	326
A.12	0 error(s), 0 warning(s)	327
A.13	Run - execuţie	327
A.14	Executat corect: a făcut "nimic"	328
A.15	Settings --> Compiler	328
A.16	Toolchain executables	329
A.17	Unde sunt acele programe	329
A.18	Multe surse în Code Blocks - setări	330
A.19	Multe surse în Code Blocks - exemple	330
A.20	mingw64 pe D:	331
A.21	search <b>path</b>	332
A.22	System properties -> Advanced	332

A.23 Environment Variables . . . . .	333
A.24 Edit Environment Variables -> New . . . . .	333
A.25 Calea și versiunea pentru gcc . . . . .	334
A.26 Settings -> Compiler . . . . .	335
A.27 Toolchain executables -> Auto-detect . . . . .	335
A.28 New -> Text Document . . . . .	336
A.29 New text Document.txt . . . . .	336
A.30 Schimbare nume și extensie . . . . .	336
A.31 Moore apps . . . . .	337
A.32 Look for another app . . . . .	337
A.33 Cale pentru codeblocks.exe . . . . .	338
A.34 Selectare codeblocks.exe . . . . .	338
A.35 Editare test01.cpp . . . . .	339
A.36 Compilare <b>test01</b> . . . . .	339
A.37 Mesaje după compilare . . . . .	340
A.38 Execuție <b>test01</b> . . . . .	340
A.39 Rezultat execuție <b>test01</b> . . . . .	341
A.40 Fișiere apărute după compilare! . . . . .	341
A.41 Creare <b>test02.cpp</b> gol! + <double click> sau <Enter> . . . . .	341
A.42 Lista programelor de utilizat . . . . .	342
A.43 Selectare <b>Code::Blocks IDE</b> pentru fișierele <b>.cpp</b> . . . . .	342
A.44 Editare+Compilare+Execuție pentru <b>test02</b> . . . . .	342
A.45 Selectare <b>tab</b> ce conține <b>test01.cpp</b> . . . . .	343
B.1 Analogie B2 cu B10 . . . . .	344

# Lista tabelelor

# Lista programelor

3.1.1	alinieri_bogdan_100.cpp	21
3.1.2	alinieri_CC1.cpp	21
3.1.3	alinieri_CC2.cpp	22
3.1.4	alinieri_CC3.cpp	23
3.1.5	alinieri_CC4.cpp	24
3.2.1	arh_eugen.cpp	28
3.2.2	arh_razvan.cpp	30
3.2.3	arh_tamio.cpp	31
3.3.1	leftmax_O(nxn).cpp	36
3.3.2	leftmax_optimn.cpp	37
3.3.3	leftmax_razvan_O(n).cpp	38
3.3.4	leftmax_razvan_O(nlog).cpp	39
3.3.5	leftmax_razvan_O(nn).cpp	40
4.1.1	pif_30p_1.cpp	45
4.1.2	pif_30p_2.cpp	46
4.1.3	pif_50p.cpp	46
4.1.4	pif_70p_1.cpp	47
4.1.5	pif_70p_2.cpp	48
4.1.6	pif_70p_3.cpp	50
4.1.7	pif_90p_1.cpp	51
4.1.8	pif_90p_2.cpp	52
4.1.9	pif_90p_3.cpp	53
4.2.1	traseu40.cpp	57
4.2.2	traseu90_1.cpp	58
4.2.3	traseu90_2.cpp	59
4.2.4	traseu90_3.cpp	60
4.2.5	traseu90_4.cpp	61
4.2.6	traseu90_5.cpp	61
4.3.1	yinyang45.cpp	64
4.3.2	yinyang90_1.cpp	64
4.3.3	yinyang90_2.cpp	65
5.1.1	castel.cpp	70
5.2.1	eq4.cpp	74
5.3.1	turnuri.cpp	78
6.1.1	adrian-100.cpp	82
6.1.2	manolache-100.cpp	84
6.1.3	manolache-v1_100.cpp	85
6.1.4	manolache-v2_100.cpp	86
6.1.5	MLT-100.cpp	88
6.2.1	adrian-100.cpp	91
6.2.2	GM_rover.cpp	93
6.2.3	MLT_Rover.cpp	94
6.2.4	MLT_Rover_STL.cpp	96
6.3.1	adrian-100.cpp	99
6.3.2	GM1.cpp	100
6.3.3	GM2.cpp	101
6.3.4	sir_100.cpp	102
6.3.5	sirEm.cpp	103
7.1.1	interesant_en1.cpp	106

7.1.2	interesant_en2.cpp	108
7.1.3	interesant_en3.cpp	109
7.1.4	interesant_gc1.cpp	111
7.1.5	interesant_gc2.cpp	112
7.1.6	interesant_gc3.cpp	113
7.1.7	interesant_mot.cpp	115
7.1.8	interesant_nv.cpp	115
7.2.1	miting.cpp	120
7.2.2	miting_en.cpp	122
7.2.3	miting_gc2.cpp	124
7.2.4	miting_gc3.cpp	126
7.2.5	miting_gc4.cpp	128
7.2.6	miting_gc5.cpp	130
8.1.1	charlie_adriana.cpp	135
8.1.2	charlie_eugen0.cpp	136
8.1.3	charlie_eugen1.cpp	137
8.1.4	charlie_eugen2.cpp	138
8.1.5	charlie_LilianaSchiopu.cpp	139
8.1.6	charlie_marcel.cpp	141
8.1.7	charlie_radu_v.cpp	142
8.1.8	charlie_SC.cpp	143
8.1.9	charlie_zoli.cpp	144
8.2.1	panda_adriana.cpp	147
8.2.2	panda_eugen0.cpp	149
8.2.3	panda_eugen1.cpp	150
8.2.4	panda_eugen2.cpp	151
8.2.5	panda_Liliana_Schiopu.cpp	153
8.2.6	panda_marcel.cpp	154
8.2.7	panda_radu.cpp	156
8.2.8	panda_zoli1.cpp	157
9.1.1	fermadaniel.cpp	163
9.1.2	fermavlad.cpp	165
9.1.3	flore_nerecursiv.cpp	166
9.1.4	flore_recurziv.cpp	169
9.2.1	triunghi_LS.cpp	173
9.2.2	triunghi_PD.cpp	175
9.2.3	zoli_triunghi.cpp	176
10.1.1	calcFUcuBS.cpp	180
10.1.2	calcFUfaraBS.cpp	181
10.1.3	calcule.cpp	181
10.1.4	vcalcule.cpp	182
10.2.1	Dzona.cpp	185
10.2.2	Gzona.cpp	187
11.1.1	compresie_cristina_sichim.cpp	191
11.1.2	compresie_eugen_nodea1.cpp	192
11.1.3	compresie_eugen_nodea2.cpp	194
11.1.4	compresie_silviu_candale.cpp	195
11.2.1	culoriEM.cpp	198
11.2.2	culori.cpp	199
11.2.3	culori1.cpp	200
11.2.4	culoriCS.cpp	200
11.2.5	culoriEN.cpp	201
11.2.6	culoriLI.cpp	202
11.2.7	culoriLS.cpp	203
11.2.8	culoriSC.cpp	204
12.1.1	ai.cpp	208
12.1.2	ai2.cpp	211
12.1.3	ai3.cpp	215
12.2.1	expresie_rec.cpp	220
12.2.2	expresie_stive1.cpp	223



12.2.3	expresie_stive2.cpp	224
13.1.1	exp_comb.cpp	228
13.1.2	exp_din.cpp	229
13.2.1	text.pas	232
14.1.1	insule94.cpp	235
14.1.2	insule.cpp	236
14.2.1	RETETA.cpp	239
14.2.2	RETETAV.cpp	241
15.1.1	COLAJ_40.cpp	246
15.1.2	Colaj_50.cpp	247
15.1.3	colaj_C.cpp	248
15.2.1	PI_1.pas	252
15.2.2	PI_2.pas	252
15.2.3	PI_3.pas	253
15.2.4	PI_OK.pas	254
16.1.1	alee.c	256
16.1.2	Alee1.java	258
16.2.1	dir_em.c	261
16.2.2	dir.cpp	263
16.2.3	dir1.java	265
16.2.4	dir2.java	267
17.1.1	ecuatii.cpp	270
17.1.2	ecuatii.java	271
17.2.1	sudest.cpp	274
17.2.2	sudest1.java	276
17.2.3	sudest2.java	277
18.1.1	lacusta.c	280
18.1.2	lacusta1.java	281
18.1.3	lacusta2.java	281
18.1.4	lacusta3.java	283
18.1.5	lacusta4.java	284
18.1.6	lacusta5.java	285
18.2.1	scara.cpp	287
18.2.2	scara.java	288
19.1.1	perle.c	292
19.1.2	perle.java	294
19.2.1	rj.cpp	297
19.2.2	rj.java	299
20.1.1	spirala2.pas	302
20.1.2	spirala1.java	304
20.1.3	spirala2.java	306
20.1.4	spirala3.java	307
20.2.1	taxe.pas	310
20.2.2	taxe.java	312
21.1.1	codstramos1.java	314
21.1.2	codstramos2.java	316
21.2.1	triangulatii.java	318
B.4.1	exponentiere_rapida1.cpp	346
B.4.2	exponentiere_rapida2.cpp	347
B.4.3	exponentiere_rapida3.cpp	348
B.4.4	exponentiere_rapida_MOD.cpp	348
B.5.1	exponentiere_naiva_MOD.cpp	349
B.5.2	exponentiere_rapida_MOD.cpp	350
B.6.1	secventa_cod.cpp	351

# Capitolul 1

## OJI 2022

### 1.1 circular

#### Problema 1 - Circular

100 de puncte

O imprimantă circulară are litere mari ale alfabetului englezesc dispuse circular de la A la Z. Imprimanta are un indicator care inițial este plasat la litera A.

Pentru a tipări o literă indicatorul imprimantei se mișcă la stânga sau dreapta.

Mișcarea indicatorului către o literă alăturată aflată la stânga sau la dreapta literei curente se realizează într-o secundă. De exemplu: pentru a tipări șirul BCY mișcarea indicatorului se va face către dreapta de la A la B într-o secundă, apoi de la B la C într-o secundă, apoi către stânga de la C la Y în 4 secunde.

În total pentru a tipări șirul BCY sunt necesare 6 secunde. Imprimanta va alege întotdeauna sensul cel mai avantajos de deplasare, astfel încât timpul de deplasare să fie minim.

Imprimanta tipărește literele în două culori roșu sau albastru. Unele litere se tipăresc cu cerneală roșie, restul cu cerneală albastră. Pentru simplitate le vom numi litere roșii și litere albastre.



#### Cerințe

Fiind date un șir de litere albastre nu neapărat distincte și mulțimea literelor roșii ale imprimantei, să se calculeze:

1. Care este timpul pentru tipărirea la imprimantă circulară a șirului de litere albastre.
2. Să se insereze între oricare două litere albastre aflate pe poziții consecutive câte o literă roșie astfel încât să se obțină timpul minim pentru tipărire și să se afișeze:
  - timpul minim
  - numărul de șiruri distincte care sunt tipărite cu timp minim
  - șirul minim lexicografic dintre toate șirurile ce sunt tipărite în acest timp

#### Date de intrare

Fișierul **circular.in** conține:

1. pe prima linie un număr natural  $c$  cu valori posibile 1 sau 2 reprezentând cerința problemei
2. pe a doua linie un șir de litere albastre, nu neapărat distincte
3. pe a treia linie mulțimea literelor roșii distincte în ordine alfabetic

#### Date de ieșire

În fișierul **circular.out** se va afișa în funcție de cerință:

- Dacă  $c = 1$ , un singur număr natural reprezentând timpul necesar pentru tipărirea la imprimantă a șirului de litere albastre
- Dacă  $c = 2$  se vor tipări trei rezultate, fiecare pe câte o linie:
  - timpul minim pentru tipărire conform cerinței, ei a doua
  - numărul de șiruri distincte care sunt tipărite cu timp minim *modulo* 666 013
  - șirul minim lexicografic ce obține acest timp

**Restricții și precizări**

- Cele două șiruri conțin doar litere mari ale alfabetului englez
- Lungimea șirului de litere albastre nu depășește 50 000 de litere
- Mulțimea literelor roșii nu depășește 25 de litere, care sunt distincte și afișate în ordine alfabetică
- Toate celelalte litere care nu se regăsesc în mulțimea literelor roșii, sunt albastre
- Pentru cazul  $c = 2$  se acordă punctaj parțial astfel:
  - 25% din punctaj, pentru afișarea timpului minim
  - 25% din punctaj, pentru afișarea numărului de șiruri ce obțin timpul minim
  - 50% din punctaj, pentru afișarea șirului minim lexicografic
- Atenție! Pentru obținerea punctajului la cerința a doua, pentru orice test, în fișierul de ieșire trebuie să existe exact trei linii care respectă formatul cerut.

#	Punctaj	Restricții
1	24	$c = 1$
2	76	$c = 2$

**Exemple:**

circular.in	circular.out	Explicații
1 BBTH AEIOU	21	Timpul de tipărire al șirului BBTH este 21 și se obține astfel: de la A la B = 1 secundă de la B la B = 0 secundă de la B la T = 8 secunde de la T la H = 12 secunde
2 BBTH AEIOU	23 4 BABATIH	Timpul minim pentru tipărirea la imprimantă este 23 și se obține pentru șirul BABATIH astfel: de la A la B = 1 secundă de la B la A = 1 secundă de la A la B = 1 secundă de la B la A = 1 secundă de la A la T = 7 secunde de la T la I = 11 secunde de la I la H = 1 secundă în total 23 de secunde. Avem 4 șiruri pentru care se obține timp minim la tipărire: BABATIH, BABATOH, BABUTIH, BABUTOH. Prima soluție în ordine lexicografică este BABATIH.
2 AMYMAMAMY BCDEFGHIJKLMNOPQRSTUVWXYZ	96 568708 ABMNYNMBABMBABMNY	Timpul minim de tipărire este 96. Avem 214358881 șiruri distincte, iar $214358881 \bmod 666013 = 568708$ . Prima soluție în ordine lexicografică este ABMNYNMBABMBABMNY.

**1.1.1 Indicații de rezolvare**

Propusă de: Prof. Boca Alina Gabriela - Colegiul Național de Informatică "Tudor Vianu" București

**Cerința 1.** Pentru rezolvarea primei cerințe se parcurge șirul de litere albastre și pentru oricare două litere alăturate se calculează cea mai mică distanță dintre litera curentă și litera următoare din șir.

Pentru a calcula distanța minimă dintre două litere  $A_i$  și  $A_{i+1}$  va trebui să calculăm minimul dintre cele două cazuri:

- de la  $A_i$  la  $A_{i+1}$  în sensul acelor de ceasornic
- de la  $A_i$  la  $A_{i+1}$  în sens opus acelor de ceasornic

Pentru a lua în considerare faptul că imprimanta începe de pe poziția  $A$ , putem considera  $A_0 = A$ .

Complexitatea temporală este  $O(N)$ .

Rezolvarea primei cerințe obține 24 de puncte.

**Cerința 2.** Pentru a rezolva cerința 2, vom *precalcula* un tablou bidimensional de  $26 \times 26$ :

$cost_{i,j}$  = numărul minim de pași pentru a ajunge de la a  $i$ -a literă din alfabet, la a  $j$ -a.

Ulterior, se parcurge șirul literelor albastre și între fiecare două litere albastre  $A_i, A_{i+1}$ , se caută în șirul literelor roșii acele litere  $R$  pentru care distanța  $cost_{A_i,R} + cost_{R,A_{i+1}}$  este minimă.

Costul minim reprezintă suma distantelor minime obținute, la care trebuie adăugat costul de a aduce capul de printare de la  $L_0 = A$  la  $L_1$ .

Pentru a construi șirul minim *lexicografic*, între oricare două litere albastre, vom insera dintre toate literele roșii ce obțin un cost minim pe cea mai mică lexicografic.

Numărul de soluții reprezintă produsul dintre numărul de posibilități de a insera o literă roșie între două litere albastre care generează distanța minimă.

Complexitatea temporală este:  $O(N \cdot \Sigma)$ , unde  $\Sigma = 26$  reprezintă dimensiunea alfabetului.

Rezolvarea celei de-a doua cerințe obține 76 de puncte.

### 1.1.2 \*Cod sursă

### 1.1.3 \*Rezolvare detaliată

## 1.2 pulsar

### Problema 2 - Pulsar

100 de puncte

*Data stelară 3210:*

Căpitanul navei USS Enterprize, Jean-Luc Picard se află într-o misiune importantă în cuadrantul Beta al galaxiei.

Acesta trebuie să ajungă cât mai rapid de la planeta Vulcan până la planeta Qo'noS dar din păcate pentru această misiune Jean-Luc Picard nu va putea să ajungă instantaneu la destinație folosind warp drive-ul navei, ci va trebui să se deplaseze în mod normal, din sector în sector.

Harta galaxiei este reprezentată sub forma unei tabele bidimensionale de dimensiune  $N \times N$ , în care fiecare celulă reprezintă un sector al galaxiei. Coordonatele sectorului în care se află planeta Vulcan sunt  $(x_s, y_s)$ , iar coordonatele sectorului în care se află planeta Qo'noS sunt  $(x_f, y_f)$ .

USS Enterprise se poate deplasa într-o unitate de timp dintr-un sector în oricare dintre sectoarele adiacente, fie pe aceeași linie, fie pe aceeași coloană. În plus, nava poate staționa o perioadă nedeterminată de timp în orice sector. Nava se poate afla doar pe un sector care la momentul actual de timp nu o pune în pericol.

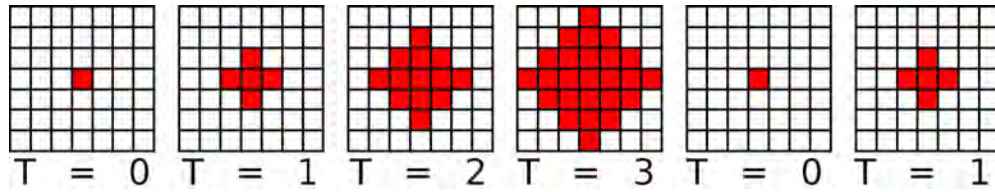
Pentru că nicio aventură nu este lipsită de pericole, drumul lui Jean-Luc Picard este presărat de pulsari, obiecte cosmice foarte periculoase care lansează în vecinătatea lor, la intervale fixe de timp, unde gravitaționale care ar putea distruge USS Enterprise.

Un pulsar  $P_i$  este caracterizat prin patru variabile  $(x_i, y_i, r_i, t_i)$ , unde  $(x_i, y_i)$  reprezintă coordonatele sectorului în care se regăsește pulsarul,  $r_i$  reprezintă raza de acțiune a pulsarului, iar  $t_i$  reprezintă starea în care se află pulsarul la momentul de început al deplasării navei.

Un pulsar  $P_i$  trece periodic printr-un număr de  $r_i$  stări de la 0 la  $r_i - 1$ . Când acesta se află în starea  $t$ , acesta afectează toate sectoarele aflate la o *distanță Manhattan* mai mică sau egală cu

$t$  față de sectorul în care se află acesta. Dacă pulsarul la un moment de timp se află în starea  $t$ , la momentul următor se va afla în starea  $(t + 1) \% r_i$ .

Un exemplu de funcționare al unui pulsar cu rază de acțiune  $r = 4$ , timp de 6 unități de timp, începând cu  $t = 0$  este următorul:



### Cerințe

Vouă vă revine rolul de a îl ajuta pe Jean-Luc Picard și să îi răspundeți la una din următoarele întrebări știind harta galaxiei:

1. Care este numărul maxim de sectoare ale galaxiei  $S_{max}$  afectate la orice moment de timp de către cel puțin un pulsar.
2. Care este timpul minim  $T_{min}$  de care are nevoie Jean-Luc Picard pentru a ajunge pe planeta Qo'noS.

### Date de intrare

Din fișierul **pulsar.in** se vor citi următoarele:

- Pe prima linie se vor afla trei numere  $C$ ,  $N$  și  $P$  separate prin câte un spațiu, reprezentând cerința ce trebuie rezolvată, dimensiunea galaxiei și numărul de pulsari din galaxie
- Pe următoarele  $P$  linii se vor afla câte patru numere separate prin spațiu,  $x_i$ ,  $y_i$ ,  $r_i$ ,  $t_i$ , reprezentând descrierea pulsarului  $P_i$
- Pe penultima linie se vor afla două numere separate printr-un spațiu reprezentând coordonatele sectorului planetei Vulcan  $x_s$  și  $y_s$
- Pe ultima linie se vor afla două numere separate printr-un spațiu reprezentând coordonatele sectorului planetei Qo'noS  $x_f$  și  $y_f$

### Date de ieșire

În fișierul **pulsar.out** se va afișa un singur număr în funcție de cerință:

- Dacă  $C = 1$ , atunci se va afișa numărul  $S_{max}$
- Dacă  $C = 2$ , atunci se va afișa numărul  $T_{min}$

### Restricții și precizări

- Distanța Manhattan dintre două coordonate  $(x_1, y_1)$  și  $(x_2, y_2)$  este definită ca:  $|x_1 - x_2| + |y_1 - y_2|$
- Nava nu va putea părăsi la niciun moment de timp harta galaxiei
- Undele pulsarilor pot părăsi harta galaxiei, dar acele sectoare nu reprezintă interes pentru problema noastră
- Se garantează că la momentul plecării, nava nu este aflată în pericol
- Se garantează că există soluție
- Pot exista mai mulți pulsari în același sector
- $C \in \{1, 2\}$
- $3 \leq N \leq 500$
- $1 \leq P \leq 15\,000$
- $0 \leq t_i < r_i \leq 6 \quad \forall 1 \leq i \leq P$
- $1 \leq x_s, y_s, x_f, y_f \leq N$
- $1 \leq x_i, y_i \leq N \quad \forall 1 \leq i \leq P$

#	Punctaj	Restricții
1	19	$C = 1$
2	22	$C = 2$ și $r_i = 1 \quad \forall 1 \leq i \leq P$
3	9	$C = 2$ , $N \leq 7$ și $r_i \leq 3 \quad \forall 1 \leq i \leq P$
4	13	$C = 2$ , $t_i = 0 \quad \forall 1 \leq i \leq P$
5	37	$C = 2$

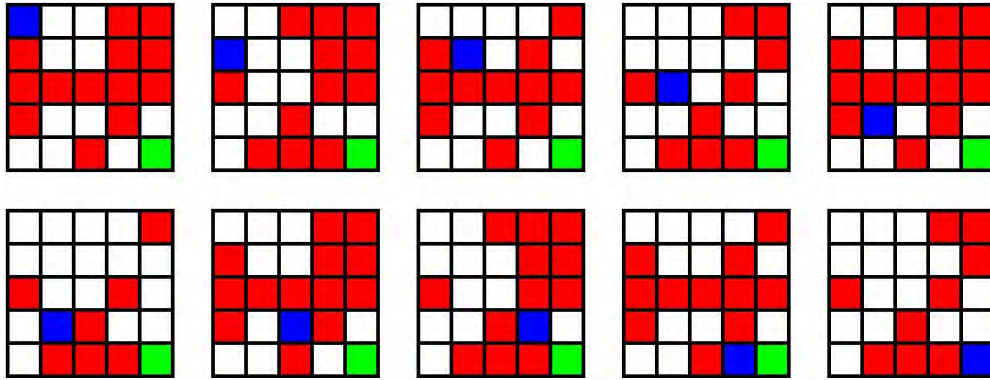


**Exemple:**

pulsar.in	pulsar.out
1 5 4 3 1 2 1 1 5 3 1 5 3 2 0 3 4 2 1 1 1 5 5	14
2 5 4 3 1 2 1 1 5 3 1 5 3 2 0 3 4 2 1 1 1 5 5	9

**Explicații:**

Mai jos se poate observa drumul realizat de USS Enterprise. Cu albastru s-a ilustrat nava, cu roșu, zonele afectate de pulsari, iar cu verde planeta Qo'nos:



Pentru primul exemplu, se observă că nu va exista niciodată un moment de timp în care pulsarii să ocupe mai mult de 14 sectoare.

În figura de mai sus este prezentat un posibil drum de durată 9. Acest timp este și minim pentru exemplul dat.

### 1.2.1 Indicații de rezolvare

*Propusă de: Stud. Tulbă-Lecu Theodor-Gabriel - Universitatea Politehnica din București*

**Observații.** Pentru a rezolva problema, inițial trebuie făcută următoarea observație: După un anumit timp minim  $T$ , pulsarele vor reveni înapoi în starea inițială de la momentul de timp  $t = 0$ . Vom numi acest timp  $T$ , *perioada* pulsarelor.

În continuare, trebuie făcută observația că dacă toate pulsarele revin în starea inițială după  $T$  unități de timp, cum un pulsar  $P_i$  de perioadă  $r_i$ , se află în starea inițială doar la momente de timp care sunt multiplii de  $r_i$ , atunci  $T$  este și el multiplu al lui  $r_i$ .

Astfel,  $T$  este cel mai mic multiplu comun al perioadelor pulsarelor:  $T = \text{cmmmc}(r_1, r_2, \dots, r_P)$ .

Cum pentru restricțiile problemei:  $1 \leq r_i \leq 6 \ \forall 1 \leq i \leq P$ , rezultă că  $T \leq 60$ .

**Cerința 1 - Subtask 1.** În urma observațiilor făcute, cerința 1, poate fi exprimată astfel: Pentru fiecare moment de timp de la 0 la  $T - 1$  câte sectoare ale galaxiei din cele  $N \times N$  sunt afectate de cel puțin un pulsar?

Acest lucru poate fi rezolvat calculând pentru fiecare moment de timp, un tablou bidimensional:

$$afectat_{i,j} = \begin{cases} 1, & \text{dacă există cel puțin un pulsar care afectează sectorul } (i, j) \\ 0, & \text{altfel} \end{cases}$$

Tabloul bidimensional afectat poate fi calculat prin marcarea pentru fiecare pulsar în parte a tuturor sectoarelor afectate de acesta la momentul de timp actual cu 1, toate valorile din afectat fiind inițializate cu 0.

Răspunsul pentru cerința 1 este maximul dintre numărul de valori de 1 din tabloul afectat, pentru fiecare moment de timp de la 0 la  $T - 1$ .

Complexitatea temporală este:  $O(T \cdot (N^2 + P \cdot R_{max}^2))$ , unde  $R_{max}$  este perioada maximă a unui pulsar.

Pentru rezolvarea corectă a cerinței 1 se pot obține 19 puncte.

### Cerința 2.

**Subtask 2.** Primul subtask al cerinței a doua reprezintă un caz particular al problemei. Dacă  $r_i = 1 \forall 1 \leq i \leq P$ , atunci toți pulsarii vor afecta doar sectoarele în care aceștia se află.

Astfel, problema devine găsirea un *drum de lungime minimă* de la sectorul  $(x_s, y_s)$  la sectorul  $(x_f, y_f)$  într-o hartă ce conține obstacole.

Acest lucru se poate rezolva utilizând *algoritmul lui Lee*.<sup>26</sup>

Complexitatea temporală este:  $O(N^2 + P)$ .

Pentru rezolvarea corectă a subtaskului 2 se pot obține 22 de puncte.

**Subtask 3.** Pentru acest subtask, cum  $N \leq 10$ , harta galaxiei are dimensiuni suficient de mici pentru ca problema să fie rezolvată utilizând metoda *backtracking*. Se vor genera toate drumurile posibile de la sectorul  $(x_s, y_s)$  la sectorul  $(x_f, y_f)$ , iar la fiecare pas se verifică dacă celula adăugată la drum, este afectată de vreun pulsar.

**Subtask-urile 4 și 5.** Pentru a rezolva complet cerința a doua, trebuie să ne folosim de observațiile făcute anterior.

Știind că harta galaxiei este periodică cu o perioadă  $T$  putem să reprezentăm starea navei sub forma unui triplet  $(x, y, t)$  unde  $x$  și  $y$  reprezintă coordonatele navei, iar  $t$  reprezintă starea hărții galaxiei.

Dacă nava se află la într-o stare  $(x, y, t)$ , aceasta la următorul moment de timp se va afla la:

- (1)  $(x, y, (t + 1) \% T)$ , dacă nava stă pe loc
- (2)  $(x, y - 1, (t + 1) \% T)$ , dacă nava se deplasează la stânga
- (3)  $(x, y + 1, (t + 1) \% T)$ , dacă nava se deplasează la dreapta
- (4)  $(x + 1, y, (t + 1) \% T)$ , dacă nava se deplasează în jos
- (5)  $(x - 1, y, (t + 1) \% T)$ , dacă nava se deplasează în sus

Astfel, nava se va deplasa într-un tabel tridimensional, a treia dimensiune fiind starea hărții, iar obstacolele vor fi create de zonele afectate de pulsari. Această problemă se poate rezolva tot cu ajutorul *algoritmului lui Lee*, care va trebui modificat pentru a funcționa pe trei dimensiuni.

Răspunsul va fi timpul minim cu care se poate ajunge din  $(x_s, y_s, 0)$  în  $(x_f, y_f, t) \forall 0 \leq t < T$ .

Complexitatea temporală este:  $O(T \cdot (N^2 + P \cdot R_{max}^2))$ .

## 1.2.2 \*Cod sursă

## 1.2.3 \*Rezolvare detaliată

<sup>26</sup> articol pbInfo - Algoritmul lui Lee, Prof. Silviu Candale

## 1.3 transport

### Problema 3 - Transport

100 de puncte

Anul 1905

Un stat din America de Sud și-a propus investiții majore în infrastructura feroviară. Brazilianul Badinho este managerul unei companii de transport feroviar pe o magistrală importantă. De-a lungul magistralei se află  $N$  stații, numerotate de la 1 la  $N$ . Fiecărei stații îi corespunde un număr  $X_i$  care reprezintă numărul de kilometri de la începutul magistralei până la stația  $i$  ( $X_1 = 0$ ). Pentru simplitate Badinho reprezintă magistrala ca o dreaptă, iar stațiile ca puncte pe dreapta respectivă, stația  $i$  aflându-se la coordonata  $X_i$ .

O rută reprezintă o submulțime de cel puțin 2 stații dintre cele  $N$ , cu semnificația că în aceste stații se vor face opriri. Orice rută operată de Badinho are 2 stații numite capete, definite ca fiind cea mai apropiată stație, inclusă în rută, de începutul magistralei respectiv cea mai îndepărtată stație, inclusă în rută, de începutul magistralei.

Compania lui Badinho va primi o subvenție pentru deschiderea unei noi rute, care va fi proporțională cu lungimea rutei deschise. Mai exact, Badinho va primi  $C$  reali (realul este moneda națională a Braziliei) pentru fiecare kilometru din noua rută. Lungimea rutei se definește ca fiind distanța dintre capete.

Badinho poate deschide două tipuri de rute:

- Regio - se fac opriri în toate stațiile dintre cele două capete
- Expres - unele stații dintre cele două capete pot fi traversate fără a opri în ele

Pentru a deschide o rută Badinho trebuie să construiască câte un depou în capetele rutei respective.

Costul pentru a construi un depou în stația  $i$  este  $D_i$  reali.

### Cerințe

Știind că Badinho trebuie să cheltuiască întreaga sumă pe care ar primi-o dintr-o subvenție, să se determine:

1. Numărul de moduri de a deschide o rută de tip Regio, modulo  $10^9 + 7$
2. Numărul de moduri de a deschide o rută de tip Expres, modulo  $10^9 + 7$

### Date de intrare

În fișierul **transport.in** se află:

- Pe prima linie tipul cerinței  $T$ , care poate avea valoarea 1 sau 2.
- Pe a doua linie  $N$  și  $C$ , separate printr-un spațiu, reprezentând numărul de stații, respectiv suma primită per kilometru ca subvenție
- Pe următoarele  $N$  linii, pe linia  $i + 2$  se află câte o pereche  $X_i$  și  $D_i$ , separate printr-un spațiu, reprezentând distanța la care se află stația  $i$  față de începutul magistralei, respectiv costul de a construi un depou în stația  $i$ .

### Date de ieșire

În fișierul **transport.out** se va afișa:

- Dacă  $T = 1$ , numărul de moduri de a deschide o rută de tip Regio, modulo  $10^9 + 7$
- Dacă  $T = 2$ , numărul de moduri de a deschide o rută de tip Expres, modulo  $10^9 + 7$

### Restricții și precizări

- Două rute se consideră distincte dacă diferă prin cel puțin o stație.
- $2 \leq N \leq 200\,000$ ,  $1 \leq C \leq 10^9$
- $0 \leq X_i, D_i \leq 10^9 \quad \forall 1 \leq i \leq N$
- $X_1 = 0$
- șirul  $X$  este sortat strict crescător:  $X_i < X_j \quad 1 \leq i < j \leq N$

- toate liniile de cale ferată ale magistralei sunt deja construite, singurele costuri pe care le va suporta Badinho sunt cele de construire a depourilor

#	Punctaj	Restricții
1	12	$T = 1, N = 1\,000$
2	26	$T = 1, N = 200\,000$
3	6	$T = 2, N = 15$
4	15	$T = 2, N = 1\,000$
5	41	$T = 2, N = 200\,000$

**Exemple:**

transport.in	transport.out
1 5 1 0 2 1 1 3 10 4 15 6 4	2
2 5 1 0 2 1 1 3 10 4 15 6 4	12

**Explicații:**

**Pentru primul exemplu:**

Rutele posibile în condițiile cerinței 1 sunt:  $\{1, 2, 3, 4, 5\}$ ,  $\{2, 3, 4, 5\}$

Ruta  $\{1, 2, 3, 4, 5\}$  conține opriri în stațiile 1, 2, 3, 4, 5. Stațiile 1 și 5 sunt cele 2 capete. Suma primită din subvenție este:  $1 \times (6 - 0) = 6$  reali (6-0 reprezintă distanța dintre stația 1 și 5), iar costul de construire a celor 2 depouri este:  $2 + 4 = 6$  reali.

**Pentru exemplul al doilea exemplu:**

Rutele posibile în condițiile cerinței 2 sunt:  $\{1, 5\}$ ,  $\{1, 2, 5\}$ ,  $\{1, 3, 5\}$ ,  $\{1, 4, 5\}$ ,  $\{1, 2, 3, 5\}$ ,  $\{1, 2, 4, 5\}$ ,  $\{1, 3, 4, 5\}$ ,  $\{1, 2, 3, 4, 5\}$ ,  $\{2, 5\}$ ,  $\{2, 3, 5\}$ ,  $\{2, 4, 5\}$ ,  $\{2, 3, 4, 5\}$

Ruta  $\{1, 2, 5\}$  conține opriri în stațiile 1, 2, 5. Stațiile 1 și 5 sunt cele 2 extreme. Suma primită din subvenție este:  $1 \times (6 - 0) = 6$  reali, iar costul de construire a celor 2 depouri e:  $2 + 4 = 6$  reali.

### 1.3.1 Indicații de rezolvare

*Propusă de: Stud. Cotor Andrei - Universitatea "Babeș-Bolyai" din Cluj-Napoca*

**Observații.**

- (1) O rută Regio reprezintă o subsecvență dinșirul de stații de forma:  $[st, st + 1, st + 2, \dots, dr]$ ,  $1 \leq st < dr \leq N$ , unde  $st$  și  $dr$  reprezintă capetele rutei.
- (2) O rută Expres reprezintă un subșir din șirul de stații de forma:  $[st, i_1, i_2, \dots, i_k, dr]$ ,  $1 \leq st < i_1 < i_2 < \dots < i_k < dr \leq N$ ,  $k \geq 0$ , unde  $st$  și  $dr$  reprezintă capetele rutei.
- (3) Relația care corespunde restricțiilor din enunț:  $D_{st} + D_{dr} = C \cdot (X_{dr} - X_{st})$ . Aceasta poate fi rescrisă în felul următor:  $D_{st} + D_{dr} = C \cdot (X_{dr} - X_{st}) \Rightarrow D_{st} + D_{dr} = C \cdot X_{dr} - C \cdot X_{st} \Rightarrow D_{st} + C \cdot X_{st} = C \cdot X_{dr} - D_{dr}$
- (4) În relația obținută la Observația 3 expresia din stânga egalului depinde doar de  $st(X_{st}, D_{st})$ , iar cea din dreapta egalului doar de  $dr(X_{dr}, D_{dr})$

**Cerința 1** ( $T = 1$ ).

**Subtask 1 (12 puncte).** Se parcurge fiecare subsecvență, fixând capătul din stânga și cel din dreapta, și se verifică condiția  $D_{st} + D_{dr} = C \cdot (X_{dr} - X_{st})$ . Dacă este respectată condiția se incrementează rezultatul.

Complexitate temporală:  $O(N^2)$ .

**Optimizare.** Soluția pentru Subtask-ul 1 poate fi optimizată parcurgând, pentru un  $dr$  fixat, doar indicii  $st$  pentru care  $(st, dr)$  pot forma o pereche de capete validă. Mai exact, după ce a fost fixat  $dr$ , se calculează valoare expresiei  $C \cdot X_{dr} - D_{dr}$  și se parcurg doar indicii  $st$  pentru care  $D_{st} + C \cdot X_{st} = C \cdot X_{dr} - D_{dr}$  (Observația 3).

Cu ajutorul acestei optimizări se pot obține 12 puncte din cele 26 acordate pentru Subtask-ul 2 (în plus față de cele acordate pentru subtask-urile precedente).

**Subtask 2 (26 puncte).** Se fixează capătul dreapta  $dr$ . Trebuie numărate câte capete stânga  $st$  există astfel încât perechea de capete  $(st, dr)$  este una validă, mai exact  $D_{st} + C \cdot X_{st} = C \cdot X_{dr} - D_{dr}$  (Observația 3).

Pentru a obține aceste valori va fi nevoie de o *normalizare*, valorile expresiilor  $D_{st} + C \cdot X_{st}$  sau  $C \cdot X_{dr} - D_{dr}$  putând fi foarte mari. Astfel pentru fiecare stație  $i$  se rețin într-un vector de lungime  $2N$ , care urmează să fie utilizat pentru normalizare, valorile  $D_i + C \cdot X_i$  și  $C \cdot X_i - D_i$ .

Complexitate temporală:  $O(N \log N)$ .

**Cerința 2** ( $T = 2$ ).

**Subtask 3 (6 puncte).**  $N$  fiind mic se poate utiliza *bactracking* pentru a genera toate subșirurile din șirul de stații. Pentru fiecare subșir generat se verifică dacă respectă condițiile din enunț.

**Subtask 4 (15 puncte).** Se fixează fiecare combinație de două capete ale unei rute (notate cu  $st$ , respectiv  $dr$ ), care respectă condițiile din enunț. Între cele două capete fixate există  $dr - st - 1$  stații. Astfel pentru perechea de capete  $(st, dr)$  numărul de rute Expres este egal cu numărul de subșiruri care se pot forma din subsecvența  $[st + 1, st + 2, \dots, dr - 1]$ , adică  $2^{dr-st-1}$ .

Complexitate temporală:  $O(N^2)$  sau  $O(N^2 \log N)$  în funcție de implementare.

**Optimizare.** Optimizarea prezentată pentru Cerința 1 poate fi utilizată și în acest caz.

Cu ajutorul acestei optimizări se pot obține 14 puncte din cele 41 acordate pentru Subtask-ul 5 (în plus față de cele acordate pentru subtask-urile precedente).

**Subtask 5 (41 puncte).** În mod asemănător cu Subtask-ul 2, se face normalizarea și se fixează capătul dreapta  $dr$ . Fie  $\{st_1, st_2, st_3, \dots, st_k\}$  mulțimea de capete stanga cu care  $dr$  formează o pereche de capete validă. Astfel numărul de rute valide care îl au capăt dreapta pe  $dr$  este:

$$2^{dr-st_1-1} + 2^{dr-st_2-1} + 2^{dr-st_3-1} + \dots + 2^{dr-st_k-1}$$

Relația se prelucurează și se obține:

$$2^{dr} \cdot \left( \frac{1}{2^{st_1+1}} + \frac{1}{2^{st_2+1}} + \frac{1}{2^{st_3+1}} + \dots + \frac{1}{2^{st_k+1}} \right)$$

Relația se înmulțește și se împarte cu  $2^N$  și se obține:

$$2^{dr-N} \cdot (2^{N-st_1-1} + 2^{N-st_2-1} + 2^{N-st_3-1} + \dots + 2^{N-st_k-1})$$

Suma  $2^{dr-st_1-1} + 2^{dr-st_2-1} + 2^{dr-st_3-1} + \dots + 2^{dr-st_k-1}$  se calculează în timpul parcurgerii pentru fixarea capătului dreapta.

Astfel complexitatea temporală este:  $O(N \log N)$ .

**1.3.2 \*Cod sursă**



**1.3.3 \*Rezolvare detaliată**

# Capitolul 2

## OJI 2021 - OSEPI

### 2.1 Labirint

#### Problema 1 - Labirint

100 de puncte

Un labirint este descris ca fiind o matrice binară cu  $N$  linii și  $M$  coloane, cu semnificația că 0 reprezintă o poziție liberă, iar 1 reprezintă o poziție în care se află un zid. Un drum în labirint este un traseu în matrice care începe cu poziția  $(1, 1)$  și ajunge în poziția  $(N, M)$  prin deplasare doar pe poziții care au valoarea 0 și sunt vecine cu poziția curentă, pe una din cele patru direcții: sus, jos, stânga, dreapta.

Lungimea unui drum este egală cu numărul de poziții vizitate.

Notăm cu  $d_0$  lungimea drumului minim de la poziția  $(1, 1)$  la poziția  $(N, M)$ . Fie  $d(i, j)$  lungimea drumului minim de la poziția  $(1, 1)$  la poziția  $(N, M)$ , dacă poziției  $(i, j)$  i se atribuie valoarea 0. Observăm că dacă poziția  $(i, j)$  conține inițial un 0, atunci  $d_0 = d(i, j)$ .

#### Cerințe

Pentru fiecare poziție  $(i, j)$ , să se verifice dacă  $d(i, j) < d_0$ .

#### Date de intrare

Pe prima linie a fișierului **labirint.in** se află două numere naturale  $N$  și  $M$ , dimensiunile matricei binare ce descrie labirintul, apoi pe următoarele  $N$  linii se vor afla câte  $M$  valori binare, ce reprezintă elementele matricei care descrie labirintul, neseperate prin spații.

#### Date de ieșire

În fișierul **labirint.out** se vor scrie  $N$  linii, iar pe fiecare linie se vor scrie  $M$  cifre, neseperate prin spații. Cifra a  $j$ -a de pe linia a  $i$ -a este 1 dacă și numai dacă  $d(i, j) < d_0$ , altfel este 0.

#### Restricții și precizări

- $1 \leq N \leq 1000$ .
- $1 \leq M \leq 1000$ .
- Pe pozițiile  $(1, 1)$  și  $(N, M)$  se vor afla valori 0.
- Se garantează că există un drum în matricea inițială între pozițiile  $(1, 1)$  și  $(N, M)$ .

#### Pentru 10 puncte

- $1 \leq N \leq 50$ .
- $1 \leq M \leq 50$ .
- $d_0 = N + M - 1$ .

#### Pentru alte 30 de puncte

- $1 \leq N \leq 50$ .

- $1 \leq M \leq 50$ .

**Exemple:**

labirint.in	labirint.out	Explicații
5 6	010000	Sunt 7 poziții cu valoarea 1 în labirint care dacă se înlocuiesc cu 0 determină obținerea unui drum de lungime mai mică decât $d_0 = 14$ . De exemplu, dacă am înlocui valoarea din $(1, 2)$ cu 0, am obține un drum de lungime $d(1, 2) = 12$ .
010001	000100	
000101	001001	
011001	010010	
010010	001000	
001000		

**Timp maxim** de executare/test: **0.5** secunde pe Windows, **0.5** secunde pe Linux

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

**Dimensiune** maximă a sursei: **15 KB**

### 2.1.1 Indicații de rezolvare

Propunător: prof. Gheorghe Manolache, stud. Ștefan-Cosmin Dăscălescu

Mai întâi vom afla, folosind *algoritmul lui Lee*, distanța minimă de la  $(1, 1)$  la  $(N, M)$ , notată în enunț cu  $d_0$ . Acum, diferența dintre soluția optimă și soluția parțială va consta în modul în care verificăm pentru fiecare poziție egală cu 1 dacă distanța se modifică.

**Soluție pentru 10 puncte.** Pentru primul grup de teste, deoarece  $d_0 = N + M - 1$ , se poate observa că nu va exista nicio zonă care să îmbunătățească răspunsul, deci se poate obține punctajul pe aceste teste afișând o matrice numai cu valori egale cu 0.

**Soluție pentru 40 de puncte.** Pentru cel de-al doilea grup de teste, se poate simula cu ajutorul *algoritmului lui Lee* înlocuirea fiecărei valori egale cu 1 și se va verifica dacă distanța de la  $(1, 1)$  la  $(N, M)$  s-a micșorat, afișându-se 1 sau 0, după caz, complexitatea acestui algoritm fiind  $O(N^2 M^2)$ . De notat că această soluție rezolvă corect și testele din primul grup 1.

**Soluție pentru 100 de puncte.** Pentru a ajunge la soluția optimă, se va observa faptul că nu e nevoie să rulăm algoritmul lui Lee de fiecare dată când modificăm valoarea unei poziții, fiind de ajuns *precalcularea* distanțelor atât din  $(1, 1)$  cât și din  $(N, M)$  către toate celelalte poziții din matrice. Astfel, pentru fiecare zonă  $(i, j)$  în care se poate ajunge atât din  $(1, 1)$ , cât și din  $(N, M)$ , distanța pe care o vom avea de la  $(1, 1)$  la  $(N, M)$  trecând printr-o poziție  $(i, j)$  egală inițial cu 1 va fi egală cu următoarea valoare:  $d(i, j) = dist_1(i, j) + dist_2(i, j) - 1$ , unde  $dist_1(i, j)$  reprezintă distanța de la  $(1, 1)$  la  $(i, j)$ , iar  $dist_2(i, j)$  reprezintă distanța de la  $(N, M)$  la  $(i, j)$ , fiind necesară scăderea lui 1 deoarece poziția  $(i, j)$  apare în ambele distanțe.

Complexitatea algoritmului pentru soluția ce obține 100 de puncte devine astfel  $O(N \cdot M)$ .

### 2.1.2 Cod sursă

**Obs:** [https://ebooks.infobits.ro/culegere\\_OJI\\_2021.pdf](https://ebooks.infobits.ro/culegere_OJI_2021.pdf)

### 2.1.3 \*Rezolvare detaliată

## 2.2 SDistanțe

### Problema 2 - SDistanțe

**100 de puncte**

Definim *distanța* dintre două șiruri de caractere de aceeași lungime ca fiind numărul minim de caractere ce trebuie modificate (înlocuite fiecare cu câte un alt caracter) în primul șir pentru a obține al doilea șir.

Vom nota distanța dintre șirurile  $a$  și  $b$  cu  $dist(a, b)$ .

De exemplu,  $\text{dist}(\text{"abc"}, \text{"aaa"}) = 2$  (înlocuim caracterul 'b' cu 'a', respectiv caracterul 'c' cu 'a'), iar  $\text{dist}(\text{"ABC"}, \text{"abc"}) = 3$  (literele mici se consideră diferite de cele mari).

Definim o *subsecvență* a unui șir  $s$  de caractere ca fiind un șir format din caractere de pe poziții consecutive din  $s$ . Considerăm două subsecvențe ca fiind distincte dacă încep sau se termină la poziții diferite. Vom nota cu  $s(i, j)$  subsecvența formată din caracterele indexate de la  $i$  la  $j$  ale șirului  $s$ . Șirurile se indexează de la 0.

Exemplu: pentru șirul  $s = \text{"abc"}$  subsecvențele sunt  $s(0; 0) = \text{"a"}$ ,  $s(1; 1) = \text{"b"}$ ,  $s(2; 2) = \text{"c"}$ ,  $s(0, 1) = \text{"ab"}$ ,  $s(1, 2) = \text{"bc"}$ ,  $s(0, 2) = \text{"abc"}$ , iar pentru șirul  $s = \text{"aa"}$  acestea sunt  $s(0, 0) = \text{"a"}$ ,  $s(1, 1) = \text{"a"}$ ,  $s(0, 1) = \text{"aa"}$ .

Se dă un șir de caractere  $s$ , care poate conține doar litere mici și mari ale alfabetului englez (de la 'a' la 'z' și de la 'A' la 'Z'). Pentru toate perechile neordonate de subsecvențe distincte ale șirului  $s$  care au lungimi egale, vrem să calculăm distanța dintre ele și să afișăm suma acestora modulo  $10^9 + 7$ .

### Cerințe

Formal, se cere suma valorilor  $\text{dist}(s(a, b), s(c, d))$ , pentru toți indicii  $a, b, c, d$  cu  $0 \leq a, b, c, d < |s|$ ,  $a < c$ ,  $a \leq b$ ,  $c \leq d$ ,  $b - a = d - c$ , **modulo**  $10^9 + 7$ .

$|s|$  reprezintă lungimea șirului  $s$ , care este indexat de la 0.

### Date de intrare

Pe singura linie a fișierului **sdistante.in** este șirul dat,  $s$ .

### Date de ieșire

Se va afișa pe singurul rând al fișierului **sdistance.out** un număr întreg reprezentând suma distanțelor, **modulo**  $10^9 + 7$ .

### Restricții și precizări

- $|s| \leq 4\,000\,000$ , unde  $|s|$  este lungimea șirului  $s$ .

#### Pentru 11 puncte

- $|s| \leq 20$
- $s$  conține doar litere mici

#### Pentru alte 5 puncte

- $|s| \leq 50$
- $s$  conține doar caracterele 'a' și 'b'

#### Pentru alte 15 puncte

- $|s| \leq 350$
- $s$  conține doar litere mici

#### Pentru alte 6 puncte

- $|s| \leq 1\,000$
- $s$  conține doar caracterele 'a' și 'b'

#### Pentru alte 30 de puncte

- $|s| \leq 5\,000$
- $s$  conține doar litere mici

#### Pentru alte 5 puncte

- $|s| \leq 100\,000$

- $s$  conține doar caracterele 'a' și 'b'

**Pentru alte 4 puncte**

- $|s| \leq 100\,000$
- $s$  conține doar litere mici

**Pentru alte 6 puncte**

- $|s| \leq 1\,000\,000$
- $s$  conține doar caracterele 'a' și 'b'

**Pentru alte 18 puncte**

- Fără alte restricții.

**Exemple:**

sdistance.in	sdistance.out	Explicații
abc	5	<ul style="list-style-type: none"> <li>• <math>dist(s(0,0), s(1,1)) = dist("a", "b") = 1</math></li> <li>• <math>dist(s(0,0), s(2,2)) = dist("a", "c") = 1</math></li> <li>• <math>dist(s(1,1), s(2,2)) = dist("b", "c") = 1</math></li> <li>• <math>dist(s(0,1), s(1,2)) = dist("ab", "bc") = 2</math></li> </ul>
aab	3	<ul style="list-style-type: none"> <li>• <math>dist(s(0,0), s(1,1)) = dist("a", "a") = 0</math></li> <li>• <math>dist(s(0,0), s(2,2)) = dist("a", "b") = 1</math></li> <li>• <math>dist(s(1,1), s(2,2)) = dist("a", "b") = 1</math></li> <li>• <math>dist(s(0,1), s(1,2)) = dist("aa", "ab") = 1</math></li> </ul>
ABa	5	<ul style="list-style-type: none"> <li>• <math>dist(s(0,0), s(1,1)) = dist("A", "B") = 1</math></li> <li>• <math>dist(s(0,0), s(2,2)) = dist("A", "a") = 1</math></li> <li>• <math>dist(s(1,1), s(2,2)) = dist("B", "a") = 1</math></li> <li>• <math>dist(s(0,1), s(1,2)) = dist("AB", "Ba") = 2</math></li> </ul>
aaaaabbbbaaaa	480	
abcdefghijklhizabcdefghijklhiz	7095	

**Timp maxim** de executare/test: **0.5** secunde pe Windows, **0.5** secunde pe Linux

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

**Dimensiune** maximă a sursei: **15 KB**

### 2.2.1 Indicații de rezolvare

Propunător: stud. Bogdan-Petru Pop

Vom nota cu  $N$  lungimea șirului  $s$ .

**Soluție în  $O(N^4)$  - 11-16 puncte în funcție de implementare.** Pentru această soluție, ajunge să găsim un algoritm simplu de calcul al distanței între două șiruri și să îl aplicăm pe toate perechile de subsecvențe. Observăm că  $dist(a, b)$  este numărul de poziții  $i$  pentru care  $a(i)$  este diferit de  $b(i)$  ( $0 \leq i < |a|$ ). Cu această observație, obținem un algoritm liniar care va fi aplicat pe  $O(N^3)$  perechi de stringuri (toate perechile  $(i, i + lungime)$ ,  $(j, j + lungime)$ ,  $0 < i < j < |a|$ ;  $i + lungime < |a|$ ), obținând un algoritm de complexitate  $O(N^4)$ .

---

#### Algorithm 1 Soluție brute-force

---

$N \leftarrow lungime(s)$

$raspuns \leftarrow 0$

**for**  $i \leftarrow 0 \dots N - 1$  **do**

**for**  $j \leftarrow i + 1 \dots N - 1$  **do**

**for**  $lungime \leftarrow 1 \dots N - j$  **do**

**for**  $indice \leftarrow 0 \dots lungime$  **do**

**if**  $s(i + indice) \neq s(j + indice)$  **then**  $raspuns \leftarrow raspuns + 1 \bmod 1.000.000.007$

---

**Soluție în  $O(N^3)$  - 31 de puncte.** Încercăm să optimizăm soluția anterioară. Observăm că un triplet  $(i, j, indice)$  contribuie de mai multe ori la răspuns. Mai exact, observăm că dacă  $s(i + indice) \neq s(j + indice)$ , atunci adunăm 1 la răspuns pentru toate valorile lui *lungime* mai mari sau egale cu *indice*. Astfel, putem renunța la a itera cu variabila *lungime*, iar în locul acestei iterații să adunăm la răspuns numărul de valori ale lui *indice* pentru care se adună 1 la verificarea  $s(i + indice) \neq s(j + indice)$ . Calculând exact, vom adăuga la răspuns  $N - (j + indice)$  pentru fiecare triplet care verifică proprietatea anterioară. Obținem astfel un algoritm de complexitate  $O(N^3)$ .

**Soluție în  $O(N^2)$  - 67 de puncte.** Asemănător cu pasul anterior, vom încerca să fixăm o anumită pereche de indici cu caractere diferite și să observăm cu cât contribuie la rezultat. Pentru asta, vom rescrie în funcție de pozițiile pe care le comparăm condițiile ca restul variabilelor să fie valide.

Notăm cu  $a$  și  $b$ ,  $a < b$ , pozițiile pe care le comparăm la fiecare pas și scriem condițiile pentru ca  $i, j, lungime, indice$  să fie valide.

$$\begin{cases} a = i + indice \\ b = j + indice \\ i \geq 0 \\ j + lungime < N \end{cases}$$

Vom rescrie indicii pentru indexa în funcție de pozițiile pe care le comparăm.

$$\begin{cases} i = a - indice \\ j = b - indice \\ a - indice \geq 0 \\ N - lungime < b - indice \end{cases}$$

Observați că  $i$  și  $j$  sunt unic determinați dacă știm toate valorile  $a, b, lungime, indice$ , deci nu este necesar să păstrăm primele două ecuații pentru a verifica validitatea unei soluții. Astfel, dacă avem  $a$  și  $b$  fixate, condițiile pe care trebuie să le îndeplinească *lungime* și *indice* pentru a fi valide sunt:

$$\begin{cases} indice \leq a \\ lungime - indice < N - b \end{cases}$$

Acest sistem are  $(a + 1) * (N - b)$  soluții care sunt perechi de numere naturale (orice valoare de la 0 la  $a$  este valabilă pentru *indice*, iar pentru *lungime - indice* putem alege orice valoare de la 0 la  $N - b - 1$ , determinând unic o valoare validă a variabilei *lungime* pentru orice valoare fixată a variabilei *indice*). Astfel, perechea  $(a, b)$  contribuie cu  $(a + 1) * (N - b)$  la răspuns.

**Soluție în  $O(N * sigma)$  unde  $sigma$  este mărimea alfabetului - 82 de puncte.** Vom fixa doar poziția  $b$  și vom încerca să găsim contribuția acesteia la răspuns. Aceasta va fi  $(a_1 + 1) * (N - b) + (a_2 + 1) * (N - b) + \dots + (a_k + 1) * (N - b)$ , unde  $a_1, a_2, \dots, a_k$  sunt indicii pentru care  $s(b) \neq s(a_i)$  și  $a_i < b$ . Dacă dăm factor comun  $(N - b)$ , obținem:

$$[(a_1 + 1) + (a_2 + 1) + \dots + (a_k + 1)] * (N - b)$$

Astfel, problema se reduce la calculul eficient al sumei  $(a_1 + 1) + (a_2 + 1) + \dots + (a_k + 1)$ . Pentru a obține suma acestor indici, ne vom folosi de proprietatea lor: conțin o altă valoare decât  $s(b)$ .

Putem ține un *vector de sume*  $sum(c)$  care conține suma de  $i + 1$  pentru indicii  $i$  mai mici decât  $b$ -ul curent pentru care  $s(i) = c$ . Acest vector poate fi actualizat în  $O(1)$  la fiecare pas, adăugând  $b + 1$  la  $sum(s(b))$ . Atunci când vrem să obținem suma  $a$ -urilor din expresia de mai sus, vom însuma pur și simplu toate valorile  $sum(c)$ ,  $c \neq s(b)$ , ceea ce poate fi realizat în  $O(sigma)$ .

**Soluție în  $O(N)$  - 100 de puncte.** Observăm că suma  $a_1 + a_2 + \dots + a_k$  este de fapt  $1 + 2 + 3 + \dots + (b - 1) - (b_1 + b_2 + \dots + b_l)$ , unde  $b_1, b_2, \dots, b_l$  sunt toate pozițiile cu proprietatea  $b_i < b$ ;  $s(b_i) = s(b)$  (practic putem să scădem din suma tuturor pozițiilor anterioare suma pozițiilor cu un caracter egal cu  $s(b)$  și rămâne suma celor care nu sunt egale cu  $b$ ). Analog  $(a_1 + 1) + (a_2 + 1) + \dots + (a_k + 1) = (0 + 1) + (1 + 1) + (2 + 1) + \dots + ((b - 1) + 1) - ((b_1 + 1) + (b_2 + 1) + \dots + (b_l + 1))$ . Suma anterioară poate fi rescris, folosind vectorul *sum*, ca:

$$\frac{b * (b + 1)}{2} - sum(b)$$

Această expresie poate fi calculată în  $O(1)$ , fapt ce ne duce la complexitatea finală  $O(N)$ .

Alternativ, putem calcula de la început cât ar fi răspunsul dacă toate caracterele din șir ar fi diferite, iar apoi să scădem numărul de "potriviri" (caractere egale pe aceeași poziție în 2 șiruri) între perechile de subsecvențe, cu o implementare asemănătoare ce folosește același vector *sum*.

### 2.2.2 Cod sursă

**Obs:** [https://ebooks.infobits.ro/culegere\\_OJI\\_2021.pdf](https://ebooks.infobits.ro/culegere_OJI_2021.pdf)

### 2.2.3 \*Rezolvare detaliată

## 2.3 Tort

### Problema 3 - Tort

100 de puncte

Alexandra, prințesa Regatului Visurilor a primit un tort și vrea să îl împartă cu prietenii ei. Astfel ea va organiza o petrecere unde îi va invita. Tortul Alexandrei este format din  $N$  bucăți, iar a  $i$ -a bucată are  $a_i$  cireșe. Alexandra va împărți tortul în mai multe secvențe continue de bucăți, astfel încât fiecare bucată este inclusă în exact o secvență, și fiecare secvență conține cel puțin o bucată de tort. Prima secvență - cea care conține prima bucată - o va mânca în noaptea de dinaintea petrecerii, iar restul bucăților le va da celorlalți prieteni invitați. Pentru a nu îi supăra, Alexandra vrea ca fiecare secvență dată unui prieten să conțină la fel de multe cireșe ca oricare altă secvență dată unui prieten (dar nu neapărat la fel de multe cireșe ca aceea mâncată de ea înaintea petrecerii). Alexandra trebuie să invite cel puțin un prieten la petrecere.

### Cerințe

Dându-se  $N$  și șirul  $a$ , să se afle numărul de moduri în care Alexandra ar putea să împartă tortul în secvențe continue, astfel încât să se respecte condițiile din enunț. Două moduri de a împărți tortul se consideră a fi distincte dacă și numai dacă există în unul o secvență care nu există în celălalt (dacă am reprezenta un mod de împărțire în secvențe prin intermediul șirului crescător al indicilor de început pentru fiecare secvență din acea împărțire, două moduri de împărțire sunt distincte dacă șirurile de indici asociate lor sunt diferite).

Formal, dându-se un șir de numere, se vrea să aflăm numărul de moduri de a împărți șirul în cel puțin două subsecvențe, astfel încât sumele elementelor tuturor subsecvențelor să fie egale, prima putând să aibă suma elementelor diferită de a celorlalte.

### Date de intrare

Prima linie a fișierului de intrare **tort.in** conține numărul  $N$ .

A doua linie conține valorile  $a_1, \dots, a_N$ , separate prin spații.

### Date de ieșire

Singura linie a fișierului de ieșire **tort.out** va conține numărul cerut.

### Restricții și precizări

- $1 \leq N \leq 200\,000$
- $1 \leq a_1, \dots, a_N \leq 400\,000$
- $a_1 + \dots + a_N \leq 400\,000$

### Pentru 12 puncte

- $1 \leq N \leq 20$

### Pentru alte 12 puncte

- $1 \leq N \leq 100$ ,  $a_1 = \dots = a_N = 1$ .

**Pentru alte 20 de puncte**

- $1 \leq N \leq 100$

**Pentru alte 28 de puncte**

- $1 \leq N \leq 1000$ ,  $a_1 + \dots + a_N \leq 2000$

**Pentru alte 28 de puncte**

- Fără alte restricții.

**Exemple:**

tort.in	tort.out	Explicații
5 1 1 2 1 1	6	<p>Împărțirile valide sunt:</p> <ol style="list-style-type: none"> <li>1. [1]; [1; 2; 1; 1]</li> <li>2. [1; 1]; [2; 1; 1]</li> <li>3. [1; 1]; [2]; [1; 1]</li> <li>4. [1; 1; 2]; [1; 1]</li> <li>5. [1; 1; 2]; [1]; [1]</li> <li>6. [1; 1; 2; 1]; [1]</li> </ol>

**Timp maxim** de executare/test: **0.5** secunde pe Windows, **0.5** secunde pe Linux

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

**Dimensiune** maximă a sursei: **15 KB**

### 2.3.1 Indicații de rezolvare

Propunător: prof. Marius Nicoli, Colegiul Național "Frații Buzești", Craiova

**Soluție pentru  $N \leq 20$ .** O abordare prin care se generează toate modurile de a descompune în secvențe șirul dat obține punctele la aceste teste. De exemplu, se pot genera toate șirurile de 0 și de 1 care încep cu 1, mai conțin încă un 1 cel puțin și care au lungimea  $n$ . Pozițiile pe care se află valoarea 1 sunt începuturile de secvențe ale unei descompunerii. Timpul de executare este de ordinul  $2^n \cdot n$ .

**Soluție pentru  $N \leq 100$ ,  $a_1 = \dots = a_N = 1$ .** Toate elementele fiind egale cu 1, odată ce fixăm prima secvență între indicii 1 și  $i$ , la soluție trebuie adunat numărul de divizori ai valorii  $n - i$  (suma numerelor de la poziția  $i$  până la poziția  $n$ ). Timpul de executare este deci  $n\sqrt{n}$  dar se poate obține unul mai bun dacă ne gândim că este vorba de *suma divizorilor* pentru fiecare număr de la 1 la  $n - 1$ , iar aceste valori se pot calcula folosind *ciurul lui Eratostene*.

**Soluție pentru  $N \leq 100$ .** Fixăm de asemenea prima secvență iar pentru restul descompunerii fixăm mai întâi suma comună pe care o dorim în restul secvențelor și apoi facem verificarea dacă descompunerea cu elementele fixate înainte este posibilă. Timpul de calcul este de ordinul  $n^2 \times$  (*suma valorilor din șirul dat*).

**Soluție pentru  $N \leq 2000$ ,  $a_1 + \dots + a_N \leq 4000$ .** Odată ce fixăm prima secvență (până la poziția  $i - 1$ ) ne dăm seama că suma comună din celelalte secvențe nu poate fi decât un divizor al valorii  $S_i$  ( $S_i$  = suma elementelor de la poziția  $i$  până la poziția  $n$ ). Astfel este necesară verificarea doar pentru valorile acestor divizori. Avem timp de ordinul  $n^2$  de la fixarea secvenței inițiale și de la verificare și se adaugă costul obținerii divizorilor lui  $S_i$  (fie obținem divizorii la întâlnirea elementului curent cu timp de ordin  $\sqrt{S_i}$ , fie îi *precalculăm* folosind *Ciurul lui Eratostene*).

**Soluție pentru 100 de puncte.** Facem o parcurgere de la final și acumulăm la o sumă  $s$  valorile din șir pe măsură ce le întâlnim, marcând într-un *vector de frecvență* în dreptul sumelor obținute. Pe acest vector, pentru valori naturale  $i$ , începând cu 1, verificăm cât putem merge plecând de la indicele  $i$  și avansând din  $i$  în  $i$  pe elemente marcate, fiecare pas făcut reprezentând de altfel o soluție. Este de fapt o *simulare* a algoritmului de la *Ciurul lui Eratostene*, aceasta fiind și cel care dă complexitatea în timp a unei soluții care se încadrează în timp pe toate testele.



**2.3.2 Cod sursă**

**Obs:** [https://ebooks.infobits.ro/culegere\\_OJI\\_2021.pdf](https://ebooks.infobits.ro/culegere_OJI_2021.pdf)

**2.3.3 \*Rezolvare detaliată**

# Capitolul 3

## OJI 2020

### 3.1 alinieri

#### Problema 1 - alinieri

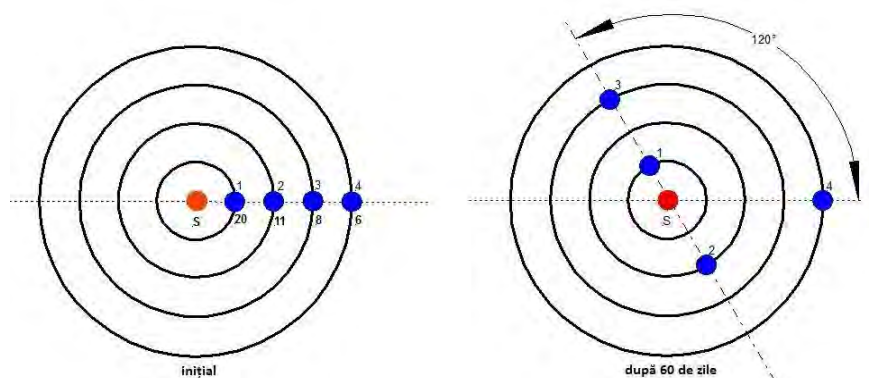
100 de puncte

Se consideră modelul unui sistem solar format din  $N$  planete care se rotesc în jurul unei stele  $S$ , în sens trigonometric. Traiectoriile planetelor se consideră circulare și de raze diferite, iar vitezele de rotație ale planetelor în jurul stelei sunt numere naturale și sunt exprimate în grade pe zi ( $^{\circ}/zi$ ).

#### Cerințe

Cunoscând numărul de planete  $N$  și vitezele lor de rotație  $V_i$ ,  $1 \leq i \leq N$  precum și două numere naturale  $P$  și  $Z$ , să se determine numărul  $A$  de alinieri a câte minimum  $P$  planete, pe o dreaptă ce trece prin centrul stelei  $S$ , după trecerea celor  $Z$  zile. Evoluția sistemului solar începe cu toate planetele așezate orizontal, în dreapta stelei  $S$ .

#### Exemplu



#### Date de intrare

Fișierul de intrare **alinieri.in** conține pe prima linie, în această ordine, numerele naturale  $N$ ,  $P$  și  $Z$ , iar pe-a doua linie,  $N$  numere naturale  $V_i$ ,  $1 \leq i \leq N$  cu semnificația de mai sus. Numerele aflate pe aceeași linie a fișierului sunt separate prin câte un spațiu.

#### Date de ieșire

Fișierul de ieșire **alinieri.out** va conține pe prima linie numărul  $A$ , cu semnificația de mai sus.

#### Restricții și precizări

- $2 \leq P \leq N \leq 10^5$
- $1 \leq Z \leq 10^6$
- $1 \leq V_i \leq 10^3$ ,  $1 \leq i \leq N$
- Pentru teste în valoare de 30 de puncte  $1 \leq Z \leq 1000$

- Pentru teste în valoare de 30 de puncte  $1 \leq N \leq 100$
- Pentru teste în valoare de 30 de puncte  $2 \leq P \leq 9$
- Se vor lua în considerare doar alinierile de la sfârșitul fiecărei zile (ora 24:00), când planetele și-au încheiat parcursul zilnic.

**Exemple:**

alinieri.in	alinieri.out	Explicații
4 3 365 20 11 8 6	8	N=4, P=3, Z=365 și V1-4 = [20,11,8,6] Prima aliniere a minimum 3 planete dintre cele 4 planete are loc după 60 de zile (conform figurii de mai sus). Evoluția celor 4 planete este următoarea: -planeta 1 efectuează 3 rotații complete și încă 1200, -planeta 2 efectuează o rotație completă și încă 3000, -planeta 3 efectuează o rotație completă și încă 1200, -planeta 4 efectuează exact o rotație. Următoarele alinieri a minimum 3 din cele 4 planete au loc după 90, 120, 180, 240, 270, 300, 360 zile. Deci în 365 zile vor avea loc 8 alinieri.
7 3 2020 10 20 10 15 20 10 20	3928	N=7, P=3, Z=2020 și V1-7 = [10,20,10,15,20,10,20] în cele 2020 de zile au avut loc 3928 alinieri a minimum 3 planete din cele 7 planete ce formează sistemul solar.
6 3 658903 17 24 12 150 200 12	58568	N=6, P=3, Z=658903 și V1-6 = [17,24,12,150,200,12] în cele 658903 de zile au avut loc 58568 alinieri a minimum 3 planete din cele 6 planete ce formează sistemul solar.

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

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

**Dimensiune** maximă a sursei: **10 KB**

### 3.1.1 Indicații de rezolvare

prof. Cheșcă Ciprian, Liceul Tehnologic "Grigore C. Moisil" Buzău

#### Varianta 1

Având în vedere că vitezele de rotație ale planetelor sunt numere naturale, atunci cu siguranță putem afirma că după un număr de 360 de zile toate planetele se vor găsi aliniate în punctul de unde au plecat, deoarece  $360 * k$  este multiplu de 360, pentru  $k$  număr natural.

Să observăm în continuare că planetele se pot alinia pe o dreaptă care trece prin centrul stelei  $S$  atât de o parte cât și de cealaltă a stelei, ceea ce înseamnă că sunt suficiente 180 de zile pentru ca toate planetele să fie cu siguranță aliniate. În această situație o planetă se poate găsi ori în punctul inițial de plecare ori decalată cu  $180^\circ$ , adică de cealaltă parte a stelei  $S$ .

Așadar în intervalul cuprins între 1 și 180 de zile un număr de  $P$  plante vor fi cu siguranță aliniate. Pentru a afla exact zilele în care loc aceste alinieri vom folosi un vector de contorizări și pentru fiecare zi, cuprinsă între 1 și 180, vom determina poziția exactă a fiecărei planete, ținând cont că această poziție este dată de un unghi cuprins între 1 și 180. Așadar această simulare determină, pentru fiecare zi, câte planete sunt aliniate și pe ce poziții. Se determină apoi câte astfel de alinieri se fac într-un interval de 360 zile și acest număr se înmulțește cu numărul  $Z/360$ . Se mai determină separat câte alinieri mai au loc în  $Z \% 360$  și se adună la totalul anterior.

#### Varianta 2

La cele explicate în varianta anterioară putem face observația că analizând datele de intrare și anume că vitezele sunt numere naturale  $\leq 10^3$  și că sunt în total maxim  $10^5$  planete este clar că vor fi planete care au aceeași viteză și se comportă similar.

Deci putem de la început să grupăm planetele care au aceeași viteză  $\% 180$  utilizând încă un *vector de frecvență*. Această soluție obține 100 puncte.

## 3.1.2 Cod sursă

Listing 3.1.1: alinieri\_bogdan\_100.cpp

---

```

1  /// sursa 100 p
2  /// std. Bogdan Sitaru
3  #include <bits/stdc++.h>
4
5  using namespace std;
6
7  typedef long long LL;
8
9  const int NMAX = 185;
10 const int mod = 180;
11
12 int N, P, Z;
13 int f[NMAX], now[NMAX];
14
15 int main()
16 {
17     freopen("alinieri.in", "r", stdin);
18     freopen("alinieri.out", "w", stdout);
19
20     assert( scanf("%d%d%d", &N, &P, &Z) == 3 );
21     assert(2 <= P && P <= 1e5);
22     assert(2 <= N && N <= 1e5);
23     assert(1 <= Z && Z <= 1e6);
24     for(int i = 0; i < N; i++)
25     {
26         int x;
27         assert( scanf("%d", &x) == 1 );
28         assert(1 <= x && x <= 1e3);
29         f[x % mod]++;
30     }
31
32     LL tot = 0, ans = 0;
33     int r = Z % mod;
34     for(int i = 0; i < 180; i++)
35     {
36         for(int j = 0; j < 180; j++)    now[j] = 0;
37         for(int j = 0; j < 180; j++)
38             now[(j * (i + 1)) % mod] += f[j];
39
40         int cnt = 0;
41         for(int j = 0; j < 180; j++)
42             cnt += now[j] >= P;
43         tot += cnt;
44         if(i < r)    ans += cnt;
45     }
46     ans += tot * (Z / 180);
47
48     printf("%lld\n", ans);
49
50     return 0;
51 }
52 /*
53 execution time : 0.234 s
54 */

```

---

Listing 3.1.2: alinieri\_CC1.cpp

---

```

1  // prof. Chesca Ciprian - sursa C - 100 p
2  // Complexitate O(N)
3
4  #include <bits/stdc++.h>
5
6  #define nmax 1001
7
8  using namespace std;
9
10 int N, P, Z, gp[nmax], w[nmax];
11
12 int main()

```

---

```

13 {
14     freopen("alinieri.in", "r", stdin);
15     freopen("alinieri.out", "w", stdout);
16
17     int t,i,x,na,nta;
18
19     scanf("%d%d%d", &N, &P, &Z);
20
21     // construiesc vectori de frecventa cu datele de intrare
22     for(i=1;i<=N;i++)
23     {
24         scanf("%d",&x);
25         gp[x]++;
26     }
27
28     nta=0;
29
30     // cate alinieri au loc in 360 zile
31     for(t=1;t<=360;t++)
32     {
33         na=0;
34         for(i=0;i<=179;i++)
35             w[i]=0;
36         for(i=1;i<=1000;i++)
37             if (gp[i]) w[(t*i)%180]+=gp[i];
38
39         for(i=0;i<=179;i++)
40             if (w[i]>=P) na++;
41         nta+=na;
42     }
43
44     // cate alinieri au loc in Z/360
45     nta*=(Z/360);
46
47     // cate alinieri mai au loc in Z%360
48     for(t=1;t<=Z%360;t++)
49     {
50         na=0;
51         for(i=0;i<=179;i++)
52             w[i]=0;
53         for(i=1;i<=1000;i++)
54             if (gp[i]) w[(t*i)%180]+=gp[i];
55
56         for(i=0;i<=179;i++)
57             if (w[i]>=P) na++;
58         nta+=na;
59     }
60
61     printf("%d\n",nta);
62
63     return 0;
64 }
65 /*
66 execution time : 0.188 s
67 */

```

Listing 3.1.3: alinieri\_CC2.cpp

```

1 // prof. Chesca Ciprian - sursa C++
2 // Complexitate O(365*N)
3
4 #include <fstream>
5 #define nmax 1000001
6 #define smax 400
7
8 using namespace std;
9
10 ifstream f("alinieri.in");
11 ofstream g("alinieri.out");
12
13 int N,P,Z,v[nmax],w[smax];
14
15 int main()
16 {
17     int t,i,na,nta;

```

```

18
19 f>>N>>P>>Z;
20 for (i=1; i<=N; i++)
21     f>>v[i];
22
23 nta=0;
24 for (t=1; t<=360; t++)
25 {
26     na=0;
27     for (i=0; i<=179; i++)
28         w[i]=0;
29
30     for (i=1; i<=N; i++)
31         w[(t*v[i])%180]++;
32
33
34     for (i=0; i<=179; i++)
35         if (w[i]>=P) na++;
36
37     nta+=na;
38 }
39
40 nta*=(Z/360);
41
42 for (t=1; t<=Z%360; t++)
43 {
44     na=0;
45     for (i=0; i<=179; i++)
46         w[i]=0;
47
48     for (i=1; i<=N; i++)
49         w[(t*v[i])%180]++;
50
51     for (i=0; i<=179; i++)
52         if (w[i]>=P) na++;
53
54     nta+=na;
55 }
56 }
57
58 g<<nta<<"\n";
59
60 f.close();
61 g.close();
62 return 0;
63 }
64 /*
65 execution time : 1.749 s
66 */

```

Listing 3.1.4: alinieri\_CC3.cpp

```

1 // prof. Chesca Ciprian
2 // Complexitate O(Z)
3
4 #include <fstream>
5 #define nmax 181
6
7 using namespace std;
8
9 ifstream f("alinieri.in");
10 ofstream g("alinieri.out");
11
12 int N,P,Z, gp[nmax], w[nmax], na, nta=0;
13
14 int main()
15 {
16     int t,i,x;
17
18     f>>N>>P>>Z;
19     for (i=1; i<=N; i++)
20     {
21         f>>x;
22         gp[x%180]++;
23     }

```

```

24
25 for (t=1;t<=Z;t++)
26 {
27     na=0;
28     for (i=0;i<=179;i++)
29         w[i]=0;
30
31     for (i=0;i<=179;i++)
32         if (gp[i]) w[(t*i)%180]+=gp[i];
33
34     for (i=0;i<=179;i++)
35         if (w[i]>=P) na++;
36
37     nta+=na;
38 }
39
40 g << nta << "\n";
41
42 f.close();
43 g.close();
44 return 0;
45 }
46 /*
47 execution time : 7.141 s
48 */

```

Listing 3.1.5: alinieri\_CC4.cpp

```

1 // prof. Chesca Ciprian - sursa C++ - 100 p
2 // Complexitate O(N)
3
4 #include <bits/stdc++.h>
5 #define nmax 181
6
7 using namespace std;
8
9 ifstream f("alinieri.in");
10 ofstream g("alinieri.out");
11
12 int N,P,Z, gp[nmax], w[nmax], na, nta, nta1=0, nta2=0;
13
14 int main()
15 {
16     int t,i,x;
17
18     f>>N>>P>>Z;
19
20     for (i=1;i<=N;i++)
21     {
22         f>>x;
23         assert(x>=1 && x <=1000);
24         gp[x%180]++;
25     }
26
27     assert(N>=1 && N<=100000);
28     assert(P>=2 && P<=N);
29     assert(Z>=1 && Z<=1000000);
30
31     for (t=1;t<=180;t++)
32     {
33         na=0;
34         for (i=0;i<=179;i++)
35             w[i]=0;
36
37         for (i=0;i<=179;i++)
38             w[(t*i)%180]+=gp[i];
39
40         for (i=0;i<=179;i++)
41             if (w[i]>=P) na++;
42
43         nta+=na;
44     }
45
46     nta*=2*(Z/360);
47

```

---

```

48 for (t=1;t<=Z%360;t++)
49 {
50     na=0;
51     for (i=0;i<=179;i++)
52         w[i]=0;
53
54     for (i=0;i<=179;i++)
55         w[(t*i)%180]+=gp[i];
56
57     for (i=0;i<=179;i++)
58         if (w[i]>=P) na++;
59
60     nta+=na;
61 }
62
63 g << nta << "\n";
64
65 f.close();
66 g.close();
67 return 0;
68 }
69 /*
70 execution time : 0.094 s
71 */

```

---

### 3.1.3 \*Rezolvare detaliată

## 3.2 arh

### Problema 2 - arh

100 de puncte

Dexter și-a definit propriul algoritm de arhivare a șirului favorit  $T$ , șir format numai din litere mici ale alfabetului englez. șirul arhivat, notat cu  $S$ , poate fi format din cifre, litere mici ale alfabetului englez, parantezele drepte  $[ ]$  și parantezele rotunde  $( )$ , precum și caractere  $*$ .

Fixi, curios din fire, descoperă algoritmul și încearcă să dezarhiveze șirul  $S$ , prin efectuarea unor transformări repetate. O transformare poate fi de unul dintre cele 3 tipuri de mai jos, unde s-a notat cu  $C$  o secvență din  $S$  formată numai din litere.

- 1) O secvență a lui  $S$  de forma  $n(C)$ , unde  $n$  este numărul natural poziționat imediat înaintea parantezei rotunde, se transformă într-o secvență  $D$  obținută prin scrierea concatenată, de  $n$  ori, a șirului  $C$ .

Exemplu: pentru secvența  $10(ab)$  avem  $n = 10$  și se obține secvența  
 $D = ababababababababab$ .

- 2) O secvență a lui  $S$  de forma  $[*C]$  se transformă într-o secvență palindromică de lungime pară, obținută prin concatenarea secvenței  $C$  cu oglinditul lui  $C$ .

Exemplu: din secvența  $[*abc]$  se obține secvența palindromică de lungime pară  $abccba$

- 3) O secvență a lui  $S$  de forma  $[C*]$  se transformă într-o secvență palindromică de lungime impară, obținută prin concatenarea secvenței  $C$  cu oglinditul lui  $C$  din care s-a eliminat primul caracter.

Exemplu: din secvența  $[abc*]$  se obține secvența palindromică de lungime impară  $abcba$ .

Un șir se consideră dezarhivat dacă este format numai din litere mici ale alfabetului englez.

### Cerințe

Fiind dat șirul arhivat  $S$  să se determine numărul de transformări, de cele 3 tipuri de mai sus, realizate de Fixi în cadrul algoritmului de dezarhivare, precum și forma finală dezarhivată  $T$  a șirului  $S$ .

### Date de intrare

Fișierul de intrare **arh.in** conține șirul de caractere arhivat  $S$ .



### Date de ieșire

Fișierul de ieșire **arh.out** conține obligatoriu două linii. Pe prima linie numărul de transformări cerut, iar pe linia a doua șirul de caractere cerut,  $T$ .

### Restricții și precizări

- 0 ≤ lungimea șirului arhivat  $S \leq 10\,000$ ; 0 ≤ lungimea șirului dezarhivat  $T \leq 100\,000$ ;
- $1 < n \leq 1\,000$ ;
- O secvență a unui șir este o succesiune de caractere aflate pe poziții consecutive în șir;
- În șirul  $S$  o cifră poate apărea numai imediat înaintea unei paranteze rotunde deschise sau imediat înaintea unei alte cifre; fiecare paranteză rotundă deschisă are imediat înaintea sa cel puțin o cifră; toate parantezele, drepte sau rotunde, se închid corect. Caracterul  $*$  poate apărea numai imediat după o paranteză dreaptă deschisă sau imediat înaintea unei paranteze drepte închise;
- O secvență a unui șir este palindromică dacă primul element al secvenței este egal cu ultimul, al doilea cu penultimul etc; oglinditul unei secvențe se obține prin scriere în ordine inversă a caracterelor sale;
- Se acordă 20% din punctajul fiecărui test pentru scrierea corectă a numărului cerut și 80% din punctajul fiecărui test pentru scrierea corectă a șirului cerut;
- Pentru 30 de puncte șirul arhivat  $S$  poate fi dezarhivat numai cu transformări de tipul 1;
- Pentru alte 30 de puncte șirul arhivat  $S$  poate fi dezarhivat numai cu transformări de tipurile 2 și 3.

### Exemple:

arh.in	arh.out	Explicații
2(a)[*a2(b)]xy[2(c)b*]d	5 aaabbbbaxycbced	2(a) = <sub>i</sub> aa 2(b) = <sub>i</sub> bb [*a2(b)] = <sub>i</sub> [*abb] = <sub>i</sub> abbbba 2(c) = <sub>i</sub> cc [2(c)b*] = <sub>i</sub> [ccb*] = <sub>i</sub> ccbcc
2(ab[cd*])a3(xyz)	3 abcdcabcdcaxyzxyzxyz	3(xyz) = <sub>i</sub> xyzxyzxyz [cd*] = <sub>i</sub> cdc 2(ab[cd*]) = <sub>i</sub> 2(abcdc) = <sub>i</sub> abcdcabcdc
abcd	0 abcd	Nu este nevoie de nicio transformare, iar șirul dezarhivat este identic cu cel arhivat.

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

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

**Dimensiune** maximă a sursei: **10 KB**

#### 3.2.1 Indicații de rezolvare

prof. Nodea Eugen, Colegiul Național "Tudor Vladimirescu", Tg-Jiu

Soluție care folosește *recursivitatea indirectă*

Problema se încadrează în categoria: evaluare de expresii (*parser* de expresii).

Șirul arhivat  $S$  poate fi format din niciuna, una sau mai multe secvențe ce pot fi rezolvate folosind cele 3 tipuri de transformări.

Descompunem șirul în atomi, prin atomi înșelegând:

- număr
- paranteză rotundă deschisă
- paranteză rotundă închisă

- paranteză pătrată deschisă
- paranteză pătrată închisă

Se definesc astfel funcțiile

- eval1(); - calculează parantezele ( )
- eval2(); - calculează parantezele [ ] în funcție de tipul 2 sau 3
- sir(); - formează șir de caractere dezarhivat
- numar(); - formează numărul  $n$
- eval(); - funcția care apelează / "adună" toate celelalte funcții

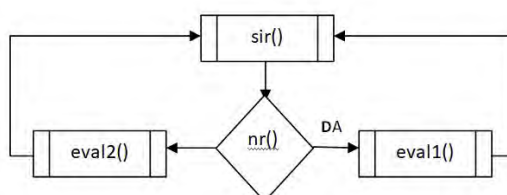
Funcția eval()

```

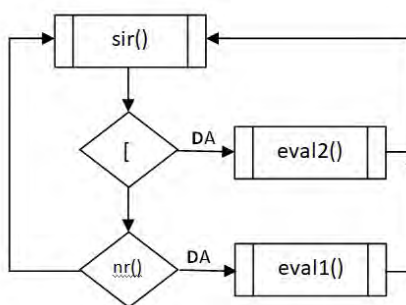
T = ""; ///initial sirul dezarhivat este vid
Cat timp ( ! sfarsit_sir_S ) executa
    T += sir(); /// daca avem secventa de litere, acestea se concateneaza la T
    daca nr() atunci /// avem transformare de tipul 1 delimitat de parenteze ( )
        C = eval1()
        T += n * C /// concatenam de n ori secventa de litere C
    altfel daca avem [ atunci /// avem transf. de tipul 2/3 delimitat de paranteze [ ]
        C = eval2()
        altfel T += sir()
sf_cat_timp

```

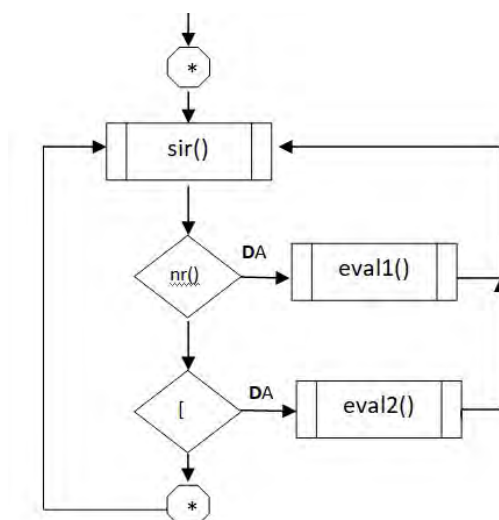
Schematic putem sintetiza funcția eval() astfel:



Funcția eval1():



Funcția eval2():



Complexitatea:  $O(n)$

O altă abordare în rezolvarea problemei o reprezintă lucru cu stive (câte o *stivă* pentru fiecare transformare în parte), ceva mai complexă ca implementare, dar care asigură punctaj maxim.

### 3.2.2 Cod sursă

Listing 3.2.1: arh\_eugen.cpp

```

1  /// Sursa oficiala - 100p
2  /// prof. Eugen Nodea
3
4  #include <bits/stdc++.h>
5
6  using namespace std;
7
8  ifstream f("arh.in");
9  ofstream g("arh.out");
10
11 string T, S;
12 int i, L, nra;
13
14 string eval1();    /// calculeaza parantezele ()
15 string eval2();    /// calculeaza parantezele []
16 string sir();      /// formeaza sir de caractere
17 int numar();       /// formeaza numarul
18
19 void eval()
20 {
21     while (i < T.size())
22     {
23         S += sir();
24         int nr = numar();
25         if (nr)
26         {
27             string r1 = eval1();
28             for(int j=1; j<=nr; ++j)
29                 S += r1;
30         }
31         else
32             if (T[i] == '[') S += eval2();
33             else S += sir();
34     }
35 }
36
37 string eval1()
38 {
39     string r = "";
40     if (T[i] == '(') i++;
41     while (T[i] != ')')
42     {

```

```

43     r += sir();
44     if (T[i] == '[') r += eval2();
45
46     r += sir();
47
48     int nr = numar();
49     if (nr)
50     {
51         string r1 = eval1();
52         for(int j=1; j<=nr; ++j)
53             r += r1;
54     }
55     r += sir();
56 }
57 i++;
58
59 return r;
60 }
61
62 string eval2()
63 {
64     int par = 0;
65     string r = "", r1 = "";
66     if (T[i] == '[') i++;
67     if (T[i] == '*') par = 1, i++;
68
69     while (T[i] != ']')
70     {
71         r += sir();
72
73         if (T[i] == '[') r += eval2();
74
75         if (isdigit(T[i]))
76         {
77             int nr = numar();
78             string r1 = eval1();
79             for(int j=1; j<=nr; ++j)
80                 r += r1;
81         }
82
83         r += sir();
84
85         if (T[i] == '*') i++;
86     }
87
88     if (T[i] == ']')
89     {
90         r1 = r;
91         if (!par)
92         {
93             r1.erase(r1.end()-1);
94         }
95         reverse(r1.begin(), r1.end());
96
97         i++;
98     }
99     return r + r1;
100 }
101
102 string sir()
103 {
104     string r = "";
105     while(isalpha(T[i]))
106         r += T[i], i++;
107     return r;
108 }
109
110 int numar()
111 {
112     int r = 0;
113     while (isdigit(T[i]))
114     {
115         r = r * 10 + (T[i] - '0');
116         i++;
117     }
118     return r;

```

```

119 }
120
121 int main()
122 {
123     f >> T;
124
125     L = T.size();
126     for(int i=0; i<L; ++i)
127         if (T[i] == '(' || T[i] == '[') nra++;
128     g << nra << "\n";
129
130     eval();
131
132     g << S;
133
134     return 0;
135 }

```

Listing 3.2.2: arh\_razvan.cpp

```

1  /// Sursa - recursivitate 100p
2  /// student Razvan Turturica
3  #include <fstream>
4  #include <cassert>
5
6  using namespace std;
7
8  ifstream cin("arh.in");
9  ofstream cout("arh.out");
10
11 string s;
12 char ans[200010];
13 int arhivari = 0;
14
15 int number(int &poz)
16 {
17     int rsp = 0;
18     while(s[poz] >= '0' && s[poz] <= '9')
19     {
20         rsp = rsp * 10 + s[poz] - '0';
21         poz++;
22     }
23     return rsp;
24 }
25
26 int mksir(int &poz, int start)
27 {
28
29     if(s[poz] == '[')
30     {
31         arhivari++;
32         if(s[poz + 1] == '*')
33         {
34             poz += 2;
35             int stop = mksir(poz, start);
36             int l = stop - start + 1;
37             for(int i = 0 ; i < l ; i++){
38                 ans[stop + 1 + i] = ans[stop - i];
39             }
40             poz += 1;
41             return mksir(poz, stop + 1 + 1);
42         }
43     }
44     else
45     {
46         poz++;
47         int stop = mksir(poz, start);
48         int l = stop - start + 1;
49         for(int i = 1 ; i < l ; i++)
50         {
51             ans[stop + i] = ans[stop - i];
52         }
53         poz += 2;
54         return mksir(poz, stop + 1);
55     }
56 }

```

```

56     else
57     {
58         if(s[poz] >= '0' && s[poz] <= '9')
59         {
60             arhivari++;
61             int rep = number(poz);
62             poz++;
63             int stop = mksir(poz, start);
64             int l = stop - start + 1;
65             for(int i = 1 ; i < rep ; i++)
66             {
67                 for(int j = start ; j <= stop ; j++)
68                 {
69                     ans[start + i * l + j - start] = ans[j];
70                 }
71             }
72             poz++;
73             return mksir(poz, stop + (rep-1) * l + 1);
74         }
75     else
76     {
77         if(s[poz] >= 'a' && s[poz] <= 'z')
78         {
79             int stop = start;
80             while(s[poz] >= 'a' && s[poz] <= 'z')
81             {
82                 ans[stop++] = s[poz];
83                 poz++;
84             }
85             return mksir(poz, stop);
86         }
87     else
88     {
89         return start - 1;
90     }
91 }
92
93 int main()
94 {
95     cin >> s;
96     assert(s.size() <= 10000);
97     int x = 0;
98     int l = -1;
99     do
100     {
101         l = mksir(x, l + 1);
102     } while(x < s.size());
103
104     assert(l <= 199999);
105     cout << arhivari << '\n' << ans;
106     return 0;
107 }

```

Listing 3.2.3: arh\_tamio.cpp

```

1  /// Sursa 100p
2  /// student Tamio-Vesa N.
3
4  #include <bits/stdc++.h>
5
6  using namespace std;
7
8  enum token_type
9  {
10     LBRACK = 0, RBRACK = 1, LPAR = 2, RPAR = 3, STAR = 4, INT = 5, STR = 6
11 };
12
13 struct token
14 {
15     token_type t;
16     int x;
17     string s;
18 };
19
20 vector<token> tokensise(string s)

```

```

21 {
22     vector<token> ret;
23     string current_s;
24     int current_x = 0;
25     int i = 0;
26
27     static token_type tab[256];
28     tab['['] = token_type::LBRACK;
29     tab[']'] = token_type::RBRACK;
30     tab['('] = token_type::LPAR;
31     tab[')'] = token_type::RPAR;
32     tab['*'] = token_type::STAR;
33     ///cerr << s << endl;
34
35 INIT:
36     if(i == (int)s.size()) goto END;
37     if(isdigit(s[i]))
38     {
39         ///cerr << '%' << endl;
40         current_x = s[i++] - '0';
41         goto INTEGER;
42     }
43     if(isalpha(s[i]))
44     {
45         current_s.push_back(s[i++]);
46         goto STRING;
47     }
48     ret.push_back(token{ tab[(unsigned)s[i++]], 0, "" });
49     goto INIT;
50
51 STRING:
52     if(isalpha(s[i]))
53     {
54         current_s.push_back(s[i++]);
55         goto STRING;
56     }
57     ret.push_back(token{ token_type::STR, 0, current_s });
58     current_s.clear();
59     goto INIT;
60
61 INTEGER:
62     if(isdigit(s[i]))
63     {
64         current_x = 10 * current_x + s[i++] - '0';
65         goto INTEGER;
66     }
67     ret.push_back(token{ token_type::INT, current_x, "" });
68     current_x = 0;
69     goto INIT;
70
71 END:
72     return ret;
73 }
74
75 enum expr_type
76 {
77     odd_pal, even_pal, repetition, basic, concat, nil
78 };
79
80 struct expr
81 {
82     expr_type t;
83     string s;
84     int reps;
85     expr *leftson = 0, *rightson = 0;
86 };
87
88 expr* parse(vector<token>::iterator& i, vector<token>::iterator e);
89
90 expr* parse_lbrack(vector<token>::iterator& i, vector<token>::iterator e)
91 {
92     assert(i->t == token_type::LBRACK);
93     expr *ret = new expr;
94
95     ++i;
96

```

```

97     if(i->t == token_type::STAR)
98     {
99         ret->t = expr_type::even_pal;
100        ++i;
101        ret->leftson = parse(i, e);
102        assert(i->t == token_type::RBRACK);
103        ++i;
104    }
105    else
106    {
107        ret->t = expr_type::odd_pal;
108        ret->leftson = parse(i, e);
109        assert(i->t == token_type::STAR);
110        ++i;
111        assert(i->t == token_type::RBRACK);
112        ++i;
113    }
114    return ret;
115 }
116
117 expr* parse_num(vector<token>::iterator& i, vector<token>::iterator e)
118 {
119     assert(i->t == token_type::INT);
120     expr *ret = new expr;
121     ret->t = expr_type::repetition;
122     ret->reps = i->x;
123     ++i;
124
125     assert(i->t == token_type::LPAR);
126     ++i;
127
128     ret->leftson = parse(i, e);
129
130     assert(i->t == token_type::RPAR);
131     ++i;
132
133     return ret;
134 }
135
136 expr* parse(vector<token>::iterator& i, vector<token>::iterator e)
137 {
138     expr* ret = new expr;
139     if(i == e || i->t == token_type::RPAR ||
140        i->t == token_type::STAR ||
141        i->t == token_type::RBRACK)
142         ret->t = nil;
143     else
144         if(i->t == token_type::INT)
145         {
146             ret->t = concat;
147             ret->leftson = parse_num(i, e);
148             ret->rightson = parse(i, e);
149         }
150     else
151         if(i->t == token_type::LBRACK)
152         {
153             ret->t = concat;
154             ret->leftson = parse_lbrack(i, e);
155             ret->rightson = parse(i, e);
156         }
157     else
158         if(i->t == token_type::STR)
159         {
160             ret->t = concat;
161             ret->leftson = new expr;
162
163             ret->leftson->t = basic;
164             ret->leftson->s = i->s;
165             ++i;
166
167             ret->rightson = parse(i, e);
168         }
169     return ret;
170 }
171
172 void evaluate(expr *e, string& s)

```



```

173 {
174     assert(e);
175     if(e->t == expr_type::concat)
176     {
177         evaluate(e->leftson, s);
178         evaluate(e->rightson, s);
179     }
180     else
181     if(e->t == expr_type::odd_pal)
182     {
183         string tmp;
184         evaluate(e->leftson, tmp);
185         s.append(tmp);
186
187         tmp.pop_back();
188         reverse(begin(tmp), end(tmp));
189         s.append(tmp);
190     }
191     else
192     if(e->t == expr_type::even_pal)
193     {
194         string tmp;
195         evaluate(e->leftson, tmp);
196         s.append(tmp);
197
198         reverse(begin(tmp), end(tmp));
199         s.append(tmp);
200     }
201     else
202     if(e->t == expr_type::repetition)
203     {
204         string tmp;
205         evaluate(e->leftson, tmp);
206         for(int i = 0; i < e->reps; ++i)
207             s.append(tmp);
208     }
209     else
210     if(e->t == expr_type::basic)
211     {
212         s.append(e->s);
213     }
214     else
215     if(e->t == expr_type::nil)
216     {
217         //
218     }
219 }
220
221 int find_operation_count(expr *e)
222 {
223     return e == nullptr ? 0 :
224         find_operation_count(e->leftson) + find_operation_count(e->rightson) +
225         (e->t == even_pal || e->t == odd_pal || e->t == repetition ? 1 : 0);
226 }
227
228 int main()
229 {
230     ifstream f("arh.in");
231     ofstream g("arh.out");
232
233     string s;
234     f >> s;
235
236     vector<token> tok = tokensise(s);
237     auto it = begin(tok);
238
239     ///for(auto x : tok)
240     ///cerr << x.t << ' ' << x.s << ' ' << x.x << endl;
241
242     expr *parse_tree = parse(it, end(tok));
243
244     string ret;
245     evaluate(parse_tree, ret);
246
247     g << find_operation_count(parse_tree) << endl << ret << endl;
248     return 0;

```

### 3.2.3 \*Rezolvare detaliată

## 3.3 leftmax

### Problema 3 - leftmax

100 de puncte

În clasa lui Dexter sunt  $N$  elevi de înălțimi distincte. La ora de sport, ei sunt așezați în linie, de la stânga la dreapta. Profesorul lor, Johnny, va selecta pentru un exercițiu elevi aflați pe poziții consecutive în linie, astfel încât cel mai înalt elev dintre cei selectați să se afle în prima jumătate a acestora.

De exemplu, dacă elevii au, în ordine, înălțimile 1, 5, 4, atunci profesorul poate să îi selecteze pe cei cu înălțimile 5 și 4, dar nu poate să îi selecteze pe cei cu înălțimile 1 și 5. Desigur, există mai multe moduri de a selecta elevii astfel încât să fie satisfăcută condiția de mai sus. Profesorul Johnny ar vrea să afle în câte moduri se poate face acest lucru.

### Cerințe

Dându-se  $N$  și înălțimile elevilor din clasă, aflați în câte moduri pot fi selectați oricâți elevi aflați pe poziții consecutive, astfel încât să fie îndeplinită condiția din enunț.

### Date de intrare

Fișierul de intrare **leftmax.in** conține, pe prima linie, numărul  $N$ , iar pe a doua linie înălțimile elevilor în ordinea în care sunt așezați în linie.

### Date de ieșire

Fișierul de ieșire **leftmax.out** conține pe prima linie răspunsul la cerință, sub formă de rest al împărțirii la 1.000.000.007 (modulo 1.000.000.007).

### Restricții și precizări

- $1 \leq N \leq 100.000$
- $1 \leq \text{înălțimea oricărui elev} \leq N$
- Dacă se selectează un număr impar de elevi, atunci considerăm că cel din mijlocul selecției se află în prima jumătate a elevilor selectați
- Pentru 10 puncte,  $N \leq 1.000$  și elevii sunt ordonați descrescător după înălțime
- Pentru alte 35 de puncte,  $N \leq 1.000$
- Pentru alte 20 de puncte,  $N \leq 30.000$

### Exemple:

leftmax.in	leftmax.out	Explicații
4 1 4 2 3	8	Sunt 4 moduri de a selecta câte un singur elev. Este un singur mod de a selecta câte doi elevi (cei cu înălțimile 4, 2). Sunt 2 moduri de a selecta câte 3 elevi (cu înălțimile 4,2,3 și 1, 4,2). Este un singur mod de a selecta toți cei 4 elevi. În total sunt 8 modulo 1.000.000.007 = 8 moduri.
7 1 2 3 4 5 6 7	7	Se pot selecta doar câte un singur elev.
7 7 6 5 4 3 2 1	28	Se pot selecta oricâți elevi pe poziții consecutive

**Timp maxim** de executare/test: **0.7** secunde

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

**Dimensiune** maximă a sursei: **10 KB**

### 3.3.1 Indicații de rezolvare

stud. Tamio-Vesa Nakajima, Oxford University

#### Complexitatea $O(N^3)$ : 15 puncte

Pentru această complexitate, putem considera toate subsecvențele de elevi consecutivi, să găsim elevul de înălțime maximă iterând prin elementele secvențelor. Rămâne doar să numărăm câte subsecvențe satisfac condiția din enunț.

#### Complexitatea $O(N^2)$ : 45 puncte

Pentru această complexitate, pornim de la soluția de 15 puncte, dar observăm că, dacă parcurgem subsecvențele în ordine crescătoare după poziția primului elev din subsecvență, iar în caz de egalitate în ordine crescătoare după poziția ultimului elev din subsecvență, atunci putem să aflăm poziția elevului de înălțime maximă în timp constant pentru fiecare subsecvență.

Acest lucru se datorează faptului ca elevul de înălțime maximă printre cei de la al  $i$ -lea până la al  $(j + 1)$ -lea elev este fie cel de înălțime maximă printre cei de la al  $i$ -lea până la al  $j$ -lea elev (pe care deja îl cunoaștem), fie fix elevul de pe poziția  $j + 1$ .

#### Complexitatea $O(N)$ : 100 puncte

Vom stoca înălțimile copiilor într-un vector. Într-o primă fază, putem folosi o *stivă* pentru a calcula două șiruri  $st$  și  $dr$  unde  $st(i)$  reprezintă poziția cea mai mică  $j$  pentru care toți copiii între  $j$  și  $i$  sunt mai mici sau egali cu cel pe poziția  $i$ , și  $dr(i)$  reprezintă poziția cea mai mare  $j$  pentru care toți copiii între  $i$  și  $j$  sunt mai mici sau egale cu cel pe poziția  $i$ . Această *precalculare* poate fi realizată în timp linear.

Acum, iterăm prin șir. Când suntem la poziția  $i$  considerăm toate subsecvențele de copii pentru care copilul cel mai înalt este pe poziția  $i$ . Din definiția lui  $st$ ,  $dr$  reiese că, dacă subsecvențele acestea sunt de la al  $x$ -lea copil la al  $y$ -lea copil, atunci  $st(i) \leq x \leq i \leq y \leq dr(i)$ .

Pe noi ne interesează acele subsecvențe unde, mai mult, avem că  $i - x \leq y - i$ . Dacă notăm pe  $i - st(i)$  cu  $L$  și pe  $dr(i) - i$  cu  $R$  atunci se poate demonstra că sunt  $(\min(L, R) + 1) * (R - L + 2 + \max(R, L)) / 2$ .

Formula se demonstrează astfel.

Luăm două cazuri, unul unde  $L \leq R$ , unul unde  $L > R$ .

- Dacă  $L \leq R$  atunci formula devine  $(L + 1) * (2 * R - L) / 2$ . În cazul acesta, putem fixa pe  $i - x$  de la 0 la  $L$ , iar pentru fiecare mod de a îl fixa, putem fixa pe  $y - i$  de la  $i - x$  la  $R$ . Astfel, vrem suma progresiei aritmetice cu  $L + 1$  termeni unde primul termen este  $R$  iar ultimul termen este  $R - L$ , adică fix formula data mai sus.
- Dacă  $L > R$ , atunci formula devine  $(R + 1) * (R + 2) / 2$ . În cazul acesta, putem fixa pe  $y - i$  de la 0 la  $R$ , iar pentru fiecare mod de a îl fixa, putem fixa pe  $i - x$  de la 0 la  $y - i$ . Astfel, vrem suma progresiei aritmetice cu  $R + 1$  termeni unde primul termen este 1 iar ultimul termen este  $R + 1$ , adică fix formula data mai sus.

Tot ce rămâne acum este însumarea acestora pentru toate valorile lui  $i$ .

### 3.3.2 Cod sursă

Listing 3.3.1: leftmax\_O(nxn).cpp

```

1  //brut - 15p
2  #include <bits/stdc++.h>
3  using namespace std;
4
5  using ll = long long;
6  using cd = complex<double>;
7
8  int main()
9  {
10     ifstream f("leftmax.in");
11     ofstream g("leftmax.out");
12
13     int n;
14
```

```

15     f >> n;
16
17     vector<int> v(n);
18     for(auto& x : v)
19         f >> x;
20
21     ll ret = 0;
22
23     for(int i = 0; i < n; ++i)
24     {
25         int k = i;
26         for(int j = i; j < n; ++j)
27         {
28             int k = i;
29             for(int l = i; l <= j; ++l)
30                 if(v[l] > v[k]) k = l;
31             if(2 * (k - i) <= j - i)
32                 ++ret;
33         }
34     }
35
36     g << ret % (1000 * 1000 * 1000 + 7) << endl;
37
38     return 0;
39 }

```

Listing 3.3.2: leftmax\_optimn.cpp

```

1  /// sursa oficiala 100p O(N)
2  /// student Tamio-Vesa Nakajima
3
4  #include <bits/stdc++.h>
5
6  using namespace std;
7
8  using ll = long long;
9  using cd = complex<double>;
10
11 int main()
12 {
13     ifstream f("leftmax.in");
14     ofstream g("leftmax.out");
15
16     // Citim inputul.
17     int n;
18     f >> n;
19     vector<int> v(n);
20     for(auto& x : v)
21         f >> x;
22
23     // Folosim o stiva pentru a calcula sirurile
24     // st si dr din solutie.
25     vector<int> st(n), dr(n);
26     stack<int> stiva;
27     for(int i = 0; i < n; ++i)
28     {
29         while(!stiva.empty() && v[stiva.top()] < v[i])
30         {
31             dr[stiva.top()] = i - 1;
32             stiva.pop();
33         }
34
35         st[i] = (stiva.empty() ? 0 : stiva.top() + 1);
36         stiva.push(i);
37     }
38
39     while(!stiva.empty())
40     {
41         dr[stiva.top()] = n - 1;
42         stiva.pop();
43     }
44
45     // Acum vom itera prin sir, retinand solutia in ret.
46     ll ret = 0;
47     for(int i = 0; i < n; ++i)

```

---

```

48     {
49         const ll L = i - st[i], R = dr[i] - i;
50         ret += (min(R, L) + 1) * (R - L + 2 + max(R, L)) / 2;
51     }
52
53     // Afisam solutia.
54     g << ret % 1000000007 << endl;
55
56     return 0;
57 }

```

---

Listing 3.3.3: leftmax.razvan\_O(n).cpp

---

```

1  ///100 p complexitate O(n)
2  /// std. Razvan Turturica
3
4  #include <fstream>
5  #include <stack>
6  #include <cassert>
7
8  using namespace std;
9
10 ifstream cin("leftmax.in");
11 ofstream cout("leftmax.out");
12
13 stack<int> st;
14 int v[100010], l[100010], r[100010], fv[100010];
15
16 long long int gauss(long long int x)
17 {
18     return x * (x + 1) / 2;
19 }
20
21 int main()
22 {
23     int n;
24     cin >> n;
25     assert(n >= 1);
26     assert(n <= 100000);
27
28     for(int i = 1 ; i <= n ; i++)
29     {
30         cin >> v[i];
31         assert(v[i] <= n);
32         assert(v[i] >= 1);
33         fv[v[i]]++;
34     }
35
36     for(int i = 1 ; i <= n ; i++)
37     {
38         assert(fv[i] == 1);
39     }
40
41     v[0] = v[n + 1] = n + 1;
42     st.push(0);
43     for(int i = 1 ; i <= n ; i++)
44     {
45         while(v[st.top()] < v[i])
46             st.pop();
47         l[i] = i - st.top() - 1;
48         st.push(i);
49     }
50
51     while(st.size())
52         st.pop();
53     st.push(n+1);
54
55     for(int i = n ; i >= 1 ; i--)
56     {
57         while(v[st.top()] < v[i])
58             st.pop();
59         r[i] = st.top() - i - 1;
60         st.push(i);
61     }
62

```

---

```

63     long long int ans = 0;
64     for(int i = 1 ; i <= n ; i++){
65         if(r[i] <= l[i])
66             ans += gauss(r[i] + 1);
67         else
68             ans += gauss(r[i] + 1) - gauss(r[i]-l[i]);
69     }
70
71     cout << ans % 1000000007;
72     return 0;
73 }

```

---

Listing 3.3.4: leftmax\_razvan\_O(nlog).cpp

---

```

1  /// 100 p complexitate O(nlogn)
2  /// std. Razvan Turturica
3  #include <fstream>
4
5  using namespace std;
6
7  ifstream cin("leftmax.in");
8  ofstream cout("leftmax.out");
9
10 int v[100010], arb[400010];
11
12 long long int gauss(long long int x)
13 {
14     return x * (x + 1) / 2;
15 }
16
17 void update(int nod, int left, int right, int poz)
18 {
19     int middle = (left + right) / 2;
20     if(left == right)
21     {
22         arb[nod] = 1;
23         return;
24     }
25     if(poz <= middle)
26         update(2 * nod, left, middle, poz);
27     else
28         update(2 * nod + 1, middle + 1, right, poz);
29     arb[nod] = arb[2 * nod] + arb[2 * nod + 1];
30 }
31
32 int getLeft(int nod, int left, int right, int poz)
33 {
34     int middle = (left + right) / 2;
35     if(left == right)
36         return left;
37     if(arb[2 * nod] < poz)
38         return getLeft(2 * nod + 1, middle + 1, right, poz - arb[2 * nod]);
39     else
40         return getLeft(2 * nod, left, middle, poz);
41 }
42
43 int getRight(int nod, int left, int right, int poz)
44 {
45     int middle = (left + right) / 2;
46     if(left == right)
47         return left;
48     if(arb[2 * nod + 1] < poz)
49         return getRight(2 * nod, left, middle, poz - arb[2 * nod + 1]);
50     else
51         return getRight(2 * nod + 1, middle + 1, right, poz);
52 }
53
54 int getSum(int nod, int left, int right, int L, int R)
55 {
56     int middle = (left + right) / 2;
57     if(L <= left && right <= R)
58         return arb[nod];
59     if(left > R || right < L)
60         return 0;
61     return getSum(2 * nod, left, middle, L, R) +

```

```

62         getSum(2 * nod + 1, middle + 1, right, L, R);
63     }
64
65     int main()
66     {
67         int n;
68         cin >> n;
69         for(int i = 1 ; i <= n ; i++)
70         {
71             int x;
72             cin >> x;
73             v[x] = i;
74         }
75
76         long long int ans = 0;
77         int l, r, x;
78         for(int i = n ; i >= 1 ; i--)
79         {
80             x = getSum(1, 1, n, 1, v[i]);
81             if(x != 0)
82                 l = v[i] - getLeft(1, 1, n, x) - 1;
83             else
84                 l = v[i] - 1;
85             x = getSum(1, 1, n, v[i], n);
86             if(x != 0)
87                 r = getRight(1, 1, n, x) - v[i] - 1;
88             else
89                 r = n - v[i];
90             l = max(l, 0);
91             r = max(r, 0);
92             update(1, 1, n, v[i]);
93             if(r <= l)
94                 ans += gauss(r + 1);
95             else
96                 ans += gauss(r + 1) - gauss(r - 1);
97         }
98
99         cout << ans % 1000000007;
100         return 0;
101     }

```

Listing 3.3.5: leftmax\_razvan\_O(nn).cpp

```

1  /// 45 p complexitate O(nxn)
2  /// std. Razvan Turturica
3
4  #include <fstream>
5
6  using namespace std;
7
8  ifstream cin("leftmax.in");
9  ofstream cout("leftmax.out");
10
11 int v[100010];
12
13 int main()
14 {
15     int n;
16     cin >> n;
17     for(int i = 1 ; i <= n ; i++)
18     {
19         cin >> v[i];
20     }
21
22     long long int ans = 0;
23     for(int i = 1 ; i <= n ; i++)
24     {
25         int maxi = 0;
26         for(int j = i ; j <= n ; j++)
27         {
28             if(v[j] > v[maxi])
29                 maxi = j;
30             if(maxi <= (i + j) / 2)
31                 ans++;
32         }
33     }

```

---

```
33     }
34
35     cout << ans % 10000000007;
36     return 0;
37 }
```

---

### 3.3.3 \*Rezolvare detaliată



# Capitolul 4

## OJI 2019

### 4.1 pif

#### Problema 1 - pif

90 de puncte

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

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

#### Cerințe

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

#### Date de intrare

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

#### Date de ieșire

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

#### Restricții și precizări

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

#### Exemple

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

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

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

**Dimensiune** maximă a sursei: **15 KB**

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

#### 4.1.1 Indicații de rezolvare

Descrierea unor soluții posibile

Structurile de date utilizate: vectori

Gradul de dificultate: 2

**Soluția 1 - 30 p** (sursa: pif\_30p-1.cpp)

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

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

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

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

Funcția returnează numărul de fapte bune rămase nerealizate.

Complexitate:  $O(k^{\frac{n}{zf+zb}})$

**Soluția 2 - 70 p** (sursa: pif\_70p-1.cpp)

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

Complexitate:  $O(k \cdot n)$

**Soluția 3 - 30 p** (sursa: pif\_30p-2.cpp)

Pentru cazul  $zt = zv = 1$  numărul de fapte bune este același indiferent de ordinea în care ar fi alese persoanele vârstnice și tinere, iar numărul de persoane noi care intră în joc în fiecare zi

este dat de un șir Fibonacci generalizat în care fiecare valoare se obține prin însumarea ultimelor  $k$  valori anterioare:  $x_n = \sum_{i=n-k}^{n-1} x_i$ .

Folosim un vector în care memorăm aceste valori.

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

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

Complexitate:  $O(n)$

**Soluția 4 - 50 p** (sursa: pif\_50p.cpp)

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

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

**Soluția 5 - 70 p** (sursa: pif\_70p-2.cpp)

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

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

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

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

Complexitate  $O(k \cdot n)$

**Soluția 6 - ?? p** (sursa pif\_??p.cpp)

**Soluția 7 - 90 p** (sursa pif\_90p.cpp)

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

Complexitate  $O(n)$

**Soluția 8 - 70 p** (sursa: pif\_70p-3.cpp)

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

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

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

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

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

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

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

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

În consecință avem:

$$t[i] = t[i - zt] + t[i - 2 * zt] + \dots + t[i - kt * zt] + \\ + v[i - zv] + v[i - 2 * zv] + \dots + v[i - kt * zv]$$

(vom considera nule valorile cu index negativ)

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

$$v[i] = t[i - (1 + kt) * zt] + t[i - (2 + kt) * zt] + \dots + t[i - (kv + kt) * zt] + \\ + v[i - (1 + kt) * zv] + v[i - (2 + kt) * zv] + \dots + v[i - (kv + kt) * zv]$$

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

#### Soluția 9 - 90 p (sursa: pif\_90p\_2.cpp)

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

De exemplu, dacă am calcula

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

care de fapt este

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

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

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

Complexitate  $O(n)$

### 4.1.2 Cod sursă

Listing 4.1.1: pif\_30p\_1.cpp

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

```

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

```

Listing 4.1.2: pif\_30p-2.cpp

```

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

```

Listing 4.1.3: pif\_50p.cpp

```

1  #include <fstream>
2
3  using namespace std;
4
5  ifstream in("pif.in");
6  ofstream out("pif.out");
7
8  int n,k,zv,zf;
9

```

```

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

```

Listing 4.1.4: pif\_70p\_1.cpp

```

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

```

```

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

```

Listing 4.1.5: pif\_70p\_2.cpp

```

1  #include <fstream>
2
3  using namespace std;
4
5  ifstream in("pif.in");
6  ofstream out("pif.out");
7
8  int n,k,zv,zf;
9  int nf[1000010],nb[1000010],s;
10
11 int main()
12 {
13     in>>n>>k>>zf>>zv;
14     nb[0]=1;
15     int vk=k/2;
16     if(k%2==1)
17         vk++;
18
19     if(zf<zv)
20     {
21         for(int i=0;i<=n;i++)
22         {
23             for(int j=1;j<=vk;j++)
24             {
25                 if(i+j*zf<=n)
26                 {
27                     nf[i+j*zf]+=nf[i];
28                     nf[i+j*zf]%=1234567;

```

```

29         }
30         else
31         {
32             s+=nf[i];
33             s%=1234567;
34         }
35
36         if (i+j*zb<=n)
37         {
38             nf[i+j*zb]+=nb[i];
39             nf[i+j*zb]%=1234567;
40         }
41         else
42         {
43             s+=nb[i];
44             s%=1234567;
45         }
46     }
47
48     for(int j=vk+1; j<=k; j++)
49     {
50         if (i+j*zf<=n)
51         {
52             nb[i+j*zf]+=nf[i];
53             nb[i+j*zf]%=1234567;
54         }
55         else
56         {
57             s+=nf[i];
58             s%=1234567;
59         }
60
61         if (i+j*zb<=n)
62         {
63             nb[i+j*zb]+=nb[i];
64             nb[i+j*zb]%=1234567;
65         }
66         else
67         {
68             s+=nb[i];
69             s%=1234567;
70         }
71     }
72 }
73 }
74 else
75 {
76     for(int i=0; i<=n; i++)
77     {
78         for(int j=1; j<=vk; j++)
79         {
80             if (i+j*zf<=n)
81             {
82                 nb[i+j*zf]+=nf[i];
83                 nb[i+j*zf]%=1234567;
84             }
85             else
86             {
87                 s+=nf[i];
88                 s%=1234567;
89             }
90
91             if (i+j*zb<=n)
92             {
93                 nb[i+j*zb]+=nb[i];
94                 nb[i+j*zb]%=1234567;
95             }
96             else
97             {
98                 s+=nb[i];
99                 s%=1234567;
100             }
101         }
102     }
103     for(int j=vk+1; j<=k; j++)
104     {

```



```

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

```

Listing 4.1.6: pif\_70p\_3.cpp

```

1 // O(n*k)
2 #include <stdio.h>
3
4 #define MOD 1234567
5 #define MAXN 1000001
6
7 int b[MAXN], f[MAXN];
8
9 int main()
10 {
11     FILE *fin = fopen("pif.in", "r");
12     FILE *fout = fopen("pif.out", "w");
13     int n, k, zf, zb, kf, kb, of, ob;
14
15     fscanf(fin, "%d %d %d %d", &n, &k, &zf, &zb);
16     if (zb < zf)
17     {
18         kb = (k + 1)/2;
19         kf = k - kb;
20         ob = 0;
21         of = kb;
22     }
23     else
24     {
25         kf = (k + 1)/2;
26         kb = k - kf;
27         of = 0;
28         ob = kf;
29     }
30
31     int target = k;
32     b[0] = 1;
33     for (int i = 1; i <= n; i++)
34     {
35         for (int j = 1; j <= kb && (j + ob)*zb <= i; j++)
36             b[i] = (b[i] + b[i - (j + ob)*zb]) % MOD;
37         for (int j = 1; j <= kb && (j + ob)*zf <= i; j++)
38             b[i] = (b[i] + f[i - (j + ob)*zf]) % MOD;
39         for (int j = 1; j <= kf && (j + of)*zb <= i; j++)
40             f[i] = (f[i] + b[i - (j + of)*zb]) % MOD;
41         for (int j = 1; j <= kf && (j + of)*zf <= i; j++)
42             f[i] = (f[i] + f[i - (j + of)*zf]) % MOD;
43
44         target = (target + 1LL*(b[i] + f[i])*(k - 1)) % MOD;
45     }

```

```

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

```

Listing 4.1.7: pif\_90p\_1.cpp

```

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

```

```

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

```

Listing 4.1.8: pif\_90p\_2.cpp

```

1 // O(n + k)
2 #include <stdio.h>

```

---

```

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

```

---

Listing 4.1.9: pif\_90p\_3.cpp

---

```

1  #include <iostream>
2  #include <fstream>
3  #include <vector>
4  #include <algorithm>
5
6  using namespace std;
7
8  static const int kModulo = 1234567;
9
10 int main()
11 {
12     ifstream cin("0-pif.in");

```

```

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

```

### 4.1.3 \*Rezolvare detaliată

## 4.2 traseu

### Problema 2 - traseu

90 de puncte

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

### Cerințe

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

### Date de intrare

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

### Date de ieșire

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

### Restricții și precizări

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

### Exemple

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

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

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

**Dimensiune** maximă a sursei: **15 KB**

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

### 4.2.1 Indicații de rezolvare

Descriere a unei/unor soluții posibile

Gradul de dificultate: 2

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

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

Pentru fiecare poziție  $(x1, y1)$  din matrice se parcurg pozițiile  $(x2, y2)$  aflate spre **Sud** și spre **Est**,  $x1 \leq x2 \leq N$  și  $y1 \leq y2 \leq M$  și se verifică dacă  $a[x1][y1] < a[x2][y2]$ .

**Soluția 2 - 90 puncte** –  $O(N * M * N)$  (sursa: traseu90\_1.cpp)

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

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

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

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

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

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

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

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

**Soluția 3 - 90 puncte** –  $O(N * M * \min(M, N))$  (sursa: traseu90\_2.cpp)

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

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

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

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

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

Bazându-ne pe aceste observații putem defini:

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

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

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

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

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

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

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

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

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

**Soluția 4 - 90 puncte** –  $O(N * M * (M + N))$  (sursa: traseu90\_3.cpp)

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

Conform observației de mai sus, are sens să definim:

$$\min[i][j] = \min\{A[p][q] \mid 1 \leq p \leq i \text{ și } 1 \leq q \leq j\}$$

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

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

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

Distingem următoarele cazuri:

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

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

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

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

**Soluția 5 - 90 puncte** –  $O(N * M * N)$  (sursa: traseu90\_4.cpp)

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

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

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

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

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

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

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

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

#### 4.2.2 Cod sursă



---

```

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

```

---

Listing 4.2.2: traseu90\_1.cpp

---

```

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

```

```

42     {
43         row[i].next = row[i].prev = &row[i];
44         for (int j = 0; j < n; j++)
45         {
46             fscanf(f, "%d", &v); v--;
47             NODE[v] = insert(row[i].prev, i, j, v);
48         }
49     }
50
51     int ans = 0;
52     for (int v = 0; v < m*n; v++)
53     {
54         struct NODE *x = NODE[v];
55         for (int i = x->r; i < m; i++)
56         {
57             struct NODE *y = row[i].prev;
58             int d = i - x->r + y->c - x->c;
59             if (y->next != y && x->c <= y->c && d > ans) ans = d;
60         }
61         ddelete(x);
62     }
63
64     if(ans == 0) fprintf(g, "0");
65     else fprintf(g, "%d", ans + 1);
66
67     fclose(f);
68     fclose(g);
69
70     return 0;
71 }

```

Listing 4.2.3: traseu90\_2.cpp

```

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

```

```

43     }
44
45     for (int i = 0; i < m; i++)
46         for (int j = 0; j < n; j++)
47             l[i][j] = (j == 0 || a[i][j] < l[i][j-1]) ? a[i][j] : l[i][j-1];
48
49     for (int i = 0; i < m; i++)
50         for (int j = n-1; j >= 0; j--)
51             r[i][j] = (j == n-1 || a[i][j] > r[i][j+1]) ? a[i][j] : r[i][j+1];
52
53     int ans = 0;
54     for (int i = 0; i < m; i++)
55     {
56         for (int j = i; j < m; j++)
57         {
58             for (int p = 0, q = 0; p < n; p++)
59             {
60                 while (q+1 < n && l[i][p] <= r[j][q+1])
61                     q++;
62                 if (p <= q && l[i][p] <= r[j][q] && ans < j-i+q-p)
63                     ans = j-i+q-p;
64             }
65         }
66     }
67
68     fprintf(g, "%d", ans + 1);
69
70     fclose(f);
71     fclose(g);
72
73     return 0;
74 }

```

Listing 4.2.4: traseu90\_3.cpp

```

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

```

```

41
42     return 0;
43 }

```

Listing 4.2.5: traseu90\_4.cpp

```

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

```

Listing 4.2.6: traseu90\_5.cpp

```

1  //O((n*m*n)
2  ///parcuregerea crescatoare a valorilor,
3  ///eliminarea elementelor vizitate
4  ///si comparare doar cu ultimul element din linie
5  #include <fstream>
6
7  using namespace std;
8
9  ifstream fin("traseu.in");
10 ofstream fout("traseu.out");
11
12 struct pozitie
13 {
14     short l,c;
15 };
16
17 int main()
18 {
19     int n,m;
20     fin>>n>>m;

```

```

21     pozitie f[n*m+5];
22     short pleft[n+5][m+5], pright[n+5][m+5];
23     int a[n+5][m+5], d, dm, i, j, i1, i2, j1, j2;
24
25     for(i=1; i<=n; i++)
26     {
27         pright[i][0]=1;
28         for(j=1; j<=m; j++)
29         {
30             fin>>a[i][j];
31             pleft[i][j]=j-1;
32             pright[i][j]=j+1;
33             f[a[i][j]]=i, j;
34         }
35         pleft[i][m+1]=m;
36     }
37     dm=0;
38     for(i=1; i<=n*m; i++)
39     {
40         i1=f[i].l; j1=f[i].c;
41         pleft[i1][pright[i1][j1]]=pleft[i1][j1];
42         pright[i1][pleft[i1][j1]]=pright[i1][j1];
43         for(i2=i1; i2<=n; i2++)
44         {
45             j2=pleft[i2][m+1];
46             if(j1<=j2) {
47                 dm=max(dm, i2-i1+j2-j1);
48             }
49         }
50     }
51
52     if(dm==0)
53         fout<<0;
54     else
55         fout<<dm+1;
56
57     fout.close();
58     fin.close();
59     return 0;
60 }

```

### 4.2.3 \*Rezolvare detaliată

## 4.3 yinyang

### Problema 3 - yinyang

90 de puncte

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

### Cerințe

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

### Date de intrare

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

### Date de ieșire

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

**Restricții și precizări**

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

**Exemple**

yinyang.in	yinyang.out	Explicații
2 3 1 2 4 3 5 6	0	Matricea dată este matrice yin-yang
2 3 6 6 5 4 6 2	3	Operațiile pot fi următoarele: swap(linia 1 , linia 2), swap(coloana 2, coloana 3), swap(coloana 1, coloana 2). Matricea dată va ajunge la final în forma yin-yang: <div style="display: flex; justify-content: space-around; align-items: center;"> <div style="border: 1px solid black; padding: 2px; margin: 2px;">6 6 5 4 6 2</div> <div style="border: 1px solid black; padding: 2px; margin: 2px;">4 6 2 6 6 5</div> <div style="border: 1px solid black; padding: 2px; margin: 2px;">4 2 6 6 5 6</div> <div style="border: 1px solid black; padding: 2px; margin: 2px;">2 4 6 5 6 6</div> </div>

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

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

**Dimensiune** maximă a sursei: 15 KB

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

**4.3.1 Indicații de rezolvare**

Descriere a unei/unor soluții posibile

Structurile de date utilizate: matrici

Gradul de dificultate: 2

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

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

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

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

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

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

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

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

## 4.3.2 Cod sursă

Listing 4.3.1: yinyang45.cpp

---

```

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

```

---

Listing 4.3.2: yinyang90\_1.cpp

---

```

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

```

---

```

35         if(matrix[i][a] != matrix[i][b])
36             return matrix[i][a] < matrix[i][b];
37     return a < b;
38 }
39
40 void sortMatrix()
41 {
42     for(int i = 1; i <= n; i++)
43         lineIndex[i] = i;
44
45     sort(lineIndex + 1, lineIndex + n + 1, cmpLine);
46
47     for(int i = 1; i <= m; i++)
48         columnIndex[i] = i;
49
50     sort(columnIndex + 1, columnIndex + m + 1, cmpColum);
51
52     for(int i = 1; i <= n; i++)
53         for(int j = 1; j <= m; j++)
54             newMatrix[i][j] = matrix[lineIndex[i]][columnIndex[j]];
55 }
56
57 bool checkSolution()
58 {
59     for(int i = 1; i <= n; i++)
60     {
61         for(int j = 1; j <= m; j++)
62         {
63             if(newMatrix[i][j] < newMatrix[i - 1][j] ||
64                newMatrix[i][j] < newMatrix[i][j - 1])
65                 return false;
66         }
67     }
68     return true;
69 }
70
71 void writeOutput()
72 {
73     if(!checkSolution())
74     {
75         printf("-1\n");
76         return ;
77     }
78     int answer = 0;
79
80     for(int i = 2; i <= n; i++)
81         for(int j = 1; j < i; j++)
82             answer += (lineIndex[i] < lineIndex[j]);
83
84     for(int i = 2; i <= m; i++)
85         for(int j = 1; j < i; j++)
86             answer += (columnIndex[i] < columnIndex[j]);
87
88     printf("%d\n", answer);
89 }
90
91 int main ()
92 {
93     freopen("yinyang.in", "r", stdin);
94     freopen("yinyang.out", "w", stdout);
95
96     readInput();
97     sortMatrix();
98     writeOutput();
99
100     return 0;
101 }

```

Listing 4.3.3: yinyang90.2.cpp

```

1  ///yin-yang
2  ///O(n*m*(n+m))
3  #include<stdio.h>
4
5  int n,m,a[1002][1002],i,j,k,x,nr,c1,c2;

```



```

6
7 int main()
8 {
9     FILE *fin,*fout;
10    fin= fopen("yinyang.in","rt");
11    fout=fopen("yinyang.out","wt");
12
13    fscanf(fin,"%d %d\n",&n,&m);
14    for(i=1;i<=n;i++)
15    {
16        for(j=1;j<=m;j++)
17        {
18            fscanf(fin,"%d",&x);
19            a[i][j]=x;
20        }
21    }
22
23    nr=0;
24    for(i=1;i<=n-1;i++)
25    {
26        for(j=i+1;j<=n;j++)
27        {
28            int r=0;c1=0;c2=0;
29            for(k=1;k<=m;k++)
30            {
31                if(a[i][k]<a[j][k])
32                {
33                    if(r==0)r=-1;
34                    c1++;
35                }
36                if(a[i][k]>a[j][k])
37                {
38                    if(r==0)r=+1;
39                    c2++;
40                }
41            }
42
43            if(c1 && c2)
44            {
45                fprintf(fout,"-1");
46                fclose(fout);
47                fclose(fin);
48                return 0;
49            }
50
51            if(r>0) nr++;
52        }
53    }
54
55    for(i=1;i<=m-1;i++)
56    {
57        for(j=i+1;j<=m;j++)
58        {
59            int r=0;c1=0;c2=0;
60            for(k=1;k<=n;k++)
61            {
62                if(a[k][i]<a[k][j])
63                {
64                    if(r==0)r=-1;
65                    c1++;
66                }
67
68                if(a[k][i]>a[k][j])
69                {
70                    if(r==0)r=+1;
71                    c2++;
72                }
73            }
74
75            if(c1 && c2)
76            {
77                fprintf(fout,"-1");
78                fclose(fout);
79                fclose(fin);
80                return 0;
81            }

```

```
82
83         if(r>0) nr++;
84     }
85 }
86
87     fprintf(fout, "%d", nr);
88     fclose(fout); fclose(fin);
89     return 0;
90 }
```

---

### 4.3.3 \*Rezolvare detaliată

# Capitolul 5

## OJI 2018

### 5.1 castel

#### Problema 1 - castel

100 de puncte

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

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

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

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

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

#### Cerințe

Cunoscând codificarea hărții castelului, să se determine:

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

#### Date de intrare

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

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

**Date de ieșire**

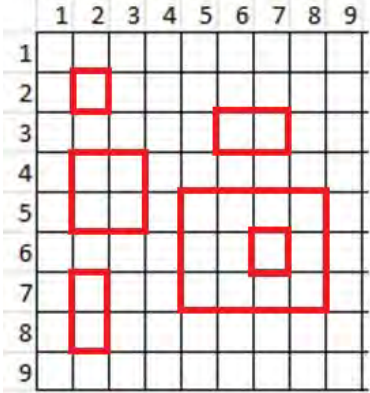
Datele de ieșire se vor scrie în fișierul **castel.out**, astfel:

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

**Restricții și precizări**

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

**Exemple**

castel.in	castel.out	Explicații
<pre> 1 9 0 2 0 0 0 0 0 0 0 4 15 1 0 0 2 2 0 0 0 10 2 0 4 11 14 1 0 4 9 12 1 2 10 10 2 0 4 3 6 5 9 8 10 12 1 0 10 8 4 1 4 15 5 1 4 13 1 4 3 2 10 6 1 4 7 1 0 8 8 8 8 0 0 8 0 0 0 0 0 0 0 </pre>	6	<p>În figură este reprezentată harta castelului</p>  <p>codificat în fișierul de intrare. Acesta conține 6 camere.</p>
<pre> 2 9 0 2 0 0 0 0 0 0 0 4 15 1 0 0 2 2 0 0 0 10 2 0 4 11 14 1 0 4 9 12 1 2 10 10 2 0 4 3 6 5 9 8 10 12 1 0 10 8 4 1 4 15 5 1 4 13 1 4 3 2 10 6 1 4 7 1 0 8 8 8 8 0 0 8 0 0 0 0 0 0 0 </pre>	11	Aria maximă a unei camere este 11.
<pre> 3 9 0 2 0 0 0 0 0 0 0 4 15 1 0 0 2 2 0 0 0 10 2 0 4 11 14 1 0 4 9 12 1 2 10 10 2 0 4 3 6 5 9 8 10 12 1 0 10 8 4 1 4 15 5 1 4 13 1 4 3 2 10 6 1 4 7 1 0 8 8 8 8 0 0 8 0 0 0 0 0 0 0 </pre>	5 5 7 8	Camera cu aria maximă are coordonatele (5,5) - (7,8)

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

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

**Dimensiune** maximă a sursei: **30 KB**

### 5.1.1 Indicații de rezolvare

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

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

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

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

### 5.1.2 Cod sursă

Listing 5.1.1: castel.cpp

```

1  #include <fstream>
2  #include <iostream>
3
4  #define Nmax 102
5  #define Cmax Nmax*Nmax
6  #define InFile "castel.in"
7  #define OutFile "castel.out"
8
9  using namespace std;
10
11 short A[Nmax][Nmax], Viz[Nmax][Nmax];
12 short N;
13 short C;
14
15 short ValBit(short No, short Pos)
16 {
17     return ((1<<Pos) & No)>>Pos;
18 }
19
20 void Fill(short lin, short col, short &Aria, short &elin, short &ecol)
21 {
22     short i, j;
23     short Clin[Cmax], Ccol[Cmax];
24     short CrrLin, CrrCol, Prim, Ultim;
25
26     Aria = 0;
27     elin = lin; ecol = col;
28     Prim = 1; Ultim = 1; Clin[Ultim] = lin; Ccol[Ultim] = col;
29     while(Prim <= Ultim)
30     {
31         CrrLin = Clin[Prim]; CrrCol = Ccol[Prim++];
32         Viz[CrrLin][CrrCol] = 1;
33         Aria++;
34
35         if(elin<=CrrLin && ecol<=CrrCol)
36             elin = CrrLin, ecol = CrrCol;
37
38         if(!ValBit(A[CrrLin][CrrCol], 3) && !Viz[CrrLin-1][CrrCol]) //Nord
39         {
40             Ultim++;
41             Clin[Ultim] = CrrLin-1;
42             Ccol[Ultim] = CrrCol;
43             Viz[CrrLin-1][CrrCol] = 1;
44         }
45
46         if(!ValBit(A[CrrLin][CrrCol], 2) && !Viz[CrrLin][CrrCol+1]) //Est
47         {
48             Ultim++;
49             Clin[Ultim] = CrrLin;
50             Ccol[Ultim] = CrrCol+1;

```

```

51         Viz[CrrLin][CrrCol+1] = 1;
52     }
53
54     if(!ValBit(A[CrrLin][CrrCol], 1) && !Viz[CrrLin+1][CrrCol]) //Sud
55     {
56         Ultim++;
57         Clin[Ultim] = CrrLin+1;
58         Ccol[Ultim] = CrrCol;
59         Viz[CrrLin+1][CrrCol] = 1;
60     }
61
62     if(!ValBit(A[CrrLin][CrrCol], 0) && !Viz[CrrLin][CrrCol-1]) //Vest
63     {
64         Ultim++;
65         Clin[Ultim] = CrrLin;
66         Ccol[Ultim] = CrrCol-1;
67         Viz[CrrLin][CrrCol-1] = 1;
68     }
69 }
70 }
71
72 int main()
73 {
74     int i, j, k;
75     short Slin[Cmax], Scol[Cmax], Svf = 0;
76     short slin, scol, elin, ecol, Aria;
77     short maxil, maxic, maxfl, maxfc, AriaMax=0, NrCamere;
78     ifstream Fin(InFile);
79
80     //citire
81     Fin >> C >> N;
82     for(i=1; i<=N; i++)
83         for(j=1; j<=N; j++)
84         {
85             Fin >> A[i][j];
86             if(A[i][j] == 9 || A[i][j] == 11 ||
87                A[i][j] == 15 || A[i][j] == 13)
88             {
89                 Svf++; Slin[Svf] = i; Scol[Svf] = j;
90             }
91         }
92     Fin.close();
93
94     //initializare
95     for(i=0; i<=N+1; i++)
96     {
97         Viz[0][i] = Viz[N+1][i] = 15;
98         Viz[i][0] = Viz[i][N+1] = 15;
99     }
100
101     for(i=1; i<=N; i++)
102         for(j=1; j<=N; j++)
103             Viz[i][j] = 0;
104
105     //rezolvare
106     NrCamere = Svf;
107     while(Svf)
108     {
109         slin = Slin[Svf]; scol = Scol[Svf];
110         Fill(slin, scol, Aria, elin, ecol);
111         Svf--;
112         if(Aria>=AriaMax)
113             maxil = slin, maxic = scol,
114             maxfl = elin, maxfc = ecol,
115             AriaMax = Aria;
116     }
117
118     ofstream Fou(OutFile);
119     if (C == 1) Fou<<NrCamere<<'\\n';
120     if (C == 2) Fou<<AriaMax<<'\\n';
121     if (C == 3) Fou<<maxil<<' ' <<maxic<<' ' <<maxfl<<' ' <<maxfc<<'\\n';
122     Fou.close();
123     return 0;
124 }

```

### 5.1.3 \*Rezolvare detaliată

## 5.2 eq4

### Problema 2 - eq4

100 de puncte

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

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

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

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

Exemple:

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

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

### Cerințe

Fiind dată o expresie corectă, să se determine:

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

### Date de intrare

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

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

### Date de ieșire

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

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

### Restricții și precizări

- coeficienții sunt numere naturale, având cel mult 4 cifre
- $2 \leq \text{lungimea expresiei} \leq 100000$
- $-500 \leq a \leq b \leq 500$
- $-10^{15} \leq E \leq 10^{15}$
- Testele care au  $C = 1$  totalizează 20 de puncte;
- Testele care au  $C = 2$  totalizează 70 de puncte;
- în cel puțin 30% dintre teste, în expresia dată apar cel mult trei dintre literele  $x, y, z$  sau  $t$ .
- Se acordă 10 puncte din oficiu.

**Exemple**

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

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

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

**Dimensiune** maximă a sursei: **30 KB**

**5.2.1 Indicații de rezolvare**

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

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

Cerința 1:

Afișăm suma dintre cei patru coeficienți și termenul liber.

Cerința 2:

Fie forma canonică

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

pe care am obținut-o prin parsare.

Avem de calculat numărul de soluții întregi ale ecuației

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

sau

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

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

Apoi avem două posibilități:

a) Procedăm similar pentru a calcula toate valorile expresiei

$$c_3z + c_4t + t_1$$

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

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

b) Determinăm toate valorile expresiei

$$c_3z + c_4t + t_1$$

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

Sortăm și  $W$ , apoi printr-un algoritm similar cu cel de interclasare determinăm numărul de valori egale din cei doi vectori, având în vedere că dacă o valoare oarecare apare de  $n_1$  ori în  $V$  și de  $n_2$  ori în  $W$ , la rezultat vom aduna valoarea  $n_1 * n_2$ .

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

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



## 5.2.2 Cod sursă

Listing 5.2.1: eq4.cpp

---

```

1  #include <iostream>
2  #include <fstream>
3  #include <algorithm>
4  #include <string>
5
6  using namespace std;
7
8  int valori[1100000];
9  string expr;
10 int cx,cy,cz,ct,tl,a,b,E;
11 int nrcrt,semn,ceri;
12
13 ifstream f("eq4.in");
14 ofstream g("eq4.out");
15
16 long long rezultat;
17
18 int main()
19 {
20     cout << "Hello world!" << endl;
21     f >> ceri;
22     f >> expr;
23     expr+='+';
24     f >> a >> b >> E;
25     semn = +1;
26     for (int i=0;i<expr.size();i++)
27     {
28         if (expr[i]=='+')
29         {
30             tl = tl + semn*nrcrt;
31             nrcrt = 0;
32             semn = +1;
33         }
34
35         if (expr[i]=='-')
36         {
37             tl = tl + semn*nrcrt;
38             nrcrt = 0;
39             semn = -1;
40         }
41
42         if (expr[i]>='0' && expr[i]<='9')
43             nrcrt = 10*nrcrt+expr[i]-'0';
44
45         if (expr[i]=='x')
46         {
47             if (expr[i-1]=='+' )
48                 nrcrt=1;
49             if (expr[i-1]=='-')
50                 nrcrt=1;
51
52             cx = cx + semn*nrcrt;
53             nrcrt = 0;
54         }
55
56         if (expr[i]=='y')
57         {
58             if (expr[i-1]=='+' )
59                 nrcrt=1;
60             if (expr[i-1]=='-')
61                 nrcrt=1;
62
63             cy = cy + semn*nrcrt;
64             nrcrt = 0;
65         }
66
67         if (expr[i]=='z')
68         {
69             if (expr[i-1]=='+' )
70                 nrcrt=1;

```

```

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

```

### 5.2.3 \*Rezolvare detaliată

## 5.3 turnuri

### Problema 3 - turnuri

100 de puncte

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

cu numărul  $T$ , ca fiind numărul turnurilor din secvența de turnuri care începe cu turnul  $S$ , se termină cu turnul  $D$  și are următoarele proprietăți:

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

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

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

### Cerințe

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

### Date de intrare

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

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

### Date de ieșire

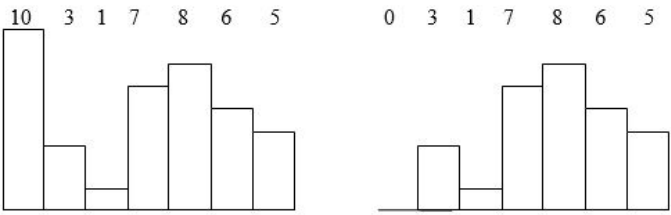
Datele de ieșire se vor scrie în fișierul **turnuri.out**, pe linii separate, astfel: pe linia  $i$  ( $1 \leq i \leq N$ ) se găsește un număr natural reprezentând coeficientul de frumusețe al ansamblului dacă restaurantul s-ar construi în locul turnului  $i$ .

### Restricții și precizări

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

### Exemple

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

7	19	<p>Figura 1 este reprezentarea grafică a fișierului de intrare.</p> <p>Dacă restaurantul se construiește în locul turnului 1 (vezi figura 2), avem următorii coeficienți de frumusețe:</p> <p>Restaurantul are coeficientul 1 (el însuși)</p> <p>Turnul 2 are coeficientul 3 (secvența compusă din turnurile 1,2 și 3)</p> <p>Turnul 3 are coeficientul 1 (el însuși)</p> <p>Turnul 4 are coeficientul 4 (secvența compusă din turnurile 1,2, 3 și 4)</p> <p>Turnul 5 are coeficientul 7 (secvența compusă din toate turnurile)</p> <p>Turnul 6 are coeficientul 2 (secvența compusă din turnurile 6 și 7)</p> <p>Turnul 7 are coeficientul 1 (el însuși)</p>
10 3 1 7 8 6 5	22	
	22	
	22	
	21	
	22	
	22	
		<p>Coeficientul de frumusețe al ansamblului este:</p> $1 + 3 + 1 + 4 + 7 + 2 + 1 = 19$  <p><b>Figura 1</b>                      <b>Figura 2</b></p>

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

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

**Dimensiune** maximă a sursei: 15KB

### 5.3.1 Indicații de rezolvare

Adrian BUDĂU, student la Universitatea București

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Ea va obține tot 60 de puncte.

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

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

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

### 5.3.2 Cod sursă

Listing 5.3.1: turnuri.cpp

---

```

1  #include <iostream>
2  #include <fstream>
3  #include <vector>
4  #include <algorithm>
5  #include <stack>
6  #include <cassert>
7
8  using namespace std;
9
10 bool is_unique(vector<int> V)
11 {
12     sort(V.begin(), V.end());
13     for (int i = 1; i < int(V.size()); ++i)
14         if (V[i] == V[i - 1])
15             return false;
16     return true;
17 }
18
19 int main()
20 {
21     ifstream cin("turnuri.in");
22     ofstream cout("turnuri.out");
23
24     int N; assert(cin >> N);
25     assert(1 <= N && N <= 100 * 1000);
26     vector<int> V(N);
27
28     for (int i = 0; i < N; ++i)
29     {
30         assert(cin >> V[i]);
31         assert(1 <= V[i] && V[i] <= 1000 * 1000 * 1000);
32     }
33
34     assert(is_unique(V));
35
36     vector<int> left1(N, -1), left2(N, -1), right1(N, N), right2(N, N);
37

```

```

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

```

### 5.3.3 \*Rezolvare detaliată

# Capitolul 6

## OJI 2017

### 6.1 caps

#### Problema 1 - caps

100 de puncte

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

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

Inițial, Miruna construiește un șir  $S$  de  $K$  litere. Apoi, ea construiește un nou șir obținut prin aplicarea operației NEXT asupra șirului  $S$ . Miruna dorește să obțină succesiv șiruri de litere din ce în ce mai lungi aplicând operația NEXT asupra șirului construit în etapa precedentă.

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

$\text{"HamhAMhAMHam"}$ ,

$\text{"HamhAMhAMHamhAMhAMHamHamhAMhAMHamHamhAMhAMhAMHam"}$

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

#### Cerințe

Miruna vă roagă să răspundeți la  $Q$  întrebări de tipul:

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

#### Date de intrare

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

#### Date de ieșire

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

#### Restricții și precizări

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

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

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

### Exemple

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

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

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

**Dimensiune** maximă a sursei: **10 KB**

### 6.1.1 Indicații de rezolvare

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

Problema construiește un șir după regula precizată.

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

Pentru a calcula numărul de apariții al acestei cifre se observă că pe fiecare secvență de lungime dublă a șirului inițial avem mereu șirul inițial și șirul CAPS. Se precalculează numărul de apariții al fiecărei litere în șirul inițial și în șirul CAPS.

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

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

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

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

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

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

3) Pentru celelalte 20 de puncte ( $K \leq 3000$  și  $Q \leq 1000$ ).

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

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

De exemplu dacă  $N = 39$  și  $K = 7$ .

Găsim  $X = 7 * 4^2$ . Obținem cele 4 intervale  $[1, 28]$   $[29, 56]$   $[57, 84]$   $[85, 112]$ . 39 se încadrează în  $[29, 56]$ , se ține minte că trebuie inversat răspunsul și se continuă recursiv cu  $N = 39 - 29 + 1 = 10$  și  $INVERSIUNI = 1$



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

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

Dacă se și precalculează de câte ori apare fiecare caracter în fiecare string de lungime  $K * 4^P$  până când  $K * 4^P$  depășește  $10^{18}$  se poate de fiecare dată când se coboară în recursivitate să se adune la răspuns de câte ori a apărut intervalele de dinainte (în exemplu dacă  $N = 57$ , atunci ar fi trebuit să se adauge la răspuns de câte ori apare caracterul găsit în șirul de mărime  $7 * 4$ , adică intervalul  $[1, 28]$  și de câte ori apare inversul lui într-un șir de mărime  $7 * 4$ , adică intervalul  $[29, 56]$ ).

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

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

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

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

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

Iar pentru a număra de câte ori apare litera răspuns în toate concatenările  $1, 2, \dots, PART - 1$  se poate face următoarea observație:

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

Deci la răspuns s-ar adăuga

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

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

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

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

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

### 6.1.2 Cod sursă

Listing 6.1.1: adrian-100.cpp

```
1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <algorithm>
5 #include <cassert>
6
7 using namespace std;
```

```

8
9 int bits(int64_t N)
10 {
11     int have = 0;
12     while (N > 0)
13     {
14         ++have;
15         N -= (N & -N);
16     }
17     return have;
18 }
19
20 int map(char c)
21 {
22     if (c >= 'A' && c <= 'Z')
23         return c - 'A';
24     return c - 'a' + 26;
25 }
26
27 char rev(char c) {
28     return c ^ 'A' ^ 'a';
29 }
30
31 int main ()
32 {
33     ifstream cin("caps.in");
34     ofstream cout("caps.out");
35
36     int K, Q; assert(cin >> K >> Q);
37     assert(1 <= K && K <= 100 * 1000);
38     assert(1 <= Q && Q <= 50 * 1000);
39
40     string S; cin >> S;
41     assert(int(S.size()) == K);
42     for (auto &c : S)
43         assert((c >= 'a' && c <= 'z') || (c >= 'A' || c <= 'Z'));
44
45     int total_count[52];
46     fill(total_count, total_count + 52, 0);
47     for (auto &c : S)
48         total_count[map(c)]++;
49
50     vector<int> positions[52];
51     for (int i = 0; i < int(S.size()); ++i)
52         positions[map(S[i])].push_back(i);
53
54     for (int i = 0; i < Q; ++i)
55     {
56         int64_t N;
57         assert(cin >> N);
58         assert(1 <= N && N <= 1000LL * 1000 * 1000 * 1000 * 1000 * 1000);
59         --N;
60
61         int64_t part = N / K;
62         int index = N % K;
63         char answer;
64         if (bits(part) % 2)
65             answer = rev(S[index]);
66         else
67             answer = S[index];
68
69         int64_t many = 0;
70         int64_t normal = part / 2, reversed = part / 2;
71         if (part % 2)
72         {
73             if (bits(part - 1) % 2)
74                 ++reversed;
75             else
76                 ++normal;
77         }
78
79         many += normal * total_count[map(answer)];
80         many += reversed * total_count[map(rev(answer))];
81         char search_for = S[index];
82
83         many += upper_bound(positions[map(search_for)].begin(),

```

```

84         positions[map(search_for)].end(), index) -
85         positions[map(search_for)].begin());
86     cout << answer << " " << many << "\n";
87 }
88 }

```

## Listing 6.1.2: manolache-100.cpp

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  ifstream in("caps.in");
6  ofstream out("caps.out");
7
8  char s1[100010], s2[100010];
9  unsigned int f1a[27][100010], f1A[27][100010], f2a[27][100010], f2A[27][100010];
10 long long l1, q, n, cat, r, dif, w;
11
12 int main()
13 {
14     in>>l1>>q; in.get();
15     //in.get(s1, 101);
16     in>>s1;
17     for(int i=0; i<l1; ++i)
18         if(s1[i]>='a')
19             {
20                 s2[i]=s1[i]-32;
21                 f1a[s1[i]-'a'][i]=1;
22                 f2a[s2[i]-'A'][i]=1;
23             }
24         else
25             {
26                 s2[i]=s1[i]+32;
27                 f1A[s1[i]-'A'][i]=1;
28                 f2a[s2[i]-'a'][i]=1;
29             }
30
31     for(int i=0; i<26; ++i)
32         for(int j=1; j<l1; ++j)
33             {
34                 f1a[i][j]+=f1a[i][j-1];
35                 f1A[i][j]+=f1A[i][j-1];
36                 f2a[i][j]+=f2a[i][j-1];
37                 f2A[i][j]+=f2A[i][j-1];
38             }
39
40     long long d=2l1*l1;
41     char c;
42     for(; q; --q)
43     {
44         in>>n;
45         cat=n/d; //nr bucati duble complete
46         r=n%d; //rest de cautat
47         long long nr=n/l1; //nr bucati mici
48         if(n%l1) nr++; //nr de bucati simple;
49         w= __builtin_parityll(nr-l1l); //numerotare de la 0
50         if(!w) //incepe cu s1
51             if(r==0l1) //se termina la sf s1
52                 {
53                     c=s1[l1-1];
54                     dif=0l1;
55                 }
56             else
57                 if(r<=l1) //se termina pe bucata s1
58                     {
59                         c=s1[r-1];
60                         if(c>='a')
61                             dif=f1a[c-'a'][r-1];
62                         else
63                             dif=f1A[c-'A'][r-1];
64                     }
65                 else // de termina pe s1 dar nu e luata si precedenta,
66                     // care e evident s2
67                 {

```

```

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

```

Listing 6.1.3: manolache-v1\_100.cpp

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  //ifstream in("caps.in");
6  ofstream out("caps.out");
7  char s1[100010],s2[100010],cc;
8  int f1a[27][100010],f1A[27][100010],f2a[27][100010],f2A[27][100010];
9  long long l1,q,n,cat,r,dif,w;
10
11 int main()
12 {
13     freopen("caps.in","r",stdin);
14     scanf("%lld %lld",&l1,&q);
15     //cout<<l1<<' ' <<q;
16     //in>>l1>>q;in.get();
17     //in.get(s1,101);
18     scanf("%c%s",&cc,s1);
19     //cout<<s1<<' ';
20     //in>>s1;
21     for(int i=0;i<l1;++i)
22         if(s1[i]>='a')
23             {s2[i]=s1[i]-32;f1a[s1[i]-'a'][i]=1;f2A[s2[i]-'A'][i]=1;}
24         else
25             {s2[i]=s1[i]+32;f1A[s1[i]-'A'][i]=1;f2a[s2[i]-'a'][i]=1;}
26     for(int i=0;i<26;++i)
27         for(int j=1;j<l1;++j)
28         {
29             f1a[i][j]+=f1a[i][j-1];
30             f1A[i][j]+=f1A[i][j-1];
31             f2a[i][j]+=f2a[i][j-1];
32             f2A[i][j]+=f2A[i][j-1];

```

```

33     }
34     long long d=2ll*ll;
35     char c;
36     for(;q;--q)
37     {
38         //in>>n;
39         scanf("%lld",&n);
40         cat=n/d;//nr bucati duble complete
41         r=n%d;//rest de cautat
42         long long nr=n/ll;//nr bucati mici
43         if(n%ll) nr++;//nr de bucati simple;
44         w= __builtin_parityll(nr-1ll); //numerotare de la 0
45         if(!w)//incepe cu s1
46             if(r==0ll) //se termina la sf s1
47                 {c=s1[l1-1];
48                   dif=0ll;
49                 }
50             else
51                 if(r<=l1) //se termina pe bucata s1
52                     {
53                         c=s1[r-1];
54                         if(c>='a')
55                             dif=f1a[c-'a'][r-1];
56                         else
57                             dif=f1A[c-'A'][r-1];
58                     }
59                 else // de termina pe s1 dar nu e luata si precedenta,
60                     // care e evident s2
61                     {
62                         c=s1[r-l1-1];
63                         if(c>='a')
64                             dif=f2a[c-'a'][l1-1]+f1a[c-'a'][r-l1-1];
65                         else
66                             dif=f2A[c-'A'][l1-1]+f1A[c-'A'][r-l1-1];
67                     }
68             else //incepe cu s2
69                 if(r==0ll) //se termina cu s2
70                     {
71                         c=s2[l1-1];
72                         dif=0ll;
73                     }
74                 else
75                     if(r<=l1) //se termina pe bucata s2
76                         {
77                             c=s2[r-1];
78                             if(c>='a')
79                                 dif=f2a[c-'a'][r-1];
80                             else
81                                 dif=f2A[c-'A'][r-1];
82                         }
83                     else //se termina pe s2 dar e luata si precedenta care e s1
84                         {
85                             c=s2[r-l1-1];
86                             if(c>='a')
87                                 dif=f1a[c-'a'][l1-1]+f2a[c-'a'][r-l1-1];
88                             else
89                                 dif=f1A[c-'A'][l1-1]+f2A[c-'A'][r-l1-1];
90                         }
91
92         unsigned long long sol=0ll;
93         if(c>='a')
94             sol=dif+ll*cat*(f1a[c-'a'][l1-1]+f2a[c-'a'][l1-1]);
95         else
96             sol=dif+ll*cat*(f1A[c-'A'][l1-1]+f2A[c-'A'][l1-1]);
97         out<<c<<' ';<<sol<<'\n';
98     }
99     out.close();
100     return 0;
101 }

```

Listing 6.1.4: manolache-v2\_100.cpp

```

1 #include <bits/stdc++.h>
2
3 using namespace std;

```



```

80         else
81             if(r<=l1) //se termina pe bucata s2
82             {
83                 c=s2[r-1];
84                 if(c>='a')
85                     dif=f2a[c-'a'][r-1];
86                 else
87                     dif=f2A[c-'A'][r-1];
88             }
89             else // se termina pe s2 dar e luata si precedenta
90                 // care e s1
91             {
92                 c=s2[r-l1-1];
93                 if(c>='a')
94                     dif=f1a[c-'a'][l1-1]+f2a[c-'a'][r-l1-1];
95                 else
96                     dif=f1A[c-'A'][l1-1]+f2A[c-'A'][r-l1-1];
97             }
98
99             unsigned long long sol=0ll;
100             if(c>='a')
101                 sol=dif+l1l*cat*(f1a[c-'a'][l1-1]+f2a[c-'a'][l1-1]);
102             else
103                 sol=dif+l1l*cat*(f1A[c-'A'][l1-1]+f2A[c-'A'][l1-1]);
104             out<<c<<' ' <<sol<<'\n';
105         }
106
107         out.close();
108         return 0;
109     }

```

Listing 6.1.5: MLT-100.cpp

```

1 // prof. Mircea Lupse-Turpan - Liceul Teoretic Grigore Moisil Timisoara
2 #include <fstream>
3 using namespace std;
4
5 ifstream fin("caps.in");
6 ofstream fout("caps.out");
7
8 const int KMax = 100005;
9 char X[KMax], Y[KMax];
10 int Frecv1[60][KMax], Frecv2[60][KMax];
11 int K, Q;
12 long long N, Nr1, Nr2, Sol;
13
14 void Read()
15 {
16     fin >> K >> Q;
17
18     for(int i = 1; i <= K; ++i)
19         fin >> X[i];
20
21     for(int i = 1; i <= K; ++i)
22         if(X[i] >= 'a' && X[i] <= 'z')
23             Y[i] = X[i] - 32;
24         else
25             Y[i] = X[i] + 32;
26
27     for(int i = 1; i <= K; ++i)
28     {
29         for(int j = 0; j < 60; ++j)
30         {
31             Frecv1[j][i] = Frecv1[j][i-1];
32             Frecv2[j][i] = Frecv2[j][i-1];
33         }
34         Frecv1[X[i] - 'A'][i]++;
35         Frecv2[Y[i] - 'A'][i]++;
36     }
37 }
38
39 char DEI(long long N, long long Nr, int Switch)
40 {
41     if(N <= K)
42     {

```

```

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

```



```

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

```

### 6.1.3 \*Rezolvare detaliată

## 6.2 rover

### Problema 2 - rover

**100 de puncte**

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

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

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

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

### Cerințe

1. Determinați numărul minim posibil de zone periculoase pe care le traversează Roverul pentru a ajunge din zona  $(1,1)$  în zona  $(N,N)$ .
2. Determinați greutatea maximă pe care o poate avea un Rover care să ajungă din zona  $(1,1)$  în zona  $(N,N)$ , fără a traversa nicio zonă periculoasă pentru el.

### Date de intrare

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

### Date de ieșire

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

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

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

### Restricții și precizări

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

### Exemple

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

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

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

**Dimensiune** maximă a sursei: **10 KB**

### 6.2.1 Indicații de rezolvare

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

#### Cerința 1

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

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

#### Cerința 2

*Căutarea binară* a soluției + *Algoritmul Lee*

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

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

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

### 6.2.2 Cod sursă

Listing 6.2.1: adrian-100.cpp

```

1 // #include <iostream>
2 // #include <fstream>
3 // #include <vector>
4 // #include <algorithm>
5 // #include <queue>
6 // #include <cassert>
7
8 #include <bits/stdc++.h>

```

```

9
10 using namespace std;
11
12 static const int dx[4] = {-1, 0, 1, 0};
13 static const int dy[4] = {0, -1, 0, 1};
14
15 int minimum(const vector< vector<int> > &A, int G)
16 {
17     int N = A.size();
18     vector< vector<int> > dist(N, vector<int>(N, numeric_limits<int>::max()/2));
19
20     queue< pair<int, int> > current, next;
21     current.emplace(0, 0);
22     dist[0][0] = int(A[0][0] < G);
23
24     int current_cost = dist[0][0];
25     while (current.size() || next.size())
26     {
27         if (current.empty())
28         {
29             current.swap(next);
30             ++current_cost;
31             continue;
32         }
33
34         int x, y; tie(x, y) = current.front();
35         current.pop();
36         if (dist[x][y] != current_cost)
37             continue;
38
39         for (int k = 0; k < 4; ++k)
40         {
41             int newx = x + dx[k];
42             int newy = y + dy[k];
43             if (newx >= 0 && newy >= 0 && newx < N && newy < N)
44                 if (dist[newx][newy] > dist[x][y] + int(A[newx][newy] < G))
45                 {
46                     dist[newx][newy] = dist[x][y] + int(A[newx][newy] < G);
47                     if (A[newx][newy] < G)
48                         next.emplace(newx, newy);
49                     else
50                         current.emplace(newx, newy);
51                 }
52         }
53     }
54     return dist[N - 1][N - 1];
55 }
56
57 int main()
58 {
59     ifstream cin("rover.in");
60     ofstream cout("rover.out");
61
62     int V; assert(cin >> V);
63     assert(1 <= V && V <= 2);
64     int N, G; assert(cin >> N);
65     assert(1 <= N && N <= 1000);
66     if (V == 1)
67     {
68         assert(cin >> G);
69         assert(1 <= G && G <= 5000);
70     }
71
72     vector< vector<int> > A(N, vector<int>(N));
73     for (int i = 0; i < N; ++i)
74         for (int j = 0; j < N; ++j)
75         {
76             assert(cin >> A[i][j]);
77             assert(0 <= A[i][j] && A[i][j] <= 10000);
78         }
79
80     if (V == 1)
81     {
82         assert(A[0][0] >= G && A[N - 1][N - 1] >= G);
83         cout << minimum(A, G) << "\n";
84         return 0;

```

```

85     }
86
87     int answer;
88     int max_value = 0;
89     for (auto &line : A)
90         for (auto &v : line)
91             max_value = max(max_value, v);
92     int step;
93     for (step = 1; step < max_value; step <= 1);
94     for (answer = 0; step; step >= 1)
95         if (minimum(A, answer + step) == 0)
96             answer += step;
97     cout << answer << "\n";
98 }

```

Listing 6.2.2: GM\_rover.cpp

```

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

```

---

```

59     {
60         int X = Q.front().p1, Y=Q.front().p2;
61         Q.pop_front();
62         for(int k=0;k<4;k++)
63         {
64             int x1=X+dx[k], y1=Y+dy[k];
65             if((x1>=1) && (x1<=n) && (y1>=-1) && (y1<= n) && b[x1][y1]==-1)
66             {
67                 if(a[x1][y1]>= w)
68                 {
69                     b[x1][y1]=b[X][Y]+1;
70                     aux.p1=x1,aux.p2=y1;
71                     Q.push_back(aux);
72                 }
73             }
74         }
75     }
76     return (b[n][n] != -1);
77 }
78
79 void p2x()
80 {
81     int sol;
82     int mx=0;
83     for (auto &line : a)
84         for (auto &v : line)
85             mx=max(mx, v);
86     int step;
87     for (step=1; step < mx; step<=&=1);
88     for (sol =0; step; step >>= 1)
89         if (Lee(sol+step))
90             sol+=step;
91     g<<sol<< "\n";
92 }
93
94 int main()
95 {
96     f>>t>>n; if(t==1) f>>G;
97     for(int i=1;i<=n; ++i)
98         for(int j=1; j<=n; ++j)
99             f>>a[i][j];
100     if(t==1)
101         p1();
102     else
103         p2x();
104     g.close();
105     return 0;
106 }

```

---

Listing 6.2.3: MLT\_Rover.cpp

---

```

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

```

```

25 void Add_Front(int V1, int V2)
26 {
27     Cell * p = new Cell;
28     p -> x = V1;
29     p -> y = V2;
30     p -> next = First;
31     if(!First)
32         Last = p;
33     First = p;
34 }
35
36 void Add_Back(int V1, int V2)
37 {
38     Cell * p = new Cell;
39     p -> x = V1;
40     p -> y = V2;
41     p -> next = NULL;
42     if(!First)
43         First = p;
44     else
45         Last -> next = p;
46     Last = p;
47 }
48
49 void Delete_Front()
50 {
51     Cell * p;
52     p = First;
53     First = First -> next;
54     delete p;
55     if(!First) Last = NULL;
56 }
57
58 bool isEmpty()
59 {
60     return (First == NULL);
61 }
62
63 void Read()
64 {
65     fin >> V >> N;
66     if(V == 1) fin >> G;
67     for(int i = 1; i <= N; ++i)
68         for(int j = 1; j <= N; ++j)
69             fin >> A[i][j];
70 }
71
72 inline bool Inside(int x, int y)
73 {
74     return ( (x >= 1) && (x <= N) && (y >= 1) && (y <= N) );
75 }
76
77 void Solve1()
78 {
79     memset(DP, -1, sizeof(DP));
80     Add_Back(1, 1);
81     DP[1][1] = 0;
82
83     while(!isEmpty())
84     {
85         int X = First -> x, Y = First -> y;
86         Delete_Front();
87         for(int k = 0; k < 4; k++)
88         {
89             int NewX = X + dx[k], NewY = Y + dy[k];
90             if(Inside(NewX, NewY) && DP[NewX][NewY] == -1)
91             {
92                 if(A[NewX][NewY] < G)
93                 {
94                     DP[NewX][NewY] = DP[X][Y] + 1;
95                     Add_Back(NewX, NewY);
96                 }
97                 else
98                 {
99                     DP[NewX][NewY] = DP[X][Y];
100                     Add_Front(NewX, NewY);

```

```

101         }
102     }
103 }
104 }
105     fout << DP[N][N] << "\n";
106 }
107
108 bool Lee(int Value)
109 {
110     memset(DP, -1, sizeof(DP));
111     Add_Back(1, 1);
112     DP[1][1] = 0;
113     while(!isEmpty())
114     {
115         int X = First -> x, Y = First -> y;
116         Delete_Front();
117         for(int k = 0; k < 4; k++)
118         {
119             int NewX = X + dx[k], NewY = Y + dy[k];
120             if(Inside(NewX, NewY) && DP[NewX][NewY] == -1)
121             {
122                 if(A[NewX][NewY] >= Value)
123                 {
124                     DP[NewX][NewY] = DP[X][Y] + 1;
125                     Add_Back(NewX, NewY);
126                 }
127             }
128         }
129     }
130     return (DP[N][N] != -1);
131 }
132
133 void Solve2()
134 {
135     int Left = 1, Right = oo, Sol = -1;
136
137     while(Left <= Right)
138     {
139         int Mid = (Left + Right) / 2;
140         if(Lee(Mid))
141         {
142             Sol = Mid;
143             Left = Mid + 1;
144         }
145         else
146             Right = Mid - 1;
147     }
148     fout << Sol << "\n";
149 }
150
151 int main()
152 {
153     Read();
154     if(V == 1)
155         Solve1();
156     else
157         Solve2();
158     return 0;
159 }

```

Listing 6.2.4: MLT\_Rover\_STL.cpp

```

1 // prof. Mircea Lupse-Turpan - Liceul Grigore Moisil Timisoara - 100 puncte
2 #include <fstream>
3 #include <deque>
4 #include <cstring>
5
6 using namespace std;
7
8 ifstream fin("rover.in");
9 ofstream fout("rover.out");
10
11 const int NMax = 505;
12 const int oo = 10000;
13

```

```

14 int A[NMax][NMax], DP[NMax][NMax];
15 int V, N, G;
16 int dx[] = {-1, 0, 1, 0}, dy[] = {0, 1, 0, -1};
17
18 deque <pair <int, int> > Q;
19
20 void Read()
21 {
22     fin >> V >> N;
23     if(V == 1) fin >> G;
24     for(int i = 1; i <= N; ++i)
25         for(int j = 1; j <= N; ++j)
26             fin >> A[i][j];
27 }
28
29 inline bool Inside(int x, int y)
30 {
31     return ( (x >= 1) && (x <= N) && (y >= 1) && (y <= N) );
32 }
33
34 void Solve1()
35 {
36     memset(DP, -1, sizeof(DP));
37     Q.push_back(make_pair(1, 1));
38     DP[1][1] = 0;
39
40     while(!Q.empty())
41     {
42         int X = Q.front().first, Y = Q.front().second;
43         Q.pop_front();
44         for(int k = 0; k < 4; k++)
45         {
46             int NewX = X + dx[k], NewY = Y + dy[k];
47             if(Inside(NewX, NewY) && DP[NewX][NewY] == -1)
48             {
49                 if(A[NewX][NewY] < G)
50                 {
51                     DP[NewX][NewY] = DP[X][Y] + 1;
52                     Q.push_back(make_pair(NewX, NewY));
53                 }
54                 else
55                 {
56                     DP[NewX][NewY] = DP[X][Y];
57                     Q.push_front(make_pair(NewX, NewY));
58                 }
59             }
60         }
61     }
62     fout << DP[N][N] << "\n";
63 }
64
65 bool Lee(int Value)
66 {
67     memset(DP, -1, sizeof(DP));
68     Q.push_back(make_pair(1, 1));
69     DP[1][1] = 0;
70     while(!Q.empty())
71     {
72         int X = Q.front().first, Y = Q.front().second;
73         Q.pop_front();
74         for(int k = 0; k < 4; k++)
75         {
76             int NewX = X + dx[k], NewY = Y + dy[k];
77             if(Inside(NewX, NewY) && DP[NewX][NewY] == -1)
78             {
79                 if(A[NewX][NewY] >= Value)
80                 {
81                     DP[NewX][NewY] = DP[X][Y] + 1;
82                     Q.push_back(make_pair(NewX, NewY));
83                 }
84             }
85         }
86     }
87     return (DP[N][N] != -1);
88 }
89

```



---

```

90 void Solve2()
91 {
92     int Left = 1, Right = oo, Sol = -1;
93
94     while(Left <= Right)
95     {
96         int Mid = (Left + Right) / 2;
97         if(Lee(Mid))
98         {
99             Sol = Mid;
100             Left = Mid + 1;
101         }
102         else
103             Right = Mid - 1;
104     }
105     fout << Sol << "\n";
106 }
107
108 int main()
109 {
110     Read();
111     if(V == 1)
112         Solve1();
113     else
114         Solve2();
115     return 0;
116 }

```

---

### 6.2.3 \*Rezolvare detaliată

## 6.3 sir

### Problema 3 - sir

100 de puncte

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

### Cerințe

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

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

### Date de intrare

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

### Date de ieșire

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

- dacă  $P$  a avut valoarea 1, valoarea reprezintă numărul de șiruri distincte care au exact  $N$  numere și se termină cu numărul  $X$ ;

• dacă  $P$  a avut valoare 2, valoarea reprezintă numărul de șiruri distincte care au exact  $N$  numere și fiecare număr se repetă de cel mult  $X$  ori.

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

### Restricții și precizări

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

### Exemple

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

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

**Memorie:** total **16 MB**

**Dimensiune** maximă a sursei: **5 KB**

### 6.3.1 Indicații de rezolvare

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

Pentru valorile citite se calculeaza (modulo 20173333)

Cerința 1: Combinări( $N-1, X-1$ );

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

Cerința 2:

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

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

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

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

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

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

Soluția este dată de  $NSOL(N+1)$ .

### 6.3.2 Cod sursă

Listing 6.3.1: adrian-100.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <cassert>
5
6 using namespace std;
7
8 static const int kModulo = 20173333;
```

```

9
10 int main()
11 {
12     ifstream cin("sir.in");
13     ofstream cout("sir.out");
14
15     int T, N, X; assert(cin >> T >> N >> X);
16     assert(1 <= T && T <= 2);
17     assert(1 <= N && N <= 100 * 1000);
18     assert(1 <= X && X <= N);
19     if (T == 1)
20     {
21         vector<int> prime(N + 1, 0);
22         for (int i = 2; i <= N; ++i)
23             if (prime[i] == 0)
24                 for (int j = i; j <= N; j += i)
25                     prime[j] = i;
26
27         vector<int> exponent(N + 1, 0);
28         for (int i = N - X + 1; i <= N - 1; ++i)
29             for (int j = i; j > 1; j /= prime[j])
30                 ++exponent[prime[j]];
31         for (int i = 1; i < X; ++i)
32             for (int j = i; j > 1; j /= prime[j])
33                 --exponent[prime[j]];
34
35         int answer = 1;
36         for (int i = 1; i <= N; ++i)
37             for (int j = 0; j < exponent[i]; ++j)
38                 answer = (1LL * answer * i) % kModulo;
39         cout << answer << "\n";
40         return 0;
41     }
42
43     vector<int> fibK(N + 2, 0);
44     fibK[1] = 1;
45     fibK[2] = 1;
46     for (int i = 3; i <= N + 1; ++i)
47     {
48         fibK[i] = (2 * fibK[i - 1] - fibK[max(i - X - 1, 0)]) % kModulo;
49         if (fibK[i] < 0)
50             fibK[i] += kModulo;
51     }
52     cout << fibK[N + 1] << "\n";
53 }

```

Listing 6.3.2: GM1.cpp

```

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

```

---

```

28 {
29     int sol=1;
30     for(int i=2;i<=p;++i)
31         sol=(1ll*sol*i)%MOD;
32     return sol;
33 }
34
35 int comb(int n, int k)
36 {
37     int w=(1ll*fact(n)*InversModular(fact(k)))%MOD;
38     w=(1ll*w*InversModular(fact(n-k)))%MOD;
39     return w;
40 }
41
42 int p1()
43 {
44     return comb(N-1,X-1);
45 }
46
47 int p2()
48 {
49     vector<int> a;
50     for(int i=0;i<=N;++i) a.push_back(0);
51     int S=1;
52     a[0]=1;
53     for(int i=1; i<=N; ++i)
54     {
55         a[i]=S;
56         if(i>=X)
57             S=((S+a[i]-a[i-X]+MOD))%MOD;
58         else
59             S=(S+a[i])%MOD;
60     }
61     return a[N];
62 }
63
64 int main()
65 {
66     f>>P>>N>>X;
67     if(P==1)
68         g<<p1()<<"\n";
69     else
70         g<<p2()<<"\n";
71     return 0;
72 }

```

---

Listing 6.3.3: GM2.cpp

---

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  ifstream f("sir.in");
6  ofstream g("sir.out");
7
8  int P,N,X;
9  const int MOD = 20173333;
10 int a[50005];
11
12 void desc(int x)
13 {
14     int e=0,d=2;
15     while(d*d<=x)
16     {
17         e=0;
18         while(x%d==0) {x/=d;e++;}
19         a[d]+=e;
20         d++;
21     }
22     a[x]++;
23 }
24
25 void desc1(int x)
26 {
27     int e=0,d=2;
28     while(d*d<=x)
29     {
30         e=0;
31         while(x%d==0) {x/=d;e++;}
32     }
33     a[x]++;
34 }

```

---

```

28         a[d] -= e;
29         d++;
30     }
31     a[x]--;
32 }
33
34 int comb1(int n, int m)
35 {
36     int i;
37     for (i = n; i >= n - m + 1; i--)
38         desc(i);
39     for (i = 2; i <= m; i++)
40         desc1(i);
41     int s = 1;
42     for (i = 2; i <= 50002; i++)
43         for (int j = 1; j <= a[i]; j++) s = (1ll * s * i) % MOD;
44     return s;
45 }
46
47 int p1()
48 {
49     return comb1(N - 1, X - 1);
50 }
51
52 int p2()
53 {
54     vector<int> a;
55     for (int i = 0; i <= N; ++i) a.push_back(0);
56     int S = 1;
57     a[0] = 1;
58     for (int i = 1; i <= N; ++i)
59     {
60         a[i] = S;
61         if (i >= X)
62             S = (S + a[i] - a[i - X] + MOD) % MOD;
63         else
64             S = (S + a[i]) % MOD;
65     }
66     return a[N];
67 }
68
69 int main()
70 {
71     f >> P >> N >> X;
72     if (P == 1)
73         g << p1() << "\n";
74     else
75         g << p2() << "\n";
76     return 0;
77 }

```

Listing 6.3.4: sir\_100.cpp

```

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

```

```

23     }
24     return v[u];
25 }
26
27 long long rec(int n, int r)
28 {
29     int i;
30     v[1]=1;v[2]=1;
31     for(i=2;i<=n;i++)
32     {
33         v[i+1]=2*v[i]%MOD;
34         if(i>r)
35         {
36             if(v[i+1]>=v[i-r])
37                 v[i+1]-=v[i-r];
38             else
39                 v[i+1]=v[i+1]+MOD-v[i-r];
40         }
41     }
42     return v[n+1];
43 }
44
45 int main()
46 {
47     fin>>p>>n>>x;
48     if(p==1)
49         fout<<comb(n,x)<<endl;
50     else
51         fout<<rec(n,x)<<endl;
52     return 0;
53 }

```

Listing 6.3.5: sirEm.cpp

```

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

```

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

---

### 6.3.3 \*Rezolvare detaliată

# Capitolul 7

## OJI 2016

### 7.1 interesant

#### Problema 1 - interesant

100 de puncte

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

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

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

#### Cerințe

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

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

#### Date de intrare

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

#### Date de ieșire

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

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

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

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

#### Restricții și precizări

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



**Exemple**

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

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

**Memorie:** total **8 MB**

**Dimensiune** maximă a sursei: **10 KB**

**7.1.1 Indicații de rezolvare**

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

Soluția propusă analizează șirurile pe parcursul citirii din fișier.

Cerința 1 - 20 puncte

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

Cerința 2 - 80 puncte

Rezolvarea cerinței presupune:

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

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

**7.1.2 Cod sursă**

Listing 7.1.1: interesant\_en1.cpp

```

1  /*
2     prof. Eugen Nodea
3     Colegiul National "Tudor Vladimirescu", Tg-Jiu
4  */
5  #include <fstream>
6  #include <cstring>
7
8  using namespace std;
9
10 ifstream f("interesant.in");
11 ofstream g("interesant.out");
12
13 const int Dim = 5005;
14 typedef char CUV[Dim];
15
16 char s[Dim], sol[Dim];
17 int N, Max, nr, Ns;
18 char cuv[201][Dim];
19 bool Erase[201];
20 int Len[201];
21
22 int main()
23 {
24     int l, i, j, ls, lc, P, L;
25     char *p;
```

```

26     bool ok;
27
28     f >> P >> Ns; f.get();
29     if ( P == 1 )
30     {
31         f >> s;
32         while( !f.eof() )
33         {
34             l = strlen(s);
35             if (l > Max)
36             {
37                 strcpy(sol, s);
38                 Max = l;
39             }
40             else
41                 if (l == Max && strcmp(sol, s) > 0 )
42                     strcpy(sol, s);
43             f >> s;
44         }
45         g << sol << "\n";
46     }
47     else
48     {
49         f >> cuv[N++];
50         Len[0] = strlen(cuv[0]);
51         f >> s;
52         while( --Ns )
53         {
54             ls = strlen(s);
55             for(i=0; i < N; ++i)
56             {
57                 if ( !Erase[i] )
58                 {
59                     lc = Len[i];
60                     if (lc >= ls)
61                     {
62                         p = strchr(cuv[i], s[0]);
63                         ok = (p != 0);
64                         for(j=1; j<lc && ok; ++j)
65                         {
66                             L = lc - (p - cuv[i]);
67                             p = strchr(p + 1, s[j]);
68                             ok = (p != 0 && L >= ls - j);
69                         }
70                         if (ok) break;
71                     }
72                     else
73                     {
74                         p = strchr(s, cuv[i][0]);
75                         ok = (p != 0);
76                         for(j=1; j<lc && ok; ++j)
77                         {
78                             L = ls - (p - s);
79                             p = strchr(p + 1, cuv[i][j]);
80                             ok = (p != 0 && L >= lc - j);
81                         }
82                         if (ok)
83                         {
84                             Erase[i] = 1;
85                             ok = 0;
86                         }
87                     }
88                 }
89             }
90
91             if (!ok)
92             {
93                 Len[N] = ls;
94                 strcpy(cuv[N++], s);
95             }
96             f >> s;
97         }
98
99         nr = N;
100        for(i=0; i < N; ++i)
101            if (Erase[i]) --nr;

```

```

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

```

Listing 7.1.2: `interessant_en2.cpp`

```

1  /*
2      prof. Eugen Nodea
3      Colegiul National "Tudor Vladimirescu", Tg-Jiu
4  */
5  # include <fstream>
6  # include <cstring>
7
8  using namespace std;
9
10 ifstream f("interesant.in");
11 ofstream g("interesant.out");
12
13 const int Dim = 5003;
14 typedef char CUV[Dim];
15
16 char s[Dim], sol[Dim];
17 int N, Max, nr, Ns;
18 char cuv[201][Dim], *p;
19 bool Erase[201];
20 int Len[201];
21
22 int main()
23 {
24     int l, i, j, ls, lc, P, L;
25     bool ok;
26
27     f >> P >> Ns;
28     f.get();
29     if ( P == 1 )
30     {
31         f >> s;
32         while( !f.eof() )
33         {
34             l = strlen(s);
35             if (l > Max)
36             {
37                 memcpy(sol, s, l);
38                 Max = l;
39             }
40             else
41                 if (l == Max && memcmp(sol, s, l) > 0 )
42                     memcpy(sol, s, l);
43             f >> s;
44         }
45         g << sol << "\n";
46     }
47     else
48     {
49         f >> cuv[N++];
50         Len[0] = strlen(cuv[0]);
51         f >> s;
52         while( --Ns ){
53             ls = strlen(s);
54             for(i=0; i < N; ++i)
55             {
56                 if ( !Erase[i] )
57                 {
58                     lc = Len[i];
59                     if (lc >= ls)
60                     {
61                         p = (char*) memchr(cuv[i], s[0], lc);
62                         ok = (p != 0);
63                         j = 1;
64                         while(j<ls && ok)

```

```

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

```

Listing 7.1.3: interesant\_en3.cpp

```

1  /*
2   prof. Eugen Nodea
3   Colegiul National "Tudor Vladimirescu", Tg-Jiu
4  */
5  #include <fstream>
6  #include <vector>
7  #include <string>
8
9  using namespace std;
10
11 ifstream f("interesant.in");
12 ofstream g("interesant.out");
13
14 string s, sol;
15 int N, p, Max, nr;
16 vector <string> cuv;
17 bool Erase[201];
18 int L[201];
19
20 int main()
21 {
22     int l, i, j, k, ls, lc, M;
23     bool ok;
24
25     f >> p >> N; f.get();

```

```

26     if ( p == 1 ){
27         for(i=1; i<=N; ++i)
28             {
29                 f >> s;
30                 l = s.length();
31                 if (l > Max)
32                     {
33                         sol = s;
34                         Max = l;
35                     }
36                 else
37                     if (l == Max && sol > s)
38                         sol = s;
39             }
40         g << sol << "\n";
41     }
42     else
43     {
44         f >> s;
45         cuv.push_back(s);
46         L[0] = cuv[0].length();
47         M = N;
48         while( --M )
49             {
50                 f >> s;
51                 ls = s.length();
52                 for(i=0; i<cuv.size(); ++i)
53                     {
54                         if ( !Erase[i] )
55                             {
56                                 lc = L[i];
57                                 if (lc >= ls)
58                                     {
59                                         k = cuv[i].find_first_of(s[0]);
60                                         ok = (k >= 0);
61                                         j = 1;
62                                         while ( j<ls && ok )
63                                             {
64                                                 k = cuv[i].find_first_of(s[j], k+1);
65                                                 ok = (k >= 0 && lc - k >= ls - j);
66                                                 ++j;
67                                             }
68                                         if (ok)
69                                             {
70                                                 ++nr;
71                                                 break;
72                                             }
73                                     }
74                                 else
75                                     {
76                                         k = s.find_first_of(cuv[i][0]);
77                                         ok = (k >= 0);
78                                         j = 1;
79                                         while ( j<lc && ok )
80                                             {
81                                                 k = s.find_first_of(cuv[i][j], k+1);
82                                                 ok = (k >= 0 && ls - k >= lc - j);
83                                                 ++j;
84                                             }
85                                         if (ok)
86                                             {
87                                                 Erase[i] = 1;
88                                                 ++nr;
89                                                 ok = 0;
90                                             }
91                                     }
92                             }
93                     }
94                 if (!ok)
95                     {
96                         cuv.push_back(s);
97                         i = cuv.size() - 1;
98                         L[i] = ls;
99                     }
100             }
101         g << N - nr << "\n";

```

```

102         for(i=0; i<cuv.size(); ++i)
103             if (!Erase[i])
104                 g << cuv[i] << "\n";
105     }
106     return 0;
107 }

```

Listing 7.1.4: interesant\_gc1.cpp

```

1  // prof. Constantin Galatan
2
3  #include <fstream>
4  #include <cstring>
5  #include <iostream>
6  #include <algorithm>
7
8  using namespace std;
9
10 ifstream fin("interesant.in");
11 ofstream fout("interesant.out");
12
13 inline bool Substr(int& n, int& m, char* a, char* b);
14
15 int N, p, n, Lmax, Ns, len[201];
16 char s[5003], smax[5003], S[201][5003];
17 bool isSubs[201];
18
19 int main()
20 {
21     fin >> p >> Ns;
22     fin.get();
23
24     if ( p == 1 )
25     {
26         while ( fin.getline(s, 10003))
27         {
28             n = strlen(s);
29             if (n > Lmax || (n == Lmax && strcmp(s, smax) < 0))
30             {
31                 Lmax = n;
32                 strcpy(smax, s);
33             }
34         }
35         fout << smax << '\n';
36         fout.close();
37         exit(0);
38     }
39     bool ok;
40     int k = 0;
41     while (fin.getline(S[N], 10003)) // s - indicele sirului curent
42     {
43         ok = true;
44         len[N] = strlen(S[N]);
45         for (int j = 0; j < N; ++j)
46         {
47             if ( isSubs[j] ) continue;
48             if ( len[N] <= len[j] )
49                 if( Substr(len[N], len[j], S[N], S[j])) // S[i] inclus in S[j]
50                 {
51                     ok = false;
52                     break;
53                 }
54             if ( Substr(len[j], len[N], S[j], S[N]) )
55             {
56                 isSubs[j] = true;
57                 k++;
58             }
59         }
60         if ( ok ) N++;
61     }
62
63     fout << N - k << '\n';
64     for (int i = 0; i < N; ++i)
65         if ( !isSubs[i] )
66             fout << S[i] << '\n';

```

```

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

```

Listing 7.1.5: interesant\_gc2.cpp

```

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

```

---

```

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

```

---

Listing 7.1.6: interesant\_gc3.cpp

---

```

1 // prof. Constantin Galatan
2 #include <fstream>
3 #include <cstring>
4 #include <algorithm>
5 #include <vector>
6
7 using namespace std;
8
9 ifstream fin("interesant.in");
10 ofstream fout("interesant.out");
11
12 inline bool Substr(int& n, int& m, char* a, char* b);
13 inline int Bs(const vector<int>& v, int pz);
14
15 int N, p, Lmax, Ns, n, k;
16 char s[5003], smax[5003], S[201][5003];
17 int len[201];
18 bool isSubs[201];
19 vector<int> P[200]['z' - '0' + 1];
20
21 int main()
22 {
23     fin >> p >> Ns;
24
25     fin.get();
26
27     if ( p == 1 )
28     {
29         while ( fin.getline(s, 10003))
30         {
31             n = strlen(s);
32             if (n > Lmax || (n == Lmax && strcmp(s, smax) < 0))
33             {
34                 Lmax = n;
35                 strcpy(smax, s);
36             }
37         }
38         fout << smax << '\n';
39         fout.close();

```

---



```

40     exit(0);
41 }
42 bool ok;
43
44
45 while (fin.getline(S[N], 5003))
46 {
47     ok = true;
48     len[N] = strlen(S[N]);
49     for (int i = 'a'; i <= 'z'; ++i)
50     {
51         P[N][i - 'a'].clear();
52         P[N][i - 'a'].shrink_to_fit();
53     }
54     for (char* c = S[N]; *c; ++c)
55         P[N][ *c - 'a' ].push_back(c - S[N]);
56
57     for (int j = 0; j < N; ++j)
58     {
59         if ( isSubs[j] ) continue;
60
61         if ( len[N] <= len[j] )
62         {
63             if ( Substr(N, j, S[N], S[j]) ) // S[N] subsir in S[j]
64             {
65                 ok = false;
66                 break;
67             }
68         }
69         else
70         {
71             if ( Substr(j, N, S[j], S[N]) )
72             {
73                 isSubs[j] = true;
74                 k++;
75             }
76         }
77     }
78     if ( ok ) N++;
79 }
80
81 fout << N - k << '\n';
82 for (int i = 0; i < N; ++i)
83     if ( !isSubs[i] )
84         fout << S[i] << '\n';
85
86 fin.close();
87 fout.close();
88
89 return 0;
90 }
91
92 inline int Bs(const vector<int>& v, int val)
93 {
94     int st = 0, dr = v.size() - 1, m, next_pos = -1;
95
96     while ( st <= dr )
97     {
98         m = (st + dr) / 2;
99         if ( v[m] > val )
100         {
101             dr = m - 1;
102             next_pos = v[m];
103         }
104         else
105             st = m + 1;
106     }
107     return next_pos;
108 }
109
110 inline bool Substr(int& i, int& j, char* a, char* b) // caut a in b
111 {
112     int n = len[i], m = len[j];
113     int pz(-1);
114     for (int i = 0; i < n; ++i)
115     {

```

```

116         pz = Bs(P[j][a[i] - 'a'], pz); // caut in P[litera] prima valoare
117                                         // mai mare decat pz
118         if ( pz == -1 ) // n-am gasit
119             return false;
120         if ( m - pz < n - i)
121             return false;
122     }
123
124     return true;
125 }

```

Listing 7.1.7: interesant\_mot.cpp

```

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

```

Listing 7.1.8: interesant\_nv.cpp

```

1 // prof. Nicu Vlad
2 #include <algorithm>
3 #include <fstream>
4 #include <cstring>
5
6 #define N 205
7 #define M 10010
8
9 using namespace std;
10
11 ifstream cin("interesant.in");
12 ofstream cout("interesant.out");
13
14 char A[N][M], AX[M],AY[M];
15 bool viz[N];
16
17 bool verif(char A[], char B[])
18 {

```

```

19     int x=strlen(A);
20     int y=strlen(B);
21     int i=0,j=0;
22     bool ok=false;
23     while(i<x&& j<y)
24     {
25         if(A[i]!=B[j]);
26         else j++;
27         i++;
28         if(i==x && j<y) return false;
29     }
30     if(j==y) return true;
31     return false;
32 }
33
34 int main()
35 {
36     int cer=0,n;
37     cin>>cer>>n;
38     if(cer==1)
39     {
40         int i=0,maxi=0;
41         for(i=0; i<n; i++)
42         {
43             cin>>AX;
44             int x=strlen(AX);
45             if(x>maxi)
46             {
47                 maxi=x;
48                 strcpy(AY,AX);
49             }
50             else
51                 if(x==maxi)
52                     if(strcmp(AY,AX)>0)
53                         strcpy(AY,AX);
54         }
55         cout<<AY;
56     }
57     else
58     {
59
60         for(int i=0; i<n; i++)
61         {
62             cin>>A[i];
63             int l=strlen(A[i]);
64             for(int j=0; j<i; j++)
65             {
66                 if(!viz[j])
67                 {
68                     int k=strlen(A[j]);
69                     if(l>=k)
70                         if(verif(A[i],A[j])) viz[j]=1;
71                     else
72                         if(verif(A[j], A[i])) viz[i]=1;
73                 }
74             }
75         }
76         int nr=n;
77         for(int j=0; j<n; j++)
78             if(viz[j]) nr--;
79         cout<<nr<<"\n";
80         for(int j=0; j<n; j++)
81             if(!viz[j]) cout<<A[j]<<"\n";
82     }
83     return 0;
84 }

```

### 7.1.3 \*Rezolvare detaliată

## 7.2 miting

### Problema 2 - miting

100 de puncte

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

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

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

### Cerințe

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

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

### Date de intrare

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

Pe prima linie, un număr natural  $p$ , care poate avea doar valoarea 1 sau 2.

Pe a doua linie două numere naturale  $n$  și  $m$ , separate printr-un spațiu.

Pe a treia linie, cuvântul *cuv*.

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

### Date de ieșire

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

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

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

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

### Restricții și precizări

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

## Exemple

miting.in	miting.out	Explicație
<pre> 1 4 5 JOS # _ # # _ S # J # # _ </pre>	9	<p>Submatricea de arie minimă care include toate literele are colțul stînga sus la linia 1 și coloana 3 și colțul dreapta jos la linia 3 și coloana 5. Aria este egală cu numărul de zone acoperite:</p> $3 \times 3 = 9.$ <p><b>Atenție! Pentru acest test se rezolvă doar cerința 1.</b></p>

Figura 7.1: Miting

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

Figura 7.2: Miting

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

Figura 7.3: Miting

Timp maxim de executare/test: 1.0 secunde

Memorie: total 16 MB

Dimensiune maximă a sursei: 10 KB

## 7.2.1 Indicații de rezolvare

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

## Cerința 1. (20 puncte)

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

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

**Cerința 2.** (80 de puncte)

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

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

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

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

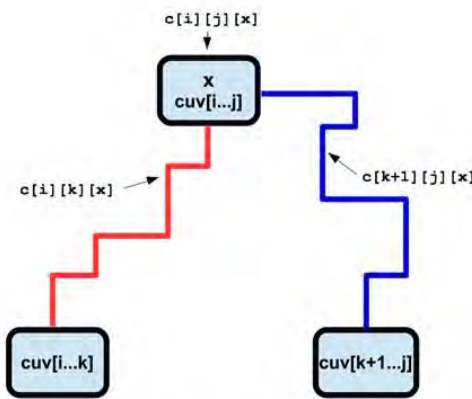


Fig. 1

Figura 7.4: mitingIR1

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

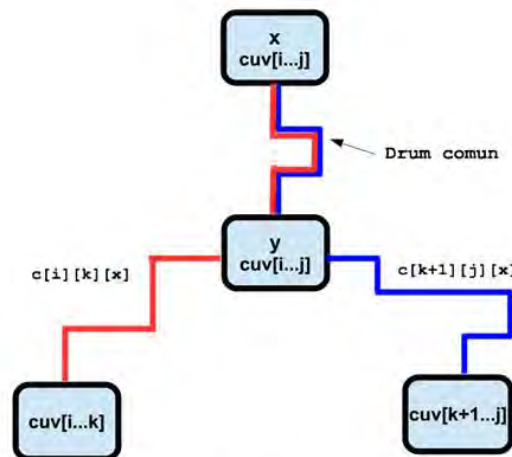


Fig. 2

Figura 7.5: mitingIR2

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

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

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

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

## 7.2.2 Cod sursă

Listing 7.2.1: miting.cpp

---

```

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

```

```

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

```



```

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

```

Listing 7.2.2: miting\_en.cpp

```

1  // prof. Eugen Noddea
2  # include <fstream>
3  # include <iostream>
4  # include <cstring>
5
6  # define INF 9999
7  # define NR 105
8  # define N 11
9  # define mp make_pair
10 # define f first
11 # define s second
12
13 using namespace std;
14
15 ifstream f("miting.in");
16 ofstream g("miting.out");
17
18 short dx[] = {0, 1, 0, -1};
19 short dy[] = {1, 0, -1, 0};
20
21 int p, n, m, l, CiQ, CsQ, K;
22 short Costmin[N][N][NR*NR], nr[NR][NR], poz[NR], L[NR*NR], C[NR*NR],
23     X[NR], Y[NR], Min, test[N][N];
24 char a[NR][NR], cuv[N];
25 pair <short, short> q[NR*NR*NR];
26
27 void bordare ()
28 {
29     for (int i=0; i<=m+1; ++i)
30         a[0][i] = a[n+1][i] = '#';
31     for (int i=0; i<=n+1; ++i)
32         a[i][0] = a[i][m+1] = '#';
33 }
34
35 void LEE (short i, short j)
36 {
37     short k, lv, cv, l, c, urm, act;
38
39     while (CiQ <= CsQ)
40     {
41         l = q[CiQ].f; c = q[CiQ].s; ++CiQ;
42         act = nr[l][c];
43         for (k=0; k<4; ++k)
44         {
45             lv = l + dx[k]; cv = c + dy[k];
46             urm = nr[lv][cv];
47             if (urm != 0 &&
48                 Costmin[i][j][urm] > Costmin[i][j][act] + 1)
49             {
50                 Costmin[i][j][urm] = Costmin[i][j][act] + 1;
51                 q[++CsQ] = mp(lv, cv);
52             }
53         }
54     }
55 }

```

```

56
57 int main ()
58 {
59     short i, j, k, lmin, cmin, lmax = 0, cmax = 0, I, J;
60
61     f >> p >> n >> m; f.get();
62     bordare ();
63     f.getline(cuv+1, N);
64     l = strlen(cuv+1);
65     for (i=1; i<=l; ++i) //pozitia unei litere in cuvant
66         poz[cuv[i]] = i;
67     for (i=1; i<=n; ++i) //citesc matricea
68         f >> (a[i]+1);
69
70     lmin = INF;
71     cmin = INF;
72     for (i=1; i<=n; ++i)
73     {
74         for (j=1; j<=m; ++j)
75         {
76             if (a[i][j] != '#')
77             { //loc pe unde se poate merge
78                 ++K;
79                 nr[i][j] = K;
80                 L[K] = i; C[K] = j;
81             }
82             if (a[i][j] != '#' && a[i][j] != '_')
83             { // e litera
84                 lmin = min(lmin, i);
85                 cmin = min(cmin, j);
86
87                 lmax = max(lmax, i);
88                 cmax = max(cmax, j);
89
90                 X[poz[a[i][j]]] = i; //pozitia literelor
91                 Y[poz[a[i][j]]] = j;
92             }
93         }
94     }
95     if (p == 1)
96     {
97         g << (lmax-lmin+1) * (cmax-cmin+1) << "\n";
98         return 0;
99     }
100     for (i=1; i<=l; ++i)
101     {
102         for (j=1; j<=K; ++j)
103             Costmin[i][j][K] = INF; // Costmin[i][j][K] - costul minim
104                                     // pentru a ajunge cu literele dintre i-j
105                                     // pe a K-a pozitie libera din matrice
106
107         Costmin[i][i][nr[X[i]][Y[i]]] = 0;
108
109         CiQ = 1; CsQ = 1;
110         q[CiQ] = mp(X[i], Y[i]);
111         LEE (i, i);
112     }
113
114     for (k=2; k<=l; ++k) //lungimea 'echipei'
115         for (i=1; i<=l-k+1; ++i)
116         { //pozitia de start a 'echipei'
117             j = i+k-1;
118             CiQ = 1; CsQ = 0;
119             test[i][j] = INF;
120
121             for (I=1; I<=K; ++I)
122             { //pozitia de pe harta care e libera
123                 Costmin[i][j][I] = INF;
124                 Min = Costmin[i][j][I];
125
126                 for (J=i; J<j; ++J)
127                     Min = min(Min, (short) (Costmin[i][J][I] +
128                                             Costmin[J+1][j][I]));
129
130                 Costmin[i][j][I] = Min;
131                 q[++CsQ] = mp(L[I], C[I]);

```

```

132         }
133
134         LEE (i, j);
135
136         for (I=1; I<=K; ++I)
137             test[i][j] = min(test[i][j], Costmin[i][j][I]);
138     }
139     Min = INF;
140     for (i=1; i<=K; ++i)
141         Min = min(Min, Costmin[1][1][i]);
142
143     if (Min == INF) g << "-1\n";
144     else g << Min << "\n";
145     return 0;
146 }

```

Listing 7.2.3: miting\_gc2.cpp

```

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

```

```

58     }
59     nc = cuv.size();
60
61
62     vector<int> pos('Z' + 1); // pozitiile caracterelor in cuvnt
63     for ( int i = 0; i < nc; ++i)
64         pos[cuv[i]] = i;
65
66     // Dupa sortare primele nc celule vor contine literele i ordinea din cuvnt
67     sort(D.begin(), D.begin() + nc,
68         [&pos](const PII& p1, const PII& p2)
69         { return pos[H[p1.first][p1.second]] < pos[H[p2.first][p2.second]]; }
70     );
71
72     n = D.size();
73
74     for (int k = 0; k < n; ++k)
75     {
76         int i = D[k].first, j = D[k].second;
77         z[i][j] = k;
78     }
79
80     for (int i = 0; i < nc; ++i)
81     {
82         for (int u = 0; u < n; ++u)
83             for (int j = 0; j < nc; ++j)
84                 c[i][j][u] = Inf;
85
86         c[i][i][i] = 0;
87     }
88
89     for ( int l = 1; l <= nc; ++l)
90         for (int i = 0; i < nc - l + 1; ++i)
91         {
92             short j = i + l - 1, old;
93             queue<int> Q;
94             for (int x = 0; x < n; ++x)
95             {
96                 if (l >= 2)
97                 {
98                     old = c[i][j][x];
99                     for (int k = i; k < j; ++k)
100                         c[i][j][x] = min(c[i][j][x],
101                             short(c[i][k][x] + c[k + 1][j][x]));
102                 }
103
104                 if ( c[i][j][x] < old || c[i][j][x] == 0)
105                     Q.push(x);
106             }
107             if ( l == nc ) break;
108             int I, J, iv, jv, x, y;
109             while ( !Q.empty() )
110             {
111                 x = Q.front(); Q.pop();
112                 I = D[x].first;
113                 J = D[x].second;
114
115                 for (int d = 0; d < 4; ++d )
116                 {
117                     iv = I + di[d];
118                     jv = J + dj[d];
119                     y = z[iv][jv];
120                     if ( Ok(iv, jv) && c[i][j][y] > c[i][j][x] + 1)
121                     {
122                         c[i][j][y] = c[i][j][x] + 1;
123                         Q.push(y);
124                     }
125                 }
126             }
127         }
128
129     short res = Inf;
130     for (short x = 0; x < n; ++x)
131         res = min(res, c[0][nc - 1][x]);
132
133     if ( res != Inf )

```

---

```

134         fout << res << '\n';
135     else
136         fout << -1 << '\n';
137
138     fout.close();
139 }
140
141 inline bool Ok(int i, int j)
142 {
143     if ( i < 0 || j < 0 || i >= N || j >= M ) return false;
144     if ( H[i][j] == '#' ) return false;
145     return true;
146 }

```

---

Listing 7.2.4: miting\_gc3.cpp

---

```

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

```

```

60     vector<int> pos('Z' + 1); // pozitiile caracterelor in cuvant
61     for ( int i = 0; i < nc; ++i)
62         pos[cuv[i]] = i;
63
64     // Dupa sortare primele nc celule vor contine literele i ordinea din cuvant
65     sort(D.begin(), D.begin() + nc, [&pos](const Pair& p1, const Pair& p2)
66         { return pos[H[p1.first][p1.second]] < pos[H[p2.first][p2.second]]; }
67         );
68
69     n = D.size();
70     for (int k = 0, i, j; k < n; ++k)
71     {
72         i = D[k].first, j = D[k].second;
73         cel[i][j] = k;
74     }
75
76     for (int i = 0; i < nc; ++i)
77     {
78         for (int x = 0; x < n; ++x)
79             for (int j = 0; j < nc; ++j)
80                 c[i][j][x] = Inf;
81
82         c[i][i][i] = 0;
83         Q[++R] = i;
84         Lee(i, i);
85     }
86
87     for ( int l = 2; l <= nc; ++l)
88         for (int i = 0; i < nc - l + 1; ++i)
89         {
90             int j = i + l - 1, cmin;
91             for (int x = 0; x < n; ++x)
92             {
93                 cmin = c[i][j][x];
94                 for (int k = i; k < j; ++k)
95                     cmin = min(cmin, c[i][k][x] + c[k + 1][j][x]);
96
97                 if ( cmin < c[i][j][x] )
98                 {
99                     c[i][j][x] = cmin;
100                     Q[++R] = x;
101                 }
102             }
103             if ( l < nc ) Lee(i, j);
104         }
105
106     short res = Inf;
107     for (short x = 0; x < n; ++x)
108         res = min(res, c[0][nc - 1][x]);
109
110     fout << res << '\n';
111
112     fout.close();
113 }
114
115 inline void Lee(short i, short j)
116 {
117     short I, J, iv, jv, x, y;
118     while ( L <= R )
119     {
120         x = Q[L++];
121         I = D[x].first; J = D[x].second;
122
123         for (int dir = 0; dir < 4; ++dir )
124         {
125             iv = I + di[dir];
126             jv = J + dj[dir];
127             y = cel[iv][jv];
128             if ( Ok(iv, jv) && c[i][j][y] > c[i][j][x] + 1)
129             {
130                 c[i][j][y] = c[i][j][x] + 1;
131                 Q[++R] = y;
132             }
133         }
134     }
135     L = 0, R = -1;

```

```

136 }
137
138 inline bool Ok(short i, short j)
139 {
140     if ( i < 0 || j < 0 || i >= N || j >= M ) return false;
141     if ( H[i][j] == '#' ) return false;
142     return true;
143 }

```

Listing 7.2.5: miting\_gc4.cpp

```

1  /* prof. Constantin Galatan
2  * */
3  #include <fstream>
4  #include <iostream>
5  #include <algorithm>
6  #include <vector>
7  #include <queue>
8
9  using namespace std;
10
11 ifstream fin("miting.in");
12 ofstream fout("miting.out");
13
14 const short MaxN = 61, MaxC = 10, MaxV = 3601, Inf = 1 << 13,
15         di[] = { -1, 0, 1, 0}, dj[] = { 0, 1, 0, -1};
16 using Pair = pair<short, short>;
17
18 short c[MaxC][MaxC][MaxV];
19 int M, N, n, nc, cel[MaxN][MaxN]; // cel[i][j] = numarul de ordine
20                                     // corespunzator celulei (i, j)
21 string cuv;
22 deque<Pair> D;
23
24 char H[MaxN][MaxN], ch;
25
26 struct State
27 {
28     short x, cost;
29     bool operator > (const State& st) const
30     {
31         return cost > st.cost;
32     }
33 };
34
35 priority_queue<State, vector<State>, greater<State> > Q;
36
37 inline void Lee(short i, short j);
38 bool Ok(short i, short j);
39
40 int main()
41 {
42     short imin = MaxN, jmin = MaxN, imax = -1, jmax = -1, p;
43
44     fin >> p >> N >> M >> cuv;
45     fin.get();
46     for (short i = 0; i < N; ++i)
47     {
48         fin.getline(H[i], 101);
49         for (short j = 0; j < M; ++j)
50         {
51             ch = H[i][j];
52             if ( isupper(ch) )
53             {
54                 D.push_front({i, j});
55                 imin = min(imin, i); jmin = min(jmin, j);
56                 imax = max(imax, i); jmax = max(jmax, j);
57             }
58             else
59                 if ( ch != '#' )
60                     D.push_back({i, j});
61         }
62     }
63     fin.close();
64

```

```

65     if ( p == 1 )
66     {
67         fout << (imax - imin + 1) * (jmax - jmin + 1) << '\n';
68         fout.close();
69         exit(0);
70     }
71
72     nc = cuv.size();
73     vector<int> pos('Z' + 1); // pozitiile caracterelor in cuvant
74     for ( int i = 0; i < nc; ++i)
75         pos[cuv[i]] = i;
76
77     // Dupa sortare primele nc celule vor contine literele i ordinea din cuvant
78     sort(D.begin(), D.begin() + nc, [&pos](const Pair& p1, const Pair& p2)
79         { return pos[H[p1.first][p1.second]] < pos[H[p2.first][p2.second]]; }
80     );
81
82     n = D.size();
83     for (int k = 0, i, j; k < n; ++k)
84     {
85         i = D[k].first, j = D[k].second;
86         cel[i][j] = k;
87     }
88
89     for (short i = 0; i < nc; ++i)
90     {
91         for (short x = 0; x < n; ++x)
92             for (short j = 0; j < nc; ++j)
93                 c[i][j][x] = Inf;
94
95         c[i][i][i] = 0;
96         Q.push({i, 0});
97         Lee(i, i);
98     }
99
100     for (short l = 2; l <= nc; ++l)
101         for (short i = 0; i < nc - l + 1; ++i)
102         {
103             short j = i + l - 1, cmin;
104             for (short x = 0; x < n; ++x)
105             {
106                 cmin = c[i][j][x];
107                 for (short k = i; k < j; ++k)
108                     cmin = min(cmin, (short)(c[i][k][x] + c[k + 1][j][x]));
109
110                 if ( cmin < c[i][j][x] )
111                     c[i][j][x] = cmin,
112                     Q.push({x, cmin});
113             }
114             if ( l < nc ) Lee(i, j);
115         }
116
117     short res = Inf;
118     for (short x = 0; x < n; ++x)
119         res = min(res, c[0][nc - 1][x]);
120
121     fout << res << '\n';
122     fout.close();
123 }
124
125 inline void Lee(short i, short j)
126 {
127     short I, J, iv, jv, x, y, cost;
128     while ( !Q.empty() )
129     {
130         x = Q.top().x; cost = Q.top().cost; Q.pop();
131         if ( c[i][j][x] < cost )
132             continue;
133         I = D[x].first; J = D[x].second;
134
135         for (int dir = 0; dir < 4; ++dir)
136         {
137             iv = I + di[dir];
138             jv = J + dj[dir];
139             y = cel[iv][jv];

```



---

```

141         if (Ok(iv, jv) && c[i][j][y] > c[i][j][x] + 1)
142         {
143             c[i][j][y] = c[i][j][x] + 1;
144             Q.push({y, c[i][j][y]});
145         }
146     }
147 }
148 }
149
150 inline bool Ok(short i, short j)
151 {
152     if (i < 0 || j < 0 || i >= N || j >= M) return false;
153     if (H[i][j] == '#') return false;
154     return true;
155 }

```

---

Listing 7.2.6: miting\_gc5.cpp

---

```

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

```

```

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

```

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

---

### 7.2.3 \*Rezolvare detaliată

# Capitolul 8

## OJI 2015

### 8.1 charlie

#### Problema 1 - charlie

100 de puncte

Charlie a decis să se joace cu literele dintr-un șir de caractere, șir ce conține doar literele mici ale alfabetului englez  $'a' \dots 'z'$ . Jocul constă în a elimina litere din șir după următoarea regulă: fie  $L1, L2, L3$  trei litere aflate pe poziții consecutive în șir, atunci litera  $L2$  poate fi eliminată dacă și numai dacă este strict mai mică lexicografic decât literele  $L1$  și  $L3$ .

Pentru a face jocul mai interesant, Charlie atașează eliminării literei  $L2$  un cost egal cu valoarea maximă dintre  $\bar{o}(L1)$  și  $\bar{o}(L3)$ , unde prin  $\bar{o}$  (litera) înțelegem numărul de ordine al literei respective în alfabet ( $\bar{o}('a') = 1, \bar{o}('b') = 2, \dots, \bar{o}('z') = 26$ ). Charlie aplică în mod repetat procedeul de eliminare și calculează suma costurilor eliminărilor efectuate.

#### Cerințe

Fiind dat un șir de caractere să se determine:

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

#### Date de intrare

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

#### Date de ieșire

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

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

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

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

#### Restricții și precizări

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

#### Exemple

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

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

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

**Dimensiune** maximă a sursei: **5 KB**

### 8.1.1 Indicații de rezolvare

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

#### Punctul a) 25 puncte

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

#### Punctul b) 75 puncte

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

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

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

```

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

```

Complexitatea :  $O(n)$

charlie - Rezolvare teoretică (prof. Stelian Ciurea)

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

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

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

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

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

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

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

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

Se disting următoarele situații:

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

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

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

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

Cazul  $a_i > a_i - 1$  se tratează similar. Cu aceasta considerăm afirmația demonstrată.

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

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

Singura problemă este complexitatea algoritmului.

### 8.1.2 Cod sursă

Listing 8.1.1: charlie\_adriana.cpp

```

1  /*
2     Sursa 100 p
3     prof. Adriana Simulescu

```

---

```

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

```

---

Listing 8.1.2: charlie\_eugen0.cpp

---

```

1  /*
2     Sursa oficiala 100p - stiva
3     Complexitate: O (n)
4     prof. Eugen Noddea
5     Colegiul National "Tudor Vladimirescu", Tg-Jiu
6  */
7  # include <fstream>
8  # include <cstring>
9
10 using namespace std;
11
12 # define max(a, b) ((a) < (b) ? (b) : (a))
13 # define Lmax 100003
14
15 ifstream f("charlie.in");
16 ofstream g("charlie.out");
17

```

---

```

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

```

---

Listing 8.1.3: charlie-eugen1.cpp

---

```

1  /*
2     Sursa 100p - implementare stack-STL
3     prof. Eugen Nodea
4     Colegiul National "Tudor Vladimirescu", Tg-Jiu
5  */
6  #include <fstream>
7  #include <cstring>
8  #include <stack>
9
10 using namespace std;
11
12 #define max(a, b) ((a) < (b) ? (b) : (a))
13 #define Lmax 100001
14
15 ifstream f("charlie.in");
16 ofstream g("charlie.out");
17
18 char s[Lmax], vf_a, vf, sol[Lmax];
19 int p, L, semn, k, i, Max, j;

```



```

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

```

Listing 8.1.4: charlie\_eugen2.cpp

```

2      Sursa 43p
3      Complexitate: O(n*n) amortizat
4      prof. Eugen Nodea
5      Colegiul National "Tudor Vladimirescu", Tg-Jiu
6  */
7  # include <fstream>
8  # include <cstring>
9  # include <cassert>
10
11  using namespace std;
12
13  # define max(a, b) ((a) < (b) ? (b) : (a))
14
15  ifstream f("charlie.in");
16  ofstream g("charlie.out");
17
18  char s[1000100];
19  int p, L, ok, k, i, Max, j, mm;
20  long long S;
21
22  int main()
23  {
24      f >> p;
25      f.get();
26      f.getline(s, 1000100);
27      L = strlen(s);
28      if (p == 1) //a)
29      {
30          k = i = 0;
31          while ( i < L )
32          {
33              j = i;
34              while (s[j] > s[j+1] && s[j+1] < s[j+2] && j + 2 < L)
35                  j += 2;
36              if (j - i >= 2)
37              {
38                  if ( j - i + 1 > Max ) Max = j - i + 1;
39                  i = j;
40              }
41              ++i;
42          }
43          g << Max << "\n";
44      }
45      else //b)
46      {
47          S = 0;
48          do
49          {
50              ok = 0;
51              //caut cel mai bun candidat
52              Max = 0; j = 0;
53              for(i=1; i<L-1; ++i)
54                  if (s[i-1] > s[i] && s[i] < s[i+1])
55                  {
56                      mm = max(s[i-1], s[i+1]);
57                      if (mm >= Max)
58                      {
59                          Max = mm;
60                          j = i;
61                      }
62                  }
63              if (Max > 0)
64              {
65                  strcpy(s+j, s+j+1);
66                  L--;
67                  ok = 1;
68                  S += (Max - 'a' + 1);
69              }
70          }while (ok);
71          g << s << "\n";
72          g << S << "\n";
73      }
74      return 0;
75  }

```

---

```

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

```

```

76     fprintf(g, "\n%d", lmax);
77 }
78     return 0;
79 }

```

---

Listing 8.1.6: charlie\_marcel.cpp

---

```

1  /*
2     Sursa 100p !!! (dar care sparge memoria)
3     prof. Marcel Dragan, Sibiu
4  */
5  #include <fstream>
6  #include <cstring>
7  #include <stack>
8
9  using namespace std;
10
11 ifstream in("charlie.in");
12 ofstream out("charlie.out");
13
14 char s[100001];
15 int p;
16
17 void afisare(stack <char> st)
18 {
19     if(!st.empty())
20     {
21         char c=st.top();
22         st.pop();
23         afisare(st);
24         out << c;
25     }
26 }
27
28 int main()
29 {
30     in>>p>>s;
31     int n=strlen(s);
32     if(p==1)
33     {
34         int ctMax=0;
35         for(int i=0;i<n;i++)
36         {
37             // cauta dif<0
38             int dif1=s[i+1]-s[i];
39             while(dif1>=0 && i+2<n)
40             {
41                 i++;
42                 dif1=s[i+1]-s[i];
43             }
44             // numara dif<0 dif>0
45             int dif2=s[i+2]-s[i+1];
46             int ct=0;
47             while(dif1<0 && dif2>0 && i+2+ct<n)
48             {
49                 ct=ct+2;
50                 dif1=s[i+1+ct]-s[i+ct];
51                 dif2=s[i+2+ct]-s[i+1+ct];
52             }
53             // i=i+ct;
54             // verific max
55             if(ctMax<ct)
56                 ctMax=ct;
57         }
58         out<<ctMax+1<<endl;
59     }
60     else
61     {
62         stack <char> st;
63         st.push(s[0]);
64         char top=s[1];
65         int val=0;
66         for(int i=2;i<n;i++)
67         {
68             while(st.size()>0 && s[i] > top && top < st.top())

```

```

69         {
70             val=val+max(s[i]-'a'+1,st.top()-'a'+1);
71             top=st.top();
72             st.pop();
73         }
74         st.push(top);
75         top=s[i];
76     }
77     afisare(st);
78     out << top<<endl;
79     out << val<<endl;
80 }
81 return 0;
82 }

```

Listing 8.1.7: charlie\_radu\_v.cpp

```

1  /*
2   Sursa 38p - idee asemanatoare bouble-sort
3   prof. Radu Visinescu, Ploiesti
4  */
5  #include <iostream>
6  #include <fstream>
7  #include <cstring>
8
9  using namespace std;
10
11 ifstream fin("charlie.in");
12 ofstream fout("charlie.out");
13
14 long long v,p,i,m=0;
15 long long p2,p3,u,save_p,save_u,d,dmax,ok;
16
17 char st[1000000];
18 char t[1000000];
19 char s[1000000];
20
21 int main()
22 {
23     fin>>p;fin>>s;
24     if (p==2) {
25         do{ok=0;
26             i=0; while(i<=strlen(s)-3)
27                 if ((s[i]>s[i+1]) && (s[i+1]<s[i+2]))
28                     {
29                         m=(int)(s[i])-(int)('a')+1;
30                         if (m<(int)(s[i+2])-(int)('a')+1)
31                             m=(int)(s[i+2])-(int)('a')+1;
32                         strcpy(t,s+i+2);
33                         strcpy(s+i+1,t);
34                         v += m; ok=1;
35                     }
36                     else i++;
37                 }while(ok);
38             fout<<s<<'\n';
39             fout<<v<<'\n';
40         }
41     else {dmax=1;save_p=0;save_u=0;
42         p2=0;u=0;
43         d=1;
44         i=0;
45         while (i<=strlen(s)-3)
46             if ((s[i]>s[i+1]) && (s[i+1]<s[i+2]))
47                 {
48                     u=i+2;i=i+2;
49                     d=u-p2+1;
50                     if (dmax<d) {save_p=p2;save_u=u;dmax=save_u-save_p+1; }
51                 }
52             else { i=i+2;p2=i;u=i;}
53         p2=1;u=1;
54         d=1;
55         i=1;
56         while (i<=strlen(s)-3)
57             if ((s[i]>s[i+1]) && (s[i+1]<s[i+2]))
58                 {

```

---

```

59         u=i+2;i=i+2;
60         d=u-p2+1;
61         if (dmax<d) {save_p=p2;save_u=u;dmax=save_u-save_p+1; }
62     }
63     else { i=i+2;p2=i;u=i;}
64     fout<<save_u-save_p+1<<"\n";
65 }
66 fin.close();
67 fout.close();
68     return 0;
69 }

```

---

Listing 8.1.8: charlie.SC.cpp

---

```

1  /*
2      Sursa 100p
3      prof. Stelian Ciurea, Sibiu
4  */
5  #include <iostream>
6  #include <fstream>
7  #include <vector>
8
9  using namespace std;
10
11 vector <int> a;
12
13 ifstream f("charlie.in");
14 ofstream g("charlie.out");
15
16 int tip,maxim=0;
17
18 int main()
19 {
20     //cout << "Hello world!" << endl;
21     f >> tip;
22     char buf;
23     while (f>>buf)
24     {
25         //cout << buf;
26         a.push_back(buf - 'a'+1);
27         maxim = max(maxim,buf - 'a'+1);
28     }
29     if (tip == 1)
30     {
31         int n = a.size();
32         a.push_back(a[n-1]);
33         a.push_back(a[n-1]);
34         a.push_back(a[n-1]);
35         a.push_back(a[n-1]);
36         int lmaxim = 0, lcrt=0;
37         for (int i=1;i<n;i+=2)
38         {
39             if (a[i]<a[i-1]&&a[i]<a[i+1])
40             {
41                 if (lcrt==0)
42                     lcrt=3;
43                 else
44                     lcrt +=2;
45                 //cout << i << ' ' << lcrt << " ";
46             }
47             else
48                 lcrt=0;
49             lmaxim = max(lmaxim,lcrt);
50         }
51         for (int i=2;i<n;i+=2)
52         {
53             if (a[i]<a[i-1]&&a[i]<a[i+1])
54             {
55                 if (lcrt==0)
56                     lcrt=3;
57                 else
58                     lcrt +=2;
59                 //cout << i << ' ' << lcrt << " ";
60             }
61             else

```

```

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

```

Listing 8.1.9: charlie\_zoli.cpp

```

1  /*
2      Sursa 48p - O(n*n) amortizat
3      prof. Zoltan Szabo
4      isj. Mures
5  */
6  #include <fstream>
7  #include <cstring>
8
9  using namespace std;
10
11 ifstream fin("charlie.in");
12 ofstream fout("charlie.out");
13
14 int main()
15 {
16     int n,p,i,max,l,cresc;
17     char cuv[100001];
18     fin>>p;
19     max=1;
20     fin>>cuv;
21     n=strlen(cuv);
22     if (p==1)
23     {
24         cresc=0;
25         l=1;
26         for(i=1;i<n;++i)
27         {
28             if ((cuv[i-1]>cuv[i] and !cresc) or (cuv[i-1]<cuv[i] and cresc))
29             {
30                 l++;           // creste lungimea
31                 cresc=1-cresc; // schimbam conditia pentru relatie
32             }
33             else
34             {
35                 if (l%2==0) l--;
36                 if (l>max) max=l;
37                 if (cuv[i-1]>cuv[i])
38                 {
39                     l=2;
40                     cresc=1;
41                 }
42                 else
43                 {

```

```

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

```

### 8.1.3 \*Rezolvare detaliată

## 8.2 panda

### Problema 2 - panda

100 de puncte

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

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

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

### Cerințe

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

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



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

### Date de intrare

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

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

### Date de ieșire

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

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

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

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

### Restricții și precizări

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

### Exemple

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

2 5 6 4 3 5 1 1 1 2 5 1 2 1 4 3 15 1278 3 1278 1278 1 16 17 18 19 254 20 21 25 26 254 254 254 27 28 29 3 2 254 2 254 4 254 254 254	6 1	Dacă notăm cu 1 țărcurile accesibile și cu 0 cele inaccesibile, obținem următoarea matrice: 0 1 0 1 1 0 1 0 1 0 1 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 Ursulețul se află în țărcul de coordonate (3,5) și poate ajunge la un singur țărc cu mâncare, după 6 secunde. Acest țărc este cel de la coordonatele (5,1); drumul parcurs este: (3,5) → (4,5) → (5,5) → (5,4) → (5,3) → (5,2) → (5,1) <b>Atenție! Pentru acest test se rezolvă doar cerința b).</b>
---	-----	---

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

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

**Dimensiune** maximă a sursei: **10 KB**

### 8.2.1 Indicații de rezolvare

Simulescu Adriana, Liceul Teoretic GRIGORE MOISIL Timișoara

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

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

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

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

Se determină distanța minimă la țărcurile accesibile precum și numărul de țărcuri aflate la această distanță minimă.

### 8.2.2 Cod sursă

Listing 8.2.1: panda\_adriana.cpp

```

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

```

```

29     {in>>papa[i].l>>papa[i].c;}
30     for(i=1;i<=n;i++)
31         for(j=1;j<=m;j++)
32             in>>a[i][j];
33     in.close();
34 }
35
36 void lee(int l,int c)
37 {
38     int st=1,dr=1,ll,cc,pas;
39     ll=1;
40     cc=c;
41     b[ll][cc]=10;
42     coada[dr].l=1;
43     coada[dr].c=c;
44     while(st<=dr)
45     {
46         ll=coada[st].l;
47         cc=coada[st].c;
48         pas=b[ll][cc];
49         if (ll-1>0&&(b[ll-1][cc]==0||b[ll-1][cc]>pas+1))
50             {b[ll-1][cc]=pas+1;
51              dr++;
52              coada[dr].l=ll-1;
53              coada[dr].c=cc;
54             }
55
56         if (ll+1<=n&&(b[ll+1][cc]==0||b[ll+1][cc]>pas+1))
57             {b[ll+1][cc]=pas+1;
58              dr++;
59              coada[dr].l=ll+1;
60              coada[dr].c=cc;
61             }
62
63         if (cc-1>0&&(b[ll][cc-1]==0||b[ll][cc-1]>pas+1))
64             {b[ll][cc-1]=pas+1;
65              dr++;
66              coada[dr].l=ll;
67              coada[dr].c=cc-1;
68             }
69
70         if (cc+1<=m&&(b[ll][cc+1]==0||b[ll][cc+1]>pas+1))
71             {b[ll][cc+1]=pas+1;
72              dr++;
73              coada[dr].l=ll;
74              coada[dr].c=cc+1;
75             }
76
77         st++;
78     }
79 }
80
81
82
83
84 void construire()
85 {int i,j,putere2=1;
86  for(i=1;i<=s;i++)
87      putere2*=2;
88  for(i=1;i<=n;i++)
89      for(j=1;j<=m;j++)
90          {if((((a[i][j]%putere2)^(kod%putere2))&(putere2-1))==putere2-1)
91              b[i][j]=0;
92              else b[i][j]=-1;
93          }
94 }
95
96 int main()
97 {
98     int i,j,nt=0,nrt=0;
99     citire();
100    construire();
101
102    if(p==1)
103        { for(i=1;i<=n;i++)
104          for(j=1;j<=m;j++)

```

```

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

```

Listing 8.2.2: panda\_eugen0.cpp

```

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

```

```

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

```

Listing 8.2.3: panda\_eugen1.cpp

```

1  /*
2      Sursa 100 p Lee+bit
3      prof. Eugen Nodaea, Tg-Jiu
4  */
5  #include <fstream>
6  #include <queue>
7
8  #define inf 1000000
9
10 using namespace std;
11
12 ifstream f("panda.in");
13 ofstream g("panda.out");
14
15 const int dx[] = {0, 1, 0, -1};
16 const int dy[] = {1, 0, -1, 0};
17
18 int p, n, m, t, k, l, c, i, j, x, y, s, Min = inf, nr = -1;
19 bool key[501][501], food[501][501];
20 int T[501][501];
21 short p2[]={1, 2, 4, 8, 16, 32, 64, 128, 256, 1024};
22
23 struct cel
24 {
25     short l, c;
26 };
27
28 queue < cel > q;

```

```

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

```

```

1  /*
2      Sursa 100 p Lee + verif. compl.
3      prof. Eugen Noddea, Tg-Jiu
4  */
5  #include <fstream>
6  #include <queue>
7
8  #define inf 1000000
9
10 using namespace std;
11
12 ifstream f("panda.in");
13 ofstream g("panda.out");
14
15 const int dx[] = {0, 1, 0, -1};
16 const int dy[] = {1, 0, -1, 0};
17
18 int p, n, m, t, k, l, c, i, j, x, y, s, Min = inf, nr = -1;
19 bool key[501][501], food[501][501];
20 int T[501][501];
21
22 struct cel
23 {
24     short l, c;
25 };
26
27 queue < cel > q;
28
29 void lee()
30 {
31     cel x, y;
32     int i;
33     x.l = 1;
34     x.c = c;
35     q.push(x);
36     T[l][c] = 0;
37     while (!q.empty())
38     {
39         x = q.front();
40         q.pop();
41         for(i=0; i<4; ++i)
42         {
43             y.l = x.l + dx[i];
44             y.c = x.c + dy[i];
45             if (key[y.l][y.c] && T[y.l][y.c] > T[x.l][x.c] + 1)
46             {
47                 q.push(y);
48                 T[y.l][y.c] = T[x.l][x.c] + 1;
49                 if (food[y.l][y.c])
50                     if (Min > T[y.l][y.c])
51                         Min = T[y.l][y.c];
52             }
53         }
54     }
55 }
56
57 bool verif(int x, int y, int s)
58 {
59     for (int i=1; i<=s; ++i)
60     {
61         if (x % 2 == y % 2) return 0;
62         x /= 2; y /= 2;
63     }
64     return 1;
65 }
66
67 int main()
68 {
69     f >> p;
70     f >> n >> m >> t;
71     f >> l >> c >> k >> s;
72
73     for(i=1; i<=t; ++i)
74     {
75         f >> x >> y;
76         food[x][y] = 1;

```

```

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

```

Listing 8.2.5: panda\_Liliana\_Schiopu.cpp

```

1  // prof. Liliana Schiopu - C.N.F.B., Craiova
2  // complexitate O(nxm)
3  // algoritmul Lee
4  #include <stdio.h>
5
6  using namespace std;
7
8  FILE *f=fopen("panda.in", "r");
9  FILE *g=fopen("panda.out", "w");
10
11 int n,m,t,i,j,p,a[510][510],b[510][510],x,y,viz[510][510];
12 int l,c,k,s,nr,coada[3][250001],ic,sc,timp[250001];
13 int il[5]={-1,0,1,0},jl[5]={0,1,0,-1};
14 int smin=300000,nt;
15
16 int acc(int x1,int x2)
17 {
18     int i=s;
19     while(i>0)
20     {
21         if((x1%2)^(x2%2)==1)
22         {
23             x1/=2;
24             x2/=2;
25             i--;
26         }
27         else return 0;
28     }
29     return 1;
30 }
31
32 int main()
33 {
34     fscanf(f, "%d", &p);
35     fscanf(f, "%d%d%d", &n, &m, &t);
36     fscanf(f, "%d%d%d%d", &l, &c, &k, &s);
37
38     for(i=1; i<=t; i++)
39     {
40         fscanf(f, "%d%d", &x, &y);
41         b[x][y]=1;
42     }

```



```

43
44     for (i=1; i<=n; i++)
45         for (j=1; j<=m; j++)
46             fscanf(f, "%d", &a[i][j]);
47
48     if (p==1)
49     {
50         for (i=1; i<=n; i++)
51             for (j=1; j<=m; j++)
52                 if (i!=1 || j!=c)
53                     if (acc(a[i][j], k))
54                         nr++;
55         fprintf(g, "%d", nr);
56     }
57     else
58     if (p==2)
59     {
60         ic=1; sc=1;
61         coada[1][ic]=1;
62         coada[2][ic]=c;
63         viz[coada[1][ic]][coada[2][ic]]=1;
64         timp[ic]=0;
65         while (ic<=sc)
66         {
67             for (i=0; i<=3; i++)
68                 if (acc(a[coada[1][ic]+i1[i]][coada[2][ic]+j1[i]], k) &&
69                     a[coada[1][ic]+i1[i]][coada[2][ic]+j1[i]]!=0)
70                     if (viz[coada[1][ic]+i1[i]][coada[2][ic]+j1[i]]==0)
71                     {
72                         sc++;
73                         coada[1][sc]=coada[1][ic]+i1[i];
74                         coada[2][sc]=coada[2][ic]+j1[i];
75                         viz[coada[1][sc]][coada[2][sc]]=1;
76                         timp[sc]=timp[ic]+1;
77                         if (b[coada[1][sc]][coada[2][sc]]==1)
78                         {
79                             if (smin>timp[sc])
80                             {
81                                 smin=timp[sc];
82                                 nt=1;
83                             }
84                             else
85                                 if (smin==timp[sc])
86                                 {
87                                     nt++;
88                                 }
89                         }
90                     }
91             ic++;
92         }
93         fprintf(g, "%d %d", smin, nt);
94     }
95     fclose(f);
96     fclose(g);
97     return 0;
98 }

```

Listing 8.2.6: panda\_marcel.cpp

```

1  /*
2     Sursa 100 p
3     prof. Marcel Dragan, Sibiu
4  */
5  #include <fstream>
6  #include <queue>
7  #include <cstring>
8
9  using namespace std;
10
11  ifstream in("panda.in");
12  ofstream out("panda.out");
13
14  int p, n, m, t, l, c, k, s, h[501][501], lee[501][501], x;
15
16  void citire()

```

```

17 {
18     in>>p>>n>>m>>t;
19     in>>l>>c>>k>>s;
20     for(int i=1;i<=n;i++)
21     {
22         for(int j=1;j<=m;j++)
23         {
24             h[i][j]=0;
25         }
26     }
27     int xt,yt;
28     for(int i=1;i<=t;i++)
29     {
30         in>>xt>>yt;
31         h[xt][yt]=2000000;
32     }
33     int cod=1;
34     for(int i=1;i<=s;i++)
35         cod=cod+2;
36     for(int i=1;i<=n;i++)
37     {
38         for(int j=1;j<=m;j++)
39         {
40             in>>lee[i][j];
41             if(lee[i][j]%cod+k%cod==cod-1)
42                 lee[i][j]=1000000;
43             else
44                 lee[i][j]=-1;
45         }
46         int stop;
47         if(i==24)
48             stop=1;
49     }
50 }
51
52 void afis(int matr[501][501])
53 {
54     int i,j;
55     for(i=1;i<=n;i++)
56     {
57         for(j=1;j<=m;j++)
58         {
59             if(matr[i][j]<100)
60                 out<<matr[i][j]<<'\\t'<<'\\t';
61             else
62                 out<<matr[i][j]<<'\\t';
63         }
64         out<<'\\n';
65     }
66     out<<'\\n';
67 }
68
69 int parurgereLee()
70 {
71     queue <int> lin,col;
72
73     lee[1][c]=1;
74     lin.push(1);
75     col.push(c);
76     int lu,cu,minT=1000000;
77
78     while(!lin.empty())
79     {
80         int lc=lin.front();
81         int cc=col.front();
82         lin.pop();
83         col.pop();
84         for(int i=-1;i<=1;i++)
85         for(int j=-1;j<=1;j++)
86         {
87             lu=lc+i;
88             cu=cc+j;
89             if(i*i!=j*j && 1<=lu && lu<=n && 1<=cu && cu<=m)
90             {
91                 int pas=1;
92                 if(lee[lu][cu]!=-1 && lee[lc][cc]+1<lee[lu][cu])

```

```

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

```

Listing 8.2.7: panda\_radu.cpp

```

1  /*
2     Sursa 100p
3     prof. Radu Visinescu, Ploiesti
4  */
5  #include <iostream>
6  #include <fstream>
7
8  using namespace std;
9
10 ifstream fin("panda.in");
11 ofstream fout("panda.out");
12
13 int n,m,t,a[501][501],lungime[501][501],tarc[501][501];
14 int l,c,k,s;
15 int p;
16
17 int ok(int n,int k,int s)
18 {int vn[10],kn[10],p,o,modulo,i;
19     modulo=1;
20     for(i=1;i<=s;i++) modulo=modulo*2;
21     n=n%modulo;k=k%modulo;

```

```

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

```

Listing 8.2.8: panda\_zoli1.cpp

```

1  /*
2     Sursa 100p
3     prof. Zoltan Szabo
4     isj Mures
5  */
6  #include <fstream>
7

```

[illegible]

```

84         }
85         else
86             b[lin][col-1]=2;          // marcam tarcul ca si loc vizitat,
87                                         // sa nu trecem inca o data
88     }
89     if (col<m and a[lin][col+1] and
90         b[lin][col+1]!=2) // tarcul nevizitat la dreapta
91     {
92         coada[0][++ultim]=lin;        // introducem coordonata tarcului
93                                         // in coada
94         coada[1][ultim]=col+1;
95         coada[2][ultim]=timp+1;      // daca valoarea timpului este >0
96                                         // atunci este fara salt
97
98         if (b[lin][col+1]==1)        // daca s-a ajuns la hrana
99         {
100             gata=1;                  // atunci ne pregatim de finish
101             nr++;                     // numaram inca o solutie cu
102                                         // acelasi timp minim
103             b[lin][col+1]=2;         // marcam tarcul ca si vizitat,
104                                         // sa nu numaram inca o data
105         }
106         else
107             b[lin][col+1]=2;          // marcam tarcul ca si loc vizitat,
108                                         // sa nu trecem inca o data
109     }
110
111     if (lin>1 and a[lin-1][col] and
112         b[lin-1][col]!=2) // tarcul de sus nevizitat
113     {
114         coada[0][++ultim]=lin-1;      // introducem coordonata tarcului
115                                         // in coada
116         coada[1][ultim]=col;
117         coada[2][ultim]=timp+1;      // daca valoarea timpului este >0
118                                         // atunci este fara salt
119
120         if (b[lin-1][col]==1)        // daca s-a ajuns la hrana
121         {
122             gata=1;                  // atunci ne pregatim de finish
123             nr++;                     // numaram inca o solutie cu acelasi
124                                         // timp minim
125             b[lin-1][col]=2;         // marcam tarcul ca si vizitat,
126                                         // sa nu numaram inca o data
127         }
128         else
129             b[lin-1][col]=2;          // marcam tarcul ca si loc vizitat,
130                                         // sa nu trecem inca o data
131     }
132
133     if (lin<n and a[lin+1][col] and
134         b[lin+1][col]!=2) // tarcul de jos nevizitat
135     {
136         coada[0][++ultim]=lin+1;      // introducem coordonata tarcului
137                                         // in coada
138         coada[1][ultim]=col;
139         coada[2][ultim]=timp+1;      // daca valoarea timpului este >0
140                                         // atunci este fara salt
141
142         if (b[lin+1][col]==1)        // daca s-a ajuns la hrana
143         {
144             gata=1;                  // atunci ne pregatim de finish
145             nr++;                     // numaram inca o solutie cu
146                                         // acelasi timp minim
147             b[lin+1][col]=2;         // marcam tarcul ca si vizitat,
148                                         // sa nu numaram inca o data
149         }
150         else
151             b[lin+1][col]=2;          // marcam tarcul ca si loc vizitat,
152                                         // sa nu trecem inca o data
153     }
154     }
155     prim=ultim+1;
156 }
157
158 fout<<timp+1<<" "<<nr<<"\n";
159 fout.close();

```

---

```
160     return 0;  
161 }
```

---

### 8.2.3 \*Rezolvare detaliată

## Capitolul 9

# OJI 2014

### 9.1 ferma

#### Problema 1 - ferma

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

100 de puncte  
*Imagine 1*

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

Figura 9.1: ferma

#### Cerințe

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

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

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

#### Date de intrare

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

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

#### Date de ieșire

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

**Varianta 1** - pentru  $v = 1$ :

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

**Varianta 2** - pentru  $v = 2$ :


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

#### Restricții și precizări



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

**Exemple**

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

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

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

**Dimensiune** maximă a sursei: **10 KB**

**9.1.1 Indicații de rezolvare**

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

**Varianta 1:**

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

**Varianta 2:**

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

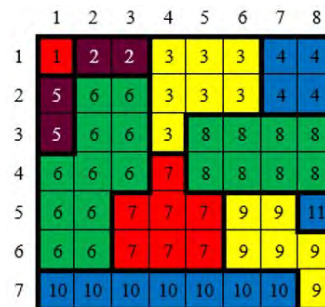


Figura 9.2: fermaIR

**9.1.2 Cod sursă**

Listing 9.1.1: fermadaniel.cpp

---

```

1 // Popa Daniel - Colegiul National "Aurel Vlaicu" Orastie
2 #include <iostream>
3 #include <cstdio>
4 #include <cstring>
5
6 using namespace std;
7
8 const int lmax=402,cmax=402;//dimensiunile maxime ale terenului
9     po[4][2]={{-1,0},{0,-1},{0,1},{1,0}};
10 char h[lmax][cmax],u[lmax*cmax];//cum sunt culturile
11 int a[lmax][cmax],//a[i][j]=x, unde x e numarul parcelei
12     s[lmax*cmax][2],vs=0,//stiva
13     t[lmax*cmax],// t[i]=x, i=nr parcelei, t[i]=aria parcelei i
14     v,m,n,nr=0;
15
16 void citire_init()
17 {
18     FILE *fin=fopen("ferma.in","r");
19     int i;
20     fscanf(fin,"%d\n%d %d\n",&v,&m,&n );
21     for(i=1;i<=m;i++)
22         {fscanf(fin,"%s\n",h[i]+1);}
23     fclose(fin);
24     //bordez marginea cu -1
25     for(i=0;i<=n+1;i++)a[0][i]=a[m+1][i]=-1;
26     for(i=1;i<=m;i++)a[i][0]=a[i][n+1]=-1;
27 }
28
29 void afis()
30 {
31     int i;
32     for(i=1;i<=m;i++)cout<<h[i]+1<<endl;
33 }
34
35 void afis2()
36 {
37     int i,j;
38     for(i=1;i<=m;i++)
39     {
40         for(j=1;j<=n;j++){cout.width(3);cout<<a[i][j];}
41         cout<<endl;
42     }
43 }
44
45 //pun coord in stiva, cresc aria suprafetei nr, si marchez ca am fost pe acolo
46 void push(int l, int c,int nr)
47 {
48     vs++;s[vs][0]=l;s[vs][1]=c;t[nr]++;a[l][c]=nr;
49 }
50
51 //extrag coord din stiva
52 void pop(int &l, int &c)
53 {
54     l=s[vs][0];c=s[vs][1];vs--;
55 }
56
57 //incep de la coord li,co cu suprafata nr
58 int arie(int li,int co,int nr)
59 {
60     int l,c;
61     unsigned char cu=h[li][co];
62     vs=0;//golesc stiva si incep sa pun in ea
63     push(li,co,nr);
64     while(vs>0)//cat timp mai am in stiva
65     {
66         pop(l,c);//extrag din stiva
67         //merg in cele 4 directii
68         if((h[l+1][c]==cu)&&(a[l+1][c]==0))push(l+1,c,nr);
69         if((h[l][c+1]==cu)&&(a[l][c+1]==0))push(l,c+1,nr);
70         if((h[l-1][c]==cu)&&(a[l-1][c]==0))push(l-1,c,nr);
71         if((h[l][c-1]==cu)&&(a[l][c-1]==0))push(l,c-1,nr);
72     }
73     return t[nr];
74 }

```

```

75
76 void v1()
77 {
78     FILE *fout=fopen("ferma.out","w");
79     int i,j,ma=0,x;
80     nr=0; // sunt 0 suprafete gasite la inceput
81     for(i=1;i<=m;i++)
82         for(j=1;j<=n;j++)
83             if(a[i][j]==0)
84                 {
85                     x=arie(i,j,++nr);
86                     if(x>ma)ma=x;
87                 }
88     fprintf(fout,"%d\n",ma);
89     fclose(fout);
90 }
91
92 int bun(int l, int c, int &cu)
93 {
94     int i,j,p=0,ma=0,z,k;
95     char s;
96     for(i=0;i<4;i++)
97         {s=h[l+po[i][0]][c+po[i][1]];
98         if(((s>='a') && (s<='z')) && (s!=h[l][c]))
99             { z=a[l+po[i][0]][c+po[i][1]];
100             p=t[z];
101             for
102 (k=0;k<4;k++)u[a[l+po[k][0]][c+po[k][1]]]=0;
103             u[z]=1;
104             for(j=0;j<4;j++)
105                 if((s==h[l+po[j][0]][c+po[j][1]]) &&
106                    (u[a[l+po[j][0]][c+po[j][1]]]==0))
107                     {p+=t[a[l+po[j][0]][c+po[j][1]]];
108                     u[a[l+po[j][0]][c+po[j][1]]]=1;}
109             if((p!=t[z]) && (p>ma)) {ma=p+1;cu=s;}
110         }
111     return ma;
112 }
113
114 void v2()
115 {
116     FILE *fout=fopen("ferma.out","w");
117     int i,j,ma=0,x,l,c,l2,c2,ma2=0,bnr=0,ku;
118     char cu;
119     nr=0; // sunt 0 suprafete gasite la inceput
120     for(i=1;i<=m;i++)
121         for(j=1;j<=n;j++)
122             if(a[i][j]==0)
123                 {x=arie(i,j,++nr);
124                 if(x>ma2){ma2=x;l2=i;c2=j;}
125                 }
126     // caut sa lipesc 2,3,4 suprafete a.i. ara lor sa fie mai mare decat
127     // cea mai mare suprafata de cultura
128     for(i=1;i<=m;i++)
129         for(j=1;j<=n;j++)
130             {
131                 x=bun(i,j,ku);
132                 if((x>ma) && (x>ma2)) {ma=x;l=i;c=j;cu=ku;}
133             }
134     if(ma==0) //nu am reusit sa unesc 2 zone cresc cea mai mare zona cu o casuta
135     {
136         cout<<"bun"<<ma2;
137         cu=h[l2][c2];bnr=a[l2][c2];
138         for(i=1;(i<=m) && (ma==0);i++) // parcurg matricea
139             for(j=1;(j<=n) && (ma==0);j++)
140                 if(h[i][j]!=cu) //daca casuta curenta e de alta culoare decat
141                     // cea ce trebuie marita
142                     if(a[i+1][j]==bnr) {l=i;c=j;ma=1;} //verific daca casuta adiacenta
143                     // face parte din parcela ce trebuie marita
144                     else if(a[i][j+1]==bnr) {l=i;c=j;ma=1;}
145                     else if(a[i][j-1]==bnr) {l=i;c=j;ma=1;}
146                     else
147                     if(a[i-1][j]==bnr) {l=i;c=j;ma=1;}
148     }
149     fprintf(fout,"%d %d\n%c\n",l,c,cu);

```

```

149     fclose(fout);
150 }
151
152 int main()
153 {
154     citire_init();
155     if(v==1)v1();
156     else v2();
157
158     return 0;
159 }

```

Listing 9.1.2: fermavlad.cpp

```

1 //Prof Nicu Vlad-Laurentiu - Liceul Teoretic "Mihail Kogalniceanu" Vaslui
2
3 #include <algorithm>
4 #include <cstdio>
5
6 using namespace std;
7
8 const int N=405;
9
10 int n, m, nrzones;
11 char a[N][N],cu;
12 int
13     b[N][N], dx[]={-1, 0, 1, 0}, dy[]={0, 1, 0, -1}, d[N*N],p;
14 bool c[N*N];
15 void filll(int x, int y)
16 {
17     b[x][y]=nrzones;
18     d[nrzones]++;
19     for(int i=0;i<4;i++)
20     {
21         if
22         (!b[x+dx[i]][y+dy[i]]&&a[x][y]==a[x+dx[i]][y+dy[i]])
23             filll(x+dx[i], y+dy[i]);
24     }
25 }
26 int main()
27 {
28     freopen("ferma.in", "r", stdin);
29     freopen("ferma.out", "w", stdout);
30     int i, j, k, l, sol=0, s=0,v,ma=0,p=0;
31     pair<int, int> soli;
32
33     scanf("%d\n%d%d", &v, &n, &m);
34     for(i=1;i<=n;i++)
35     {
36
37         scanf("%s", a[i]+1);
38
39     }
40     for(i=1;i<=n;i++)
41     {
42         for(j=1;j<=m;j++)
43         {
44             if(!b[i][j])
45             {
46                 nrzones++;
47                 filll(i, j);
48                 if(ma<d[nrzones]){ma=d[nrzones];}
49             }
50         }
51     }
52     for(i=1;i<=n;i++)
53     {
54         for(j=1;j<=m;j++)
55         {
56             char aux=a[i][j];
57             for(k=0;k<4;k++)
58             {
59                 a[i][j]=a[i+dx[k]][j+dy[k]];

```

```

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

```

Listing 9.1.3: flore\_nerecursiv.cpp

```

1 //Florentina Ungureanu - Colegiul National de Informatica Piatra-Neamt
2 #include <fstream>
3 #include <iostream>
4 #include <string.h>
5
6 #define nmax 410
7
8 using namespace std;
9
10 char a[nmax][nmax], c, cmax, cs;
11 unsigned b[nmax][nmax], z[nmax*nmax], smax, ssmax, s;
12 unsigned short x[nmax*nmax], y[nmax*nmax];
13 unsigned m, n, np, max, imax, jmax, is, js;
14
15 ifstream f("ferma.in");
16 ofstream g("ferma.out");
17
18 void suprafata(unsigned i, unsigned j)
19 {
20     int p, u;
21     p=u=1;
22     x[p]=i; y[p]=j;
23     while(p<=u)
24     {
25         i=x[p]; j=y[p];
26         if (!b[i+1][j]&&a[i+1][j]==c)
27         {
28             s++; b[i+1][j]=np;
29             x[++u]=i+1, y[u]=j;
30         }
31         if (!b[i][j+1]&&a[i][j+1]==c)
32         {
33             s++; b[i][j+1]=np;
34             x[++u]=i, y[u]=j+1;
35         }
36         if (!b[i-1][j]&&a[i-1][j]==c)
37         {
38             s++; b[i-1][j]=np;
39             x[++u]=i-1, y[u]=j;
40         }
41         if (!b[i][j-1]&&a[i][j-1]==c)
42         {
43             s++; b[i][j-1]=np;
44             x[++u]=i, y[u]=j-1;
45         }
46     }
47     p++;
48 }
49
50 void maxim(int i, int j, int a)

```

```

45 {
46     if(s>ssmax)
47     {
48         ssmax=s;
49         imax=i;
50         jmax=j;
51         cmax=a;
52     }
53 }
54
55 int main()
56 {
57     unsigned i, j, v;
58
59     char cuv[405];
60     f>>v;
61     f>>m>>n;
62     f.get();
63     for(i=1; i<=m; i++)
64     {
65         f.getline(cuv, 405);
66         strcpy(a[i]+1, cuv);
67     }
68     np=0;
69     for(i=1; i<=m; i++)
70         for(j=1; j<=n; j++)
71             if(!b[i][j])
72             {
73                 np++;
74                 s=1; c=a[i][j]; b[i][j]=np;
75                 suprafata(i, j);
76                 if(s>smax)
77                 {
78                     smax=s; is=i; js=j; cs=c;
79                 }
80                 z[np]=s;
81             }
82     if(v==1) {g<<smax<<'\\n'; f.close(); g.close(); return 0;}
83     ssmax=0;
84     for(i=1; i<=m; i++)
85         for(j=1; j<=n; j++)
86         {
87             if(a[i][j-1]==a[i][j+1]&&b[i][j-1]!=b[i][j+1]
88                 &&a[i][j-1]==a[i+1][j]&&
89                 b[i][j-1]!=b[i+1][j]&&b[i][j+1]!=b[i+1][j]
90                 &&
91                 a[i][j-1]==a[i-1][j]&&b[i][j-1]!=b[i-1][j]&&
92                 b
93                 [i][j+1]!=b[i-1][j]&&b[i+1][j]!=b[i-1][j])
94             {
95                 s
96                 =z[b[i][j-1]]+z[b[i][j+1]]+z[b[i+1][j]]+z[b[i-1][j]]+1;
97                 maxim(i, j, a[i][j-1]);
98             }
99             else
100             if(a[i][j-1]==a[i][j+1]&&b[i][j-1]!=b[i][j+1]
101                 &&a[i][j-1]==a[i+1][j]&&
102                 b
103                 [i][j-1]!=b[i+1][j]&&b[i][j+1]!=b[i+1][j])
104             {
105                 s
106                 =z[b[i][j-1]]+z[b[i][j+1]]+z[b[i+1][j]]+1;
107                 maxim(i, j, a[i][j-1]);
108             }
109             else
110             if(a[i][j-1]==a[i+1][j]&&b[i][j-1]!=b[i+1][j]

```

```

110             &&a[i][j-1]==a[i-1][j]&&
111             b
112             (i)[j-1]!=b[i-1][j]&&b[i+1][j]!=b[i-1][j])
113             {
114                 s
115                 =z[b[i][j-1]]+z[b[i+1][j]]+z[b[i-1][j]]+1;
116                 maxim(i,j,a[i][j-1]);
117             }
118             else
119             if(a[i][j+1]==a[i+1][j]&&b[i][j+1]!=b[i+1][j]
120             &&a[i][j+1]==a[i-1][j]&&
121             b
122             (i)[j+1]!=b[i-1][j]&&b[i+1][j]!=b[i-1][j])
123             {
124                 s
125                 =z[b[i][j+1]]+z[b[i+1][j]]+z[b[i-1][j]]+1;
126                 maxim(i,j,a[i][j+1]);
127             }
128             else
129             { if(a[i][j-1]==a[i][j+1]&&b[i][j-1]!=b[i][j+1])
130             {
131                 s=z[b[i][j-1]]+z[b[i][j+1]]+1;
132                 maxim(i,j,a[i][j-1]);
133             }
134             if
135             (a[i+1][j]==a[i-1][j]&&b[i+1][j]!=b[i-1][j])
136             {
137                 s=z[b[i+1][j]]+z[b[i-1][j]]+1;
138                 maxim(i,j,a[i+1][j]);
139             }
140             if
141             (a[i][j-1]==a[i-1][j]&&b[i][j-1]!=b[i-1][j])
142             {
143                 s=z[b[i][j-1]]+z[b[i-1][j]]+1;
144                 maxim(i,j,a[i][j-1]);
145             }
146             if
147             (a[i][j-1]==a[i+1][j]&&b[i][j-1]!=b[i+1][j])
148             {
149                 s=z[b[i][j-1]]+z[b[i+1][j]]+1;
150                 maxim(i,j,a[i][j-1]);
151             }
152             if
153             (a[i][j+1]==a[i-1][j]&&b[i][j+1]!=b[i-1][j])
154             {
155                 s=z[b[i][j+1]]+z[b[i-1][j]]+1;
156                 maxim(i,j,a[i][j+1]);
157             }
158             if
159             (a[i][j+1]==a[i+1][j]&&b[i][j+1]!=b[i+1][j])
160             {
161                 s=z[b[i][j+1]]+z[b[i+1][j]]+1;
162                 maxim(i,j,a[i][j+1]);
163             }
164             }
165             }
166             if(ssmax)
167             {
168                 g<<imax<<' '<<jmax<<' \n';
169                 g<<cmax<<' \n';
170             }
171             else
172             for(i=1;i<=m;i++)
173                 for(j=1;j<=n;j++)
174                     if(a[i][j]!=cs&&
175                     (
176                     b[i][j-1]==b[is][js]||b[i][j+1]==b[is][js]||
177                     b
178                     [i+1][j]==b[is][js]||b[i-1][j]==b[is][js]))
179                     {
180                         g<<i<<' '<<j<<' \n';
181                         g<<cs<<' \n';
182                         i=m+1;j=n+1;
183                     }
184             }
185             f.close();
186             g.close();

```

```

173     return 0;
174 }

```

Listing 9.1.4: flore\_recurziv.cpp

```

1 //Florentina Ungureanu - Colegiul National de Informatica Piatra-Neamt
2 #include <fstream>
3 #include <string.h>
4
5 #define nmax 410
6
7 using namespace std;
8
9 char a[nmax][nmax], c, cmax, cs;
10 unsigned b[nmax][nmax], z[nmax*nmax], smax, ssmax, s;
11 unsigned m, n, np, max, imax, jmax, is, js;
12
13 ifstream f("ferma.in");
14 ofstream g("ferma.out");
15
16 void suprafata(unsigned i, unsigned j)
17 {
18     if (!b[i+1][j] && a[i+1][j] == c)
19     {
20         s++; b[i+1][j] = np;
21         suprafata(i+1, j);
22     }
23     if (!b[i][j+1] && a[i][j+1] == c)
24     {
25         s++; b[i][j+1] = np;
26         suprafata(i, j+1);
27     }
28     if (!b[i-1][j] && a[i-1][j] == c)
29     {
30         s++; b[i-1][j] = np;
31         suprafata(i-1, j);
32     }
33     if (!b[i][j-1] && a[i][j-1] == c)
34     {
35         s++; b[i][j-1] = np;
36         suprafata(i, j-1);
37     }
38 }
39
40 int main()
41 {
42     unsigned i, j, v;
43     char cuv[405];
44     f >> v;
45     f >> m >> n;
46     f.get();
47     for(i=1; i<=m; i++)
48     {
49         f.getline(cuv, 405);
50         strcpy(a[i]+1, cuv);
51     }
52     np=0;
53     for(i=1; i<=m; i++)
54     for(j=1; j<=n; j++)
55     if(!b[i][j])
56     {
57         np++;
58         s=1; c=a[i][j]; b[i][j]=np;
59         suprafata(i, j);
60         if(s>smax)
61         {
62             smax=s; is=i; js=j; cs=c;
63         }
64         z[np]=s;
65     }
66     if(v==1) {g<<smax<<'\\n'; f.close(); g.close(); return 0;}
67     ssmax=0;
68     for(i=1; i<=m; i++)
69     for(j=1; j<=n; j++)
70     {
71         if(a[i][j-1]==a[i][j+1] && b[i][j-1]!=b[i][j+1]
72            &&
73            a[i][j-1]==a[i+1][j] && b[i][j-1]!=b[i+1][j] &&
74            b[i][j+1]!=b[i+1][j])

```



```

69         &&
a[i][j-1]==a[i-1][j]&&b[i][j-1]!=b[i-1][j]&&
70         b
[i][j+1]!=b[i-1][j]&&b[i+1][j]!=b[i-1][j])
71     {
72         s
=z[b[i][j-1]]+z[b[i][j+1]]+z[b[i+1][j]]+z[b[i-1][j]]+1;
73         if(s>ssmax)
74         {
75             ssmax=s;
76             imax=i;
77             jmax=j;
78             cmax=a[i][j-1];
79         }
80     }
81     else
82     if(a[i][j-1]==a[i][j+1]&&b[i][j-1]!=b[i][j+1]
83         &&
a[i][j-1]==a[i+1][j]&&b[i][j-1]!=b[i+1][j]&&
84         b[i][j+1]!=b[i+1][j])
85     {
86         s
=z[b[i][j-1]]+z[b[i][j+1]]+z[b[i+1][j]]+1;
87         if(s>ssmax)
88         {
89             ssmax=s;
90             imax=i;
91             jmax=j;
92             cmax=a[i][j-1];
93         }
94     }
95     else
96     if(a[i][j-1]==a[i][j+1]&&b[i][j-1]!=b[i][j+1]
97         &&
a[i][j-1]==a[i-1][j]&&b[i][j-1]!=b[i-1][j]&&
98         b[i-1][j]!=b[i][j+1])
99     {
100         s
=z[b[i][j-1]]+z[b[i][j+1]]+z[b[i-1][j]]+1;
101         if(s>ssmax)
102         {
103             ssmax=s;
104             imax=i;
105             jmax=j;
106             cmax=a[i][j-1];
107         }
108     }
109     else
110     if(a[i][j-1]==a[i+1][j]&&b[i][j-1]!=b[i+1][j]
111         &&
a[i][j-1]==a[i-1][j]&&b[i][j-1]!=b[i-1][j]&&
112         b[i+1][j]!=b[i-1][j])
113     {
114         s
=z[b[i][j-1]]+z[b[i+1][j]]+z[b[i-1][j]]+1;
115         if(s>ssmax)
116         {
117             ssmax=s;
118             imax=i;
119             jmax=j;
120             cmax=a[i][j-1];
121         }
122     }
123     else
124     if(a[i][j+1]==a[i+1][j]&&b[i][j+1]!=b[i+1][j]
125         &&
a[i][j+1]==a[i-1][j]&&b[i][j+1]!=b[i-1][j]&&
126         b[i+1][j]!=b[i-1][j])
127     {
128         s
=z[b[i][j+1]]+z[b[i+1][j]]+z[b[i-1][j]]+1;
129         if(s>ssmax)
130         {
131             ssmax=s;
132             imax=i;
133             jmax=j;

```

```

130             cmax=a[i][j+1];
131         }
132     }
133     else
134     { if(a[i][j-1]==a[i][j+1]&&b[i][j-1]!=b[i][j+1])
135         {
136             s=z[b[i][j-1]]+z[b[i][j+1]]+1;
137             if(s>ssmax)
138             {
139                 ssmax=s;
140                 imax=i;
141                 jmax=j;
142                 cmax=a[i][j-1];
143             }
144             if
145             (a[i+1][j]==a[i-1][j]&&b[i+1][j]!=b[i-1][j])
146             {
147                 s=z[b[i+1][j]]+z[b[i-1][j]]+1;
148                 if(s>ssmax)
149                 {
150                     ssmax=s;
151                     imax=i;
152                     jmax=j;
153                     cmax=a[i+1][j];
154                 }
155             }
156             if
157             (a[i][j-1]==a[i-1][j]&&b[i][j-1]!=b[i-1][j])
158             {
159                 s=z[b[i][j-1]]+z[b[i-1][j]]+1;
160                 if(s>ssmax)
161                 {
162                     ssmax=s;
163                     imax=i;
164                     jmax=j;
165                     cmax=a[i][j-1];
166                 }
167             }
168             if
169             (a[i][j-1]==a[i+1][j]&&b[i][j-1]!=b[i+1][j])
170             {
171                 s=z[b[i][j-1]]+z[b[i+1][j]]+1;
172                 if(s>ssmax)
173                 {
174                     ssmax=s;
175                     imax=i;
176                     jmax=j;
177                     cmax=a[i][j-1];
178                 }
179             }
180             if
181             (a[i][j+1]==a[i-1][j]&&b[i][j+1]!=b[i-1][j])
182             {
183                 s=z[b[i][j+1]]+z[b[i-1][j]]+1;
184                 if(s>ssmax)
185                 {
186                     ssmax=s;
187                     imax=i;
188                     jmax=j;
189                     cmax=a[i][j+1];
190                 }
191             }
192             if
193             (a[i][j+1]==a[i+1][j]&&b[i][j+1]!=b[i+1][j])
194             {
195                 s=z[b[i][j+1]]+z[b[i+1][j]]+1;
196                 if(s>ssmax)
197                 {
198                     ssmax=s;
199                     imax=i;
200                     jmax=j;
201                     cmax=a[i][j+1];
202                 }
203             }
204         }
205     }
206 }

```

```

200         }
201         if(ssmax)
202         {
203             g<<imax<<' '<<jmax<<'\n';
204             g<<cmax<<'\n';
205         }
206         else
207             for(i=1; i<=m; i++)
208                 for(j=1; j<=n; j++)
209                     if(a[i][j]!=cs&&
210                        (
211                            b[i][j-1]==b[is][js] || b[i][j+1]==b[is][js] ||
212                            b
213                            [i+1][j]==b[is][js] || b[i-1][j]==b[is][js]))
214                         {
215                             g<<i<<' '<<j<<'\n';
216                             g<<cs<<'\n';
217                             i=m+1; j=n+1;
218                         }
219         f.close();
220         g.close();
221         return 0;
222     }

```

### 9.1.3 \*Rezolvare detaliată

## 9.2 triunghi

### Problema 2 - triunghi

100 de puncte

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

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

1. Păstrează cel mai scurt bețișor, cel de lungime 2, și creează un set nou adăugând alte bețișoare de lungime mai mare sau egală cu cel inițial. De exemplu, următoarele 5 lungimi sunt corecte: 2, 2, 12, 50, 30.
2. Păstrează toate bețișoarele, și anume 2, 7, 12, pe care le va completa cu alte bețișoare de diferite lungimi (mai scurte sau mai lungi), astfel ca proprietatea anti-triunghi să se păstreze. Următoarele 5 lungimi respectă proprietatea anti-triunghi: 2, 7, 12, 4, 1.

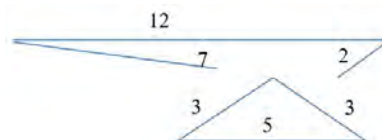


Figura 9.3: triunghi

### Cerințe

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

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

**Varianta 2.** Printre cele  $k$  elemente ale șirului construit se regăsesc toate elementele șirului inițial.

### Date de intrare

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

**Date de ieşire**

Fişierul de ieşire **triunghi.out** va conţine  $k$  numere pe o singură linie.

Dacă valoarea lui  $v$  este 1, atunci fişierul va conţine  $k$  numere naturale cu proprietatea anti-triunghi, separate prin spaţiu, în care cel mai mic element este identic cu minimul şirului dat în fişierul de intrare.

Dacă valoarea lui  $v$  este 2, atunci fişierul va conţine  $k$  numere naturale cu proprietatea anti-triunghi, separate prin spaţiu, printre care se regăsesc toate elementele şirului iniţial.

**Restricţii şi precizări**

- $3 \leq n < k \leq 46$ ;
- $1 \leq$  lungimea unui beţişor  $\leq 2.000.000.000$ ;
- Pentru rezolvarea corectă a primei cerinţe se acordă 30 de puncte, iar pentru cerinţa a doua se acordă 70 de puncte;
- Se garantează că întotdeauna există soluţie;
- Soluţia nu este unică - se admite orice răspuns corect.

**Exemple**

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

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

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

**Dimensiune** maximă a sursei: **5 KB**

**9.2.1 Indicaţii de rezolvare**

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

**Prima cerinţă**

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

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

Dintre toate şirurile cu *proprietatea anti-triunghi*, şirul lui *Fibonacci* este cel care creşte cel mai încet (1, 1, 2, 3, 5, 8, 13, 21, ...)

De aceea, cel mai bun răspuns pentru prima cerinţă, este şirul  $\min * f(i)$ , cu elementele:

$\min, \min, 2 * \min, 3 * \min, 5 * \min, 8 * \min, 13 * \min, 21 * \min, \dots$

**A doua cerinţă**

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

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

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

**9.2.2 Cod sursă**

Listing 9.2.1: `triunghi_LS.cpp`

```

1 // Lukacs Sandor LICEUL ONISIFOR GHIBU ORADEA
2 #include <iostream>
3 #include <fstream>
4
5 using namespace std;
6
7 ifstream f("triunghi.in");
```

```

8  ofstream g("triunghi.out");
9
10 int n,k,m,choise;//m-valoarea minima din vector
11 int nr;//numarul de bete adaugate
12 int v[50], u[50]; //v - initial, u-dupa adaugare
13
14 void citire()
15 {
16     int i;
17     f>>choise>>n>>k;
18     m=2000000000;
19     for(i=1;i<=n;i++)
20     {
21         f>>v[i];//citesc betioasere si calculez valoarea minima
22         if(v[i]<m) m=v[i];
23         u[i]=v[i];//pun toate betele in solutie pt. pct. b
24     }
25 }
26
27 void solutie1()
28 {
29     int b1,b2,b3;
30     g<<m<<" "<<m<<" ";//afisez cel mai mic bat de doua ori
31     nr=2;//am doua bete
32     b1=b2=m;
33     while(nr<k)
34     { //cat timp nu am k bete adaugate
35         b3=b2+b1;//calculez urmatoarea lungime cu care nu pot forma triunghi
36         b1=b2;b2=b3;//trec la urmatoarele 2 bete
37         g<<b2<<" ";//afisez lungimea noului bat
38         nr++; //contabilizez batul
39     }
40     g<<"\n";
41 }
42
43 void sortare()
44 { //sortez betele initiale
45     int i,j,aux;
46     for(i=1;i<n;i++)
47         for(j=i+1;j<=n;j++)
48             if(v[i]>v[j]) {aux=v[i];v[i]=v[j];v[j]=aux;}
49 }
50
51 void solutie2()
52 {
53     //incerc sa pun bete inainte de fiecare bat i cu i<=1,n
54     int i,b1,b2,b3,j; // b1-penultimul bat pus,
55                     // b2 - ultimul bat pus,
56                     // b3 - noul bat care se adauga
57     i=1;//incepand cu primul bat din stiva initiala
58     nr=n;//am cele n bete initiale
59     b1=0;//penultimul bat pus
60     b2=1;//ultimul bat pus
61     if(b1+b2+v[i]>v[i+1])
62     { //daca adaug batul de lungime 1 pot depasi al doilea in cazul 2 2 5 7
63         b1=v[i];b2=v[i+1]; //atunci incep cu cele mai mici 2 bete
64     }
65 }
66 else
67 { //altfel adaug batul de lungime 1
68     nr = nr + 1;
69     u[nr]=1;
70 }
71
72 //incerc sa pun bete inainte de batul v[i]
73 while(nr<k && i<=n)
74 { //cat timp nu am pus k bete si mai sunt bete inainte carora pot pune
75     b3=b2+b1;//incerc sa pun b3
76     if(b3+b2<=v[i])
77     { //daca suma dintre ultimul bat adaugat si nnoul bat e ok
78         nr++; //creste numarul de bete adaugate
79         u[nr]=b3;//retin noul bat
80         b1=b2;//penultimul bat
81         b2=b3;//ultimul bat
82     }
83     else

```

```

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

```

Listing 9.2.2: triunghi\_PD.cpp

```

1  // Popa Daniel Colegiul National Aurel Vlaicu Orastie
2  #include <iostream>
3  #include <fstream>
4  #include <algorithm>
5  #include <cstring>
6
7  using namespace std;
8
9  ifstream fin("triunghi.in");
10 ofstream fout("triunghi.out");
11
12 const long long ma=2000000001;
13 long long a[100], n, v, k, da=sizeof(a), de=sizeof(a[0]);
14
15 void v1()
16 {int i, x, mi=ma, a, b, c;
17   for(i=1; i<=n; i++)
18   {
19       fin>>x;
20       if(x<mi) mi=x;
21   }
22   a=b=mi;
23   fout<<a<<' '<<b;
24   for(i=3; i<=k; i++)
25   {
26       c=a+b;
27       fout<<' '<<c;
28       a=b; b=c;
29   }
30 }
31
32 void v2()
33 {long long i, x, y, z;

```

```

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

```

Listing 9.2.3: zoli\_triunghi.cpp

```

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

```

```

37     }
38     else
39     {
40         if (a[1]==a[2])
41         {
42             b[1]=b[2]=a[1];
43             ok[1]=ok[2]=1;           //in sirul b bifam toate elementele
44             pa=3;                   // ce apartin lui a
45         }
46         else
47         {
48             b[1]=b[2]=1;
49             if (a[1]==1)           // se bifeaza elementul 1 in cazul
50             {
51                 ok[1]=1;           // in care este element din a
52                 pa=2;
53             }
54             else
55                 pa=1;
56         }
57
58         pb=2;
59         while (pa<n)
60         {
61             x=b[pb]+b[pb-1];
62
63             if (b[pb]+x<=a[pa] && x<=a[pa+1]-a[pa])
64                 b[++pb]=x;
65             else
66             {
67                 b[++pb]=a[pa++];
68                 ok[pb]=1;
69             }
70         }
71
72         while (pa==n)
73         {
74             x=b[pb]+b[pb-1];
75             if(b[pb]<=a[pa]-x)
76                 b[++pb]=x;
77             else
78             {
79                 b[++pb]=a[pa++];
80                 ok[pb]=1;
81             }
82         }
83
84         while (b[pb]<=2000000000-b[pb-1])
85         {
86             b[pb+1]=b[pb]+b[pb-1];
87             pb++;
88         }
89
90         for (i=1;i<=pb;i++)
91             if (ok[i])
92                 fout<<b[i]<<" ";
93
94         x=k-n;
95         for(i=1;x && i<=pb;i++)
96             if (!ok[i])
97             {
98                 fout<<b[i]<<" ";
99                 --x;
100             }
101         fout<<"\n";
102     }
103
104     fin.close();
105     fout.close();
106
107     return 0;
108 }

```



**9.2.3 \*Rezolvare detaliată**

# Capitolul 10

## OJI 2013

### 10.1 calcule

#### Problema 1 - calcule

100 de puncte

Gigel a studiat recent șirurile cu  $n$  elemente, numere naturale. Pentru un astfel de șir  $S$ , Gigel dorește să afle răspunsul la întrebările:

- a) Care este numărul minim de *subșiruri* strict crescătoare în care se poate partiționa  $S$ ?
- b) Care este numărul de *secvențe*, modulo 20011, cu suma elementelor divizibilă cu  $k$  care se pot obține din  $S$ ?

#### Cerințe

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

#### Date de intrare

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

#### Date de ieșire

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

#### Restricții și precizări

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

#### Exemple

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

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

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

**Dimensiune** maximă a sursei: **30 KB**

### 10.1.1 Indicații de rezolvare

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

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

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

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

### 10.1.2 Cod sursă

Listing 10.1.1: calcFUcuBS.cpp

```

1  #include<fstream>
2
3  using namespace std;
4
5  int n,k,S[100002],s,nsolcresc,r[100002];
6  long long nrsolsk;
7
8  int bsearch(int x)
9  {
10     int i=0,j=nsolcresc,m;
11     while(i<j)
12         m=(i+j)/2, S[m]>=x? i=m+1:j=m;
13     return i;
14 }
15
16 int main()
17 {
18     int i,x,poz;
19
20     ifstream f("calcule.in");
21     ofstream g("calcule.out");
22
23     f>>n>>k;
24     for(i=1;i<=n;i++)
25     {
26         f>>x;
27         s=(s+x)%k;
28         r[s]=(r[s]+1)%20011;
29         poz=bsearch(x);
30         S[poz]=x;

```

```

31         if(poz==nsolcresc) nsolcresc++;
32     }
33
34     nrsolsk=r[0]*(r[0]+1)/2%20011;
35     for(i=1;i<k;i++)
36         nrsolsk=(nrsolsk+r[i]*(r[i]-1)/2)%20011;
37
38     g<<nsolcresc<<"\n"<<nrsolsk<<"\n";
39     f.close();
40     g.close();
41     return 0;
42 }

```

Listing 10.1.2: calcFUfaraBS.cpp

```

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

```

Listing 10.1.3: calcul.e.cpp

```

1  #include<fstream>
2
3  using namespace std;
4
5  #define MOD 20011
6
7  int n,k,v[100010],i,sol,x,t,r[50010],s;
8  unsigned long long sol1;
9
10 ifstream f("calcul.e.in");
11 ofstream g("calcul.e.out");
12
13 int bs(int a)
14 {
15     int s=0,d,m;

```

---

```

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

```

---

Listing 10.1.4: vcalcule.cpp

---

```

1  #include<fstream>
2
3  using namespace std;
4
5  ifstream in("calcule.in");
6  ofstream out("calcule.out");
7
8  int a[100001], r[100000];
9
10 int main()
11 { int n, i, k, x, j, s=0, sx=0;
12   r[0]=1;
13   in>>n>>k;
14   for (i=0; i<n; i++)
15   { in>>x;
16     sx=(sx+x)%k;
17     s=(s+r[sx])%20011;
18     r[sx]++;
19     for (j=x-1; a[j]==0&& j>0; j--);
20     a[j]--; a[x]++;
21   }
22   x=0;
23   for (i=1; i<20001; i++)
24       x=x+a[i];
25   out<<x<<'\\n'<<s;
26   return 0;
27 }

```

---

### 10.1.3 \*Rezolvare detaliată

## 10.2 zona

### Problema 2 - zona

100 de puncte

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

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

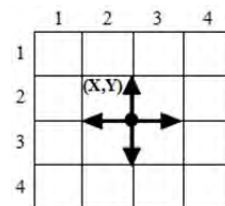


Figura 10.1: zona

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

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

### Cerințe

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

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

### Date de intrare

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

### Date de ieșire

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

### Restricții și precizări

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

### Exemple

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

2 3 7 18 2 3 3 3 4 3 4 1 1 1 1 1 2 2 2 3 3 4	16 11	<p>După cei 18 pași de la plecare ajunge în punctul situat în colțul din dreapta jos al pătratului unitate de coordonate (3,4). Ultimii 16 pași parcurși delimitează 11 pătrate unitate.</p>
--	-------	--

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

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

**Dimensiune** maximă a sursei: **30 KB**

### 10.2.1 Indicații de rezolvare

#### Soluție 1

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

Programul de rezolvare poate fi descompus în următoarele 5 etape:

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

**PAS. II.** Parcurgerea traseului până la prima poziție ce se repetă.

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

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

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

La fiecare pas al traseului:

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

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

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

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

**PAS IV.** Aplicăm următorul *algorithm de umplere*:

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

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

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

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

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

#### Soluție 2

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

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

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

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

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

#### Soluție 3

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

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

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

## 10.2.2 Cod sursă

Listing 10.2.1: Dzona.cpp

```
1 #include <fstream>
2
3 using namespace std;
4
```



```

5  ifstream fin("zona.in");
6  ofstream fout("zona.out");
7
8  int a[100][100], x,y,n,L,s,s1,Nr,cx[50000],cy[50000];
9  int dx[4]={-1, 0, 1, 0};
10 int dy[4]={ 0, 1, 0,-1};
11
12 void coordonate(int k,int x, int y, int &i, int &j){
13     if(k==1){
14         i=x-1;
15         j=y;
16     }
17     if(k==2){
18         i=x;
19         j=y+1;
20     }
21     if(k==3){
22         i=x+1;
23         j=y;
24     }
25     if(k==4){
26         i=x;
27         j=y-1;
28     }
29 }
30
31 void cit(){
32     int i,k,x1,y1,i1,j1;
33     fin>>x>>y>>n>>L;
34     x1=x;y1=y;
35     cx[0]=x;cy[0]=y;
36     a[x][y]=1;
37     for(i=1;i<=L;i++)
38     {
39         fin>>k;
40         coordonate(k,x1,y1,i1,j1);
41         cx[i]=i1;
42         cy[i]=j1;
43         if(a[i1][j1]==1) break;
44         a[i1][j1]=1;
45         x1=i1;y1=j1;
46     }
47     Nr=0;
48     for(i=0;i<L;i++)
49         if (cx[i]==i1&&cy[i]==j1)
50             break;
51         else{
52             a[cx[i]][cy[i]]=0;
53             Nr++;
54         }
55     Nr=L-Nr;
56 }
57
58 void afis(){
59     int i,j;
60     for(i=1;i<=n;i++){
61         for(j=1;j<=n;j++){
62             fout<<a[i][j]<<" ";
63             fout<<'\\n';
64         }
65     }
66
67 void fill(int x, int y){
68     int k, x1, y1;
69     s++;
70     a[x][y]=2;
71     for(k=0;k<4;k++){
72         x1=x+dx[k];
73         y1=y+dy[k];
74         if(x1>0&&y1>0&&x1<n+1&&y1<n+1&&a[x1][y1]==0)
75             fill(x1,y1);
76     }
77 }
78
79 void fill1(int x, int y){
80     int k, x1, y1;

```

```

81     sl++;
82     a[x][y]=2;
83     for(k=0;k<4;k++){
84         x1=x+dx[k];
85         y1=y+dy[k];
86         if(x1>0&&y1>0&&x1<n+1&&y1<n+1&&a[x1][y1]==1)
87             fill1(x1,y1);
88     }
89 }
90
91 int verif(int x, int y){
92     int i1,i2,j1,j2;
93     if(a[x][y]!=0) return 0;
94     i1=x;
95     while(i1>0&&a[i1][y]==0) i1--;
96     if(i1==0) return 0;
97     i2=x;
98     while(i2<=n&&a[i2][y]==0) i2++;
99     if(i2==n+1) return 0;
100    j1=y;
101    while(j1>0&&a[x][j1]==0) j1--;
102    if(j1==0) return 0;
103    j2=y;
104    while(j2<=n&&a[x][j2]==0) j2++;
105    if(j2==n+1) return 0;
106    return 1;
107 }
108
109 int main()
110 {
111     cit();
112     //afis();
113     int i,j;
114     for(i=1;i<=n;i++){
115         for(j=1;j<=n;j++){
116             if(verif(i,j)){
117                 fill(i,j);
118                 //fout<<i<<" "<<j<<"\n";
119             }
120         }
121     }
122     fill1(cx[L],cy[L]);
123     //afis();
124     fout<<Nr<<"\n";
125     fout<<s+sl/2-1;
126     fout.close();
127     return 0;
128 }

```

Listing 10.2.2: Gzona.cpp

```

1  #include<fstream>
2
3  using namespace std;
4
5  ifstream f("zona.in");
6  ofstream g("zona.out");
7
8  int v[200][200];
9  int d[2510], n,l,x,y,sol;
10 int p,a1,a2,ax,by;
11
12 void citire()
13 {
14     f>>x>>y>>n>>l;
15     //n=100;
16     for(int i=1;i<=l;++i) f>>d[i];
17     x+=55;y+=55;
18     ax=x;by=y;
19     for(int i=0;i<=n+58;++i) v[0][i]=v[n+58][i]=v[i][0]=v[i][n+58]=-1;
20 }
21
22 void fill(int a,int b)
23 {
24     if(v[a][b]==0)
25     {
26         v[a][b]=-1;

```



```
103             case 3:
104                 a0++;
105                 break;
106             case 4:
107                 b0--;
108         }
109     }
110 }
111 }
112
113 int main() {
114     citire();
115     al=merg(ax,by);
116     //afis(v);
117     sterg(ax,by);
118     fill(1,1);
119
120     for(int i=1;i<=n+55;++i)
121         for(int j=1;j<=n+55;++j)
122             if(v[i][j]==0) fill11(i,j);
123
124     for(int i=2;i<=n+55;++i)
125         for(int j=1;j<=n+55;++j)
126             if(v[i][j]>0 && v[i][j+1]>0 && v[i-1][j]>0 && v[i-1][j+1]>0)
127                 a2++;
128
129     g<<a1<<'\\n'<<a2<<'\\n';
130     g.close();
131     return 0;
132 }
```

### 10.2.3 \*Rezolvare detaliată

# Capitolul 11

## OJI 2012

### 11.1 compresie

#### Problema 1 - compresie

100 de puncte

Se consideră un text memorat într-o matrice  $M$ , definită prin coordonatele colțului stânga sus  $(x1, y1)$  și coordonatele colțului dreapta jos  $(x2, y2)$ .

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

Șirul de caractere  $C_M$  este construit prin aplicarea următoarelor reguli:

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

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

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

- matricea este împărțită în 4 submatrice  $A, B, C, D$  după cum este ilustrat în figura alăturată, unde coordonatele colțului stânga sus ale submatricei  $A$  sunt  $(x1, y1)$ , iar coordonatele colțului dreapta jos sunt  $((x2 + x1)/2, (y2 + y1)/2)$ ;

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

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

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

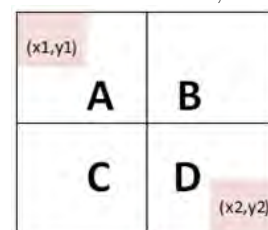


Figura 11.1: compresie

#### Cerințe

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

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

#### Date de intrare

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

#### Date de ieșire

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

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

#### Restricții și precizări

- $2 \leq N \leq 1000$
- $0 \leq nr \leq 1000000$
- $2 \leq \text{lungimea șirului compresat} \leq 1000000$
- Textul memorat inițial în matricea  $M$  conține numai caractere din mulțimea literelor mici  $\{a, \dots, z\}$ .
- Pentru rezolvarea corectă a cerinței a) se acordă 20% din punctaj, iar pentru rezolvarea corectă a ambelor cerințe se acordă tot punctajul.

**Exemple**

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

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

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

**Dimensiune** maximă a sursei: **10 KB**

**11.1.1 Indicații de rezolvare**

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

**Cerința 1**

Numărul de împărțiri care au fost necesare pentru obținerea textului compresat

-  $nr$  = numărul de stelețe '\*'

**Cerința 2**

Rezolvarea cerinței se face prin *divide et impera*.

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

Vom parcurge șirul compresat cu variabila  $k$ .

**divide:**

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

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

**stăpânește:**

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

- Dacă  $sir[k] \in \{ '0', \dots, '9' \}$  atunci se va forma numărul  $x$ , iar submatricea definită de coordonatele colțului stânga sus  $(x1, y1)$  și de coordonatele colțului dreapta jos  $(x2, y2)$  va fi completată cu caracterul din șir asociat numărului format.

Nu este necesară etapa "combină".

**11.1.2 Cod sursă**

Listing 11.1.1: compresie\_cristina\_sichim.cpp

```
1  /*
2  Solutie prof. Cristina Sichim
```

```

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

```

Listing 11.1.2: compresie\_eugen\_nodea1.cpp

```

1  /*
2     Solutie prof. Eugen Nodea
3  */
4  # include <cstdio>
5  # include <cmath>
6  # include <cstring>

```

```

7
8 using namespace std;
9
10 char s[1000001];
11 char M[1001][1001];
12 int L, N, nr, i, j, k;
13
14 void reconstruieste(short x1, short y1, short x2, short y2)
15 {
16     short mx, my, x, i, j;
17
18     //conditia de oprire
19     if (x1<=x2 && y1<=y2 && k<L)
20     {
21         //stapaneste
22         if (x1==x2 && y1==y2)
23             M[x1][y1] = s[k++];
24         else
25             if (s[k]>='0' && s[k] <='9')
26             {
27                 x = 0;
28                 while (s[k]>='0' && s[k]<='9')
29                 {
30                     x = x*10 + (s[k] - '0');
31                     ++k;
32                 }
33
34                 //completez submatricea
35                 for (i=x1; i<=x2; ++i)
36                     for (j=y1; j<=y2; ++j)
37                         M[i][j] = s[k];
38                 ++k;
39             }
40             else //s[k] == '*'
41             {
42                 //divide
43                 mx = (x2 + x1) >> 1; my = (y2 + y1) >> 1;
44                 ++k;
45                 // reconstruiesc submatricile A,B,C,D
46                 reconstruieste(x1, y1, mx, my); //A
47                 reconstruieste(x1, my + 1, mx, y2); //B
48                 reconstruieste(mx + 1, y1, x2, my); //C
49                 reconstruieste(mx + 1, my + 1, x2, y2); //D
50             }
51     }
52 }
53
54 int main()
55 {
56     freopen("compresie.in", "r", stdin);
57     freopen("compresie.out", "w", stdout);
58
59     fgets(s, 1000001, stdin);
60     L = strlen(s);
61
62     //punctul 1
63     k = 0; nr = 0; j = 0;
64     for (i=0; i < L; ++i)
65         if (s[i] == '*') ++nr;
66         else
67             if (s[i]>='0' && s[i]<='9')
68                 k = k*10 + (s[i] - '0');
69             else
70                 if (s[i]>='a' && s[i]<='z')
71                 {
72                     if (s[i-1]<'a' && k)
73                         j+=k, k=0;
74                     else
75                         ++j;
76                 }
77     N = (int) sqrt((double) j);
78     printf("%d\n", nr);
79
80     //punctul 2
81     k=0;
82     reconstruieste(1, 1, N, N);

```



```

83     for (i=1; i<=N; ++i)
84     {
85         for (j=1; j<=N; ++j)
86             printf("%c",M[i][j]);
87         printf("\n");
88     }
89     return 0;
90 }

```

Listing 11.1.3: compresie\_eugen\_nodea2.cpp

```

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

```

```

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

```

Listing 11.1.4: compresie\_silviu\_candale.cpp

```

1  /*
2      Solutie prof. Silviu Candale
3  */
4  #include <fstream>
5  #include <iostream>
6  #include <cassert>
7  #include <cstring>
8  #include <cmath>
9
10 #define NN 1000
11 // #define DEBUG
12
13 using namespace std;
14
15 ifstream fin("compresie.in");
16 ofstream fout("compresie.out");
17
18 int n;
19 char s[NN*NN+10], a[NN+2][NN+2];
20 int poz;
21 int cifra(char c)
22 {
23     return c>='0' && c<='9';
24 }
25
26 void reconstituire(int i1,int j1, int i2, int j2)
27 {
28     if(i1>i2 || j1>j2)
29         return;
30     if(s[poz] == '*')
31     {
32         int mi=(i1+i2)/2, mj=(j1+j2)/2;
33         poz++;
34         reconstituire(i1 ,j1 ,mi ,mj);
35         reconstituire(i1 , mj+1 , mi , j2);
36         reconstituire(mi+1 , j1 , i2 , mj);
37         reconstituire(mi+1 , mj+1 , i2 , j2);
38     }
39     else
40     {
41         while( cifra(s[poz]) )

```

```

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

```

### 11.1.3 \*Rezolvare detaliată

## 11.2 culori

### Problema 2 - culori

100 de puncte

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

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

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

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

**Cerințe**

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

**Date de intrare**

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

**Date de ieșire**

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

**Restricții și precizări**

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

**Exemple**

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

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

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

**Dimensiune** maximă a sursei: **10 KB**

**11.2.1 Indicații de rezolvare**

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

Notăm

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

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

Se observă că au loc următoarele relații:

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

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

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

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

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

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

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

$=_i nr[i, 1] = nr[i, 5]$  și

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

și

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

Așadar:

$nr[i, 2] = nr[i-1, 1] + nr[i-1, 3] = nr[i-2, 2] + 2 * nr[i-2, 2] = 3 * nr[i-2, 2]$  pt  $i > 2$

$nr[i, 1] = nr[i-1, 2] = 3 * nr[i-3, 2] = 3 * nr[i-2, 1]$  pt  $i > 4$

$$\begin{aligned} \text{nr}[i, 3] &= 2 * \text{nr}[i-1, 2] = 6 * \text{nr}[i-3, 2] = 3 * \text{nr}[i-2, 3] \quad \text{pt } i > 4 \\ \Rightarrow S[i] &= 2 * \text{nr}[i, 1] + 2 * \text{nr}[i, 2] + \text{nr}[i, 3] = \\ &= 6 * \text{nr}[i-2, 1] + 6 * \text{nr}[i, 2, 2] + 3 * \text{nr}[i-2, 3] = 3 * S[i-2] \end{aligned}$$

In concluzie obtinem:

$$S[1] = 5$$

$$S[2] = 8$$

$$S[3] = 14$$

$$S[2k] = 3^{(k-1)} * S[2] \quad \text{pt } k > 1$$

$$\text{si } S[2k+1] = 3^{(k-1)} * S[3] \quad \text{pt } k > 1$$

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

## 11.2.2 Cod sursă

Listing 11.2.1: culoriEM.cpp

---

```

1 // culori O(n*L) - L = lungimea rezultatului prof. Eugen Mot
2 #include <stdio.h>
3 #include <string.h>
4
5 #define C 1500
6
7 int w[2][C], b[2][C], r[2][C], g[2][C], y[2][C]; // de la white, blue, red etc.
8
9 void copy(int a[], int b[])
10 {
11     int i;
12     for(i=0; i<C; i++) b[i]=a[i];
13 }
14
15 void sum(int a[], int b[], int c[])
16 {
17     int i, t=0, r;
18     c[0]=(a[0]>=b[0]?a[0]:b[0]);
19     for(i=1; i<=c[0]; i++)
20     {
21         r=a[i]+b[i]+t;
22         c[i]=(r>9?r-10:r);
23         t=r>9;
24     }
25     if(t)
26     {
27         c[0]++;
28         c[c[0]]=1;
29     }
30 }
31
32 int main()
33 {
34     FILE *fi, *fo;
35     int i, j, k, n;
36
37     fi=fopen("culori.in", "r");
38     fscanf(fi, "%d", &n);
39     fclose(fi);
40
41     w[1][0]=b[1][0]=r[1][0]=g[1][0]=y[1][0]=1;
42     w[1][1]=b[1][1]=r[1][1]=g[1][1]=y[1][1]=1;
43
44     for(i=2; i<=n; i++)
45     {
46         j=i&1;
47         k=1-j;
48         copy(b[k], w[j]);
49         sum(w[k], r[k], b[j]);
50         sum(b[k], g[k], r[j]);
51         sum(r[k], y[k], g[j]);
52         copy(g[k], y[j]);
53     }
54
55     j=n&1;
56     k=1-j;

```

---

```

57     sum(w[j],b[j],w[k]);
58     sum(w[k],r[j],b[k]);
59     sum(b[k],g[j],r[k]);
60     sum(r[k],y[j],g[k]);
61     fo=fopen("culori.out","w");
62     for (i=g[k][0];i>0;i--)
63         fprintf(fo,"%d",g[k][i]);
64     fclose(fo);
65     return 0;
66 }

```

---

Listing 11.2.2: culori.cpp

---

```

1  // prof. Carmen Popescu - Col. Nat. Gh. Lazar Sibiu
2  #include <fstream>
3
4  using namespace std;
5
6  ifstream f("culori.in");
7  ofstream g("culori.out");
8
9  int n;
10 string s;
11
12 void s_la_p(int p)
13 {
14     string::iterator it;
15     string::reverse_iterator rit;
16
17     int i,t,c;
18
19     for (i=1;i<=p;i++)
20     {
21         t=0;
22         for ( it=s.begin() ; it<s.end(); it++)
23         {
24             c=( * it) - '0';
25             c=c*3+t;
26             t=c/10;
27             c=c%10;
28             ( * it)=c+'0';
29         }
30         if (t>0)
31             s += t+'0';
32     }
33
34     for ( rit=s.rbegin(); rit<s.rend(); rit++)
35         g<<( *rit);
36
37     g<<"\n";
38 }
39
40 int main()
41 {
42     int k;
43     f>>n;
44     if (n==1)
45         g<<"5\n";
46     else
47         if (n==2)
48             g<<"8\n";
49         else
50             if (n==3)
51                 g<<"14\n";
52             else
53             {
54                 if (n%2==0)
55                     s="8";
56                 else
57                     s="41";
58                 k=n/2;
59                 s_la_p(k-1);
60             }
61 }

```

---

Listing 11.2.3: culori1.cpp

---

```

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

```

---

Listing 11.2.4: culoriCS.cpp

---

```

1 // sursa prof. Cristina Sichim
2 # include <fstream>
3
4 # define dim 2000
5
6 using namespace std;
7
8 ifstream fi("culori.in");
9 ofstream fo("culori.out");
10
11 int w[2][dim],b[2][dim],r[2][dim],g[2][dim],y[2][dim],lv,lc,i,n;
12
13 void copie(int v1[],int v2[])
14 {
15     for(int i=0;i<=v2[0];i++)
16         v1[i]=v2[i];}
17
18 void suma(int rez[],int v1[], int v2[])
19 {
20     int i,t=0,u=v1[0];
21     if(v2[0]>u) u=v2[0];
22     for(i=1;i<=u;i++)
23         rez[i]=(v1[i]+v2[i]+t)%10, t=(v1[i]+v2[i]+t)/10;
24     if(t)
25         rez[u+1]=t, rez[0]=u+1;
26     else
27         rez[0]=u;
28 }
29
30 void afis(int v[])
31 {
32     for(int i=v[0];i>=1;i--)

```

---

```

33         fo<<v[i];
34         fo<<' \n';
35     }
36
37     int main()
38     {
39         fi>>n;
40         w[1][1]=w[1][0]=b[1][1]=b[1][0]=r[1][1]=r[1][0]
41             =g[1][1]=g[1][0]=y[1][1]=y[1][0]=1;
42         lv=1;
43         for(i=2;i<=n;i++)
44         {
45             lc=1-lv;
46             copie(w[lc],b[lv]);
47             suma(b[lc],w[lv],r[lv]); //b=w+r
48             suma(r[lc],b[lv],g[lv]); //r=b+g
49             suma(g[lc],r[lv],y[lv]); //g=r+y
50             copie(y[lc],g[lv]);
51             lv=lc;
52         }
53
54         lc=1-lv;
55         suma(w[lc],w[lv],b[lv]);
56         suma(w[lv],w[lc],g[lv]);
57         suma(w[lc],w[lv],r[lv]);
58         suma(w[lv],w[lc],y[lv]);
59
60         for(i=w[lv][0];i>=1;i--)
61             fo<<w[lv][i];
62         fo<<' \n';
63         fi.close();
64         fo.close();
65         return 0;
66     }

```

Listing 11.2.5: culoriEN.cpp

```

1  /*
2      Solutie prof. Eugen Nodea
3  */
4  #include <fstream>
5
6  using namespace std;
7
8  ifstream f ("culori.in");
9  ofstream g ("culori.out");
10
11 int n, t, k, i, j, p, y, x;
12 short a[1301];
13
14 int main()
15 {
16     f>>n;
17     if (n==1)
18     {
19         g<<5<<' \n';
20         return 0;
21     }
22
23     if (n%2)
24         p = (n-3)/2, a[1] = 4, a[2] = 1, k = 2;
25     else
26         p = (n-2)/2, a[1] = 8, k = 1;
27
28     t = 0;
29     for (i=1; i<=p; ++i)
30     {
31         for (j=1; j<=k; ++j)
32         {
33             x = a[j] * 3 + t;
34             a[j] = x%10; t=x/10;
35         }
36         while (t)
37         {
38             a[++k] = t%10;

```



```

39         t = t/10;
40     }
41 }
42
43 for (i=k; i>1; --i)
44     g<<a[i];
45 g<<a[1]<<'\\n';
46
47 return 0;
48 }

```

Listing 11.2.6: culoriLI.cpp

```

1  #include <fstream>
2
3  using namespace std;
4
5  typedef struct
6  {
7      int nrc;
8      int c[2001];
9  } NUMAR;
10
11 ifstream fi("culori.in");
12 ofstream fo("culori.out");
13
14 int n,i;
15 NUMAR W1,W2,R1,R2,B1,B2,G1,G2,Y1,Y2;
16 NUMAR rez,X;
17
18 void initializare(NUMAR &X)
19 {
20     int i;
21     for (i=1;i<=2000;i++)
22         X.c[i]=0;
23     X.c[1]=1;
24     X.nrc=1;
25 }
26
27 void suma(NUMAR A, NUMAR B, NUMAR &C)
28 {
29     int s,i,t;
30     t=0;
31     initializare(C);
32     C.c[1]=0;
33     for (i=1;i<=2000;i++)
34     {
35         s=t+A.c[i]+B.c[i];
36         C.c[i]=s%10;
37         t=s/10;
38     }
39     i=2000;
40     while (C.c[i]==0)
41         i--;
42     C.nrc=i;
43 }
44
45 void scrie(NUMAR X)
46 {
47     int i;
48     //fo<<X.nrc<<"\\n";
49     for (i=X.nrc;i>=1;i--)
50         fo<<X.c[i];
51     fo<<"\\n";
52 }
53
54 int main()
55 {
56     fi>>n;
57     initializare(W1);
58     initializare(R1);
59     initializare(B1);
60     initializare(G1);
61     initializare(Y1);
62     for (i=2;i<=n;i++)

```

```

63     {
64         W2=B1;
65         suma (G1,B1,R2);
66         suma (W1,R1,B2);
67         suma (R1,Y1,G2);
68         Y2=G1;
69
70         W1=W2;
71         R1=R2;
72         B1=B2;
73         G1=G2;
74         Y1=Y2;
75     }
76     suma (W1,R1,rez);
77     X=rez;
78     suma (X,B1,rez);
79     X=rez;
80     suma (X,G1,rez);
81     X=rez;
82     suma (X,Y1,rez);
83     scrie(rez);
84     fi.close();
85     fo.close();
86     return 0;
87 }

```

Listing 11.2.7: culoriLS.cpp

```

1  // sursa Liliana Schiopu
2  #include<fstream>
3
4  using namespace std;
5
6  ifstream f("culori.in");
7  ofstream g("culori.out");
8
9  int n,i,A[10000];
10 unsigned long long nr=1;
11
12 void mul(int A[], int B)
13 {
14     int i, t = 0;
15     for (i = 1; i <= A[0] || t; i++, t /= 10)
16         A[i] = (t += A[i] * B) % 10;
17     A[0] = i - 1;
18 }
19
20 int main()
21 {
22     f>>n;
23     if(n==1) g<<5;
24     else
25         if(n==2) g<<8;
26         else
27             if(n==3) g<<14;
28             else
29                 if(n%2==0)
30                 {
31                     A[1]=8;
32                     A[0]=1;
33                     for (i=1;i<=(n-2)/2;i++)
34                         mul(A,3);
35                     for (i=A[0];i>=1;i--)
36                         g<<A[i];
37                 }
38                 else
39                     if(n%2!=0)
40                     {
41                         A[1]=14;
42                         A[0]=1;
43                         for (i=1;i<=(n-3)/2;i++)
44                             mul(A,3);
45                         for (i=A[0];i>=1;i--)
46                             g<<A[i];
47                     }

```

```

48     f.close();
49     g.close();
50     return 0;
51 }

```

Listing 11.2.8: culoriSC.cpp

```

1  // Sura prof. Silviu Candale
2  #include <fstream>
3  #include <iostream>
4  #include <cassert>
5
6  #define BAZA 100000000
7  #define NRC 8
8
9  //pentru debug
10 // #define TOATE true
11 //end of debug
12
13 using namespace std;
14
15 ifstream fin ("culori.in");
16 ofstream fout ("culori.out");
17
18 typedef int NrMare[5000];
19
20 int n;
21
22 NrMare x;
23
24 void Atrib(NrMare x, int a)
25 {
26     x[0] = 0;
27     while(a)
28     {
29         x[++x[0]] = a % BAZA;
30         a /= BAZA;
31     }
32 }
33
34 int NrCif(int n)
35 {
36     if( n<10 )
37         return 1;
38     int r = 0 ;
39     while( n )
40         ++r, n /= 10;
41     return r;
42 }
43
44 void Produs(NrMare x, int n)
45 {
46     int t=0, tmp;
47     for(int i=1; i<=x[0]; ++i)
48     {
49         tmp = x[i]*n+t;
50         x[i] = tmp % BAZA;
51         t = tmp /BAZA;
52     }
53
54     while(t)
55         x [ ++x[0] ] = t % BAZA, t /= BAZA;
56 }
57
58 void Afis(ostream &out, NrMare x)
59 {
60     out << x[ x[0] ];
61     for(int i = x[0]-1 ; i ; --i)
62     {
63         int p = BAZA;
64         while(p/10>x[i])
65             out << 0, p /= 10;
66         out << x[i] ;
67     }
68     out << endl;

```

```
69  }
70
71  int main(){
72      fin >> n;
73      assert( 0 < n && n < 5001 );
74
75      #ifdef TOATE
76      for(int m = 1 ; m<=5000 ; ++m){
77          cout << m << endl;
78          n = m;
79      }
80      #endif
81
82      if(n==1)
83      {
84          Atrib(x,5);
85      }
86      else
87      {
88          if(n%2==1)
89              Atrib(x,14), n--;
90          else
91              Atrib(x,8);
92          n = n/2 - 1;
93          for(; n ; --n)
94              Produs(x,3);
95      }
96      Afis(fout,x);
97
98      #ifdef TOATE
99      }
100      #endif
101
102      return 0;
103 }
```

### 11.2.3 \*Rezolvare detaliată

# Capitolul 12

## OJI 2011

### 12.1 ai

#### Problema 1 - ai

100 de puncte

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

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

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

Într-unul dintre nodurile rețelei se află o țintă. În alte două noduri se află câte o sursă ce emite o rază laser. O astfel de sursă emite raza într-o singură direcție. Raza laser are o grosime neglijabilă. Cele două surse sunt astfel orientate încât raza emisă de fiecare "lovește" ținta.

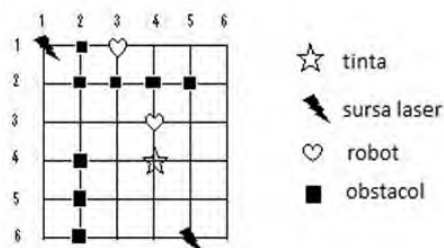
Cele două noduri în care sunt plasate sursele sunt astfel alese încât cele două raze nu se intersectează decât în nodul unde se află ținta.

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

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

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

#### Cerințe



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

### Date de intrare

Fișierul **ai.in** conține:

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

### Date de ieșire

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

### Restricții și precizări

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

### Exemple

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

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

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

**Dimensiune** maximă a sursei: **5 KB**

### 12.1.1 Indicații de rezolvare

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

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

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

De exemplu, dacă  $xsursa$  și  $xtinta$  și  $ysursa$  și  $ytinta$ , atunci o poziție de coordonate  $x$  și  $y$  este coliniară cu sursa și ținta dacă  $x = xsursa + (dx/cmmdc) * t$ ,  $y = ysursa + (dy/cmmdc) * t$ , unde  $t = 0, 1$ , ... până se "ating" coordonatele țintei.

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

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

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

Fie  $min11$  și  $min12$  duratele minime în care primul robot poate apăra ținta față de prima și respectiv a doua sursă.

Analog  $min21$  și  $min22$  pentru robotul 2.

Rezultatul este minimul dintre cele două combinații: robotul 1 parează sursa 1 și robotul 2 sursa 2, respectiv robotul 1 parează sursa 2 și robotul 2 sursa 1, adică rezultat =  $\min(\max(min11, min22), \max(min12, min21))$  unde cu  $\min$  și  $\max$  am notat funcții care dau minimul, respectiv maximul dintre două valori.

### 12.1.2 Cod sursă

Listing 12.1.1: ai.cpp

```

1  /*
2      Stelian Ciurea
3      lungimea maxima in k^2
4      Complexitate: O(n*n)
5  */
6  #include <iostream>
7  #include <fstream>
8  #include <queue>
9  #include <algorithm>    // std::sort
10
11 #define kmax 1000000
12 #define nmax 1000
13
14 using namespace std;
15
16 struct obstacol
17 {
18     int x,y;
19 };
20
21 struct nod
22 {
23     int x,y,nrpasi;
24 };
25
26 obstacol obs[kmax];
27
28 int a[nmax+2][nmax+2], opt[nmax+2][nmax+2];
29 nod poz1[nmax+2], poz2[nmax+2];
30 int n, x11, x12, y11, y12, xt, yt, xr1, xr2, yr1, yr2, x, y;
31 int i, j, k, t, nrpoz1, nrpoz2, lmax, lc, kt;
32
33 ifstream fin("ai.in");
34 ofstream fout("ai.out");
35
36 int min11, min12, min21, min22;
37 int depx[4]={0,1,0,-1};

```

```

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

```



```

114     {
115         d = ggt(abs(dx),abs(dy));
116         dx = abs(dx)/d, dy=abs(dy)/d;
117         if (x11 < xt && y11 < yt)
118             for (t=0,x=x11,y=y11;x+dx*t<xt;t++)
119                 poz1[t].x=x+dx*t, poz1[t].y=y+dy*t;
120         if (x11 > xt && y11 > yt)
121             for (t=0,x=x11,y=y11;x-dx*t>xt;t++)
122                 poz1[t].x=x-dx*t, poz1[t].y=y-dy*t;
123         if (x11 < xt && y11 > yt)
124             for (t=0,x=x11,y=y11;x+dx*t<xt;t++)
125                 poz1[t].x=x+dx*t, poz1[t].y=y-dy*t;
126         if (x11 > xt && y11 < yt)
127             for (t=0,x=x11,y=y11;x-dx*t>xt;t++)
128                 poz1[t].x=x-dx*t, poz1[t].y=y+dy*t;
129     }
130     nrpoz=t;
131 }
132
133 int main()
134 {
135     fin >> n;
136     fin >> xt >> yt >> x11 >> y11 >> x12 >> y12 >> xr1 >> yr1 >> xr2 >> yr2;
137     fin >> k;
138     for (i=1;i<=k;i++)
139     {
140         fin >> x >> y;
141         obs[i].x=x; obs[i].y=y;
142         a[x][y]=1;
143         kt++;
144     }
145     // cout << kt << endl;
146     lmax = 0;
147
148     sort(obs+1,obs+k+1,compx);
149     for (i=1;i<=k;i++)
150     {
151         //cout << obs[i].x<<" "<<obs[i].y<<" ";
152         if (obs[i].x == obs[i-1].x && obs[i].y == obs[i-1].y+1)
153             lc++;
154         else
155             lc=0;
156         if (lc > lmax)
157             lmax = lc;
158     }
159
160     sort(obs+1,obs+k+1,compy);
161     for (i=1;i<=k;i++)
162     {
163         //cout << obs[i].x<<" "<<obs[i].y<<" ";
164         if (obs[i].y == obs[i-1].y && obs[i].x == obs[i-1].x+1)
165             lc++;
166         else
167             lc=0;
168         if (lc > lmax)
169             lmax = lc;
170     }
171
172     if (lmax>=1)
173         fout << lmax+1 << '\n';
174     else
175         fout << "0\n";
176
177     calcpoz(x11,y11,poz1,nrpoz1);
178     // for (i=0;i<nrpoz1;i++)
179     //     cout << poz1[i].x<<" "<<poz1[i].y<<endl;
180     calcpoz(x12,y12,poz2,nrpoz2);
181     // for (i=0;i<nrpoz2;i++)
182     //     cout << poz2[i].x<<" "<<poz2[i].y<<endl;
183
184     for (i=0;i<=n+1;i++)
185         a[i][0] = a[0][i] = a[n+1][i] = a[i][n+1] = 1;
186     for (i=1;i<=n;i++)
187         for (j=1;j<=n;j++)
188             opt[i][j] = n*n+1;
189

```

```

190     a[xt][yt] = 1;
191     Lee(xr1, yr1);
192     min11 = min12 = n*n + 2;
193     for (i=0;i<nrpoz1;i++)
194     {
195         x = poz1[i].x; y = poz1[i].y;
196         if (min11 > opt[x][y])
197             min11 = opt[x][y];
198     }
199     for (i=0;i<nrpoz2;i++)
200     {
201         x = poz2[i].x; y = poz2[i].y;
202         if (min12 > opt[x][y])
203             min12 = opt[x][y];
204     }
205
206     cout << min11<<" " << min12 << endl;
207     /*
208     for (i=1;i<=n;i++)
209     {
210         for (j=1;j<=n;j++)
211         {
212             cout.width(4);
213             cout << opt[i][j];
214         }
215         cout << endl;
216     }
217     */
218
219     Lee(xr2, yr2);
220     min21 = min22 = n*n + 2;
221     for (i=0;i<nrpoz1;i++)
222     {
223         x = poz1[i].x; y = poz1[i].y;
224         if (min21 > opt[x][y])
225             min21 = opt[x][y];
226     }
227     for (i=0;i<nrpoz2;i++)
228     {
229         x = poz2[i].x; y = poz2[i].y;
230         if (min22 > opt[x][y])
231             min22 = opt[x][y];
232     }
233     cout << min21<<" " << min22 << endl;
234     /*
235     for (i=1;i<=n;i++)
236     {
237         for (j=1;j<=n;j++)
238         {
239             cout.width(4);
240             cout << opt[i][j];
241         }
242         cout << endl;
243     }
244     */
245     if (max(min11,min22) < max(min12,min21))
246         fout << max(min11, min22) << endl;
247     else
248         fout << max(min12, min21) << endl;
249     fin.close();
250     fout.close();
251     return 0;
252 }

```

Listing 12.1.2: ai2.cpp

```

1  /*
2      Stelian Ciurea
3      lungimea maxima in k^2
4      Complexitate: O(n*n)
5  */
6  #include <iostream>
7  #include <fstream>
8  #include <algorithm>
9  #include <queue>

```

```

10 #include <cmath>
11
12 #define kmax 1000000
13 #define nmax 1000
14
15 using namespace std;
16
17 struct obstacol
18 { int x,y;
19 };
20
21 struct nod
22 {
23     int x,y,nrpasi;
24 };
25
26 obstacol obs[kmax];
27
28 int a[nmax+2][nmax+2], opt[nmax+2][nmax+2];
29 nod poz1[nmax+2], poz2[nmax+2];
30 int n, x11, x12, y11, y12, xt, yt, xr1, xr2, yr1, yr2, x, y;
31 int i,j,k,t,nrpoz1,nrpoz2,lmax,lc,kt;
32
33 ifstream fin("ai.in");
34 ofstream fout("ai.out");
35
36 int min11, min12, min21, min22;
37 int depx[4]={0,1,0,-1};
38 int depy[4]={1,0,-1,0};
39
40 int compx(obstacol a, obstacol b)
41 {
42     if (a.x > b.x)
43         return 0;
44     if (a.x == b.x && a.y > b.y)
45         return 0;
46     return 1;
47 }
48
49 int compy(obstacol a, obstacol b)
50 {
51     if (a.y > b.y)
52         return 0;
53     if (a.y == b.y && a.x > b.x)
54         return 0;
55     return 1;
56 }
57
58 void Lee(int xst, int yst)
59 {
60     queue <int> qx,qy;
61     int xu, yu, i, xc, yc, j;
62     for (i=1;i<=n;i++)
63         for (j=1;j<=n;j++)
64             opt[i][j] = n*n+1;
65     qx.push(xst);
66     qy.push(yst);
67     opt[xst][yst] = 0;
68     while (!qx.empty())
69     {
70         xc = qx.front();
71         yc=qy.front();
72         qx.pop(); qy.pop();
73         for (i=0;i<4;i++)
74         {
75             xu = xc + depx[i];
76             yu = yc + depy[i];
77             if (a[xu][yu] == 0)
78                 if (opt[xu][yu] > opt[xc][yc] + 1)
79                 {
80                     opt[xu][yu] = opt[xc][yc] + 1;
81                     qx.push(xu);
82                     qy.push(yu);
83                 }
84         }
85     }

```

```

86 }
87
88 int ggt(int a, int b)
89 {
90     while (a!=b)
91         if (a>b) a-=b; else b-=a;
92     return a;
93 }
94
95 void calcpoz(int x11, int y11, nod poz1[], int &nrpocz)
96 {
97     int i,d,dx,dy,t=0;
98     dx = xt - x11;
99     dy = yt - y11;
100     if (dx == 0) //sunt in linie
101         if (yt < y11)
102             for (i= yt+1; i<=y11; i++)
103                 poz1[t].x = xt, poz1[t].y=i, t++;
104         else
105             for (i= y11; i<yt; i++)
106                 poz1[t].x = xt, poz1[t].y=i, t++;
107     else
108         if (dy == 0) //sunt in coloana
109             if (xt < x11)
110                 for (i= xt+1; i<=x11; i++)
111                     poz1[t].x = i, poz1[t].y=yt, t++;
112             else
113                 for (i= x11; i<xt; i++)
114                     poz1[t].x = i, poz1[t].y=yt, t++;
115         else
116         {
117             d = ggt(abs((double)dx),abs((double)dy));
118             dx = abs((double)dx)/d, dy=abs((double)dy)/d;
119             if (x11 < xt && y11 < yt)
120                 for (t=0,x=x11,y=y11;x+dx*t<xt;t++)
121                     poz1[t].x=x+dx*t, poz1[t].y=y+dy*t;
122             if (x11 > xt && y11 > yt)
123                 for (t=0,x=x11,y=y11;x-dx*t>xt;t++)
124                     poz1[t].x=x-dx*t, poz1[t].y=y-dy*t;
125             if (x11 < xt && y11 > yt)
126                 for (t=0,x=x11,y=y11;x+dx*t<xt;t++)
127                     poz1[t].x=x+dx*t, poz1[t].y=y-dy*t;
128             if (x11 > xt && y11 < yt)
129                 for (t=0,x=x11,y=y11;x-dx*t>xt;t++)
130                     poz1[t].x=x-dx*t, poz1[t].y=y+dy*t;
131         }
132     nrpocz=t;
133 }
134
135 int main()
136 {
137     fin >> n;
138     fin >> xt >> yt >> x11 >> y11 >> x12 >> y12 >> xr1 >> yr1 >> xr2 >> yr2;
139     fin >> k;
140     for (i=1;i<=k;i++)
141     {
142         fin >> x >> y;
143         obs[i].x=x;
144         obs[i].y=y;
145         a[x][y]=1;
146         kt++;
147     }
148     // fout << kt << endl;
149     lmax = 0;
150
151     sort(obs+1,obs+k+1,comp);
152     for (i=1;i<=k;i++)
153     {
154         //cout << obs[i].x<<" "<<obs[i].y<<" ";
155         if (obs[i].x == obs[i-1].x && obs[i].y == obs[i-1].y+1)
156             lc++;
157         else
158             lc=0;
159         if (lc > lmax)
160             lmax = lc;
161     }

```

```

162
163     sort(obs+1,obs+k+1,compy);
164     for (i=1;i<=k;i++)
165     {
166         //cout << obs[i].x<<" "<<obs[i].y<<" ";
167         if (obs[i].y == obs[i-1].y && obs[i].x == obs[i-1].x+1)
168             lc++;
169         else
170             lc=0;
171         if (lc > lmax)
172             lmax = lc;
173     }
174
175     if (lmax>=1)
176         fout << lmax+1 << '\n';
177     else
178         fout << "0\n";
179     calcpoz(xl1,yl1,poz1,nrpoz1);
180     // for (i=0;i<nrpoz1;i++)
181     //     cout << poz1[i].x<<" "<<poz1[i].y<<endl;
182     calcpoz(xl2,yl2,poz2,nrpoz2);
183     // for (i=0;i<nrpoz2;i++)
184     //     cout << poz2[i].x<<" "<<poz2[i].y<<endl;
185
186     for (i=0;i<=n+1;i++)
187         a[i][0] = a[0][i] = a[n+1][i] = a[i][n+1] = 1;
188
189     a[xt][yt] = 1;
190     Lee(xr1, yr1);
191     min11 = min12 = n*n + 2;
192     for (i=0;i<nrpoz1;i++)
193     {
194         x = poz1[i].x; y = poz1[i].y;
195         if (min11 > opt[x][y])
196             min11 = opt[x][y];
197     }
198     for (i=0;i<nrpoz2;i++)
199     {
200         x = poz2[i].x; y = poz2[i].y;
201         if (min12 > opt[x][y])
202             min12 = opt[x][y];
203     }
204
205     cout << min11<<" " << min12 << endl;
206     /*
207     for (i=1;i<=n;i++)
208     {
209         for (j=1;j<=n;j++)
210         {
211             cout.width(4);
212             cout << opt[i][j];
213         }
214         cout << endl;
215     }
216     */
217
218     Lee(xr2, yr2);
219     min21 = min22 = n*n + 2;
220     for (i=0;i<nrpoz1;i++)
221     {
222         x = poz1[i].x; y = poz1[i].y;
223         if (min21 > opt[x][y])
224             min21 = opt[x][y];
225     }
226     for (i=0;i<nrpoz2;i++)
227     {
228         x = poz2[i].x; y = poz2[i].y;
229         if (min22 > opt[x][y])
230             min22 = opt[x][y];
231     }
232     cout << min21<<" " << min22 << endl;
233     /*
234     for (i=1;i<=n;i++)
235     {
236         for (j=1;j<=n;j++)
237         {

```

```

238         cout.width(4);
239         cout << opt[i][j];
240     }
241     cout << endl;
242 }
243 */
244 if (max(min11,min22) < max(min12,min21))
245     fout << max(min11, min22) << endl;
246 else
247     fout << max(min12, min21) << endl;
248 fin.close();
249 fout.close();
250 return 0;
251 }

```

Listing 12.1.3: ai3.cpp

```

1  /*
2      Constantin Galatan
3      Complexitate: O(n*n)
4  */
5  #include <fstream>
6  #include <algorithm>
7  #include <queue>
8
9  using namespace std;
10
11 #define DIM 1001
12 #define pb push_back
13 #define mp make_pair
14 #define INF 0x3f3f3f3f
15
16 typedef pair<int, int> PII;
17
18 ifstream fin("ai.in");
19 ofstream fout("ai.out");
20
21 int R1[DIM][DIM];           // distantele minime ale robotului 1
22 int R2[DIM][DIM];           // distantele minime ale robotului 2
23 int t1r1 = INF, t2r1 = INF; // timpii la care robotul 1 protejeaza
24                               // tinta de cele doua surse
25 int t1r2 = INF, t2r2 = INF; // timpii la care robotul 2 protejeaza
26                               // tinta de cele doua surse
27 int n;
28 int I0, J0, is1, js1, is2, js2, // tinta, sursele
29     ir1, jr1, ir2, jr2;         // robotii
30 const int di[] = { -1, 0, 1, 0 },
31           dj[] = { 0, 1, 0, -1 };
32 bool obst[DIM][DIM];
33
34 queue<PII> Q;                 // coada - retine nodurile in ordinea vizitarii
35
36 int robot = 1;
37 int iv, jv;
38
39 void Read();
40 int LMaxZid();
41 bool OK(int, int);
42 void Lee(int, int, int[][DIM]);
43 bool AnuleazaSursa(int, int);
44
45 int main()
46 {
47     Read();
48     fout << LMaxZid() << '\n';
49     Lee(ir1, jr1, R1);
50     robot = 2;
51     Lee(ir2, jr2, R2);
52     fout << min(max(t1r1, t2r2), max(t2r1, t1r2)) << '\n';
53     fout.close();
54     return 0;
55 }
56
57 int LMaxZid()
58 {

```

```

59     int j1, Lmax = 0, L;
60     for ( int i = 1; i <= n; ++i )
61         for (int j = 1; j <= n; j++)
62             if ( obst[i][j] )
63                 {
64                     j1 = j;
65                     while ( obst[i][j] ) j++;
66                     L = j - j1;
67                     if ( L > Lmax ) Lmax = L;
68                 }
69
70     int i1;
71     for ( int j = 1; j <= n; ++j )
72         for (int i = 1; i <= n; i++)
73             if ( obst[i][j] )
74                 {
75                     i1 = i;
76                     while ( obst[i][j] ) i++;
77                     L = i - i1;
78                     if ( L > Lmax ) Lmax = L;
79                 }
80     return Lmax;
81 }
82
83 void Lee(int ir, int jr, int R[][DIM])
84 {
85     bool ok1 = false, ok2 = false;
86     R[ir][jr] = 0;
87     Q.push(mp(ir, jr));
88     int i, j;
89     while ( !Q.empty() )
90     {
91         i = Q.front().first;
92         j = Q.front().second;
93         Q.pop();
94         for ( int d = 0; d < 4; ++d )
95             {
96                 iv = i + di[d];
97                 jv = j + dj[d];
98                 if ( OK(iv, jv) && R[i][j] + 1 < R[iv][jv])
99                     {
100                         R[iv][jv] = R[i][j] + 1;
101                         Q.push(mp(iv, jv));
102
103                         // daca prima sursa e anulata
104                         if ( !ok1 && (ok1 = AnuleazaSursa(is1, js1)) )
105                             {
106                                 if ( robot == 1 )
107                                     t1r1 = R[iv][jv];
108                                 else
109                                     t1r2 = R[iv][jv];
110                             }
111
112                         // daca a doua sursa e anulata
113                         if ( !ok2 && (ok2 = AnuleazaSursa(is2, js2)) )
114                             {
115                                 if ( robot == 1 )
116                                     t2r1 = R[iv][jv];
117                                 else
118                                     t2r2 = R[iv][jv];
119                             }
120                         // daca ambele surse au fost anulate
121                         if ( ok1 && ok2 )
122                             return;
123                     }
124             }
125     }
126 }
127
128 bool AnuleazaSursa(int i, int j)
129 {
130     if ( iv == i && jv == j )         return true;    // pe sursa
131
132     if ( iv == i && i == I0 )         // pe aceeaasi orizontala sau verticala cu sursa
133     {
134         if ( j <= jv && jv < J0 )     return true;    // SRT

```

```

135         if ( J0 < jv && jv<= j ) return true;    // TRS
136     }
137
138     if ( jv == j && j == J0 )
139     {
140         if ( I0 < iv && iv <= i ) return true;    // TRS
141         if ( i <= iv && iv < I0 ) return true;    // SRT
142     }
143
144     if ( i > iv && iv > I0 && jv > j && J0 > jv &&
145         (i - I0)*(jv - j) == (i - iv) * (J0 - j) )    // / S R T
146         return true;
147
148
149     if ( I0 > iv && iv > i && jv > J0 && j > jv &&
150         (I0 - i) * (jv - J0) == (I0 - iv) * (j - J0) )    // / T R S
151         return true;
152
153     if ( I0 > iv && iv > i && J0 > jv && jv > j &&
154         (I0 - i) * (jv - j) == (J0 - j) * (iv - i) )    // \ S R T
155         return true;
156
157     if ( i > iv && iv > I0 && j > jv && jv > J0 &&
158         (j - J0) * (i - iv) == (j - jv) * (i - I0) )    // \ T R S
159         return true;
160
161     return false;
162 }
163
164 bool OK(int i, int j)
165 {
166     if ( obst[i][j] ) return false;
167     if ( i == I0 && j == J0 ) return false;
168     if ( i < 1 || i > n || j < 1 || j > n )
169         return false;
170     return true;
171 }
172
173 void Read()
174 {
175     fin >> n;
176     for ( int i = 1; i <= n; ++i )
177         for ( int j = 1; j <= n; ++j )
178             R1[i][j] = R2[i][j] = INF;
179
180     fin >> I0 >> J0;    // tinta
181     obst[I0][J0] = true;    // tinta e obstacol
182     fin >> is1 >> js1 >> is2 >> js2;    // sursele
183     fin >> ir1 >> jr1 >> ir2 >> jr2;    // robotii
184     int x, y, K;
185     fin >> K;
186     for ( int i = 0; i < K; ++i )
187     {
188         fin >> x >> y;
189         obst[x][y] = true;
190     }
191     fin.close();
192 }

```

### 12.1.3 \*Rezolvare detaliată

## 12.2 expresie

### Problema 2 - expresie

100 de puncte

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

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



Evaluarea expresiei se face după următoarele reguli:

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

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

- dacă în expresie întâlnim un  $k$ -șir delimitat de paranteze rotunde rezultatul evaluării acestui  $k$ -șir va fi reprezentat de suma maximă a unui secvență ce aparține  $k$ -șirului, unde prin secvență se înțelege o succesiune de numere aflate pe poziții consecutive în șir;

Exemplu:  $(-2, 4, -1, 3, -2, -3, 2) = 6$  secvența de sumă maximă este  $4, -1, 3$  a cărei sumă este egală cu 6.

- dacă în expresie întâlnim un  $k$ -șir delimitat de paranteze pătrate, elementele  $k$ -șirului fiind numerotate  $1, 2, \dots, k$ , rezultatul evaluării acestui  $k$ -șir va fi reprezentat de valoarea elementului aflat pe poziția  $\lceil (k+1)/2 \rceil$  dacă șirul ar fi ordonat crescător (mediana unui șir);

Exemplu:  $[-2, 9, 10, 3, 5] = 5$  șirul ordonat  $[-2, 3, 5, 9, 10] = 5$  iar valoarea expresiei este egală cu 5.

- evaluarea parantezelor se face dinspre interior spre exterior.

### Cerințe

Fiind dată o expresie aritmetică ponderată să se determine:

- câte numere întregi conține expresia aritmetică;
- care este valoarea expresiei aritmetice.

### Date de intrare

Fișierul de intrare **expresie.in** conține pe prima linie un șir de caractere ce reprezintă o expresie aritmetică ponderată.

### Date de ieșire

Fișierul de ieșire **expresie.out** va conține pe prima linie numărul de numere întregi din expresie, iar pe următoarea linie va fi scris un număr ce reprezintă valoarea expresiei aritmetice.

### Restricții și precizări

- expresia se consideră corectă
- $3 \leq$  lungimea unei expresii  $\leq 100000$
- șirul prin care se codifică expresia poate să conțină doar următoarele caractere: cifre, paranteze rotunde și pătrate deschise și închise, caracterul virgulă, caracterul minus
- pentru rezolvarea primei cerințe se obține 20% din valoarea fiecărui test
- 10% dintre teste nu vor conține paranteze
- 20% dintre teste nu vor conține paranteze imbricate

### Exemple

expresie.in	expresie.out	Explicații
2,(-4,1,-1,5)	6	Expresia conține 6 numere întregi
	7	Valoarea expresiei este: $2+5 = 7$
(3,-1,4),[2,3,1,8]	7	$6+2$
	8	
(2,-1,[1,2,3,4,5],-4,1)	9	$(2,-1,3,-4,1) = 4$
	4	

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

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

**Dimensiune** maximă a sursei: 5 KB

### 12.2.1 Indicații de rezolvare

Autor. prof. Eugen Nodea - C. N. "Tudor Vladimirescu" Tg. Jiu

**Cerința 1:**

$nr$  - numărul de numere ce se află în expresie se determină extrem de ușor acesta fiind egal cu numărul de virgule conținute de expresie +1

$$nr = nv + 1$$

**Cerința 2:**

**Soluție 1** - folosind *stive* (Eugen Nodea)

$st$  - stiva în care reținem  $k$ -sirul

$r$  - stiva în care reținem pe ce poziție se deschide o paranteză deschisă,  $kr$  vârful stivei

$d$  - stiva în care reținem pe ce poziție se deschide o paranteză dreaptă,  $kd$  vârful stivei

Algoritmul este următor:

- fie  $k$  vârful stivei  $st$

- dacă întâlnim o paranteză rotundă deschisă, reținem poziția de început a unui  $k$ -șir delimitat de paranteze rotunde în stiva  $r[+ + kr] = k + 1$

- dacă întâlnim o paranteză dreaptă deschisă, reținem poziția de început a unui  $k$ -șir delimitat de paranteze drepte în stiva  $d[+ + kd] = k + 1$

- dacă întâlnim numere le adăugăm în stivă ( $st[+ + k] = x$ )

- atunci când întâlnim o paranteză rotundă închisă, vom determina secvența de sumă maximă pentru  $k$ -șirul de valori cuprins între indicii  $r[kr], \dots, k$ , și actualizăm stivele

Algoritmul de determinare a *secvenței de sumă maximă* este clasic:

- atunci când întâlnim o paranteză dreaptă închisă, vom determina mediana  $k$ -șirului de valori cuprins între indicii  $d[kd], \dots, k$ , și actualizăm stivele

$x = d[kd]$ ;

$y = \text{quickselect}(st, x, k, (k+x)/2)$ ;

$k = x$ ;  $-kd$ ;  $st[k] = x$ ;

- Pentru aflarea valorii expresiei se va face suma elementelor din stivă

**quickselect()** - algoritmul de determinare a *mediane* unui *vector* ( $k$ -șir) este asemănător algoritmului de sortare rapidă *quicksort*. Algoritmul alege ca "pivot" o anumită poziție, în cazul nostru poziția mediană  $[(k+1)/2]$ , și încearcă să aducă pe aceea poziție valoarea din șirul ordonat. După execuția subprogramului elementele aflate la stânga medianei sunt mai mici, dar nu obligatoriu ordonate, iar elementele aflate în dreapta medianei sunt mai mari.

Complexitatea algoritmului de determinare a secvenței de sumă maximă este  $O(kmax)$

Complexitatea algoritmului de determinare a medianei este  $O(kmax)$ .

Există un algoritm mai performant decât **quickselect()**, algoritmul **binmedian()**.

Referințe:

[http://en.wikipedia.org/wiki/Selection\\_algorithm](http://en.wikipedia.org/wiki/Selection_algorithm)

**Soluția 2** (prof. Dan Pracsiu)

Descompunem expresia în atomi, prin atomi înțelegând:

- număr
- paranteză rotundă deschisă
- paranteză rotundă închisă
- paranteză pătrată deschisă
- paranteză pătrată închisă

Vom utiliza o stivă de numere întregi în care se vor depune unii din atomi: numerele (care sunt cuprinse între  $-99$  și  $99$ ), parantezele rotunde deschise (codificate prin valoarea 1000) și parantezele pătrate deschise (codificate prin valoarea 2000).

Se parcurge expresia și se completează stiva cu atomi. La identificarea unei paranteze rotunde închise, se calculează suma maximă  $Smax$  a secvenței de elemente care se extrag din vârful stivei până la întâlnirea valorii 1000. Această valoare  $Smax$  se depune în vârful stivei în locul valorii 1000. Complexitatea determinării secvenței de sumă maximă este  $O(n)$ , unde  $n$  este numărul de numere.

La identificarea în expresie a unei paranteze pătrate închise, se extrag de pe stivă toate numerele până la întâlnirea valorii 2000. Pentru aceste numere trebuie să se determine *valoarea mediană*. Pentru aceasta utilizăm o *sortare* (funcția *sort* din STL, sau *QuickSort*, sau *MergeSort*) de complexitate  $O(n \log n)$ . Valoarea mediană se depune apoi în vârful stivei, în locul valorii 2000.

La terminarea parcurgerii expresiei, în stivă vor rămâne doar unul sau mai multe numere cărora trebuie să li se determine suma. Această sumă reprezintă și rezultatul evaluării expresiei.

Complexitatea totală pentru această abordare este  $O(n \log n)$ , suficient pentru punctaj maxim.

### Soluția 3 - *recursivitate indirectă* (Constantin Gălățan)

Expresia dată poate fi formată din una sau mai multe *subexpresii*. Acestea pot fi la rândul lor: termeni simpli (numere), sume, intervale pentru care se cere valoarea mediană, intervale pentru care se cere suma, intervale pentru care se cere subsecvența de sumă maximă sau altă subexpresie.

Se definesc funcțiile Suma(), Mediana(), Secvența().

Se parcurge șirul de caractere. În funcție de valoarea primului caracter al șirului, expresia se *parsează* astfel:

a) Dacă primul caracter este '[' atunci expresia este un interval pentru care se va calcula valoarea mediană sau este o sumă al cărei prim termen este o mediană. În ambele cazuri se apelează funcția Mediana() începând cu poziția următoare din șir.

Se rețin termenii medianei într-un vector pe măsură ce aceștia sunt calculați. La întâlnirea caracterului ']', funcția Mediana() returnează valoarea medianei acestui vector. Pentru calculul medianei am folosit *algoritmul nth\_element()*, care determină în timp liniar poziția medianei.

b) Dacă primul caracter este '(', atunci se apelează funcția Secvența() începând cu poziția următoare din șir.

Aceasta reține într-un vector valorile termenilor secvenței pe măsură ce aceștia sunt calculați. La întâlnirea caracterului ')' se apelează Suma(), iar la întâlnirea unui caracter '[' se apelează Mediana().

c) Dacă primul caracter al expresiei este '-' sau o cifră, atunci se apelează Suma().

Această funcție apelează la rândul său Secvența() sau Mediana() după cum caracterul curent este '(' sau '['.

Complexitatea:  $O(n)$

## 12.2.2 Cod sursă

Listing 12.2.1: expresie\_rec.cpp

```

1  /*
2      Constantin Galatan
3      Recursivitate indirecta
4      Mediana - nth_element
5      Complexitate O(n)
6  */
7  #include <fstream>
8  #include <algorithm>
9  #include <vector>
10
11 using namespace std;
12
13 ifstream fin("expresie.in");
14 ofstream fout("expresie.out");
15
16 typedef vector<int> VI;
17 typedef VI::iterator IT;
18
19 string expr;
20 int n;
21 VI v, a;
22 int i, termen, semn, val, med;
23
24 int Expresie();
25 int Suma();
26 int Mediana(VI& v);
27 int Secventa(VI& v);
28 int SecvMax(IT a, IT b);
29 int NrNumere();
30
31 int main()
32 {
33     getline(fin, expr);
34     n = expr.length();
35
36     fout << NrNumere() << '\n';
37     fout << Expresie() << '\n';
38
39     fout.close();

```

```

40     fin.close();
41     return 0;
42 }
43
44 int NrNumere()
45 {
46     int cnt(0);
47     for ( int i = 0; i < n; ++i )
48     {
49         if ( expr[i] == '(' || expr[i] == ')' || expr[i] == ',' ||
50             expr[i] == '[' || expr[i] == ']' || expr[i] == '-' )
51             continue;
52         for (cnt++; i < n && expr[i] >='0' && expr[i] <= '9'; i++);
53     }
54     return cnt;
55 }
56
57 int Expresie()
58 {
59     int ret(0);
60     if ( expr[0] == '[' )
61     {
62         i++;
63         ret = Mediana(v);
64     }
65     else
66     if ( expr[0] == '(' )
67     {
68         i++;
69         ret = Secventa(v);
70     }
71     else
72     if ( expr[0] == '-' || (expr[0] >='0' && expr[0] <= '9') )
73         ret = Suma();
74     if ( i < n && expr[i] == ',' )
75         ret += Suma();
76     return ret;
77 }
78
79 int Mediana(VI& v)
80 {
81     if ( i == n || expr[i] == ']' )
82     {
83         i++;
84         IT it = (v.begin() + (v.size() + 1) / 2 - 1);
85         nth_element(v.begin(), it, v.end());
86         int med = *it;
87         v.clear();
88         return med;
89     }
90
91     if ( expr[i] == '[' )
92     {
93         VI a;
94         i++;
95         med = Mediana(a);
96         v.push_back(med);
97         return Mediana(v);
98     }
99     if ( expr[i] == '(' )
100     {
101         VI a;
102         i++;
103         val = Secventa(a);
104         v.push_back(val);
105         return Mediana(v);
106     }
107     if ( expr[i] == '-' )
108     {
109         i++;
110         return Mediana(v);
111     }
112     semn = 1;
113     if ( expr[i] == '-' )
114         semn = -1, i++;
115     termen = 0;

```

```

116     for ( ; i < n && expr[i] >= '0' && expr[i] <= '9'; ++i )
117         termen = termen * 10 + expr[i] - '0';
118     v.push_back(semn*termen);
119     return Mediana(v);
120 }
121
122 int Suma()
123 {
124     if ( i == n )
125         return 0;
126     if ( expr[i] == ')' )
127         return 0;
128     if ( expr[i] == ',' )
129     {
130         i++;
131         return Suma();
132     }
133     if ( expr[i] == '(' )
134     {
135         VI a;
136         i++;
137         return Secventa(a) + Suma();
138     }
139     if ( expr[i] == '[' )
140     {
141         VI a;
142         i++;
143         return Mediana(a) + Suma();
144     }
145
146     semn = 1;
147     if ( expr[i] == '-' )
148     {
149         semn = -1;
150         i++;
151     }
152     termen = 0;
153     for ( ; expr[i] >= '0' && expr[i] <= '9'; ++i )
154         termen = termen * 10 + expr[i] - '0';
155
156     return semn * termen + Suma();
157 }
158
159 int SecvMax(IT a, IT b)
160 {
161     int s = *a, max = *a;
162     for ( IT it = a + 1; it != b; ++it )
163     {
164         if ( s + *it < *it )
165             s = *it;
166         else
167             s += *it;
168         if ( s > max )
169             max = s;
170     }
171     return max;
172 }
173
174 int Secventa(VI& v)
175 {
176     if ( i == n || expr[i] == ')' )
177     {
178         i++;
179         int secv = SecvMax(v.begin(), v.end());
180         v.clear();
181         return secv;
182     }
183     if ( expr[i] == ',' )
184     {
185         i++;
186         return Secventa(v);
187     }
188     if ( expr[i] == '[' )
189     {
190         VI a;

```

```

192         i++;
193         v.push_back(Mediana(a));
194         return Secventa(v);
195     }
196     if ( expr[i] == '(' )
197     {
198         VI a;
199         i++;
200         v.push_back(Secventa(a));
201         return Secventa(v);
202     }
203
204     semn = 1;
205     if ( expr[i] == '-' )
206         semn = -1, i++;
207     termen = 0;
208     for ( ;expr[i] >= '0' && expr[i] <= '9'; ++i )
209         termen = termen * 10 + expr[i] - '0';
210     v.push_back(semn * termen);
211
212     return Secventa(v);
213 }

```

Listing 12.2.2: expresie.stive1.cpp

```

1  /*
2      Eugen Nodea
3      Implementare stive
4      Mediana - quickselect()
5      Complexitate O(n)
6  */
7  #include <fstream>
8  #include <cstring>
9  #include <algorithm>
10
11 #define nmax 100002
12 #define swap(a,b) temp=(a); (a)=(b); (b)=temp;
13
14 using namespace std;
15
16 ifstream f("expresie.in");
17 ofstream g("expresie.out");
18
19 char ex[nmax];
20 int L,i,k,x,j,kd,kr,y,rst[nmax/2],dst[nmax/2];
21 int st[nmax],sm,Max,nr=0,S,sol;
22
23 //determinare mediana - alg. asemanator quicksort
24 int quickselect(int *a, int st, int dr, int k)
25 {
26     int i,ir,j,l,mid;
27     int p,temp;
28
29     l=st; ir=dr;
30     for(;;)
31     {
32         if (ir <= l+1)
33         {
34             if (ir == l+1 && a[ir] < a[l]) { swap(a[l],a[ir]); }
35             return a[k];
36         }
37         else {
38             mid=(l+ir) >> 1;
39             swap(a[mid],a[l+1]);
40             if (a[l] > a[ir]) { swap(a[l],a[ir]); }
41             if (a[l+1] > a[ir]) { swap(a[l+1],a[ir]); }
42             if (a[l] > a[l+1]) { swap(a[l],a[l+1]); }
43
44             i=l+1; j=ir;
45             p=a[l+1];
46             for (;;)
47             {
48                 do i++; while (a[i] < p);
49                 do j--; while (a[j] > p);
50                 if (j < i) break;

```

```

51         swap(a[i],a[j]);
52     }
53     a[l+1]=a[j]; a[j]=p;
54     if (j >= k) ir=j-1;
55     if (j <= k) l=i;
56 }
57 }
58 }
59
60 int main()
61 {
62     f.getline(ex,nmax);
63     L=strlen(ex);
64     i=k=kd=kr=0;
65     while (i < L)
66     {
67         if (ex[i] == '(')
68         {
69             ++i; rst[++kr]=k+1;
70         }
71         if (ex[i] == '[')
72         {
73             ++i; dst[++kd]=k+1;
74         }
75         if ('0' <= ex[i] && ex[i] <= '9' || ex[i] == '-')
76         {
77             if (ex[i] == '-') sm=-1, ++i;
78             else sm=1;
79             x=0;
80             while ('0' <= ex[i] && ex[i] <= '9' && i < L)
81             {
82                 x=x*10+ex[i]-'0';
83                 ++i;
84             }
85             st[++k]=sm*x;
86             if (ex[i] == ',') ++i,++nr;
87         }
88         if (ex[i] == ')')
89         {
90             //determinam subsecventa de suma maxima
91             x=rst[kr];
92             S=st[x]; Max=S;
93             for (j=x+1; j<=k; ++j)
94             {
95                 if (S+st[j] < st[j]) S = st[j];
96                 else S += st[j];
97                 if (S > Max) Max=S;
98             }
99             k=x; --kr; st[k]=Max;
100             ++i;
101         }
102         if (ex[i] == ']')
103         {
104             //determinam mediana
105             x=dst[kd];
106             y=quickselect(st,x,k,(k+x)/2);
107             k=x; --kd; st[k]=y;
108             ++i;
109         }
110         if (ex[i] == ',') ++i,++nr;
111     }
112     for (i=1, sol=0; i<=k; ++i)
113     {
114         sol += st[i];
115     }
116     g<< nr+1 <<'\\n'<< sol <<'\\n';
117     return 0;
118 }

```

Listing 12.2.3: expresie\_stive2.cpp

```

1  /*
2  prof. Dan Pracsu
3  Stive
4  Mediana - sortare quicksort
5  Complexitate O(n)

```

```

6  */
7  #include<fstream>
8  #include<algorithm>
9  #include <cstring>
10
11 using namespace std ;
12
13 char s[100002] ;
14 int n, k ,v;
15 int st[100001];
16
17 int SumaMax()
18 {
19     int maxim, suma;
20     maxim = st[k] ;
21     suma = st[k] > 0 ? st[k] : 0;
22     k-- ;
23     while (st[k] != 1000)
24     {
25         suma += st[k] ;
26         if (suma > maxim) maxim = suma;
27         if (suma < 0) suma = 0 ;
28         k-- ;
29     }
30     return maxim ;
31 }
32
33 int Mediana()
34 {
35     int i, nr;
36     i = k ;
37     while (st[i] != 2000) i-- ;
38     sort(st + i + 1, st + k + 1) ;
39     nr = (k + i + 1) / 2 ;
40     k = i ;
41     return st[nr] ;
42 }
43
44 int main()
45 {
46     int ii, semn, x ;
47     ifstream fin("expresie.in") ;
48     fin >> s ;
49     n = strlen(s) ;
50     fin.close() ;
51
52     k = -1;
53     for (ii=0 ; ii < n ; )
54     {
55         if (s[ii] == '(') { st[++k] = 1000; ii++; }
56         else if (s[ii] == '[') {st[++k] = 2000 ; ii++ ;}
57         else if (s[ii] == ')')
58         {
59             x = SumaMax() ;
60             st[k] = x ;
61             ii++ ;
62         }
63         else if (s[ii] == ']')
64         {
65
66             x = Mediana() ;
67             st[k] = x ;
68             ii++ ;
69         }
70         else if (s[ii] != ',') // e un numar
71         {
72             semn = 1;
73             if (s[ii] == '-')
74                 {semn = -1; ii++;}
75             x = 0 ;
76             while (ii < n && '0' <= s[ii] && s[ii] <= '9')
77             {
78                 x = x * 10 + (s[ii] - '0') ;
79                 ii++ ;
80             }
81             x = x * semn;

```



```
82         st[++k] = x ;
83     }
84     else ii++ ,v++;
85 }
86
87 int suma = 0 ;
88 for (ii=0 ; ii<=k ; ii++)
89     suma += st[ii] ;
90
91 ofstream fout("expresie.out") ;
92 fout << v+1 << "\n";
93 fout << suma << "\n";
94 fout.close() ;
95 return 0 ;
96 }
```

---

### 12.2.3 \*Rezolvare detaliată

# Capitolul 13

## OJI 2010

### 13.1 Expoziție

#### Problema 1 - Expoziție

100 de puncte

Ilinca este o fetiță căreia îi place foarte mult să deseneze; ea a făcut multe desene pe care le-a numerotat de la 1 la  $d$  și apoi le-a multiplicat (toate copiile poartă același număr ca și originalul după care au fost făcute). În vacanță s-a hotărât să-și deschidă propria expoziție pe gardul bunicilor care are mai multe scânduri; pe fiecare scândură ea așează o planșă (un desen original sau o copie). Ilinca ține foarte mult la desenele ei și dorește ca fiecare desen să apară de cel puțin  $k$  ori (folosind originalul și copiile acestuia).

Ilinca se întreabă în câte moduri ar putea aranja expoziția. Două moduri de aranjare sunt considerate distincte dacă diferă cel puțin prin numărul unei planșe (de exemplu: 2 1 3 3 este aceeași expoziție ca și 2 3 1 3, dar este diferită de 2 1 3 1 și de 1 3 3 1).

#### Cerințe

Cunoscând  $n$  numărul de scânduri din gard,  $d$  numărul desenelor originale și  $k$  numărul minim de apariții al fiecărui desen, să se determine în câte moduri poate fi aranjată expoziția, știind că Ilinca are la dispoziție oricâte copii dorește.

#### Date de intrare

Fișierul de intrare **expozitie.in** va conține 3 numere:

$n$   $d$   $k$  - numărul de scânduri, numărul desenelor originale, respectiv numărul minim de apariții

#### Date de ieșire

Fișierul de ieșire **expozitie.out** va conține un singur număr:

$nr$  - numărul modurilor distincte de aranjare a expoziției

#### Restricții și precizări

- $n$ ,  $k$ ,  $d$  sunt numere naturale
- $1 \leq n \leq 500$
- $1 \leq d \leq 500$
- $0 \leq k \leq n$

#### Exemple

expozitie.in	expozitie.out	Explicații
3 2 1	2	Sunt 3 scânduri, 2 desene originale și fiecare desen trebuie să apară cel puțin o dată. Există 2 moduri de aranjare. Planșele ar putea fi așezate astfel: 1 2 1 1 2 2

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

**Memorie:** total **2 MB** din care pentru stivă **1 MB**

**Dimensiune** maximă a sursei: **5 KB**

### 13.1.1 Indicații de rezolvare

????? - ?????

#### Soluția 1

Deoarece fiecare desen trebuie să apară de cel puțin  $k$  ori, ne vom asigura de acest lucru afișând fiecare desen de exact  $k$  ori, au fost ocupate astfel  $k * d$  scânduri, au mai rămas libere  $r = n - k * d$ , fiecare desen mai apare pe lângă cele  $k$  apariții de  $k_1, k_2, \dots, k_d$  ori. Dacă  $r = 0$  numărul de aranjări este 1. Dacă  $r < 0$  numărul de aranjări este 0. În celelalte cazuri considerăm ecuația:

$$k_1 + k_2 + k_3 + \dots + k_d = r, 0 \leq k_i \leq r \quad (1)$$

Problema se reduce la a determina numărul de soluții ale ecuației (1), acest număr este egal cu  $C_{r+d-1}^r = C_{r+d-1}^{d-1}$ .

Pentru demonstrație se reprezintă soluția ecuației (1) ca o secvență binară, formată din  $k_1$  elemente 0, urmate de un element 1, urmat de  $k_2$  elemente 0, urmate de un element 1, ș.a.m.d., se adaugă în final  $k_d$  elemente 0. În secvența construită sunt  $r$  elemente 0 ( $k_1 + k_2 + k_3 + \dots + k_d = r$ ), numărul total de elemente este  $n + r - 1$ . Numărul posibilităților de a aranja cele  $r$  elemente 0 este  $C_{r+d-1}^r$ .

Pentru calculul combinărilor se utilizează *triunghiul lui Pascal* care se bazează pe următoarea formulă de recurență:

$$C_m^0 = C_m^m = 1 \text{ și } C_m^n = C_{m-1}^n + C_{m-1}^{n-1}$$

Fie  $m = r + d - 1$ , trebuie să calculăm  $C_m^r$ .

În triunghiul lui Pascal vom utiliza *doar 2 linii*, linia curentă ( $Lc$ ) și linia precedentă ( $Lp$ ), deoarece rezultatul poate fi un număr foarte mare, vom implementa operațiile cu numere mari. (exp\_comb.cpp)

#### Soluția 2

Problema se poate rezolva și prin *metoda programării dinamice*.

Se elimină din  $n$  cele  $k * d$  scânduri pe care se afișează câte  $k$  desene de fiecare tip. Rămâne să se afișeze planșele pe  $r = n - k * d$  scânduri.

Dacă  $r = 0$  numărul de moduri de aranjare este 1. Dacă  $r < 0$  numărul este 0. În celelalte cazuri considerăm  $a[i][j]$  = numărul modurilor de aranjare dacă avem  $i$  scânduri și  $j$  desene originale; deoarece ordinea desenelor nu contează le afișăm în *ordine crescătoare* a numărului în scris pe acestea.

Este ușor de dedus că:

$a[1][j] = j$ , pentru  $j = 1, d$  (avem o scândură și  $j$  desene)

$a[i][1] = 1$ , pentru  $i = 1, n$  (avem  $i$  scânduri și 1 desen)

$a[i][j] = a[i-1][j] + a[i][j-1]$ ,  $i = 2, n$  și  $j = 2, d$  ( $a[i][j-1]$  reprezintă toate posibilitățile de a afișa  $j-1$  desene pe  $i$  scânduri, trebuie adăugate toate posibilitățile de afișare care includ desenul  $j$  acest număr este  $a[i-1][j]$ , la cele  $i-1$  scânduri adăugăm o scândură, pe aceasta se poate afișa doar desenul  $j$ , soluțiile fără  $j$  fiind deja numărate).

Soluția se obține în  $a[r][d]$ .

Este suficient să memorăm *doar două coloane*, coloana curentă și coloana precedentă.

Deoarece rezultatul poate fi un număr foarte mare, vom implementa operațiile cu numere mari. (expo\_din.cpp)

### 13.1.2 Cod sursă

Listing 13.1.1: exp\_comb.cpp

```

1  #include<fstream>
2
3  #define LgNr 500
4
5  using namespace std;
6
7  typedef int BIG[LgNr];
8  int n,d,m,i,j,k,r,q;
9  BIG Lc[501],Lp[501];
10
11 void sum(BIG a, BIG b, BIG & s)
12 {
13     int i,cifra,t = 0,max;
14     if(a[0] < b[0])

```

```

15     {
16         max=b[0];
17         for(i=a[0]+1;i<=b[0];i++)
18             a[i] = 0;
19     }
20     else
21     {
22         max=a[0];
23         for(i=b[0]+1;i<=a[0];i++)
24             b[i]=0;
25     }
26
27     for(i=1;i<=max;i++)
28     {
29         cifra=a[i]+b[i]+t;
30         s[i]=cifra%10;
31         t=cifra/10;
32     }
33
34     if(t) s[i]=t; else i--;
35     s[0]=i;
36 }
37
38 int main()
39 {
40     ifstream fin("expozitie.in");
41     ofstream fout("expozitie.out");
42
43     fin>>n>>d>>k;
44     r=n-k*d;
45     if(r<0)
46         fout<<0;
47     else
48         if(r==0)
49             fout<<1;
50         else
51         {
52             m=r+d-1;
53             Lc[0][0]=Lc[0][1]=Lp[0][0]=Lp[0][1]=1;
54             for(i=1;i<=m;i++)
55             {
56                 for(j=1;j<=i;j++)
57                     sum(Lp[j],Lp[j-1],Lc[j]);
58
59                 for(j=0;j<=i;j++)
60                     for(q=0;q<=Lc[j][0];q++)
61                         Lp[j][q]=Lc[j][q];
62             }
63
64             for(j=Lc[r][0];j>=1;j--)
65                 fout<<Lc[r][j];
66         }
67
68     fout.close();
69     return 0;
70 }

```

Listing 13.1.2: exp\_din.cpp

```

1  #include<fstream>
2
3  #define LgNr 500
4
5  using namespace std;
6
7  typedef int BIG[LgNr];
8
9  int n,d,m,i,j,k,r,q,aux;
10 BIG Lc[501],Lp[501];
11
12 void sum(BIG a, BIG b, BIG & s)
13 {
14     int i,cifra,t = 0,max;
15     if(a[0] < b[0])
16     {

```

```

17         max=b[0];
18         for(i=a[0]+1;i<=b[0];i++)
19             a[i] = 0;
20     }
21     else
22     {
23         max=a[0];
24         for(i=b[0]+1;i<=a[0];i++)
25             b[i]=0;
26     }
27
28     for(i=1;i<=max;i++)
29     {
30         cifra=a[i]+b[i]+t;
31         s[i]=cifra%10;
32         t=cifra/10;
33     }
34
35     if(t)s[i]=t; else i--;
36     s[0]=i;
37 }
38
39 int main()
40 {
41     ifstream fin("expozitie.in");
42     ofstream fout("expozitie.out");
43
44     fin>>n>>d>>k;
45     r=n-k*d;
46     if(r<0)
47         fout<<0;
48     else
49         if(r==0)
50             fout<<1;
51         else
52         {
53             m=r+d-1;
54             for(i=1;i<=r;i++)
55                 Lp[i][0]=Lp[i][1]=1;
56             for(i=2;i<=d;i++)
57             {
58                 aux=i;
59                 Lc[1][0]=0;
60                 while(aux)
61                 {
62                     Lc[1][0]++;
63                     Lc[1][Lc[1][0]]=aux%10;
64                     aux=aux/10;
65                 }
66
67                 for(j=2;j<=r;j++)
68                     sum(Lp[j],Lc[j-1],Lc[j]);
69
70                 for(j=0;j<=r;j++)
71                     for(q=0;q<=Lc[j][0];q++)
72                         Lp[j][q]=Lc[j][q];
73             }
74
75             for(j=Lc[r][0];j>=1;j--)
76                 fout<<Lc[r][j];
77         }
78
79     fout.close();
80     return 0;
81 }

```

### 13.1.3 \*Rezolvare detaliată

## 13.2 Text

### Problema 2 - Text

100 de puncte

Ion Petre, ca oricare adolescent, este pasionat atât de jocuri, cât și de informatică. Ultimul astfel de joc este acela de a elimina dintr-un text cuvinte astfel încât fiecare cuvânt rămas să fie urmat de un cuvânt care începe cu aceeași literă cu care se termină cuvântul precedent. Face excepție de la această regulă numai ultimul cuvânt.

### Cerințe

Pentru un text dat, se cere să se afișeze numărul de cuvinte din text, apoi numărul minim de cuvinte ce pot fi eliminate astfel încât în textul rămas orice cuvânt (cu excepția ultimului) să se termine cu aceeași literă cu care începe cuvântul următor, iar în final să se afișeze cuvintele din text rămase după eliminare, fiecare cuvânt fiind afișat pe câte o linie.

### Date de intrare

Fișierul **text.in** conține un text scris pe mai multe linii. Pe fiecare linie se află cuvinte formate din litere mici ale alfabetului latin. Cuvintele sunt despărțite între ele prin exact câte un spațiu.

### Date de ieșire

Fișierul **text.out** va conține pe primele doua linii două numerele  $x$  și  $y$ , unde  $x$  va fi numărul de cuvinte din text, iar  $y$  numărul minim de cuvinte ce trebuie eliminate. Pe liniile următoare se vor afișa, în ordine, cuvintele rămase după eliminarea celor  $y$  cuvinte, câte un cuvânt pe o linie.

### Restricții și precizări

- Numărul de cuvinte din text este maximum 20000.
- Lungimea maximă a unui cuvânt este 20.
- Fiecare linie de text din fișierul de intrare are cel mult 200 de caractere.
- în fișier pot exista rânduri goale.
- Se acordă 10% din punctaj pentru determinarea corectă a numărului de cuvinte din text.
- Se acordă 40% din punctaj pentru rezolvarea corectă a primelor două cerințe.
- Pentru rezolvarea corectă a tuturor cerințelor se acordă tot punctajul.

### Exemple

text.in	text.out	Explicații
pentru ca nu are	19	Din întregul text care este format din 19 cuvinte se elimină 13 cuvinte și se obțin, în ordine, cuvintele: ion, nu, urmareste, emisiuni, interesante, evident
timp ion spune ca nu urmareste nici	13	
emisiuni interesante si evident nici altfel	ion	
de	nu	
emisiuni	urmareste	
	emisiuni	
	interesante	
	evident	

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

**Memorie:** total **2 MB** din care pentru stivă **1 MB**

**Dimensiune** maximă a sursei: **5 KB**

### 13.2.1 Indicații de rezolvare

????? - ?????

Se memorează și se numără cuvintele din text.

Se calculează pentru fiecare cuvânt lungimea maximă a unui subșir de cuvinte care îndeplinesc condiția din enunț.

Se determină valoarea maximă dintre lungimile calculate și se reconstituie soluția pe baza unui vector de legături de precedentă.

Se poate implementa o soluție  $O(n^2)$  sau  $O(n * \log n)$  sau  $O(n)$ .

Pentru implementarea soluției liniare (optime), se reține pentru fiecare literă, lungimea maximă a unui subșir care se termină cu litera respectivă și poziția ultimului cuvânt din acel subșir.

Soluția **text.pas** implementează algoritmul liniar.

## 13.2.2 Cod sursă

Listing 13.2.1: text.pas

---

```

1  const fi='text.in';fo='text.out';
2
3  var cuv:array[0..20000]of string[20];
4      l,p:array[0..20000]of integer;
5      lmax,pmax:array['a'..'z']of integer;
6      ncuv,lsol,psol:integer;
7      lit:char;
8
9  procedure citire;
10 var f:text;
11     s:string;
12     i:byte;
13 begin
14     assign(f,fi);reset(f);
15     ncuv:=0;
16     while not seekeof(f) do begin
17         readln(f,s);
18         if s='' then continue;
19         while s[1]=' ' do delete(s,1,1);
20         while s[length(s)]=' ' do delete(s,length(s),1);
21         i:=pos(' ',s);
22         while i<>0 do begin
23             inc(ncuv);
24             cuv[ncuv]:=copy(s,1,i-1);
25             delete(s,1,i);
26             i:=pos(' ',s)
27         end;
28         inc(ncuv);cuv[ncuv]:=s
29     end;
30     close(f)
31 end;
32
33 procedure dinamica;
34 var i:integer;
35 begin
36     for i:=1 to ncuv do begin
37         lit:=cuv[i][1];
38         l[i]:=lmax[lit]+1;
39         p[i]:=pmax[lit];
40         lit:=cuv[i][length(cuv[i])];
41         if l[i]>lmax[lit] then begin
42             lmax[lit]:=l[i];
43             pmax[lit]:=i
44         end
45     end;
46     lsol:=0;
47     for lit:='a' to 'z' do
48         if lmax[lit]>lsol then begin
49             lsol:=lmax[lit];
50             psol:=pmax[lit]
51         end
52 end;
53
54 procedure scriere;
55 var f:text;
56     i:integer;
57 begin
58     assign(f,fo);rewrite(f);
59     writeln(f,ncuv,' ',ncuv-lsol);
60     i:=psol;
61     while p[i]<> 0 do begin
62         l[p[i]]:=i;
63         i:=p[i];
64     end;
65     while i<>psol do begin
66         write(f,cuv[i],' ');
67         i:=l[i]
68     end;
69     writeln(f,cuv[psol]);
70     close(f)

```

---

```
71 end;  
72  
73 begin  
74     citire;  
75     dinamica;  
76     scriere  
77 end.
```

---

### 13.2.3 \*Rezolvare detaliată



# Capitolul 14

## OJI 2009

### 14.1 Insule

#### Problema 1 - Insule

100 de puncte

Arhipelagul *RGB* este format din insule care aparțin țărilor *R*, *G* și *B*. Putem reprezenta harta arhipelagului ca o matrice cu  $n$  linii și  $m$  coloane cu elemente din mulțimea  $\{0, 1, 2, 3\}$ . Un element egal cu 0 reprezintă o zonă acoperită de apă; un element egal cu 1 reprezintă o zonă de pământ aparținând unei insule din țara *R*, iar un element egal cu 2 reprezintă o zonă de pământ aparținând unei insule din țara *G*, iar un element egal cu 3 reprezintă o zonă de pământ aparținând unei insule din țara *B*.

Se consideră că două elemente ale matricei sunt *vecine* dacă ele au aceeași valoare și fie sunt consecutive pe linie, fie sunt consecutive pe coloană. Două elemente aparțin aceleiași insule dacă ele sunt vecine sau dacă se poate ajunge de la un element la celălalt pe un drum de-a lungul căruia oricare două elemente consecutive sunt vecine.

Pentru a încuraja relațiile de colaborare dintre țările *R* și *G*, se dorește construirea unui pod care să unească o insulă aparținând țării *R* de o insulă aparținând țării *G*. Podul trebuie să respecte următoarele condiții:

- să înceapă pe o zonă cu apă consecutivă pe linie sau coloană cu o zonă aparținând țării *R*;
- să se termine pe o zonă cu apă consecutivă pe linie sau coloană cu o zonă aparținând țării *G*;
- să traverseze numai zone acoperite cu apă;
- oricare două elemente consecutive ale podului trebuie să fie vecine;
- lungimea podului să fie minimă (lungimea podului este egală cu numărul de elemente traversate de pod).

#### Cerințe

Data fiind harta arhipelagului să se determine câte insule aparțin fiecărei țări, precum și lungimea minimă a unui pod care să satisfacă condițiile din enunț.

#### Date de intrare

Fișierul de intrare **insule.in** conține pe prima linie numerele naturale  $n$  și  $m$ , separate prin spațiu. Pe următoarele  $n$  linii este descrisă harta arhipelagului. Pe fiecare dintre aceste  $n$  linii sunt scrise câte  $m$  valori din mulțimea  $\{0, 1, 2, 3\}$ ; valorile nu sunt separate prin spații.

#### Date de ieșire

Fișierul de ieșire **insule.out** va conține o singură linie pe care vor fi scrise patru numere naturale separate prin spații  $NR\ NG\ NB\ Lg$ , unde  $NR$  reprezintă numărul de insule aparținând țării *R*,  $NG$  numărul de insule aparținând țării *G*,  $NB$  numărul de insule aparținând țării *B*, iar  $Lg$  lungimea minimă a podului.

#### Restricții și precizări

- $1 < n, m \leq 100$
- Se garantează că pe hartă există cel puțin un element 1, un element 2 și un element 0.
- Se acordă 40% din punctaj pentru determinarea corectă a numărului de insule din fiecare țară; se acordă punctaj integral pentru rezolvarea corectă a tuturor cerințelor.
- Începutul și sfârșitul podului pot să coincidă.
- Pentru datele de test există întotdeauna soluție.

**Exemple**

insule.in	insule.out	Explicații
6 7 1000320 0110313 3333000 2033000 2203011 2000010	4 2 3 4	țara $R$ are 4 insule, țara $G$ are 2 insule, iar țara $B$ are 3 insule. Lungimea minimă a unui pod care poate fi construit este 4; de exemplu, podul traversează celulele (6, 5), (6, 4), (6, 3), (6, 2).

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

**14.1.1 Indicații de rezolvare**

????? - ?????

I. Pentru de determina numărul de insule utilizăm un *algorithm de fill*.

II. Pentru a determina lungimea minimă a podului utilizăm un *algorithm Lee* clasic. Se folosește o *coadă* în care inițial sunt plasate elemente cu valoarea 0 care au cel puțin un vecin 1.

**14.1.2 Cod sursă**

Listing 14.1.1: insule94.cpp

```

1 //Mot Nistor-Eugen - 94 puncte
2 #include <stdio.h>
3
4 #define M 151
5
6 char a[M][M], q[2][(M/4)*M];
7 int ni[4], c;
8
9 void fill(int x, int y)
10 {if(a[x][y]==c)
11     {a[x][y]=c+3;
12     fill(x-1,y); fill(x,y-1); fill(x+1,y); fill(x,y+1);}
13 }
14 int main()
15 {FILE *fi,*fo;
16  int i,j,k,m,n,t,iq,sq,sf; char ss[M];
17  fi=fopen("insule.in","rt");
18
19  fscanf(fi,"%d %d",&m,&n);
20  for(i=1;i<=m;i++) { fscanf(fi,"%s",ss);
21      for(j=1;j<=n;j++) a[i][j]=ss[j-1]-'0';}
22  fclose(fi);
23
24  for(i=0;i<=m+1;i++) {a[i][0]=255; a[i][n+1]=255;}
25  for(j=0;j<=n+1;j++) {a[0][j]=255; a[m+1][j]=255;}
26
27  for(i=1;i<=m;i++)
28      for(j=1;j<=n;j++)
29          if(a[i][j]==1 || a[i][j]==2 || a[i][j]==3)
30              {c=a[i][j]; ni[c]++; fill(i,j);}
31
32  sq=0; t=0; sf=0;
33  for(i=1;i<=m;i++) for(j=1;j<=n;j++)
34      if(a[i][j]==4 &&
35          (a[i+1][j]==0 || a[i-1][j]==0 || a[i][j+1]==0 || a[i][j-1]==0))
36          {sq++; q[0][sq]=i; q[1][sq]=j;}
37
38  while (sf==0)
39      {iq=sq; t++;
40      for(k=1; k<=sq; k++)
41          {i=q[0][k]; j=q[1][k];
42          if(a[i][j+1]==5 || a[i][j-1]==5 || a[i+1][j]==5 || a[i-1][j]==5)
43              {sf=2; break;}
44          if(a[i+1][j]==0) {a[i+1][j]=7; iq++; q[0][iq]=i+1; q[1][iq]=j;}
45          if(a[i-1][j]==0) {a[i-1][j]=7; iq++; q[0][iq]=i-1; q[1][iq]=j;}

```

---

```

46     if(a[i][j+1]==0) {a[i][j+1]=7; iq++; q[0][iq]=i; q[1][iq]=j+1;}
47     if(a[i][j-1]==0) {a[i][j-1]=7; iq++; q[0][iq]=i; q[1][iq]=j-1;}}
48     if(iq==sq) { if(sf==0) {sf=1; t=0;}}
49     else {for(i=1; i<=iq-sq; i++) {q[0][i]=q[0][sq+i]; q[1][i]=q[1][sq+i];}
50         sq=iq-sq;}}
51
52 fo=fopen("insule.out","wt");
53 fprintf(fo,"%d %d %d %d\n",ni[1],ni[2],ni[3],t-1);
54 fclose(fo); return 1; }

```

---

Listing 14.1.2: insule.cpp

---

```

1  //Emanuela Cerchez; 100 puncte
2  #include <fstream>
3  #include <string.h>
4
5  using namespace std;
6
7  #define InFile  "insule.in"
8  #define OutFile "insule.out"
9  #define NMax 105
10
11 int n, m;
12 char s[NMax][NMax];
13 int dl[]={-1, 0, 1, 0};
14 int dc[]={ 0, 1, 0, -1};
15 int nr, ng, nb, lgpod;
16 struct Poz
17 {
18     int l, c;
19 } C[NMax*NMax];
20
21 void read();
22 void fill();
23 void pod();
24 void write();
25
26 int main()
27 {
28     read();
29     fill();
30     pod();
31     write();
32     return 0;
33 }
34
35 void read()
36 {
37     int i, j;
38     ifstream fin(InFile);
39     fin>>n>>m;
40     for (i=1; i<=n; i++)
41         for (j=1; j<=m; j++)
42             fin>>s[i][j];
43     fin.close();
44 }
45
46 void write()
47 {
48     ofstream fout(OutFile);
49     fout<<nr<<' ' <<ng<<' ' <<nb<<' ' <<lgpod<<'\n';
50     fout.close();
51 }
52
53 void sterge(int i, int j, char c)
54 {
55     int k;
56     if (s[i][j]==c)
57     {
58         s[i][j]=c+3;
59         for (k=0; k<4; k++)
60             sterge(i+dl[k], j+dc[k], c);
61     }
62 }
63

```



```

140         lgpod=d[y.l][y.c];
141     }
142 }
143 }
144 }

```

### 14.1.3 \*Rezolvare detaliată

## 14.2 Rețetă

### Problema 2 - Rețetă

100 de puncte

Mama mea este profesoară de informatică, dar îi place foarte mult să gătească. Recent am descoperit caietul ei de rețete, care arată foarte neobișnuit. Fiecare rețetă este scrisă pe un singur rând pe care sunt precizate produsele folosite, cantitățile, precum și ordinea în care se execută operațiile. De exemplu:

```
(unt 50 zahar 250 ou 4)5
```

ceea ce înseamnă că se amestecă 50 grame unt cu 250 grame zahăr și cu 4 ouă timp de 5 minute.

Pentru fiecare produs mama folosește întotdeauna aceeași unitate de măsură, așa că unitățile de măsură nu mai sunt precizate. Numele produsului este scris întotdeauna cu litere mici, iar produsele și cantitățile sunt separate prin spații (unul sau mai multe). Produsele care se amestecă împreună sunt încadrate între paranteze rotunde; după paranteza rotundă închisă este specificat timpul de preparare.

Evident, mama are și rețete mai complicate:

```
((zahar 100 ou 3)5 unt 100 nuca 200)4 (lapte 200 cacao 50 zahar 100) 3)20
```

Să traducem această rețetă: se amestecă 100 grame zahăr cu 3 ouă timp de cinci minute; apoi se adaugă 100 grame unt și 200 grame nucă, amestecând totul încă 4 minute. Se amestecă 200 ml lapte cu 50 grame de cacao și 100 grame zahăr timp de 3 minute, apoi se toarnă peste compoziția precedentă și se amestecă totul timp de 20 minute.

Observați că înainte sau după parantezele rotunde pot să apară sau nu spații.

### Cerințe

Data fiind o rețetă să se determine timpul total de preparare, precum și cantitățile necesare din fiecare produs.

### Date de intrare

Fișierul de intrare **reteta.in** conține pe prima linie un șir de caractere care reprezintă o rețetă.

### Date de ieșire

Fișierul de ieșire **reteta.out** va conține pe prima linie timpul total necesar pentru prepararea rețetei. Pe următoarele linii sunt scrise ingredientele în ordine lexicografică (ordinea din dicționar), câte un ingredient pe o linie. Pentru fiecare ingredient este specificat numele urmat de un spațiu apoi de cantitatea totală necesară.

### Restricții și precizări

- $0 < \text{Lungimea unei rețete} \leq 1000$
- $1 \leq \text{Numărul de ingrediente} \leq 100$
- Numele unui ingredient este scris cu maxim 20 litere mici ale alfabetului englez.
- Timpii de preparare sunt numere naturale  $< 100$
- Cantitățile specificate în rețete sunt numere naturale  $< 1000$
- Pentru determinarea corectă a timpului total se acordă 30% din punctajul pe test; pentru determinarea corectă a timpului total și afișarea corectă a ingredientelor (ordonate lexicografic) se acordă integral punctajul pe test.

**Exemple**

reteta.in	reteta.out
(( (zahar 100 ou 3) 5 unt 100 nuca 200) 4 (lapte 200 cacao 50 zahar 100) 3) 20	32 cacao 5 lapte 20 nuca 200 ou 3 unt 100 zahar 20

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

**14.2.1 Indicații de rezolvare**

???? - ?????

**Pasul I**

Primul pas este să determinăm timpul total de preparare. În acest scop vom însuma numere care apar imediat după o paranteză închisă

Mai exact:

cât timp mai există o paranteză închisă:

- determin numărul următor
- îl însumez cu timpul total
- elimin din șir acest număr și paranteza ) care îl precedă.

După primul pas șirul conține secvențe de tipul produs cantitate separatorii existenți fiind spații și paranteze (.

Vom defini *structura Produs* în care reținem *numele* produsului precum și *cantitatea* totală din produsul respectiv.

Vom declara un vector cu componente de tip *Produs* în care reținem produsele în ordinea alfabetică.

**Pasul II**

Cât timp mai există produse

- extragem un produs
- extragem numărul natural care urmează
- caut produsul respectiv în lista de produse (dacă nu găsim produsul respectiv, îl inserez în vectorul de produse pe poziția corectă, astfel încât vectorul să rămână sortat)
- însumez cantitatea de produs extrasă la cantitatea totală din produsul respectiv.

**14.2.2 Cod sursă**

Listing 14.2.1: RETETA.cpp

```

1 //prof. Emanuela Cerchez; 100 puncte
2 #include <fstream>
3 #include <string.h>
4 #include <stdlib.h>
5
6 using namespace std;
7
8 #define InFile "reteta.in"
9 #define OutFile "reteta.out"
10 #define LgMax 1001
11 #define NrMax 100
12 #define LgP 21
13
14 int nr;           //numarul de produse
15 int n;           //lungime reteta
16 char s[LgMax];   //reteta
17 long int timp;
18
19 struct Produs

```

```

20 {
21     char nume[LgP];
22     long int cant;
23 } P[NrMax]; //produsele in ordine alfabetica
24
25 void read();
26 long int det_timp();
27 void cantitati();
28 void write();
29 int cauta(char *);
30
31 int main()
32 {
33     read();
34     timp=det_timp();
35     cantitati();
36     write();
37     return 0;
38 }
39
40 void read()
41 {
42     int i, nr, d, j;
43     ifstream fin (InFile);
44     fin.getline(s,LgMax-1);
45     n=strlen(s);
46     fin.close();
47 }
48
49 void write()
50 {
51     int i;
52     ofstream fout (OutFile);
53     fout<<timp<<"\n";
54     for (i=0; i<nr; i++)
55         fout<<P[i].nume<<" " <<P[i].cant<<"\n";
56     fout.close();
57 }
58
59 long int det_timp()
60 // pentru a determina timpul total insumez numerele aflate
61 // dupa parantezele inchise
62 // elimin apoi aceste numere din sir
63 {
64     char *p=s, *q;
65     int a;
66     long int timp=0;
67     while (p)
68     {
69         p=strchr(s, '(');
70         if (p)
71         {
72             q=p+1;
73
74             //sar eventualele spatii
75             while (q[0]==' ') q++;
76
77             //identific numarul
78             a=q[0]-'0';
79             if (q[1]>='0' && q[1] <='9')
80             {
81                 a=a*10+q[1]-'0';
82                 q++;
83             }
84             timp+=a;
85
86             //sterg din sir caracterele de la p+1 la q inclusiv
87             *p=NULL;
88             strcat(s,q+1);
89         }
90     }
91
92     return timp;
93 }
94
95 void cantitati()

```

```

96 {
97     char *p=s;
98     int poz;
99     p=strtok(s, " ( )");
100     while (p)
101     {
102         //p indica numele unui produs
103         poz=cauta(p);
104
105         //extrag cantitatea
106         p=strtok(NULL, " ( )");
107         P[poz].cant+=atoi(p);
108
109         //extrag urmatorul produs
110         p=strtok(NULL, " ( )");
111     }
112 }
113
114 int cauta(char * x)
115 //cauta secvential produsul x in vectorul de produse
116 //daca produsul este gasit se returneaza pozitia sa
117 //daca produsul nu este gasit se insereaza in vectorul de produse,
118 //astfel incat vectorul de produse sa ramana sortat.
119 {
120     int i, j;
121     for (i=0; i<nr && strcmp(P[i].nume,x)<0; i++);
122     if (strcmp(x, P[i].nume)==0) return i;
123
124     //deplasez elementele de la pozitia i la nr-1 cu o pozitie la stanga
125     for (j=nr; j>i; j--) P[j]=P[j-1];
126     nr++;
127     strcpy(P[i].nume,x);
128     P[i].cant=0;
129     return i;
130 }

```

Listing 14.2.2: RETETAV.cpp

```

1 //prof. Ilie Vieru 100 puncte
2 #include<fstream>
3 #include<string.h>
4 #include<stdlib.h>
5
6 using namespace std;
7
8 ifstream f("reteta.in");
9 ofstream g("reteta.out");
10
11 struct nod
12 {
13     char nume[21];
14     long cant;
15 } v[300];
16
17 char s[10500], *p, *q, *r;
18 int su, cant, n;
19
20 int fcomp(const void *x, const void *y)
21 {
22     nod a=((nod*)x), b=((nod*)y);
23     return strcmp(a.nume,b.nume);
24 }
25
26 int main()
27 {
28     f.getline(s,1002);
29     strcat(s, " ");
30     while(p=strchr(s, '('))
31         if( *(p+1)=='(')
32             strcpy(p,p+1);
33         else
34             *p=' ';
35     p=s;
36     while(p=strchr(p, '('))
37     {

```



```

38     while( *(p+1)==' ')
39         strcpy(p+1,p+2);
40
41     q=strchr(p,' ');
42     r=strchr(p+1,' ');
43     if(r && r<q) q=r;
44     *q=0;
45     su+=atoi(p+1);
46     if(r==q)
47         *p=' ';
48     else
49         *p=' ';
50     strcpy(p+1,q+1);
51 }
52
53 g<<su<<"\n";
54 int i;
55 p=strtok(s+1," ");
56 while(p)
57 {
58     i=0;
59     while(i<n && strcmp(v[i].nume,p))
60         i++;
61     if(strcmp(v[i].nume,p))
62     {
63         strcpy(v[n].nume,p);
64         i=n;
65         n++;
66     }
67
68     p=strtok(NULL," ");
69     v[i].cant+=atoi(p);
70     p=strtok(NULL," ");
71 }
72
73 qsort(v,n,sizeof(nod), fcomp);
74
75 for(i=0;i<n;i++)
76     g<<v[i].nume<<" "<<v[i].cant<<"\n";
77
78 f.close();
79 g.close();
80 return 0;
81 }

```

### 14.2.3 \*Rezolvare detaliată

# Capitolul 15

## OJI 2008

### 15.1 Colaj

#### Problema 1 - Colaj

100 de puncte

La etapa finală a *Concursului pe Echipe al Micilor Artiști*, pe primul loc s-au clasat două echipe  $A$  și  $B$ , cu același punctaj. Comisia de Evaluare, pentru a le departaja, a introdus o nouă probă de baraj care vizează atât talentul copiilor, cât și istețimea lor.

Astfel, echipa  $A$  trebuie să realizeze un colaj alb-negru având la dispoziție o planșă dreptunghiulară de culoare albă și  $n$  dreptunghiuri de culoare neagră. Membrii acestei echipe vor trebui să lipească pe planșă toate dreptunghiurile, cu laturile paralele cu laturile planșei.

Pot exista și dreptunghiuri lipite în interiorul altui dreptunghi, sau dreptunghiuri care se intersectează, sau dreptunghiuri cu laturi pe laturile planșei, precum și suprafețe din planșă neacoperite cu dreptunghiuri.

După ce așează toate dreptunghiurile, membrii echipei  $A$  trebuie să comunice echipei  $B$  numărul  $n$  de dreptunghiuri negre primite, lungimea  $m$  a laturii orizontale a planșei, lungimea  $p$  a laturii verticale a planșei, și coordonatele vârfurilor din stânga-jos și dreapta-sus ale fiecărui dreptunghi de pe planșă (coordoanate referitoare la reperul cartezian  $xOy$  cu originea  $O$  în colțul din stânga-jos a planșei și cu axa de coordonate  $Ox$ , respectiv  $Oy$ , pe dreapta suport a laturii orizontale, respectiv a laturii verticale a planșei).

Pentru a câștiga concursul, echipa  $B$  trebuie să ghicească numărul zonelor continue maxime de culoare albă, conținute de colajul realizat de echipa  $A$ .

O zonă albă este considerată continuă dacă oricare ar fi două puncte  $P, Q$  din zona respectivă, se poate uni punctul  $P$  de punctul  $Q$  printr-o linie dreaptă sau frântă care să nu intersecteze interiorul nici unui dreptunghi negru. O zonă albă continuă este considerată maximală dacă nu există o altă zonă albă continuă de arie mai mare care să includă zona respectivă.

#### Cerințe

Scrieți un program care să citească numărul  $n$  al dreptunghiurilor negre primite de echipa  $A$ , lungimile  $m$  și  $p$  ale laturilor planșei, coordonatele vârfurilor din stânga-jos și dreapta-sus ale fiecărui dreptunghi negru primit, și care să determine numărul zonelor continue maxime de culoare albă existente în colajul realizat de echipa  $A$ .

#### Date de intrare

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

$n$
$m$ $p$
$a_1$ $b_1$ $c_1$ $d_1$
...
$a_n$ $b_n$ $c_n$ $d_n$

- pe prima linie, o valoare naturală  $n$ , reprezentând numărul de dreptunghiuri negre date echipei  $A$

- pe a doua linie, 2 numere naturale,  $m$  și  $p$ , separate prin spațiu, reprezentând lungimile laturilor planșei

- următoarele  $n$  linii conțin câte patru numere naturale, separate prin câte un spațiu, care reprezintă coordonatele  $(a_1, b_1)$  și  $(c_1, d_1)$  ale vârfurilor din stânga-jos și dreapta-sus ale primului dreptunghi, ..., coordonatele  $(a_n, b_n)$  și  $(c_n, d_n)$  ale vârfurilor din stânga-jos și dreapta-sus ale celui de-al  $n$ -lea dreptunghi.

**Date de ieșire**

Fișierul de ieșire **colaj.out** va conține o singură linie pe care se va scrie un singur număr natural reprezentând numărul zonelor continue maximale de culoare albă, conținute de colaj.

**Restricții și precizări**

- $1 \leq n \leq 100$ ,
- $a_1 < c_1 \leq m, a_2 < c_2 \leq m, \dots, a_n < c_n \leq m$ ,
- $b_1 < d_1 \leq p, b_2 < d_2 \leq p, \dots, b_n < d_n \leq p$
- Toate coordonatele vârfurilor dreptunghiurilor și lungimile laturilor planșei sunt numere naturale,  $0 < m, p < 8000$
- Dacă  $(x, y)$  și  $(z, t)$  sunt coordonatele a două vârfuri din două dreptunghiuri distincte, atunci:  $x \neq z$  și  $y \neq t$ .
- în 40% din teste:  $n < 30, m \leq 180, p \leq 180$ ;
- în 40% din teste:  $70 \leq n \leq 100, 180 < m < 1000, 180 < p < 1000$ ;
- în 20% din teste:  $50 < n < 80, 7000 < m < 8000, 7000 < p < 8000$

**Exemple**

colaj.in	colaj.out	Explicații
7 17 16 1 1 10 5 2 6 8 8 0 9 17 15 3 2 4 11 5 3 6 12 7 4 12 13 14 10 16 14	6	$n = 7, m = 17, p = 16$ . Sunt 7 dreptunghiuri negre. Colajul realizat de echipa A este cel din desenul alăturat. Se observă 6 zone continue maximale de culoare albă conținute de colaj (cele numerotate în figura alăturată).

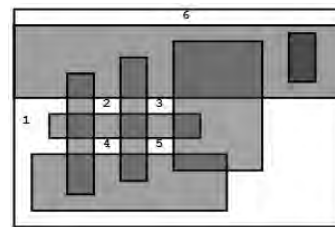


Figura 15.1: colaj

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

**15.1.1 Indicații de rezolvare**

Se consideră următorul exemplu

prof. Carmen Mincă

colaj.in	colaj.out	Explicație
5 25 15 1 1 16 5 14 4 21 12 4 7 15 8 19 6 25 9 2 10 23 14 5 2 6 11 8 3 12 13	5	Colajul realizat de echipa A este cel din desenul alăturat. Se observă 5 suprafețe albe distincte conținute de colaj.

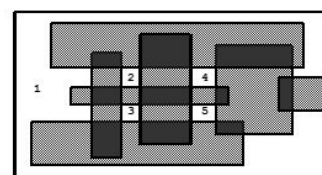


Figura 15.2: colaj

O soluție se poate obține pe baza metodei împărțirii în "zone elementare", care nu se intersectează cu nici un dreptunghi. Pentru exemplul dat, se obține următoarea împărțire în zone elementare:

Sunt reprezentate și vârfurile planșei:  $(0;0)$ ,  $(25;0)$ ,  $(0;15)$  și  $(25;15)$ . Se observă că aceste zone elementare formează o matrice  $A$  având un număr de linii, respectiv coloane, din mulțimea  $\{2n-1, 2n, 2n+1\}$ .

Numărul minim de linii  $2n-1$ , respectiv coloane:  $2n-1$ , se obține dacă există dreptunghiuri cu laturile pe marginile de jos și sus, respectiv din stânga și dreapta, ale planșei. Numărul de linii, respectiv coloane, crește cu 1 dacă niciun dreptunghi nu este situat pe marginea de jos, respectiv din stânga, a planșei.

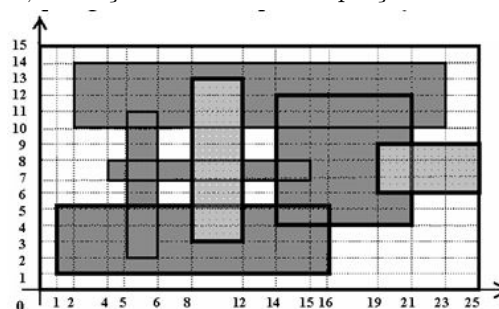


Figura 15.3: colaj

Numărul de linii, respectiv coloane, crește cu 1 dacă niciun dreptunghi nu este situat pe marginea de sus, respectiv din dreapta, a planșei.

Fie  $x_0$ , respectiv  $y_0$ , valoarea care se adaugă la  $2n - 1$  pentru a se obține numărul de coloane, respectiv linii, ale matricei  $A$ . Pentru matricea din exem

Elementul  $A[i][j]$  al matricei va avea valoarea 1 dacă și numai dacă există cel puțin un dreptunghi care să conțină zona elementară respectivă.

Matricea este inversată față de axa  $Ox$ .

Pentru exemplul dat, matricea  $A$  are  $2n + 1$  linii și  $2n$  coloane, iar conținutul ei este cel alăturat.

Pentru construcția matricei vom folosi doi vectori  $X$  și  $Y$ . Vectorul  $X$  va reține toate abscisele vârfurilor dreptunghiurilor. Vectorul  $Y$  va reține toate ordonatele vârfurilor dreptunghiurilor.

Zonele elementare din desen corespund câte unui element  $A[i][j]$  din matrice și sunt dreptunghiuri care au coordonatele vârfurilor opuse:  $(X[i], Y[j])$  și  $(X[i + 1], Y[j + 1])$   $i, j = 1, 2, \dots, 2n$ .

Dacă nu există dreptunghiuri incluse în interiorul unui alt dreptunghi, atunci vectorii au câte  $2n$  componente, fiecare având toate valorile distincte. Altfel sunt memorate coordonatele dreptunghiurilor care un sunt incluse într-un alt dreptunghi.

Se sortează crescător vectorii. Pentru exemplul dat, vectorii vor avea conținutul:

$X = (1, 2, 4, 5, 6, 8, 12, 14, 15, 16, 19, 21, 23)$

$Y = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14)$

Se memorează coordonatele vârfurilor dreptunghiurilor din exemplul dat într-un vector  $V$  cu  $n$  elemente de tip structură:

```
struct dr { int a,b,c,d; int pa,pb,pc,pd;};
```

unde:

1. (a,b) sunt coordonatele vârfului stânga-jos al dreptunghiului
2. (c,d) sunt coordonatele vârfului dreapta-sus al dreptunghiului
3. pa este poziția pe care apare a în vectorul sortat X
4. pb este poziția pe care apare b în vectorul sortat Y
5. pc este poziția pe care apare c în vectorul sortat X
6. pd este poziția pe care apare d în vectorul sortat Y

Se inițializează matricea  $A$  cu 0. Pentru fiecare dreptunghi, se marchează cu 1 toate zonele acoperite de acesta:

```
for (k=1; k<=n; k++)
    for (j=v[k].pa; j<v[k].pc; j++)
        for (i=v[k].pb; i<v[k].pd; i++) a[i+y0][j+x0]=1;
```

Se borpdează matricea cu 1, pentru a nu ieși în exteriorul ei în timpul aplicării algoritmului *FILL*. Se caută fiecare element din matrice cu valoarea 0, se umple atât elementul cât și vecinii acestuia cu valoarea 1 și se numără zonele care au valoarea 0. Acest număr va fi numărul de suprafețe albe din colaj.

După aplicarea *FILL*-ului matricea  $A$  din exemplu va avea toate valorile egale cu 1.

Sursele *colajC.cpp* și *colajP.pas* constituie implementarea a acestei soluții.

O soluție care obține un punctaj parțial constă în a contrui o matrice  $A$  cu  $p$  linii și  $m$  coloane, elementele ei memorând valori 0 și 1.

Fiecărui dreptunghi cu vârful stânga-jos, respectiv dreapta-sus, de coordonate  $(x_a, y_a)$ , respectiv  $(x_b, y_b)$ , îi corespunde în matricea  $A$  o zonă în care toate elementele au valoarea 1:

```
for (l=y_a; l<y_b; l++)
    for (c=x_a; c<x_b; c++) a[l][c]=1;
```

Pornind de la primul element din matrice cu valoarea 0, aplicând algoritmului *FILL*, umplem atât elementul cât și vecinii acestuia cu valoarea 1 și numărăm zonele care au valoarea 0. Acest număr nr va fi numărul de suprafețe albe din colaj.

Dezavantajul utilizării acestei metode constă în spațiul de memorie disponibil, insuficient pentru memorarea matricei  $A$ . Nu întotdeauna este posibilă memorarea unei matrice cu  $m \times p$  ( $m, p < 8000$ ) și cu elemente de tip *char/byte*. Prin "comprimarea" dreptunghiurilor utilizând

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	0	1	0	1	0	1	1	1	1	1	0	0
0	0	0	0	1	0	1	0	1	1	1	1	1	1	1
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	1	0	1	0	1	1	1	1	1	1	1
0	0	0	0	1	0	1	0	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 15.4: colaj

metoda împărțirii în "zone elementare", se reduce spațiul necesar memorării matricei de tip *char* la cel mult  $201 \times 201$  octeti.

Un alt dezavantaj rezultă din timpul mare de executare, datorat aplicării *algoritmului recursiv FILL* pentru o matrice cu un număr mare de componente, și a dimensiunii mici a memoriei corespunzătoare *segmentului de stivă*. (vezi sursa *colaj\_40.cpp* care obține 40 p).

O îmbunătățire a punctajului se poate obține atunci când valorile  $m$  și  $p$  permit declararea unei matrice  $A$  de tip *char* care să memoreze toate zonele corespunzătoare dreptunghiurilor, prin eliminarea anumitor linii și coloane. Necesitând un număr mare de operații datorate ștergerilor liniilor, respectiv coloanelor, identice din  $A$  se depășește timpul de execuție.

Se elimină anumite linii ale matricei  $A$  astfel: dacă liniile  $l_i, l_{i+1}, \dots, l_k$  din  $A$  sunt identice, reținem linia  $l_i$ , restul fiind șterse, nefiind necesare, realizându-se astfel o *comprimare* pe linii a matricei  $A$ , fiecare grupă de linii identice fiind înlocuită cu o singură linie de acest tip.

Analog, dacă coloanele  $c_i, c_{i+1}, \dots, c_k$  din  $A$  sunt identice, atunci se păstrează în  $A$  doar coloana  $c_i$ , restul se șterg nefiind necesare, realizând astfel o *comprimare* pe coloane a matricei  $A$ , pentru fiecare grupă de coloane identice păstrându-se în  $A$  o singură coloană de acest tip. (vezi sursa *colaj\_50.cpp* care obține 50 p).

### 15.1.2 Cod sursă

Listing 15.1.1: COLAJ\_40.cpp

---

```

1  #include <stdio.h>
2  #include <conio.h>
3  #include <iostream>
4
5  using namespace std;
6
7  int m,p,nl,nc,nr;
8  int n;
9  char a[230][230];
10 FILE *f,*g;
11
12 void citire()
13 {
14     f=fopen("colaj.in","r");
15     fscanf(f,"%d%d%d",&n,&m,&p);
16     nl=p-1;
17     nc=m-1;
18     int i,j;
19     for(i=0;i<=nl;i++)
20         for(j=0;j<=nc;j++)
21             a[i][j]=0;
22     int xa,ya,xb,yb,l,c;
23     for(i=1;i<=n;i++)
24     { fscanf(f,"%d%d%d",&xa,&ya,&xb,&yb);
25         for(l=ya;l<yb;l++)
26             for(c=xa;c<xb;c++)
27                 a[l][c]=1;
28     }
29     fclose(f);
30 }
31
32 void fill(int i,int j)
33 {
34     a[i][j]=1;
35     if((j>0)&&(a[i][j-1]==0)) fill(i,j-1);
36     if((j<nc)&&(a[i][j+1]==0)) fill(i,j+1);
37     if((i>0)&&(a[i-1][j]==0)) fill(i-1,j);
38     if((i<nl)&&(a[i+1][j]==0)) fill(i+1,j);
39 }
40
41
42 void supr()
43 { int i,j;
44     for(i=0;i<=nl;i++)
45         for(j=0;j<=nc;j++)
46             if(a[i][j]==0){nr++;fill(i,j);}
47 }
48

```

```

49 int main()
50 { citire();
51   supr();
52   g=fopen("colaj.out", "w");
53   fprintf(g, "%d", nr);
54   fclose(g);
55   return 0;
56 }

```

Listing 15.1.2: Colaj\_50.cpp

```

1  #include <stdio.h>
2
3  int m,p,nl,nc,nr;
4  int n;
5  char a[240][240];
6  FILE *f,*g;
7
8  void citire()
9  {                                     //citire date
10   f=fopen("colaj.in", "r");
11   fscanf(f, "%d%d%d", &n, &m, &p);
12   nl=p-1;
13   nc=m-1;
14   int i,j;
15   for(i=0; i<=nl; i++)
16     for(j=0; j<=nc; j++)
17       a[i][j]=0;
18   int xa,ya,xb,yb,l,c;
19   for(i=1; i<=n; i++)
20     { fscanf(f, "%d%d%d%d", &xa, &ya, &xb, &yb);
21       for(l=ya; l<yb; l++)
22         for(c=xa; c<xb; c++)
23           a[l][c]=1;
24     }
25   fclose(f);
26 }
27
28 int ver_l(int l)
29 {for(int j=0; j<=nc; j++)
30   if(a[l-1][j]!=a[l][j])return 0;
31   return 1;
32 }
33
34 void sterg_l(int l)
35 { int i,j;
36   for(i=1; i<nl; i++)
37     for(j=0; j<=nc; j++)
38       a[i][j]=a[i+1][j];
39   nl--;
40 }
41
42 void linii()
43 {
44   int i=1;
45   while(i<=nl)
46     { if(ver_l(i)) sterg_l(i);
47       else i++;}
48 }
49
50 int ver_c(int c)
51 {for(int j=0; j<=nl; j++)
52   if(a[j][c-1]!=a[j][c])return 0;
53   return 1;
54 }
55
56 void sterg_c(int c)
57 { int i,j;
58   for(j=c; j<nc; j++)
59     for(i=0; i<=nl; i++)
60       a[i][j]=a[i][j+1];
61   nc--;
62 }
63
64 void coloane()

```

---

```

65 {int j=1;
66   while(j<=nc)
67     { if(ver_c(j)) sterg_c(j);
68       else j++;}
69   }
70
71 void fill(int i,int j)
72 {
73     a[i][j]=1;
74     if((j>0)&&(a[i][j-1]==0)) fill(i,j-1);
75     if((j<nc)&&(a[i][j+1]==0)) fill(i,j+1);
76     if((i>0)&&(a[i-1][j]==0)) fill(i-1,j);
77     if((i<nl)&&(a[i+1][j]==0)) fill(i+1,j);
78 }
79
80 void supr()
81 { int i,j;
82   for(i=0;i<=nl;i++)
83     for(j=0;j<=nc;j++)
84       if(a[i][j]==0){nr++;fill(i,j);}
85 }
86
87 int main()
88 { citire();
89   linii();coloane();
90   supr();
91   g=fopen("colaj.out","w");
92   fprintf(g,"%d",nr);
93   //cout<<nr<<' ';
94   fclose(g);
95   return 0;
96 }
```

---

Listing 15.1.3: colaj\_C.cpp

---

```

1  //prof. Carmen Minca
2  #include <stdio.h>
3
4  struct dr
5  {
6      int a,b,c,d;
7      int pa,pb,pc,pd;
8  } v[115];
9  int m,p,x[200],y[200];
10 int n,k,nr,nl,nc,x0=1,y0=1;
11 char a[210][210];
12 FILE *f,*g;
13
14 int ver(int i, int xa, int xb, int xc, int xd)
15 { int j;
16   for(j=1;j<=i;j++)
17     if((v[j].a<xa)&&(v[j].b<xb)&&(v[j].c>xc)&&(v[j].d>xd)) return 0;
18   return 1;
19 }
20
21 void citire()
22 { //citire date+ crearea lui X si Y
23   f=fopen("colaj.in","r");
24   fscanf(f,"%d%d%d",&n,&m,&p);
25   int i,xa,xb,xc,xd,r=0;
26   for(i=1;i<=n;i++)
27     { fscanf(f,"%d%d%d%d",&xa,&xb,&xc,&xd);
28       if(ver(r,xa,xb,xc,xd))
29         { r++;
30           v[r].a=xa; v[r].b=xb; v[r].c=xc; v[r].d=xd;
31           x[++k]=v[r].a; y[k]=v[r].b;
32           x[++k]=v[r].c; y[k]=v[r].d;
33           if((v[r].a==0)|| (v[r].c==0))x0--; //verific daca exista dreptunghi
34           if((v[r].b==0)|| (v[r].d==0))y0--; //cu un varf pe Ox sau Oy sau in O
35           //daca nu am doar pe Oy =>inarc in matrice de la coloana 2
36           //daca nu am doar pe Ox=>inarc in matrice de la linia 2
37           //nici pe Ox nici pe Oy=> incep cu linia 2 coloana 2
38           //am varf in O, OK
39         } }
40   nr=r;
```

```

41     fclose(f);
42 }
43
44 int poz(int z[],int i,int j)
45 { int m=-1, ind, s;
46   while(i<j)
47   { ind=0;
48     if(z[i]>z[j])
49     { ind =1; s=z[i]; z[i]=z[j]; z[j]=s;}
50     if(ind)
51     {if(m==-1) i++;
52      else j--;
53      m=-m;
54     }
55     else
56     {if(m==-1) j--;
57      else i++;
58     }
59     return i;
60 }
61
62 void dv(int z[], int s, int d) //sortez prin Qsort
63 { if(d>s)
64   {int k;
65    k=poz(z,s,d);
66    dv(z,s,k-1);dv(z,k+1,d);}
67 }
68
69 int caut(int z[], int a)
70 { int i=1, j,m; //caut fiecare coordonata varf in X sau Y
71   j=k;
72   while(i<=j)
73   {m=(i+j)/2;
74    if(z[m]==a) return m;
75    else
76    {if(a<z[m]) j=m-1;
77     else i=m+1;
78    }
79   }
80   return i;
81 }
82 void pozitii() //stabilesc pozitia varfurilor dreptunghiurilor
83 {int i; //fata de zonele elemntare
84   for(i=1;i<=n;i++)
85   { v[i].pa=caut(x,v[i].a);
86     v[i].pb=caut(y,v[i].b);
87     v[i].pc=caut(x,v[i].c);
88     v[i].pd=caut(y,v[i].d);
89   }
90 }
91
92 void matrice()
93 { int i,j,k;
94   for(k=1;k<=n;k++) //formez matricea
95   for(j=v[k].pa;j<v[k].pc;j++)
96   for(i=v[k].pb;i<v[k].pd ;i++)a[i+y0][j+x0]=1;
97 }
98
99 void bordare()
100 { int i;
101   nl=y0+2*n-1;
102   nc=x0+2*n-1;
103   if(m>x[k])nc++; //determin numarul de linii si de coloane ale matricei
104   if(p>y[k])nl++;
105   for(i=0;i<=nl+1;i++) //bordez matricea cu 1
106   a[i][0]=a[i][nc+1]=1;
107   for(i=1;i<=nc+1;i++)
108   a[0][i]=a[nl+1][i]=1;
109 }
110
111 void fill(int i,int j)
112 { if(a[i][j]==0)
113   { a[i][j]=1;
114     fill(i,j-1); fill(i,j+1); fill(i-1,j); fill(i+1,j);
115   }
116 }

```



```

117
118 void supr()
119 { int i,j;
120   for(i=1;i<=nl;i++)
121     for(j=1;j<=nc;j++)
122       if(!a[i][j]){nr++;fill(i,j);}
123 }
124
125 int main()
126 { citire();
127   dv(x,1,k); dv(y,1,k); //sortez vectorii x,y cu Quicksort
128   pozitii(); //caut v[k].pa...pd
129   matrice(); //formez matricea
130   bordare();
131   supr();
132   g=fopen("colaj.out","w");
133   fprintf(g,"%d",nr);
134   //printf("\n%d",nr);
135   fclose(g);
136   return 0;
137 }

```

### 15.1.3 \*Rezolvare detaliată

## 15.2 Piața

### Problema 2 - Piața

100 de puncte

Ionuț pleacă la sfârșit de săptămână să se relaxeze într-un parc de distracții. La intrarea în parc se află o piață mare, pavată cu plăci de marmură de aceeași dimensiune. Fiecare placă are scris pe ea un singur număr dintre  $f(1)$ ,  $f(2)$ ,  $f(3)$ , ...,  $f(n)$ , unde  $f(k)$  este suma cifrelor lui  $k$ , pentru  $k$  din mulțimea  $\{1, 2, \dots, n\}$ . Piața are forma unui tablou bidimensional cu  $n$  linii și  $n$  coloane. Plăcile care alcătuiesc piața sunt așezate astfel:

- pe prima linie sunt plăci cu numerele  $f(1)$ ,  $f(2)$ , ...,  $f(n-2)$ ,  $f(n-1)$ ,  $f(n)$  (în această ordine de la stânga la dreapta);
- pe linia a doua sunt plăci cu numerele  $f(n)$ ,  $f(1)$ ,  $f(2)$ ,  $f(3)$ , ...,  $f(n-1)$ , (în această ordine de la stânga la dreapta);
- pe linia a treia sunt plăci cu numerele  $f(n-1)$ ,  $f(n)$ ,  $f(1)$ ,  $f(2)$ ,  $f(3)$ , ...,  $f(n-2)$  (în această ordine de la stânga la dreapta);
- ...
- pe ultima linie sunt plăci cu numerele  $f(2)$ , ...,  $f(n-2)$ ,  $f(n-1)$ ,  $f(n)$ ,  $f(1)$  (în această ordine de la stânga la dreapta).

Părinții lui Ionuț vor ca și în această zi, fiul lor să rezolve măcar o problemă cu sume. Astfel aceștia îi propun lui Ionuț să determine suma numerelor aflate pe porțiunea dreptunghiulară din piață având colțurile în pozițiile în care se găsesc așezați ei. Tatăl se află pe linia  $iT$  și coloana  $jT$  (colțul stânga-sus), iar mama pe linia  $iM$  și coloana  $jM$  (colțul dreapta-jos). Porțiunea din piață pentru care se dorește suma este în formă dreptunghiulară, cu laturile paralele cu marginile pieței (vezi zona plină din exemplu). Dacă Ionuț va calcula suma cerută, atunci el va fi recompensat în parcul de distracții, de către părinții lui.

### Cerințe

Determinați suma cerută de părinții lui Ionuț.

### Date de intrare

Fișierul de intrare **piata.in** conține pe prima linie numărul natural  $n$  reprezentând dimensiunea pieței. Pe linia a doua se află despărțite printr-un spațiu numerele naturale  $iT$  și  $jT$ . Pe linia a treia se află despărțite printr-un spațiu numerele naturale  $iM$  și  $jM$ .

### Date de ieșire

Fișierul de ieșire **piata.out**, va conține pe prima linie suma cerută.

**Restricții și precizări**

$$2 \leq n \leq 40000$$

$$1 \leq iT, jT, iM, jM \leq n$$

$$iT \leq iM$$

$$jT \leq jM$$

Suma cerută de părinții lui Ionuț nu depășește niciodată valoarea 2100000000.

20% din teste au  $n \leq 250$

30% din teste au  $250 \leq n \leq 10000$

30% din teste au  $10001 \leq n \leq 28000$

20% din teste au  $28001 \leq n \leq 40000$

**Exemple**

piata.in	piata.out	Explicații																																				
6 2 3 6 5	51	<div>Piața arată astfel:</div> <table><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr><tr><td>6</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>5</td><td>6</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>4</td><td>5</td><td>6</td><td>1</td><td>2</td><td>3</td></tr><tr><td>3</td><td>4</td><td>5</td><td>6</td><td>1</td><td>2</td></tr><tr><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>1</td></tr></table> <div>Suma numerelor din porțiunea cerută (marcată mai sus) este 51.</div>	1	2	3	4	5	6	6	1	2	3	4	5	5	6	1	2	3	4	4	5	6	1	2	3	3	4	5	6	1	2	2	3	4	5	6	1
1	2	3	4	5	6																																	
6	1	2	3	4	5																																	
5	6	1	2	3	4																																	
4	5	6	1	2	3																																	
3	4	5	6	1	2																																	
2	3	4	5	6	1																																	

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

**15.2.1 Indicații de rezolvare**

prof. Doru Popescu Anastasiu

Se observă că un element de pe linia  $i$ , coloana  $j$  este egal cu:

$su(j-i+1)$ , dacă  $j \geq i$   
 $su(n+j-i+1)$ , dacă  $j < i$

unde:

$su(k)$  este suma cifrelor lui  $k$ .

Dacă nu ne dăm seama de acest lucru va trebui să utilizăm un vector cu elementele de pe prima linie, după care folosind elementele lui putem accesa fiecare componentă din tablou.

Nu trebuie să construim tabloul pentru a calcula suma dorită.

O linie (începând cu a doua) din tabloul ce se definește în enunț se poate construi în funcție de precedentă.

Pentru a calcula suma cerută, trebuie să calculăm suma de pe prima linie a subtabloului (cu colțul stânga sus  $(iT, jT)$  și colțul din dreapta jos  $(iM, jM)$ ), după care suma de pe linia  $i$  ( $i > iT$ ) din subtablou este egală cu suma de pe linia  $i - 1$  din tablou, din care scădem ultimul element al acestei linii (de pe coloana  $jM$ , pentru că nu mai face parte din linia  $i$ ) și adunăm elementul de pe coloana  $jT$ , linia  $i$  (care este singur element de pe linia  $i$  ce nu se regăsește și pe linia  $i - 1$  din subtablou)

```

{suma de pe linia iT}
s:=0;
for j:=jT to jM do
  if j>=iT then s:=s+su(j-iT+1)
    else s:=s+su(n+j-iT+1);
{sumele de pe liniile iT+1, iT+2, ..., iM}
s1:=s;{suma de pe linia anterioara}
for i:=iT+1 to iM do
  begin
    {elementul de pe linia i, coloana jM}
    if jM>=i-1 then e1:=su(jM-(i-1)+1) else e1:= su(n+jM-(i-1)+1);
    {elementul de pe linia i, coloana jT}
    if jT>=i then e2:=su(jT-i+1) else e2:= su(n+jT-i+1);
  end

```

```

    s:=s+s1-e1+e2;
    s1:=s1-e1+e2;
end;

```

Se scrie în fișier *s*

## 15.2.2 Cod sursă

Listing 15.2.1: PI\_1.pas

---

```

1  program dpa_piata;
2  const fi='piata.in';
3        fo='piata.out';
4  var f:text;
5        n,iT,jT,iM,jM,i,j:longint;
6        s,e1,e2,s1:longint;
7        a:array[1..170,1..170]of integer;
8
9  function fu(k:longint):longint;
10 var s:longint;
11 begin
12   s:=0;
13   while k>0 do
14   begin
15     s:=s+(k mod 10);
16     k:=k div 10;
17   end;
18   fu:=s;
19 end;
20
21 begin
22 assign(f,fi);reset(f);
23 readln(f,n);
24 readln(f,iT,jT);
25 readln(f,iM,jM);
26 close(f);
27 {linia 1}
28 for j:=1 to n do a[1,j]:=fu(j);
29 {liniile 2, 3, ..., n}
30 for i:=2 to n do
31 begin
32   for j:=2 to n do
33     a[i,j]:=a[i-1,j-1];
34   a[i,1]:=a[i-1,n];
35 end;
36 s:=0;
37 for i:=iT to iM do
38   for j:=jT to jM do
39     s:=s+a[i,j];
40 assign(f,fo);rewrite(f);
41 writeln(f,s);
42 close(f);
43 end.

```

---

Listing 15.2.2: PI\_2.pas

---

```

1  program dpa_piata;
2  const fi='piata.in';
3        fo='piata.out';
4  var f:text;
5        n,iT,jT,iM,jM,i,j:longint;
6        s,e1,e2,s1:longint;
7        a:array[1..250,1..250]of byte;
8
9  function fu(k:longint):byte;
10 var s:longint;
11 begin
12   s:=0;
13   while k>0 do
14   begin
15     s:=s+(k mod 10);

```

---

```

16     k:=k div 10;
17     end;
18     fu:=s;
19 end;
20
21 begin
22 assign(f,fi);reset(f);
23 readln(f,n);
24 readln(f,iT,jT);
25 readln(f,iM,jM);
26 close(f);
27 {linia 1}
28 for j:=1 to n do a[1,j]:=fu(j);
29 {liniile 2, 3, ..., n}
30 for i:=2 to n do
31 begin
32   for j:=2 to n do
33     a[i,j]:=a[i-1,j-1];
34   a[i,1]:=a[i-1,n];
35 end;
36 s:=0;
37 for i:=iT to iM do
38   for j:=jT to jM do
39     s:=s+a[i,j];
40 assign(f,fo);rewrite(f);
41 writeln(f,s);
42 close(f);
43 end.

```

Listing 15.2.3: PL3.pas

```

1 program dpa_piata;
2 const fi='piata.in';
3       fo='piata.out';
4 var f:text;
5     n,iT,jT,iM,jM,i,j:longint;
6     s:longint;
7     v,v1:array[1..28002]of byte;
8
9 function fu(k:longint):byte;
10 var s:byte;
11 begin
12   s:=0;
13   while k>0 do
14     begin
15       s:=s+(k mod 10);
16       k:=k div 10;
17     end;
18   fu:=s;
19 end;
20
21 begin
22 assign(f,fi);reset(f);
23 readln(f,n);
24 readln(f,iT,jT);
25 readln(f,iM,jM);
26 close(f);
27 {linia 1}
28 s:=0;
29 for j:=1 to n do v1[j]:=fu(j);
30 if iT=1 then
31   for j:=jT to jM do s:=s+v1[j];
32 {liniile 2, 3, ..., n}
33 for i:=2 to n do
34 begin
35   for j:=2 to n do
36     v[j]:=v1[j-1];
37   v[1]:=v1[n];
38   v1:=v;
39   if (iT<=i)and(i<=iM) then
40     for j:=jT to jM do s:=s+v1[j];
41 end;
42 assign(f,fo);rewrite(f);
43 writeln(f,s);
44 close(f);

```

45 end.

---

Listing 15.2.4: PI.OK.pas

---

```
1 program dpa_piata;
2 const fi='piata.in';
3       fo='piata.out';
4 var f:text;
5     n,iT,jT,iM,jM,i,j:longint;
6     s:longint;
7     v,v1:array[1..28002] of byte;
8
9 function fu(k:longint):byte;
10 var s:byte;
11 begin
12   s:=0;
13   while k>0 do
14     begin
15       s:=s+(k mod 10);
16       k:=k div 10;
17     end;
18   fu:=s;
19 end;
20
21 begin
22 assign(f,fi);reset(f);
23 readln(f,n);
24 readln(f,iT,jT);
25 readln(f,iM,jM);
26 close(f);
27 {linia 1}
28 s:=0;
29 for j:=1 to n do v1[j]:=fu(j);
30 if iT=1 then
31   for j:=jT to jM do s:=s+v1[j];
32 {liniile 2, 3, ..., n}
33 for i:=2 to n do
34 begin
35   for j:=2 to n do
36     v[j]:=v1[j-1];
37   v[1]:=v1[n];
38   v1:=v;
39   if (iT<=i)and(i<=iM) then
40     for j:=jT to jM do s:=s+v1[j];
41 end;
42 assign(f,fo);rewrite(f);
43 writeln(f,s);
44 close(f);
45 end.
```

---

### 15.2.3 \*Rezolvare detaliată

# Capitolul 16

## OJI 2007

### 16.1 Alea

Parcul oraşului a fost neglijat mult timp, astfel că acum toate aleile sunt distruse. Prin urmare, anul acesta Primăria şi-a propus să facă reamenajări.

Parcul are forma unui pătrat cu latura de  $n$  metri şi este înconjurat de un gard care are exact două porţi. Proiectanţii de la Primărie au realizat o hartă a parcului şi au trasat pe hartă un caroiş care împarte parcul în  $n \times n$  zone pătrate cu latura de 1 metru. Astfel harta parcului are aspectul unei matrice pătratică cu  $n$  linii şi  $n$  coloane. Liniile şi respectiv coloanele sunt numerotate de la 1 la  $n$ . Elementele matricei corespund zonelor pătrate de latură 1 metru. O astfel de zonă poate să conţină un copac sau este liberă.

Edilii oraşului doresc să paveze cu un număr minim de dale pătrate cu latura de 1 metru zonele libere (fără copaci) ale parcului, astfel încât să se obţină o alea continuă de la o poartă la alta.

#### Cerinţă

Scrieţi un program care să determine numărul minim de dale necesare pentru construirea unei alei continue de la o poartă la cealaltă.

#### Date de intrare

Fişierul de intrare **alee.in** conţine pe prima linie două valori naturale  $n$  şi  $m$  separate printr-un spaţiu, reprezentând dimensiunea parcului, respectiv numărul de copaci care se găsesc în parc.

Fiecare dintre următoarele  $m$  linii conţine câte două numere naturale  $x$  şi  $y$  separate printr-un spaţiu, reprezentând poziţiile copacilor în parc ( $x$  reprezintă linia, iar  $y$  reprezintă coloana zonei în care se află copacul).

Ultima linie a fişierului conţine patru numere naturale  $x_1$   $y_1$   $x_2$   $y_2$ , separate prin câte un spaţiu, reprezentând poziţiile celor două porţi ( $x_1$ ,  $y_1$  reprezintă linia şi respectiv coloana zonei ce conţine prima poartă, iar  $x_2$ ,  $y_2$  reprezintă linia şi respectiv coloana zonei ce conţine cea de a doua poartă).

#### Date de ieşire

Fişierul de ieşire **alee.out** va conţine o singură linie pe care va fi scris un număr natural care reprezintă numărul minim de dale necesare pentru construirea aleii.

#### Restricţii şi precizări

- $1 \leq n \leq 175$
- $1 \leq m < n * n$
- Alea este continuă dacă oricare două plăci consecutive au o latură comună.
- Alea începe cu zona unde se găseşte prima poartă şi se termină cu zona unde se găseşte cea de a doua poartă.
- Poziţiile porţilor sunt distincte şi corespund unor zone libere.
- Pentru datele de test există întotdeauna soluţie.

#### Exemplu

alee.in	alee.out	Explicații
8 6 2 7 3 3 4 6 5 4 7 3 7 5 1 1 8 8	15	O modalitate de a construi aleea cu număr minim de dale este: OOO----- --OO--x- --xO----- ---OOx-- ---xO--- ----OO-- --x-xOO- -----OO (cu x am marcat copacii, cu - zonele libere, iar cu O dalele aleii).

**Timp maxim de execuție/test:** 1 secundă

### 16.1.1 Indicații de rezolvare

Este o problemă "clasică" a cărei rezolvare se bazează pe algoritmul lui Lee.

Reprezentarea informațiilor:

Vom utiliza o matrice  $A$  în care inițial vom reține valoarea  $-2$  pentru zonele libere, respectiv valoarea  $-1$  pentru zonele în care se află un copac.

Ulterior, pe măsură ce explorăm matricea în scopul determinării unei alei de lungime minimă vom reține în matrice pentru pozițiile explorate un număr natural care reprezintă distanța minimă de la poziția primei porți la poziția respectivă (exprimată în numărul de dale necesare pentru construirea aleii).

Vom parcurge matricea începând cu poziția primei porți.

Pozițiile din matrice care au fost explorate le vom reține într-o coadă, în ordinea explorării lor.

La fiecare pas vom extrage din coadă o poziție  $(x, y)$ , vom explora toți cei 4 vecini liberi neexplorați ai poziției extrase și îi vom introduce în coadă. Când explorăm un vecin, reținem în matricea  $A$  pe poziția sa distanța minimă  $(1 + A[x][y])$ .

Procedeul se repetă cât timp coada nu este vidă și poziția în care se află cea de a doua poartă nu a fost explorată.

Rezultatul se va obține în matricea  $A$ , pe poziția celei de a doua porți.

Alocarea memoriei pentru coadă ar putea reprezenta o problemă (indicele de linie și indicele de coloană trebuie să fie memorati pe 1 byte, deoarece coada poate avea maxim  $175 * 175$  elemente).

O soluție de a "economisi" spațiul de memorie este de a utiliza coada în mod circular.

Problema poate fi abordată și recursiv, dar datorită adâncimii recursiei o astfel de abordare obține 80 de puncte.

### 16.1.2 Cod sursă

Listing 16.1.1: alee.c

```

1  /* prof. Emanuela Cerchez, Liceul de Informatica "Grigore Moisil" Iasi */
2  #include <stdio.h>
3  #define DimMaxParc 177
4  #define DimMaxCoadă 30625
5
6  //deplasările pe linie si coloana pe direcțiile N,E,S,V
7  const int dx[5] = {0, -1, 0, 1, 0};
8  const int dy[5] = {0, 0, 1, 0, -1};
9
10 typedef struct
11 {
12     unsigned char l, c; //poziția în Parc
13 }Element;
14
15 typedef Element Coadă[DimMaxCoadă];
16
17 int A[DimMaxParc][DimMaxParc];
18 /**
19 A[i][j]=-1 dacă în poziția i,j se afla un copac

```

```

20 A[i][j]=-2 daca zona i,j este libera
21 A[i][j]=d, daca zona i,j este libera situata la distanta d de prima poarta
22 */
23
24 int ix, iy, ox, oy;
25 int n, m;
26 int NrDale;
27
28 void Citire(void)
29 {
30     int k, i, j;
31
32     FILE * Fin=freopen("alee.in", "r", stdin);
33
34     scanf("%d%d", &n, &m); //citesc dimensiunile parcului si numarul de pomi
35     //marchez cu -2 pozitiile libere
36     for (i=1; i<=n; i++)
37         for (j=1; j<=m; j++) A[i][j] = -2;
38     for (k=1; k<=m; k++)
39     {
40         scanf("%d %d", &i, &j); //citesc coordonatele obstacolelor
41         A[i][j] = -1;           //marchez obstacolele cu -1
42     }
43     //citesc pozitiile portii de intrare si de iesire
44     scanf("%d %d %d %d", &ix, &iy, &ox, &oy);
45     fclose(Fin);
46 }
47
48 void Bordare(void)
49 {
50     //bordez parcul cu pomi}
51     int i, j;
52     for (i=1; i<=n; i++)
53         {A[i][0] = -1;A[i][n+1] = -1;
54          A[0][i] = -1;A[n+1][i] = -1;}
55 }
56
57 void DeterminNrDale(void)
58 {
59     Coadă C;
60     int IncC, SfC, k;    //inceputul si sfarsitul cozii
61     Element x, y;
62
63     //initializez coada
64     x.l = ix; x.c = iy; A[ix][iy] = 1;
65     //pun pozitia initiala, poarta de intrare, in coada}
66     IncC = SfC = 0; C[IncC] = x;
67     //parcurg parcul
68     while (IncC <=SfC && A[ox][oy]==-2)
69     {
70         //extrag un element din coada
71         x = C[IncC++];
72         //ma deplasez in cele patru directii posibile
73         for (k=1; k<=4; k++)
74         {
75             y.l = x.l + dx[k]; y.c = x.c + dy[k];
76             //y - urmatoarea pozitie in directia k
77             if (A[y.l][y.c]==-2)
78                 //y- pozitie libera cu distanta minima necalculata
79                 {
80                     A[y.l][y.c] = A[x.l][x.c]+1;
81                     //inserez pozitia y in coada
82                     C[++SfC] =y;
83                 }
84         }
85     }
86     if (A[ox][oy]==-2) printf("Nu exista solutie\n");
87     else NrDale=A[ox][oy];
88 }
89
90 void Afisare(void)
91 {
92     //afisez solutia
93     FILE * Fout=freopen("alee.out", "w", stdout);
94     printf("%d\n", NrDale);
95     fclose(Fout);

```



```

96  }
97
98  void main(void)
99  {
100     Citire();
101     Bordare();
102     DeterminNrDale();
103     Afisare();
104  }

```

### 16.1.3 Rezolvare detaliată

Listing 16.1.2: Aleel.java

```

1
2  %\begin{verbatim}
3  import java.io.*;
4  class aleel
5  {
6      static StreamTokenizer st;
7      static PrintWriter out;
8
9      static final int DimMaxParc=177;
10     static final int DimMaxCoadă=30625;
11
12     //deplasările pe linie și coloana pe direcțiile N,E,S,V
13     static final int[] dx = {0, -1, 0, 1, 0};
14     static final int[] dy = {0, 0, 1, 0, -1};
15
16     static int[] ql=new int[DimMaxCoadă];
17     static int[] qc=new int[DimMaxCoadă];
18
19     static int[][] A=new int[DimMaxParc][DimMaxParc];
20     /*
21      A[i][j]=-1 zona i,j este copac
22      A[i][j]=-2 zona i,j este libera
23      A[i][j]=d, zona i,j este libera, la distanța d de prima poartă
24     */
25
26     static int ix, iy, ox, oy;
27     static int n, m;
28     static int NrDale;
29
30     public static void main(String[] args) throws IOException
31     {
32         st=new StreamTokenizer(new BufferedReader(new FileReader("aleel.in")));
33         out=new PrintWriter(new BufferedWriter(new FileWriter("aleel.out")));
34
35         Citire();
36         Bordare();
37         DeterminNrDale();
38
39         out.println(NrDale);
40         out.close();
41     } // main
42
43     static void DeterminNrDale()
44     {
45         int IncC, SfC, k; //inceputul și sfarsitul cozii
46         int xl,xc,yl,yc;
47
48         //initializez coada
49         xl=ix;
50         xc=iy;
51         A[ix][iy]=1;
52
53         //pun poziția inițială, poarta de intrare, în coadă
54         IncC=SfC= 0;
55         ql[IncC]=xl;
56         qc[IncC]=xc;
57
58         //parcurs parcul

```

```

59     while(IncC<=SfC && A[ox][oy]==-2)
60     {
61         //extrag un element din coada
62         xl=ql[IncC];
63         xc=qc[IncC];
64         IncC++;
65
66         //ma deplasez in cele patru directii posibile
67         for(k=1;k<=4;k++)
68         {
69             yl=xl+dx[k];
70             yc=xc+dy[k];
71
72             //y = urmatoarea pozitie in directia k
73             if(A[yl][yc]==-2) // y=pozitie libera cu distanta minima necalculata
74             {
75                 A[yl][yc]=A[xl][xc]+1;
76
77                 //inserez pozitia y in coada
78                 ++SfC;
79                 ql[SfC]=yl;
80                 qc[SfC]=yc;
81             } // if
82         } // for
83     } // while
84
85     if(A[ox][oy]==-2) System.out.println("Nu exista solutie ...!!!");
86     else NrDale=A[ox][oy];
87 } //DeterminNrDale(...)
88
89 static void Bordare()
90 {
91     //bordez parcul cu pomi
92     int i, j;
93     for(i=1;i<=n;i++)
94     {
95         A[i][0]=-1;
96         A[i][n+1]=-1;
97         A[0][i]=-1;
98         A[n+1][i]=-1;
99     }
100 } // Bordare(...)
101
102 static void Citire() throws IOException
103 {
104     int k, i, j;
105
106     //citesc dimensiunile parcului si numarul de pomi
107     st.nextToken(); n=(int)st.nval;
108     st.nextToken(); m=(int)st.nval;
109
110     //marchez cu -2 pozitiile libere
111     for(i=1;i<=n;i++)
112         for(j=1;j<=m;j++) A[i][j]=-2;
113
114     //citesc coordonatele obstacolelor
115     for(k=1;k<=m;k++)
116     {
117         st.nextToken(); i=(int)st.nval;
118         st.nextToken(); j=(int)st.nval;
119         A[i][j]=-1; //marchez obstacolele cu -1
120     }
121
122     //citesc pozitiile portii de intrare si de iesire
123     st.nextToken(); ix=(int)st.nval;
124     st.nextToken(); iy=(int)st.nval;
125     st.nextToken(); ox=(int)st.nval;
126     st.nextToken(); oy=(int)st.nval;
127 } //Citire()
128 } // class
129 %\end{verbatim}

```

## 16.2 Dir

Costel trebuie să realizeze, împreună cu echipa sa, o aplicație software pentru gestiunea fișierelor de pe hard-disc, sarcina sa fiind aceea de a scrie un modul pentru determinarea căilor tuturor fișierelor de date aflate în structura arborescentă a folderelor de pe disc. Membrii echipei au stabilit o codificare proprie pentru memorarea structurii fișierelor de pe disc, utilizând un șir de caractere. Specificațiile tehnice sunt următoarele:

- folderul este un fișier de tip special, care poate conține fișiere și/sau foldere (acestea fiind considerate subfoldere ale folderului respectiv);
- numele folderelor încep cu o literă, au maxim 30 de caractere și sunt scrise cu majuscule;
- numele fișierelor de date încep cu o literă, au maxim 30 de caractere și sunt scrise cu minuscule;
- caracterele utilizate pentru numele fișierelor și folderelor sunt literele alfabetului englez și cifrele arabe;
- reprezentarea structurii fișierelor sub forma unui șir de caractere se realizează după următoarea regulă:

NUME\_FOLDER(lista\_de\_foldere\_si\_fisiere)

unde lista\_de\_foldere\_si\_fisiere, posibil vidă, conține fișierele și/sau subfolderele folderului NUME\_FOLDER, separate prin virgulă. Subfolderele se reprezintă respectând aceeași regulă.

De exemplu, structura de fișiere și foldere din figura de mai jos

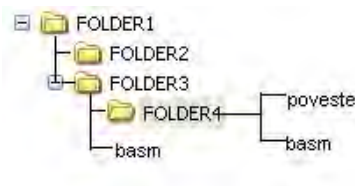


Figura 16.1: Dir

se reprezintă prin șirul de caractere:

FOLDER1 ( FOLDER2 ( ) , FOLDER3 ( FOLDER4 ( poveste , basm ) , basm ) )

### Cerință

Scrieți un program care, cunoscând șirul de caractere ce codifică o structură de fișiere de pe disc, determină calea pentru fiecare fișier de date din structură. Prin cale a unui fișier se înțelege o succesiune de foldere, fiecare folder fiind urmat de caracterul \ (backslash), începând de la folderul aflat pe cel mai înalt nivel al structurii (primul specificat în șirul ce codifică structura de fișiere), până la subfolderul în care se află fișierul de date respectiv și terminată cu numele fișierului. Căile determinate vor fi afișate în ordine lexicografică.

### Date de intrare

Fișierul de intrare **dir.in** conține pe prima linie șirul de caractere ce codifică structura de fișiere de pe disc.

### Date de ieșire

Fișierul de ieșire **dir.out** va conține pe prima linie un număr natural  $N$  reprezentând numărul de fișiere de date găsite. Pe următoarele  $N$  linii se vor scrie, în ordine lexicografică, căile ce permit identificarea fișierelor găsite, în formatul:  $F1 \backslash F2 \backslash \dots \backslash F_n \backslash \text{fișier}$ , câte o cale pe o linie.

### Restricții și precizări

Șirul de caractere ce codifică structura de fișiere este nevid și conține maxim 1600 de caractere. Structura de foldere conține cel puțin un folder și cel puțin un fișier.

Numărul de fișiere de date este cel mult 100.

Lungimea căii unui fișier este de cel mult 255 caractere.

Șirul  $x_1 x_2 \dots x_n$  este mai mic lexicografic decât șirul  $y_1 y_2 \dots y_m$ , dacă există  $k$  astfel încât  $x_1 = y_1$ ,  $x_2 = y_2$ , ...,  $x_{k-1} = y_{k-1}$  și  $(x_k < y_k \text{ sau } k = n + 1)$ .

### Punctaj

Pentru determinarea corectă a numărului de fișiere de date se acordă 30% din punctaj. Dacă numărul de fișiere de date a fost determinat corect și căile sunt corect afișate în ordine lexicografică se acordă punctajul integral.

**Exemplu**

dir.in
FOLDER1 ( FOLDER2 ( ) , FOLDER3 ( FOLDER4 ( poveste , basm ) , basm ) )

dir.out
3
FOLDER1\FOLDER3\FOLDER4\basm
FOLDER1\FOLDER3\FOLDER4\poveste
FOLDER1\FOLDER3\basm

**Timp maxim de execuție/test:** 1 secundă

### 16.2.1 Indicații de rezolvare

Pentru determinarea tuturor căilor fișierelor din structura dată vom utiliza o stivă ST, care va reține, la un moment dat succesiunea directoarelor de la folderul rădăcină și până la folderul curent (ultimul descoperit).

Algoritmul de rezolvare este următorul:

1. Inițial stiva este vidă.
2. Parcurgem șirul de intrare caracter cu caracter și identificăm fișierele și/sau folderele, astfel: O secvență de caractere reprezintă:
  - un folder dacă este delimitată de caracterele "(" și "(" sau "," și "(" și conține doar majuscule;
  - un fișier dacă este delimitată de caracterele "(" și "," sau "," și "," sau "," și ")";

Dacă este identificat un folder, atunci numele acestuia se introduce în stivă și începem explorarea fișierelor și folderelor subordonate acestuia.

Dacă este identificat un fișier, atunci calea asociată este compusă din folderele din stivă, de la bază către vârf, separate prin '\'. Calea determinată este memorată în tabloul de șiruri de caractere Rez.

3. După epuizarea fișierelor și folderelor din directorul curent acesta este extras din stivă.

Se repetă procedeul până când stiva devine vidă.

În final, sortăm lexicografic șirurile de caractere memorate în tabloul Rez.

Problema admite și o soluție recursivă.

### 16.2.2 Cod sursă

Listing 16.2.1: dir\_em.c

```

1  /*Emanuela Cerchez - Liceul de Informatica "Grigore Moisil" Iasi */
2  #include <stdio.h>
3  #include <string.h>
4
5  #define InFile  "dir.in"
6  #define OutFile "dir.out"
7  #define LGS  1601
8  #define NMaxF 100
9  #define LGC  256
10
11 typedef char Nume[31];
12 char s[LGS];
13 Nume sol[200];
14 FILE *fout;
15 int poz=0, nr=0;
16 char a[NMaxF][LGC];
17 int nrsol=0;
18
19 void citire(void);
20 void structura(void);
21 void fisier(void);
22 void afisare(void);
23 void sortare(void);

```

```

24
25 int main()
26 {
27     citire();
28     structura();
29     sortare();
30     afisare();
31     return 0;
32 }
33
34 void citire()
35 {
36     FILE * fin = fopen(InFile, "r");
37     fscanf(fin, "%s", s);
38     fclose(fin);
39 }
40
41 void structura()
42 {
43     char *p;
44     if (!s[poz]) return;
45     p=strchr(s+poz, '(');
46     *p=NULL;
47     strcpy(sol[nr++], s+poz);
48     poz+=strlen(s+poz)+1;
49     if (s[poz]==' ')
50     { nr--; poz++; }
51     else
52     {
53         do
54         {
55             if (s[poz]>='A' && s[poz]<='Z')
56                 structura();
57             else
58                 if (s[poz]>='a' && s[poz]<='z')
59                     fisier();
60             if (s[poz++]==' ') break;
61         }
62         while (1);
63         nr--;
64     }
65 }
66
67 void fisier()
68 {char nf[31];
69 int i;
70 for (i=poz; s[i]>='a' && s[i]<='z' || s[i]<='9' && s[i]>='0'; i++)
71     nf[i-poz]=s[i];
72 nf[i-poz]=NULL;
73 for (i=0; i<nr; i++)
74     {strcat(a[nrsol], sol[i]);
75      strcat(a[nrsol], "\\");}
76 strcat(a[nrsol], nf);
77 nrsol++;
78 poz+=strlen(nf);
79 }
80
81 void afisare(void)
82 {
83     FILE * fout=fopen(OutFile, "w");
84     int i;
85     fprintf(fout, "%d\n", nrsol);
86     for (i=0; i<nrsol; i++)
87         fprintf(fout, "%s\n", a[i]);
88     fclose(fout);
89 }
90
91 void sortare(void)
92 {
93     int i, ok;
94     char aux[LGC];
95     do
96     {ok=1;
97      for (i=0; i<nrsol-1; i++)
98          if (strcmp(a[i], a[i+1])>0)
99              {strcpy(aux, a[i]);

```

```

100         strcpy(a[i],a[i+1]);
101         strcpy(a[i+1],aux);
102         ok=0;
103     }
104 }
105 while (!ok);
106 }

```

Listing 16.2.2: dir.cpp

```

1  /* Alin Burta - C.N. "B.P. Hasdeu Buzau */
2  #include <fstream>
3  #include <string.h>
4
5  using namespace std;
6
7  #define Fin  "dir.in"
8  #define Fout "dir.out"
9  #define LGM  3001
10 #define MaxS 256
11
12 typedef char *TStiva[MaxS];
13 TStiva ST;
14 char *Sir;      //Sirul dat
15 int VF;         //varful stivei
16 char *Rez[MaxS]; //tablou pentru memorarea solutiilor
17 int NrSol;      //numarul solutiilor
18
19 ofstream g(Fout);
20
21 void citire();
22 void rezolvare();
23 void afisare();
24 void init();
25 void Add(char *s);
26 char* Del();
27
28 int main()
29 {
30     init();
31     citire();
32     rezolvare();
33     afisare();
34     return 0;
35 }
36
37 void rezolvare()
38 {
39     int i,j,N;
40     int inc, sf, term;
41
42     N=strlen(Sir)-1;
43     char *dirC=new char[MaxS];
44     //determin directorul radacina si-l introduc in stiva
45     dirC[0]='\0';
46     i=0;
47     while(Sir[i]!='(') dirC[i]=Sir[i], i++;
48     dirC[i]='\0';
49     Add(dirC);
50     //cat timp stiva e nevida
51     while(VF)
52     {
53         term=0;
54         if(Sir[i]==')') {Del();i++;continue; }
55         if(Sir[i]=='(' && Sir[i]==')') VF--;
56         else
57         {
58             //caut folder
59             while(i<=N && !strchr(",(",Sir[i])) i++;
60             if(i<N) inc=i,i++;
61             else term=1;
62             while(i<=N && !strchr(",()",Sir[i])) i++;
63             if(i<=N) sf=i;
64             else term=1;
65

```

```

66     if(!term)
67     {
68         j=inc+1; while(j<sf) dirC[j-inc-1]=Sir[j],j++; dirC[sf-inc-1]='\0';
69         if(strchr(", ",Sir[inc]) && Sir[sf]=='(')
70         {
71             //inserez folderul in stiva
72             Add(dirC);
73         }
74     else
75     {
76         //am gasit un nume de fisier
77         if(strcmp(dirC,"")!=0)
78         {
79             //compun calea catre fisier si memorez rezultatul
80             NrSol++;
81             char *tmp=new char[MaxS];
82             char *bkslash="\\ ";
83             tmp[0]='\0';
84             for(j=1;j<=VF;j++) {strcat(tmp,ST[j]); strcat(tmp,bkslash);}
85             strcat(tmp,dirC);
86             Rez[NrSol]=new char[strlen(tmp)+1];
87             strcpy(Rez[NrSol],tmp);
88         }
89     }
90 }
91 else Del();
92 }
93 }
94 }
95
96 void afisare()
97 {
98     int i,ok=1;
99     char *tmp=new char[MaxS];
100     //sortez alfabetic caile gasite
101
102     while(ok)
103     {
104         ok=0;
105         for(i=1;i<NrSol;i++)
106             if(strcmp(Rez[i],Rez[i+1])>0)
107             {
108                 strcpy(tmp,Rez[i]);
109                 strcpy(Rez[i],Rez[i+1]);
110                 strcpy(Rez[i+1],tmp);
111                 ok=1;
112             }
113     }
114
115     g<<NrSol<<'\n';
116     for(i=1;i<=NrSol;i++) g<<Rez[i]<<'\n';
117     g.close();
118 }
119
120 void citire()
121 {
122     ifstream f(Fin);
123     f.get(Sir,3000,'\n');
124     f.get();
125     f.close();
126 }
127
128 void init()
129 {
130     int i;
131     Sir=new char[LGM];
132     VF=0;
133     NrSol=0;
134 }
135
136 void Add(char *s)
137 {
138     if(VF<MaxS) VF++, ST[VF]=strdup(s);
139 }
140
141 char* Del()

```

```

142 {
143     if(VF) {VF--; return ST[VF+1];}
144     return NULL;
145 }

```

### 16.2.3 Rezolvare detaliată

Listing 16.2.3: dir1.java

```

1  import java.io.*;
2  class dir1
3  {
4      static BufferedReader br;
5      static PrintWriter out;
6
7      static final int LGM=3001;
8      static final int MaxS=256;
9
10     static String[] ST=new String[MaxS];           // stiva
11     static String Sir;                             // sirul dat
12     static int VF;                                 // varful stivei
13     static String[] Rez=new String[MaxS];          // tablou pentru memorarea solutiilor
14     static int NrSol;                              // numarul solutiilor
15
16     public static void main(String[] args) throws IOException
17     {
18         br=new BufferedReader(new FileReader("dir.in"));
19         out=new PrintWriter(new BufferedWriter(new FileWriter("dir.out")));
20
21         Sir=br.readLine();
22         //System.out.println(Sir);
23
24         rezolvare();
25         afisare();
26
27         out.close();
28     } // main
29
30     static void afisare()
31     {
32         int i,ok=1;
33         String tmp=new String();
34
35         //sortez alfabetic caile gasite
36         while(ok==1)
37         {
38             ok=0;
39             for(i=1;i<NrSol;i++)
40                 if(Rez[i].compareTo(Rez[i+1])>0)
41                 {
42                     tmp=Rez[i];
43                     Rez[i]=Rez[i+1];
44                     Rez[i+1]=tmp;
45                     ok=1;
46                 }
47         }
48
49         out.println(NrSol);
50         for(i=1;i<=NrSol;i++) out.println(Rez[i]);
51     } // afisare(...)
52
53     static void rezolvare()
54     {
55         int i,j,N;
56         int inc=0, sf=0, term=0;
57
58         VF=0;                                       // stiva = vida
59         NrSol=0;
60
61         N=Sir.length()-1;                          // pozitia ultimului caracter
62         String dirC=new String();
63

```



```

64     //determin directorul radacina si-l introduc in stiva
65     dirC="";
66     i=0;
67     while(Sir.charAt(i)!='(') {dirC+=Sir.charAt(i); i++;}
68     VF++;
69     ST[VF]=new String(dirC);
70     //afisstiva();
71
72     //cat timp stiva e nevida
73     while(VF!=0)
74     {
75         //System.out.println(i+" --> "+Sir.charAt(i));
76         term=0;
77         if(Sir.charAt(i)=='(')
78         {
79             VF--;
80             i++;                // urmatorul caracter
81             continue;
82         }
83
84         //caut folder
85         while(i<=N && Sir.charAt(i)!=',' && Sir.charAt(i)!='(') i++;
86         if(i<N) inc=i++; else term=1;
87         while(i<=N && Sir.charAt(i)!=',' && Sir.charAt(i)!='(' && Sir.charAt(i)!='(') i++;
88         if(i<=N) sf=i; else term=1;
89
90         //System.out.println("inc = "+inc+" "+Sir.charAt(inc)+" sf = "+sf+" "+Sir.charAt(
91             sf));
92         if(sf==inc+1) continue;
93
94         if(term==0)
95         {
96             dirC="";
97             j=inc+1;
98             while(j<sf) dirC+=Sir.charAt(j++);
99
100            //System.out.println("dirC = "+dirC);
101
102            if(
103                (Sir.charAt(inc)!='(' && Sir.charAt(sf)=='(') ||
104                (Sir.charAt(inc)!=',' && Sir.charAt(sf)=='(')
105            )
106            {
107                //inserez folderul in stiva
108                VF++;
109                ST[VF]=new String(dirC);
110                //System.out.println(dirC+" --> stiva");
111                //afisstiva();
112            }
113            else
114            {
115                //am gasit un nume de fisier
116                if(!dirC.equals(""))
117                {
118                    //compun calea catre fisier si memorez rezultatul
119                    NrSol++;
120                    String tmp=new String();
121                    String bkslash="\\";
122                    tmp="";
123                    for(j=1;j<=VF;j++) {tmp+=ST[j]; tmp+=bkslash;}
124                    tmp+=dirC;
125                    Rez[NrSol]=new String(tmp);
126
127                    //System.out.println(dirC+" ==> SOLUTIE: "+tmp);
128                }
129            }
130            //if
131            else VF--;
132        }
133    }
134
135    static void afisstiva()
136    {
137        int i;
138        for(i=VF;i>=1;i--) System.out.println(i+" : "+ST[i]);
139        System.out.println();

```

```

139     }
140 } // class

```

## Versiunea 2:

Listing 16.2.4: dir2.java

```

1  import java.io.*;
2  import java.util.StringTokenizer;
3  class dir2
4  {
5      static BufferedReader br;
6      static PrintWriter out;
7
8      static final int LGM=3001;
9      static final int MaxS=256;
10
11     static String[] ST=new String[MaxS];    // stiva
12     static String Sir;                      // sirul dat
13     static int VF;                          // varful stivei
14     static String[] Rez=new String[MaxS];   // tablou pentru memorarea solutiilor
15     static int NrSol;                       // numarul solutiilor
16
17     public static void main(String[] args) throws IOException
18     {
19         br=new BufferedReader(new FileReader("dir.in"));
20         out=new PrintWriter(new BufferedWriter(new FileWriter("dir.out")));
21         Sir=br.readLine();
22         rezolvare();
23         afisare();
24         out.close();
25     } // main
26
27     static void afisare()
28     {
29         int i,ok=1;
30         String tmp=new String();
31
32         //sortez alfabetic caile gasite
33         while(ok==1)
34         {
35             ok=0;
36             for(i=1;i<NrSol;i++)
37                 if(Rez[i].compareTo(Rez[i+1])>0)
38                 {
39                     tmp=Rez[i];
40                     Rez[i]=Rez[i+1];
41                     Rez[i+1]=tmp;
42                     ok=1;
43                 }
44         }
45
46         out.println(NrSol);
47         for(i=1;i<=NrSol;i++) out.println(Rez[i]);
48     } // afisare(...)
49
50     static void rezolvare()
51     {
52         StringTokenizer str=new StringTokenizer(Sir,"(,)",true);
53         String token;
54
55         int i,j;
56         int inc=0, sf=0;
57
58         VF=0;                                // stiva = vida
59         NrSol=0;
60
61         //determin directorul radacina si-l introduc in stiva
62         token = str.nextToken();
63         sf=inc+token.length();
64
65         VF++;
66         ST[VF]=new String(token);
67

```

```
68     while(str.hasMoreTokens())
69     {
70         inc=sf;
71
72         if(Sir.charAt(sf)=='(')
73         {
74             VF--;
75             sf++;
76             continue;
77         }
78
79         if(Sir.charAt(inc+1)=='(') { sf=inc+1; continue;}
80
81         //caut folder
82         token = str.nextTok();
83         sf=inc+token.length()+1;
84
85         if(
86             (Sir.charAt(inc)!='(' && Sir.charAt(sf)=='(') ||
87             (Sir.charAt(inc)!=',' && Sir.charAt(sf)=='(')
88         )
89         {
90             //inserez folderul in stiva
91             VF++;
92             ST[VF]=new String(token);
93         }
94         else
95         {
96             //am gasit un nume de fisier
97             if(!token.equals(""))
98             {
99                 //compun calea catre fisier si memorez rezultatul
100                 NrSol++;
101                 String tmp=new String();
102                 String bkslash="\\\\";
103                 tmp="";
104                 for(j=1; j<=VF; j++) {tmp+=ST[j]; tmp+=bkslash;}
105                 tmp+=token;
106                 Rez[NrSol]=new String(tmp);
107             }
108             } // else
109
110         inc=sf;
111     } // while
112 } // rezolvare
113 } // class
```

---

# Capitolul 17

## OJI 2006

### 17.1 Ecuații

*Emanuela Cerchez*

Să considerăm ecuații de gradul I, de forma:

$$\text{expresie}_1 = \text{expresie}_2$$

Expresiile specificate sunt constituite dintr-o succesiune de operanzi, între care există semnul  $+$  sau semnul  $-$  (cu semnificația binecunoscută de adunare, respectiv scădere).

Fiecare operand este fie un număr natural, fie un număr natural urmat de litera  $x$  (litera  $x$  reprezentând necunoscuta), fie doar litera  $x$  (ceea ce este echivalent cu  $1x$ ).

De exemplu:

$$2x - 5 + 10x + 4 = 20 - x$$

Observați că în ecuațiile noastre nu apar paranteze și necunoscuta este întotdeauna desemnată de litera mică  $x$ .

#### Cerință

Scrieți un program care să rezolve ecuații de gradul I, în formatul specificat în enunțul problemei.

#### Datele de intrare

Fisierul de intrare **ecuatii.in** conține pe prima linie un număr natural  $n$ , reprezentând numărul de ecuații din fișier. Pe fiecare dintre următoarele  $n$  linii este scrisă câte o ecuație.

#### Datele de ieșire

În fișierul de ieșire **ecuatii.out** vor fi scrise  $n$  linii, câte una pentru fiecare ecuație din fișierul de intrare. Pe linia  $i$  va fi scrisă soluția ecuației de pe linia  $i + 1$  din fișierul de intrare.

Dacă soluția ecuației este un număr real, atunci acesta se va scrie cu 4 zecimale. Răspunsul este considerat corect dacă diferența în valoare absolută dintre soluția corectă și soluția concurentului este  $< 0.001$ .

În cazul în care ecuația admite o infinitate de soluții, se va scrie mesajul *infinit* (cu litere mici).

Dacă ecuația nu admite soluții, se va scrie mesajul *imposibil* (de asemenea cu litere mici).

#### Restricții și precizări

- $1 \leq n \leq 10$
- Lungimea unei ecuații nu depășește 255 caractere.
- Ecuațiile nu conțin spații.
- Numerele naturale care intervin în ecuații sunt  $\leq 1000$ .
- Punctajul pe un test se acordă dacă și numai dacă toate ecuațiile din testul respectiv au fost rezolvate corect.

**Exemple**

ecuatii.in	ecuatii.out
3	3.2527
2x-4+5x+300=98x	infinit
x+2=2+x	imposibil
3x+5=3x+2	

**Timp maxim de execuție/test:** 1 secundă

**17.1.1 Indicații de rezolvare***Soluția comisiei*

Vom citi ecuația într-un șir de caractere, apoi vom împărți șirul în două subșiruri (membrul stâng, cu alte cuvinte caracterele până la semnul egal formează primul șir, și membrul drept, cu alte cuvinte caracterele de după semnul egal formează al doilea șir).

Problema este de a determina pentru fiecare dintre cele două subșiruri coeficientul lui  $x$  și termenul liber.

Să notăm:

$nr1$  termenul liber din membrul stâng

$nr2$  termenul liber din membrul drept

$nrx1$  coeficientul lui  $x$  din membrul stâng

$nrx2$  coeficientul lui  $x$  din membrul drept.

Soluția ecuației este  $(nr2 - nr1)/(nrx1 - nrx2)$  dacă  $nrx1 \neq nrx2$ .

Dacă  $nrx1 = nrx2$ , atunci ecuația este imposibilă (dacă  $nr1 \neq nr2$ ) sau nedeterminată (dacă  $nr1 = nr2$ ).

Pentru a determina  $nr1$  și  $nrx1$ , respectiv  $nr2$  și  $nrx2$ , se prelucrează cele două șiruri, identificând coeficienții necunoscutei, respectiv termenii liberi.

**17.1.2 Cod sursă**

Listing 17.1.1: ecuatii.cpp

```

1  #include <stdio.h>
2  #include <math.h>
3  #include <stdlib.h>
4  #include <string.h>
5
6  #define InFile "ecuatii.in"
7  #define OutFile "ecuatii.out"
8  #define LgMax 300
9
10 char s[LgMax], s1[LgMax], s2[LgMax];
11 int n;
12 long nr1, nr2, nrx1, nrx2;
13
14 void rezolva(char *, long &, long &);
15
16 int main()
17 {
18     FILE * fin=fopen(InFile, "r");
19     FILE * fout=fopen(OutFile, "w");
20
21     int i;
22     char *p;
23
24     fscanf(fin, "%d", &n);
25     for (i=0; i<n; i++)
26     {
27         fscanf(fin, "%s", s1);
28         p=strchr(s1, '=');
29         strcpy(s2, p+1);
30         *p=NULL;
31         rezolva(s1, nr1, nrx1);
32         rezolva(s2, nr2, nrx2);
33         if (nrx1==nrx2)
34             if (nr1==nr2)
35                 fprintf(fout, "infinit\n");

```

```

36         else
37             fprintf(fout, "imposibil\n");
38     else
39         fprintf(fout, "%.4lf\n", ((double) (nr2-nr1) / (nrx1-nrx2)));
40     }
41
42     fclose(fin);
43     fclose(fout);
44     return 0;
45 }
46
47 void rezolva(char *s, long &nr, long &nrx)
48 {
49     char *p, ss[LgMax];
50     long v;
51     int semn=1, l;
52
53     strcpy(ss, s);
54     p=strtok(ss, "+-");
55     nr=0;
56     nrx=0;
57     while (p)
58     {
59         l=strlen(p);
60         if (p[0]=='x')
61             nrx+=semn;
62         else
63             if (p[l-1]=='x')
64             {
65                 p[l-1]=NULL;
66                 v=atol(p);
67                 nrx=nrx+semn*v;
68             }
69             else
70             {
71                 v=atol(p);
72                 nr=nr+semn*v;
73             }
74
75             if (s[p+l-ss]=='+')
76                 semn=1;
77             else
78                 semn=-1;
79
80             p=strtok(NULL, "+-");
81     }
82 }

```

### 17.1.3 Rezolvare detaliată

Listing 17.1.2: ecuatii.java

```

1  import java.io.*;
2  import java.util.StringTokenizer;
3  class ecuatii
4  {
5      static BufferedReader br;
6      static StringTokenizer st;
7      static PrintWriter out;
8
9      static String s, s1, s2;
10
11     static int n;
12     static int nr, nrx;
13     static int nr1, nr2, nrx1, nrx2;
14     static double x;           // solutia
15
16     public static void main(String[] args) throws IOException
17     {
18         br=new BufferedReader(new FileReader("ecuatii.in"));
19         st=new StringTokenizer(br);
20         out=new PrintWriter(new BufferedWriter(new FileWriter("ecuatii.out")));

```

```

21
22     int i,pozitieegal;
23
24     st.nextToken(); n=(int)st.nval;
25     br.readLine(); // citit EOL ... de pe prima linie ... !!!
26
27     for (i=0; i<n; i++)
28     {
29         s=br.readLine();
30
31         pozitieegal=s.indexOf("=");
32         s1=s.substring(0,pozitieegal); // ... atentie ... !!!
33         s2=s.substring(pozitieegal+1);
34
35         rezolva(s1); nr1=nr; nrx1=nrx;
36         rezolva(s2); nr2=nr; nrx2=nrx;
37
38         if(nrx1==nrx2)
39             if(nr1==nr2) out.println("infinit");
40             else out.println("imposibil");
41         else
42         {
43             // vezi ... Class NumberFormat ... dar aici ...
44             String sol="";
45             x=(double) (nr2-nr1) / (nrx1-nrx2);
46             if(x<0) {sol+="-"; x=-x;}
47             x=x+0.00005;
48
49             int xi=(int)x;
50             sol+=xi;
51             sol+=".";
52             x=x-xi;
53
54             int yi=(int) (x*10000);
55             sol+=yi;
56
57             if(yi<9) sol+="000"; else
58                 if(yi<99) sol+="00"; else
59                     if(yi<999) sol+="0";
60             out.println(sol);
61         }
62     }
63     out.close();
64 } // main
65
66 static void rezolva(String s)
67 {
68     StringTokenizer str=new StringTokenizer(s,"+-");
69     String token;
70     int v;
71     int semn=1,lg=-1;
72
73     nr=0; nrx=0;
74     while(str.hasMoreTokens())
75     {
76         token = str.nextToken();
77         lg=lg+token.length()+1;
78         if (token.startsWith("x"))
79         {
80             nrx+=semn;
81         }
82         else
83         {
84             if(token.endsWith("x"))
85             {
86                 v=Integer.parseInt(token.replace("x",""));
87                 nrx=nrx+semn*v;
88             }
89             else
90             {
91                 v=Integer.parseInt(token);
92                 nr=nr+semn*v;
93             }
94         }
95     }
96     if(lg<s.length())

```

```

97         if(s.substring(lg,lg+1).equals("+")) semn=1; else semn=-1;
98     } // while
99 } // rezolva(...)
100 } // class

```

## 17.2 Sudest

*Alin Burtza*

Fermierul Ion deține un teren de formă pătrată, împărțit în  $N \times N$  pătrate de latură unitate, pe care cultivă cartofi. Pentru recoltarea cartofilor fermierul folosește un robot special proiectat în acest scop. Robotul pornește din pătratul din stânga sus, de coordonate  $(1, 1)$  și trebuie să ajungă în pătratul din dreapta jos, de coordonate  $(N, N)$ .

Traseul robotului este programat prin memorarea unor comenzi pe o cartelă magnetică. Fiecare comandă specifică direcția de deplasare (sud sau est) și numărul de pătrate pe care le parcurge în direcția respectivă. Robotul strânge recolta doar din pătratele în care se oprește între două comenzi.

Din păcate, cartela pe care se află programul s-a deteriorat și unitatea de citire a robotului nu mai poate distinge direcția de deplasare, ci numai numărul de pași pe care trebuie să-i facă robotul la fiecare comandă. Fermierul Ion trebuie să introducă manual, pentru fiecare comandă, direcția de deplasare.

### Cerință

Scrieți un program care să determine cantitatea maximă de cartofi pe care o poate culege robotul, în ipoteza în care Ion specifică manual, pentru fiecare comandă, direcția urmată de robot. Se va determina și traseul pe care se obține recolta maximă.

### Datele de intrare

Fișierul de intrare **sudest.in** are următoarea structură:

- Pe linia 1 se află numărul natural  $N$ , reprezentând dimensiunea parcelei de teren.
- Pe următoarele  $N$  linii se află câte  $N$  numere naturale, separate prin spații, reprezentând cantitatea de cartofi din fiecare pătrat unitate.
- Pe linia  $N + 2$  se află un număr natural  $K$  reprezentând numărul de comenzi aflate pe cartela magnetică.
- Pe linia  $N + 3$  se află  $K$  numerele naturale  $C_1, \dots, C_K$ , separate prin spații, reprezentând numărul de pași pe care trebuie să-i efectueze robotul la fiecare comandă.

### Datele de ieșire

Fișierul de ieșire **sudest.out** va conține pe prima linie cantitatea maximă de cartofi recoltată de robot. Pe următoarele  $K + 1$  linii vor fi scrise, în ordine, coordonatele pătratelor unitate ce constituie traseul pentru care se obține cantitate maximă de cartofi, câte un pătrat unitate pe o linie. Coordonatele scrise pe aceeași linie vor fi separate printr-un spațiu. Primul pătrat de pe traseu va avea coordonatele 11, iar ultimul va avea coordonatele  $NN$ . Dacă sunt mai multe trasee pe care se obține o cantitate maximă de cartofi recoltată se va afișa unul dintre acestea.

### Restricții și precizări

- $5 \leq N \leq 100$
- $2 \leq K \leq 2 * N - 2$
- $1 \leq C_1, \dots, C_K \leq 10$
- Cantitatea de cartofi dintr-un pătrat de teren este număr natural între 0 și 100.
- Pentru fiecare set de date de intrare se garantează că există cel puțin un traseu.
- Se consideră că robotul strânge recolta și din pătratul de plecare  $(1, 1)$  și din cel de sosire  $(N, N)$ .
- Pentru determinarea corectă a cantității maxime recoltate se acordă 50% din punctajul alocat testului respectiv; pentru cantitate maximă recoltată și traseu corect se acordă 100%.



**Exemplu**

sudest.in	sudest.out	Explicații
6	29	Un alt traseu posibil este:
1 2 1 0 4 1	1 1	1 1
1 3 3 5 1 1	3 1	1 3
2 2 1 2 1 10	5 1	1 5
4 5 3 9 2 6	6 1	2 5
1 1 3 2 0 1	6 5	6 5
10 2 4 6 5 10	6 6	6 6
5		dar costul său este $1 + 1 + 4 + 1 + 5 + 10 = 22$
2 2 1 4 1		

**Timp maxim de execuție/test:** 1 secundă

**17.2.1 Indicații de rezolvare**

*soluția comisiei*

Reprezentarea informațiilor

- $N$  - numărul de linii
- $K$  - numărul de comenzi
- $A[Nmax][Nmax]$ ; - memorează cantitatea de produs
- $C[Nmax][Nmax]$ ; -  $C[i][j]$  = cantitatea maximă de cartofi culeasă pe un traseu ce pornește din  $(1, 1)$  și se termină în  $(i, j)$ , respectând condițiile problemei
- $P[Nmax][Nmax]$ ; -  $P[i][j]$  = pasul la care am ajuns în poziția  $i, j$  culegând o cantitate maximă de cartofi
- $Move[2 * Nmax]$ ; - memorează cele  $K$  comenzi

Parcurs șirul celor  $k$  mutări. La fiecare mutare marcheaz pozițiile în care pot ajunge la mutarea respectivă.

Mai exact, parcurs toate pozițiile în care am putut ajunge la pasul precedent (cele marcate în matricea  $P$  corespunzător cu numărul pasului precedent) și pentru fiecare poziție verific dacă la pasul curent pot să execut mutarea la sud.

În caz afirmativ, verific dacă în acest caz obțin o cantitate de cartofi mai mare decât cea obținută până la momentul curent (dacă da, rețin noua cantitate, și marcheaz în matricea  $P$  poziția în care am ajuns cu indicele mutării curente).

În mod similar procedez pentru o mutare spre est.

**17.2.2 Cod sursă**

Listing 17.2.1: sudest.cpp

```

1  #include <fstream>
2
3  using namespace std;
4
5  #define Fin  "sudest.in"
6  #define Fout "sudest.out"
7  #define Nmax 100
8
9  int N;          //numarul de linii
10 int K;          //numarul de comenzi
11 int A[Nmax][Nmax]; //memoreaza cantitatea de produs
12 int C[Nmax][Nmax]; //C[i][j]=cantitatea maxima de cartofi culeasa pe
13                    // un traseu ce porneste din (1,1) si se termina
14                    // in (i,j), respectand conditiile problemei
15 unsigned char P[Nmax][Nmax];
16 int Move[2*Nmax]; //memoreaza cele K comenzi
17
18 ofstream g(Fout);

```

```

19
20 void read_data()
21 {
22     int i, j;
23     ifstream f(Fin);
24     f>>N;
25     for(i=0; i<N; i++)
26         for(j=0; j<N; j++) f>>A[i][j];
27     f>>K;
28     for(i=1; i<=K; i++) f>>Move[i];
29     f.close();
30 }
31
32 void init()
33 {
34     int i, j;
35     for(i=0; i<N; i++)
36         for(j=0; j<N; j++) {C[i][j]=-1; P[i][j]=255;}
37 }
38
39 int posibil(int x, int y)
40 {
41     return 0<=x && 0<=y && x<N && y<N;
42 }
43
44
45 void drum(int x, int y, int pas)
46 {
47     int i, gasit;
48     if(x==0 && y==0) g<<1<<" "<<1<<'\\n';
49     else
50     {
51         gasit=0;
52         if (posibil(x, y-Move[pas]))
53             if (C[x][y-Move[pas]]==C[x][y]-A[x][y] && P[x][y-Move[pas]]==pas-1)
54             {
55                 drum(x, y-Move[pas], pas-1);
56                 g<<x+1<<" "<<y+1<<endl;
57                 gasit=1;
58             }
59         if (!gasit)
60             if (posibil(x-Move[pas], y))
61                 if (C[x-Move[pas]][y]==C[x][y]-A[x][y] && P[x-Move[pas]][y]==pas-1)
62                 {
63                     drum(x-Move[pas], y, pas-1);
64                     g<<x+1<<" "<<y+1<<endl;
65                 }
66     }
67
68 void solve()
69 {
70     int i, j, h;
71     P[0][0]=0; C[0][0]=A[0][0];
72     for (i=1; i<=K; i++)
73         for (j=0; j<N; j++)
74             for (h=0; h<N; h++)
75                 if (P[j][h]==i-1)
76                 {
77                     if (posibil(j+Move[i], h))
78                         if (C[j][h]+A[j+Move[i]][h]>C[j+Move[i]][h])
79                         {
80                             P[j+Move[i]][h]=i;
81                             C[j+Move[i]][h]=C[j][h]+A[j+Move[i]][h];
82                         }
83                     if (posibil(j, Move[i]+h))
84                         if (C[j][h]+A[j][Move[i]+h]>C[j][Move[i]+h])
85                         {
86                             P[j][Move[i]+h]=i;
87                             C[j][Move[i]+h]=C[j][h]+A[j][Move[i]+h];
88                         }
89                 }
90     g<<C[N-1][N-1]<<'\\n';
91     drum(N-1, N-1, K);
92     g.close();
93 }
94
95 int main()
96 {
97     read_data();
98     init();
99     solve();

```

```

95     return 0;
96 }

```

### 17.2.3 Rezolvare detaliată

Variantă după soluția oficială:

Listing 17.2.2: sudest1.java

```

1  import java.io.*;
2  class Sudest1
3  {
4      static StreamTokenizer st;
5      static PrintWriter out;
6      static final int Nmax=101;
7      static int N, K;
8      static int[][] A=new int[Nmax][Nmax]; // A[i][j]=cantitatea de cartofi
9      static int[][] C=new int[Nmax][Nmax]; // C[i][j]=cantitatea maxima ...
10     static int[][] P=new int[Nmax][Nmax]; // pas
11     static int[] Move=new int[2*Nmax]; // comenzile
12
13     public static void main(String[] args) throws IOException
14     {
15         st=new StreamTokenizer( new BufferedReader(new FileReader("sudest.in")));
16         out=new PrintWriter(new BufferedWriter(new FileWriter("sudest.out")));
17         read_data();
18         init();
19         solve();
20     } // main(...)
21
22     static void read_data() throws IOException
23     {
24         int i,j,cmd;
25         st.nextToken(); N=(int)st.nval;
26         for(i=1;i<=N;i++)
27             for(j=1;j<=N;j++) { st.nextToken(); A[i][j]=(int)st.nval; }
28         st.nextToken(); K=(int)st.nval;
29         for(cmd=1;cmd<=K;cmd++) { st.nextToken(); Move[cmd]=(int)st.nval; }
30     } // read_data()
31
32     static void init()
33     {
34         int i,j;
35         for(i=1;i<=N;i++) for(j=1;j<=N;j++) { C[i][j]=-1; P[i][j]=255; }
36     } // init()
37
38     static boolean posibil(int x,int y) {return 1<=x && 1<=y && x<=N && y<=N;}
39
40     static void solve()
41     {
42         int i,j, cmd;
43         P[1][1]=0;
44         C[1][1]=A[1][1];
45         for(cmd=1; cmd<=K; cmd++)
46             for(i=1; i<=N; i++)
47                 for(j=1; j<=N; j++)
48                     if(P[i][j]==cmd-1)
49                     {
50                         if(posibil(i+Move[cmd],j)) // SUD
51                         if(C[i][j]+A[i+Move[cmd]][j]>C[i+Move[cmd]][j])
52                         {
53                             P[i+Move[cmd]][j]=cmd;
54                             C[i+Move[cmd]][j]=C[i][j]+A[i+Move[cmd]][j];
55                         }
56                         if(posibil(i,j+Move[cmd])) // EST
57                         if(C[i][j]+A[i][j+Move[cmd]]>C[i][j+Move[cmd]])
58                         {
59                             P[i][j+Move[cmd]]=cmd;
60                             C[i][j+Move[cmd]]=C[i][j]+A[i][j+Move[cmd]];
61                         }
62                     } // if
63         out.println(C[N][N]);          drum(N,N,K);          out.close();
64     } // solve()

```

```

65
66 static void drum(int x, int y, int pas)
67 {
68     int i;
69     boolean gasit;
70     if(x==1 && y==1) out.println("1 1");
71     else
72     {
73         gasit=false;
74         if(posibil(x,y-Move[pas]))
75             if(C[x][y-Move[pas]]==C[x][y]-A[x][y] && P[x][y-Move[pas]]==pas-1)
76             {
77                 drum(x,y-Move[pas],pas-1);
78                 out.println(x+" "+y);
79                 gasit=true;
80             }
81         if(!gasit)
82             if(posibil(x-Move[pas],y))
83                 if(C[x-Move[pas]][y]==C[x][y]-A[x][y] && P[x-Move[pas]][y]==pas-1)
84                 {
85                     drum(x-Move[pas],y,pas-1);
86                     out.println(x+" "+y);
87                 }
88     } // else
89 } // drum(...)
90 } // class

```

**Varianta folosind coada:**

Listing 17.2.3: sudest2.java

```

1 import java.io.*; // 1l, ...,nn --> 1,...,n*n pozitii in matrice !
2 class Sudest2
3 {
4     static StreamTokenizer st;
5     static PrintWriter out;
6     static final int nmax=101;
7     static int n, k;
8     static int[][] a=new int[nmax][nmax]; // a[i][j]=cantitatea de cartofi
9     static int[][] c=new int[nmax][nmax]; // c[i][j]=cantitatea maxima ...
10    static int[][] p=new int[nmax][nmax]; // pozitia anterioara optima
11    static int[] move=new int[2*nmax]; // comenzile
12    static int ic,sc,scv,qmax=1000; // qmax=100 este prea mic ...!!!
13    static int[] q=new int[qmax]; // coada
14
15    public static void main(String[] args) throws IOException
16    {
17        st=new StreamTokenizer(new BufferedReader(new FileReader("sudest.in")));
18        out=new PrintWriter(new BufferedWriter(new FileWriter("sudest.out")));
19        read_data();
20        init();
21        solve();
22    } // main(...)
23
24    static void read_data() throws IOException
25    {
26        int i,j,cmd;
27        st.nextToken(); n=(int)st.nval;
28        for(i=1;i<=n;i++)
29            for(j=1;j<=n;j++) { st.nextToken(); a[i][j]=(int)st.nval; }
30        st.nextToken(); k=(int)st.nval;
31        for(cmd=1;cmd<=k;cmd++) { st.nextToken(); move[cmd]=(int)st.nval; }
32    } // read_data()
33
34    static void init()
35    {
36        int i,j;
37        for(i=1;i<=n;i++) for(j=1;j<=n;j++) {c[i][j]=-1; p[i][j]=255;}
38    } // init()
39
40    static void solve()
41    {
42        int i,j,ii,jj,cmd;
43        p[1][1]=0; c[1][1]=a[1][1];

```

---

```

44     ic=0; sc=1; q[ic]=1; // pozitia [1][1] --> q
45     scv=1; // ultimul din coada la distanta 1 de [1][1]
46     for(cmd=1; cmd<=k; cmd++)
47     {
48         while(ic!=scv)
49         {
50             i=(q[ic]-1)/n+1; j=(q[ic]-1)%n+1; ic=(ic+1)%qmax; // scot din coada
51             // propag catre Sud
52             ii=i+move[cmd];
53             jj=j;
54             if((ii>=1)&&(ii<=n))
55                 if(c[i][j]+a[ii][jj]>c[ii][jj])
56                 {
57                     c[ii][jj]=c[i][j]+a[ii][jj];
58                     p[ii][jj]=(i-1)*n+j;
59                     q[sc]=(ii-1)*n+jj; sc=(sc+1)%qmax; // pun in coada
60                 }
61
62             // propag catre Est
63             jj=j+move[cmd];
64             ii=i;
65             if((jj>=1)&&(jj<=n))
66                 if(c[i][j]+a[ii][jj]>c[ii][jj])
67                 {
68                     c[ii][jj]=c[i][j]+a[ii][jj];
69                     p[ii][jj]=(i-1)*n+j;
70                     q[sc]=(ii-1)*n+jj; sc=(sc+1)%qmax; // pun in coada
71                 }
72             }// while
73             scv=sc;
74         }// for
75
76         out.println(c[n][n]); drum(n,n); out.close();
77     }// solve()
78
79     static void drum(int i, int j)
80     {
81         if(i*j==0) return;
82         drum((p[i][j]-1)/n+1, (p[i][j]-1)%n+1);
83         out.println(i+" "+j);
84     }// drum(...)
85 }// class

```

---

# Capitolul 18

## OJI 2005

### 18.1 Lăcusta

*Nistor Eugen Moț*

Se consideră o matrice dreptunghiulară cu  $m$  linii și  $n$  coloane, cu valori naturale. Traversăm matricea pornind de la colțul stânga-sus la colțul dreapta-jos. O traversare constă din mai multe deplasări. La fiecare deplasare se execută un salt pe orizontală și un pas pe verticală. Un salt înseamnă că putem trece de la o celulă la oricare alta aflată pe aceeași linie, iar un pas înseamnă că putem trece de la o celulă la celula aflată imediat sub ea. Excepție face ultima deplasare (cea în care ne aflăm pe ultima linie), când vom face doar un salt pentru a ajunge în colțul dreapta-jos, dar nu vom mai face și pasul corespunzător. Astfel traversarea va consta din vizitarea a  $2m$  celule.

#### Cerință

Scrieți un program care să determine suma minimă care se poate obține pentru o astfel de traversare.

#### Datele de intrare

Fișierul de intrare **lacusta.in** conține pe prima linie două numere naturale separate printr-un spațiu  $m$   $n$ , reprezentând numărul de linii și respectiv numărul de coloane ale matricei. Pe următoarele  $m$  linii este descrisă matricea, câte  $n$  numere pe fiecare linie, separate prin câte un spațiu.

#### Datele de ieșire

Fișierul de ieșire **lacusta.out** va conține o singură linie pe care va fi scrisă suma minimă găsită.

#### Restricții și precizări

- $1 \leq m, n \leq 100$
- Valorile elementelor matricei sunt numere întregi din intervalul  $[1, 255]$ .

#### Exemple

lacusta.in	lacusta.out	Explicatie
4 5 3 4 5 7 9 6 6 3 4 4 6 3 3 9 6 6 5 3 8 2	28	Drumul este: $(1, 1) \rightarrow (1, 3) \rightarrow$ $(2, 3) \rightarrow (2, 2) \rightarrow$ $(3, 2) \rightarrow (3, 3) \rightarrow$ $(4, 3) \rightarrow (4, 5)$

**Timp maxim de executare:** 1 secundă/test

#### 18.1.1 Indicații de rezolvare

*Ginjo nr. 15/3 martie 2005*

Pentru rezolvarea acestei probleme vom utiliza metoda *programării dinamice*.

Vom nota prin  $A$  matricea dată și vom construi o matrice  $B$  ale cărei elemente  $b_{ij}$  vor conține sumele minime necesare pentru a ajunge în celula  $(i, j)$  pornind din celula  $(i - 1, j)$ .

Vom completa inițial elementele de pe a doua linie a matricei  $B$ . Valoarea  $b_{2,1}$  va fi  $\infty$  deoarece în această celulă nu se poate ajunge. Valorile celorlalte elemente  $b_{2,i}$  vor fi calculate pe baza formulei:  $b_{2,i} = a_{1,1} + a_{1,i} + a_{2,i}$ .

Pentru celelalte linii, valorile  $b_{ij}$  vor fi calculate pe baza formulei:

$$b_{i,j} = a_{i,j} + a_{i-1,j} + \min(b_{i-1,k}),$$

unde  $k$  variază între 1 și  $n$ . Evident, relația nu este valabilă pentru elementul de pe coloana  $k$  care corespunde minimului, deoarece nu se poate coborî direct, ci trebuie efectuat un salt orizontal. În această situație vom alege al doilea minim de pe linia anterioară.

În final alegem minimul valorilor de pe ultima linie a matricei  $B$  (fără a lua în considerare elementul de pe ultima coloană a acestei linii) la care adăugăm valoarea  $a_{mn}$ .

## 18.1.2 Cod sursă

Listing 18.1.1: lacusta.c

---

```

1  #include <stdio.h>
2  #define M 100
3
4  int main()
5  {
6      FILE *fi,*fo;
7      unsigned int a[M][M],b[M][M],m,n,i,j,min1,min2,jmin;
8      fi=fopen("lacusta.in", "rt");
9
10     fscanf(fi,"%u %u", &m, &n);
11     for(i=0; i<m; i++)
12         for(j=0; j<n; j++)
13             fscanf(fi,"%u",&a[i][j]);
14
15     fclose(fi);
16     b[1][0]=32000;
17     for(i=1; i<n; i++)
18         b[1][i]=a[0][0]+a[0][i]+a[1][i];
19     for(i=1; i<m-1; i++)
20     {
21         if(b[i][0]<=b[i][1])
22         {
23             min1=b[i][0];
24             min2=b[i][1];
25             jmin=0;
26         }
27         else
28         {
29             min1=b[i][1];
30             min2=b[i][0];
31             jmin=1;
32         }
33         for(j=2; j<n; j++)
34             if(b[i][j]<min1)
35             {
36                 min2=min1;
37                 min1=b[i][j];
38                 jmin=j;
39             }
40         else
41             if(b[i][j]<min2)
42                 min2=b[i][j];
43         for(j=0; j<n; j++)
44             if(j!=jmin)
45                 b[i+1][j]=min1+a[i][j]+a[i+1][j];
46         else
47             b[i+1][j]=a[i][j]+a[i+1][j]+min2;
48     }
49
50     min1=b[m-1][0];
51     for (j=1; j<n; j++)
52         if(b[m-1][j]<min1) min1=b[m-1][j];
53
54     fo=fopen("lacusta.out", "wt");
55     fprintf(fo,"%u\n", min1+a[m-1][n-1]);
56     fclose(fo);
57     return 0;
58 }
```

---

### 18.1.3 Rezolvare detaliată

Prima variantă:

Listing 18.1.2: lacusta1.java

---

```

1 import java.io.*;
2 class Lacusta1
3 {
4     static final int oo=100000;
5     static int m,n;
6     static int[][] a,b;    // 0 <= i <= m-1;    0 <= j <= n-1
7
8     public static void main(String[] args) throws IOException
9     {
10         long t1,t2;
11         t1=System.currentTimeMillis();
12
13         int i,j;
14
15         StreamTokenizer st=new StreamTokenizer(
16             new BufferedReader(new FileReader("lacusta.in")));
17         PrintWriter out=new PrintWriter(
18             new BufferedWriter(new FileWriter("lacusta.out")));
19
20         st.nextToken(); m=(int)st.nval;
21         st.nextToken(); n=(int)st.nval;
22
23         a=new int[m][n];
24         b=new int[m][n];
25
26         for(i=0;i<m;i++)
27             for(j=0;j<n;j++)
28             {
29                 st.nextToken(); a[i][j]=(int)st.nval;
30             }
31         for(i=0;i<m;i++) for(j=0;j<n;j++) b[i][j]=oo;
32
33         // prima linie (i=0) din b este oo
34         // a doua linie (i=1) din b
35         for(j=1;j<n;j++) b[1][j]=a[0][0]+a[0][j]+a[1][j];
36
37         // urmatoarele linii din b
38         for(i=2;i<m;i++)
39             for(j=0;j<n;j++)
40                 b[i][j]=a[i][j]+a[i-1][j]+minLinia(i-1,j);
41
42         // "obligatoriu" (!) si ultima linie (i=n-1) dar ... fara coborare
43         b[m-1][n-1]=minLinia(m-1,n-1)+a[m-1][n-1];
44
45         out.println(b[m-1][n-1]);
46         out.close();
47         t2=System.currentTimeMillis();
48         System.out.println("Timp = "+(t2-t1));
49     } // main()
50
51     static int minLinia(int ii, int jj) // min pe linia=ii fara pozitia jj==col
52     {
53         int j,min=oo;
54         for(j=0;j<n;j++)
55             if(j!=jj)
56                 if(b[ii][j]<min) min=b[ii][j];
57         return min;
58     } // minLinia(...)
59 } // class

```

---

A doua variantă:

Listing 18.1.3: lacusta2.java

---

```

1 import java.io.*;    // suplimentar ... si traseul !
2 class Lacusta2
3 {
4     static final int oo=100000;
5     static int m,n;

```

---



```

6  static int[][] a,b;    // 1 <= i <= m;    1 <= j <= n
7
8  public static void main(String[] args) throws IOException
9  {
10     long t1,t2;
11     t1=System.currentTimeMillis();
12
13     int i,j,min,jmin,j0;
14
15     StreamTokenizer st=new StreamTokenizer(
16         new BufferedReader(new FileReader("lacusta.in")));
17     PrintWriter out=new PrintWriter(
18         new BufferedWriter(new FileWriter("lacusta.out")));
19     st.nextToken(); m=(int)st.nval;
20     st.nextToken(); n=(int)st.nval;
21     a=new int[m+1][n+1];
22     b=new int[m+1][n+1];
23
24     for(i=1;i<=m;i++)
25     for(j=1;j<=n;j++)
26     {
27         st.nextToken(); a[i][j]=(int)st.nval;
28     }
29     for(i=1;i<=m;i++) for(j=1;j<=n;j++) b[i][j]=oo;
30
31     // prima linie (i=1) din b este oo
32     // a doua linie (i=2) din b
33     for(j=2;j<=n;j++) b[2][j]=a[1][1]+a[1][j]+a[2][j];
34
35     // urmatoarele linii din b
36     for(i=3;i<=m;i++)
37     for(j=1;j<=n;j++) b[i][j]=a[i][j]+a[i-1][j]+minLinia(i-1,j);
38
39     // "obligatoriu" (!) si ultima linie (i=n) dar ... fara coborare
40     b[m][n]=minLinia(m,n)+a[m][n];
41
42     out.println(b[m][n]);
43     out.close();
44
45     jmin=-1; // initializare aiurea !
46     j0=1;    // pentru linia 2
47     System.out.print(1+" "+1+" --> ");
48     for(i=2;i<=m-1;i++) // liniile 2 .. m-1
49     {
50         min=oo;
51         for(j=1;j<=n;j++)
52         if(j!=j0)
53         if(b[i][j]<min) { min=b[i][j]; jmin=j;}
54         System.out.print((i-1)+" "+jmin+" --> ");
55         System.out.print(i+" "+jmin+" --> ");
56         j0=jmin;
57     }
58
59     j0=n;
60     min=oo;
61     for(j=1;j<=n;j++)
62     if(j!=j0)
63     if(b[i][j]<min) { min=b[i][j]; jmin=j;}
64     System.out.print((i-1)+" "+jmin+" --> ");
65     System.out.print(i+" "+jmin+" --> ");
66     System.out.println(m+" "+n);
67
68     t2=System.currentTimeMillis();
69     System.out.println("Timp = "+(t2-t1));
70 } // main()
71
72 static int minLinia(int ii, int jj) // min pe linia=ii fara pozitia jj==col
73 {
74     int j,min=oo;
75     for(j=1;j<=n;j++)
76     if(j!=jj)
77     if(b[ii][j]<min) min=b[ii][j];
78     return min;
79 } // minLinia(...)
80 } // class

```

Varianta 3:

Listing 18.1.4: lacusta3.java

```

1 import java.io.*; // fara matricea de costuri (economie de "spatiu")
2 class Lacusta3 // traseul este ... pentru depanare
3 { // daca se cere ... se poate inregistra si apoi ...
4     static final int oo=100000;
5     static int m,n;
6     static int[][] a; // 1 <= i <= m;      1 <= j <= n
7
8     public static void main(String[] args) throws IOException
9     {
10         long t1,t2;
11         t1=System.currentTimeMillis();
12
13         int i,j;
14         int minc,minsol,jmin,j0;
15
16         StreamTokenizer st=new StreamTokenizer(
17             new BufferedReader(new FileReader("lacusta.in")));
18         PrintWriter out=new PrintWriter(
19             new BufferedWriter(new FileWriter("lacusta.out")));
20         st.nextToken(); m=(int)st.nval;
21         st.nextToken(); n=(int)st.nval;
22         a=new int[m+1][n+1];
23
24         for(i=1;i<=m;i++)
25             for(j=1;j<=n;j++)
26             {
27                 st.nextToken(); a[i][j]=(int)st.nval;
28             }
29
30         minsol=oo;
31         System.out.print(1+" "+1+" --> ");
32
33         // a doua linie (i=2)
34         minc=oo;
35         jmin=-1;
36         for(j=2;j<=n;j++)
37             if(a[1][1]+a[1][j]+a[2][j]<minc) {minc=a[1][1]+a[1][j]+a[2][j]; jmin=j;}
38         System.out.print(1+" "+jmin+" --> ");
39         System.out.print(2+" "+jmin+" --> ");
40         minsol=minc;
41         j0=jmin;
42
43         jmin=-1; // initializare aiurea !
44         for(i=3;i<=m-1;i++)
45         {
46             minc=oo;
47             for(j=1;j<=n;j++)
48                 if(j!=j0)
49                     if(a[i-1][j]+a[i][j]<minc)
50                         {minc=a[i-1][j]+a[i][j]; jmin=j;}
51             System.out.print((i-1)+" "+jmin+" --> ");
52             System.out.print(i+" "+jmin+" --> ");
53             minsol+=minc;
54             j0=jmin;
55         }
56
57         j0=n;
58         minc=oo;
59         for(j=1;j<=n;j++)
60             if(j!=j0)
61                 if(a[m-1][j]+a[m][j]<minc)
62                     {minc=a[m-1][j]+a[m][j]; jmin=j;}
63         System.out.print((m-1)+" "+jmin+" --> ");
64         System.out.print(m+" "+jmin+" --> ");
65         minsol+=minc+a[m][n];
66         System.out.println(m+" "+n);
67
68         out.println(minsol);
69         out.close();
70
71         t2=System.currentTimeMillis();
72         System.out.println("Timp = "+(t2-t1));

```

```

73     } // main()
74 } // class

```

Varianta 4:

Listing 18.1.5: lacusta4.java

```

1  import java.io.*; // fara matricea de costuri (economie de "spatiu")
2  class Lacusta4    // si ... fara matricea initiala (numai doua linii din ea !)
3  {                  // calculez pe masura ce citesc cate o linie !
4      static final int oo=100000;
5      static int m,n;
6      static int[][] a; // 1 <= i <= m;      1 <= j <= n
7
8      public static void main(String[] args) throws IOException
9      {
10         long t1,t2;
11         t1=System.currentTimeMillis();
12
13         int i,j,ii,jj;
14         int minc,minsol,jmin,j0;
15
16         StreamTokenizer st=new StreamTokenizer(
17             new BufferedReader(new FileReader("lacusta.in")));
18         PrintWriter out=new PrintWriter(
19             new BufferedWriter(new FileWriter("lacusta.out")));
20         st.nextToken(); m=(int)st.nval;
21         st.nextToken(); n=(int)st.nval;
22         a=new int[2][n+1];
23
24         for(i=1;i<=2;i++) // citesc numai primele doua linii
25             for(j=1;j<=n;j++)
26             {
27                 st.nextToken(); a[i%2][j]=(int)st.nval;
28             }
29
30         minsol=oo;
31         System.out.print(1+" "+1+" --> ");
32
33         // a doua linie (i=2)
34         minc=oo;
35         jmin=-1;
36         for(j=2;j<=n;j++)
37             if(a[1%2][1]+a[1][j]+a[2%2][j]<minc)
38                 {minc=a[1%2][1]+a[1][j]+a[2%2][j]; jmin=j;}
39         System.out.print(1+" "+jmin+" --> ");
40         System.out.print(2+" "+jmin+" --> ");
41         minsol=minc;
42         j0=jmin;
43
44         jmin=-1; // initializare aiurea !
45         for(i=3;i<=m-1;i++) // citesc mai departe cate o linie
46         {
47             for(j=1;j<=n;j++) { st.nextToken(); a[i%2][j]=(int)st.nval; }
48             minc=oo;
49             for(j=1;j<=n;j++)
50                 if(j!=j0)
51                     if(a[(i-1+2)%2][j]+a[i%2][j]<minc)
52                         {minc=a[(i-1+2)%2][j]+a[i%2][j]; jmin=j;}
53             System.out.print((i-1)+" "+jmin+" --> ");
54             System.out.print(i+" "+jmin+" --> ");
55             minsol+=minc;
56             j0=jmin;
57         }
58
59         //citesc linia m
60         for(j=1;j<=n;j++) { st.nextToken(); a[m%2][j]=(int)st.nval; }
61
62         j0=n;
63         minc=oo;
64         for(j=1;j<=n;j++)
65             if(j!=j0)
66                 if(a[(m-1+2)%2][j]+a[m%2][j]<minc)
67                     {minc=a[(m-1+2)%2][j]+a[m%2][j]; jmin=j;}
68         System.out.print((i-1)+" "+jmin+" --> ");
69         System.out.print(i+" "+jmin+" --> ");

```

```

70     minsol+=minc+a[m%2][n];
71     System.out.println(m+" "+n);
72
73     out.println(minsol);
74     out.close();
75
76     t2=System.currentTimeMillis();
77     System.out.println("Timp = "+(t2-t1));
78 } // main()
79 } // class

```

Varianta 5:

#### Listing 18.1.6: lacusta5.java

```

1  import java.io.*;    // numai o linie din matricea initiala !
2  class Lacusta5
3  {
4      static final int oo=100000;
5      static int m,n;
6      static int[] a;    // 1 <= j <= n
7
8      public static void main(String[] args) throws IOException
9      {
10         long t1,t2;
11         t1=System.currentTimeMillis();
12
13         int i,j,ii,jj,all,aa;
14         int minc,minsol,jmin,j0;
15
16         StreamTokenizer st=new StreamTokenizer(
17             new BufferedReader(new FileReader("lacusta.in")));
18         PrintWriter out=new PrintWriter(
19             new BufferedWriter(new FileWriter("lacusta.out")));
20         st.nextToken(); m=(int)st.nval;
21         st.nextToken(); n=(int)st.nval;
22         a=new int[n+1];
23
24         // citesc numai prima linie
25         for(j=1;j<=n;j++) { st.nextToken(); a[j]=(int)st.nval; }
26         System.out.print(1+" "+1+" --> ");
27         all=a[1];
28
29         // citesc a doua linie
30         st.nextToken(); a[1]=(int)st.nval;
31         minc=oo;
32         jmin=-1;
33         for(j=2;j<=n;j++)
34         {
35             aa=a[j];
36             st.nextToken(); a[j]=(int)st.nval;
37             if(aa+a[j]<minc) {minc=aa+a[j]; jmin=j;}
38         }
39         System.out.print(1+" "+jmin+" --> "+2+" "+jmin+" --> ");
40         minsol=all+minc;
41         j0=jmin;
42
43         // citesc mai departe cate o linie si ...
44         for(i=3;i<=m-1;i++)
45         {
46             minc=oo;
47             jmin=-1;
48             for(j=1;j<=n;j++)
49             {
50                 aa=a[j];
51                 st.nextToken(); a[j]=(int)st.nval;
52                 if(j!=j0) if(aa+a[j]<minc) {minc=aa+a[j]; jmin=j;}
53             }
54             System.out.print((i-1)+" "+jmin+" --> "+i+" "+jmin+" --> ");
55             minsol+=minc;
56             j0=jmin;
57         }
58
59         //citesc linia m (primele n-1 componente)
60         minc=oo;
61         jmin=-1;

```

```

62     for (j=1; j<=n-1; j++)
63     {
64         aa=a[j];
65         st.nextToken(); a[j]=(int)st.nval;
66         if (aa+a[j]<minc) {minc=aa+a[j]; jmin=j;}
67     }
68     System.out.print((m-1)+" "+jmin+" --> "+m+" "+jmin+" --> ");
69     minsol+=minc;
70     j0=jmin;
71
72     // citesc ultimul element
73     st.nextToken(); a[n]=(int)st.nval;
74     minsol+=a[n];
75     System.out.println(m+" "+n);
76
77     out.println(minsol);
78     out.close();
79
80     t2=System.currentTimeMillis();
81     System.out.println("Timp = "+(t2-t1));
82 } // main()
83 } // class

```

## 18.2 Scara

*Marinel Șerban, Emanuela Cerchez*

Ion și-a construit o vilă pe frumosul vârf al unui munte. Acum proiectează o scară specială, pe care va urca de la șosea până la vilă. Diferența de nivel dintre șosea și vilă este  $H$  (deci aceasta trebuie să fie înălțimea totală a scării). Scara va avea  $N$  trepte, toate de aceeași lățime, dar de înălțimi distincte două câte două.

Ion a sesizat că efortul pe care îl depune pentru a urca o treaptă este egal cu înălțimea treptei. Dar dacă el urcă  $x$  trepte deodată, efortul depus este egal cu media aritmetică a înălțimilor acestor  $x$  trepte pe care le urcă deodată + un efort de valoare constantă  $p$  (necesar pentru a-și lua avânt).

Fiind un tip athletic, Ion poate urca mai multe trepte deodată, dar suma înălțimilor treptelor urcate deodată nu trebuie să depășească o valoare maximă  $M$ .

### Cerință

Scrieți un program care să determine efortul minim necesar pentru a urca pe o scară construită conform restricțiilor problemei, precum și o modalitate de a construi scara care va fi urcată cu efort minim.

### Datele de intrare

Fișierul de intrare *scara.in* va conține pe prima linie 4 numere naturale separate prin câte un spațiu  $H$   $N$   $M$   $p$  (cu semnificația din enunț).

### Datele de ieșire

Fișierul de ieșire *scara.out* va conține

- pe prima linie va fi scris efortul minim necesar (cu 2 zecimale cu rotunjire);
- pe cea de a doua linie vor fi scrise  $N$  numere naturale nenule care reprezintă înălțimile celor  $N$  trepte ale scării (în ordinea de la șosea către vilă), separate prin câte un spațiu.

### Restricții și precizări

- $0 < H \leq 75$
- $0 < N \leq 8$
- $0 < M < 14$
- $0 \leq p \leq 10$
- Pentru datele de test, problema are întodeauna soluție.
- Dacă există mai multe soluții (modalități de a construi scara astfel încât să obțineți efortul minim dorit), veți afișa prima soluție în ordine lexicografică.
- Spunem că vectorul  $x = (x_1, x_2, \dots, x_k)$  precedă în ordine lexicografică vectorul  $y = (y_1, y_2, \dots, y_k)$  dacă există  $i \geq 1$  astfel încât  $x_j = y_j$ , pentru orice  $j < i$  și  $x_i < y_i$ .
- Dacă a doua zecimală a efortului minim este 0, sau chiar ambele zecimale sunt 0 nu este necesar să le afișați. Deci în exemplu s-ar fi putut scrie efortul minim 9 sau 9.0.

### Punctaj

Se acordă 40% din punctaj pentru prima cerință (efortul minim).

Dacă efortul minim este corect și se afișează și o soluție corectă (care respectă restricțiile problemei și corespunde efortului minim), dar această soluție nu este prima din punct de vedere lexicografic, se obține 80% din punctaj.

Pentru rezolvarea corectă și completă a ambelor cerințe se obține 100% din punctaj.

#### Exemple

scara.in	scara.out
10 4 5 2	9.00
	1 4 2 3

**Timp maxim de executare:** 5 secunde/test

### 18.2.1 Indicații de rezolvare

*Ginjo nr. 15/3 martie 2005*

Pentru rezolvarea acestei probleme vom utiliza *metoda backtracking*.

Pentru aceasta vom construi un vector  $a$  care va conține înălțimile scărilor, precum și un vector  $sol$  care va conține, în fiecare moment, cea mai bună soluție.

Deasemenea, vom construi un vector  $b$  care va conține efortul care trebuie depus pentru a urca scările (al  $i$ -lea element al acestui vector indică efortul necesar pentru a urca primele  $i$  scări).

Pentru a ne asigura că înălțimile sunt distincte vom păstra un vector de valori logice în care se vor marca înălțimile folosite în fiecare moment.

Deasemenea, la fiecare pas, va fi păstrată suma totală a înălțimilor scărilor construite la pașii anteriori.

În timpul algoritmului, trebuie să ne asigurăm că valorile vectorului  $b$  sunt cuprinse între 1 și  $M$  și sunt distincte. Deasemenea, suma lor trebuie să fie egală cu  $H$ .

La fiecare pas  $i$ , valoarea  $b_i$  va fi calculată pe baza formulei:

$$b_i = \min(b_{i-1} + a_i, \min(b_j + p + (a_{j+1} + a_{j+2} + \dots + a_i) / (i - j)))$$

unde  $j$  variază între 1 și  $i - 2$ , iar sumele de forma  $a_{j+1} + a_{j+2} + \dots + a_i$  sunt luate în considerare numai dacă sunt cel mult egale cu  $M$ .

Dacă am reușit să ajungem la ultimul pas, vom verifica dacă soluția curentă este mai bună decât cea considerată anterior a fi cea mai bună. În acest caz atât vectorul  $sol$ , precum și efortul total minim, sunt actualizate.

În final, vom afișa efortul minim obținut, precum și elementele vectorului  $sol$ .

### 18.2.2 Cod sursă

Listing 18.2.1: scara.cpp

```

1 #include <fstream>
2 #include <iomanip>
3
4 using namespace std;
5
6 #define InFile "scara.in"
7 #define OutFile "scara.out"
8 #define NMax 101
9 #define HMax 101
10
11 int sol[NMax], solmin[NMax], uz[HMax], htot;
12 double ef[NMax], efmin;
13 //ef[i]=efortul minim cu care urc primele i trepte din sol
14 int N, H, p, M;
15
16 void Citire();
17 void Gen(int);
18 void Afișare();
19 double efort ();
20
21 int main()
22 {
```

```

23 Citire();
24 efmin=(double)H*N+1;
25 Gen(1);
26 Afisare();
27 return 0;
28 }
29
30 void Citire()
31 {ifstream fin(InFile);
32  fin>>H>>N>>M>>p;
33  fin.close();}
34
35 void Afisare()
36 {
37  int i;
38  ofstream fout(OutFile);
39  fout<<setprecision(2)<<efmin<<endl;
40  for (i=1; i<N; i++)
41      fout<<solmin[i]<<' ';
42  fout<<solmin[N]<<endl;
43  fout.close();
44  }
45
46 double efort()
47 {
48  int k, j;
49  double x, sum;
50  for (k=1; k<=N; k++)
51      {x=sol[k]+ef[k-1]; // urc cu efort minim primele k-1 trepte
52       // si apoi urc treapta k de inaltime i
53       sum=sol[k];
54       for (j=2; k-j>=0; j++)
55           { // urc deodata j trepte k, k-1, ..., k-j+1, daca
56             // suma inaltimeilor lor este <=M
57             // in sum calculez suma inaltimeilor
58             sum+=sol[k-j+1];
59             if (sum>M) break;
60             if (sum/j+p+ef[k-j]<x)
61                 x=sum/j+ef[k-j]+p;
62           }
63       ef[k]=x;
64  }
65  return ef[N];
66  }
67
68 void Gen(int k)
69 {
70  int i;
71  double x;
72  if (k==N+1)
73      {
74      if (htot==H)
75          {x=efort();
76           if (x<efmin && efmin-x>0.001)
77               {efmin=x;
78                for (i=1; i<=N; i++) solmin[i]=sol[i]; }
79          }
80      }
81  else
82      for (i=1; i<=H && htot+i<=H && i<=M; i++)
83          if (!uz[i])
84              {
85              //care ar fi efortul minim daca as pune inaltimea treptei k=i?
86              sol[k]=i; htot+=i; uz[i]=1;
87              Gen(k+1);
88              uz[i]=0; htot-=i;
89              }
90  }

```

### 18.2.3 Rezolvare detaliată

```

1  import java.io.*;           // 1+2+...+(n-1)=(n-1)n/2 ==> hmax=h-(n-1)n/2 !!!
2  class Scara                 // aranjamente 1,2,...,hmax pe n pozitii
3  {                           // cu restrictia: suma elementelor = h (backtracking)
4      static int h,n,m,p;     // hmaxOK=min(hmax,m)      !!!
5      static int hmax,s;      // timp mare testele 3 si 4 ==> 13.5 sec
6      static int[] x,xsol;    // Java = de 5 ori mai lent ???
7      static double[] e;      // si calculatorul meu e lent !!!
8      static double esol;
9      static boolean[] ales;   // elimin cautarea ==> timp injumatatit !!
10     static final int oo=8*13+1; // infinit !!!
11
12     public static void main(String[] args) throws IOException
13     {
14         long t1,t2;
15         t1=System.currentTimeMillis();
16
17         StreamTokenizer st=new StreamTokenizer(
18             new BufferedReader(new FileReader("1-scara.in")));
19         PrintWriter out=new PrintWriter(
20             new BufferedWriter(new FileWriter("scara.out")));
21
22         st.nextToken(); h=(int)st.nval;
23         st.nextToken(); n=(int)st.nval;
24         st.nextToken(); m=(int)st.nval;
25         st.nextToken(); p=(int)st.nval;
26
27         x=new int[n+1];
28         e=new double[n+1];
29         xsol=new int[n+1];
30
31         hmax=h-((n-1)*n)/2;
32         if(m<hmax) hmax=m; // restrictie ptr mai multe trepte ==> si pentru una !
33
34         ales=new boolean[hmax+1];
35
36         esol=oo;
37         s=0;
38         f(1);
39
40         out.println(((double)((int)(esol*100+0.5))/100)); // 2 zecimale cu rotunjire!!!
41         for(int k=1;k<=n;k++) out.print(xsol[k]+" ");
42         out.println();
43         out.close();
44         t2=System.currentTimeMillis();
45         System.out.println("Timp = "+(t2-t1));
46     } // main()
47
48     static void f(int k)      // plasez valori pe pozitia k
49     {
50         int i,j;
51         for(i=1;i<=hmax;i++)
52         {
53             if(alles[i]) continue;
54             if(s+i>h) break; // e deja prea mult !!!
55
56             x[k]=i;
57             alles[i]=true;
58             s=s+i;
59
60             if(k<n) f(k+1); else efort();
61
62             s=s-i;
63             alles[i]=false;
64         }
65     } // f(...)
66
67     static void efort()
68     {
69         if(s!=h) return;
70
71         int i,j,k,sij;
72         double e1,e2;
73
74         for(i=1;i<=n;i++)
75         {

```



```
76     e1=e[i-1]+x[i];      // pas=o treapta ==> efort=inaltime treapta
77     e2=oo;
78     sij=x[i]+x[i-1];     // pas=mai multe trepte ==> p+media_aritmetica
79     j=i-1;
80     while((j>=1)&&(sij<=m))
81     {
82         if(e[j-1]+p+(double)sij/(i-j+1)<e2)
83             e2=e[j-1]+p+(double)sij/(i-j+1);
84         j--;
85         sij+=x[j];
86     }
87     if(e1<e2) e[i]=e1; else e[i]=e2;
88 }//for i
89
90 if(e[n]<esol-0.001)
91 {
92     esol=e[n];
93     for(k=1;k<=n;k++) xsol[k]=x[k];
94 }
95 }// efort()
96 }//class
```

---

# Capitolul 19

## OJI 2004

### 19.1 Perle

Granița nu se trece ușor. Asta pentru că Balaurul Arhirel (mare pasionat de informatică) nu lasă pe nimeni să treacă decât după ce răspunde la niște întrebări ...

În acea țară există trei tipuri de perle normale (le vom nota cu 1, 2 și 3) și trei tipuri de perle magice (le vom nota cu  $A$ ,  $B$  și  $C$ ). Perlele magice sunt deosebite prin faptul că se pot transforma în alte perle (una sau mai multe, normale sau magice).

Perla magică de tipul  $A$  se poate transforma în orice perlă normală (una singură).

Perla magică de tipul  $B$  se poate transforma într-o perlă normală de tipul 2 și una magică de tipul  $B$ , sau într-o perlă normală de tipul 1, una magică de tipul  $A$ , una normală de tipul 3, una magică de tipul  $A$  și una magică de tipul  $C$ .

Perla magică de tipul  $C$  se poate transforma într-o perlă normală de tipul 2 sau într-o perlă normală de tipul 3, una magică de tipul  $B$  și una magică de tipul  $C$  sau într-o perlă normală de tipul 1, una normală de tipul 2 și una magică de tipul  $A$ .

Ca să rezumăm cele de mai sus putem scrie:

$A$	$\longrightarrow$	1		2		3
$B$	$\longrightarrow$	$2B$		$1A3AC$		
$C$	$\longrightarrow$	2		$3BC$		$12A$

Balaurul Arhirel ne lasă la început să ne alegem o perlă magică (una singură), iar apoi folosind numai transformările de mai sus trebuie să obținem un anumit șir de perle normale. Când o perlă magică se transformă, perlele din stânga și din dreapta ei rămân la fel (și în aceeași ordine). De asemenea ordinea perlelor rezultate din transformare este chiar cea prezentată mai sus.

De exemplu, dacă balaurul ne cere să facem șirul de perle 21132123, putem alege o perlă magică de tipul  $B$  și următorul șir de transformări:

$$B \longrightarrow 2B \longrightarrow 21A3AC \longrightarrow 21A3A12A \longrightarrow 21132123.$$

Întrucât Balaurul nu are prea multă răbdare, el nu ne cere decât să spunem dacă se poate sau nu obține șirul respectiv de perle.

#### Cerință

Să se determine pentru fiecare șir de intrare dacă se poate obține prin transformările de mai sus sau nu (alegând orice primă perlă magică, la fiecare șir).

#### Datele de intrare

Fișierul de intrare **perle.in** are următoarea structură:

- pe prima linie numărul  $N$ , reprezentând numărul de șiruri din fișierul de intrare
- urmează  $N$  linii; a  $i$ -a linie dintre cele  $N$  descrie șirul  $i$ , printr-o succesiune de numere naturale despărțite de câte un spațiu. Primul număr reprezintă lungimea șirului  $L_i$ , iar următoarele  $L_i$  numere sunt tipurile de perle normale, în ordine, de la stânga la dreapta.

#### Datele de ieșire

Fișierul **perle.out** va conține  $N$  linii. Pe linia  $i$  se va scrie un singur număr 1 sau 0 (1 dacă se poate obține șirul respectiv (al  $i$ -lea) și 0 dacă nu se poate).

#### Restricții și precizări

- $0 < N < 11$

- $0 < L_i < 10001$ , pentru oricare  $i$

**Exemplu**

perle.in	perle.out
3	1
8 2 1 1 3 2 1 2 3	0
2 2 2	1
1 3	

**Timp maxim de executare:** 1 secundă/test

**19.1.1 Indicații de rezolvare**

*Mihai Stroe, Ginfo nr. 14/4 aprilie 2004*

Inițial, problema pare dificilă, dar, după examinarea transformărilor posibile, se observă că este destul de simplă.

Fie un anumit șir de perle. Presupunem că acesta a fost obținut dintr-o perlă magică și ne oprim la prima contradicție.

Să determinăm, la început, perla magică din care s-ar putea obține șirul.

Dacă șirul are lungimea 1, el se poate obține dintr-o perlă magică A.

Dacă șirul începe cu 12 și are trei perle, s-ar putea obține dintr-o perlă C.

Dacă șirul începe cu 2 și are cel puțin două perle, singura șansă de a-l obține este de a începe cu o perlă magică de tip B. Similar dacă începe cu 1 și are mai mult de trei perle.

Dacă șirul începe cu 3, s-ar putea obține numai dintr-o perlă de tip C.

Acum, având în vedere că am stabilit singura perlă magică din care sunt șanse să se obțină șirul, vom verifica dacă șirul poate fi într-adevăr obținut.

Pentru aceasta, putem scrie câte o funcție pentru fiecare perlă magică. O astfel de funcție se aplică pe șirul de intrare și execută operații (avansări pe șir sau apeluri ale funcțiilor pentru alte tipuri de perle) în funcție de tipul perlelor întâlnite.

Dacă la un moment dat nu există nici o regulă de continuare (de exemplu, pentru șirul 22), șirul nu poate fi obținut.

Funcția pentru A nu este necesară, tratarea perlei magice A putând fi inclusă în celelalte două funcții. Se observă că, în cadrul fiecărei funcții, acțiunile efectuate sunt alese determinist. Astfel, la fiecare pas se alege o acțiune posibilă sau se întrerupe căutarea și se semnalează faptul că șirul nu poate fi obținut.

Funcțiile se apelează recursiv. Pentru a evita depășirea stivei, recursivitatea se poate simula iterativ.

**Analiza complexității**

La fiecare moment, deciziile se iau în timp constant. Ordinul de complexitate al operației de citire a datelor de intrare este  $O(L)$ , unde  $L$  reprezintă suma lungimilor șirurilor de perle din fișierul de intrare.

Fiecare caracter din fiecare șir este parcurs o singură dată.

Ordinul de complexitate al algoritmului este  $O(L_i)$  pentru un șir de perle de lungime  $L_i$ .

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este  $O(L)$ .

**19.1.2 Cod sursă**

Listing 19.1.1: perle.c

```

1 #include <stdio.h>
2 #define NMAX 10000
3 int st[2][NMAX+10]; /* 1, 2, 3 e evident si 4, 5, 6 e A, B, C */
4
5 int main(void)
6 {
7     int t,l,v,i;
8     int niv[2];
9     int valid[2];
10
11     freopen("perle.in","r",stdin);
12     freopen("perle.out","w",stdout);
13

```

```

14     scanf("%d",&t);
15     while (t--)
16     {
17         scanf("%d",&l);
18         if (l == 1)
19         {
20             scanf("%d",&v);
21             printf("1\n");
22             continue;
23         }
24         else
25         {
26             valid[1] = valid[0] = 1;
27             niv[1] = niv[0] = 1;
28             st[0][0] = 5;
29             st[1][0] = 6;
30             while (l--)
31             {
32                 scanf("%d",&v);
33                 for(i = 0; i < 2; i++) /* trist dar doar 2 valori vreau */
34                 {
35                     if (valid[i])
36                     {
37                         if (st[i][niv[i]-1] < 4)
38                         {
39                             valid[i] = (st[i][--niv[i]]==v);
40                             continue;
41                         }
42
43                         if (st[i][niv[i]-1] == 4)
44                         {
45                             niv[i]--;
46                             continue;
47                         }
48
49                         if (st[i][niv[i]-1]==5)
50                         {
51                             if (v == 3) valid[i] = 0;
52                             if (v == 1)
53                             {
54                                 niv[i]--;
55                                 st[i][niv[i]++] = 6;
56                                 st[i][niv[i]++] = 4;
57                                 st[i][niv[i]++] = 3;
58                                 st[i][niv[i]++] = 4;
59                             }
60
61                             /* 2-ul e ok :D */
62                             continue;
63                         }
64
65                         /* e C in stiva */
66                         if (v == 2) niv[i]--;
67                         if (v == 1)
68                         {
69                             niv[i]--;
70                             st[i][niv[i]++] = 4;
71                             st[i][niv[i]++] = 2;
72                         }
73
74                         if (v == 3)
75                         {
76                             niv[i]--;
77                             st[i][niv[i]++] = 6;
78                             st[i][niv[i]++] = 5;
79                         }
80                     }
81                     if (!niv[i] && l)
82                         valid[i] = 0;
83                 }
84             }
85
86             printf("%d\n",((!niv[0] && valid[0]) || (!niv[1] && valid[1])));
87         }
88     }
89

```

```

90     return 0;
91 }

```

### 19.1.3 Rezolvare detaliată

Listing 19.1.2: perle.java

```

1  import java.io.*;
2  class Perle
3  {
4      static final int a=4, b=5, c=6;
5      static byte[] x,y;
6      static int n,lg,j1,j2;
7      static boolean ok;
8
9      public static void main(String[] args) throws IOException
10     {
11         long t1,t2;
12         t1=System.currentTimeMillis();
13         int k,i;
14
15         StreamTokenizer st=new StreamTokenizer(
16             new BufferedReader(new FileReader("perle.in")));
17         PrintWriter out=new PrintWriter(
18             new BufferedWriter(new FileWriter("perle.out")));
19         st.nextToken(); n=(int)st.nval;
20         for(k=1;k<=n;k++)
21         {
22             st.nextToken(); lg=(int)st.nval;
23             x=new byte[lg+5];
24             y=new byte[lg+5];
25             for(i=1;i<=lg;i++) {st.nextToken(); x[i]=(byte)st.nval;}
26
27             ok=true;
28             determinStart();
29             if(!ok) {out.println(0); continue;}
30
31             //afis();
32             j1=1;
33             j2=2;                // prima pozitie libera
34
35             while((ok)&&(j1<=lg))
36             {
37                 cautNeterminal();
38                 if(ok) schimbNeterminal();
39                 if(j2>lg+1) ok=false;
40                 //afis();
41             }
42
43             if(ok) out.println(1); else out.println(0);
44         } // for k
45
46         out.close();
47         t2=System.currentTimeMillis();
48         System.out.println("Timp = "+(t2-t1));
49     } // main()
50
51     static void schimbNeterminal()
52     {
53         if(y[j1]==a) {y[j1]=x[j1]; j1++;}
54         else
55             if(y[j1]==b)
56             {
57                 if(x[j1]==1) inserez1A3AC();
58                 else if(x[j1]==2) inserez2B();
59                 else ok=false;
60             }
61         else // y[j1]==c
62         {
63             if(x[j1]==1) inserez12A();
64             else if(x[j1]==2) {y[j1]=2; j1++;}
65             else inserez3BC();

```

```

66     }
67 } // schimbNeterminal()
68
69 static void cautNeterminal()
70 {
71     while((j1<j2) && (y[j1]<a) && ok)
72     {
73         if(y[j1]!=x[j1]) ok=false;
74         j1++;
75     }
76 } // cautNeterminal()
77
78 static void inserez1A3AC()
79 {
80     int j;
81     j2=j2+4;
82     for(j=j2-1; j>=j1+4; j--) y[j]=y[j-4];
83     y[j1+0]=1;
84     y[j1+1]=a;
85     y[j1+2]=3;
86     y[j1+3]=a;
87     y[j1+4]=c;
88 } // inserez1A3AC()
89
90 static void inserez2B()
91 {
92     int j;
93     j2=j2+1;
94     for(j=j2-1; j>=j1+1; j--) y[j]=y[j-1];
95     y[j1+0]=2;
96     y[j1+1]=b;
97 } // inserez2B()
98
99 static void inserez12A()
100 {
101     int j;
102     j2=j2+2;
103     for(j=j2-1; j>=j1+2; j--) y[j]=y[j-2];
104     y[j1+0]=1;
105     y[j1+1]=2;
106     y[j1+2]=a;
107 } // inserez12A()
108
109 static void inserez3BC()
110 {
111     int j;
112     j2=j2+2;
113     for(j=j2-1; j>=j1+2; j--) y[j]=y[j-2];
114     y[j1+0]=3;
115     y[j1+1]=b;
116     y[j1+2]=c;
117 } // inserez3BC()
118
119 static void determinStart() // determin "neterminalul" de start
120 {
121     if(x[1]==1)
122     {
123         if(lg==1) y[1]=a;
124         else if((lg==3) && (x[2]==2)) y[1]=c;
125         else if(lg>=5) y[1]=b;
126         else ok=false;
127     }
128     else
129     if(x[1]==2)
130     {
131         if(lg==1) y[1]=a;
132         else if(lg>=2) y[1]=b;
133         else ok=false;
134     }
135     else
136     if(x[1]==3)
137     {
138         if(lg==1) y[1]=a;
139         else if(lg>=3) y[1]=c;
140         else ok=false;
141     }

```

```

142     } // determinStart()
143
144     static void afis()
145     {
146         afisv(x); afisv(y); System.out.println();
147     }
148
149     static void afisv(byte[] v)
150     {
151         int i;
152         for(i=1; i<v.length; i++)
153             if(v[i]==a) System.out.print('A');
154             else if(v[i]==b) System.out.print('B');
155             else if(v[i]==c) System.out.print('C');
156             else if(v[i]==0) System.out.print(" ");
157             else System.out.print(v[i]);
158         System.out.println();
159     } // afisv(...)
160 } // class

```

## 19.2 Romeo și Julieta

În ultima ecranizare a celebrei piese shakespeariene Romeo și Julieta trăiesc într-un oraș modern, comunică prin e-mail și chiar învață să programeze. Într-o secvență tulburătoare sunt prezentate frământările interioare ale celor doi eroi încercând fără succes să scrie un program care să determine un punct optim de întâlnire.

Ei au analizat harta orașului și au reprezentat-o sub forma unei matrice cu  $n$  linii și  $m$  coloane, în matrice fiind marcate cu spațiu zonele prin care se poate trece (străzi lipsite de pericole) și cu  $X$  zonele prin care nu se poate trece. De asemenea, în matrice au marcat cu  $R$  locul în care se află locuința lui Romeo, iar cu  $J$  locul în care se află locuința Julietei.

Ei se pot deplasa numai prin zonele care sunt marcate cu spațiu, din poziția curentă în oricare dintre cele 8 poziții învecinate (pe orizontală, verticală sau diagonală).

Cum lui Romeo nu îi place să aștepte și nici să se lase așteptat n-ar fi tocmai bine, ei au hotărât că trebuie să aleagă un punct de întâlnire în care atât Romeo, cât și Julieta să poată ajunge în același timp, plecând de acasă. Fiindcă la întâlniri amândoi vin într-un suflet, ei estimează timpul necesar pentru a ajunge la întâlnire prin numărul de elemente din matrice care constituie drumul cel mai scurt de acasă până la punctul de întâlnire. Și cum probabil există mai multe puncte de întâlnire posibile, ei vor să îl aleagă pe cel în care timpul necesar pentru a ajunge la punctul de întâlnire este minim.

### Cerință

Scrieți un program care să determine o poziție pe hartă la care Romeo și Julieta pot să ajungă în același timp. Dacă există mai multe soluții, programul trebuie să determine o soluție pentru care timpul este minim.

### Datele de intrare

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

- pe prima linie numerele naturale  $NM$ , care reprezintă numărul de linii și respectiv de coloane ale matricei, separate prin spațiu;
- pe fiecare dintre următoarele  $N$  linii se află  $M$  caractere (care pot fi doar  $R$ ,  $J$ ,  $X$  sau spațiu) reprezentând matricea.

### Datele de ieșire

Fișierul de ieșire **rj.out** va conține o singură linie pe care sunt scrise trei numere naturale separate prin câte un spațiu  $tmin\ x\ y$ , având semnificația:

- $x\ y$  reprezintă punctul de întâlnire ( $x$  - numărul liniei,  $y$  - numărul coloanei);
- $tmin$  este timpul minim în care Romeo (respectiv Julieta) ajunge la punctul de întâlnire.

### Restricții și precizări

- $1 < N, M < 101$
- Liniile și coloanele matricei sunt numerotate începând cu 1.
- Pentru datele de test există întotdeauna soluție.

### Exemple

**rj.in**

```

5 8
XXR XXX
 X X X
J X X X
      XX
XXX XXXX

```

**rj.out**

```
4 4 4
```

*Explicație:*

Traseul lui Romeo poate fi: (1,3), (2,4), (3,4), (4,4). Timpul necesar lui Romeo pentru a ajunge de acasă la punctul de întâlnire este 4.

Traseul Julietei poate fi: (3,1), (4,2), (4,3), (4,5). Timpul necesar Julietei pentru a ajunge de acasă la punctul de întâlnire este de asemenea 4.

În plus 4, este punctul cel mai apropiat de ei cu această proprietate.

**Timp maxim de executare:** 1 secundă/test

### 19.2.1 Indicații de rezolvare

*Mihai Stroe, Ginfo nr. 14/4 aprilie 2004*

Problema se rezolvă folosind *algoritmul lui Lee*.

Se aplică acest algoritm folosind ca puncte de start poziția lui Romeo și poziția Julietei.

Vom folosi o matrice  $D$  în care vom pune valoarea 1 peste tot pe unde nu se poate trece, valoarea 2 în poziția în care se află Romeo inițial, valoarea 3 în poziția în care se află Julieta inițial și valoarea 0 în rest.

La fiecare pas  $k$  vom parcurge această matrice și în pozițiile vecine celor care au valoarea 2 vom pune valoarea 2, dacă acestea au valoarea 0 sau 3. Dacă o poziție are valoare 3, înseamnă că la un moment de timp anterior Julieta se putea afla în poziția respectivă. La același pas  $k$  vom mai parcurge matricea o dată și în pozițiile vecine celor care au valoarea 3 vom pune valoarea 3, dacă acestea au valoarea 0.

Dacă la pasul  $k$  poziția vecină uneia care are valoarea 3, are valoarea 2, atunci ne vom opri și  $k$  reprezintă momentul minim de timp după care cei doi se întâlnesc, iar poziția care are valoare 2 reprezintă locul întâlnirii.

La prima vedere s-ar părea că numărul  $k$  nu reprezintă momentul de timp minim la care cei doi se întâlnesc. Vom demonstra că algoritmul este corect prin metoda reducerii la absurd. Pentru aceasta avem în vedere că pozițiile marcate cu 2 reprezintă toate locurile în care se poate afla Romeo după cel mult  $k$  pași, iar cele marcate cu 3 reprezintă toate locurile în care se poate afla Julieta după cel mult  $k$  pași. Dacă  $k$  nu reprezintă momentul de timp minim la care cei doi se întâlnesc înseamnă că acesta a fost determinat mai devreme și algoritmul s-a oprit deja.

#### Analiza complexității

Ordinul de complexitate al operației de citire a datelor de intrare este  $O(M \times N)$ .

Ordinul de complexitate al acestui algoritm este  $O(k \times M \times N)$ , unde  $k$  reprezintă momentul în care cei doi se întâlnesc.

Ordinul de complexitate al operației de scriere a rezultatului este  $O(1)$ .

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este  $O(k \times M \times N)$ .

### 19.2.2 Cod sursă

Listing 19.2.1: rj.cpp

```

1 #include <fstream>
2
3 using namespace std;
4
5 #define InFile "rj.in"
6 #define OutFile "rj.out"
7 #define NMax 102
8 #define NV 8

```



```

9
10 int n, m, xr, yr, xj, yj;
11 int dl[NV]={0, 1, 0, -1, -1, 1, -1, 1};
12 int dc[NV]={1, 0, -1, 0, -1, 1, 1, -1};
13 char l[NMax][NMax];
14 int r[NMax][NMax];
15
16 void citire(void);
17 void afisare(int [NMax][NMax]);
18 void parcurge (int, int, int[NMax][NMax]);
19
20 int main()
21 {
22     int j[NMax][NMax];
23     citire();
24     parcurge(xr, yr, r);
25     parcurge(xj, yj, j);
26     afisare(j);
27     return 0;
28 }
29
30 void citire(void)
31 {
32     int i, k;
33     char c;
34
35     ifstream f(InFile);
36     f>>n>>m;
37
38     for (i=0; i<=n+1; i++) l[i][0]=l[i][m+1]='X';
39     for (i=0; i<=m+1; i++) l[0][i]=l[n+1][i]='X';
40     f.get(c);
41
42     for (i=1; i<=n; i++)
43     {
44         for (k=1; k<=m; k++)
45         {
46             f.get(c);
47             l[i][k]=c;
48             if (l[i][k]=='R') {xr=i; yr=k; l[i][k]=' ';}
49             if (l[i][k]=='J') {xj=i; yj=k; l[i][k]=' ';}
50         }
51         f.get(c);
52     }
53     f.close();
54 }
55
56 void parcurge (int x0, int y0, int d[NMax][NMax])
57 {
58     struct Punct
59     {
60         int l, c;
61     } C[NMax*NMax], p;
62
63     int inc=0, sf=0, i, k;
64     for (i=0; i<=n+1; i++)
65         for (k=0; k<=m+1; k++)
66             d[i][k]=-1;
67
68     C[0].l=x0;
69     C[0].c=y0;
70     d[x0][y0]=1;
71     while (inc<=sf)
72     {
73         p=C[inc++];
74         for (i=0; i<NV; i++)
75             if (l[p.l+dl[i]][p.c+dc[i]]==' ' && d[p.l+dl[i]][p.c+dc[i]]==-1)
76             {
77                 d[p.l+dl[i]][p.c+dc[i]]=1+d[p.l][p.c];
78                 C[++sf].l=p.l+dl[i];
79                 C[sf].c=p.c+dc[i];
80             }
81     }
82 }
83
84 void afisare(int j[NMax][NMax])

```

```

85 {
86     ofstream f(OutFile);
87     int tmin=NMax*NMax+5, xmin=-1, ymin=-1, i, k;
88
89     for (i=1; i<=n; i++)
90         for (k=1; k<=m; k++)
91             if (r[i][k]==j[i][k])
92                 if (r[i][k]<tmin && r[i][k]!=-1)
93                     {
94                         tmin=r[i][k];
95                         xmin=i;
96                         ymin=k;
97                     }
98     f<<tmin<<' ' <<xmin<<' ' <<ymin<<endl;
99     f.close();
100 }

```

### 19.2.3 Rezolvare detaliată

Listing 19.2.2: rj.java

```

1  import java.io.*;
2  class RJ
3  {
4      static final int zid=10000;
5      static int m,n;
6      static int[][] xr,xj;
7      static int[] qi=new int[5000]; // coada sau coada circulara mai bine !
8      static int[] qj=new int[5000]; // coada
9
10     static int ic, sc; // ic=inceput coada
11
12     public static void main(String[] args) throws IOException
13     {
14         long t1,t2;
15         t1=System.currentTimeMillis();
16
17         int i,j,ir=0,jr=0,ij=0,jj=0, imin, jmin, tmin;
18         String s;
19
20         BufferedReader br=new BufferedReader(new FileReader("rj.in"));
21         StreamTokenizer st=new StreamTokenizer(br);
22         PrintWriter out=new PrintWriter(
23             new BufferedWriter(new FileWriter("rj.out")));
24
25         st.nextToken(); m=(int)st.nval;
26         st.nextToken(); n=(int)st.nval;
27
28         xr=new int[m+2][n+2]; // matrice bordata cu zid !
29         xj=new int[m+2][n+2]; // matrice bordata cu zid !
30
31         br.readLine(); // citeste CRLF !!!
32         for (i=1; i<=m; i++)
33         {
34             s=br.readLine();
35             for (j=1; j<=n; j++)
36                 if (s.charAt(j-1)=='X') xr[i][j]=xj[i][j]=zid; // zid!
37                 else if (s.charAt(j-1)=='R') {ir=i; jr=j; xr[i][j]=1;}
38                 else if (s.charAt(j-1)=='J') {ij=i; jj=j; xj[i][j]=1;}
39         }
40
41         for (i=0; i<=m+1; i++) xr[i][0]=xr[i][n+1]=xj[i][0]=xj[i][n+1]=zid; // E si V
42         for (j=0; j<=n+1; j++) xr[0][j]=xr[m+1][j]=xj[0][j]=xj[m+1][j]=zid; // N si S
43
44         ic=sc=0; // coada vida;
45         qi[sc]=ir; qj[sc]=jr; sc++; // (ir,jr) --> coada
46         while (ic!=sc)
47         {
48             i=qi[ic]; j=qj[ic]; ic++; // scot din coada
49             fill(xr,i,j);
50         }
51     }

```

---

```

52     ic=sc=0;                                // coada vida;
53     qi[sc]=ij;  qj[sc]=jj;  sc++; // (ij,jj) --> coada
54     while(ic!=sc)
55     {
56         i=qi[ic];  j=qj[ic];  ic++; // scot din coada
57         fill(xj,i,j);
58     }
59
60     tmin=10000;
61     imin=jmin=0;
62     for(i=1;i<=m;i++)
63         for(j=1;j<=n;j++)
64             if(xr[i][j]==xj[i][j])
65                 if(xj[i][j]!=0) // pot exista pozitii ramase izolate !
66                     if(xr[i][j]<tmin) {tmin=xr[i][j]; imin=i; jmin=j;}
67
68     out.println(tmin+" "+imin+" "+jmin);
69     out.close();
70     t2=System.currentTimeMillis();
71     System.out.println("Timp = "+(t2-t1));
72 } // main()
73
74 static void fill(int[][] x,int i, int j)
75 {
76     int t=x[i][j]; // timp
77     if(x[i-1][j]==0) {x[i-1][j]=t+1;  qi[sc]=i-1; qj[sc]=j;  sc++;} // N
78     if(x[i-1][j+1]==0) {x[i-1][j+1]=t+1; qi[sc]=i-1; qj[sc]=j+1; sc++;} // NE
79     if(x[i-1][j-1]==0) {x[i-1][j-1]=t+1; qi[sc]=i-1; qj[sc]=j-1; sc++;} // NV
80
81     if(x[i+1][j]==0) {x[i+1][j]=t+1;  qi[sc]=i+1; qj[sc]=j;  sc++;} // S
82     if(x[i+1][j+1]==0) {x[i+1][j+1]=t+1; qi[sc]=i+1; qj[sc]=j+1; sc++;} // SE
83     if(x[i+1][j-1]==0) {x[i+1][j-1]=t+1; qi[sc]=i+1; qj[sc]=j-1; sc++;} // SV
84
85     if(x[i][j+1]==0) {x[i][j+1]=t+1;  qi[sc]=i;  qj[sc]=j+1; sc++;} // E
86     if(x[i][j-1]==0) {x[i][j-1]=t+1;  qi[sc]=i;  qj[sc]=j-1; sc++;} // V
87 } // fil(...)
88 } // class

```

---

## Capitolul 20

# OJI 2003

### 20.1 Spirala

Se consideră un automat de criptare format dintr-un tablou cu  $n$  linii și  $n$  coloane, tablou ce conține toate numerele de la 1 la  $n^2$  așezate "șerpuit" pe linii, de la prima la ultima linie, pe liniile impare pornind de la stânga către dreapta, iar pe cele pare de la dreapta către stânga (ca în figura alăturată).

Numim "amestecare" operația de desfășurare în spirală a valorilor din tablou în ordinea indicată de săgeți și de reasezare a acestora în același tablou, "șerpuit" pe linii ca și în cazul precedent.

De exemplu, desfășurarea tabloului conduce la șirul: 1 2 3 4 5 12 13 14 15 16 9 8 7 6 11 10, iar reasezarea acestuia în tablou conduce la obținerea unui nou tablou reprezentat în cea de-a doua figură alăturată.

1	2	3	4
8	7	6	5
9	10	11	12
16	15	14	13

Figura 20.1: Spirala1

1	2	3	4
14	13	12	5
15	16	9	8
10	11	6	7

Figura 20.2: Spirala2

<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>
6	7	8	<b>5</b>
11	<b>10</b>	15	14
<b>16</b>	9	12	<b>13</b>

Figura 20.3: Spirala3

După orice operație de amestecare se poate relua procedeul, efectuând o nouă amestecare. S-a observat un fapt interesant: că după un număr de amestecări, unele valori ajung din nou în poziția inițială (pe care o aveau în tabloul de pornire). De exemplu, după două amestecări, tabloul de  $4 \times 4$  conține 9 dintre elementele sale în exact aceeași poziție în care se aflau inițial (vezi elemente marcate din figură).

### Cerință

Pentru  $n$  și  $k$  citite, scrieți un program care să determine numărul minim de amestecări ale unui tablou de  $n$  linii necesar pentru a ajunge la un tablou cu exact  $k$  elemente aflate din nou în poziția inițială.

### Datele de intrare

Fișierul de intrare **spirala.in** conține pe prima linie cele două numere  $n$  și  $k$  despărțite printr-un spațiu.

### Datele de ieșire

Fișierul de ieșire **spirala.out** conține o singură linie pe care se află numărul de amestecări determinat.

### Restricții și precizări

- $3 \leq N \leq 50$
- Datele de intrare sunt alese astfel încât numărul minim de amestecări necesare să nu depășească  $2 * 10^9$
- La încheierea programului nu se va solicita apăsarea unei taste

### Exemple

spirala.in	spirala.out	spirala.in	spirala.out
4 9	2	6 36	330

**Timp maxim de executare:** 1 secundă/test

## 20.1.1 Indicații de rezolvare

Se calculează pentru fiecare poziție numărul de amestecări după care se repetă poziția respectivă (perioada principală).

Se calculează pentru toți divizorii  $d$  ai celui mai mic multiplu comun al numerelor calculate (ținut ca factori primi și exponenții corespunzători) numărul de poziții care se repetă după  $d$  amestecări.

Sursa comisiei generează divizorii cu bkt pe exponenții descompunerii în factori primi.

## 20.1.2 Cod sursă

Listing 20.1.1: spirala2.pas

```

1  const max=50;
2      d:array[1..2,0..3]of shortint=((0,1,0,-1),(1,0,-1,0));
3
4  type divi=record care,cati:longint end;
5      fprim=record care,ex:word end;
6      mat=array[0..max+1,0..max+1]of word;
7      vect=array[1..max*max]of word;
8      vectm=array[1..200]of byte;
9
10 var x:vect;
11     dv:array[1..200]of divi;cd:byte;
12     fp:array[1..200]of fprim;cfp:byte;
13     sol:vectm;
14     n:byte;v,min,cati:longint;
15
16 procedure calc;
17 var a,b,c:mat;
18     v:vect;
```

```

19     i, j, k, imp, rez, sens:longint;
20 begin
21     for i:=0 to n+1 do begin a[i,0]:=0;a[i,n+1]:=0;a[0,i]:=0;a[n+1,i]:=0 end;
22     for i:=1 to n do
23         for j:=1 to n do c[i,j]:=0;
24     k:=1;
25     for i:=1 to n do
26         if i mod 2=1 then
27             for j:=1 to n do begin a[i,j]:=k; inc(k) end
28         else
29             for j:=n downto 1 do begin a[i,j]:=k; inc(k) end;
30     b:=a; imp:=1; rez:=0; sens:=0;
31     repeat
32         i:=1; j:=1; sens:=0; k:=1;
33         repeat
34             v[k]:=a[i,j]; a[i,j]:=0;
35             if a[i+d[1,sens],j+d[2,sens]]=0 then sens:=(sens+1) mod 4;
36             i:=i+d[1,sens];
37             j:=j+d[2,sens];
38             inc(k)
39         until k>n*n;
40         k:=1;
41         for i:=1 to n do
42             if i mod 2=1 then
43                 for j:=1 to n do begin
44                     a[i,j]:=v[k];
45                     if (v[k]=b[i,j]) and (c[i,j]=0) then begin
46                         c[i,j]:=imp; inc(rez); inc(x[imp])
47                     end;
48                     inc(k)
49                 end
50             else
51                 for j:=n downto 1 do begin
52                     a[i,j]:=v[k];
53                     if (v[k]=b[i,j]) and (c[i,j]=0) then begin
54                         c[i,j]:=imp; inc(rez); inc(x[imp])
55                     end;
56                     inc(k)
57                 end;
58             inc(imp)
59         until rez=n*n
60     end;
61
62 procedure calc2;
63 var e:boolean;
64     i, j, k, n, ex:longint;
65 begin
66     cd:=0;
67     for i:=1 to max*max do
68         if x[i]>0 then
69             begin
70                 inc(cd);
71                 dv[cd].care:=i;
72                 dv[cd].cati:=x[i]
73             end;
74     cfp:=0;
75     for i:=2 to cd do begin
76         k:=dv[i].care;
77         for j:=1 to cfp do begin
78             n:=fp[j].care;
79             ex:=0;
80             while k mod n=0 do begin
81                 k:=k div n; inc(ex)
82             end;
83             if ex>fp[j].ex then fp[j].ex:=ex
84         end;
85         j:=2;
86         while k>1 do begin
87             if k mod j=0 then begin
88                 inc(cfp); fp[cfp].care:=j;
89                 fp[cfp].ex:=0;
90                 while k mod j=0 do begin
91                     inc(fp[cfp].ex);
92                     k:=k div j
93                 end
94             end;

```

```

95         inc(j)
96     end
97 end
98 end;
99
100 procedure calc3;
101 var i,j:byte;tot,k:longint;
102     soll:vectm;
103 begin
104     tot:=0;
105     for i:=1 to cd do begin
106         k:=dv[i].care;soll:=sol;
107         for j:=1 to cfp do
108             while (soll[j]>0) and (k mod fp[j].care=0) do
109                 begin
110                     k:=k div fp[j].care;
111                     dec(soll[j])
112                 end;
113             if k=1 then tot:=tot+dv[i].cati
114             end;
115             write(tot,'-',v,' ');
116             if tot=cati then
117                 if v<min then min:=v
118             end;
119
120 procedure back(i:byte);
121 var j:byte;p:longint;
122 begin
123     if i>cfp then calc3
124     else begin
125         p:=1;
126         for j:=0 to fp[i].ex do
127             if v<maxlongint/p then begin
128                 sol[i]:=j;
129                 v:=v*p;
130                 back(i+1);
131                 v:=v div p;
132                 p:=p*fp[i].care;
133             end;
134             sol[i]:=0;
135         end
136     end;
137
138 begin
139     readln(n,cati);
140     calc; calc2;
141     min:=maxlongint;
142     v:=1;
143
144     back(1);
145
146     writeln;
147     if min=maxlongint then writeln(0)
148     else writeln(min);
149     readln
150 end.

```

### 20.1.3 Rezolvare detaliată

Prima variantă:

Listing 20.1.2: spirala1.java

```

1 import java.io.*;           // testele 5,6,7,8,9,10 ==> timp mare!
2 class Spirala1
3 {
4     static int n,k;
5     static int[] pr;
6     static int[] p;
7
8     public static void main(String[] args) throws IOException
9     {
10         long t1,t2;

```

```

11     t1=System.currentTimeMillis();
12
13     int r=0,i,npf;
14     StreamTokenizer st=new StreamTokenizer(new BufferedReader(
15         new FileReader("spiral.in")));
16     PrintWriter out=new PrintWriter(new BufferedWriter(
17         new FileWriter("spiral.out")));
18     st.nextToken();n=(int)st.nval;
19     st.nextToken();k=(int)st.nval;
20
21     p=new int[n*n+1];
22     pr=new int[n*n+1];
23
24     constrp();
25     r=0;
26     for(i=1;i<=n*n;i++) pr[i]=i;
27
28     npf=nrPuncteFixe(pr);
29     while(npf!=k)
30     while((npf!=k)&&(r<4200))    // test 4 4200 ? 14280
31     {
32         r++;
33         pr=inmp(pr,p);
34         npf=nrPuncteFixe(pr);
35     }
36     System.out.print(r+":\t"); afisv(pr); System.out.println(" ==> "+npf);
37
38     out.println(r);
39     out.close();
40     t2=System.currentTimeMillis();
41     System.out.println(t2-t1);
42 }
43
44 static void constrp()
45 {
46     int i,j,k,v,kp=0;
47     int[][] a=new int[n+1][n+1];
48     i=1; v=0;
49     for(k=1;k<=n/2;k++)
50     {
51         for(j=1;j<=n;j++) a[i][j]=++v;
52         for(j=n;j>=1;j--) a[i+1][j]=++v;
53         i=i+2;
54     }
55     if(n%2==1) for(j=1;j<=n;j++) a[n][j]=++v;
56
57     // afism(a);
58     for(k=1;k<=n/2;k++) // contur dreptunghi k
59     {
60         i=k;
61         for(j=k;j<=n+1-k;j++) p[++kp]=a[i][j];
62         j=n+1-k;
63         for(i=k+1;i<=n-k;i++) p[++kp]=a[i][j];
64         i=n+1-k;
65         for(j=n+1-k;j>=k;j--) p[++kp]=a[i][j];
66         j=k;
67         for(i=n-k;i>=k+1;i--) p[++kp]=a[i][j];
68     }
69     if(n%2==1) p[n*n]=a[n/2+1][n/2+1]; // corectie ptr n=impar !!!
70     // afisv(p);
71 } // constrp()
72
73 static int[] inmp(int[] a,int[] b)
74 {
75     int i;
76     int[] c=new int[n*n+1];
77     for(i=1;i<=n*n;i++) c[i]=a[b[i]];
78     return c;
79 }
80
81 static int nrPuncteFixe(int[] pr)
82 {
83     int i, s=0;
84     for(i=1;i<=n*n;i++) if(pr[i]==i) s++;
85     return s;
86 }

```



```

87
88 static void afism(int[][] x)
89 {
90     int i, j;
91     for(i=1; i<=n; i++)
92     {
93         System.out.println();
94         for(j=1; j<=n; j++) System.out.print(x[i][j]+"\t");
95     }
96     System.out.println();
97 }
98
99 static void afisv(int[] x)
100 {
101     int i;
102     for(i=1; i<=n*n; i++) System.out.print(x[i]+" ");
103 }
104 } // class

```

A doua variantă:

Listing 20.1.3: spirala2.java

```

1 import java.io.*;          // testele 5, 6, ... ==> timp mare!
2 class Spirala2             // v1      33s   17s
3 {                          // v2      28s   14s ==> nu este un castig prea mare !
4     static int n,k,nn;
5     static int[] pr;
6     static int[] p;
7     static int[] pp;
8
9     public static void main(String[] args) throws IOException
10    {
11        long t1,t2;
12        t1=System.currentTimeMillis();
13
14        int r=0,i,npf;
15        StreamTokenizer st=new StreamTokenizer(new BufferedReader(
16            new FileReader("spirala.in")));
17        PrintWriter out=new PrintWriter(new BufferedWriter(
18            new FileWriter("spirala.out")));
19        st.nextToken(); n=(int)st.nval;
20        st.nextToken(); k=(int)st.nval;
21        nn=n*n-n-1;
22        p=new int[n*n+1];
23        pr=new int[n*n+1];
24        pp=new int[nn+1];
25
26        constrp();
27        r=0;
28        for(i=1; i<=nn; i++) pr[i]=i;
29        for(i=n+2; i<=n*n; i++) pp[i-n-1]=p[i]-n-1;
30
31        npf=nrPuncteFixe(pr);
32        while((npf!=k-n-1)&&(r<81840))
33        {
34            r++;
35            pr=inmp(pr,pp);
36            npf=nrPuncteFixe(pr);
37
38            if(r%1000==0) System.out.println(r+" "+npf);
39        }
40        System.out.println(r+" "+npf);
41        out.println(r);
42        out.close();
43        t2=System.currentTimeMillis();
44        System.out.println(t2-t1);
45    }
46
47    static void constrp()
48    {
49        int i,j,k,v,kp=0;
50        int[][] a=new int[n+1][n+1];
51        i=1; v=0;
52        for(k=1; k<=n/2; k++)
53        {

```

```

54     for(j=1; j<=n; j++) a[i][j]=++v;
55     for(j=n; j>=1; j--) a[i+1][j]=++v;
56     i=i+2;
57 }
58 if(n%2==1) for(j=1; j<=n; j++) a[n][j]=++v;
59
60 for(k=1; k<=n/2; k++) // contur dreptunghi k
61 {
62     i=k;
63     for(j=k; j<=n+1-k; j++) p[++kp]=a[i][j];
64     j=n+1-k;
65     for(i=k+1; i<=n-k; i++) p[++kp]=a[i][j];
66     i=n+1-k;
67     for(j=n+1-k; j>=k; j--) p[++kp]=a[i][j];
68     j=k;
69     for(i=n-k; i>=k+1; i--) p[++kp]=a[i][j];
70 }
71 if(n%2==1) p[n*n]=a[n/2+1][n/2+1]; // corectie ptr n=impar !!!
72 } // constrp()
73
74 static int[] inmp(int[] a, int[] b)
75 {
76     int i;
77     int[] c=new int[n*n+1];
78     for(i=1; i<=nn; i++) c[i]=a[b[i]];
79     return c;
80 }
81
82 static int nrPuncteFixe(int[] pr)
83 {
84     int i, s=0;
85     for(i=1; i<=nn; i++) if(pr[i]==i) s++;
86     return s;
87 }
88 } // class

```

Listing 20.1.4: spirala3.java

```

1  import java.io.*;           // OK ! cu cmmmc al ciclurilor
2  class Spirala3
3  {
4      static int n,k,nc,ncsol, nd;
5      static int[] p;
6      static int[] c;
7      static boolean[] vizitat;
8      static long[] lgc;           // lg cicluri
9      static long[] lgcsol;        // lg cicluri din sol ...
10     static long[] dcmmmc=new long[15]; // divizorii lui cmmm
11
12     static int[] npf;           // nr puncte fixe
13     static long min=Long.MAX_VALUE;
14
15     public static void main(String[] args) throws IOException
16     {
17         long t1,t2;
18         t1=System.currentTimeMillis();
19
20         int i;
21
22         StreamTokenizer st=new StreamTokenizer(new BufferedReader(
23             new FileReader("spirala.in")));
24         PrintWriter out=new PrintWriter(new BufferedWriter(
25             new FileWriter("spirala.out")));
26
27         st.nextToken(); n=(int)st.nval;
28         st.nextToken(); k=(int)st.nval;
29
30         p=new int[n*n+1];
31         c=new int[n*n+1];
32         vizitat=new boolean[n*n+1];
33
34         constrp();
35         cicluri();
36
37         nc=0;

```

```

38     for (i=2; i<=n*n; i++) if (c[i]>0) nc++;
39
40     lgc=new long[nc];
41     npf=new int[nc];
42     lgcsol=new long[nc];
43
44     k=k-c[1];
45
46     nc=0;
47     for (i=2; i<=n*n; i++) if (c[i]>0) {lgc[nc]=i; npf[nc]=i*c[i]; nc++;}
48
49     for (i=0; i<(1<<nc); i++) if (suma(i)==k) verific(i);
50     out.println(min);
51     out.close();
52
53     t2=System.currentTimeMillis();
54     System.out.println("Timp="+ (t2-t1));
55 }
56
57 static boolean amGasit(int val, long[] x, int p, int u)    // x=vector crescator
58 {
59     int m;
60     while (p<=u)
61     {
62         m=(p+u)/2;
63         if (x[m]==val) return true;
64         if (x[m]<val) p=m+1; else u=m-1;
65     }
66     return false;
67 }
68
69 static void verific(int i)
70 {
71     int d, j;
72     int ncsol=0;
73     long cmmm;
74     cmmm=1L;
75
76     j=0;
77     while (i!=0)
78     {
79         if (i%2!=0)                                // bit=1 pe pozitia j
80         {
81             cmmm=cmmmc(cmmm, lgc[j]);
82             if (cmmm>2000000000) return; // din enunt !!!
83             lgcsol[ncsol]=lgc[j];
84             ncsol++;                                // nr cicluri
85         }
86         j++;
87         i/=2;
88     }
89
90     // verific divizorii lui cmmm - pot aparea combinatii ...!!!
91     descFact(cmmm);
92     int ndl=1<<nd;
93     for (i=0; i<ndl; i++)
94     {
95         d=divizor(i);
96         if (!amGasit(d, lgcsol, 0, ncsol-1))
97             if (amGasit(d, lgc, 0, nc-1)) return;
98     }
99
100     if (cmmm<min) min=cmmm;
101 } // verific(...)
102
103 static int divizor(int i) // pentru generarea tuturor divizorilor
104 {
105     int p=1, j=0;
106     for (j=0; j<nd; j++) if ((i&(1<<j))!=0) p*=dcmmm[j];
107     return p;
108 }
109
110 static int suma(int i)
111 {
112     int s=0, j=0;
113     for (j=0; j<nc; j++) if ((i&(1<<j))!=0) s+=npf[j];

```

```

114     return s;
115 }
116
117 static void constrp()
118 {
119     int i, j, k, v, kp=0;
120     int[][] a=new int[n+1][n+1];
121     i=1; v=0;
122     for(k=1; k<=n/2; k++)
123     {
124         for(j=1; j<=n; j++) a[i][j]=++v;
125         for(j=n; j>=1; j--) a[i+1][j]=++v;
126         i=i+2;
127     }
128     if(n%2==1) for(j=1; j<=n; j++) a[n][j]=++v;
129
130     for(k=1; k<=n/2; k++) // contur dreptunghi k
131     {
132         i=k;
133         for(j=k; j<=n+1-k; j++) p[++kp]=a[i][j];
134         j=n+1-k;
135         for(i=k+1; i<=n-k; i++) p[++kp]=a[i][j];
136         i=n+1-k;
137         for(j=n+1-k; j>=k; j--) p[++kp]=a[i][j];
138         j=k;
139         for(i=n-k; i>=k+1; i--) p[++kp]=a[i][j];
140     }
141     if(n%2==1) p[n*n]=a[n/2+1][n/2+1]; // corectie ptr n=impar !!!
142 } // constrp()
143
144 static void cicluri()
145 {
146     int i, j, np;
147     for(i=1; i<=n*n; i++)
148     {
149         if(vizitat[i]) continue;
150         vizitat[i]=true;
151         np=1;
152         j=p[i]; vizitat[j]=true;
153         while(j!=i) {np++; j=p[j]; vizitat[j]=true;}
154         c[np]++;
155     }
156 } // cicluri()
157
158 static long cmmmc(long a, long b) // cmmmc(a,b)=(a*b)/cmmdc(a,b)=...
159 {
160     long dab, da;
161     long rez;
162     dab=cmmdc(a,b);
163     da=cmmdc(dab,a);
164     a/=da;
165     dab/=da;
166     b/=dab;
167     rez=a*b;
168     return rez;
169 } // cmmmc(...)
170
171 static long cmmdc(long a, long b)
172 {
173     long d,i,r;
174     if(a>b) {d=a; i=b;} else {d=b; i=a;}
175     while(i!=0) {r=d%i; d=i; i=r;}
176     return d;
177 } // cmmdc(...)
178
179 static void descFact(long nr)
180 {
181     nd=0; // nr divizori
182     long d=2;
183     if((nr==0) || (nr==1)) return;
184     while(nr%d==0) {nr=nr/d; dcmm[nr]=d; nd++;}
185     d=3;
186     while((d*d<=nr) && (nr!=1)) // poate sa ramana un numar prim!
187     {
188         while(nr%d==0) {nr=nr/d; dcmm[nr]=d; nd++;}
189         d=d+2;

```

```

190     }
191     if(nr!=1) {dcmmm[nd]=d; nd++;};
192 }// descFact(...)
193 }// class

```

## 20.2 Taxe

Într-o țară în care corupția este în floare și economia la pământ, pentru a obține toate aprobările necesare în scopul demarării unei afaceri, investitorul trebuie să treacă prin mai multe camere ale unei clădiri în care se află birouri.

Clădirea are un singur nivel în care birourile sunt lipite unele de altele formând un caroi aj pătrat de dimensiune  $n \times n$ . Pentru a facilita accesul în birouri, toate camerele vecine au uși între ele. În fiecare birou se află un funcționar care pretinde o taxă de trecere prin cameră (taxă ce poate fi, pentru unele camere, egală cu 0). Investitorul intră încrezător prin colțul din stânga-sus al clădirii (cum se vede de sus planul clădirii) și dorește să ajungă în colțul opus al clădirii, unde este ieșirea, plătind o taxă totală cât mai mică.

### Cerință

Știind că el are în buzunar  $S$  euro și că fiecare funcționar îi ia taxa de cum intră în birou, se cere să se determine dacă el poate primi aprobările necesare și, în caz afirmativ, care este suma maximă de bani care îi rămâne în buzunar la ieșirea din clădire.

### Datele de intrare

Fișierul de intrare **taxe.in** conține pe prima linie cele două numere  $S$  și  $n$  despărțite printr-un spațiu, iar pe următoarele  $n$  linii câte  $n$  numere separate prin spații ce reprezintă taxele cerute de funcționarii din fiecare birou.

### Datele de ieșire

Fișierul de ieșire **taxe.out** conține o singură linie pe care se află numărul maxim de euro care îi rămân în buzunar sau valoarea  $-1$  dacă investitorului nu-i ajung banii pentru a obține aprobarea.

### Restricții și precizări

- $3 \leq N \leq 100$
- $1 \leq S \leq 10000$
- Valorile reprezentând taxele cerute de funcționarii din birouri sunt numere naturale, o taxă nedepășind valoarea de 200 de euro.
- La încheierea programului nu se va solicita apăsarea unei taste

### Exemplu

taxe.in	taxe.out
10 3	3
1 2 5	
1 3 1	
0 8 1	

**Timp maxim de executare:** 1 secundă/test

### 20.2.1 Indicații de rezolvare

Se aplică un algoritm de tip Lee care expandează o coadă ce conține inițial doar starea  $(1, 1, S)$  cu toate stările în care se poate ajunge dintr-o poziție dată.

Se adaugă stările noi sau se actualizează stările în care se poate ajunge cu mai mulți bani în buzunar.

### 20.2.2 Cod sursă

Listing 20.2.1: taxe.pas

```

1 program taxe;
2 const nm=100;

```

```

3  type element=record
4      x,y:byte;
5      v:word;
6  end;
7
8  var co:array[1..2000] of element;
9      nco:word;
10     a:array[1..nm,1..nm] of byte;
11     sp:array[1..nm,1..nm] of word;
12     n,m:byte;
13     S:word;
14
15 procedure citire;
16 var i,j:byte;
17     f:text;
18 begin
19     assign(f,'pitule.in');reset(f);
20     readln(f,s,n,m);
21     for i:=1 to n do
22         for j:=1 to m do
23             read(f,a[i,j]);
24 close(f);
25 end;
26
27 procedure add(xi,yi:byte;vi:word);
28 var i:word;au:element;
29 begin
30     inc(nco);
31     sp[xi,yi]:=vi;
32     with co[nco] do
33     begin
34         x:=xi;y:=yi;v:=vi;
35     end;
36     i:=nco;
37     while (i>1)and(co[i-1].v<co[i].v)do
38     begin
39         au:=co[i-1];
40         co[i-1]:=co[i];
41         co[i]:=au;
42         dec(i);
43     end;
44 end;
45
46 procedure extr(var xi,yi:byte;var vi:word);
47 begin
48     with co[nco] do
49     begin
50         xi:=x;
51         yi:=y;
52         vi:=v;
53     end;
54     dec(nco);
55 end;
56
57 procedure rezolva;
58 var x,y:byte;v:word;f:boolean;
59 begin
60     fillchar(sp,sizeof(sp),255);
61     add(1,1,a[1,1]);
62
63     f:=true;
64     while (nco>0)and f do
65     begin
66         extr(x,y,v);
67         if (x=n)and(y=m) then f:=false
68         else
69         begin
70             if x<n then if sp[x+1,y]>sp[x,y]+a[x+1,y] then
71                 add(x+1,y,sp[x,y]+a[x+1,y]);
72             if y<m then if sp[x,y+1]>sp[x,y]+a[x,y+1] then
73                 add(x,y+1,sp[x,y]+a[x,y+1]);
74             if x>1 then if sp[x-1,y]>sp[x,y]+a[x-1,y] then
75                 add(x-1,y,sp[x,y]+a[x-1,y]);
76             if y>1 then if sp[x,y-1]>sp[x,y]+a[x,y-1] then
77                 add(x,y-1,sp[x,y]+a[x,y-1]);
78         end;

```

```

79     end;
80 end;
81
82 procedure scrie;
83 var f:text;
84 begin
85   assign(f,'spaga.out');rewrite(f);
86   if sp[n,m]>S then writeln(f,-1)
87   else writeln(f,S-sp[n,m]);
88   close(f);
89 end;
90
91 Begin
92 citire;
93 rezolva;
94 scrie;
95 End.

```

### 20.2.3 Rezolvare detaliată

Listing 20.2.2: taxe.java

```

1  import java.io.*;
2  class Taxe
3  {
4      static int n,s;
5      static int[][] taxa;
6      static int[][] taxaMin;
7      static final int infinit=200*200;
8
9      public static void main(String[] args) throws IOException
10     {
11         int i,j,min;
12         boolean amOptimizare;
13         long t1,t2;
14         t1=System.currentTimeMillis();
15
16         StreamTokenizer st=new StreamTokenizer(
17             new BufferedReader(new FileReader("taxe.in")));
18         PrintWriter out=new PrintWriter(
19             new BufferedWriter(new FileWriter("taxe.out")));
20
21         st.nextToken(); s=(int)st.nval;
22         st.nextToken(); n=(int)st.nval;
23
24         taxa=new int[n+2][n+2];
25         taxaMin=new int[n+2][n+2];
26
27         for(i=1;i<=n;i++)
28             for(j=1;j<=n;j++) { st.nextToken(); taxa[i][j]=(int)st.nval; }
29         for(i=0;i<=n+1;i++)
30             for(j=0;j<=n+1;j++) taxaMin[i][j]=infinit;
31
32         taxaMin[1][1]=taxa[1][1];
33         amOptimizare=true;
34         while(amOptimizare)
35         {
36             amOptimizare=false;
37             for(i=1;i<=n;i++)
38                 for(j=1;j<=n;j++)
39                 {
40                     min=minTaxeVecini(i,j);
41                     if(min+taxa[i][j]<taxaMin[i][j])
42                     {
43                         taxaMin[i][j]=min+taxa[i][j];
44                         amOptimizare=true;
45                     }
46                 }
47             }
48         if(taxaMin[n][n]<=s) out.println(s-taxaMin[n][n]); else out.println(-1);
49         out.close();
50         t2=System.currentTimeMillis();

```

```
51     System.out.println("Timp="+ (t2-t1));
52 } // main()
53
54 static int minTaxeVecini(int i, int j)
55 {
56     int min1,min2;
57     min1=minim(taxaMin[i-1][j],taxaMin[i+1][j]);
58     min2=minim(taxaMin[i][j-1],taxaMin[i][j+1]);
59     return minim(min1,min2);
60 }
61
62 static int minim(int a, int b)
63 {
64     if(a<b) return a; else return b;
65 }
66 }
```

---



# Capitolul 21

## OJI 2002

### 21.1 Cod strămoș

Principală misiune a unei expediții științifice este de a studia evoluția vieții pe o planetă nou descoperită. În urma studiilor efectuate, cercetătorii au asociat fiecărui organism viu descoperit pe acea planetă un cod caracteristic.

Codul caracteristic este un număr natural de maximum 200 de cifre zecimale nenule.

De asemenea, cercetătorii au observat că pentru orice organism viu de pe planetă, codurile caracteristice ale strămoșilor săi pe scara evoluției se pot obține prin ștergerea unor cifre din codul caracteristic al organismului respectiv, iar un organism este cu atât mai evoluat cu cât codul său caracteristic are o valoare mai mare.

#### Cerință

Date fiind codurile caracteristice ale două organisme vii diferite, scrieți un program care să determine codul caracteristic al celui mai evoluat strămoș comun al lor.

#### Datele de intrare

Fișierul de intrare **cod.in** conține

$n$  - codul caracteristic al primului organism

$m$  - codul caracteristic al celui de-al doilea organism

#### Datele de ieșire

Fișierul de ieșire **cod.out** conține pe prima linie:

$p$  - codul celui mai evoluat strămoș comun al lui  $n$  și  $m$

#### Exemplu

cod.in	cod.out
7145	75
847835	

**Timp maxim de executare:** 1 secundă/test

#### 21.1.1 \*Indicații de rezolvare

#### 21.1.2 \*Cod sursă

#### 21.1.3 Rezolvare detaliată

Listing 21.1.1: codstramos1.java

```
1 import java.io.*; // cate modalitati de stergere exista pentru x si pentru y
2 class CodStramos1 // astfel incat sa ramana coduri egale de lg maxima in x si y
3 { // si afisare matrice;
4     static int [][] a; // sunt afisate codurile ramase nu cele sterse
```

```

5     static String xx,yy; // asa ca apar dubluri de coduri !!!
6     static char[] x,y,z;
7     static int n,m;
8     static int nsol=0;
9     static PrintWriter out;
10
11    public static void main(String[] args) throws IOException
12    {
13        int i;
14        BufferedReader br=new BufferedReader(new FileReader("cod.in"));
15        out=new PrintWriter(new BufferedWriter(new FileWriter("cod.out")));
16        xx=br.readLine();
17        yy=br.readLine();
18        n=xx.length(); // coloane
19        m=yy.length(); // linii
20        x=new char[n+1];
21        y=new char[m+1];
22        for(i=0;i<n;i++) x[i+1]=xx.charAt(i);
23        for(i=0;i<m;i++) y[i+1]=yy.charAt(i);
24
25        matrad();
26        afism(a);
27
28        z=new char[a[m][n]+1];
29        sol(m,n,a[m][n]);
30        out.close();
31    }
32
33    static void matrad()
34    {
35        int i,j;
36        a=new int[m+1][n+1];
37
38        for(i=1;i<=m;i++)
39        for(j=1;j<=n;j++)
40        if(x[j]==y[i]) a[i][j]=1+a[i-1][j-1];
41        else a[i][j]=max(a[i-1][j],a[i][j-1]);
42    }
43
44    static void sol(int lin, int col,int k) throws IOException
45    {
46        int i,j,kk;
47        if(k==0)
48        {
49            ++nsol;
50            System.out.print(nsol+" : \t");
51            afisv(z);
52            return;
53        }
54        i=lin;
55        while((i>0)&&(a[i][col]==k))
56        {
57            j=col;
58            while((j>0)&&(a[i][j]==k))
59            {
60                while((j>0)&&
61                    (a[i][j]==k)&&
62                    (x[j]!=y[i]||(a[i-1][j-1]!=(k-1))))
63                {
64                    j--;
65                }
66
67                if((j>0)&&(a[i][j]==k)&&(a[i-1][j-1]==(k-1))&&(x[j]==y[i]))
68                {
69                    z[k]=y[i]; // sau x[j];
70                    sol(i-1,j-1,k-1);
71                }
72                j--;
73            }
74            i--;
75        } //while
76    }
77
78    static int max(int a, int b)
79    {
80        if(a>b) return a; else return b;

```

```

81     }
82
83     static void afisv(char[] v)
84     {
85         int i;
86         for (i=1; i<=v.length-1; i++)
87         {
88             System.out.print(v[i]);
89             out.print(v[i]);
90         }
91         System.out.println();
92         out.println();
93     }
94
95     static void afism(int[][] a)
96     {
97         int i, j;
98
99         System.out.print("    ");
100        for (j=0; j<n; j++) System.out.print(xx.charAt(j)+" ");
101        System.out.println("x");
102
103        System.out.print("    ");
104        for (j=0; j<=n; j++) System.out.print(a[0][j]+" ");
105        System.out.println();
106
107        for (i=1; i<=m; i++)
108        {
109            System.out.print(yy.charAt(i-1)+" ");
110
111            for (j=0; j<=n; j++) System.out.print(a[i][j]+" ");
112            System.out.println();
113        }
114        System.out.println("y\n");
115    }
116 }

```

Listing 21.1.2: codstramos2.java

```

1  import java.io.*;           // Cod maxim lexicografic
2  class CodStramos2          // Cod minim lexicografic ==> modificare simpla!
3  {
4      static int [][] a;
5      static String xx, yy;
6      static char[] x, y, z;
7      static int n, m;
8
9      public static void main(String[] args) throws IOException
10     {
11         int i;
12         BufferedReader br=new BufferedReader(new FileReader("cod.in"));
13         xx=br.readLine();
14         yy=br.readLine();
15         n=xx.length(); // coloane
16         m=yy.length(); // linii
17         x=new char[n+1];
18         y=new char[m+1];
19         for (i=0; i<n; i++) x[i+1]=xx.charAt(n-1-i);
20         for (i=0; i<m; i++) y[i+1]=yy.charAt(m-1-i);
21         matrad();
22         z=new char[a[m][n]+1];
23         sol(m, n, a[m][n]);
24         PrintWriter out=new PrintWriter(
25             new BufferedWriter( new FileWriter("cod.out")));
26         for (i=z.length-1; i>=1; i--) out.print(z[i]);
27         out.close();
28     }
29
30     static void matrad()
31     {
32         int i, j;
33         a=new int[m+1][n+1];
34
35         for (i=1; i<=m; i++)
36             for (j=1; j<=n; j++)

```

```

37         if(x[j]==y[i]) a[i][j]=1+a[i-1][j-1];
38         else          a[i][j]=max(a[i-1][j],a[i][j-1]);
39     }
40
41     static void sol(int lin, int col,int k) throws IOException
42     {
43         int i,j,kk;
44         //if(k==0) return;
45         i=lin;
46         while((i>0) && (a[i][col]==k))
47         {
48             j=col;
49             while((j>0) && (a[i][j]==k))
50             {
51                 while((j>0) &&
52                     (a[i][j]==k) &&
53                     (x[j]!=y[i] || (a[i-1][j-1]!=(k-1))))
54                 {
55                     j--;
56                 }
57
58                 if((j>0) && (a[i][j]==k) && (a[i-1][j-1]==(k-1)) && (x[j]==y[i]))
59                     if(y[i]>z[k])
60                     {
61                         z[k]=y[i]; // sau x[j];
62                         for(kk=1;kk<k;kk++) z[kk]='0'-1; // curat z (cod<'0') in fata lui k
63                         if(k>1) sol(i-1,j-1,k-1);
64                     }
65                 j--;
66             }
67             i--;
68         } //while
69     }
70
71     static int max(int a, int b)
72     {
73         if(a>b) return a; else return b;
74     }
75 }

```

## 21.2 Triangulații

O triangulație a unui poligon convex este o mulțime formată din diagonale ale poligonului care nu se intersectează în interiorul poligonului ci numai în vârfuri și care împart toată suprafața poligonului în triunghiuri.

Fiind dat un poligon cu  $n$  vârfuri notate  $1, 2, \dots, n$  să se genereze toate triangulațiile distincte ale poligonului. Două triangulații sunt distincte dacă diferă prin cel puțin o diagonală.

**Datele de intrare:** în fișierul text **triang.in** se află pe prima linie un singur număr natural reprezentând valoarea lui  $n$  ( $n \leq 11$ ).

**Datele de ieșire:** în fișierul text **triang.out** se vor scrie:

- pe prima linie, numărul de triangulații distincte;
- pe fiecare din următoarele linii câte o triangulație descrisă prin diagonalele ce o compun. O diagonală va fi precizată prin două numere reprezentând cele două vârfuri care o definesc; cele două numere ce definesc o diagonală se despart prin cel puțin un spațiu, iar între perechile de numere ce reprezintă diagonalele dintr-o triangulație se va lăsa de asemenea minimum un spațiu.

### Exemplu

triang.in	triang.out
5	5 1 3 1 4 2 4 2 5 5 2 5 3 3 5 3 1 4 2 1 4

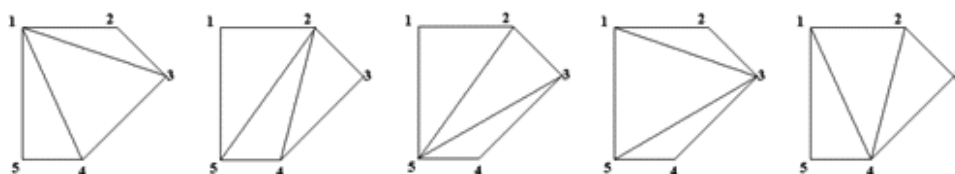


Figura 21.1: Triang

**Timp maxim de executare:**

7 secunde/test pe un calculator la 133 MHz.

3 secunde/test pe un calculator la peste 500 MHz.

**21.2.1 \*Indicații de rezolvare****21.2.2 \*Cod sursă****21.2.3 Rezolvare detaliată**

Listing 21.2.1: triangulatii.java

---

```

1  import java.io.*;                                // merge si n=12 in 3 sec
2  class Triangulatii
3  {
4      static int n;                                // numar varfuri poligon
5      static int ndt=n-3;                          // numar diagonale in triangulatie
6      static int nd=n*(n-3)/2; // numarul tuturor diagonalelor
7      static int[] x;
8      static int[] v1,v2;
9      static int nsol=0;
10     static PrintWriter out;
11
12     public static void main(String[] args) throws IOException
13     {
14         long t1,t2;
15         t1=System.currentTimeMillis();
16         StringTokenizer st=new StringTokenizer(
17             new BufferedReader(new FileReader("triang.in")));
18         out=new PrintWriter(new BufferedWriter(new FileWriter("triang.out")));
19         st.nextToken(); n=(int)st.nval;
20         ndt=n-3;
21         nd=n*(n-3)/2;
22         x=new int[ndt+1];
23         v1=new int[nd+1];
24         v2=new int[nd+1];
25
26         if(n==3) out.println(0);
27         else
28         {
29             out.println(catalan(n-2));
30             diagonale();
31             f(1);
32         }
33         out.close();
34         t2=System.currentTimeMillis();
35         System.out.println("nsol = "+nsol+" Timp = "+(t2-t1));
36     }
37
38     static void afisd() throws IOException
39     {
40         int i;
41         ++nsol;
42         for(i=1;i<=ndt;i++) out.print(v1[x[i]]+" "+v2[x[i]]+" ");
43         out.println();
44     }

```

```

45
46 static void diagonale()
47 {
48     int i, j, k=0;
49     i=1;
50     for (j=3; j<=n-1; j++) {v1[++k]=i; v2[k]=j;}
51
52     for (i=2; i<=n-2; i++)
53     for (j=i+2; j<=n; j++) {v1[++k]=i; v2[k]=j;}
54 }
55
56 static boolean seIntersecteaza(int k, int i)
57 {
58     int j; // i si x[j] sunt diagonalele !!!
59     for (j=1; j<=k-1; j++)
60     if ((v1[x[j]]<v1[i]) && (v1[i]<v2[x[j]]) && (v2[x[j]]<v2[i]) ) ||
61         (v1[i]<v1[x[j]]) && (v1[x[j]]<v2[i]) && (v2[i]<v2[x[j]]))
62         return true;
63     return false;
64 }
65
66 static void f(int k) throws IOException
67 {
68     int i;
69     for (i=x[k-1]+1; i<=nd-ndt+k; i++)
70     {
71         if (seIntersecteaza(k, i)) continue;
72         x[k]=i;
73         if (k<ndt) f(k+1); else afisd();
74     }
75 }
76
77 static int catalan(int n)
78 {
79     int rez;
80     int i, j;
81     int d;
82     int[] x=new int[n+1];
83     int[] y=new int[n+1];
84
85     for (i=2; i<=n; i++) x[i]=n+i;
86     for (j=2; j<=n; j++) y[j]=j;
87
88     for (j=2; j<=n; j++)
89     for (i=2; i<=n; i++)
90     {
91         d=cmmdc(y[j], x[i]);
92         y[j]=y[j]/d;
93         x[i]=x[i]/d;
94         if (y[j]==1) break;
95     }
96     rez=1;
97     for (i=2; i<=n; i++) rez*=x[i];
98     return rez;
99 }
100
101 static int cmmdc (int a, int b)
102 {
103     int d, i, c, r;
104     if (a>b) {d=a; i=b;} else {d=b; i=a;}
105     while (i!=0) {c=d/i; r=d%i; d=i; i=r;}
106     return d;
107 }
108 } // class

```

---



# Appendix A

## ”Instalare” C++

Ca să putem ”lucra” cu C++ avem nevoie de

- un compilator pentru C++, și
- un IDE (Integrated Development Enviroment) pentru C++.

### A.1 Kit\_OJI\_2017

Poate că cel mai ușor este să se descarce fișierul

[http://www.cnlr.ro/resurse/download/Kit\\_OJI\\_2017.rar](http://www.cnlr.ro/resurse/download/Kit_OJI_2017.rar)  
[https://cdn.kilonova.ro/p/WXbRLG/Kit\\_OJI\\_2017.rar](https://cdn.kilonova.ro/p/WXbRLG/Kit_OJI_2017.rar)  
[http://olimpiada.info/oji2018/Kit\\_OJI\\_2017.rar](http://olimpiada.info/oji2018/Kit_OJI_2017.rar)  
[https://www.liis.ro/Documents/download/Kit\\_OJI\\_2017.rar](https://www.liis.ro/Documents/download/Kit_OJI_2017.rar)

folosit de către elevi la școală și la olimpiade.

Fișierele din arhivă sunt:



Figura A.1: Fișierele din Kit\_OJI\_2017

Se lansează în execuție fișierul OJlkit\_2017.exe.

Instalarea este foarte ușoară (este de tipul ”next -> next -> ... -> next”) iar pe internet sunt multe site-uri de ajutor. De exemplu:

<https://www.pbinfo.ro/?pagina=intrebari-afisare&id=26>  
<https://www.youtube.com/watch?v=CLkWRvAwL08>  
<https://infoas.ro/lectie/112/tutorial-instalare-codeblocks-usor-introducere-in-in>  
[https://www.competentedigitale.ro/c/oji2019/Kit\\_OJI\\_2017.rar](https://www.competentedigitale.ro/c/oji2019/Kit_OJI_2017.rar)

Există numeroase alternative la CodeBlocks: Dev-C++, Microsoft Visual Studio, Eclipse, NetBeans, CodeLite, CLion, KDevelop, etc.



Kitul `Kit_OJI_2017` se instalează implicit pe `C:\OJI\`

- IDE-ul **Code::Blocks**<sup>27</sup>,
- compilatorul pentru C: `gcc.exe`,
- compilatorul pentru C++: `g++.exe`<sup>28</sup>
- `make.exe`
- `gdb.exe`
- și... altele! 😊

La sfârșitul instalării apar pe ecran două link-uri:

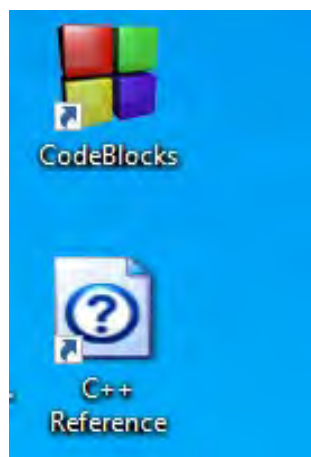


Figura A.2: CodeBlocks & C++ Reference

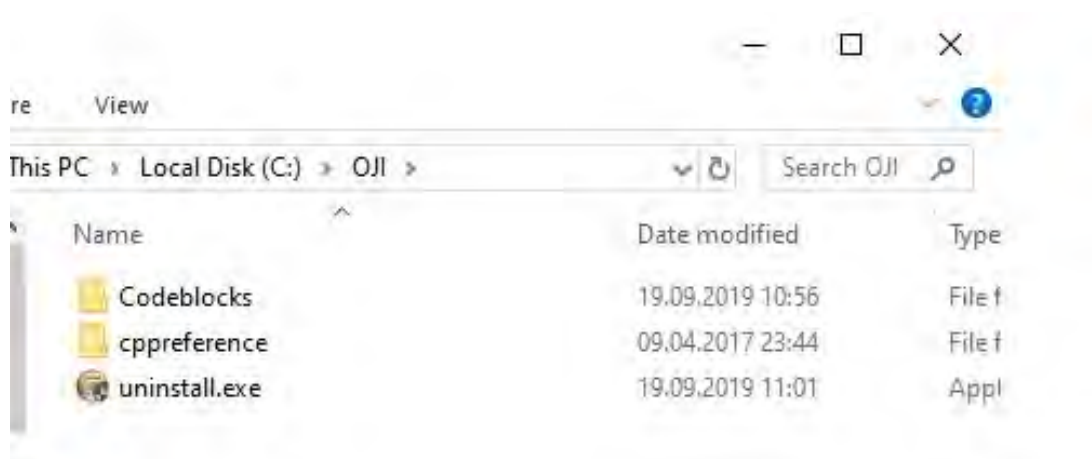


Figura A.3: Ce conține `C:\OJI\`

### A.1.1 Code::Blocks

Pentru versiuni mai noi, de Code::Blocks și compilatoare, se poate accesa site-ul

<http://www.codeblocks.org/downloads/binaries>

de unde se poate descărca, de exemplu, `codeblocks-20.03mingw-setup.exe`.

Versiuni mai vechi, dar foarte bune, se pot descărca de la adresa

<sup>27</sup> **Code::Blocks** este un *IDE* (integrated development environment) pentru C/C++, un fel de Notepad ... mai sofisticat, cu *multe butoane* care lansează în execuție diverse programe, de exemplu `g++.exe`

<sup>28</sup> `g++.exe` este compilatorul de C++, programul care va verifica dacă instrucțiunile noastre sunt ok sau nu ... și care ne va supăra mereu cu erorile pe care ni le arată ... 😊 ... Este "o mică artă" să înțelegem mesajele de eroare pe care le vedem pe ecran și să fim în stare să "depanăm" programele noastre!

<http://www.codeblocks.org/downloads/source/5>

Mai precis:

<https://sourceforge.net/projects/codeblocks/files/Binaries/20.03/>

<http://sourceforge.net/projects/codeblocks/files/Binaries/17.12>

<http://sourceforge.net/projects/codeblocks/files/Binaries/16.01>

### A.1.2 Folder de lucru

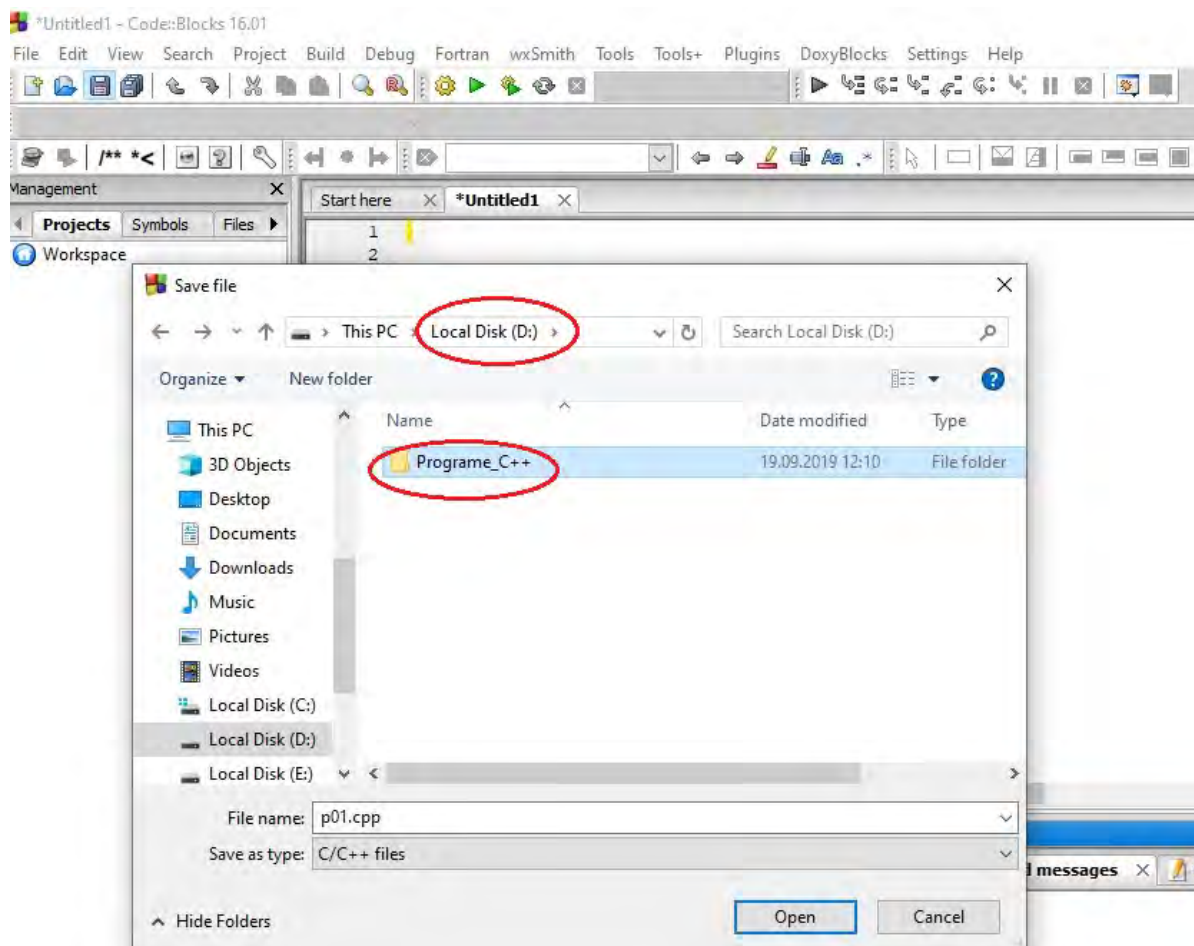


Figura A.4: Folder de lucru

De preferat este să avem cel puțin două partiții: C:\, D:\, ... și să "lăsăm în pace" partiția C:\ pentru sistemul de operare și programele instalate!

În figura de mai sus se poate observa că pe D:\ există un folder de lucru pentru programele în C++, folder care se numește **Programe\_C++**. Aici vom salva toate programele C++ pe care le vom scrie, eventual organizate pe mai multe subfoldere.

Acum, să intrăm în folderul **Programe\_C++**.

Click-dreapta cu mouse-ul și selectăm "New -> Text document" ca în figura următoare.

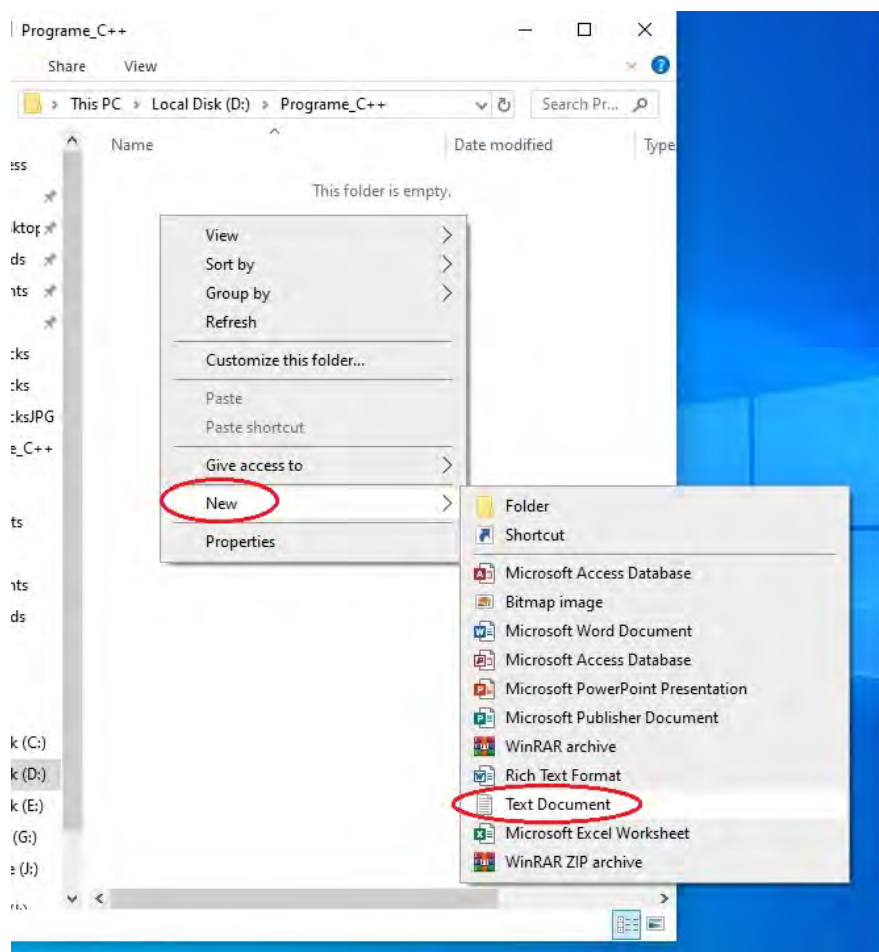


Figura A.5: New -&gt; Text document

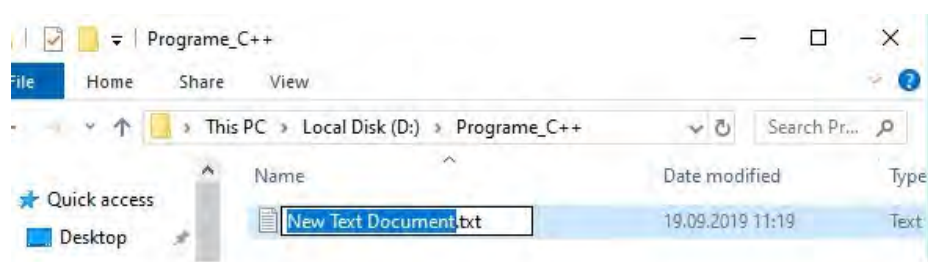


Figura A.6: Schimbare nume fișier și nume extensie fișier

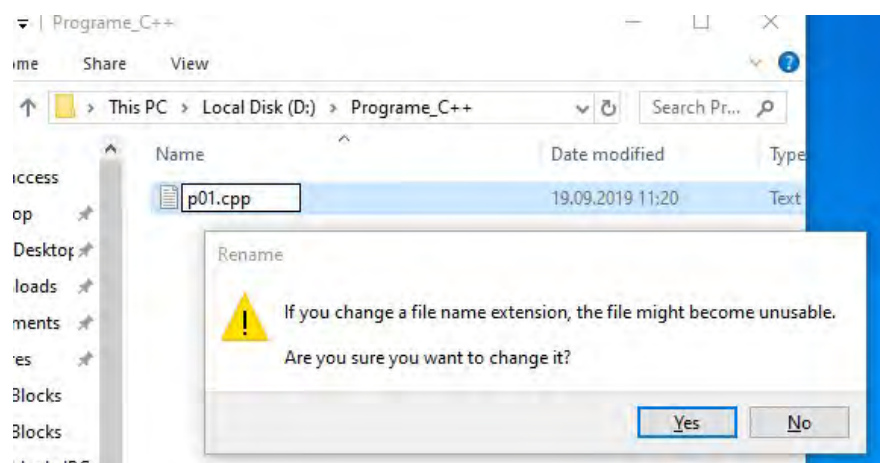


Figura A.7: Confirmare schimbare extensie în .cpp

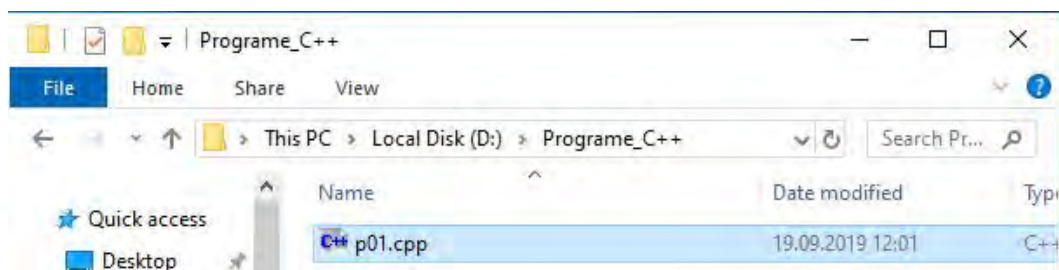


Figura A.8: Pregătit pentru Code::Blocks

Dacă vom executa două click-uri pe numele fișierului **p01.cpp** sau un click pentru a marca fișierul și apoi un **Enter**, se va declanșa **Code::Blocks** cu **p01.cpp** în fereastra de editare și, ce este și mai important, cu **Programme\_C++** ca **folder curent** de lucru pentru **Code::Blocks**. Adică, aici vor apărea toate fișiere generate de **Code::Blocks** pentru **p01.cpp**.

### A.1.3 Utilizare Code::Blocks

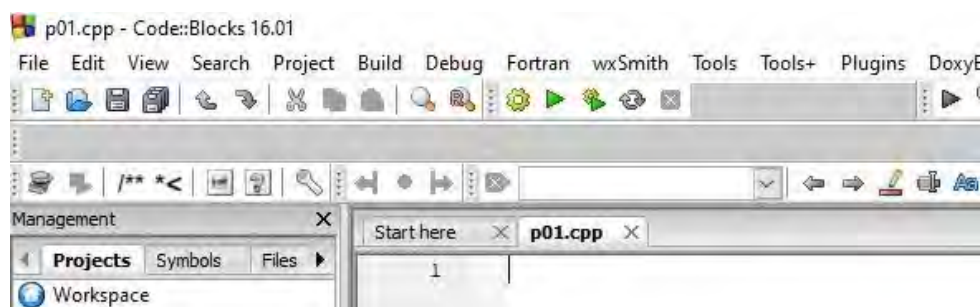


Figura A.9: Pregătit pentru a scrie cod de program C++ în Code::Blocks

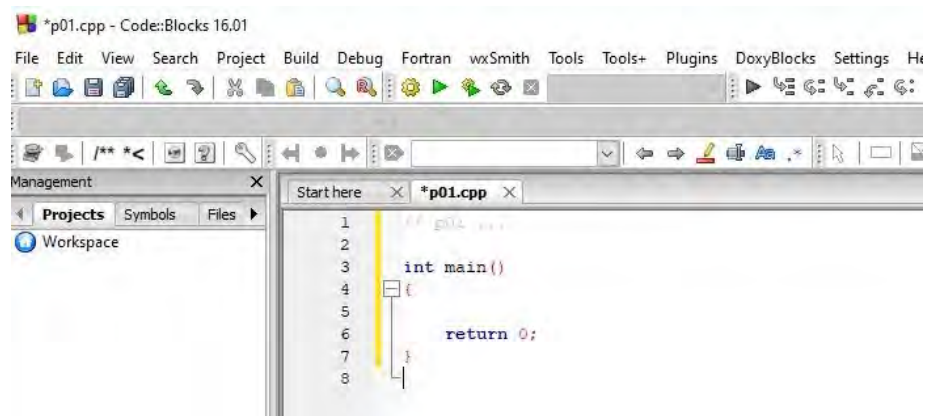


Figura A.10: Primul cod de program C++ în Code::Blocks

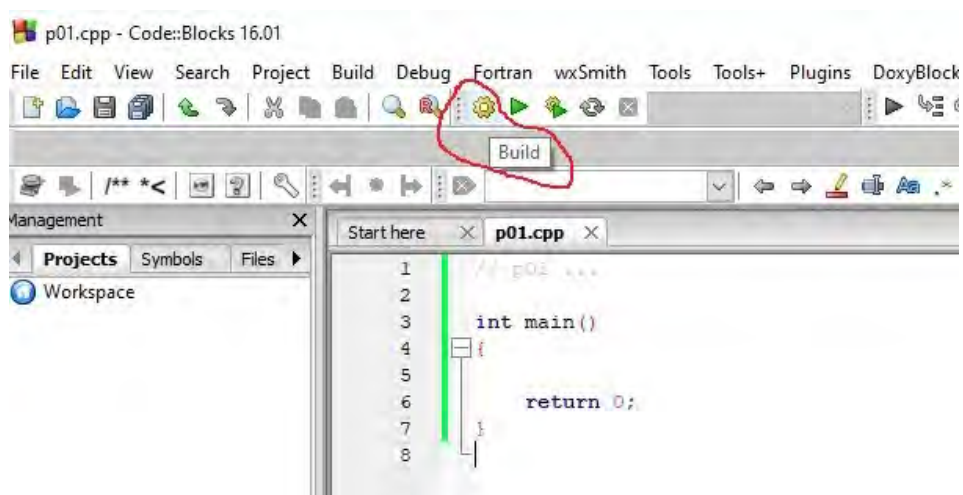


Figura A.11: Build - compilare



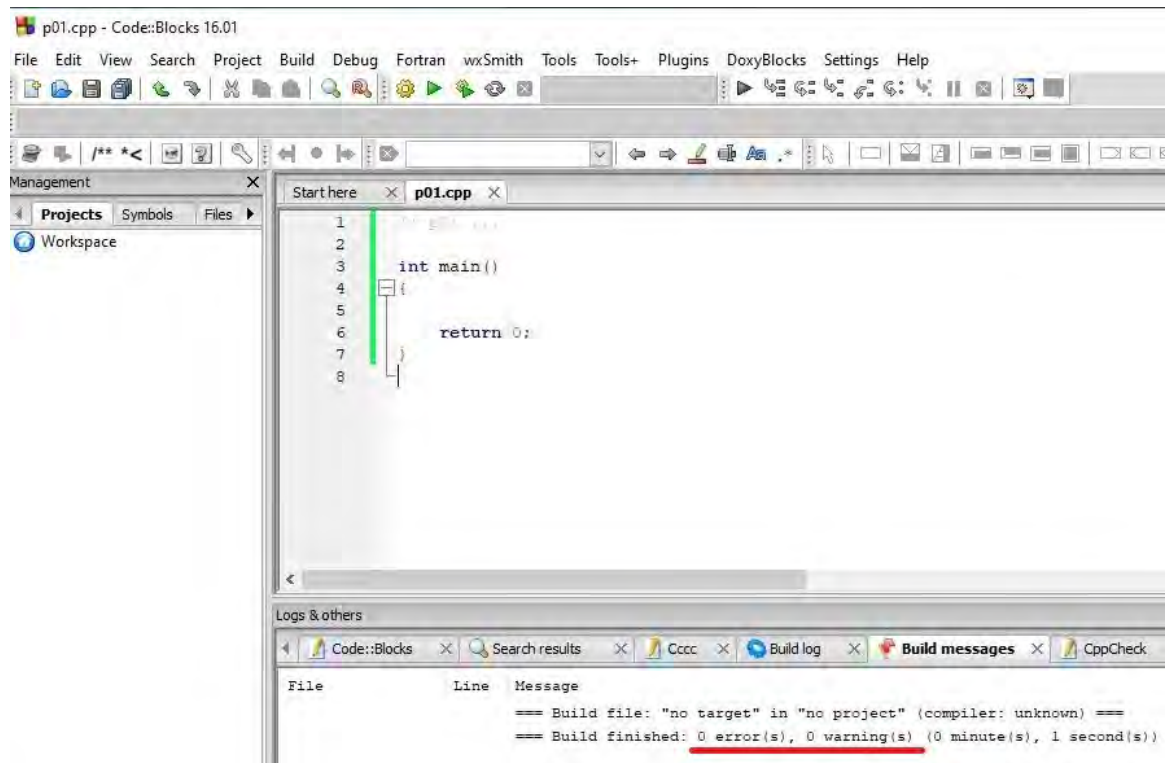


Figura A.12: 0 error(s), 0 warning(s)

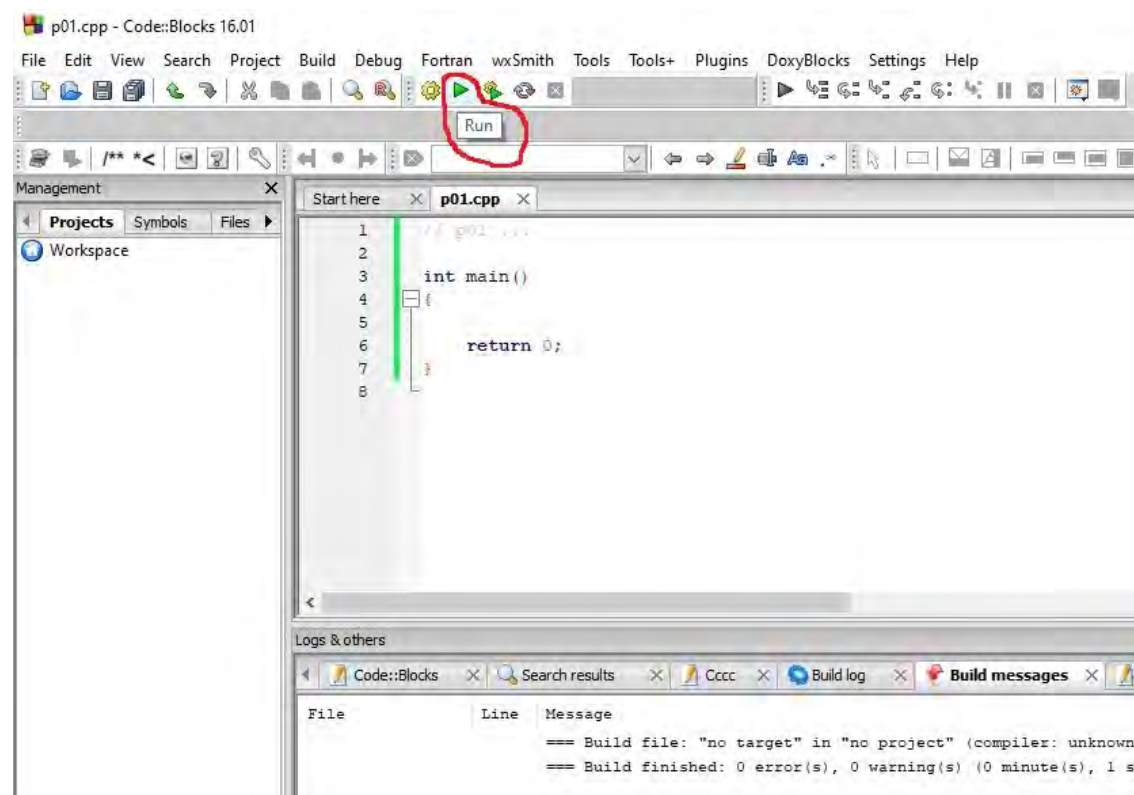


Figura A.13: Run - execuție

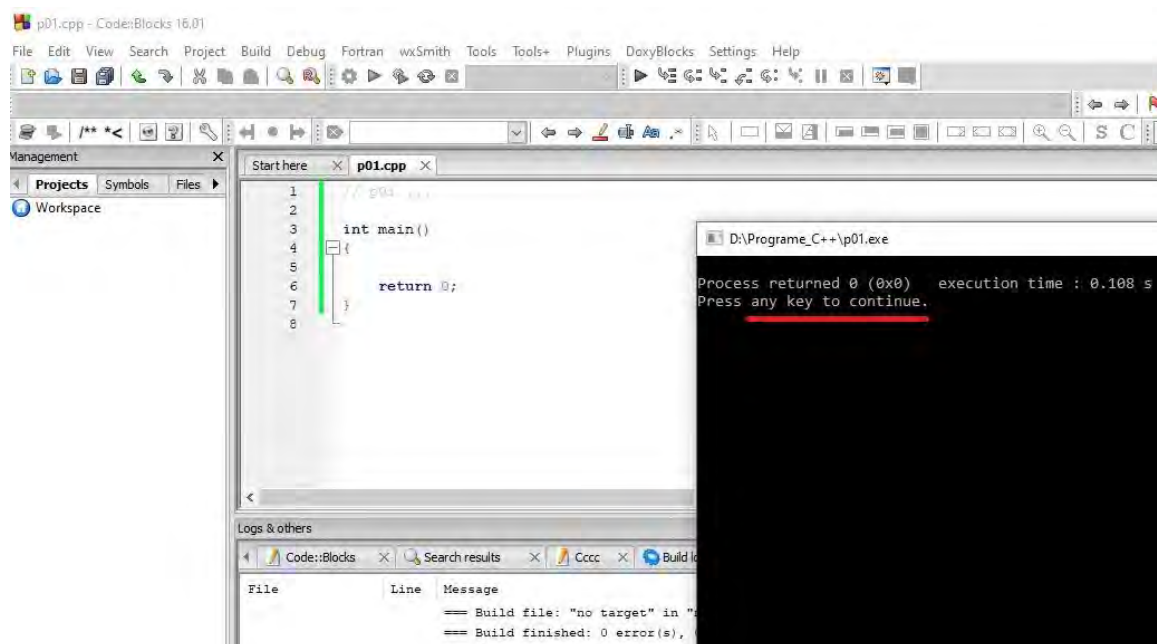


Figura A.14: Executat corect: a făcut "nimic"

#### A.1.4 Setări Code::Blocks

De preferat este să lăsăm setările implicite (sunt stabilite totuși de niște specialiști!) dar, dacă vrem, putem să umblăm și noi prin setări!

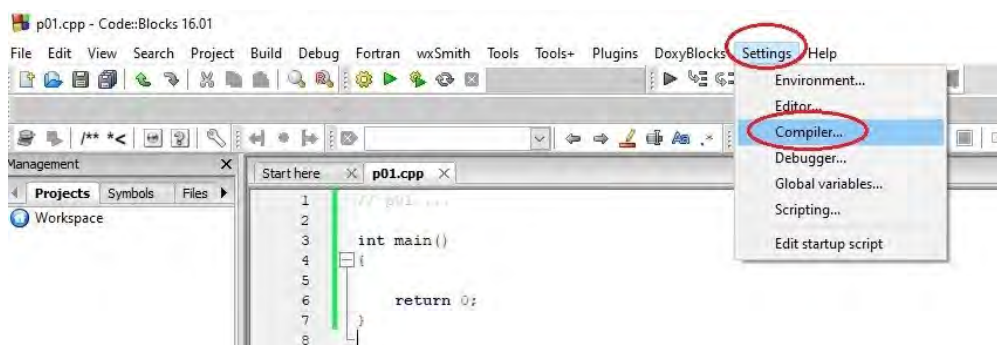


Figura A.15: Settings --&gt; Compiler

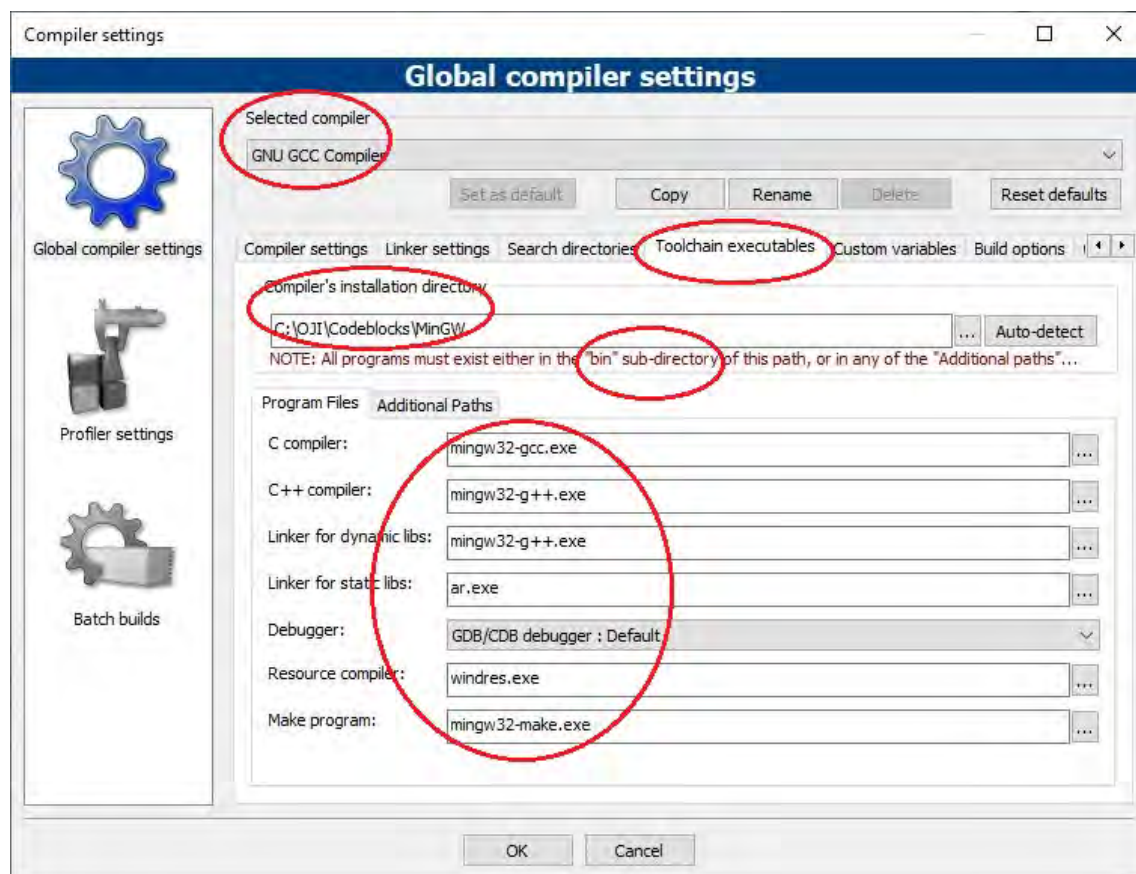


Figura A.16: Toolchain executables

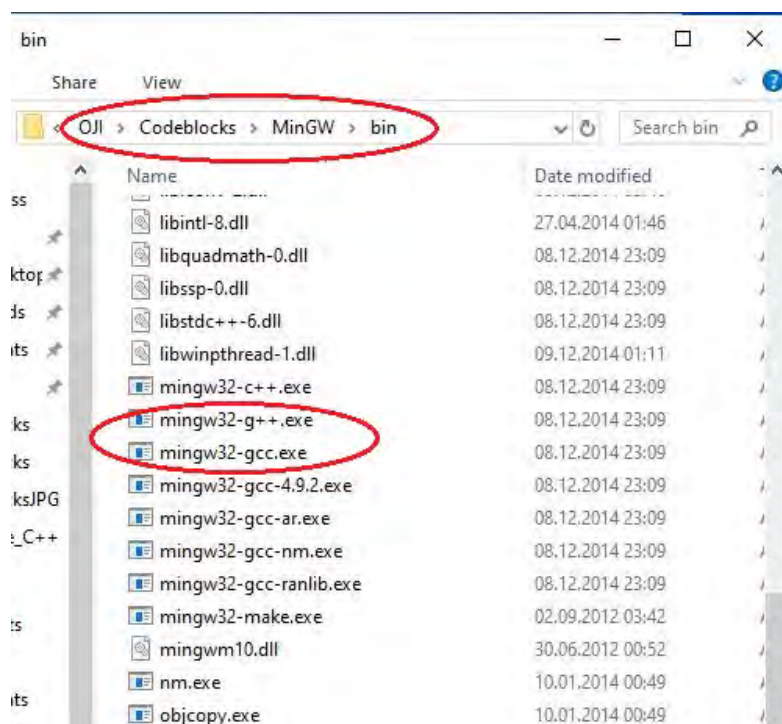


Figura A.17: Unde sunt acele programe



### A.1.5 Multe surse în Code Blocks

**Settings → Environment:** pe calculatorul meu setările sunt:

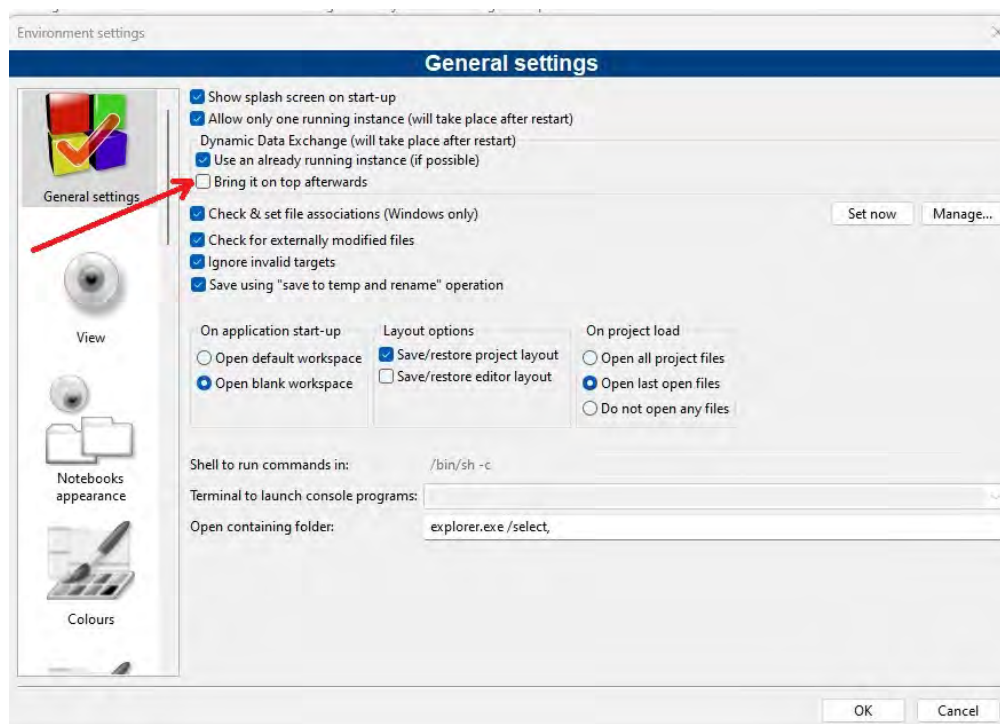


Figura A.18: Multe surse în Code Blocks - setări

Dacă avem fișierele p01.cpp, ..., p05.cpp, în folderul nostru de lucru, și facem dublu-click pe fiecare ... vor apărea toate ...

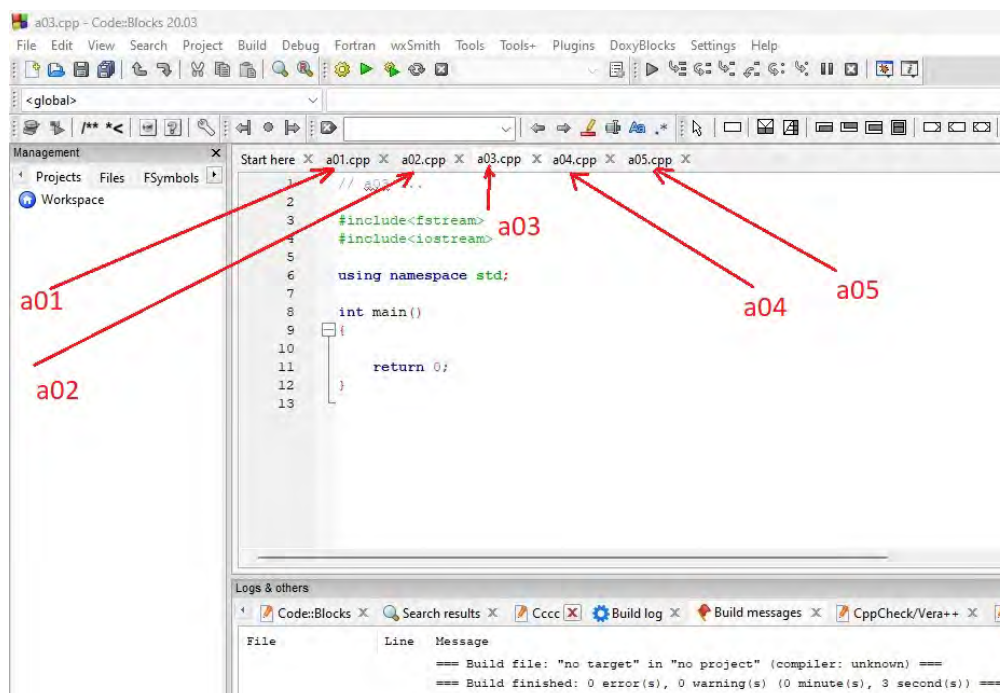


Figura A.19: Multe surse in Code Blocks - exemple

## A.2 winlibs

### A.2.1 GCC și MinGW-w64 pentru Windows

Se descarcă de la

<http://winlibs.com/#download-release>

unul dintre fișierele:

- winlibs-x86\_64-posix-seh-gcc-10.2.0-llvm-11.0.0-mingw-w64-8.0.0-r3.7z dimensiune fișier = 148 MB
- winlibs-x86\_64-posix-seh-gcc-10.2.0-llvm-11.0.0-mingw-w64-8.0.0-r3.zip dimensiune fișier = 324 MB
- winlibs-x86\_64-posix-seh-gcc-10.2.0-mingw-w64-8.0.0-r3.7z dimensiune fișier = 52.1 MB
- winlibs-x86\_64-posix-seh-gcc-10.2.0-mingw-w64-8.0.0-r3.zip dimensiune fișier = 141 MB

Se dezarchivează și se mută folderul **mingw64** pe C: sau D: sau ...

Eu l-am dezarhivat pe cel mai mic și l-am pus pe D:

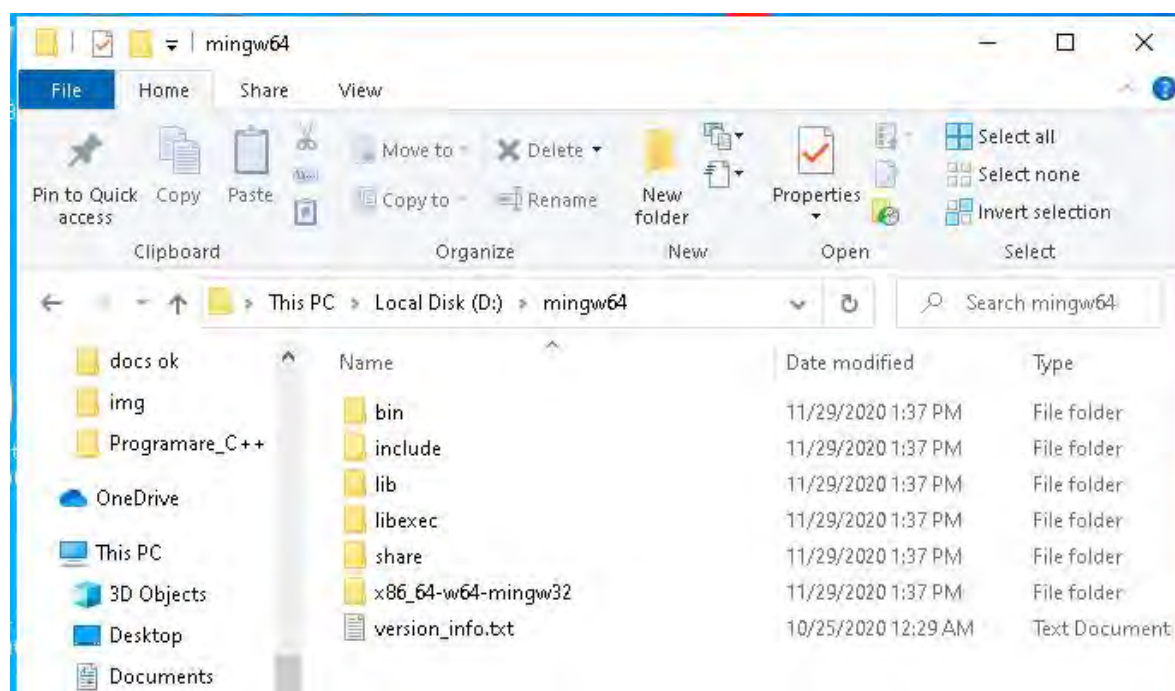
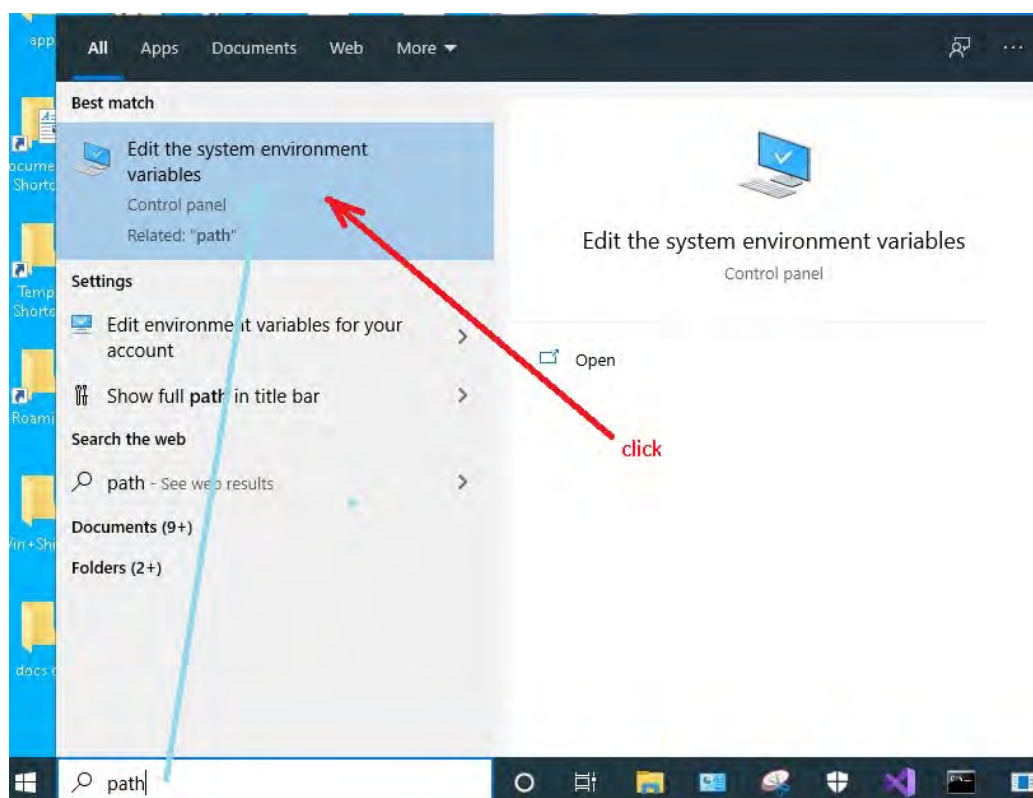


Figura A.20: mingw64 pe D:

### A.2.2 PATH

Trebuie pusă în PATH calea pentru D:\mingw64\bin\. În "Type here to search" scrieți: **path** și apoi click pe "**Edit the system environment variables**", ca în figura următoare:

Figura A.21: search **path**

Apare fereastra "System properties":

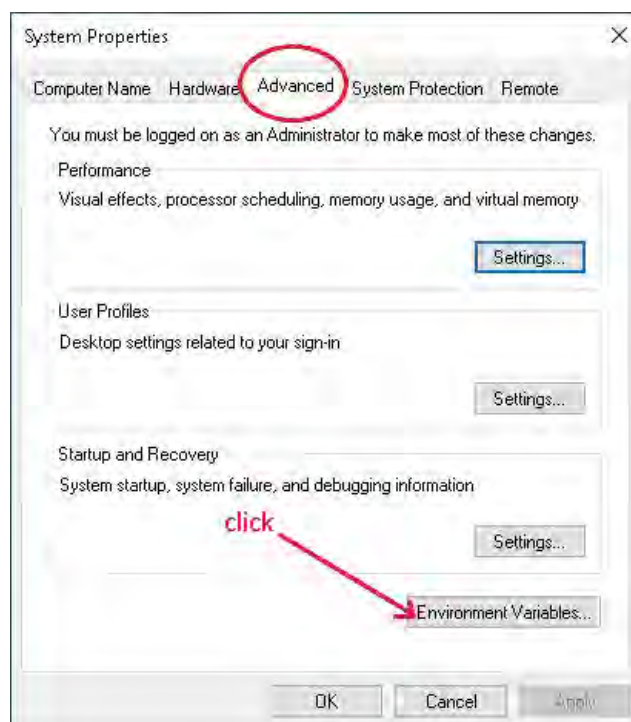


Figura A.22: System properties – Advanced

Fereastra este poziționată pe tab-ul "Advanced". Se selectează "**Environment Variables**".

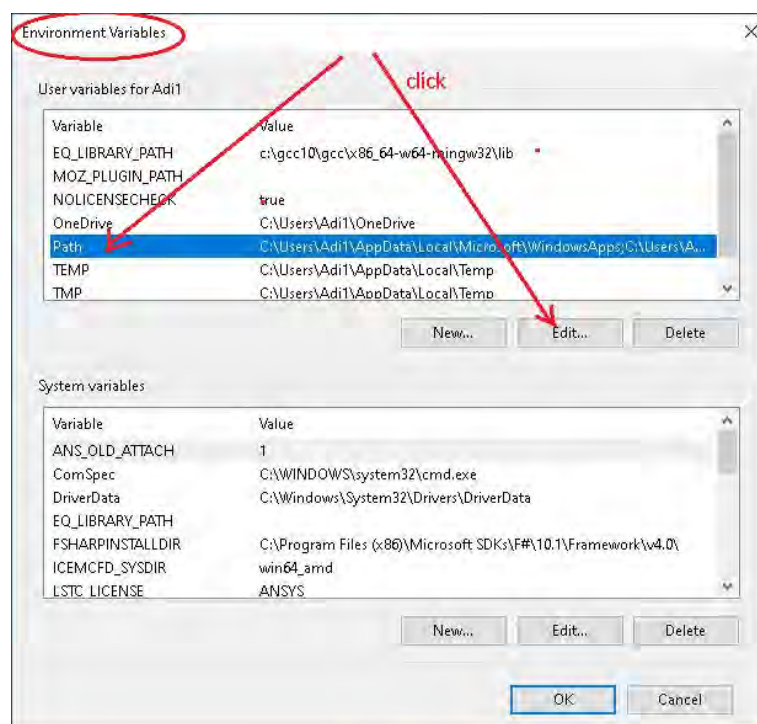


Figura A.23: Environment Variables

Se selecteaza "**Path**" și click pe "**Edit**". Apare fereastra "Edit Environment Variables"

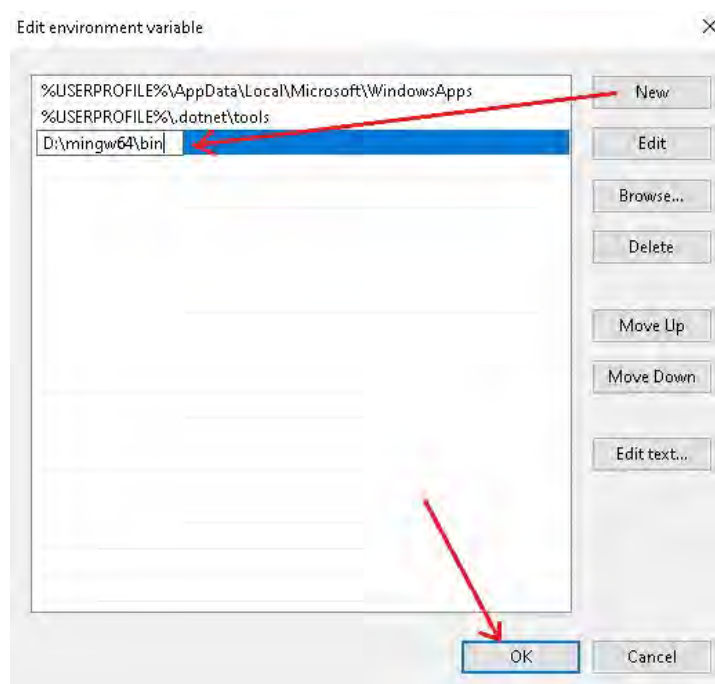


Figura A.24: Edit Environment Variables – New

Se selectează "**New**", se scrie calea **D:\mingw64\bin** și se finalizează cu click pe "**OK**", "**OK**", ..., "**OK**" până la sfârșit!

Se verifică calea și versiunea pentru "**gcc**":

- **C:\path**
- **C:\gcc –version** (Atentie! sunt 2 caractere - consecutive)



```

C:\>path
PATH=C:\WINDOWS\system32\WBEM;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Program Files\MikTeX 2.9\miktex\bin\x64\;C:\Program Files\dotnet\;C:\WINDOWS\System32\OpenSSH\;C:\Program Files (x86)\Common Files\Acronis\SnapAPI\;C:\Users\Adi1\AppData\Local\Microsoft\WindowsApps;C:\Program Files\Microsoft SQL Server\130\Tools\Binn\;C:\Program Files\Microsoft SQL Server\Client SDK\ODBC\170\Tools\Binn\;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\WINDOWS\System32\OpenSSH\;C:\Users\Adi1\AppData\Local\Microsoft\WindowsApps;C:\Users\Adi1\dotnet\tools;D:\mingw64\bin;

C:\>gcc --version
gcc MinGW-w64 x86_64-posix-seh, built by Brecht Sanders, 10.2.0
Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

```

Figura A.25: Calea și versiunea pentru gcc

Dacă totul este OK atunci se trece la instalarea **IDE**-ului preferat (Integrated Development Environment<sup>29</sup>), de exemplu **Code::Blocks 20.03** (sau Eclipse, Visual Studio Code, Dev C++, NetBeans, și altele).

#### Observatie: Pentru Windows 11

"Setting the path and variables in Windows 11

In the System > About window,  
click the Advanced system settings link  
at the bottom of the Device specifications section.

In the System Properties window,  
click the Advanced tab,  
then click the Environment Variables button  
near the bottom of that tab."

### A.2.3 CodeBlocks

CodeBlocks se poate descărca de la <http://www.codeblocks.org/downloads/26>. Sunt mai multe variante dar eu am descărcat numai **codeblocks-20.03-setup.exe** care are 35.7 MB. La instalare am lăsat totul implicit, deci ... **Next, Next, ..., Next** până la sfârșit!

La prima lansare în execuție este necesară setarea locației compilatorului de C++ (**gcc**-ul pe care tocmai l-am instalat!).

<sup>29</sup> [https://en.wikipedia.org/wiki/Integrated\\_development\\_environment](https://en.wikipedia.org/wiki/Integrated_development_environment)  
[https://ro.wikipedia.org/wiki/Mediu\\_de\\_dezvoltare](https://ro.wikipedia.org/wiki/Mediu_de_dezvoltare)

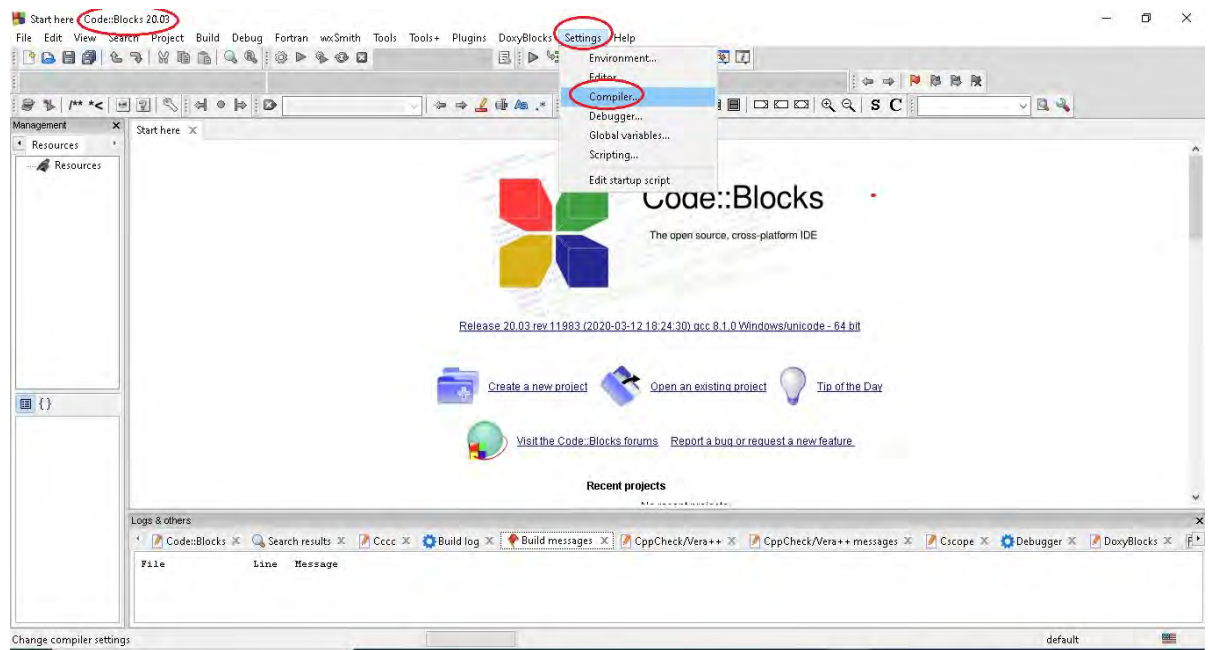


Figura A.26: Settings – Compiler

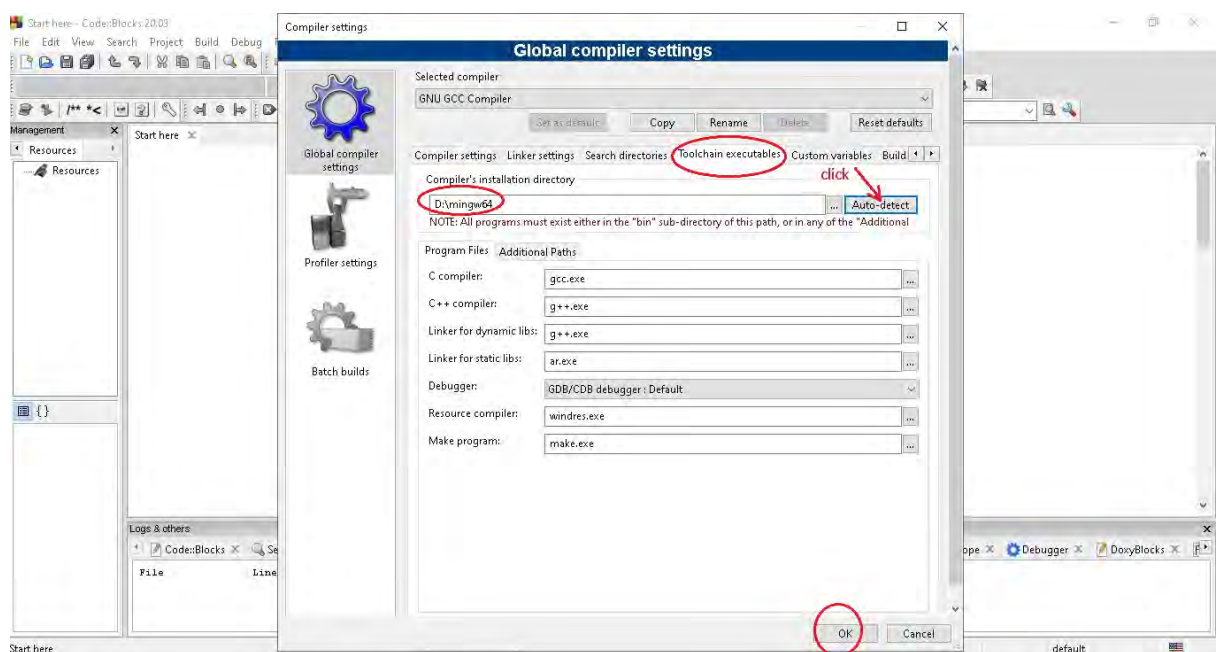


Figura A.27: Toolchain executables – Auto-detect

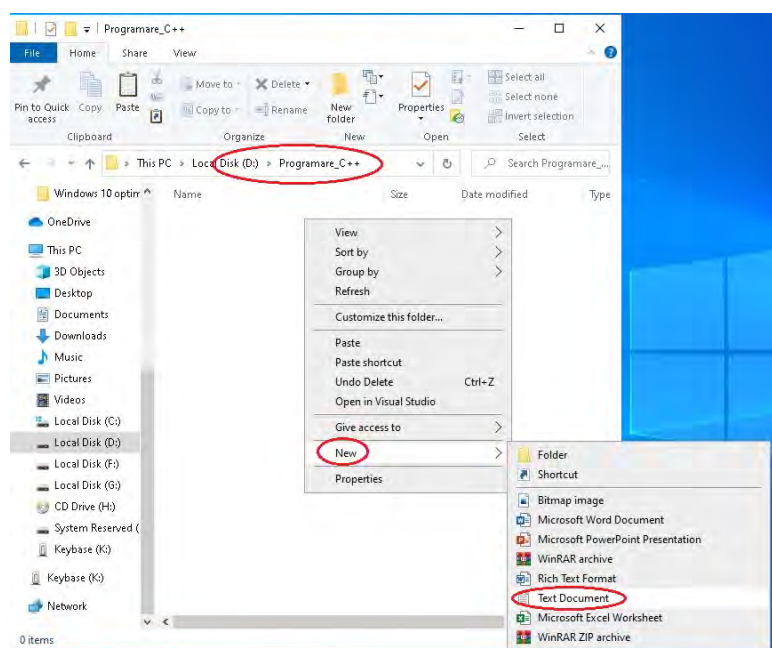


Figura A.28: New -&gt; Text Document

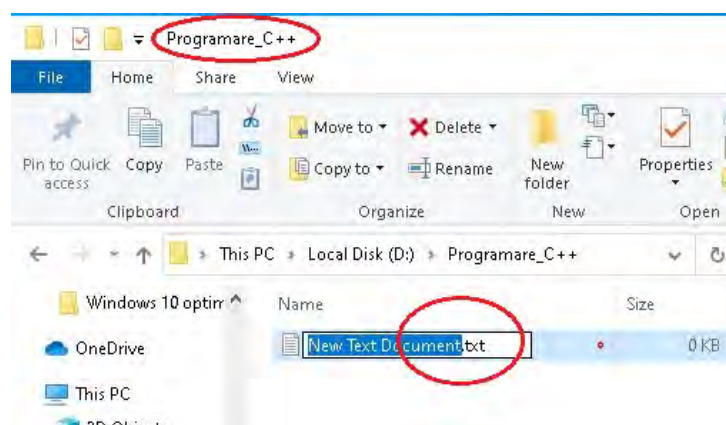


Figura A.29: New text Document.txt

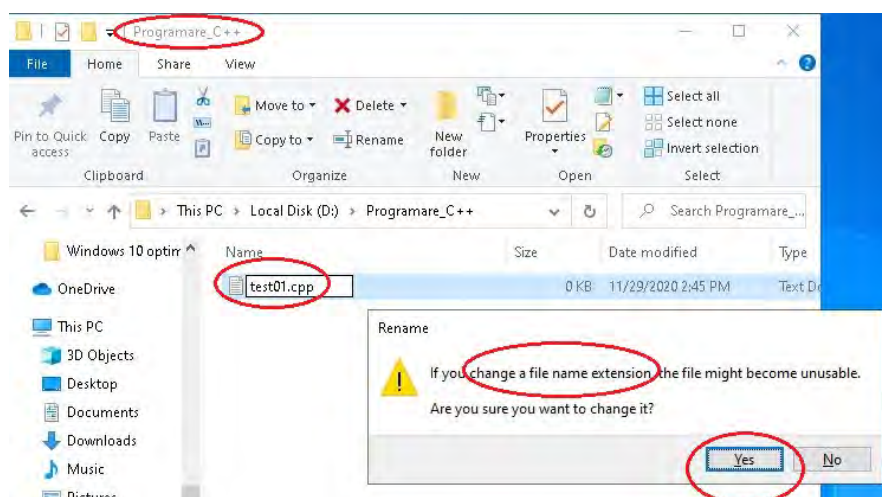


Figura A.30: Schimbare nume și extensie

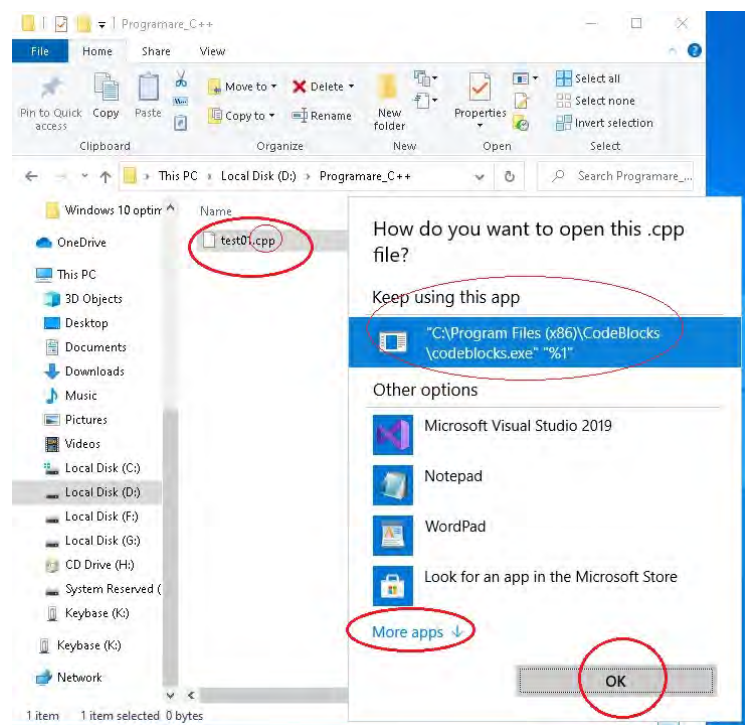


Figura A.31: Moore apps

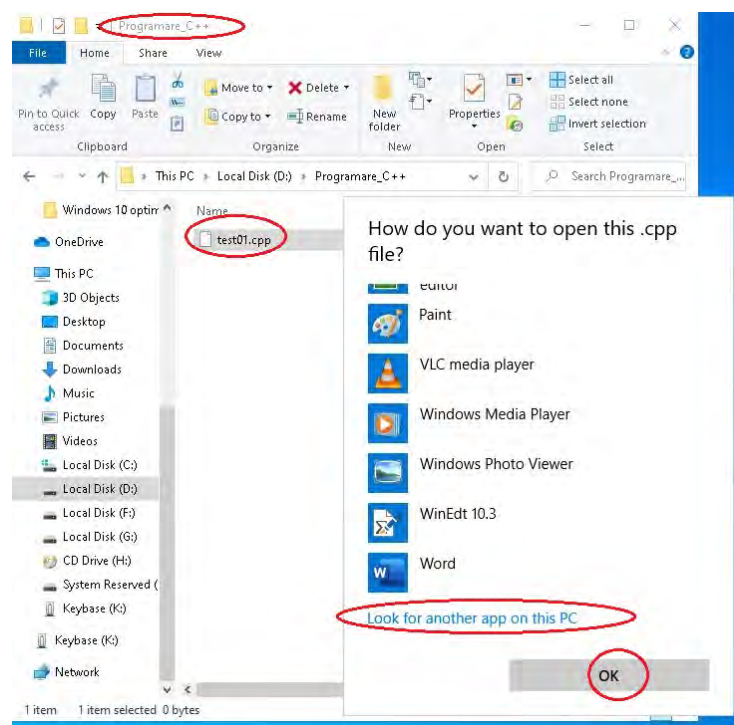


Figura A.32: Look for another app



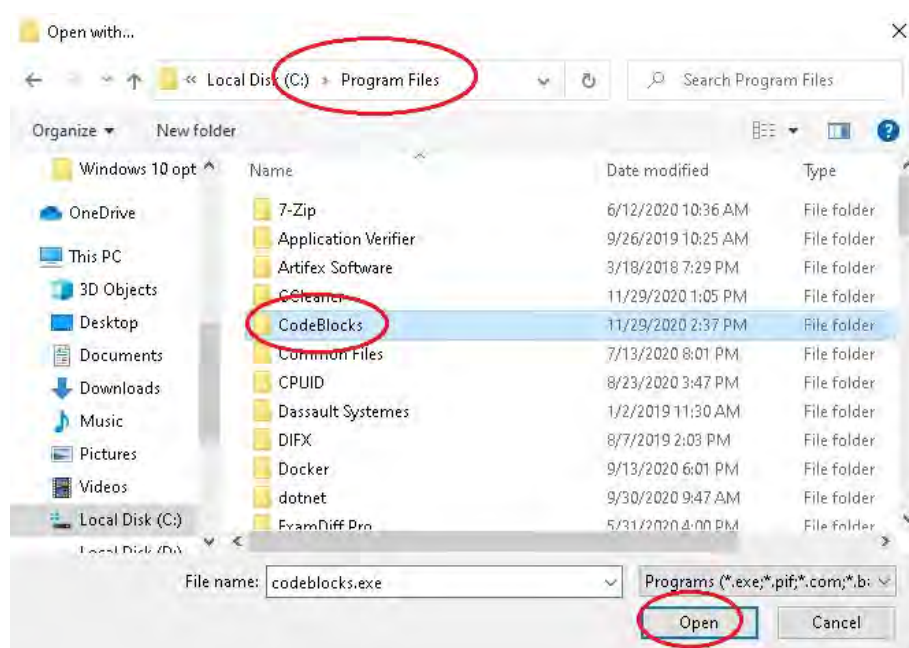


Figura A.33: Cale pentru codeblocks.exe

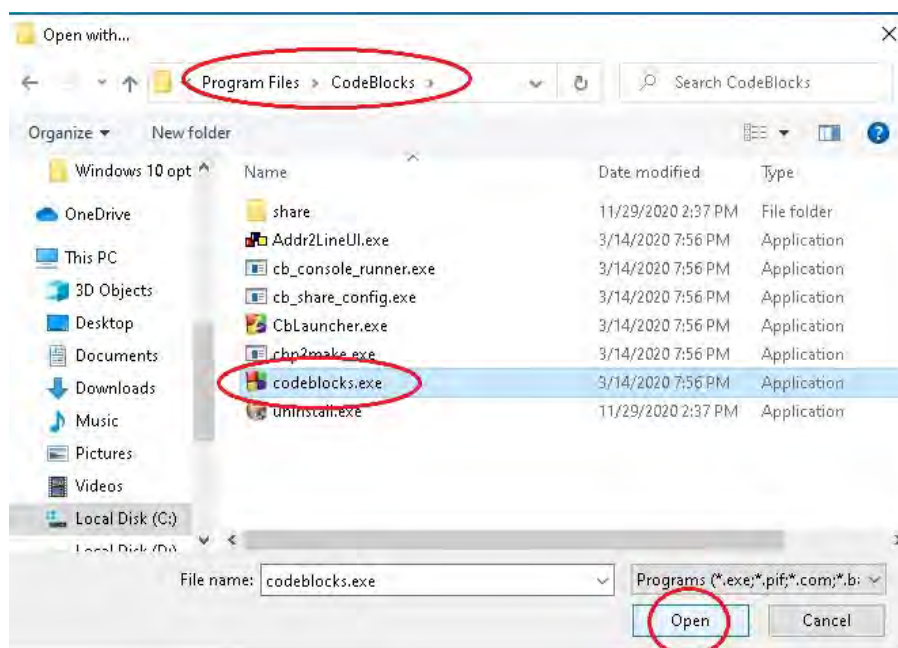


Figura A.34: Selectare codeblocks.exe

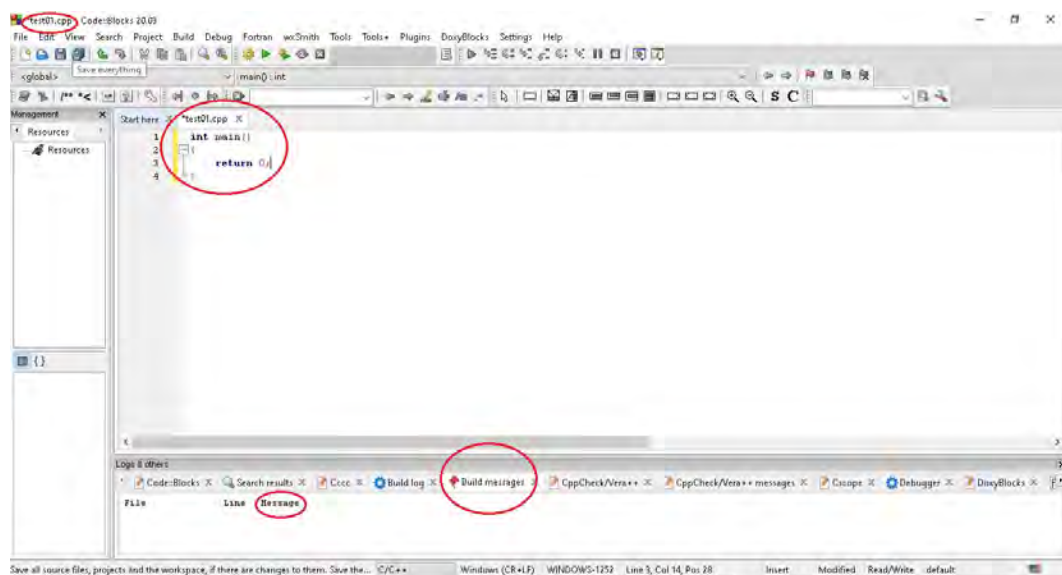


Figura A.35: Editare test01.cpp

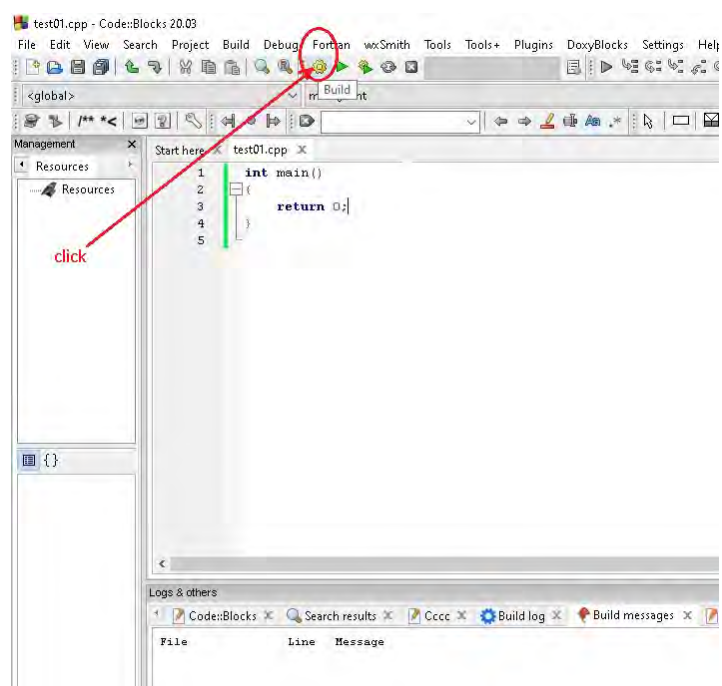


Figura A.36: Compilare test01

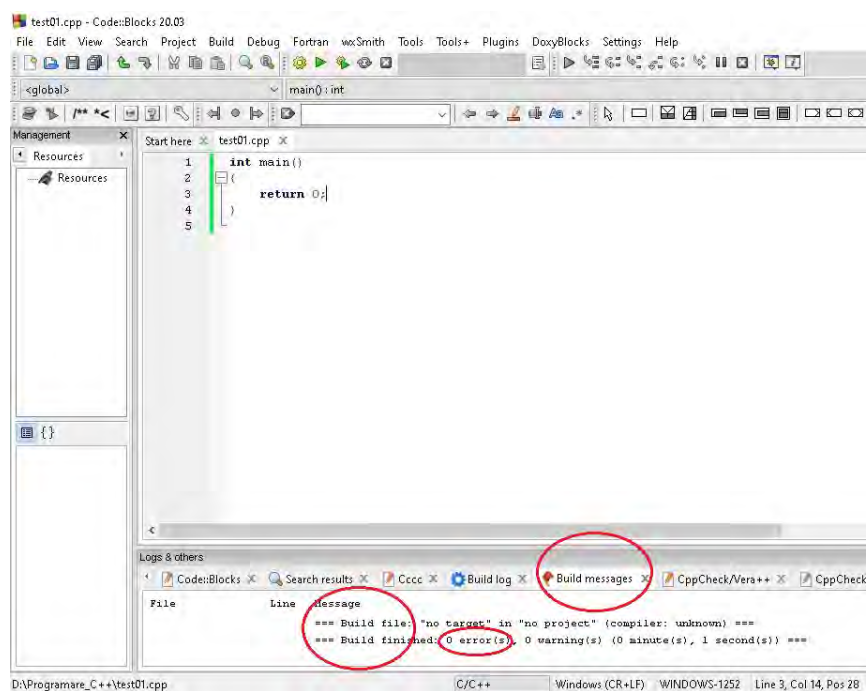
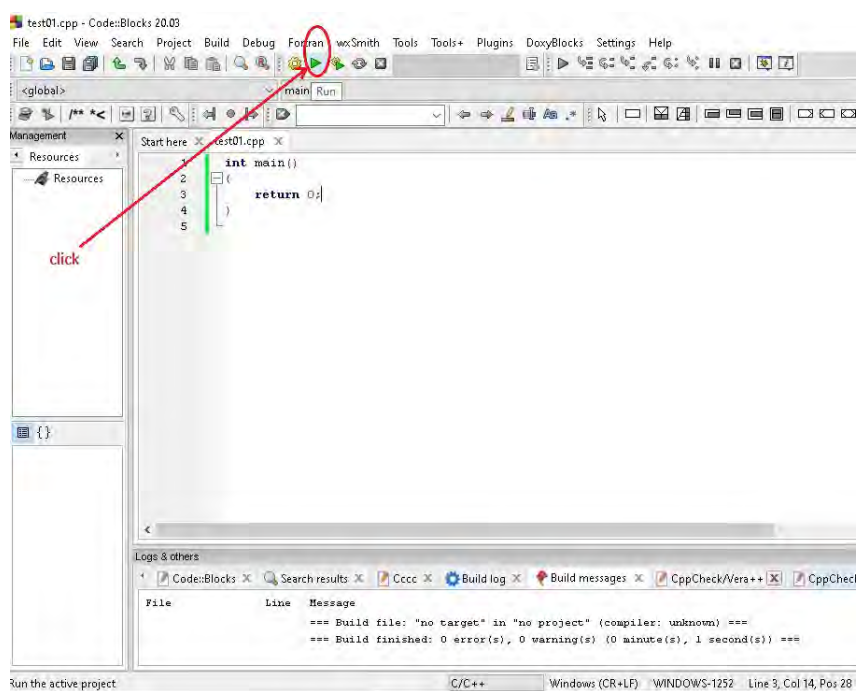


Figura A.37: Mesaje după compilare

Figura A.38: Execuție `test01`

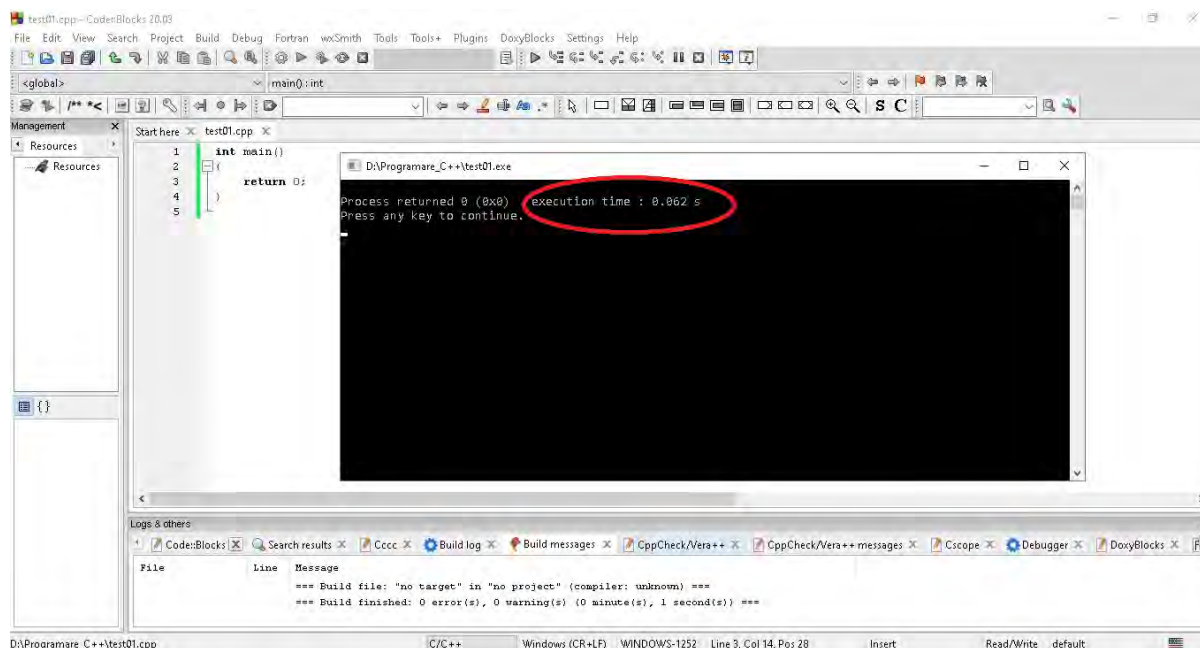
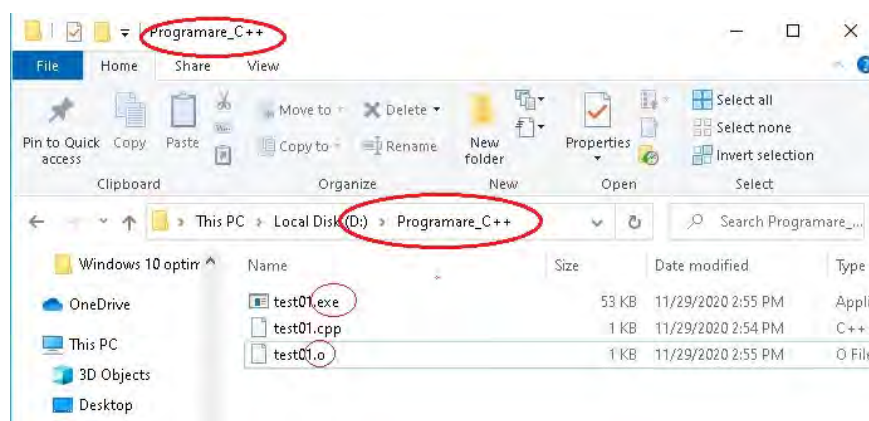
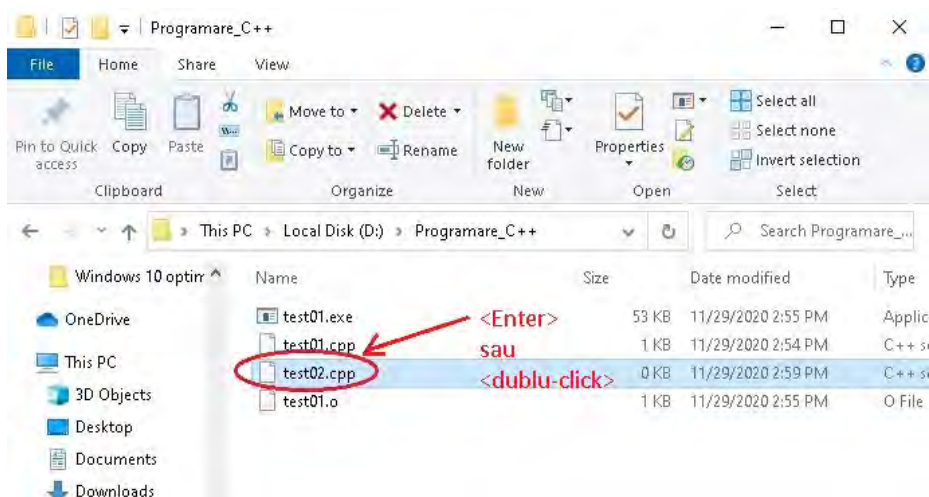
Figura A.39: Rezultat execuție **test01**

Figura A.40: Fișiere apărute după compilare!

Figura A.41: Creare **test02.cpp** gol! + {dublu click}, sau {Enter}

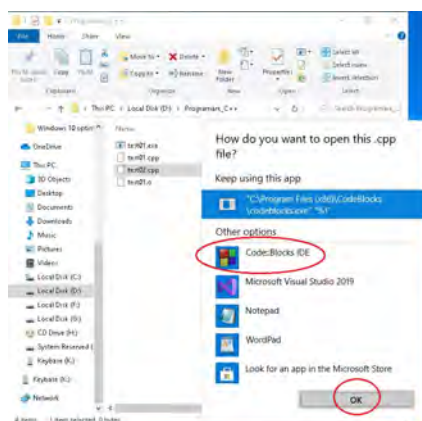
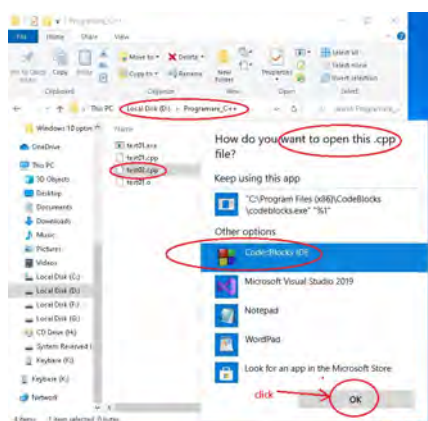
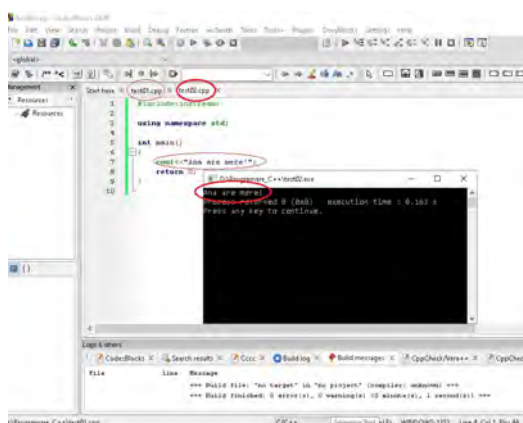
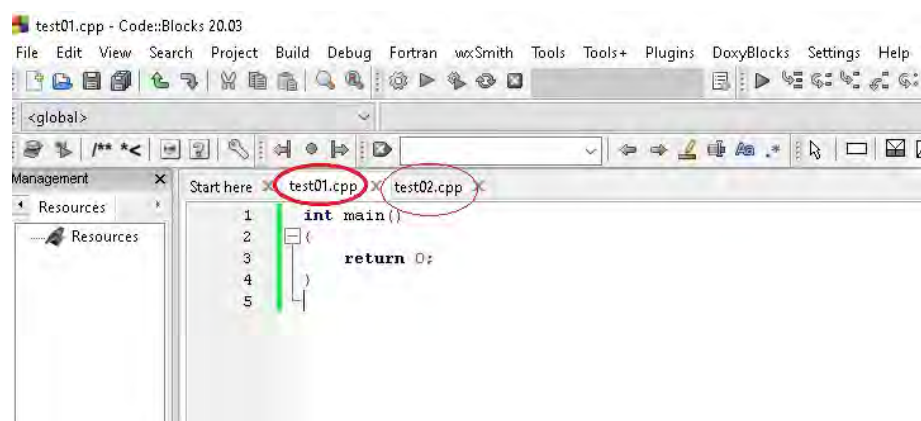


Figura A.42: Lista programelor de utilizat

Figura A.43: Selectare **Code::Blocks IDE** pentru fișierele .cppFigura A.44: Editare+Compilare+Execuție pentru **test02**



Figura A.45: Selectare **tab** ce conține **test01.cpp**

## Appendix B

# Exponențiere rapidă

### B.1 Analogie baza 2 cu baza 10

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2		in baza 10		nr/10	nr/10	nr/10						
3			nr=234	23	2	0	= catul(nr:10)					
4		ultima cifra	4	3	2		= rest(nr:10)		2	3	4	cifrele
5		din nr							*	*	*	inmultire
6									10^2	10^1	10^0	puterile bazei
7									100	10	1	
8									200	30	4	suma = 234
9		in baza 2		nr/2	nr/2	nr/2	nr/2	nr/2	nr/2	nr/2	nr/2	
10			nr=11101010	1110101	111010	11101	1110	111	11	1	0	= catul(nr:2)
11		ultima cifra	0	1	0	1	0	1	1	1		= rest(nr:2)
12		din nr										
13				1	1	1	0	1	0	1	0	cifrele
14				*	*	*	*	*	*	*	*	inmultire
15				2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	puterile bazei
16				128	64	32	16	8	4	2	1	
17				128	64	32	0	8	0	2	0	suma = 234

Figura B.1: Analogie B2 cu B10

În figura B.1 este considerat numărul  $nr = 234$  care în baza 2 se scrie sub forma 11101010 (celula C10).

Pe R3 (rândul 3) este arătat ce se întâmplă cu  $nr$  atunci când se fac împărțiri succesive prin  $baza = 10$ : se "șterge" ultima cifră (care este egală cu  $nr \% 10$  adică ultima cifră este de fapt restul împărțirii lui  $nr$  la  $baza = 10$ ).

Pe R10 (rândul 10) este arătat ce se întâmplă cu  $nr$  atunci când se fac împărțiri succesive prin  $baza = 2$ : se "șterge" ultima cifră (care este egală cu  $nr \% 2$  adică ultima cifră este de fapt restul împărțirii lui  $nr$  la  $baza = 2$ , la fel cum se întâmplă în baza 10).

**Observația 1.** Cifrele se obțin în ordine inversă. Prima cifră obținută ca rest este de fapt ultima cifră din număr (vezi R4 și R11).

Pe R6 și R16 sunt precizate puterile bazei.

Valoarea numărului  $nr$  este suma produselor dintre cifre și puterile corespunzătoare:

- în baza 10:  $4 \cdot 1 + 3 \cdot 10 + 2 \cdot 100 = 234$  (R3 și R6)
- în baza 2:  $0 \cdot 1 + 1 \cdot 2 + 0 \cdot 4 + 1 \cdot 8 + 0 \cdot 16 + 1 \cdot 32 + 1 \cdot 64 + 1 \cdot 128 = 234$  (R13 și R16)

## B.2 Notatii, relații și formule

$$\begin{aligned}
 a^{234} &= a^{1*128+1*64+1*32+0*16+1*8+0*4+1*2+0*1} = \\
 &= a^{1*128} * a^{1*64} * a^{1*32} * a^{0*16} * a^{1*8} * a^{0*4} * a^{1*2} * a^{0*1} = \\
 &= (a^{128})^1 * (a^{64})^1 * (a^{32})^1 * (a^{16})^0 * (a^8)^1 * (a^4)^0 * (a^2)^1 * (a^1)^0.
 \end{aligned}$$

Dacă notăm  $a_k = a^{2^k}$  atunci

$$a^{234} = (a_7)^1 * (a_6)^1 * (a_5)^1 * (a_4)^0 * (a_3)^1 * (a_2)^0 * (a_1)^1 * (a_0)^0.$$

Șirul  $a_0, a_1, a_2, \dots$  se obține ușor prin ridicări la putere ( $a_k = a_{k-1}^2$ ).

$$\begin{aligned}
 (a_0 = a) &\rightarrow (a_1 = a_0^2 = a^2) \rightarrow (a_2 = a_1^2 = a^4) \rightarrow (a_3 = a_2^2 = a^8) \rightarrow (a_4 = a_3^2 = a^{16}) \\
 &\rightarrow (a_5 = a_4^2 = a^{32}) \rightarrow (a_6 = a_5^2 = a^{64}) \rightarrow (a_7 = a_6^2 = a^{128})
 \end{aligned}$$

Dacă notăm  $e_k =$  "exponentul (puterea)" la care apare  $a_k$  în produs, atunci putem observa ușor că șirul  $e_0, e_1, e_2, e_3, \dots$  se obține prin împărțiri succesive ale lui  $n$  prin 2 și preluând restul rezultatului. Pentru a formaliza acest lucru vom considera șirul  $n_0, n_1, n_2, n_3, \dots$  definit astfel:

$$\begin{cases} n_0 = n, \\ n_k = n_{k-1}/2 \text{ (câtul împărțirii!)} \quad k \geq 1 \end{cases}$$

Folosind aceste notații, putem scrie:

$$e_k = n_k \% 2, \text{ (restul împărțirii!)} \quad k \geq 0$$

Dacă notăm și produsul parțial  $p_k = (a_k)^{e_k} * \dots * (a_1)^1 * (a_0)^0$  obținem în final relațiile:

$$\begin{cases} n_0 = n; \\ a_0 = a; \\ e_0 = n_0 \% 2 \in \{0, 1\}; \\ p_0 = a_0^{e_0} \text{ sau, altfel scris: } p_0 = \begin{cases} 1 * a_0, & \text{dacă } e_0 = 1; \\ 1, & \text{dacă } e_0 = 0; \end{cases} \\ \\ n_k = n_{k-1}/2 \quad k \geq 1; \text{ ( } n_{k-1} \neq 0 \text{ desigur! ... dar și } n_{k-1} \neq 1 \text{ 😊)} \\ a_k = a_{k-1}^2, \quad k \geq 1; \\ e_k = n_k \% 2 \in \{0, 1\}, \quad k \geq 0; \\ p_k = \begin{cases} p_{k-1} * a_k, & k \geq 0, \text{ dacă } e_k = 1; (a_k^1 = a_k \text{ modifică produsul } p_{k-1}) \\ p_{k-1}, & k \geq 0, \text{ dacă } e_k = 0; (a_k^0 = 1 \text{ nu modifică produsul } p_{k-1}) \end{cases} \end{cases} \quad (\text{B.2.1})$$

## B.3 Pregătire pentru scrierea codului!

Relația  $n_k = n_{k-1}/2$  înseamnă că  $n_{nou} = n_{vechi}/2$  (sunt explicații pentru începători ... nu pentru avansați!) și se poate programa sub forma  $n=n/2$ ;

Relația  $a_k = a_{k-1}^2$  înseamnă că  $a_{nou} = a_{vechi}^2$  și se poate programa sub forma  $a=a*a$ ;

Relația  $p_k = p_{k-1} * a_k$  înseamnă că  $p_{nou} = p_{vechi} * a_{nou}$  și se poate programa sub forma  $p=p*a$ ;

Relația  $p_k = p_{k-1}$  înseamnă că  $p_{nou} = p_{vechi}$  și se poate programa sub forma  $p=p$ ; care înseamnă că nu se schimbă  $p$ , deci ... mai bine nu mai scriem nicio instrucțiune! 😊



## B.4 Codul

Codul pentru relațiile (B.2.1) devine:

Listing B.4.1: exponentiere\_rapida1.cpp

---

```

1  #include<iostream> // actualizare p dupa actualizare a si n
2
3  using namespace std;
4
5  int exponentiere_rapida(int a, int n) // p=a^n
6  {
7      int nk1, nk;
8      int ak1, ak;
9      int ek1, ek;
10     int pk1, pk;
11
12     int k=0;
13     nk1=n;
14     ak1=a;
15     ek1=nk1%2;
16     if(ek1==1)
17         pk1=ak1;
18     else
19         pk1=1;
20
21     while(nk1>1)
22     {
23         k++;
24         nk=nk1/2;
25         ak=ak1*ak1;
26         ek=nk%2;
27         if(ek==1)
28             pk=pk1*ak;
29         else
30             pk=pk1;
31
32         // devin valori "vechi" inainte de noua actualizare
33         nk1=nk;
34         ak1=ak;
35         ek1=ek;
36         pk1=pk;
37     }
38
39     return pk;
40 }
41
42 int main()
43 {
44     int a=2;
45     //int n=234; // a^n = prea mare !!!
46     //int n=30;
47     //int n=6; // par: 2^6 = 64 ... usor de verificat!
48     int n=5; // impar: 2^5 = 32 ... usor de verificat!
49
50     int rez=exponentiere_rapida(a,n);
51
52     cout<<a<<"^"<<n<<" = "<<rez<<"\n";
53
54     return 0;
55 }
56 /*
57 2^30 = 1073741824
58
59 Process returned 0 (0x0) execution time : 0.076 s
60 Press any key to continue.
61 */

```

---

**Observația 2.** În acest cod actualizarea lui  $p$  se face după actualizările pentru  $a$  și  $n$ .

**Observația 3.** În codul următor actualizarea lui  $p$  se face înaintea actualizărilor pentru  $a$  și  $n$ , corespunzător relațiilor (B.4.2).

Listing B.4.2: exponentiere\_rapida2.cpp

---

```

1  #include<iostream> // actualizare p inainte de actualizare a si n
2
3  using namespace std;
4
5  int exponentiere_rapida(int a, int n) // p=a^n
6  {
7      int nk1, nk;
8      int ak1, ak;
9      int ek1, ek;
10     int pk1, pk;
11
12     nk1=n;
13     ak1=a;
14     pk1=1;
15
16     while(nk1>0)
17     {
18         ek=nk1%2;
19         if(ek==1)
20             pk=pk1*ak1;
21         else
22             pk=pk1;
23         ak=ak1*ak1;
24         nk=nk1/2;
25
26         // devin valori "vechi" inainte de noua actualizare
27         nk1=nk;
28         ak1=ak;
29         pk1=pk;
30     }
31
32     return pk;
33 }
34
35 int main()
36 {
37     int a=2;
38     //int n=234; // a^n = prea mare !!!
39     //int n=30;
40     int n=6; // par: 2^6 = 64 ... usor de verificat!
41     //int n=5; // impar: 2^5 = 32 ... usor de verificat!
42
43     int rez=exponentiere_rapida(a,n);
44
45     cout<<a<<"^"<<n<<" = "<<rez<<"\n";
46
47     return 0;
48 }
49 /*
50 2^30 = 1073741824
51
52 Process returned 0 (0x0)   execution time : 0.076 s
53 Press any key to continue.
54 */

```

---


$$\begin{cases}
 \text{inițializări:} \\
 p_{-1} = 1; \\
 n_{-1} = n; \\
 a_{-1} = a; \\
 \text{calcul pentru } k \geq 0: \\
 e_k = n_{k-1} \% 2 \in \{0, 1\}; k \geq 0 \\
 p_k = \begin{cases} p_{k-1} * a_{k-1}, & k \geq 0, \text{ dacă } e_k = 1; (a_{k-1}^1 = a_{k-1} \text{ modifică produsul } p_{k-1}) \\ p_{k-1}, & k \geq 0, \text{ dacă } e_k = 0; (a_{k-1}^0 = 1 \text{ nu modifică produsul } p_{k-1}) \end{cases} \\
 \text{actualizări } (k \geq 0): \\
 a_k = a_{k-1}^2, \quad k \geq 0; \\
 n_k = n_{k-1}/2 \quad k \geq 0; (\text{și } n_{k-1} \neq 0 \text{ desigur!})
 \end{cases} \quad (\text{B.4.2})$$

**Observația 4.** Instrucțiunile care sunt "în plus" se pot elimina. Codul următor arată acest lucru:

Listing B.4.3: exponentiere\_rapida3.cpp

---

```

1  #include<iostream> // actualizare p inainte de actualizare a si n
2                          // ... si simplificarea codului ...
3
4  using namespace std;
5
6  int exponentiere_rapida(int a, int n) // p=a^n
7  {
8      int nk;
9      int ak;
10     int ek;
11     int pk;
12
13     nk=n;
14     ak=a;
15     pk=1;
16
17     while(nk>0)
18     {
19         ek=nk%2;
20         if(ek==1)
21             pk=pk*ak;
22         else
23             pk=pk; // nu are rost ... !!!
24         ak=ak*ak;
25         nk=nk/2;
26     }
27
28     return pk;
29 }
30
31 int main()
32 {
33     int a=2;
34     //int n=234; // a^n = prea mare !!!
35     //int n=30;
36     int n=6; // par: 2^6 = 64 ... usor de verificat!
37     //int n=5; // impar: 2^5 = 32 ... usor de verificat!
38
39     int rez=exponentiere_rapida(a,n);
40
41     cout<<a<<"^"<<n<<" = "<<rez<<"\n";
42
43     return 0;
44 }
45 /*
46 2^30 = 1073741824
47
48 Process returned 0 (0x0)   execution time : 0.076 s
49 Press any key to continue.
50 */

```

---

**Observația 5.** Produsul poate deveni foarte mare și din cauza asta se cere rezultatul modulo un număr prim. Codul următor arată acest lucru:

Listing B.4.4: exponentiere\_rapida\_MOD.cpp

---

```

1  #include<iostream>
2
3  using namespace std;
4
5  int MOD=1000;
6  int nropr; // numar operatii (inmultiri) la exponentiere rapida
7
8  int exponentiere_rapida(int a, int n) // p=a^n
9  {
10     int p = 1;
11     while(n>0)
12     {
13         if(n % 2 == 1) p = (p * a)%MOD;

```

---

---

```

14     a = (a * a)%MOD;
15     n = n / 2;
16     nropr=nropr+6; // n%2, p*a, a*a, (a * a)%MOD si n/2
17 }
18 return p;
19 }
20
21 int main()
22 {
23     int a=1234;
24     int n=1e+9; // 10^9
25     int rezn; // rezultat exponentiere naiva
26
27     rezn=exponentiere_rapida(a,n);
28
29     cout<<"rezn = "<<rezn<<"    nropr = "<<nropr<<"\n";
30
31     return 0;
32 }
33 /*
34 rezn = 376    nropr = 180
35
36 Process returned 0 (0x0)    execution time : 0.021 s
37 Press any key to continue.
38 */

```

---

## B.5 Chiar este rapidă?

De ce se numește ”rapidă”?

Să presupunem că vrem să calculăm  $1234^{1\,000\,000\,000}$  și pentru că rezultatul are foarte multe cifre ... ne vom mulțumi, la fiecare calcul, cu ultimele 3 cifre!

Aceasta este metoda ”naivă”:

Listing B.5.1: exponentiere\_naiva\_MOD.cpp

---

```

1  #include<iostream>
2
3  using namespace std;
4
5  int MOD=1000;
6  int nropr=0; // numar operatii (inmultiri) la exponentiere naiva
7
8  int exponentiere_naiva(int a, int n) // p=a^n
9  {
10     int p = 1;
11     for(int k=1; k<=n; k++)
12     {
13         p=(p*a)%MOD;
14         nropr=nropr+1;
15     }
16     return p;
17 }
18
19 int main()
20 {
21     int a=1234;
22     int n=1e+9; // 10^9
23     int rezn; // rezultat exponentiere naiva
24
25     rezn=exponentiere_naiva(a,n);
26
27     cout<<"rezn = "<<rezn<<"    nropr = "<<nropr<<"\n";
28
29     return 0;
30 }
31 /*
32 rezn = 376    nropr = 2000000000
33
34 Process returned 0 (0x0)    execution time : 21.239 s
35 Press any key to continue.
36 */

```

---

Iar aceasta este metoda ”rapidă”:

Listing B.5.2: exponentiere\_rapida\_MOD.cpp

```

1  #include<iostream>
2
3  using namespace std;
4
5  int MOD=1000;
6  int nropr; // numar operatii (inmultiri) la exponentiere rapida
7
8  int exponentiere_rapida(int a, int n) // p=a^n
9  {
10     int p = 1;
11     while(n>0)
12     {
13         if(n % 2 == 1) p = (p * a)%MOD;
14         a = (a * a)%MOD;
15         n = n / 2;
16         nropr=nropr+6; // n%2, p*a, ..%MOD*a, (a * a) ... %MOD si n/2
17     }
18     return p;
19 }
20
21 int main()
22 {
23     int a=1234;
24     int n=1e+9; // 10^9
25     int rezn; // rezultat exponentiere naiva
26
27     rezn=exponentiere_rapida(a,n);
28
29     cout<<"rezn = "<<rezn<<"    nropr = "<<nropr<<"\n";
30
31     return 0;
32 }
33 /*
34 rezn = 376    nropr = 180
35
36 Process returned 0 (0x0)    execution time : 0.021 s
37 Press any key to continue.*/

```

### Numărul de operații:

- cu metoda naivă acest număr este 2 000 000 000
- cu metoda rapidă este 180.

### Timpul de execuție (pe calculatorul pe care am testat eu!):

- cu metoda naivă este 21.239 secunde
- cu metoda rapidă este 0.021 secunde

deci ... cam de 1000 de ori mai rapidă!

Iar ca număr de operații ... una este să faci 2 miliarde 🤔 de operații și alta este să faci numai 180 😊 de operații de același tip (de fapt sunt numai 30 de pași dar la fiecare pas se fac 5 sau 6 operații aritmetice)!

## B.6 Rezumat intuitiv!

Revenind la relația

$$a^{234} = (a^{128})^1 * (a^{64})^1 * (a^{32})^1 * (a^{16})^0 * (a^8)^1 * (a^4)^0 * (a^2)^1 * (a^1)^0$$

să privim cu atenție ce este aici!

Putem calcula ușor acest produs de la dreapta spre stânga!

Secvența  $a^2, a^4, a^8, a^{16}, a^{32}, a^{64}, a^{128}$  se poate genera cu instrucțiunea  $a=a*a$ ;

Puterile expresiilor (marcate cu roșu, cele care sunt între paranteze) se obțin (tot de la dreapta spre stânga) prin împărțiri succesive ale lui  $n$  la 2 și preluând restul. Dacă restul este 0 atunci puterea respectivă este 0 iar  $(\dots)^0 = 1$  ... deci, nu mai apare în produs!

Asta este tot! 😄

Deci, secvența de cod este:

Listing B.6.1: secventa\_cod.cpp

```
1  int p = 1;
2  while (n > 0)
3  {
4      if (n % 2 == 1) p = (p * a);
5      a = a * a;
6      n = n / 2;
7  }
```

și nu mai trebuie<sup>30</sup> ”atâtea formule matematice”! 😄

---

<sup>30</sup> Este o glumă!

# Index

- `_builtin_parity`, 82
- șirul lui Fibonacci, 173
- algoritm de umplere, 162, 185
- algoritm de umplere (fill), 70
- algoritm FILL, 235
- algoritm Lee, 120, 147, 235
- algoritmul FILL, 245
- algoritmul fill, 185
- algoritmul Lee, 91, 208
- algoritmul lui Lee, 6, 12
- algoritmul nth\_element, 220
- atoi, 241, 242
- backtracking, 6
- bactracking, 9
- `begin()`, 122, 126, 130
- bool, 122, 126, 130
- căutare binară, 91, 180
- char, 124, 130
- CINOR, vi
- Ciurul lui Eratostene, 17
- ciurul lui Eratostene, 17
- cmmdc, 208
- coadă, 70, 235
- const, 122, 126
- deque, 122, 126, 130
- distanța Manhattan, 3
- divide et impera, 191
- drum de lungime minimă, 6
- `empty()`, 122, 126, 153
- first, 122, 126
- forma canonică, 73
- `front()`, 122, 126, 153
- `get()`, 122, 124
- `getline`, 124, 224, 241, 242
- greater, 130
- Greedy, 134
- I.d.k.:
  - I don't know who the author is., v
- inline, 122, 126, 130
- lambda, 122, 126, 130
- Lee, 122, 124, 151, 153
- lexicografic, 3
- Liceul Militar Dimitrie Cantemir, vi
- liste înlănțuite, 56
- liste circulare, 56
- liste dublu înlănțuite, 56
- `make_pair`, 124
- mediana, 224
- mediana unui vector, 219
- MergeSort, 219
- metoda programării dinamice, 228
- minim local, 135
- normalizare, 9
- numere mari, 198, 228
- operator, 130
- pair, 122, 124, 130
- parsare, 220
- partiție, 179
- perioada, 5
- `pop()`, 122
- precalcul, 3, 17, 36
- precalculare, 12
- priority\_queue, 130
- progresia geometrică, 173
- proprietatea anti-triunghi, 173
- push, 122, 153
- `push_back`, 122, 126, 130
- `push_front`, 122, 126, 130
- qsort, 242
- queue, 122, 126, 153
- QuickSort, 219
- quicksort, 219
- recursivitate indirectă, 26, 220
- second, 122, 126
- secvență, 179
- secvență de sumă maximă, 219
- segment de stivă, 246
- short, 122, 124
- simulare, 17
- `size()`, 122, 126, 130
- sizeof, 242
- sort, 122, 126, 130
- stack, 138, 142
- stivă, 28, 36

- stiva, 134, 219
- STL, 219
- strcat, 241, 242
- strchr, 241, 242
- strcmp, 241, 242
- strcpy, 241, 242
- string, 126, 130
- strlen, 124, 224, 241
- strtok, 241, 242
- struct, 130, 153, 241, 242
- structură, 239
- subșir, 179
- suma divizorilor, 17
- swap, 224
- tablou Young, 57
- trage cu tunul, iv
- triunghiul lui Pascal, 228
- two pointers technique, 56
- Universitatea Ovidius, vi
- using, 126
- vector, 122, 126, 130
- vector de frecvență, 17, 20
- vector de sume, 15



# Bibliografie

- [1] Aho, A., Hopcroft, J., Ullman, J.D.; Data structures and algorithms, Addison Wesley, 1983
- [2] Andreica M.I.; Elemente de algoritmică - probleme și soluții, Cibernetica MC, 2011
- [3] Andonie R., Gârbacea I.; Algoritmi fundamentali, o perspectivă C++, Ed. Libris, 1995
- [4] Atanasiu, A.; Concursuri de informatică. Editura Petriom, 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] Răbăea, A.; [https://math.univ-ovidius.ro/Doc/Admitere/CentruPregatire/2007/Info/Lista\\_probleme\\_2000-2007.pdf](https://math.univ-ovidius.ro/Doc/Admitere/CentruPregatire/2007/Info/Lista_probleme_2000-2007.pdf)
- [30] Răbăea, A.; [https://math.univ-ovidius.ro/Doc/Admitere/CentruPregatire/2007/Info/Rezolvari\\_C09.pdf](https://math.univ-ovidius.ro/Doc/Admitere/CentruPregatire/2007/Info/Rezolvari_C09.pdf)
- [31] Răbăea, A.; [https://math.univ-ovidius.ro/Doc/Admitere/CentruPregatire/2007/Info/Rezolvari\\_C10.pdf](https://math.univ-ovidius.ro/Doc/Admitere/CentruPregatire/2007/Info/Rezolvari_C10.pdf)
- [32] Răbăea, A.; [https://math.univ-ovidius.ro/Doc/Admitere/CentruPregatire/2007/Info/Rezolvari\\_C11.pdf](https://math.univ-ovidius.ro/Doc/Admitere/CentruPregatire/2007/Info/Rezolvari_C11.pdf)
- [33] Răbăea, A.; [https://math.univ-ovidius.ro/Doc/Admitere/CentruPregatire/2007/Info/Rezolvari\\_Baraj.pdf](https://math.univ-ovidius.ro/Doc/Admitere/CentruPregatire/2007/Info/Rezolvari_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] Vlada, M.; Gândirea Algoritmică - O Filosofie Modernă a Matematicii și Informaticii, CNIV-2003, Editura Universității din București, 2003
- [44] Weis, M.A.; Data structures and Algorithm Analysis, Ed. The Benjamin/Cummings Publishing Company. Inc., Redwoods City, California, 1995.
- [45] Winston, P.H., Narasimhan, S.; On to JAVA, Addison-Wesley, 1996
- [46] Wirth N.; Algorithms + Data Structures = Programs, Prentice Hall, Inc 1976
- [47] \*\*\* - Gazeta de Informatică, Editura Libris, 1991-2005
- [48] \*\*\* - [https://github.com/DinuCr/CS/blob/master/Info/stuff%20stuff/Rezolvari\\_C09.pdf](https://github.com/DinuCr/CS/blob/master/Info/stuff%20stuff/Rezolvari_C09.pdf)
- [49] \*\*\* - <https://dokumen.tips/documents/rezolvaric09.html>
- [50] \*\*\* - <https://www.scribd.com/doc/266218102/Rezolvari-C09>
- [51] \*\*\* - <https://www.scribd.com/document/396362669/Rezolvari-C10>
- [52] \*\*\* - <https://www.scribd.com/document/344769195/Rezolvari-C11>
- [53] \*\*\* - <https://www.scribd.com/document/364077679/Rezolvari-C11-pdf>
- [54] \*\*\* - <https://needoc.net/rezolvari-c11-pdf>

- [55] \*\*\* - <https://vdocumente.com/algorithmi-i-structuri-de-date.html>
- [56] \*\*\* - <https://pdfslide.net/documents/algorithmi-si-structuri-de-date-1-note-de-cuprins-1-oji-2002-clasa-a-ix-a-1-11.html>
- [57] \*\*\* - <https://www.infoarena.ro/ciorna>
- [58] \*\*\* - <https://infoarena.ro/olimpici>
- [59] \*\*\* - <https://www.infogim.ro/>
- [60] \*\*\* - <https://www.pbinfo.ro/>
- [61] \*\*\* - <http://www.cplusplus.com/>
- [62] \*\*\* - <http://www.cplusplus.com/doc/tutorial/operators/>
- [63] \*\*\* - <http://www.infolcup.com/>
- [64] \*\*\* - <http://www.olimpiada.info/>
- [65] \*\*\* - <http://www.usaco.org/>
- [66] \*\*\* - <http://algopedia.ro/>
- [67] \*\*\* - <http://campion.edu.ro/>
- [68] \*\*\* - <http://varena.ro/>
- [69] \*\*\* - [http://rmi.lbi.ro/rmi\\_2019/](http://rmi.lbi.ro/rmi_2019/)
- [70] \*\*\* - <https://codeforces.com/>
- [71] \*\*\* - <https://cpbook.net/>
- [72] \*\*\* - <https://csacademy.com/>
- [73] \*\*\* - <https://gazeta.info.ro/revigoram-ginfo/>
- [74] \*\*\* - <https://oj.uz/problems/source/22>
- [75] \*\*\* - <https://profs.info.uaic.ro/~infogim/2019/index.html>
- [76] \*\*\* - <https://wandbox.org/>
- [77] \*\*\* - [https://en.cppreference.com/w/cpp/language/operator\\_alternative](https://en.cppreference.com/w/cpp/language/operator_alternative)
- [78] \*\*\* - <https://en.cppreference.com/w/cpp/algorithm>
- [79] \*\*\* - <https://www.ejoi2019.si/>
- [80] \*\*\* - <https://en.wikipedia.org/wiki/Algorithm>
- [81] \*\*\* - [https://en.wikipedia.org/wiki/List\\_of\\_algorithms](https://en.wikipedia.org/wiki/List_of_algorithms)
- [82] \*\*\* - [https://en.wikipedia.org/wiki/List\\_of\\_data\\_structures](https://en.wikipedia.org/wiki/List_of_data_structures)
- [83] \*\*\* - <https://cs.pub.ro/images/NewsLabsImages/Teste-admitere-informatica-UPB-2020.pdf>

# Lista autorilor

## problemelor și indicațiilor

Adrian Budău, 77

Alin Burța, 70

Bogdan-Petru Pop, 14

Carmen Mincu a, 244

Carmen Popescu, 197

Cheșcă Ciprian, 20

Constantin Gălățan, 118

Constantin Tudor, vi

Dan Pracsiu, 219

Daniela Marcu, 208

Doru Popescu Anastasiu, 251

Eugen Nodea, 134, 218

Florentina Ungureanu, 162

Gheorghe Dodescu, vi

Gheorghe Manolache, 12, 81, 180

Ion Văduva, vi

Marius Nicoli, 17

Mihai Stroe, 292, 297

Mircea Lupșe-Turpan, 91

Nicu Vlad-Laurențiu, 106

Nodea Eugen, 26

Ovidiu Marcu, 208

Radu Vișinescu, 184

Rodica Pîntea, 99

Simulescu Adriana, 147

Stefan-Cosmin Dăscălescu, 12

Stelian Ciurea, 73, 208

Szabo Zoltan, 173

Tamio-Vesa Nakajima, 36

ulbă-Lecu Theodor-Gabriel, 5

---

# WHAT'S THE NEXT?

---

ORNL'S FRONTIER FIRST TO BREAK THE EXAFLOP CEILING



[HTTPS://EN.WIKIPEDIA.ORG/WIKI/FRONTIER\\_\(SUPERCOMPUTER\)#/MEDIA/FILE:FRONTIER\\_SUPERCOMPUTER\\_\(2\).JPG](https://en.wikipedia.org/wiki/Frontier_(supercomputer)#/media/File:Frontier_Supercomputer_(2).jpg)

*Over time  
the following steps  
will lead you to the value  
you seek for yourself  
now!*