2024

Olimpiada Națională de Informatică pentru gimnaziu

Etapa județeană și etapa națională

Baraje de selecție a lotului național de juniori și a echipelor reprezentative ale României

Organizate în parteneriat de

SEPI
Societatea pentru Excelență și Performanță în Informatică

și Ministerul Educației și Cercetării

Infobits Academy

2024
Olimpiada Națională de Informatică pentru gimnaziu
Etapa județeană și etapa națională
Baraje de selecție a lotului național de juniori
și a echipelor reprezentative ale României

Enunțuri, soluții, surse

Tehnoredactare

Emanuela Cerchez, Cătălin Frâncu

Coperta

Emanuela Cerchez (pe baza unui design generat de Chat GPT)

ISBN

978-630-6559-23-7

Această lucrare este o **resursă educațională deschisă**, oferită gratuit tuturor celor care aspiră la excelență în informatică.

Societatea pentru Excelență și Performanță în Informatică
E-mail: contact@sepi.ro
www.sepi.ro

Editura L&S SOFT
Telefon: 0727.731.947
E-mail: hello@infobits.ro
www.infobits.ro
ebooks.infobits.ro
Compania noastră oferă de peste 30 de ani manuale școlare aprobate de Ministerul Educației și auxiliare ce respectă programa școlară, aplicații online, precum și cursuri de Informatică și T.I.C., utile oricărei persoane care dorește să se pregătească în aceste domenii.

Autori

Comisia centrală a Olimpiadei Naționale de Informatică 2024, secțiunea Gimnaziu

Etapa județeană și etapa națională

Clasa a V-a

- Prof. Raluca Costineanu, Colegiul Național „Ștefan cel Mare” Suceava - coordonator
- Prof. Marius Nicoli Marius, Colegiul Național „Frații Buzești” Craiova
- Prof. Alina Pintescu, Colegiul Național „Gheorghe Șincai” Baia Mare
- Prof. Ionel-Vasile Piț-Rada, Colegiul Național „Traian” Drobeta-Turnu Severin
- Prof. Dorin-Mircea Rotar, Colegiul Național „Samuil Vulcan” Beiuș
- Prof. Elena Rotaru, Colegiul Național Iași
- Prof. Marinel Șerban, Colegiul Național „Emil Racoviță” Iași
- Prof. Adrian Panaete, Colegiul Național „A. T. Laurian”, Botoșani

Clasa a VI-a

- Prof. Adrian Doru Pintea, Inspectoratul Școlar Județean Cluj, Cluj-Napoca - coordonator
- Prof. Dan Pracsiu, Liceul Teoretic „Emil Racoviță” Vaslui
- Prof. Alina Gabriela Boca, Colegiul Național de Informatică „Tudor Vianu” București
- Prof. Eugenia-Cristiana Iordaiche, Liceul Teoretic „Grigore Moisil” Timișoara
- Prof. Roxana Gabriela Tîmplaru, Colegiul „Ștefan Odobleja” Craiova
- Prof. Ana Maria Arișanu, Colegiul Național „Mircea cel Bătrân” Râmnicu Vâlcea
- Prof. Camelia Alice Georgescu, Colegiul Național „Mihai Viteazul” Ploiești
- Prof. Petru Simion Oprița, Liceul „Regina Maria” Dorohoi
- Stud. Giulian Buzatu, Facultatea de Matematică și Informatică, Universitatea București
- Stud. Iulian Oleniuc, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza” Iași

Clasa a VII-a

- Prof. Emanuela Cerchez, Colegiul Național „Emil Racoviță” Iași - coordonator
- Prof. Filonela Bălașa, Colegiul Național „Grigore Moisil” București
- Prof. Florentina Ungureanu, Colegiul Național de Informatică Piatra Neamț
- Prof. Claudiu-Cristian Gorea-Zamfir, Liceul de Informatică „Grigore Moisil” Iași
- Stud. Dumitru Ilie, Facultatea de Matematică-Informatică, Universitatea București
- Stud. Ioan-Cristian Pop, Universitatea Politehnica București
- Prof. Nistor Moț, Școala „Dr. Luca” Brăila
- Lector dr. Paul Diac, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza” Iași
- Stud. Theodor-Gabriel Tulba-Lecu, Universitatea Politehnica București

Clasa a VIII-a

- Prof. Flavius Boian, Colegiul Național „Spiru Haret” Târgu Jiu - coordonator
- Prof. Isabela Patricia Coman, Colegiul Național de Informatică „Tudor Vianu” București
- Prof. Lucia Miron, Colegiul Național „Costache Negruzzi” Iași
- Prof. Stelian Ciurea, Universitatea „Lucian Blaga” Sibiu
- Prof. Dan Octavian Dumitrașcu, Colegiul Național „Dinicu Golescu” Câmpulung
- Prof. Daniel Popa, Colegiul Național „Aurel Vlaicu” Orăștie

- Livia Măgureanu, Google România
- Stud. Bogdan Ioan Popa, Facultatea de Matematică și Informatică, Universitatea București
- Stud. Mircea Măierean, Facultatea de Matematică și Informatică, Universitatea Babeș-Bolyai Cluj Napoca

Barajul de selecție a lotului național de juniori

- Prof. Emanuela Cerchez, Colegiul Național „Emil Racoviță" Iași - coordonator
- Prof. Adrian Panaete, Colegiul Național „August Treboniu Laurian", Botoșani
- Prof. Ionel-Vasile Piț-Rada, Colegiul Național „Traian", Drobeta Turnu Severin
- Stud. Dumitru Ilie, Facultatea de Matematică-Informatică, Universitatea București
- Prof. Marinel Șerban, Colegiul Național „Emil Racoviță", Iași
- Prof. Flavius Boian, Colegiul Național „Spiru Haret", Târgu-Jiu
- Lector dr. Paul Diac, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza" Iași
- Prof. Claudiu-Cristian Gorea-Zamfir, Liceul de Informatică „Grigore Moisil" Iași
- Stud. Ioan-Cristian Pop, Universitatea Politehnica București
- Prof. Nistor Moț, Școala „Dr. Luca" Brăila
- Prof. Dan Pracsiu, Liceul Teoretic „Emil Racoviță", Vaslui
- Prof. Isabela Patricia Coman, Colegiul Național de informatică „Tudor Vianu", București
- Stud. Bogdan Ioan Popa, Facultatea de Matematică și Informatică, Universitatea București
- Stud. Livia Măgureanu, Facultatea de Limbi și Literaturi Străine, Universitatea București
- Stud. Giulian Buzatu, Facultatea de Matematică-Informatică, Universitatea București
- Ing. Theodor-Gabriel Tulbă-Lecu, Jane Street, Londra

Taberele de pregătire a lotului național de juniori și de selecție a echipelor reprezentative ale României pentru competițiile internaționale Cluj 10-15 mai 2024, Orăștie 24-29 mai 2024

- Prof. Emanuela Cerchez, Colegiul Național „Emil Racoviță" Iași - coordonator
- Prof. Adrian Panaete, Colegiul Național „August Treboniu Laurian" Botoșani
- Prof. Ciprian Chește, Liceul Tehnologic „Grigore C. Moisil" Buzău
- Prof. Ionel-Vasile Piț-Rada, Colegiul Național „Traian" Drobeta Turnu Severin
- Prof. Mihai Bunget, Colegiul Național „Tudor Vladimirescu" Târgu Jiu
- Lector dr. Paul Diac, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza" Iași
- Prof. Dan Pracsiu, Liceul Teoretic „Emil Racoviță" Vaslui
- Prof. Gheorghe Eugen Nodea, Centrul Județean de Excelență Gorj
- Prof. Daniela Elena Lica, Centrul Județean de Excelență Prahova
- Stud. Ioan-Cristian Pop, Universitatea Politehnica București
- Stud. Dumitru Ilie, Facultatea de Matematică-Informatică, Universitatea București
- Stud. Giulian Buzatu, Facultatea de Matematică-Informatică, Universitatea București
- Stud. Mircea Măierean, Facultatea de Matematică și Informatică, Universitatea „Babeș-Bolyai" Cluj-Napoca
- Stud. Bogdan-Ioan Popa, Facultatea de Matematică-Informatică, Universitatea București
- Stud. Andrei Onuț, Yale University, S.U.A.
- Instr. Cristian Frâncu, Nerdvana Education, București
- Prof. Marinel Șerban, Colegiul Național „Emil Racoviță" Iași
- Prof. Zoltan Szabo, Inspectoratul Școlar Județean Mureș

Cuprins

I Olimpiada Județeană de Informatică 2024 **5**

1. OJI 2024, clasa a V-a **7**
1.1. Problema Bomboane . . . 7
1.2. Rezolvarea problemei Bomboane . . . 8
1.3. Cod-sursă pentru problema Bomboane . . . 9
1.4. Problema Microbist . . . 11
1.5. Rezolvarea problemei Microbist . . . 12
1.6. Cod-sursă pentru problema Microbist . . . 13

2. OJI 2024, clasa a VI-a **15**
2.1. Problema Avid . . . 15
2.2. Rezolvarea problemei Avid . . . 16
2.3. Cod-sursă pentru problema Avid . . . 17
2.4. Problema Perechi . . . 20
2.5. Rezolvarea problemei Perechi . . . 21
2.6. Cod-sursă pentru problema Perechi . . . 22

3. OJI 2024, clasa a VII-a **25**
3.1. Problema Parking . . . 25
3.2. Rezolvarea problemei Parking . . . 28
3.3. Cod-sursă pentru problema Parking . . . 30
3.4. Problema Ron . . . 32
3.5. Rezolvarea problemei Ron . . . 33
3.6. Cod-sursă pentru problema Ron . . . 34

4. OJI 2024, clasa a VIII-a **37**
4.1. Problema MUN . . . 37
4.2. Rezolvarea problemei MUN . . . 39
4.3. Cod-sursă pentru problema MUN . . . 39
4.4. Problema Robotron . . . 42
4.5. Rezolvarea problemei Robotron . . . 44
4.6. Cod-sursă pentru problema Robotron . . . 45

II Olimpiada Națională de Informatică 2024 **47**

5. ONI 2024, clasa a V-a **49**
5.1. Problema P13 . . . 49
5.2. Rezolvarea problemei P13 . . . 50
5.3. Cod-sursă pentru problema P13 . . . 53
5.4. Problema Puzzle . . . 60
5.5. Rezolvarea problemei Puzzle . . . 61

5.6. Cod-sursă pentru problema Puzzle 62
5.7. Problema Robinhood 64
5.8. Rezolvarea problemei Robinhood 65
5.9. Cod-sursă pentru problema Robinhood 66

6. ONI 2024, clasa a VI-a **69**
6.1. Problema Codificare 69
6.2. Rezolvarea problemei Codificare 70
6.3. Cod-sursă pentru problema Codificare 71
6.4. Problema Esm 73
6.5. Rezolvarea problemei Esm 74
6.6. Cod-sursă pentru problema Esm 75
6.7. Problema Sim 77
6.8. Rezolvarea problemei Sim 79
6.9. Cod-sursă pentru problema Sim 81

7. ONI 2024, clasa a VII-a **83**
7.1. Problema Expresie 83
7.2. Rezolvarea problemei Expresie 84
7.3. Cod-sursă pentru problema Expresie 84
7.4. Problema Pictura 87
7.5. Rezolvarea problemei Pictura 89
7.6. Cod-sursă pentru problema Pictura 89
7.7. Problema Secv9 93
7.8. Rezolvarea problemei Secv9 94
7.9. Cod-sursă pentru problema Secv9 95

8. ONI 2024, clasa a VIII-a **97**
8.1. Problema Geologie 97
8.2. Rezolvarea problemei Geologie 98
8.3. Cod-sursă pentru problema Geologie 99
8.4. Problema Kmajo 101
8.5. Rezolvarea problemei Kmajo 102
8.6. Cod-sursă pentru problema Kmajo 102
8.7. Problema Mandms 104
8.8. Rezolvarea problemei Mandms 106
8.9. Cod-sursă pentru problema Mandms 107

9. Baraj selecție lot juniori ONI 2024 **109**
9.1. Problema Drei 109
9.2. Rezolvarea problemei Drei 111
9.3. Cod-sursă pentru problema Drei 114
9.4. Problema Soldați 117
9.5. Rezolvarea problemei Soldați 119
9.6. Cod-sursă pentru problema Soldați 120
9.7. Problema Tramvaie 121
9.8. Rezolvarea problemei Tramvaie 122
9.9. Cod-sursă pentru problema Tramvaie 123

III Tabăra de pregătire a lotului național de informatică juniori, Cluj, 10-15 mai 2024 **125**

10. Barajul 1 **127**
10.1. Problema Bug . . . 127
10.2. Rezolvarea problemei Bug . . . 128
10.3. Cod-sursă pentru problema Bug . . . 131
10.4. Problema Nooverlap . . . 138
10.5. Rezolvarea problemei Nooverlap . . . 139
10.6. Cod-sursă pentru problema Nooverlap . . . 140
10.7. Problema Pase . . . 143
10.8. Rezolvarea problemei Pase . . . 144
10.9. Cod-sursă pentru problema Pase . . . 146

11. Barajul 2 **151**
11.1. Problema Copaci . . . 151
11.2. Rezolvarea problemei Copaci . . . 152
11.3. Cod-sursă pentru problema Copaci . . . 153
11.4. Problema Pmo . . . 155
11.5. Rezolvarea problemei Pmo . . . 156
11.6. Cod-sursă pentru problema Pmo . . . 160
11.7. Problema Pokemoni . . . 168
11.8. Rezolvarea problemei Pokemoni . . . 169
11.9. Cod-sursă pentru problema Pokemoni . . . 171

IV Tabăra de pregătire a lotului național de informatică juniori, Orăștie, 24-29 mai 2024 **173**

12. Barajul 3 **175**
12.1. Problema Drumuri . . . 175
12.2. Rezolvarea problemei Drumuri . . . 176
12.3. Cod-sursă pentru problema Drumuri . . . 178
12.4. Problema Gimigpt . . . 180
12.5. Rezolvarea problemei Gimigpt . . . 182
12.6. Cod-sursă pentru problema Gimigpt . . . 182
12.7. Problema P2 . . . 187
12.8. Rezolvarea problemei P2 . . . 188
12.9. Cod-sursă pentru problema P2 . . . 188

13. Barajul 4 **191**
13.1. Problema Farfurii . . . 191
13.2. Rezolvarea problemei Farfurii . . . 192
13.3. Cod-sursă pentru problema Farfurii . . . 196
13.4. Problema Numbers . . . 201
13.5. Rezolvarea problemei Numbers . . . 202
13.6. Cod-sursă pentru problema Numbers . . . 202
13.7. Problema Nyse . . . 205
13.8. Rezolvarea problemei Nyse . . . 207
13.9. Cod-sursă pentru problema Nyse . . . 208

Partea I

Olimpiada Națională de Informatică
- etapa județeană -

17 martie 2024

Capitolul 1

OJI 2024, clasa a V-a

1.1 Problema Bomboane

Propusă de: prof. Marius Nicoli, Colegiul Național „Frații Buzești", Syncro Soft, Craiova

Copiii de la școala din oraș primesc daruri înaintea vacanței. Există o cutie foarte mare care conține N bomboane ce le pot fi distribuite tuturor copiilor prezenți la festivitatea care s-a organizat, astfel încât, întotdeauna, toți să primească același număr de bomboane, B.

Cerințe

1. Să se determine valoarea maximă a lui B, știind că la festivitate sunt prezenți exact X copii, iar după distribuire este posibil să fie lăsate unele bomboane în cutie.
2. Să se determine numărul maxim de copii care pot fi prezenți la festivitate, astfel încât să fie distribuite *toate* bomboanele din cutie, iar valoarea lui B să fie mai mare sau egală cu 2.
3. Să se determine numărul minim de bomboane care pot fi lăsate în cutie, după distribuire, astfel încât la festivitate să fie prezenți cel puțin X copii, iar valoarea lui B să fie mai mare sau egală cu Y. Corespunzător numărului de bomboane lăsate obținut și condițiilor precizate, se determină, de asemenea, numărul de copii prezenți precum și valoarea lui B. În cazul în care sunt mai multe variante ce respectă aceste condiții, se alege cea pentru care numărul de copii prezenți este maxim posibil.

Date de intrare

Fișierul de intrare `bomboane.in` conține pe prima linie 4 numere naturale C N X Y, în această ordine, separate prin câte un spațiu. Valoarea C reprezintă cerința de rezolvat, iar celelalte au semnificația din enunț. Observăm că pentru unele cerințe nu sunt necesare toate cele 4 valori, dar ele vor fi prezente în fișierul de intrare.

Date de ieșire

Fișierul de ieșire `bomboane.out` va conține rezultatele pe o singură linie, astfel: pentru primele două cerințe, se va scrie un număr natural reprezentând valoarea cerută; pentru a treia cerință, se vor scrie 3 numere separate prin câte un spațiu reprezentând, în această ordine: numărul minim cerut de bomboane care pot fi lăsate în cutie, numărul de copii prezenți și valoarea lui B, în condițiile precizate în cerință.

Restricții

- $1 \leq C \leq 3$
- $1 \leq N, X, Y \leq 2\,000\,000\,000$
- Se garantează că pentru toate datele de intrare există soluție.
- Se garantează că pentru toate datele de intrare corespunzătoare cerinței 3, soluția se obține lăsând în cutie cel mult 100 de bomboane.

#	Puncte	Restricții
1	19	$C = 1$
2	28	$C = 2$
3	53	$C = 3$

Exemple

bomboane.in	bomboane.out	Explicații
1 51 5 8	10	Rezolvăm cerința $C = 1$, pentru $N = 51$ și $X = 5$ (valoarea $Y = 8$ nu este necesară rezolvării cerinței). Se pot distribui câte $B = 10$ bomboane fiecăruia dintre cei $X = 5$ copii. În cutie este lăsată o bomboană.
2 51 5 8	17	Rezolvăm cerința $C = 2$, pentru $N = 51$ (valorile $X = 5$ și $Y = 8$ nu sunt necesare rezolvării cerinței). Pot fi prezenți maximum 17 copii și fiecare primește câte $B = 3$ bomboane.
3 51 5 8	1 5 10	Rezolvăm cerința $C = 3$, pentru $N = 51$, $X = 5$ și $Y = 8$. Numărul minim de bomboane lăsate în cutie pentru a satisface cerința este 1. În aceste condiții numărul maxim de copii care primesc bomboane este 5 și fiecare primește câte $B = 10$ bomboane.

1.2 Rezolvarea problemei Bomboane

Mai jos, notațiile N, X, Y au semnificația din enunț.

Cerința 1

Valoarea cerută se obține ca rezultat al expresiei N/X.

Cerința 2

Numărul de copii și numărul de bomboane trebuie să fie divizori ai lui N și totodată produsul lor trebuie să fie egal cu N. Divizorii cu această proprietate se obțin căutând un divizor d, iar perechea lui va fi atunci N/d. Este suficient să găsim pe d ca fiind cel mai mic divizor propriu al lui N și soluția va fi N/d.

Cerința 3

Datele de intrare ne permit să căutăm soluție considerând pe rând cazurile: nu lăsăm în cutie bomboane, lăsăm o singură bomboană, lăsăm două bomboane, etc. La prima astfel de valoare pentru care putem determina o distribuire, ne oprim. Odată fixat numărul b de bomboane lăsate în cutie (parcurgând valorile de la 0 la 100 cu o instrucțiune repetitivă), rămâne ca toate celelalte bomboane $n = N - b$ să fie distribuite. Avem și în acest caz de determinat o pereche de divizori ai lui n al căror produs este chiar n. Aceste perechi sunt deci de forma $(d, n/d)$. Este suficient să căutăm astfel de perechi cât timp $d \leq n/d$. Pentru fiecare pereche determinată $(d, n/d)$ analizăm dacă putem avea d = numărul de copii și n/d numărul de bomboane și se respectă condițiile $d \geq X$ și $n/d \geq Y$ sau respectiv d = numărul de bomboane și n/d numărul de copii și se respectă condițiile $d \geq Y$ și $n/d \geq X$.

Cerința 3 poate fi abordată și în modul descris în continuare. Notăm cu x numărul de copii căutat, cu y numărul de bomboane căutat și cu dif numărul de bomboane care vor rămâne în cutie. O soluție simplă este aceea de a parcurge cu x toate valorile $1, 2, \ldots N$ și la fiecare pas calculăm $y = N/x$, $dif = N - x \cdot y$ și păstrăm tripletul (dif,x,y) cu diferența dif minimă, iar la aceeași diferență minimă x să fie maxim. Când păstrăm un triplet avem grijă să fie respectate și condițiile $x >= X$ și $y \geq Y$. Din ideea anterioară se obține o soluție mai rapidă analizând, pe rând, două cazuri: Cazul $x \leq y$ când parcurgem valorile $x = 1, 2, \ldots$ cât timp $x \leq N/x$, apoi Cazul $x \geq y$ când parcurgem valorile $y = 1, 2, \ldots$ cât timp $y \leq N/y$

1.3 Cod-sursă pentru problema Bomboane

```
#include <fstream>
using namespace std;

int n, i, j, rest, solc, solb, m, gasit = 0, c, x, y;

int main () {
    ifstream fin ("bomboane.in");
    ofstream fout("bomboane.out");

    fin>>c>>n>>x>>y;

    if (c == 1) {
        fout<<n/x<<"\n";
        return 0;
    }

    if (c == 2) {
        for (i=2;;i++)
            if (n%i == 0) {
                fout<<n/i<<"\n";
                return 0;
            }
    }

    for (rest = 0; rest <= 100; rest++) {
        m = n-rest;
        for (i=min(x, y); i<=m/i; i++) {
            if (m%i == 0) {
                j = m/i;
                if (i >= x && j >= y) {
                    gasit = 1;
                    if (i >= solc) {
                        solc = i;
```

```
                        solb = j;
                    }
                }
                if (j >= x && i >= y) {
                    gasit = 1;
                    if (j >= solc) {
                        solc = j;
                        solb = i;
                    }
                }
            }
        }
        if (gasit == 1)
            break;
    }
    fout<<rest<<" "<<solc<<" "<<solb<<"\n";
    return 0;
}
```

1.4 Problema Microbist

Propusă de: prof. Dorin-Mircea Rotar, Colegiul Național „Samuil Vulcan" Beiuș

Gigel descoperă că este mare microbist și un fan adevărat al echipei Juventus Torino. El a urmărit evoluția unui meci disputat între Juventus Torino și AC Milan și a notat pe o foaie cele **N** goluri în ordinea în care ele au fost marcate. La fiecare gol marcat de Juventus a scris pe foaie cifra 1 și la fiecare gol marcat de Milan a scris pe foaie cifra 2.

Scorul meciului, la un moment dat, se exprimă prin două numere, primul reprezentând numărul total de goluri marcate până la acel moment de prima echipă, Juventus Torino, iar al doilea reprezentând numărul total de goluri marcate până la acel moment de a doua echipă, AC Milan. **Scorul este egal** dacă cele două numere sunt egale, iar o echipă **conduce** echipa adversă în joc dacă numărul de goluri marcate de ea este strict mai mare decât cele marcate de echipa adversă. **Scorul final** este cel obținut la încheierea jocului.

Spunem că **revenirea în forță** este o situație în care o echipă, *care este condusă*, înscrie un număr corespunzător de goluri *până când preia conducerea*, fără ca echipa adversă să fi marcat vreun gol în tot acest timp.

Cerințe

1. Determinați scorul final.
2. Determinați numărul de scoruri egale care au fost înregistrate pe parcursul jocului, începând cu cel de pornire. Scorul de pornire, $0 - 0$, este considerat egal.
3. Determinați numărul de goluri corespunzător celei mai mari reveniri în forță din joc (numărul maxim de goluri succesive marcate de o echipă la o revenire în forță).

Date de intrare

Fișierul de intrare `microbist.in` conține pe prima linie două numere naturale C și N separate printr-un spațiu. C reprezintă cerința ce trebuie rezolvată, iar N numărul de goluri marcate. Pe următoarea linie se găsesc, separate prin câte un spațiu, N valori, reprezentând golurile marcate, conform descrierii de mai sus.

Date de ieșire

Prima linie a fișierului de ieșire `microbist.out` va conține:

- pentru C = 1 două numere naturale separate de un spațiu, reprezentând, în această ordine, numărul de goluri marcate de Juventus Torino respectiv numărul de goluri marcate de AC Milan;
- pentru C = 2 numărul cerut de scoruri egale;
- pentru C = 3 numărul maxim de goluri cerut.

Restricții

- $1 \leq C \leq 3$
- $1 \leq N \leq 100\,000$
- Se garantează că pentru toate datele de test corespunzătoare cerinței 3, există cel puțin o revenire în forță

#	Puncte	Restricții
1	21	$C = 1$
2	24	$C = 2$
3	55	$C = 3$

Exemple

microbist.in	microbist.out	Explicații
1 5 1 2 1 2 2	2 3	Se rezolvă cerința $C = 1$ și s-au înregistrat $N = 5$ goluri. Scorul final este $2 - 3$.
2 5 1 1 2 2 2	2	Se rezolvă cerința $C = 2$ și s-au înregistrat $N = 5$ goluri. Scorurile înregistrate pe parcursul jocului au fost, în această ordine: $0 - 0$, $1 - 0$, $2 - 0$, $2 - 1$, $2 - 2$, $2 - 3$. Dintre acestea două au fost egale: $0 - 0$ și $2 - 2$.
3 6 1 1 2 2 2 2	3	Se rezolvă cerința $C = 3$ și s-au înregistrat $N = 6$ goluri. Cea mai mare revenire în forță s-a produs atunci când AC Milan era condusă cu $2 - 0$ și apoi a marcat succesiv trei goluri, preluând în acel moment conducerea, scorul devenind $2 - 3$.
3 11 1 2 2 2 1 1 1 1 2 1 1	3	Se rezolvă cerința $C = 3$ și s-au înregistrat $N = 11$ goluri. Cea mai mare revenire în forță a fost atunci când Juventus era condusă cu $1 - 3$ și apoi revine în forță marcând succesiv trei goluri și preluând conducerea cu $4 - 3$.

1.5 Rezolvarea problemei Microbist

Cerința 1

Pentru a afla scorul final, vom parcurge șirul de goluri și vom număra câte goluri a marcat fiecare echipă. Pentru aceasta putem folosi două variabile contor pe care să le actualizăm în momentul citirii în funcție de echipa care marchează.

Cerința 2

Pentru a determina câte scoruri au fost egale, vom ține evidența numărului de goluri marcate de fiecare echipă, la fel ca mai sus, și vom incrementa un contor de fiecare dată când scorurile devin egale.

Cerința 3

Putem determina secvențe de elemente egale aflate pe poziții consecutive. O astfel de secvență este revenire în forță dacă atunci când ea începe, echipa pe care o reprezintă este condusă și atunci când se termină echipa pe care o reprezintă conduce cu un gol.

Detectarea și calcularea revenirii în forță pentru cerința 3:

După primul gol, se actualizează scorul echipei corespunzătoare și se determină echipa care este condusă. Pentru următoarele goluri marcate vom ține cont de următoarele aspecte:

- dacă golul curent este diferit de cel anterior (marchează altă echipă) atunci testăm dacă putem începe o nouă revenire în forță și în caz afirmativ stabilim contorul de revenire la unu.
- dacă golul curent este marcat de aceeași echipă ca și golul anterior atunci creștem contorul pentru revenirea în forță și dacă s-a schimbat echipa câștigătoare atunci putem actualiza maximul pentru revenirea în forță și resetăm contorul pentru revenire.

1.6 Cod-sursă pentru problema Microbist

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    ifstream cin("microbist.in");
    ofstream cout("microbist.out");
    int C, N, gol,gol_ant;
    cin >> C >> N;
    int scorJuventus = 0, scorMilano = 0;
    int scoruriEgale = 1, maxRevenire = 0, revenireCurenta = 0;
    int echipaInferioara = 0; // 0 - nicio echipa, 1 - Juventus, 2 - Milano
    cin >> gol_ant;
    if (gol_ant == 1){
            ++scorJuventus;
            echipaInferioara = 2;
    }
    else{
            ++scorMilano;
            echipaInferioara = 1;
    }

    for (int i = 2; i <= N; ++i) {
        cin >> gol;
        // Cerinta 1: Scor
        if (gol == 1) ++scorJuventus; else ++scorMilano;

        // Cerinta 2: Scoruri egale
        if (scorJuventus == scorMilano){
                ++scoruriEgale;
                revenireCurenta++;
        }
        // Cerinta 3: Revenire in forta
        else if(gol != gol_ant){
            if(scorJuventus>scorMilano && gol == 2){
                echipaInferioara = 2;
                revenireCurenta = 1;
            }
            else if(scorMilano>scorJuventus && gol==1){
                echipaInferioara = 1;
                revenireCurenta = 1;
            }
        }
        else{
            revenireCurenta++;
            if(scorJuventus>scorMilano && echipaInferioara == 1){
```

```
                maxRevenire=max(maxRevenire,revenireCurenta);
                echipaInferioara = 2;
                revenireCurenta = 0;
            }
            if(scorJuventus<scorMilano && echipaInferioara == 2){
                maxRevenire=max(maxRevenire,revenireCurenta);
                echipaInferioara = 1;
                revenireCurenta = 0;
            }
        }
        gol_ant = gol;
    }

    if (C == 1) {
        cout << scorJuventus << " " << scorMilano << "\n";
    } else if (C == 2) {
        cout << scoruriEgale << "\n";
    } else if (C == 3) {
        cout << maxRevenire << "\n";
    }
    return 0;
}
```

Capitolul 2

OJI 2024, clasa a VI-a

2.1 Problema Avid

Propusă de: stud. Giulian Buzatu, Facultatea de Matematică și Informatică, Universitatea București

Alex este un băiat căruia îi place să citească și care contorizează cât de mult a citit pe parcursul ultimelor n zile. Mai precis, el și-a notat câte pagini a citit în fiecare dintre acestea. Chiar dacă pasiunea lui este literatura, își dorește să progreseze și la informatică. Alex și-a pus două întrebări legate de șirul format din numărul de pagini citite de el în ultimele n zile, dar după ce a petrecut câteva zile gândindu-se la ele și-a dat seama că sunt prea dificile pentru el. Ajutați-l să găsească răspunsurile!

Cerințe

Fie numărul n, numărul p și acel șir de valori notate de Alex în cele n zile. Determinați răspunsul la următoarele întrebări care îl frământă pe Alex:

1. Câte triplete de numere aflate pe poziții consecutive în șirul dat îndeplinesc condiția ca cel mai mare divizor comun al lor să aibă cel mult p divizori naturali?
2. Care este lungimea maximă a unei secvențe din șirul dat, în care cel mai mare divizor comun al oricărui triplet de numere situate pe poziții consecutive are cel mult p divizori naturali?

Date de intrare

Fișierul `avid.in` conține pe prima linie un număr natural C, având valoarea 1 sau 2, reprezentând numărul întrebării. Pe a doua linie se află două numere naturale n și p, în această ordine, cu semnificația din enunț. A treia linie din fișier conține n numere naturale reprezentând șirul de valori notate în cele n zile. Numerele aflate pe aceeași linie a fișierului sunt separate prin câte un spațiu.

Date de ieșire

Fișierul `avid.out` va conține un singur număr, reprezentând răspunsul pentru întrebarea dată, C.

Restricții

- $1 \leq C \leq 2$

- $3 \leq n \leq 1\,000\,000$
- $2 \leq p \leq 100$
- $1 \leq a_i \leq 5\,000\,000$, unde a_i este numărul de pagini citite de Alex în ziua i (Alex citește la o viteză impresionantă)
- Pentru prima cerință, se garantează că există cel puțin un triplet cu proprietatea indicată.
- Pentru a doua cerință, se garantează că există cel puțin o secvență validă cu proprietatea indicată.

#	Puncte	Restricții
1	12	$C = 1$, $n \leq 1\,000$
2	17	$C = 1$, $1\,000 < n \leq 1\,000\,000$
3	29	$C = 2$, $n \leq 1\,000$
4	42	$C = 2$, $1\,000 < n \leq 1\,000\,000$

Exemple

avid.in	avid.out	Explicații
1 10 3 12 48 36 6 3 7 12 16 24 3	6	cmmdc$(12, 48, 36) = 12$, care are 6 divizori naturali. cmmdc$(48, 36, 6) = 6$, care are 4 divizori naturali. cmmdc$(36, 6, 3) = 3$, care are 2 divizori naturali. cmmdc$(6, 3, 7) = 1$, care are 1 divizor natural. cmmdc$(3, 7, 12) = 1$, care are 1 divizor natural. cmmdc$(7, 12, 16) = 1$, care are 1 divizor natural. cmmdc$(12, 16, 24) = 4$, care are 3 divizori naturali. cmmdc$(16, 24, 3) = 1$, care are 1 divizor natural. Deci, 6 dintre cele 8 triplete au cel mult $p = 3$ divizori naturali.
2 7 2 12 48 36 6 3 7 12	5	Pentru că $p = 2$, cea mai lungă secvență este $36, 6, 3, 7, 12$.

2.2 Rezolvarea problemei Avid

Cerința 1

Pentru 12 puncte, se citesc numerele din fișierul de intrare într-un vector și numărăm câte triplete respectă condiția din enunț. Vom calcula cel mai mare divizor comun al fiecărui triplet de valori de pe poziții consecutive, după care vom calcula numărul de divizori al acestuia, iar dacă este mai mic sau egal cu p, vom incrementa răspunsul. Vom folosi algoritmul lui Euclid prin împărțiri repetate și descompunerea în factori primi până la radical. Complexitatea este $\mathcal{O}(n(\log \max\{a_i\} + \sqrt{\max\{a_i\}})) = \mathcal{O}(n\sqrt{\max\{a_i\}})$.

Pentru restul de 17 puncte, vom folosi același procedeu pentru a calcula numărul de triplete, dar vom calcula mai eficient numărul de divizori. Vom precalcula numerele prime până la $\sqrt{5\,000\,000} \approx 2\,236$, folosind ciurul lui Eratostene. Astfel, când vom face descompunerea în factori primi până la radicalul numărului nostru, vom folosi doar numerele prime. Complexitatea este $\mathcal{O}(n\sqrt{\max\{a_i\}} + \sqrt{\max\{a_i\}} \log\log \sqrt{\max\{a_i\}})$. Observăm că, deși complexitatea este similară, algoritmul este mult mai rapid în practică.

Cerința 2

Pentru 29 de puncte, se citesc numerele din fișierul de intrare într-un vector. Parcurgem vectorul și verificăm dacă tripletul care se termină pe poziția curentă respectă condiția din enunț, folosind algoritmul lui Euclid prin împărțiri repetate și descompunerea în factori primi până la radical. În caz afirmativ, incrementăm lungimea secvenței curente. Altfel, se încearcă actualizarea maximului, doar dacă lungimea curentă este cel puțin 3, și după se resetează lungimea secvenței curente la valoarea 2. La finalul parcurgerii se va mai încerca o dată actualizarea maximului, cu aceeași condiție ca mai sus, deoarece este posibil ca secvența curentă să fie maximală. Complexitatea este $\mathcal{O}(n(\log \max\{a_i\} + \sqrt{\max\{a_i\}})) = \mathcal{O}(n\sqrt{\max\{a_i\}})$.

Pentru restul de 42 de puncte, se procedează similar, dar vom calcula mai eficient numărul de divizori, folosind modul descris în cel de-al doilea paragraf de la descrierea cerinței 1. Complexitatea este $\mathcal{O}(n\sqrt{\max\{a_i\}} + \sqrt{\max\{a_i\}} \log\log \sqrt{\max\{a_i\}})$. Observăm că, deși complexitatea este similară, algoritmul este mult mai rapid în practică.

Observație:

Pentru ambele cerințe putem să rezolvăm problema fără a reține datele citite într-un vector. Totuși, pentru simplitate și pentru că limita de memorie este generoasă, nu este nevoie să facem acest lucru.

Observație:

Descompunerea în factori primi a lui n (mergând până la $\sqrt{n}$) este folosită pentru determinarea numărului său de divizori astfel: fie $n = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}$, atunci numărul n are $(e_1 + 1)(e_2 + 1) \cdots (e_k + 1)$ divizori, conform regulii produsului.

Observație:

Pentru soluția fără ciur, numărul de divizori al lui n poate fi calculat în $\mathcal{O}(\sqrt{n})$ și fără formula precedentă. Este suficient să parcurgem divizorii lui n cuprinși între 1 și $\sqrt{n}$. De fiecare dată când găsim un divizor d, deducem că și n/d este un divizor al lui n, așa că adunăm 2 la rezultat. Singura excepție constă în cazul în care $d = n/d$, când se adună doar 1.

2.3 Cod-sursă pentru problema Avid

```
#include <fstream>
using namespace std;
ifstream f("avid.in");
ofstream g("avid.out");
const int NMAX=1000000,CIURMAX=2300;
int v[NMAX+5],not_prime[CIURMAX+5],primes[350];

int main()
```

```
{   int C, N, p, no_primes=0, cmmdc_curent;
    // Precalculam numerele prime pana la CIURMAX
    not_prime[0]=not_prime[1]=1;
    for(int i=4; i<=CIURMAX; i+=2)
        not_prime[i]=1;
    for(int i=3; i*i<=CIURMAX; i+=2)
        if(!not_prime[i])
            for(int j=i*i; j<=CIURMAX; j+=i)
                not_prime[j]=1;
    // Adaugam numerele prime intr-un vector separat, pentru a le folosi la
    // calcularea numarului de divizori
    primes[no_primes++]=2;
    for(int i=3; i<CIURMAX; i+=2)
        if(!not_prime[i])
            primes[no_primes++]=i;
    // Citirea datelor de intrare
    f>>C>>N>>p;
    for(int i=1; i<=N; i++)
        f>>v[i];
    // Rezolvarea cerintelor
    if(C==1)
    {
        // Trebuie sa calculam cate triplete respecta conditia din enunt
        int cnt=0;
        for(int i=3; i<=N; i++)
        {
            // cmmdc(a,b,c)=cmmdc(cmmdc(a,b),c)
            // Algoritmul lui Euclid cu impartiri - cmmdc(v[i-2],v[i-1])
            int a=v[i-2],b=v[i-1],r;
            while(b)
            {
                r=a%b;
                a=b;
                b=r;
            }
            // Algoritmul lui Euclid cu impartiri - cmmdc(cmmdc(v[i-2], v[i-1]), v[i])
            b=v[i];
            while(b)
            {
                r=a%b;
                a=b;
                b=r;
            }
            cmmdc_curent=a;
            // Calculam numarul de divizori in O(sqrt(cmmdc_curent))
            int d=0,power=0,ans=1,copie=cmmdc_curent;
            while(d<no_primes && primes[d]*primes[d]<=copie)
            {
                while(copie%primes[d]==0)
                    copie/=primes[d],power++;
                if(power)
                    ans*=(power+1),power=0;
                d++;
            }
            if(copie>1)
                ans*=2;
            if(ans<=p) // Verificam daca numarul de divizori e cel mult p
                cnt++;
        }
        g<<cnt;
    }
```

```
    else
    {
        // Trebuie sa gasim lungimea celei mai mari secvente care respecta conditia din enunt
        int lung_curenta=2, maxx=0; // lungimea curenta este initializata cu 2,
        // deoarece daca putem lua tripletul (v[i-2],v[i-1],v[i]), atunci
        // lungimea secventei va fi 3
        for(int i=3; i<=N; i++)
        {
            // cmmdc(a,b,c)=cmmdc(cmmdc(a,b),c)
            // Algoritmul lui Euclid cu impartiri - cmmdc(v[i-2],v[i-1])
            int a=v[i-2],b=v[i-1],r;
            while(b)
            {
                r=a%b;
                a=b;
                b=r;
            }
            // Algoritmul lui Euclid cu impartiri - cmmdc(cmmdc(v[i-2], v[i-1]), v[i])
            b=v[i];
            while(b)
            {
                r=a%b;
                a=b;
                b=r;
            }
            cmmdc_curent=a;
            // Calculam numarul de divizori in O(sqrt(cmmdc_curent))
            int d=0,power=0,ans=1,copie=cmmdc_curent;
            while(d<no_primes && primes[d]*primes[d]<=copie)
            {
                while(copie%primes[d]==0)
                    copie/=primes[d],power++;
                if(power)
                    ans*=(power+1),power=0;
                d++;
            }
            if(copie>1)
                ans*=2;
            if(ans<=p) // Verificam daca numarul de divizori e cel mult p
                lung_curenta++;
            else
            {
                // Daca lungimea curenta nu este cel putin 3, inseamna ca nu
                // avem nici macar un triplet in secventa actuala
                // Deci nu vom actualiza maximul
                if(lung_curenta>=3 && lung_curenta>maxx)
                    maxx=lung_curenta;
                lung_curenta=2;
                // Reinitializam lungimea curenta cu 2, din aceleasi motive
                // pentru care am initializat-o initial cu 2
            }
        }
        // Daca secventa se termina pe pozitia N, ea nu ar fi fost luata in
        // considerare fara acest if
        if(lung_curenta>=3 && lung_curenta>maxx)
            maxx=lung_curenta;
        g<<maxx;
    }
    return 0;
}
```

2.4 Problema Perechi

Propusă de: prof. Alina-Gabriela Boca, Colegiul Național de Informatică Tudor Vianu București

Oglinditul unui număr natural x este numărul obținut prin parcurgerea cifrelor lui x de la dreapta la stânga, ignorându-se cifrele nule de pe ultimele poziții ale lui x. De exemplu, oglinditul lui 103 este 301, în timp ce oglinditul lui 2500 este 52. O pereche de numere naturale distincte x și y se numește pereche *oglindită* dacă atât x este oglinditul lui y, cât și y este oglinditul lui x. De exemplu, numerele $x = 42$ și $y = 24$ formează o pereche oglindită, însă numerele $x = 1$ și $y = 100$ nu formează o pereche oglindită.

Un număr natural x este considerat *palindrom* dacă x este egal cu oglinditul său. De exemplu, numărul 42124 este palindrom. Din două numere distincte se poate forma un număr nou prin alipirea unuia la dreapta celuilalt. De exemplu, din numerele 124 și 42 se pot obține numerele 12442 (din alipirea lui 42 la dreapta lui 124) și 42124 (din alipirea lui 124 la dreapta lui 42).

Cerințe

Fie un șir de numere naturale $a_1, a_2, \ldots, a_n$. Determinați:

1. Numărul perechilor de indici (i, j), cu $1 \leq i < j \leq n$, având proprietatea că a_i și a_j formează o pereche oglindită.
2. Cel mai mare număr palindrom care se poate forma prin alipirea a două numere distincte din șir.

Date de intrare

Fișierul `perechi.in` conține pe prima linie un număr natural C, având valoarea 1 sau 2, reprezentând numărul cerinței. Pe a doua linie se află numărul natural n. A treia linie din fișier conține șirul de numere naturale $a_1, a_2, \ldots, a_n$, separate prin câte un spațiu.

Date de ieșire

Fișierul `perechi.out` va conține un singur număr, reprezentând rezultatul corespunzător pentru cerința dată.

Restricții

- $1 \leq C \leq 2$
- $1 \leq n \leq 100\,000$
- $1 \leq a_i < 10\,000$
- Se garantează că pentru cerința 1 există în șirul dat cel puțin o pereche oglindită, iar la cerința 2 există în șirul dat cel puțin un număr palindrom.

#	Puncte	Restricții
1	27	$C = 1$, $n \leq 10\,000$
2	23	$C = 1$, $10\,000 < n \leq 100\,000$
3	27	$C = 2$, $n \leq 10\,000$
4	23	$C = 2$, $10\,000 < n \leq 100\,000$

Exemple

perechi.in	perechi.out	Explicații
1 5 21 12 21 12 21	6	Există 6 perechi de indici cu proprietatea că valorile corespunzătoare lor formează perechi oglindite: $(1, 2)$, $(1, 4)$, $(2, 3)$, $(2, 5)$, $(3, 4)$ și $(4, 5)$. Fiecare dintre aceste perechi oglindite este compusă din valorile 12 și 21.
1 6 13 97 76 67 76 31	3	Există 3 perechi de indici cu proprietatea că valorile corespunzătoare lor formează perechi oglindite: $(1, 6)$, $(3, 4)$ și $(4, 5)$. Aceste perechi oglindite formate sunt: $(13, 31)$, $(76, 67)$, respectiv, $(67, 76)$.
2 6 24 79 42 97 123 124	42124	Se pot forma următoarele numere palindrom: 2442, 4224, 7997, 9779 și 42124. Cel mai mare dintre acestea este 42124.

2.5 Rezolvarea problemei Perechi

Cerința 1

Pentru 27 de puncte, se citesc numerele din fișierul de intrare într-un vector, se parcurg elementele vectorului, iar pentru fiecare element aflat pe poziția i, cu $1 \leq i \leq n-1$,2se verifică toate elementele aflate pe pozițiile j, cu $i+1 \leq j \leq n$, numărându-se toate elementele care sunt egale cu oglinditul lui a_i și formează o pereche oglindită cu a_i. Complexitatea algoritmului este $\mathcal{O}(n^2)$.

Pentru 50 de puncte, se construiește un vector de frecvență pe baza numerelor citite. Se parcurge vectorul de frecvență și, pentru fiecare valoare x care există în acesta, se calculează oglinditul lui x în variabila y. Dacă y există în vectorul de frecvență, este strict mai mare decât x și formează o pereche oglindită cu x, atunci se va aduna la numărul de perechi produsul dintre frecvența lui x și frecvența lui y. Din moment ce n poate fi 100 000, rezultatul poate depăși valoarea maximă reprezentabilă pe tipul de date `int`. Complexitatea algoritmului este $\mathcal{O}(n + \max)$, unde max este numărul maxim din șir.

Cerința 2

Pentru 27 de puncte, se citesc numerele din fișierul de intrare într-un vector. Se parcurg elementele vectorului, iar fiecare element aflat pe poziția i, cu $1 \leq i \leq n-1$, se compune cu toate elementele aflate pe pozițiile j, cu $i+1 \leq j \leq n$, verificându-se dacă cele două valori rezultate din compunere sunt numere palindrom. Pe parcurs, se reține valoarea maximă obținută. Complexitatea algoritmului este $\mathcal{O}(n^2)$.

Pentru 50 de puncte, se construiește un vector de frecvență pe baza numerelor citite. Se parcurge vectorul de frecvență și, pentru fiecare valoare x care există în acesta, se calculează oglinditul lui x în variabila y.

Dacă y există în vectorul de frecvență și este diferit de x, atunci se compun cele două numere, rezultând două numere palindrom.

Din numărul y se elimină pe rând câte o cifră. Se verifică dacă numărul rezultat există în vectorul de frecvență și, în caz afirmativ, se alipește la începutul numărului x inițial.

Din numărul x se elimină pe rând câte o cifră. Se verifică dacă numărul rezultat există în vectorul de frecvență și, în caz afirmativ, se alipește ul numărului y inițial.

Pe parcurs, dintre toate numerele palindrom obținute în acest proces, se reține maximul.

Complexitatea algoritmului este $\mathcal{O}(n + \max)$, unde max este numărul maxim din șir.

2.6 Cod-sursă pentru problema Perechi

```
#include <fstream>
using namespace std;
ifstream fin("perechi.in");
ofstream fout("perechi.out");
int f[10001];

int main()
{int n, i, c, x, y, px, py, j, a, b, cy, cx;
 int maxim=0, nrx, nry, cnr, ogl_nr, ogl_a, ogl_b, ca, cb, mx=0, z;
 long long int nr;
 fin>>c>>n;
 if (c==1)
    {nr=0;
     for (i=1; i<=n; i++)
         { fin>>x;
           f[x]++;
           if (mx<x) mx=x;
         }
     for (i=1; i<=mx; i++)
         {
          if (f[i]>0)
             { x=i;
               // oglinditul si numarul de cifre ale lui x
               y=0; nrx=0;
               while (x>0)
                     {
                      y=y*10+x%10;
                      x=x/10;
                      nrx++;
                     }
               if (f[y]>0 && i<y)
                  { cy=y; nry=0;
                    //numarul de cifre ale lui y
                    while (y>0)
                          {
                           nry++;
                           y=y/10;
                          }
                    if (nrx==nry)
                        nr=nr+(long long int)f[i]*f[cy];
                  }
             }
         }
     fout<<nr;
    }
    else
    {nr=0;
     for (i=1; i<=n; i++)
         {
          fin>>x;
          f[x]=1;
```

```
      if (mx<x) mx=x;
      }
  for (i=1; i<=mx; i++)
     {
      if (f[i]==1)
         { x=i; cx=x; y=0;
           while (cx>0)
                 {
                  y=y*10+cx%10;
                  cx=cx/10;
                 }
           px=1; cx=x;
           while (cx>0)
                 {
                  px=px*10;
                  cx/=10;
                 }
           if (y!=x && f[y]==1)
              { cy=y; py=1;
                while (cy>0)
                      { py=py*10;
                        cy/=10;
                      }
                nr=x*py+y; cnr=nr; ogl_nr=0;
                while (cnr>0)
                      {
                       ogl_nr=ogl_nr*10+cnr%10;
                       cnr=cnr/10;
                      }
                if (nr==ogl_nr && nr>maxim) maxim=nr;
                nr=y*px+x; cnr=nr; ogl_nr=0;
                while (cnr>0)
                      {
                       ogl_nr=ogl_nr*10+cnr%10;
                       cnr=cnr/10;
                      }
                if (nr==ogl_nr && nr>maxim) maxim=nr;
           }
           x=i; cx=x; y=0;
           while (cx>9)
                 {
                  y=y*10+cx%10;
                  cx=cx/10;
                  if (f[y]==1 && y!=x)
                     {
                      a=y*px+x; ogl_a=0; ca=a;
                      while (ca>0)
                            {
                             ogl_a=ogl_a*10+ca%10;
                             ca=ca/10;
                            }
                      if (a==ogl_a && maxim<a) maxim=a;
                     }
                 }
           z=0; cx=x;
           while (cx>0)
                 {
                  z=z*10+cx%10;
                  cx=cx/10;
                 }
           int p=1; y=0;
```

```
                while (z>9)
                      {
                       y=y+p*(z%10);
                       p=p*10;
                       z=z/10;
                       if (f[y]==1)
                          {
                           a=x*p+y; ogl_a=0; ca=a;
                           while (ca>0)
                                 {
                                  ogl_a=ogl_a*10+ca%10;
                                  ca=ca/10;
                                 }
                           if (a==ogl_a && maxim<a) maxim=a;
                          }
                       }
            }
        }
      fout<<maxim;
    }
    return 0;
}
```

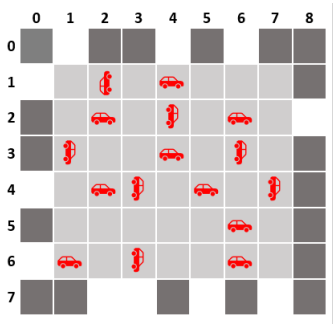
Capitolul 3

OJI 2024, clasa a VII-a

3.1 Problema Parking

Propusă de: prof. Florentina Ungureanu, Colegiul Național de Informatică Piatra Neamț

Mark a construit o parcare dreptunghiulară, pe care a împărțit-o, utilizând marcaje, în locuri de parcare pătrate, organizate pe n linii (numerotate de la 1 la n) și m coloane (numerotate de la 1 la m). Astfel, un loc de parcare poate fi identificat prin numărul liniei și numărul coloanei pe care acesta se află. Orice mașină poate fi parcată în interiorul unui loc de parcare, paralel cu liniile orizontale de marcaj, sau paralel cu liniile verticale, fără a depăși conturul pătratului corespunzător. Mark a construit un zid de împrejmuire a parcării, întrerupt în dreptul unor locuri de parcare, pentru a permite ieșirea mașinilor din parcare. Considerăm că zidul este plasat pe linia 0 (nord), coloana 0 (vest), linia $n+1$ (sud) și coloana $m+1$ (est).



O mașină parcată paralel cu marcajele orizontale se poate deplasa pe linie, spre stânga sau spre dreapta, putând ieși din parcare dacă ajunge la o întrerupere a zidului de la vest sau de la est, și doar dacă nu există o nicio altă mașină parcată în sensul ei de deplasare către ieșire. O mașină parcată paralel cu marcajele verticale se poate deplasa pe coloană în sus sau în jos, putând ieși din parcare dacă ajunge la o întrerupere a zidului de la spre nord sau de la sud, și doar dacă nu există nicio altă mașină parcată în sensul ei de deplasare către ieșire. Mașinile se pot deplasa mergând în față sau în marșarier. De exemplu, mașina parcată în locul (2,2) paralel cu marcajele orizontale, nu poate ieși din parcare, deoarece dacă se deplasează spre vest ajunge la zid, iar dacă se deplasează spre est, nu poate ajunge la întreruperea din zid corespunzătoare acelei linii din cauza altor mașini parcate pe traseu, dar mașina din locul (2,6) poate ieși deplasându-se spre est.

Ieșirea din parcare se face în serii consecutive, mașinile dintr-o serie începând deplasarea simultan (după ce au ieșit toate mașinile din seria anterioară); din seria curentă fac parte toate mașinile care, exact la acel moment, au drumul liber spre cel puțin o întrerupere de zid (nu este necesară mișcarea niciunei alte mașini rămase în parcare, inclusiv a celor din seria curentă), chiar dacă traseul lor către ieșire se intersectează cu traseul altor mașini din aceeași serie, aflate în deplasare. În prima serie ies toate mașinile care pot pleca din parcare imediat, fără a fi condiționate de mutarea sau părăsirea parcării de către alte mașini.

Cerințe

Scrieți un program care, cunoscând dimensiunile parcării, pozițiile întreruperilor din zid, numărul de mașini, iar pentru fiecare mașină numărul liniei și al coloanei corespunzătoare locului în care este parcată și modul de parcare a acesteia, rezolvă următoarele două cerințe:

1. determină numărul de mașini care pot ieși din parcare fără a fi condiționate de mutarea sau de părăsirea parcării de către alte mașini (numărul de mașini care pot ieși în prima serie);
2. determină numărul total mașini care pot ieși din parcare, precum și numărul de serii în care se realizează ieșirea tuturor acestor mașini.

Date de intrare

Fișierul de intrare `parking.in` conține pe prima linie numerele naturale c, n, m, k și q reprezentând, în ordine, numărul cerinței, numărul de linii și numărul de coloane ale parcării, numărul de întreruperi ale zidului parcării, respectiv, numărul de mașini parcate. Pe următoarele k linii se află câte două numere naturale i și j reprezentând pozițiile (linia, respectiv coloana) întreruperilor din zidul parcării. Pe următoarele q linii se află câte trei numere naturale x, y și p reprezentând, în ordine, numărul liniei și al coloanei corespunzătoare locului în care este parcată o mașină și modul de parcare a acesteia ($p = 0$ pentru parcare paralelă cu marcajele orizontale, respectiv $p = 1$ pentru parcare paralelă cu marcajele verticale). Valorile scrise pe aceeași linie sunt separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire `parking.out` va conține o singură linie ce va conține:

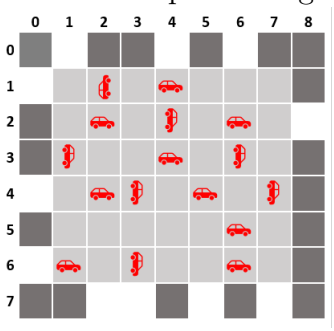
- numărul de mașini care au ieșit din parcare în prima serie, dacă $c = 1$;
- numărul total de mașini care au ieșit din parcare și numărul de serii în care s-a realizat ieșirea mașinilor, separate printr-un spațiu, dacă $c = 2$.

Restricții

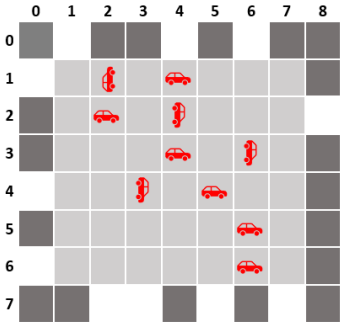
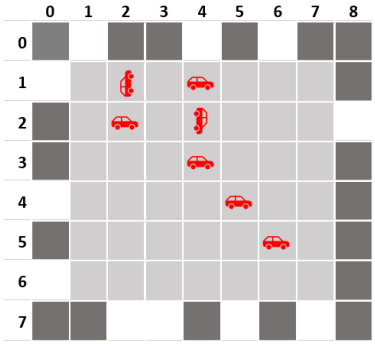
- $2 \leq n, m \leq 1\,000$
- $2 \leq k \leq 2 \cdot n + 2 \cdot m$
- $2 \leq q \leq min(n \cdot m, 100\,000)$
- $1 \leq i \leq n$, pentru $j = 0$ sau $j = m + 1$ și $1 \leq j \leq m$, pentru $i = 0$ sau $i = n + 1$
- $1 \leq x \leq n$, $1 \leq y \leq m$, $0 \leq p \leq 1$
- Se garantează pentru datele de test că numărul de serii obținute va fi cel mult 10 000, dacă $c = 2$.
- Pe parcursul deplasării pentru a ieși din parcare mașinile vor circula astfel încât nu se vor ciocni.

#	Puncte	Restricții
1	12	$C = 1$, $n, m, q \leq 150$
2	16	$C = 1$ și nu există restricții suplimentare.
3	12	$C = 2$, $n, m, q \leq 150$
4	60	$C = 2$ și nu există restricții suplimentare.

Exemple

parking.in	parking.out	Explicații
1 6 7 11 16 0 1 0 4 0 6 7 2 7 3 7 5 7 7 1 0 4 0 6 0 2 8 1 2 1 1 4 0 2 2 0 2 4 1 2 6 0 3 1 1 3 4 0 3 6 1 4 2 0 4 3 1 4 5 0 4 7 1 5 6 0 6 1 0 6 3 1 6 6 0	6	Datele corespund imaginii următoare.  Pot ieși din parcare fără a aștepta mutarea sau plecarea altor mașini cele 6 mașini aflate în locurile de parcare din pozițiile: (2,6) – spre est (3,1) – spre nord (4,2) – spre vest (4,7) – spre sud (6,1) – spre vest (6,3) – spre sud

Exemple

parking.in	parking.out	Explicații
2 6 7 11 16 0 1 0 4 0 6 7 2 7 3 7 5 7 7 1 0 4 0 6 0 2 8 1 2 1 1 4 0 2 2 0 2 4 1 2 6 0 3 1 1 3 4 0 3 6 1 4 2 0 4 3 1 4 5 0 4 7 1 5 6 0 6 1 0 6 3 1 6 6 0	10 3	Seria 1: ies din parcare cele 6 mașini, specificate la exemplul precedent. Seria 2: pot ieși din parcare cele 3 mașini aflate în locurile de parcare din pozițiile: (3,6) – spre nord (4,3) – spre sud (6,6) – spre vest  Seria 3: poate ieși din parcare o singură mașină aflată în locul de parcare din poziția: (4,5) – spre vest  Vor putea părăsi parcarea în total $6 + 3 + 1 = 10$ mașini în 3 serii.

3.2 Rezolvarea problemei Parking

Vom reprezenta o mașină ca o structură (`struct`) cu 4 câmpuri: linia și coloana pe care se află mașina, orientarea mașinii (0 pentru orizontal, 1 pentru vertical) și numărul seriei în care mașina a ieșit, respectiv 0 dacă mașina este încă în parcare. Vom reține mașinile într-un vector vm, astfel vom putea identifica în continuare fiecare mașină prin poziția sa în vector (un număr între 1 și q). O ieșire din parcare o vom identifica prin poziția sa (linia și coloana pe care se află). Vom reține ieșirile într-un vector vi.

Pentru a identifica mașinile care pot ieși din parcare putem parcurge vectorul de mașini și pentru fiecare mașină verificăm dacă poate ieși din parcare în seria curentă (ajunge la o ieșire deplasându-se în una dintre cele două direcții de mișcare posibile) și dacă da, o reținem în seria curentă. O astfel de abordare obține un punctaj substanțial, dar pe testele mari va obține depășire de timp, datorită numărului mare de mașini, complexitatea fiind $\mathcal{O}(q \cdot nrserii)$.

O abordare mai eficientă ar fi să parcurgem ieșirile și să verificăm, pentru fiecare ieșire, dacă există o mașină care poate ieși în seria curentă utilizând ieșirea respectivă. Apar 4 cazuri:

1. dacă ieșirea se află pe coloana 0, această ieșire poate fi utilizată doar de prima mașină de pe linia pe care se află ieșirea (dacă aceasta este poziționată orizontal);
2. dacă ieșirea se află pe coloana $m+1$, această ieșire poate fi utilizată doar de ultima mașină de pe linia pe care se află ieșirea (dacă aceasta este poziționată orizontal);
3. dacă ieșirea se află pe linia 0, această ieșire poate fi utilizată doar de prima mașină de pe coloana pe care se află ieșirea (dacă aceasta este poziționată vertical);
4. dacă ieșirea se află pe linia $n+1$, această ieșire poate fi utilizată doar de ultima mașină de pe coloana pe care se află ieșirea (dacă aceasta este poziționată vertical).

Când analizăm cazurile (1) și (2), respectiv (3) și (4) trebuie să avem grijă la situația în care există o singură mașină pe linie, respectiv pe coloană, situație în care această mașină este atât prima, cât și ultima (deci trebuie să evităm să scoatem din parcare aceeași mașină de două ori și să contorizăm eronat mașinile care ies din parcare). Pentru a trata simplu și eficient cele 4 cazuri vom construi două matrici auxiliare ml și mc, având n linii și m coloane, respectiv m linii și n coloane. Pe linia i ($1 \leq i \leq n$) în matricea ml se află lista mașinilor de pe linia i a parcării, sortate crescător după coloană. Pe linia j ($1 \leq j \leq m$) a matricii mc se află lista mașinilor de pe coloana j a parcării, sortate crescător după linie. Ca urmare pentru ieșirile de pe coloana 0 verificăm doar prima mașină de pe linia corespunzătoare din ml, pentru ieșirile de pe coloana $m+1$ doar ultima mașină de pe linia corespunzătoare din ml. Similar, pentru ieșirile de pe linia 0 verificăm doar prima mașină de pe coloana corespunzătoare din mc, iar pentru ieșirile de pe linia $n+1$, doar ultima mașină de pe coloana corespunzătoare din mc.

Pentru cerința 1 efectuăm ieșirea mașinilor din prima serie. Pentru cerința 2 vom efectua ieșirea mașinilor în serii succesive, până când în seria curentă nu mai iese nicio mașină din parcare. Pentru ieșirea mașinilor dintr-o serie vom proceda în două etape:

1. Parcurgem ieșirile și identificăm mașinile care pot ieși în seria curentă, reținând aceste mașini într-un vector și memorând în structura care reprezintă mașina numărul seriei curente.
2. Parcurgem vectorul care conține mașinile care pot ieși în seria curentă și eliminăm aceste mașini din matricile ml și mc.

Este obligatorie ieșirea mașinilor dintr-o serie în două etape, în alt mod nu putem simula simultaneitatea ieșirii mașinilor din parcare. Dacă vom elimina mașinile succesiv pe măsură ce identificăm că pot ieși, este posibil să iasă din parcare în seria curentă mașini care nu ar fi trebuit să iasă (de exemplu mașina a poate ieși din parcare în seria curentă; mașina a blochează mașina b; dacă eliminăm imediat mașina a, atunci când ajungem la mașina b aceasta nu va mai fi blocată și va ieși și ea în aceeași serie cu a, ceea ce este evident incorect).

Complexitatea algoritmului: pentru fiecare serie, în etapa 1 se parcurg toate ieșirile pentru a identifica mașinile ($\mathcal{O}(K)$), iar în etapa 2 trebuie să eliminăm mașinile care ies în seria curentă din ml și mc (din una dintre matrici se poate elimina în $\mathcal{O}(1)$, dar pentru cealaltă matrice este necesară o căutare secvențială a mașinii urmată de eliminarea acesteia, deci complexitatea va fi $\mathcal{O}(lg)$, unde cu lg notăm numărul de mașini existente pe linia/coloana respectivă (deci putem spune că, în cazul cel mai defavorabil, eliminarea unei mașini din ml, respectiv mc se realizează în timp liniar în funcție de n, respectiv de m).

Există algoritmi mai eficienți de rezolvare a acestei probleme, dar necesită cunoștințe care depășesc programa clasei a VII-a.

3.3 Cod-sursă pentru problema Parking

```
#include <fstream>
#include <vector>
#include <algorithm>
#define NMAX 1004
#define QMAX 100002
#define KMAX 4002
using namespace std;
ifstream fin("parking.in");
ofstream fout("parking.out");
int cerinta, n, m, k, q, nrserii, total;
struct masina {short int lin, col; int serie; char p;};
struct iesire {short int lin, col;};
masina vm[QMAX];
iesire vi[KMAX];
int ml[NMAX][NMAX];
int mc[NMAX][NMAX];
int lgl[NMAX];
int lgc[NMAX];
int s[NMAX], lgs;

bool comparlin(int a, int b)
{return vm[a].col<vm[b].col;}
bool comparcol(int a, int b)
{return vm[a].lin<vm[b].lin;}

void elimin (int a[], int &lg, int x);
int serie();
int main()
{int i, nr, lin, col, poz;
  fin>>cerinta>>n>>m>>k>>q;
  //citesc iesirile
  for (i=1; i<=k; i++)
    {fin>>vi[i].lin>>vi[i].col;}
  //citesc masinile
  for (i=1; i<=q; i++)
    {fin>>lin>>col>>poz;
      ml[lin][lgl[lin]++]=i;
      mc[col][lgc[col]++]=i;
      vm[i].lin=lin; vm[i].col=col; vm[i].p=poz;
    }
  //sortez masinile
  for (i=1; i<=n; i++) sort(ml[i], ml[i]+lgl[i],comparlin);
  for (i=1; i<=m; i++) sort(mc[i], mc[i]+lgc[i],comparcol);
  if (cerinta==1) {fout<<serie()<<'\n';}
  else
    {while (1)
        {nr=serie();
          if (nr==0) break;
          else total+=nr;
        }
      fout<<total<<' '<<nrserii-1<<'\n';
    }
  return 0;
}

int serie()
{int rez=0, i, prima, ultima, lin, col, x, poz, j;
  nrserii++;
  //identific masinile care pot iesi in aceasta serie
```

```
  lgs=0;
  for (i=1; i<=k; i++)
    if (vi[i].col==0) //V
      {lin=vi[i].lin;
        //daca prima masina de pe linie este orizontala, poate iesi
        if (lgl[lin]) //exista masini
          {prima=ml[lin][0];
            if (vm[prima].p==0)
              if (vm[prima].serie==0) {vm[prima].serie=nrserii; rez++; s[lgs++]=prima;}
          }
      }
    else
      if (vi[i].col==m+1) //E
        {lin=vi[i].lin;
          if (lgl[lin]) //exista masini
            {ultima=ml[lin][lgl[lin]-1];
              if (vm[ultima].p==0)
                if (vm[ultima].serie==0) {vm[ultima].serie=nrserii; rez++; s[lgs++]=ultima;}
            }
        }
      else
        if (vi[i].lin==0) //N
          {col=vi[i].col;
            //daca prima masina de pe coloana este veticala, poate iesi
            if (lgc[col]) //exista masini
              {prima=mc[col][0];
                if (vm[prima].p==1)
                  if (vm[prima].serie==0) {vm[prima].serie=nrserii; rez++; s[lgs++]=prima;}
              }
          }
        else //S
          {col=vi[i].col;
            if (lgc[col]) //exista masini
              {ultima=mc[col][lgc[col]-1];
                if (vm[ultima].p==1)
                  if (vm[ultima].serie==0) {vm[ultima].serie=nrserii; rez++; s[lgs++]=ultima;}
              }
          }
  for (i=0; i<lgs; i++)
    {x=s[i]; lin=vm[x].lin; col=vm[x].col;
      //elimin masina x de pe linia lin
      for (poz=0; poz<lgl[lin] && ml[lin][poz]!=x; poz++);
      //x se afla pe pozitia poz, elimin
      for (j=poz+1; j<lgl[lin]; j++) ml[lin][j-1]=ml[lin][j];
      lgl[lin]--;
      //elimin masina x de pe coloana col
      for (poz=0; poz<lgc[col] && mc[col][poz]!=x; poz++);
      //x se afla pe pozitia poz, elimin
      for (j=poz+1; j<lgc[col]; j++) mc[col][j-1]=mc[col][j];
      lgc[col]--;
    }
  return lgs;
}
```

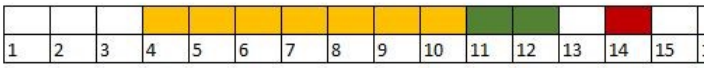
3.4 Problema Ron

Propusă de: prof. Filonela Bălașa, Colegiul Național „Grigore Moisil" București

Ron Weasley dorește să-l ajute pe Harry Potter să construiască baghete magice. Hermione este plecată, așa că Ron trebuie să se descurce singur și știm cum ies vrăjile lui... Din acest motiv, are nevoie de ajutorul tău. Ron are crengi de soc de diferite lungimi și o riglă magică marcată, în dreptul fiecărui centimetru, cu numere naturale consecutive, începând cu 1. Un număr natural aflat pe rigla magică este fermecat dacă are proprietatea că este obținut prin ridicarea la puterea a doua a unui număr prim. Puterea unei crengi este egală cu numărul de numere fermecate acoperite de aceasta pe rigla magică.

Pentru a crea o baghetă, Ron așază crengile pe riglă cu capătul din stânga al fiecărei crengi exact în dreptul numărului natural marcat pe riglă. În cazul în care două sau mai multe crengi se suprapun sau se ating se formează o singură baghetă. Dacă două sau mai multe crengi de soc așezate pe riglă nu se suprapun și nu se ating, se obțin baghete diferite.

De exemplu, dacă Ron așază pe riglă o creangă de soc de lungime 7 centimetri, cu capătul din stânga la poziția marcată cu numărul 4, creanga va avea puterea 2 (fiindcă va acoperi numerele fermecate 4 și 9). Dacă mai așază pe riglă o altă creangă de lungime 2 centimetri, cu capătul din stânga la poziția marcată cu numărul 11, această creangă va avea puterea 0 și cele două crengi așezate vor forma împreună o baghetă pentru că se ating. Dacă mai așază pe riglă o altă creangă, de lungime 1 centimetru, cu capătul din stânga la poziția marcată cu numărul 14, această creangă va avea puterea 0 și va forma singură o baghetă pentru că nu se suprapune și nu se atinge de alte crengi.



Cerințe

Scrieți un program care, cunoscând numărul de crengi de soc și pentru fiecare dintre acestea poziția pe riglă la care se plasează capătul din stânga și lungimea crengii măsurată în centimetri, rezolvă următoarele două cerințe:

1. să se determine puterea cea mai mare pe care o are una dintre crengile folosite de Ron;
2. să se determine numărul de baghete magice realizate de Ron.

Date de intrare

Fișierul de intrare `ron.in` conține pe prima linie numerele c și n reprezentând cerința care trebuie rezolvată (1 sau 2), respectiv numărul de crengi de soc. Pe următoarele n linii sunt descrise crengile de soc, câte o creangă pe o linie, sub forma a două numere naturale *poz* și *lg* reprezentând, în această ordine, numărul natural aflat pe rigla magică în dreptul căruia a fost așezat capătul din stânga al crengii, respectiv lungimea în centimetri a acesteia. Numerele scrise pe aceeași linie sunt separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire `ron.out` va conține o singură linie pe care va fi scris un număr natural reprezentând rezultatul la cerința c din fișierul de intrare.

Restricții

- $2 \leq n \leq 50\,000$
- $1 \leq poz, lg \leq 10^9$

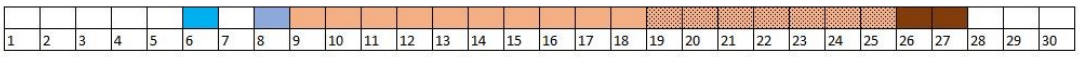
#	Puncte	Restricții
1	22	$C = 1$; $n \leq 1\,000$; $poz, lg \leq 100\,000$
2	20	$C = 1$ și nu există restricții suplimentare.
3	25	$C = 2$; $n \leq 1\,000$; $poz, lg \leq 100\,000$
4	33	$C = 2$ și nu există restricții suplimentare.

Exemple

ron.in	ron.out
1 4 19 9 6 1 9 17 8 1	2
2 4 19 9 6 1 9 17 8 1	2

Explicații

Prima creangă are puterea 1 (acoperă numărul fermecat 25), a doua creangă are puterea 0, a treia creangă are puterea 2 (acoperă numerele fermecate 9 și 25), iar a patra creangă are puterea 0. Pentru primul exemplu ($c = 1$), puterea maximă a uneia dintre crengile folosite de Ron este 2. Pentru al doilea exemplu ($c = 2$), se obțin două baghete: prima, a treia și a patra creangă realizează împreună o baghetă pentru că se suprapun sau se ating. A doua creangă realizează singură o baghetă.



3.5 Rezolvarea problemei Ron

Observație inițială

O creangă de lungime lg așezată pe riglă cu capătul din stânga în dreptul numărului poz reprezintă, din punct de vedere matematic, intervalul $[poz, poz + lg - 1]$.

Cerința 1

Pentru rezolvarea cerinței 1, este necesar să determinăm numărul maxim de numere fermecate aflate în unul dintre intervalele corespunzătoare crengilor de soc. Matematic, numărul pătratelor numerelor prime dintr-un interval $[x, y]$ este egal cu numărul numerelor prime din intervalul $[\sqrt{x}, \sqrt{y}]$.

Vom utiliza Ciurul lui Eratostene pentru a determina numerele prime mai mici sau egale cu $\sqrt{2000000000}$, apoi vom construi un vector auxiliar a în care vom memora pentru fiecare număr natural x, numărul numerelor prime mai mici sau egale cu x. Numărul pătratelor numerelor prime din intervalul $[x, y]$ este $a[\sqrt{y}] - a[\sqrt{x-1}]$.

Vom citi succesiv descrierea fiecărei crengi de soc, vom determina numărul numerelor fermecate acoperite de fiecare creangă și vom reține maximul.

Cerința 2

Pentru rezolvarea cerinței 2 este necesar să determinăm numărul de intervale obținute în urma reuniunii celor n intervale corespunzătoare crengilor de soc.

Vom ordona intervalele crescător după extremitatea inițială (capătul din stânga). Vom parcurge intervalele în ordine și vom reuni toate intervalele care pot fi reunite în unul singur (să numim rezultatul reuniunii interval-reuniune). La fiecare pas comparăm capătul din stânga al intervalului curent (st_i) cu capătul din dreapta al intervalului-reuniune curent (să-l notăm drR). Dacă $st_i \leq 1 + drR$ (cele două crengi se suprapun sau se ating) atunci intervalul curent poate fi adăugat la intervalul-reuniune curent (adică $drR = dr_i$, dacă $dr_i > drR$). În caz contrar, intervalul-reuniune curent nu mai poate fi extins, ca urmare creăm un nou interval-reuniune, pe care îl inițializăm cu intervalul curent.

Problema admite numeroase abordări pentru punctaje parțiale. De exemplu, pentru valorile mici ale lui *poz* și *lg* se poate utiliza un vector caracteristic în care vor fi marcate cu 1 pozițiile ocupate de crengile de soc, problema reducându-se astfel la determinarea numărului de secvențe formate numai din 1.

3.6 Cod-sursă pentru problema Ron

```
#include <fstream>
#include <algorithm>
#include<cmath>
using namespace std;
ifstream f("ron.in");
ofstream g("ron.out");

struct creanga {int a,b;} v[50005];
bool p[50005];
int a[50005];

bool compar (creanga x, creanga y)
{
  if ((x.a<y.a)||(x.a==y.a&&x.b<=y.b)) return true;
  return false;
}
int main()
{int n, i, j, nr, st, dr, k=1, x, cer, maxim=-1, rst, rdr;
 a[0]=0; a[1]=0;
 for (i = 2; i<=50000; i ++)
     {a[i]=a[i-1];
      if (p[i] == 0)
         {a[i]++;
          for (j = 2 ; i * j <= 50000; j++) p[i*j] = 1;
         }
    }
 f>>cer>>n;
 for (i=0; i<n; i++)
```

```
    {f>>v[i].a>>x;
     v[i].b=v[i].a+x-1;
     if (v[i].a==0) rst=0;
        else
        rst=sqrt(v[i].a-1);
    rdr=sqrt(v[i].b);
    nr=a[rdr]-a[rst];
    if (nr>maxim) maxim=nr;
    }
f.close();
if (cer==2)
   {
    sort(v,v+n,compar);
    st=v[0].a; dr=v[0].b;
    i=1;
    do {
       if (v[i].a<=dr+1)
          {if (v[i].b>dr) dr=v[i].b; }
           else
           {
               k++;
               st=v[i].a;
               dr=v[i].b;
           }
        i++;
       }while(i<n);
   }
   if (cer==2) g<<k<<'\n';
       else g<<maxim;
   g.close();
   return 0;
}
```

Capitolul 4

OJI 2024, clasa a VIII-a

4.1 Problema MUN

Propusă de: prof. Lucia Miron, Colegiul Național „Costache Negruzzi" Iași

La o conferință MUN (Model United Nations) participă N delegați din întreaga lume. Fiecare delegat primește un cod format din cel puțin una și cel mult 10 litere mari distincte ale alfabetului englez. Delegații din aceeași țară au codul format din exact aceleași litere, eventual dispuse în altă ordine. Codurile a doi delegați din țări distincte diferă prin cel puțin o literă care apare în unul, dar nu și în celălalt. Numărul de delegați din țara gazdă este cel puțin jumătate plus unu din numărul total de delegați care participă la conferință și sunt cel puțin două țări reprezentate la conferință. La acest gen de conferință există o masă rotundă. Delegații care vor lua loc la masa rotundă sunt cei care dezbat (numiți **vorbitori**), iar toți ceilalți vor primi statut de **supervizori**. Rolul fiecărui delegat precum și numărul de delegați care iau loc la masa rotundă nu sunt prestabilite, dar protocolul prevede că la această masă nu pot sta doi delegați din aceeași țară în poziții alăturate (la stânga sau la dreapta).

Cerințe

1. Să se determine D, **numărul delegațiilor, adică numărul de țări** reprezentate la conferință de cel puțin un delegat.
2. Să se determine două numere naturale, S și V, S reprezentând **numărul minim de delegați care pot primi statut de supervizor**, iar V **numărul de vorbitori corespunzător numărului** S **determinat**.
3. Să se afișeze codurile corespunzătoare numărului maxim de vorbitori ce pot sta la masa rotundă, în ordinea așezării la masă, începând de la oricare dintre ei, astfel încât dacă sunt mai multe posibilități de aranjare se va afișa cea mai mică din punctul de vedere lexicografic.

Date de intrare

Pe prima linie a fișierului de intrare `mun.in` este scris un număr natural c, reprezentând cerința ($1, 2$ sau 3).

Pe linia a doua a fișierului este scris un număr natural N, reprezentând numărul delegaților participanți la conferință, iar pe a treia linie sunt scrise N cuvinte formate din cel puțin una și cel mult 10 litere mari ale alfabetului englez, reprezentând codurile acestor delegați. Cuvintele sunt separate prin câte un spațiu.

Date de ieșire

Dacă $c = 1$, pe prima linie a fișierului de ieșire `mun.out` va fi scris un număr natural D, cu semnificația din enunț.

Dacă $c = 2$, pe prima linie a fișierului de ieșire `mun.out` vor fi scrise două numere naturale S și V, separate printr-un spațiu, cu semnificația din enunț.

Dacă $c = 3$, pe prima linie a fișierului de ieșire `mun.out` va fi scris șirul de cuvinte, separate prin câte un spațiu, reprezentând codurile vorbitorilor care stau la masa rotundă, conform cerinței 3.

Restricții

- $5 \leq N \leq 10000$
- Spunem că șirul $x_1, x_2 \dots x_v$ este mai mic din punctul de vedere lexicografic decât șirul $y_1, y_2 \dots y_v$ dacă există un indice k $(1 \leq k \leq v)$ astfel încât $x_i = y_i$, pentru orice $1 \leq i < k$ și $x_k < y_k$.

#	Puncte	Restricții
1	30	$C = 1$, codurile delegațiilor sunt formate dintr-o singură literă și $N \leq 1000$
2	10	$C = 1$
3	20	$C = 2$, $N \leq 1000$
4	10	$C = 2$
5	20	$C = 3$, $N \leq 1000$
6	10	$C = 3$

Exemple

mun.in	mun.out	Explicații
1 5 A B B A A	2	Sunt trei delegați cu codul A și doi delegați cu codul B (sunt 2 delegații)
2 5 ABC BA AB BCA ABC	1 4	Codurile ABC, BCA și ABC sunt formate din aceleași litere, deci delegații cu aceste coduri sunt din aceeași țară. Celelalte două coduri sunt pentru doi delegați proveniți din altă țară. Rezultă că sunt două țări reprezentate la conferință. Pentru a respecta protocolul putem așeza cel mult 4 persoane la masa rotundă. A cincea persoană va primi statutul de supervizor.
3 5 ABC XA AX BCA BAC	ABC AX BAC XA	Mai există și alte modalități de aranjare circulară(cum ar fi ABC XA BCA AX), dar aranjarea delegaților în ordinea **ABC AX BAC XA** este cea mai mică din punctul de vedere lexicografic.

4.2 Rezolvarea problemei MUN

Cerința 1

Soluția în complexitate $O(N \log N)$ obține punctaj maxim, adică 40 de puncte. Se determină pentru fiecare cod, codul ordonat crescător; se ordonează delegații crescător după codul ordonat, dacă doi delegați de pe poziții consecutive au codul ordonat diferit, se incrementează numărul de țări. Deoarece valoarea lui N nu este foarte mare, există și alte abordări mai puțin eficiente care în funcție de implementare pot obține punctajul maxim. În cazul testelor cu $N \leq 1000$ și codurile formate dintr-un singur caracter, o soluție care utilizează vector de apariții pentru caractere, corect implementată, va obține 30 de puncte.

Cerința 2

Soluția în complexitate $O(N)$ obține punctaj maxim, adică 30 de puncte. Se determină în mx_freq, frecvența elementului majoritar (țara gazdă) utilizând un algoritm liniar, pe vectorul ordonat obținut la cerința 1, se calculează lungimea maximă a unei secvențe care are codurile ordonate crescător egale (sunt mai multe abordări, se poate utiliza și algoritmul clasic de determinare a elementului majoritar). Valoarea afișată pentru S este $N - 2 \cdot (N - \text{mx\_freq})$. Valoarea afișată pentru V va fi $2 \cdot (N - \text{mx\_freq})$.

Cerința 3

Soluția în complexitate $O(N \log N)$ obține punctaj maxim, adică 30 de puncte. Se vor construi doi vectori de coduri: codurile delegaților care fac parte din țara gazdă (hosts[]) și codurile celorlalți delegați (rest[]); se vor ordona lexicografic, se va afișa un cod dintr-o mulțime și unul din cealaltă mulțime conform ordinii lexicografice, având grijă să nu se depășească numărul de persoane de la masa rotundă. O soluție care determină ordinea lexicografică în complexitate $O(N^2)$ poate obține în funcție de implementare 70 - 80 de puncte.

4.3 Cod-sursă pentru problema MUN

```
#include <bits/stdc++.h>
using namespace std;
const int C_MAX = 10;
const int N_MAX = 10000;

struct Participant
{
    char country[C_MAX + 5];
    char id[C_MAX + 5];
    int len;
};

int N;
char host[C_MAX + 5];
Participant P[N_MAX + 5];
Participant hosts[N_MAX + 5], rest[N_MAX + 5];

bool cmp(Participant &a, Participant &b)
{
    int x = strcmp(a.country, b.country);
    int y = strcmp(a.id, b.id);
    if (x < 0) return true;
```

```
    if (x > 0) return false;
    if (y < 0) return true;
    return false;
}

bool cmp_str(Participant &a, Participant &b)
{
    return strcmp(a.id, b.id) < 0;
}

int main()
{
#ifndef LOCAL
    freopen("mun.in", "r", stdin);
    freopen("mun.out", "w", stdout);
#endif
    ios::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);

    int task;
    cin >> task >> N;
    for(int i = 1; i <= N; i++)
    {
        cin >> P[i].id;
        strcpy(P[i].country, P[i].id);
        P[i].len = strlen(P[i].id);
        sort(P[i].country, P[i].country + P[i].len);
    }

    sort(P + 1, P + N + 1, cmp);

    int mx_freq = 0, curr_freq = 1;
    int countries = 1;
    strcpy(host, P[1].country);
    for(int i = 2; i <= N; i++)
    {
        if (strcmp(P[i].country, P[i - 1].country) == 0)
            curr_freq++;
        else
        {
            countries++;
            curr_freq = 1;
        }
        if(mx_freq < curr_freq)
        {
            mx_freq = curr_freq;
            strcpy(host, P[i].country);
        }
    }

    if(task == 1)
    {
        cout << countries << "\n";
        return 0;
    }
    if(task == 2)
    {
        cout << N - 2 * (N - mx_freq) << " " << 2 * (N - mx_freq) << "\n";
        return 0;
    }
```

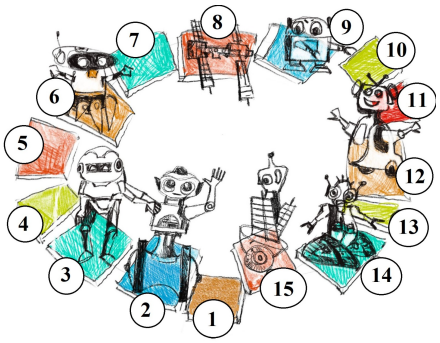
```
    int n_hosts = 0, n_rest = 0;
    for(int i = 1; i <= N; i++)
    {
        if(strcmp(P[i].country, host) == 0)
            hosts[++n_hosts] = P[i];
        else
            rest[++n_rest] = P[i];
    }

    sort(hosts + 1, hosts + n_hosts + 1, cmp_str);
    sort(rest + 1, rest + n_rest + 1, cmp_str);

    if(strcmp(hosts[1].id, rest[1].id) < 0)
    {
        for(int i = 1; i <= n_hosts && i <= n_rest; i++)
            cout << hosts[i].id << " " << rest[i].id << " ";
    }
    else
    {
        for(int i = 1; i <= n_hosts && i <= n_rest; i++)
            cout << rest[i].id << " " << hosts[i].id << " ";
    }
    return 0;
}
```

4.4 Problema Robotron

Propusă de: prof. Isabela Patricia Coman, Colegiul Național de Informatică „Tudor Vianu” București

În sistemul solar **Stelarion** sunt 99 de planete. Planeta **Hazard** găzduiește campionatul de **Robotron** pe echipe. Jucătorii sunt înregistrați în ordinea sosirii lor, indiferent de planeta de pe care provin. Anul acesta s-au înscris în campionat N jucători de pe M planete. Jucătorul înregistrat al i-lea (cu i de la 1 la N) primește două numere: E_i — numărul trecut pe ecuson și P_i — puterea jucătorului. Numărul trecut pe ecuson este format din codul planetei jucătorului CP (numărul format din ultimele două cifre de pe ecuson) și codul jucătorului CJ (numărul format din restul cifrelor). Jucătorii care reprezintă aceeași planetă vor fi repartizați în aceeași echipă, poziția pe care o ocupă inițial în echipă fiind în ordinea înregistrării lor în concurs. În cadrul campionatului se vor disputa mai multe runde, iar la fiecare rundă va fi desemnată o echipă câștigătoare.

Regulamentul de concurs



- O rundă constă în parcurgerea unui circuit format din L căsuțe, numerotate de la 1 la L, în sens orar, plasate ca în figură.
- Fiecare echipă dispune de un pion care este plasat în căsuța 1 la începutul fiecărei runde; o mutare a jucătorului al i-lea în ordinea înregistrării constă în deplasarea pionului echipei proprii cu P_i căsuțe, în sens orar.
- În fiecare rundă, echipele joacă ciclic, în secvențe succesive, fiecare secvență fiind în ordinea strict crescătoare a codurilor planetelor de pe care provin. Într-o secvență joacă toate echipele, doar ultima secvență a rundei putând fi incompletă, după caz. Ordinea echipelor nu se schimbă de la o rundă la alta. Când este rândul unei echipe să joace, unul dintre membrii acesteia mută pionul echipei respective.
- La prima rundă, în cadrul fiecărei echipe, jucătorii ocupă pozițiile inițiale, stabilite la înscriere. Pentru fiecare altă rundă care urmează, în fiecare echipă, se schimbă poziția jucătorilor, aceștia permutându-se circular. Astfel cel care a fost primul la runda anterioară devine ultimul, iar cel care fusese al doilea va fi acum pe prima poziție.
- La fiecare rundă, în cadrul fiecărei echipe, jucătorii mută pionul echipei pe rând, în ordinea poziției pe care o ocupă în echipă în runda respectivă. După jucătorul care ocupă ultima poziție în echipă la runda curentă va muta jucătorul care ocupă prima poziție.
- **Runda se încheie când pionul uneia dintre echipe a parcurs tot circuitul, ajungând din nou la poziția** 1 **sau trecând peste aceasta**. Această echipă este desemnată ca fiind echipa câștigătoare a rundei, iar jucătorul care aduce victoria echipei sale este cel care face ultima mutare, de tipul precizat.

Cerințe

1. Să se determine numărul M al echipelor participante și codul H al planetei gazdă Hazard, știind că numărul jucătorilor din echipa planetei gazdă este strict mai mare decât numărul jucătorilor oricărei alte echipe.
2. Să se determine codul planetei de pe care provine echipa câștigătoare la runda K și codul jucătorului care aduce victoria acestei echipei la aceasta rundă.

Date de intrare

Fișierul de intrare `robotron.in` conține pe prima linie un număr C, reprezentând numărul cerinței, pe a doua linie, 3 numere naturale N, L și K cu semnificația din enunț. Pe următoarele N linii sunt datele jucătorilor: pe a i-a dintre aceste linii se află câte două numere naturale E_i și P_i, cu semnificația din enunț. Numerele de pe aceeași linie a fișierului sunt separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire `robotron.out` conține pe prima linie, pentru $C = 1$, numerele M și H , iar pentru $C = 2$, codul planetei și al jucătorului precizate la cerința 2. Numerele de pe aceeași linie a fișierului sunt despărțite printr-un spațiu.

Restricții

- $1 \leq N \leq 10^5$; $1 \leq L \leq 10^9$; $1 \leq K \leq 10^9$
- $101 \leq E_i < 10^9$; $1 \leq P_i \leq 10^6$; $1 \leq CP \leq 99$

#	Puncte	Restricții
1	20	$C = 1$, $N \leq 1000$
2	30	$C = 1$
3	10	$C = 2$, $N \leq 10000$, $L \leq 10000$, $K \leq 1000$
4	10	$C = 2$, $N \leq 10000$, $L \leq 10000$
5	30	$C = 2$

Exemple

robotron.in	robotron.out	Explicații
1 7 23 2 245 5 3103 5 3203 2 3303 5 2245 6 3003 3 231745 1	2 3	Sunt 7 jucători și ei vor fi împărțiți în 2 echipe ($M = 2$), reprezentând planetele cu codurile **3** (jucătorii cu codurile 31, 32, 33, 30) și **45** (jucătorii cu codurile 2, 22, 2317). Codul planetei gazdă Hazard este $H = 3$, echipa planetei 3 fiind cea mai numeroasă.

2 7 23 2 2145 5 3103 5 3203 2 3303 5 2245 6 3003 3 2345 1	45 21	Traseul conține 23 de căsuțe, iar la prima rundă membrii echipelor au pozițiile stabilite la înscriere și vor muta în ordinea acestor poziții: Echipa planetei cu codul **3** are următorii jucători, cu codurile și puterile aferente: 31 5, 32 2, 33 5, 30 3 Echipa planetei cu codul **45** are următorii jucători, cu codurile și puterile aferente: 21 5, 22 6, 23 1 La runda a 2-a pozițiile membrilor se schimbă, primii jucători din fiecare echipă trecând la coadă: Echipa planetei cu codul **3**: 32 2, 33 5, 30 3, 31 5 Echipa planetei cu codul **45**: 22 6, 23 1, 21 5 Echipa planetei cu codul 45 termină jocul în runda a 2-a după 6 mutări, care, pentru această echipă, vor fi efectuate în ordine de jucătorii 22, 23, 21, 22, 23 și 21. Deci ultima mutare o va face jucătorul cu codul $CJ = 21$, care ajunge în căsuța 2, deci trece peste căsuța 1. Ordinea în care mută jucătorii celor două echipe în cadrul acestei runde, precum și numerele căsuțelor ocupate succesiv de cei doi pioni, sunt ilustrate în tabelul de mai jos.

Codurile jucătorilor care mută		32	22	33	23	30	21	31	22	32	23	33	21
Poziția pionului echipei 3	1	3		8		11		16		18		23	
Poziția pionului echipei 45	1		7		8		13		19		20		2

4.5 Rezolvarea problemei Robotron

Cerința 1.

Pentru determinarea valorilor M și H, vom folosi un vector de frecvență. Pentru fiecare ecuson, determinăm echipa din care face parte, luând ultimele două cifre din acesta. M va fi egal cu numărul de poziții din vectorul de frecvența pentru care valoarea este diferită de 0, iar H va fi poziția din vector pentru care frecvența are valoarea maximă.

Cerința 2.

Pentru fiecare echipă, precalculăm sume parțiale. Pentru o anumită echipa i, avem $s[j]$ – suma elementelor aflate pe pozițiile $1, 2, \ldots j$. În această problema, $s[j]$ reprezintă poziția pe care se

va afla pionul după ce va muta al j-lea jucător. La calcularea sumelor, trebuie să ținem cont de faptul că ordinea jucătorilor se permută circular. Jucătorul care inițial este al j-lea în echipă, va fi în runda k pe poziția $(j + (k - 1) \mod cnt) \mod cnt$, unde cnt este numărul total de membri din echipa. Pentru fiecare echipă, simulăm parcurgerea. Având sumele parțiale calculate, putem determina de câte ori mută toată echipa până când jocul ar fi câștigat de echipa respectivă. Pentru ultima secvență de mutări, vom caută secvențial care este jucătorul care va aduce victoria echipei sale. Dintre toate aceste simulări, reținem care este echipa care va face cele mai puține mutări, și jucătorul care aduce victoria echipei sale în acest caz. În cazul în care există mai multe echipe care au număr egal de mutări minime pentru a trece de căsuța de start, se va selecta prima dintre ele, deoarece echipele mută în ordinea codului planetei lor.

Menționăm că există implementări în care simulăm fiecare mutare. Pentru toate echipele, determinăm jucătorul care mută pionul. Vom menține pentru fiecare echipă poziția pionului pe tablă, și aceasta se va actualiza după fiecare mutare. Repetăm această simulare, până când un jucător trece de căsuța de start, moment în care se termină jocul. Acest algoritm ar trebui să obțină, în funcție de implementare, aproximativ 60 de puncte.

4.6 Cod-sursă pentru problema Robotron

```
#include <fstream>
#define NMAX 100000                         // n maxim de jucatori
#define MMAX 99                             // n maxim de echipe
#define LM 2000000000                       // maxim de mutari ce pot fi facute de o echipa
using namespace std;

ifstream fin  ( "robotron.in" );
ofstream fout ( "robotron.out" );

struct player {
  int cp;                   // cod planeta
  int pw;                   // putere
}E[MMAX+1][NMAX+1];         // maxim MMAX echipe a cu maxim NMAX membrii
int L, n[MMAX+1];           // Lungimea traseului, n de membrii ai fiecarei echipe
long long S[NMAX+1];        // sume partiale pentru puterile tuturor candidatilor unei echipe

long long Sim (int i, int p, int &j) { // echipa i, primul jucator, ultimul jucator
  int N = n[i];                        // lungimea listei curente
  long long sum=S[N-1];
  // suma puterilor din echipa i, e ultimul elem din vect de sume partiale
  long long moves=L/sum*N;
  //L/s[i]-de cate ori a mutat toata echipa * N de membrii ai echipei
  long long rest=L-(L/sum)*sum; // cate casute mai am de sarit
  if (rest == 0)                // daca nu mai am casute de sarit
    j=(p+N-1)%N;
    //jucatorul care termina jocul este ultimul din lista circulara ce incepe la pozitia p
  else{         //cautam secvential restul in vectorul de sume partiale
    j = 0;
    while (S[j] < rest) {// cautam secvential restul pe vectorul de sume partiale
      moves ++; j++;     // si contorizam n de mutari
    }
    moves++;
    j=(j+p)%N;   // pozitia jucatorului care va muta ultimul
  }
  return moves;  // n de mutari necesar echipei i pentru a termina traseul
}

int main(){
```

```
  int cer, K, ecuson, P,N;
  int M = 0;              // n de echipe participante
  int maxj = 0;           // n maxim de jucatori dintr-o echipa
  int Hazard;             // codul-ul echipei Hazard, echipa cu cei mai multi participanti
  fin>>cer>>N>>L>>K; // cer - cerinta, L - lungimea traseului, K numarul rundei de analizat
  while (fin >> ecuson >> P) { // citim datele jucatorilor si ii repartizam pe echipe
    int i = ecuson % 100;      // codul planetei
    if ( n[i] == 0 )           // daca echipa i e vida
      M ++;                    // mai am o echipa in plus
    E[i][n[i]].cp=ecuson/100;  // adaug in echipa planetei i codul jucatorului
    E[i][n[i]].pw = P;         // si puterea acestuia
    n[i]++;                    // creste numarul membrilor echipe planetei i
    if ( n[i] > maxj ){        // determinam codul planetei Hazard
      maxj = n[i];             // n maxim de membrii dintr-o echipa
      Hazard = i;              // codul planetei Hazard
    }
  }

  if ( cer == 1 )
    fout << M << " " << Hazard;  // n de echipe, codul planetei Hazard
  else if ( cer == 2 ){
    int min_moves=LM+1;
    int winer_team,winer_player;
    for ( int i = 1; i <= MMAX; i ++ ){ // pt fiecare echipa
      int nr = n[i];                    // extragem n de membrii ai echipei i
      if ( nr ) {                       // daca n de membrii ai echipei e nenul,
        int p=(K-1)%nr;
        //determinam pozitia jucatorului care va muta primul din echipa planetei i
        S[0] = E[i][p].pw;
        //sume partiale pentru puterile echipei planetei i, incepand cu pozitia p
        for (int j=1; j<nr; j++) // sume partiale incepand cu pozitia p
          S[j] = S[j-1] + E[i][(j+p)%nr].pw;
        int j;  // variabila in care transmit jucatorul care va finaliza cursa
        int moves=Sim(i, p, j);
        if (moves < min_moves) {
          min_moves = moves;    // N minim de mutari
          winer_team = i;       // n de ordine al echipei care termina jocul
          winer_player = E[i][j].cp; // numele jucatorului care termina jocul
        }
      }
    }
    fout << winer_team << " " << winer_player;
  }
  return 0;
}
```

Partea a II-a

Olimpiada Națională de Informatică - etapa națională -

Iași, 22-26 aprilie 2024

Capitolul 5

ONI 2024, clasa a V-a

5.1 Problema P13

Propusă de: prof. Ionel-Vasile Piț-Rada, Colegiul Național „Traian" Drobeta-Turnu Severin

Se dau N numere naturale care conțin în scrierea lor doar cifre din mulțimea $\{0, 1, 2, 3, 4\}$. În continuare prin **număr special** se înțelege un număr natural în care apare cel puțin o cifră impară și fiecare cifră impară apare de număr par de ori.

Cerințe

1. Să se calculeze numărul de elemente din șirul dat care sunt **numere speciale**.
2. Să se calculeze numărul secvențelor din șirul dat care au lungimea cel mult egală cu K și în care un singur element este **număr special**.
3. Să se determine și să se afișeze pentru fiecare secvență de lungime K numărul minim de elemente care ar trebui eventual eliminate astfel încât concatenând elementele rămase să obținem un **număr special**, iar dacă din secvență nu se poate obține un număr special atunci se va afișa **-1**.

Date de intrare

În fișierul `p13.in` se găsesc pe prima linie, separate prin câte un spațiu, C, N și K, unde C reprezintă cerința ce trebuie rezolvată, N numărul de valori care trebuie citite, K lungimea cerută la cerințele 2 și 3. Pe a doua linie, separate prin câte un spațiu se află cele N numere.

Date de ieșire

În fișierul `p13.out` se va scrie pe prima linie, valoarea obținută pentru cerințele 1 și 2, iar pentru cerința 3 se vor afișa valorile obținute separate prin câte un spațiu.

Restricții

- $1 \leq C \leq 3$
- $1 \leq K \leq N \leq 100\,000$
- Fiecare număr din șir este un număr natural cu cel mult 6 cifre.

#	Puncte	Restricții
1	21	$C = 1$
2	40	$C = 2$
3	39	$C = 3$

Exemple

p13.in	p13.out	Explicații
1 4 3 121 423 43 3003	2	Numerele speciale sunt 121 și 3003
2 4 3 121 423 43 3003	6	Secvențele de lungime cel mult 3 care conțin exact un singur număr special sunt: 121, 121 423, 121 423 43, 3003, 43 3003, 423 43 3003
3 4 3 121 423 43 3003	0 0	Prin concatenare ambele secvente de lungime 3 devin numere speciale 12142343 și 423433003.

5.2 Rezolvarea problemei P13

Soluția 1

Propusă de: prof. Adrian Panaete, Colegiul Național „A. T. Laurian", Botoșani

Pentru fiecare poziție i din șir calculăm:

$U[i]$ = câte cifre de 1 conține numărul de pe poziția i;

$T[i]$ = câte cifre de 3 conține numărul de pe poziția i.
În funcție de paritățile numerelor $U[i]$ și $T[i]$ vom asocia poziției i un tip $t[i]$ după cum urmează:

- $t[i] = 0$ dacă $U[i] = T[i] = 0$;
- $t[i] = 1$ dacă $U[i] =$ impar $T[i] =$ par;
- $t[i] = 2$ dacă $U[i] =$ par $T[i] =$ impar;
- $t[i] = 3$ dacă $U[i] =$ impar $T[i] =$ impar;
- $t[i] = 4$ dacă $U[i] =$ par $T[i] =$ par și măcar una dintre valorile $U[i]$, $T[i]$ este nenulă.

Numărul de pe poziția i este special atunci când $t[i] = 4$.

Cerința 1

Pentru cerința 1 este suficient să numărăm câte numere din șir au tipul 4.

Cerința 2

Pentru cerința 2 vom număra secvențele după capătul lor din dreapta. Notăm acest capăt cu dr.

Dacă înaintea poziției dr nu am numere speciale atunci nu avem secvențe soluție.

Considerăm acum $s1$ = cea mai apropiată poziție care conține număr special și $s1 \leq dr$

Considerăm $s2$ cea mai apropiată poziție de $s1$ care conține număr special și $s2 < s1$ (dacă nu există o astfel de poziție se ia $s2 = 0$).

Să considerăm un interval soluție [*st*,*dr*].
Observăm că:

1. $st \leq s1$ pentru a avea cel puțin un număr special în interval;
2. $st \geq s2 + 1$ pentru a nu avea un al doilea număr special în interval, sau pentru a avea capătul stânga ≥ 1;
3. $st \geq dr - k + 1$ (pentru ca lungimea secvenței să nu depășească k).

Astfel valoarea minimă pe care o poate lua capătul st este $stMin = max(s2 + 1, dr - k + 1)$ iar valoarea maximă este $stMax = s1$.

Dacă $stMin \leq stMax$ se adună la soluție toate variantele posibile în care putem alege poziția st adică $stMax - stMin + 1$.

Cerința 3

Pentru fiecare interval [*st*,*dr*] de lungime k vom memora:

- $cntU$ = numărul de 1 din interval;
- $cntT$ = numărul de 3 din interval;
- $Last[i] \leq dr$ ultima poziție situată în fața poziției dr unde avem un număr de tip i (0, 1, 2, 3 sau 4).

Asta înseamnă că prin lipirea tuturor numerelor din interval se va forma un număr NR cu $cntU$ cifre de 1 și $cntT$ cifre de 3 iar acest număr va avea un tip TP cu valoarea 0, 1, 2, 3 sau 4 calculat conform regulii descrise mai sus.

De asemenea observăm că pe intervalul [*st*,*dr*] există măcar un număr de tip i dacă $last[i] \geq st$ și astfel vom ști sigur pentru fiecare tip dacă avem sau nu numere de acel tip pe interval. Dacă NR este de tip 0 atunci răspunsul la cerința 3 este -1 deoarece indiferent de eliminări vom rămâne cu număr de tip 0.

Dacă NR este de tip 4 atunci răspunsul la cerința 3 este 0 pentru că NR deja este special.

Dacă NR este de tip 1 avem două alternative:

- eliminăm un număr de tip 1 (o eliminare);
- eliminăm un număr de tip 2 și unul de tip 3 (două eliminări).

Dacă NR este de tip 2 avem două alternative:

- eliminăm un număr de tip 2 (o eliminare);
- eliminăm un număr de tip 1 și unul de tip 3 (două eliminări).

Dacă NR este de tip 3 avem două alternative:

- eliminăm un număr de tip 3 (o eliminare);
- eliminăm un număr de tip 1 și unul de tip 2 (două eliminări).

În toate aceste cazuri va trebui să verificăm dacă avem valori de tipurile necesare eliminărilor, dar asta știm folosind valorile $last[i]$.

De asemenea trebuie să ne asigurăm că, după eliminare, ne mai rămân valori de 1 sau 3. Deoarece noi știm exact din care poziții vom face una sau două eliminări, vom ști dacă nu am obținut eliminarea a $cntU$ de 1 și a $cntT$ de 3 (situație în care nu putem folosi acel tip de eliminare).

Evident că pentru fiecare tip 1, 2 sau 3 vom încerca prima dată alternativa cu o eliminare. Dacă nu este posibilă o eliminare atunci încercăm alternativa cu două eliminări. Dacă nici aceasta nu este posibilă atunci răspunsul va fi -1.

Observație finală: prin cunoașterea valorilor $U[i]$ și $T[i]$ este foarte ușor să recalculăm valorile $cntU$, $cntT$, $last[i]$ pentru orice interval $[st,dr]$ de lungime k știind aceste valori corespunzătoare intervalului $[st-1,dr-1]$.

Soluția 2

Propusă de: prof. Pițt-Rada Ionel-Vasile, Colegiul Național Traian, Drobeta-Turnu Severin

Cerința 1

Se parcurg numerele citite și se verifică la fiecare dintre ele proprietatea de număr special și se actualizează contorul când este cazul.

Cerința 2

Presupunem că numerele date sunt păstrate într-un vector $V[1], V[2], \dots V[n]$.

Pentru fiecare număr special (să zicem $V[p]$) vom căuta spre stânga și spre dreapta două poziții $p1$ și $p2$ astfel încât să avem $1 \le p1 \le p \le p2 \le n$ și $p - p1 + 1 \le K$ și $p2 - p + 1 \le K$ și în secvența $V[p1], \dots V[p2]$ să avem ca număr special doar pe $V[p]$.

Observăm că, în secvența determinată orice secvență inclusă care are lungimea cel puțin K va conține și numărul special $V[p]$.

Numărul total de secvențe incluse în secvența $V[p1], \dots V[p2]$ este $1 + 2 + \dots + (p2 - p1 + 1)$ care conform formulei lui Gauss este egal cu $(p2 - p1 + 1) \cdot (p2 - p1 + 2)/2$.

Din aceste secvențe va trebui să eliminăm secvențele incluse în secvențele $V[p1], \dots V[p-1]$ (cele din stânga numărului special $V[p]$) și $V[p+1], \dots V[p2]$ (cele din dreapta numărului special $V[p]$) care sunt în număr de $(p - p1) \cdot (p - p1 + 1)/2$ și respectiv $(p2 - p) \cdot (p2 - p + 1)/2$, deoarece ele nu conțin numărul special $V[p]$. De asemenea trebuie să eliminăm secvențele care au lungime mai mare decât K. Acestea există numai dacă $p2 - p1 + 1 > K$. În acest caz avem secvențele care au capătul stâng x în $[p1 \dots p2 - K]$ și capătul drept în $[x + K \dots p2]$. Numărul lor este $1+2+\dots+(p2-p1-K+1)$ și conform formulei Gauss avem $(p2-p1-K+1)\cdot(p2-p1-K+2)/2$.

Cerința 3

Analizăm pe rând fiecare secvență de lungime K.

Singurele cazuri în care nu se poate obține prin concatenare un număr special, după eventuale ștergeri, sunt următoarele:

1. în numerele din secvență nu apar cifre impare;
2. singurele numere din secvență care conțin cifre impare trebuie șterse pentru a rezolva imparitatea numărului de apariții ale cifrelor impare.

Cazul (2) conține două subcazuri:

(2.1) avem un singur număr care conține cifre impare, dar care „strică” paritatea cifrelor impare din secvență;

(2.2) avem doar două numere care conțin cifre impare și unul „strică” paritatea cifrei impare 1 din secvență, iar al doilea „strică” paritatea cifrei impare 3 din secvență.

Analizând cele două subcazuri observăm că ar trebui numărate separat elementele care conțin număr total impar de cifre de 1 (vom nota numărul acestora cu $y1$) și respectiv de 3 (numărul acestora cu $y3$), respectiv ambele (număr notat cu $y13$), respectiv ar trebui numărate elementele care conțin cifre impare, dar în număr par de apariții (numere speciale, vom nota cu ysp).
Dacă dintre $y1$, $y3$ și $y13$ sunt toate impare, atunci nu trebuie șters nimic.
Dacă dintre $y1$, $y3$ și $y13$ toate sunt pare, atunci nu trebuie șters nimic.
Dacă dintre $y1$, $y3$ și $y13$ unul este par nenul, iar celelalte două sunt impare, atunci trebuie șters un număr care să scadă numărul par.
Dacă dintre $y1$, $y3$ și $y13$ două sunt pare, iar al treilea este impar, atunci trebuie șters un număr care să scadă pe cel impar.
Dacă dintre $y1$, $y3$ și $y13$ unul este nul iar celelalte sunt impare, atunci trebuie șterse două numere care să scadă pe cele două impare.

5.3 Cod-sursă pentru problema P13

```
#include <bits/stdc++.h>
using namespace std;
ifstream f("p13.in");
ofstream g("p13.out");
const int N = 100010;
int c, n, k, spOld=-1, spNew=0, U[N], T[N], t[N], last[5]={-1,-1,-1,-1,-1}, sol1, sol3, UU, TT;
unsigned sol2;
int main()
{
    f>>c>>n>>k;
    for (int i=1; i<=n; i++)
    {   int x;
        f>>x;
        for (; x; x/=10)
        {
            if (x%10==1) U[i]++;
                else
                if (x%10==3) T[i]++;
        }
        if (U[i]%2 && T[i]%2) t[i]=3;
            else
            if (U[i]%2) t[i]=1;
               else
               if (T[i]%2) t[i]=2;
                  else
                  if (T[i]+U[i]>0) t[i]=4;
                      else t[i]=0;
        if (t[i]==4)
        {
            sol1++;
            spOld=spNew;
            spNew=i;
        }
        if (spOld>=0)
        {
            int st=max(spOld+1,i-k+1);
            if (st<=spNew) sol2+=spNew-st+1;
        }
    }
    if (c==1)
```

```
    {
        g<<sol1<<'\n';
        return 0;
    }
    if (c==2)
    {
        g<<sol2<<'\n';
        return 0;
    }
    for (int i=1; i<k; i++)
    {
        UU+=U[i];
        TT+=T[i];
        last[t[i]]=i;
    }
    for (int st=1,dr=k; dr<=n; st++,dr++)
    {
        UU+=U[dr];
        TT+=T[dr];
        last[t[dr]]=dr;
        sol3=-1;
        if (UU%2&&TT%2)
            {
            if (last[3]>=st&&(UU>U[last[3]]||TT>T[last[3]]))
                sol3=1;
                else
                if (last[1]>=st&&last[2]>=st &&
                    (UU>U[last[1]]+U[last[2]]||TT>T[last[1]]+T[last[2]]))
                    sol3=2;
            }
            else
            if (UU%2)
               {
                if (last[1]>=st&&(UU>U[last[1]]||TT>T[last[1]]))
                    sol3=1;
                    else
                    if (last[2]>=st&&last[3]>=st &&
                        (UU>U[last[2]]+U[last[3]]||TT>T[last[2]]+T[last[3]]))
                       sol3=2;
               }
               else
               if (TT%2)
                  {
                    if (last[2]>=st&&(UU>U[last[2]]||TT>T[last[2]]))
                        sol3=1;
                        else
                        if (last[1]>=st&&last[3]>=st &&
                            (UU>U[last[1]]+U[last[3]]||TT>T[last[1]]+T[last[3]]))
                            sol3=2;
                  }
                  else
                  if (UU+TT>0) sol3=0;
        g<<sol3<<' ';
        UU-=U[st];
        TT-=T[st];
    }
    return 0;
}
```

```cpp
#include <fstream>
using namespace std;
ifstream  fin("p13.in");
ofstream fout("p13.out");
int C,N,K,v[100002],vsp[100002],w[100002];
int main()
{
    fin>>C>>N>>K;
    int x;
    for(int i=1; i<=N; i++)
    {
        fin>>v[i];
        x=v[i];
        int n1=0,n3=0,cif;
        do{
            cif=x%10;
            if(cif==1)n1++;
            else if(cif==3)n3++;
            x=x/10;
        }while(x>0);

        if(n1%2!=0 && n3%2!=0)w[i]=13;
        else if(n1%2!=0)w[i]=1;
        else if(n3%2!=0)w[i]=3;

        if((n1>0 && n1%2==0 && n3==0)||
           (n3>0 && n3%2==0 && n1==0)||
           (n1>0 && n1%2==0 && n3>0 && n3%2==0)){
            vsp[i]=1;
        }
    }
    if(C==1)
    {
        int c1=0;
        for(int i=1;i<=N;i++){
            if(vsp[i]==1){
                c1++;
            }
        }
        fout<<c1;
    }
    if(C==2)
    {
        unsigned c2=0,j=0,p1=0,p2=0,z=0,y=0,z1,z2;
        for(int i=1; i<=N; i++)
        {
            if(vsp[i]==1){
                j=i;
                while(j-1>=1 && vsp[j-1]==0 && j-1+(K-1)>=i){
                    j--;
                }
                p1=j;
                j=i;
                while(j+1<=N && vsp[j+1]==0 && j+1<=i+(K-1)){
                    j++;
                }
                p2=j;
                if((p2-p1+1)%2==0)z=(p2-p1+2)*((p2-p1+1)/2);
                else z=(p2-p1+1)*((p2-p1+2)/2);
                if((i-p1)%2==0)z1=(i-p1+1)*((i-p1)/2);
                else z1=(i-p1)*((i-p1+1)/2);
```

```
                if((p2-i)%2==0)z2=(p2-i+1)*((p2-i)/2);
                else z2=(p2-i)*((p2-i+1)/2);
                z=z - z1-z2;
                if(p2-p1+1>K){
                    if((p2-p1-K+1)%2==0)y=(p2-p1-K+1)*(p2-p1-K+2)/2;
                    else y=(p2-p1-K+2)*(p2-p1-K+1)/2;
                    z=z-y;
                }
                c2=c2+z;
            }
        }
        fout<<c2;
    }
    if(C==3){
        int y1=0,y3=0,y13=0,ysp=0,ok;
        for(int i=1; i<=K; i++){
            if(vsp[i]==1)ysp++;
            if(w[i]==1)y1++;
            else if(w[i]==3)y3++;
            else if(w[i]==13)y13++;
        }
        ok=-1;
        if(y1==0 && y3==0 && y13==0 && ysp==0){
            ok=-1;
        }
        else{
            if(y1%2!=0 && y3%2!=0 && y13%2!=0){
                ok=0;
            }
            else{
                if(y1%2==0 && y3%2==0 && y13%2==0){
                    ok=0;
                }
                else{
                    if((y1%2 + y3%2 + y13%2==2) &&
                       (((y1%2==0)&&(y1>0)) + ((y3%2==0)&&(y3>0)) + ((y13%2==0)&&(y13>0))==1)){
                        int yy1=y1, yy3=y3, yy13=y13;
                        if(yy1%2==0 && yy1>0){
                            yy1--;
                        }
                        else{
                            if(yy3%2==0 && yy3>0){
                                yy3--;
                            }
                            else{
                                if(yy13%2==0 && yy13>0){
                                    yy13--;
                                }
                            }
                        }
                        ok=1;
                        if(yy1==0 && yy3==0 && yy13==0 && ysp==0){
                            ok=-1;
                        }
                    }
                    else{
                        if((y1%2+y3%2+y13%2==1) && ((y1%2==0)+(y3%2==0)+(y13%2==0)==2)){
                            int yy1=y1, yy3=y3, yy13=y13;
                            if(yy1%2!=0){
                                yy1--;
                            }
```

```
                        else{
                            if(yy3%2!=0){
                                yy3--;
                            }
                            else{
                                if(yy13%2!=0){
                                    yy13--;
                                }
                            }
                        }
                        ok=1;
                        if(yy1==0 && yy3==0 && yy13==0 && ysp==0){
                            ok=-1;
                        }
                    }
                    else{
                        if((y1%2+y3%2+y13%2==2) && ((y1==0)+(y3==0)+(y13==0)==1)){
                            int yy1=y1, yy3=y3, yy13=y13;
                            if(yy1==0){
                                yy3--;
                                yy13--;
                            }
                            else{
                                if(yy3==0){
                                    yy1--;
                                    yy13--;
                                }
                                else{
                                    yy1--;
                                    yy3--;
                                }
                            }
                            ok=2;
                            if(yy1==0 && yy3==0 && yy13==0 && ysp==0){
                                ok=-1;
                            }
                        }
                    }
                }
            }
        }
    }
    fout<<ok<<" ";
    for(int i=2;i+K-1<=N;i++){
        if(vsp[i-1]==1)ysp--;
        if(w[i-1]==1)y1--;
        else if(w[i-1]==3)y3--;
        else if(w[i-1]==13)y13--;

        x=v[i+K-1];
        if(vsp[i+K-1]==1)ysp++;

        if(w[i+K-1]==1)y1++;
        else if(w[i+K-1]==3)y3++;
        else if(w[i+K-1]==13)y13++;

        ok=-1;
        if(y1==0 && y3==0 && y13==0 && ysp==0){
            ok=-1;
        }
        else{
```

```
if(y1%2!=0 && y3%2!=0 && y13%2!=0){
    ok=0;
}
else{
    if(y1%2==0 && y3%2==0 && y13%2==0){
        ok=0;
    }
    else{
        if((y1%2 + y3%2 + y13%2==2) &&
           (((y1%2==0)&&(y1>0))+((y3%2==0)&&(y3>0))+((y13%2==0)&&(y13>0))==1)){
            int yy1=y1, yy3=y3, yy13=y13;
            if(yy1%2==0 && yy1>0){
                yy1--;
            }
            else{
                if(yy3%2==0 && yy3>0){
                    yy3--;
                }
                else{
                    if(yy13%2==0 && yy13>0){
                        yy13--;
                    }
                }
            }
            ok=1;
            if(yy1==0 && yy3==0 && yy13==0 && ysp==0){
                ok=-1;
            }
        }
        else{
            if((y1%2+y3%2+y13%2==1) && ((y1%2==0)+(y3%2==0)+(y13%2==0)==2)){
                int yy1=y1, yy3=y3, yy13=y13;
                if(yy1%2!=0){
                    yy1--;
                }
                else{
                    if(yy3%2!=0){
                        yy3--;
                    }
                    else{
                        if(yy13%2!=0){
                            yy13--;
                        }
                    }
                }
                ok=1;
                if(yy1==0 && yy3==0 && yy13==0 && ysp==0){
                    ok=-1;
                }
            }
            else{
                if((y1%2+y3%2+y13%2==2) && ((y1==0)+(y3==0)+(y13==0)==1)){
                    int yy1=y1, yy3=y3, yy13=y13;
                    if(yy1==0){
                        yy3--;
                        yy13--;
                    }
                    else{
                        if(yy3==0){
                            yy1--;
                            yy13--;
```

```
                                    }
                                    else{
                                        yy1--;
                                        yy3--;
                                    }
                                }
                                ok=2;
                                if(yy1==0 && yy3==0 && yy13==0 && ysp==0){
                                    ok=-1;
                                }
                            }
                        }
                    }
                }
            }
            fout<<ok<<" ";
        }
    }
    return 0;
}
```

5.4 Problema Puzzle

Propusă de: prof. Dorin-Mircea Rotar, Colegiul Național „Samuil Vulcan" Beiuș

Maria a primit cadou de ziua ei o cutie cu piese de puzzle, etichetate cu numere naturale. Pentru a-l rezolva trebuie să lipească între ele, în ordinea în care le extrage din cutie, cât mai multe piese, formând astfel grupuri de piese.

Două piese de puzzle se pot lipi între ele dacă numerele de pe etichetele lor au cel puțin trei cifre comune. La operația de lipire a două piese se obține un grup de piese care va fi etichetat cu numărul obținut prin alipirea primelor patru cifre de pe prima etichetă cu ultimele patru cifre de pe cea de a doua etichetă (dacă numerele de pe etichete nu au cel puțin patru cifre se păstrează doar cele existente fără a adăuga altele). O piesă se poate lipi de o altă piesă sau de un grupul de piese anterior creat (dacă acesta există), sau poate forma singură un grup.

De exemplu dacă pe etichete avem numerele 133454 și 3523143 atunci putem lipi cele două piese deoarece au în comun cinci cifre (o cifră 1, două cifre 3, o cifră 4 și o cifră 5). În urma lipirii obținem un grup cu eticheta 13343143.

Pentru a rezolva jocul Maria extrage pe rând din cutie câte o piesă. Dacă aceasta se poate lipi de ultimul grup format atunci o lipește de el și actualizează eticheta grupului, altfel pune deoparte grupul respectiv și începe să formeze un nou grup pornind cu piesa extrasă.

După ce formează grupurile Maria alege etichete de grupuri în următoarea ordine: prima dată alege eticheta cu număr maxim de cifre distincte. Dacă sunt mai multe, o alege pe cea cu valoarea cea mai mică. Dintre etichetele rămase alege următoarea etichetă după aceeași regulă. Continuă procedeul până când a ales K etichete.

Cerințe

Cunoscând cele N numere naturale care se găsesc pe etichetele pieselor de joc, în ordinea în care aceste se extrag din cutie, să se determine:

1. Numărul de grupuri pe care le obține Maria după ce rezolvă jocul de puzzle;
2. Cele K numere înscrise pe etichetele grupurilor alese de Maria.

Date de intrare

În fișierul `puzzle.in` se găsesc pe prima linie, separate prin câte un spațiu, C, N și K, unde C reprezintă cerința ce trebuie rezolvată, N numărul de piese din joc, K numărul de valori cerute la cerința 2. Pe linia următoare, separate prin câte un spațiu, sunt cele N numere înscrise pe etichetele pieselor de joc, în ordinea în care se extrag din cutie.

Date de ieșire

Dacă cerința este 1, fișierul de ieșire `puzzle.out` va conține pe prima linie un număr natural G, reprezentând numărul de grupuri obținut după finalizarea jocului.

Dacă cerința este 2 pe prima linie a fișierului de ieșire `puzzle.out` se vor scrie, separate prin câte un spațiu, cele K numere înscrise pe etichetele grupurilor alese de Maria.

Restricții

- $1 \leq C \leq 2$

- $1 \leq N \leq 100\,000$
- $1 \leq K \leq 10$
- Numerele de pe etichetele pieselor sunt numere naturale cu cel mult nouă cifre;
- Pentru toate datele de test există soluție.

#	Puncte	Restricții
1	53	$C = 1$
2	7	$C = 2$, $K = 1$
3	13	$C = 2$, $K = 2$
4	27	$C = 2$, $K > 2$

Exemple

puzzle.in	puzzle.out	Explicații
1 6 1 13345 23143 4343 784532 432 7826	2	Piesa cu eticheta 13345 se poate lipi cu piesa cu eticheta 23143 obținând astfel grupul cu eticheta 13343143 la care se poate lipi piesa cu eticheta 4343 și obținem 13344343 care nu se mai poate lipi cu următoarea piesă cu eticheta 784532, dar această piesă se va lipi cu piesa etichetată cu 432 și obținem 7845432 care în continuare se lipește cu piesa 7826 și obținem a doua etichetă a unui grup 78457826.
2 6 1 13345 23143 4343 784532 432 7826	78457826	În urma grupării pieselor obținem două grupuri etichetate cu 13344343 și respectiv 78457826. Dintre acestea 78457826 are 6 cifre distincte, iar 13344343 are doar 3 cifre distincte. Deci valoarea căutată este 78457826.

5.5 Rezolvarea problemei Puzzle

În implementare soluției utilizarea vectorilor și a funcțiilor auxiliare ajută la gestionarea logică a unirii etichetelor și la verificarea compatibilității între piese, făcând codul modular și mai ușor de urmărit.

Cerința 1

Pentru prima cerință am folosit un vector de string în care rețin etichetele grupurile formate în joc. Se începe cu un grup inițial format din prima etichetă citită. Fiecare etichetă nouă citită este verificată dacă poate fi lipită la ultimul grup activ, bazat pe numărul de cifre comune. Dacă eticheta curentă se poate lipi, se actualizează eticheta grupului. Dacă nu, se începe un nou grup cu eticheta respectivă. La sfârșitul iterației prin etichete, numărul total de grupuri formate este înregistrat. Se afișează numărul de grupuri determinat.

Cerința 2

Pentru rezolvarea celei de-a doua cerințe am folosit un vector de perechi de numere întregi unde primul termen din pereche reprezintă numărul de cifre distincte a grupării, iar al doilea termen este eticheta grupării sub formă negativă. Etichetele grupurilor sunt sortate folosind sortarea prin selecție pentru primele K grupări după numărul de cifre distincte, iar în caz de egalitate, după valoarea numerică a etichetei, prioritate având cele cu valoarea mai mică. Se selectează primele K etichete conform ordinii stabilite. Se afișează etichetele selectate, separate prin spațiu.

5.6 Cod-sursă pentru problema Puzzle

```
#include <fstream>

using namespace std;
#define Nmax 100010
ifstream f("puzzle.in");
ofstream g("puzzle.out");
int C, N, K;
int v[Nmax], k, nr[Nmax];
int main()
{
    f >> C >> N >> K >> v[++k];
    for(int i = 2; i <= N; ++i)
    {
        int x, y, Y, frX[10] = {}, frY[10] = {}, cate = 0;
        f >> Y;
        x = v[k]; y = Y;
        do
            frX[x % 10]++, x /= 10;
        while(x);
        do
            frY[y % 10]++, y /= 10;
        while(y);
        for(int c = 0; c <= 9; ++c)
            cate += min(frX[c], frY[c]);
        if(cate >= 3)
        {
            int p = 10;
            if(Y > 9) p = 100;
            if(Y > 99) p = 1000;
            if(Y > 999) p = 10000;
            x = v[k];
            while(x > 9999)
                x /= 10;
            y = Y % p;
            v[k] = x * p + y;
        }
        else v[++k] = Y;
    }
    if(C == 1) g << k << '\n';
    else{
        for(int i = 1; i <= k; ++i)
        {
            int x = v[i], fr[10] = {};
            do
                fr[x % 10]++, x /= 10;
            while(x);
            for(int c = 0; c <= 9; ++c)
                if(fr[c]) nr[i]++;
```

```
        }
        for(int i = 1; i <= K; ++i)
        {
            int mx = nr[i], p = i, mn = v[i];
            for(int j = i + 1; j <= k; ++j)
                if(nr[j] > mx || (nr[j] == mx && v[j] < mn))
                    mx = nr[j], mn = v[j], p =j;
            swap(nr[i], nr[p]);
            swap(v[i], v[p]);
        }
        for(int i = 1; i <= K; ++i)
            g << v[i] << ' ';
        g << '\n';
    }
    return 0;
}
```

5.7 Problema Robinhood

Propusă de: prof. Marinel Șerban, Colegiul Național „Emil Racoviță" Iași

Robin Hood și Little John au hotărât să stabilească care dintre ei este cel mai bun arcaș. Pentru aceasta au construit n ținte așezate în linie dreaptă și numerotate de la 1 la n. Au stabilit apoi distanța de tragere. Cei doi se deplasează prin fața țintelor în linie dreaptă la distanța stabilită de comun acord.

Ei încearcă să atingă cu săgețile toate cele n ținte procedând în felul următor: Robin pleacă din dreptul țintei 1 și se deplasează până în dreptul țintei n, apoi se întoarce înapoi spre ținta 1 și așa mai departe... John pleacă din dreptul țintei n și se deplasează până la ținta 1, apoi se întoarce înapoi spre ținta n și așa mai departe... Fiecare dintre cei doi concurenți parcurge spațiul dintre două ținte consecutive într-o secundă. Robin trage o dată după fiecare p secunde, iar John trage o dată după fiecare q secunde, fiecare în ținta în dreptul căreia se află. Cei doi pot trage simultan în aceeași țintă sau într-una deja atinsă. Concursul se încheie în momentul în care fiecare țintă a fost atinsă cel puțin o dată.

Cerințe

1. Se cere să se determine timpul în care se termină concursul.
2. Care sunt țintele atinse exact o dată în timpul concursului?
3. Care sunt țintele atinse de cele mai multe ori în timpul concursului?

Date de intrare

Fișierul de intrare `robinhood.in` conține pe prima linie o valoare naturală C, reprezentând cerința. Pe linia a doua a fișierului de intrare se găsește un număr natural n, reprezentând numărul de ținte, iar pe linia a treia două numere naturale p q, separate printr-un spațiu, reprezentând intervalul de timp la care trag cei doi arcași.

Date de ieșire

Dacă cerința este 1, fișierul de ieșire `robinhood.out` conține pe prima linie un număr natural t, reprezentând timpul în care cei doi arcași ating toate țintele. Dacă cerința este 2 pe prima linie a fișierului de ieșire se vor afișa în ordine crescătoare, separate prin câte un spațiu, numerele de ordine ale țintelor atinse o singură dată. În cazul în care nici o țintă nu a fost atinsă exact o dată, se va afișa valoare 0. Dacă cerința este 3, pe prima linie a fișierului de ieșire se va afișa un număr natural reprezentând numărul maxim de săgeți care au atins o țintă, iar pe linia următoare se vor afișa în ordine crescătoare, separate prin câte un spațiu, numerele de ordine ale țintelor respective.

Restricții

- $1 \leq C \leq 3$
- $3 \leq n \leq 10\,000$
- $1 \leq p, q \leq 500$
- Pentru toate testele există soluție

#	Puncte	Restricții
1	53	$C = 1$
2	21	$C = 2$
3	26	$C = 3$

Exemple

robinhood.in	robinhood.out	Explicații
1 5 2 3	9	timp | 1 | 2 | 3 | 4 | 5 | ținta secunda 1 | | r | | j | | secunda 2 | | | $\underline{R}$ j | | | ținta 3 secunda 3 | | $\underline{J}$ | | r | | ținta 2 secunda 4 | j | | | | $\underline{R}$ | ținta 5 secunda 5 | | j | | r | | secunda 6 | | | $\underline{R}$ $\underline{J}$ | | | ținta 3 secunda 7 | | r | | j | | secunda 8 | $\underline{R}$ | | | | j | ținta 1 secunda 9 | | r | | $\underline{J}$ | | ținta 4 În acest tabel, literele r și j indică pozițiile lui Robin Hood și respectiv Little John, iar literele $\underline{R}$ și $\underline{J}$ indică momentele în care aceștia trag cu arcul.
2 5 2 3	1 2 4 5	Țintele care au fost atinse cu o singură săgeată sunt țintele 1, 2, 4 și 5.
3 5 2 3	3 3	Ținta 3 a fost atinsă de 3 ori: de 2 ori de Robin și o dată de John.

5.8 Rezolvarea problemei Robinhood

După citirea datelor se inițializează un vector *tinta*[] cu valoarea 0, indicând faptul că nici o țintă nu a fost atinsă. Se inițializează cu 0 alte două variabile *atinse* și *secunde*, care rețin câte ținte au fost atinse și, respectiv cât timp a trecut.

Două variabile $tinta\_robin = 1$ (Robin Hood pleacă de la ținta 1) și $tinta\_john = n$ (Little John de la ținta n) rețin numărul țintei în dreptul căreia se află la un moment dat fiecare dintre cei doi arcași. Alte două variabile $dir\_robin = true$ și $dir\_john = false$ indică direcțiile celor doi arcași.

Într-o structură repetitivă se simulează apoi concursul până când toate țintele au fost atinse (variabila *atinse* devine egală cu n).

Cerința 1

După terminarea simulării concursului, dacă cerința este 1 se afișează valoarea variabilei *secunde*.

Cerința 2

Dacă cerința este 2, se parcurge vectorul *tinta*[] contorizându-se valorile 1, apoi se afișează contorul.

Cerința 3

Dacă cerința este 3 se determină maximul elementelor vectorului *tinta*[], care se afișează, apoi, parcurgând din nou vectorul se afișează indicii elementelor egale cu maximul determinat.

5.9 Cod-sursă pentru problema Robinhood

```
#include <bits/stdc++.h>
using namespace std;

ifstream fin("robinhood.in");
ofstream fout("robinhood.out");

int tinta[10001], cerinta;
bool dir_robin, dir_john;
int i, n, p, q, tinta_robin, tinta_john, o_data, max_sageti, cate_max, atinse, secunde;

int main()
{
   fin >> cerinta >> n;                              //citire
   fin >> p >> q;
   for (i = 1; i <= n; ++i) tinta[i] = 0;         //toate tintele neatinse
   atinse = 0;
   secunde = 0;
   tinta_robin = 1;                               //robin pleaca de la 1
   tinta_john = n;                                //john de la n
   dir_robin = true;
   dir_john = false;                              //directiile celor 2
   do
   {
      secunde++;                                  //a mai trecut o secunda
      if (tinta_robin == n)                       //daca Robin a ajuns la tinta n
         dir_robin = false;                       //schimba sensul
      else if (tinta_robin == 1)                  //la fel daca a ajuns la tinta 1
         dir_robin = true;
      if (tinta_john == n)                        //la fel daca John a ajuns la tinta n
         dir_john = false;                        //schimba sensul
      else if (tinta_john == 1)                   //la fel daca a juns la tinta 1
         dir_john = true;
      if (dir_robin) ++tinta_robin;               //Robin merge spre dreapta
      else --tinta_robin;                         //Robin merge spre stanga
      if (dir_john) ++tinta_john;                 //John merge spre dreapta
      else --tinta_john;                          //John merge spre stanga
      if (secunde % p == 0)                       //au trecut p secunde?
      {
         if (tinta[tinta_robin] == 0) ++atinse;
         ++tinta[tinta_robin];                    //Robin trage in tinta la care este
      }
      if (secunde % q == 0)                       //au trecut q secunde?
```

```
      {
         if (tinta[tinta_john] == 0) ++atinse;
         ++tinta[tinta_john];                    //John trage in tinta la care este
      }
   }
   while (atinse < n);                           //concursul continua daca nu au fost
   if (cerinta == 1)                             //atinse toate tintele
   {
      fout << secunde << '\n';
      return 0;
   }
   if (cerinta == 2)                             //afisare cerinta 2
   {
      o_data = 0;                                //contor cate tinte au fost atinse o singura data
      for (i = 1; i <= n; ++i)
         if (tinta[i] == 1)                      //am gasit o tinta atinsa o data
         {
            ++o_data;                            //numar si
            fout << i << ' ';                    //afisez
         }
      fout << '\n';
      return 0;
   }
   if (cerinta == 3)                             //afisare cerinta 3
   {
      for (i = 1; i <= n; ++i)                   //caut numarul maxim de sageti intr-o tinta
         if (tinta[i] > max_sageti)
            max_sageti = tinta[i];               //numar sageti si
      fout << max_sageti << '\n';                //afisez cerinta 3
      for (i = 1; i <= n; ++i)
         if (tinta[i] == max_sageti)
            fout << i << ' ';
      fout << '\n';
      return 0;
   }
}
```

Capitolul 6

ONI 2024, clasa a VI-a

6.1 Problema Codificare

Propusă de: Ana Maria Arișanu, Colegiul Național „Mircea cel Bătrân" Râmnicu Vâlcea

În cadrul cercului de lingvistică Ioana a studiat diferite sisteme de codificare a mesajelor, însă i-a atras atenția *codificarea par-impar* care se aplică numerelor naturale. În această codificare fiecare cifră a unui număr crește cu valoarea 1 dacă cifra este pară, respectiv scade cu valoarea 1 dacă cifra este impară. Astfel, aplicând codificarea *par-impar* numărului 25467 se obține 34576, în timp ce numărului 123 îi corespunde 32 (conform regulii de codificare ar fi fost 032, însă un număr nenul nu poate începe cu 0).

Cerințe

1. Dându-se un șir de n numere naturale, să se determine cel mai mic și cel mai mare număr din șir care, prin codificarea *par-impar*, devin mai mari decât valorile lor inițiale.
2. Să se determine câte numere naturale de k cifre cu prima cifră *cif* devin palindrom prin codificarea *par-impar*.

Date de intrare

Fișierul de intrare `codificare.in` conține pe prima linie un număr natural C.

Dacă $C = 1$, a doua linie conține un număr natural n, iar a treia linie conține un șir de n numere naturale, separate prin câte un spațiu.

Dacă $C = 2$, a doua linie conține un număr natural k și o cifră *cif*, separate printr-un spațiu.

Date de ieșire

Dacă $C = 1$, fișierul de ieșire `codificare.out` va conține două valori separate printr-un spațiu, reprezentând cel mai mic și cel mai mare număr din șirul dat care, prin codificare, devin mai mari decât valorile lor inițiale.

Dacă $C = 2$, fișierul de ieșire `codificare.out` va conține un număr natural reprezentând numărul de numere naturale de k cifre cu prima cifră *cif* care devin palindrom prin codificare.

Restricții

- $C \in \{1, 2\}$

- $1 < n < 1\,000\,000$
- $1 < k < 10$
- $1 \leq cif < 10$
- Pentru $C = 1$, valorile din șir sunt numere naturale de maximum 9 cifre.
- Se garantează că cel puțin un număr din șir devine mai mare în urma codificării *par-impar*.

#	Puncte	Restricții
1	50	$C = 1$
2	50	$C = 2$

Exemple

codificare.in	codificare.out	Explicații
1 6 865 6 988 20 7 5	6 865	$C = 1$, se rezolvă cerința 1. Numerele de pe a treia linie care prin codificare devin mai mari sunt 865, 6 și 20. Cel mai mic dintre acestea este 6, iar cel mai mare este 865.
1 5 123 945 13 759 865	865 865	$C = 1$, se rezolvă cerința 1. Singurul număr de pe a treia linie care prin codificare devine mai mare este 865.
2 4 3	10	$C = 2$, se rezolvă cerința 2. Numerele de 4 cifre care au prima cifră 3 și care prin codificare devin numere palindrom sunt 3003, 3113, 3223, 3333, 3443, 3553, 3663, 3773, 3883 și 3993.

6.2 Rezolvarea problemei Codificare

Cerința 1

Soluția 1:

Pentru fiecare număr:

1. se codifică conform regulii din enunț
2. se compară numărul citit cu cel obținut prin codificare
3. dacă numărul obținut prin codificare este mai mare, numărul citit se ia în calcul pentru minim și maxim.

Soluția 2:

Se observă că un număr devine mai mare prin codificare doar dacă prima sa cifră este pară. Pentru fiecare număr:

1. se determină prima sa cifră
2. dacă prima cifră este pară, numărul citit se ia în calcul pentru minim și maxim.

În consecință, numărul de pași pe care-i efectuăm în total este $n \cdot 9$, unde 9 este numărul maxim de cifre.

Cerința 2:

Soluția 1:

Fiecare număr de k cifre care are prima cifră cif se codifică (numerele sunt de la $\overline{cif00\cdots00}$ la $\overline{cif99\cdots99}$). Dacă numărul obținut prin codificare este palindrom, se numără.

Soluția 2:

Fie $x = \overline{cifa_1\cdots a_{k-1}}$ un număr de k cifre ce are prima cifră cif.

Caz 1:

$cif \neq 1$

Numărul obținut prin codificare este palindrom dacă și numai dacă au loc egalitățile $a_{k-1} = cif$, $a_1 = a_{k-2}$, $a_2 = a_{k-3}$ și așa mai departe. Rezultatul este $10^{\lfloor (k-1)/2 \rfloor}$.

Caz 2:

$cif = 1$

Când $k = 2$, rezultatul este 10, numerele fiind $10, 11, 12, \ldots, 19$.

Când $k = 3$, rezultatul este 19, numerele fiind $100, 111, 122, \ldots, 199, 110,\ 112, 113, \ldots, 119$.

Când $k = 4$, rezultatul este 109, numerele fiind de forma $\overline{111a}$ (10 numere), $\overline{11aa}$ (cu $a \neq 1$, 9 numere) și $\overline{1aba}$ (cu $a \neq 1$, 90 de numere).

Când $k = 5$, rezultatul este 199, numerele fiind de forma $\overline{1111a}$ (10 numere), $\overline{111aa}$ (cu $a \neq 1$, 9 numere), $\overline{11aba}$ (cu $a \neq 1$, 90 de numere) și $\overline{1abba}$ (cu $a \neq 1$, 90 de numere).

Când $k = 6$, rezultatul este 1099, numerele fiind de forma $\overline{11111a}$ (10 numere), $\overline{1111aa}$ (cu $a \neq 1$, 9 numere), $\overline{111aba}$ (cu $a \neq 1$, 90 de numere), $\overline{11abba}$ (cu $a \neq 1$, 90 de numere) și $\overline{1abcba}$ (cu $a \neq 1$, 900 de numere).

Continuând raționamentul, rezultatul, în acest caz, este

$$\begin{cases} 2 \cdot 10^{\lfloor k/2 \rfloor} - 1 & \text{pentru } k \text{ impar,} \\ 10^{\lfloor k/2 \rfloor} + 10^{\lfloor (k-1)/2 \rfloor} - 1 & \text{pentru } k \text{ par.} \end{cases}$$

În consecință, numărul de pași pe care-i efectuăm în total este $\lfloor k/2 \rfloor$.

6.3 Cod-sursă pentru problema Codificare

```
#include <fstream>
using namespace std;
ifstream f("codificare.in");
ofstream g("codificare.out");
int prima_cifra(int n)
{
  while (n>9) n=n/10;
  return n;
}

int main()
{ int p, minn=1000000000, maxx=-1,n,x;
  f>>p;
```

```
  if (p==1)
     {
      f>>n;
      for (int i=1; i<=n;i++)
          {
           f>>x;
           if (prima_cifra(x)%2==0)
              {
               if (x<minn) minn=x;
               if (x>maxx) maxx=x;
              }
          }
      g<<minn<<' '<<maxx;
     }
     else
     {
      int k, c, i;
      long long p10=1;
      f>>k>>c;
      if (c!=1)
         {
          for (i=1; i<=(k-1)/2; i++) p10=p10*10;
          g<<p10;
         }
         else
         {
          for (i=1; i<=k/2; i++) p10=p10*10;
          if (k%2==0)
              g<<p10+p10/10-1;
             else
              g<<p10+p10-1;
         }
      }
}
```

6.4 Problema Esm

Propusă de: prof. Dan Pracsiu, Liceul Teoretic „Emil Racoviță" Vaslui

Spunem că o secvență de numere $(a_i, a_{i+1}, \ldots, a_j)$ este **esm** dacă:

- are cel puțin 3 elemente
- există cel puțin două numere a_x și a_y în acea secvență, cu $i \leq x < y < j$, astfel încât $a_x \cdot a_y = a_j$

De exemplu, secvența $(54, 7, 22, 6, 9, 42)$ este **esm** deoarece $7 \cdot 6 = 42$.

Cerințe

Se dă un șir $a_1, a_2, \ldots, a_n$ de numere naturale. Să se determine:

1. Numărul de secvențe **esm** din șir de lungime 3.
2. Numărul de secvențe **esm** din șir care se termină cu a_n.
3. Numărul de secvențe **esm** din șir.

Date de intrare

Fișierul de intrare `esm.in` conține pe prima linie un număr natural C, pe a doua linie numărul natural n, iar pe a treia linie, separate prin câte un spațiu, elementele șirului $a_1, a_2, \ldots, a_n$.

Date de ieșire

Fișierul de ieșire `esm.out` va conține un singur număr natural X:

- Dacă $C = 1$, atunci X va fi numărul de secvențe **esm** din șir de lungime 3.
- Dacă $C = 2$, atunci X va fi numărul de secvențe **esm** din șir care se termină cu a_n.
- Dacă $C = 3$, atunci X va fi numărul de secvențe **esm** din șir.

Restricții

- $C \in \{1, 2, 3\}$
- $3 \leq n \leq 100\,000$
- $1 \leq a_i \leq 100\,000$, pentru orice $i \in \{1, 2, \ldots, n\}$
- Lungimea unei secvențe este egală cu numărul de elemente din secvență.

#	Puncte	Restricții
1	30	$C = 1$
2	30	$C = 2$
3	40	$C = 3$

Exemple

esm.in	esm.out	Explicații
1 8 2 3 6 18 1 18 3 5	3	Secvențele **esm** din șir de lungime 3 sunt: $(2, 3, 6)$; $(3, 6, 18)$; $(18, 1, 18)$.
2 8 5 8 20 2 4 7 5 40	3	Secvențele **esm** din șir care se termină cu 40 sunt: $(5, 8, 20, 2, 4, 7, 5, 40)$; $(8, 20, 2, 4, 7, 5, 40)$; $(20, 2, 4, 7, 5, 40)$.
3 8 2 2 4 8 1 8 16 7	9	Secvențele **esm** din șir sunt: $(2, 2, 4)$; $(2, 2, 4, 8)$; $(2, 4, 8)$; $(2, 2, 4, 8, 1, 8)$; $(2, 4, 8, 1, 8)$; $(4, 8, 1, 8)$; $(8, 1, 8)$; $(2, 2, 4, 8, 1, 8, 16)$; $(2, 4, 8, 1, 8, 16)$.

6.5 Rezolvarea problemei Esm

Cerința 1

Se parcurge șirul și, pentru orice secvență de lungime 3 de forma (a_i, a_{i+1}, a_{i+2}), se verifică dacă $a_i \cdot a_{i+1} = a_{i+2}$.

Cerința 2

Considerăm $x = a_n$.

Deoarece numerele din șir sunt mai mici sau egale cu 100 000, construim un vector poz de lungime 100 000 în care, pentru fiecare $i \in \{1, 2, \ldots, 100\,000\}$, $poz_i = j$ are semnificația „cea mai din dreapta poziție unde apare numărul i în secvența $(a_1, a_2, \ldots, a_{n-1})$ este la poziția j”. Dacă i nu apare în șir, atunci $poz_i = 0$.

Vom determina divizorii lui x și, pentru toate perechile de divizori (u, v) cu proprietatea că $u \cdot v = x$, aflăm cea mai din dreapta poziție P cu proprietatea că există o pereche (d_1, d_2) de divizori ai lui x astfel încât $poz_{d_1} \geq P$ și $poz_{d_2} \geq P$. De exemplu, dacă avem $n = 10$, $a = (7, 2, 3, 12, 8, 5, 4, 7, 6, 24)$, atunci $x = 24$ și $P = 7$, iar divizorii pereche ce au pozițiile după P sunt 4 și 6.

De aici obținem că numărul de secvențe esm de forma $(a_i, a_{i+1}, \ldots, a_n)$ este egal cu P. Aceste secvențe sunt: $(a_1, a_2, \ldots, a_n), (a_2, a_3, \ldots, a_n), \ldots, (a_P, a_{P+1}, \ldots, a_n)$. Algoritmul are $n - 1$ pași pentru construirea vectorului poz, la care se adaugă $\sqrt{x}$ pași pentru determinarea divizorilor pereche ai lui x.

Observație importantă:

Este posibil ca x să fie pătrat perfect, caz în care avem perechea de divizori (r, r) a lui x, unde $r = \sqrt{x}$. Deci va trebui să reținem ultimele două poziții ale lui r, nu doar cea mai din dreapta.

Cerința 3

Pentru determinarea tuturor secvențelor din șir care sunt esm vom proceda asemănător soluției de la cerința 2, în care, pentru fiecare $i \in \{3, \ldots, n\}$, trebuie să aflăm câte secvențe esm se termină cu $x = a_i$. Construim vectorul poz cu secvența $(a_1, a_2, \ldots, a_{i-1})$ și aflăm divizorii pereche ai lui a_i. Apoi, pentru a trece la aflarea numărului de secvențe care se termină cu a_{i+1}, actualizăm $poz_{a_i} = i$ (adică îl introducem pe a_i în poz) și aflăm câte secvențe esm se termină cu $x = a_{i+1}$.

În consecință, numărul de pași pe care-i efectuăm în total este $n \cdot \sqrt{\text{MAX}}$, unde MAX este valoarea maximă din șirul a.

6.6 Cod-sursă pentru problema Esm

```
#include <bits/stdc++.h>
using namespace std;
ifstream  fin("esm.in");
ofstream fout("esm.out");

int a[100003], n, P;
int poz1[100003], poz2[100003];

int main()
{
    fin >> P >> n;
    for (int i = 1; i <= n; i++) fin >> a[i];
    if (P == 1)
    {
        int i, cnt = 0;
        for (i = 3; i <= n; i++)
            if (1LL * a[i - 2] * a[i - 1] == a[i]) cnt++;
        fout << cnt << "\n";
    }
    else if (P == 2)
    {
        int i, x, p;
        for (i = 1; i < n; i++)
        {
            x = a[i];
            if (poz1[x] == 0) poz1[x] = i;
            else
            {
                poz2[x] = poz1[x];
                poz1[x] = i;
            }
        }
        x = a[n]; p = 0;
        for (i = 1; i * i < x; i++)
            if (x % i == 0)
            {
                if (poz1[i] > 0 && poz1[x / i] > 0)
                    p = max(p, min(poz1[i], poz1[x / i]));
            }
        if (i * i == x)
```

```
        {
            if (poz1[i] > 0 && poz2[i] > 0)
                p = max(p, poz2[i]);
        }
        fout << p << "\n";
    }
    else
    {
        int i, j, p, x;
        long long cnt = 0;
        poz1[a[2]] = 2;
        if (a[1] == a[2]) poz2[a[1]] = 1;
        else poz1[a[1]] = 1;
        for (j = 3; j <= n; j++)
        {
            x = a[j];
            p = 0;
            for (i = 1; i * i < x; i++)
                if (x % i == 0)
                {
                    if (poz1[i] > 0 && poz1[x / i] > 0)
                        p = max(p, min(poz1[i], poz1[x / i]));
                }
            if (i * i == x)
            {
                if (poz1[i] > 0 && poz2[i] > 0)
                    p = max(p, poz2[i]);
            }
            cnt += p;
            /// pun pe x=a[j] in vectorii de frecventa
            if (poz1[x] == 0) poz1[x] = j;
            else
            {
                poz2[x] = poz1[x];
                poz1[x] = j;
            }
        }
        fout << cnt << "\n";
    }
    return 0;
}
```

6.7 Problema Sim

Propusă de: stud. Iulian Oleniuc, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza" Iași

Paftenie trăiește într-un oraș pătratic, împărțit în $n \times n$ regiuni pătratice, așezate pe n linii, numerotate de la 1 la n, și n coloane, numerotate de la 1 la n. În fiecare astfel de regiune există o cafenea, iar în dreptul fiecărei cafenele se află un indicator către nord ($\uparrow$), sud ($\downarrow$), vest ($\leftarrow$) sau est ($\rightarrow$). Prin „cafeneaua (i, j)" ne vom referi la cafeneaua de la linia i și coloana j.

În prima zi, în fiecare cafenea își ia micul dejun exact unul dintre cei n^2 cetățeni ai orașului. Începând cu a doua zi, fiecare cetățean își va lua micul dejun în cafeneaua vecină celei în care și-a luat micul dejun în ziua precedentă, conform direcției de pe indicatorul asociat acesteia. Procesul se repetă timp de k zile.

Pornind într-o zi de la cafeneaua (i, j), cafeneaua vecină la care se ajunge în ziua următoare este $(i-1, j)$ dacă direcția indicatorului este *nord* sau $(i+1, j)$ dacă direcția indicatorului este *sud* sau $(i, j-1)$ dacă direcția indicatorului este *vest* sau $(i, j+1)$ dacă direcția indicatorului este *est*.

Paftenie definește *gradul de fericire al unui cetățean* ca fiind numărul de cetățeni cu care a luat micul dejun împreună cel puțin o dată în timpul celor k zile. Mai mult, Paftenie definește *gradul de fericire al orașului* drept suma gradelor de fericire ale cetățenilor săi.

Cerințe

Cum Paftenie este prea distras de micul său dejun englezesc cu cârnăciori și fasole fiartă, apelează la voi pentru a determina:

1. Numărul maxim de cetățeni care iau micul dejun împreună în cea de-a doua zi.
2. Gradul de fericire al orașului său.

Date de intrare

Fișierul de intrare `sim.in` conține pe prima linie un număr natural C, iar pe a doua linie numerele naturale n și k, separate printr-un spațiu. Următoarele n linii din fișier conțin câte n numere naturale din mulțimea $\{1, 2, 3, 4\}$ (1 pentru *nord*, 2 pentru *sud*, 3 pentru *vest*, 4 pentru *est*), separate prin câte un spațiu. Al j-lea număr de pe linia $i + 2$ din fișier reprezintă direcția indicatorului asociat cafenelei (i, j).

Date de ieșire

Fișierul de ieșire `sim.out` va conține un singur număr natural X:

1. Dacă $C = 1$, atunci X va fi numărul maxim de cetățeni care iau micul dejun împreună în cea de-a doua zi.
2. Dacă $C = 2$, atunci X va fi gradul de fericire al orașului lui Paftenie.

Restricții

- $C \in \{1, 2\}$
- $2 \leq n \leq 1\,000$
- $k \in \{1\,000, 1\,000\,000\,000\}$
- Se garantează că nu există niciun indicator care, odată urmat, duce la părăsirea orașului.

#	Puncte	Restricții
1	20	$C = 1$, $n \leq 30$, $k = 1\,000$
2	30	$C = 2$, $n \leq 30$, $k = 1\,000$
3	20	$C = 2$, $n \leq 30$, $k = 1\,000\,000\,000$
4	30	$C = 2$, $30 < n \leq 1\,000$, $k = 1\,000\,000\,000$

Exemple

sim.in	sim.out	Explicații
1 3 1000 4 3 2 4 2 2 4 3 3	3	Orașul este reprezentat grafic în tabloul bidimensional (matricea) A, iar cetățenii săi sunt numerotați precum în matricea A_1. În fiecare celulă a matricei A_i, cu $1 \leq i \leq 2$, sunt cetățenii care, în ziua i, iau micul dejun în cafeneaua respectivă. Numărul maxim de cetățeni care iau masa împreună în a doua zi este 3, aceștia fiind situați în cafeneaua $(3, 2)$.
2 5 1000 4 3 2 4 2 4 2 2 3 2 1 4 4 2 1 1 4 1 2 1 1 3 3 3 1	30	Orașul este reprezentat grafic în tabloul bidimensional (matricea) B, iar cetățenii săi sunt numerotați precum în matricea B_1. În fiecare celulă a matricei B_i, cu $1 \leq i \leq 4$, sunt cetățenii care, în ziua i, iau micul dejun în cafeneaua respectivă. Tabelul T conține întâlnirile care au loc până la finalul celor 1000 de zile. A doua coloană conține lista cetățenilor cu care se întâlnește cetățeanul din prima coloană. Răspunsul este $3+2+2+3+2+3+2+2+2+3+2+2+2 = 30$.

Explicații

$A =$

→	←	↓
→	↓	↓
→	←	←

$A_1 =$

1	2	3
4	5	6
7	8	9

$A_2 =$

2	1	
	4	3
8	5, 7, 9	6

$B =$

→	←	↓	→	↓
→	↓	↓	←	↓
↑	→	→	↓	↑
↑	→	↑	↓	↑
↑	←	←	←	↑

$B_1 =$

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

$B_2 =$

2	1			4
11	6	3, 9		5, 15
16	7	8, 12, 18	13	10, 20
21		17	14	25
22	23	24	19	

$B_3 =$

1	2			
16	11			4, 10, 20
21	6	3, 7, 9, 17	8, 12, 18	5, 15, 25
22			13	
23	24	19	14	

$B_4 =$

2	1			
21	16			5, 15, 25
22	11	6	3, 7, 9, 17	4, 10, 20
23			8, 12, 18	
24	19	14	13	

$T =$

3	$7, 9, 17$
4	$10, 20$
5	$15, 25$
7	$3, 9, 17$
8	$12, 18$
9	$3, 7, 17$
10	$4, 20$
12	$8, 18$
15	$5, 25$
17	$3, 7, 9$
18	$8, 12$
20	$4, 10$
25	$5, 15$

6.8 Rezolvarea problemei Sim

Cerința 1

Reținem valorile citite într-o matrice *arrow*. Prima cerință presupune să construim o a doua matrice *count*, astfel încât în $count[x][y]$ să reținem numărul de cetățeni care în a doua zi iau micul dejun în cafeneaua (x, y). Parcurgem matricea *arrow* dată și, pentru fiecare poziție (x_1, y_1), calculăm în (x_2, y_2) poziția unde se ajunge din (x_1, y_1) urmând indicatorul $arrow[x_1][y_1]$, după care incrementăm valoarea $count[x_2][y_2]$. La final, calculăm valoarea maximă din matricea *count*.

Complexitatea algoritmului este $\mathcal{O}(n^2)$.

Cerința 2 – Subtask 1

Pentru $n \leq 30$ și $k = 10^3$, este suficient să efectuăm o simulare brută a celor k zile. În acest sens, vom folosi două matrice $count_C$ și $count_P$, în care vom reține câți cetățeni iau micul dejun în fiecare cafenea în ziua curentă și, respectiv, în ziua precedentă.

Pentru prima zi, toate celulele matricei $count_P$ vor fi inițializate cu valoarea 1. La începutul fiecărei noi zile, conținutul matricei $count_C$ este copiat în $count_P$, după care $count_C$ este reinițializată cu zerouri.

Calculul lui $count_C$ în funcție de $count_P$ este similar procedeului de la prima cerință. Valoarea $count_P[x_1][y_1]$ este adăugată la $count_C[x_2][y_2]$, dar nu înainte de a adăuga la răspunsul final valoarea

$$2 \cdot count_C[x_2][y_2] \cdot count_P[x_1][y_1].$$

Această operație semnifică prima întâlnire dintre fiecare cetățean deja „ajuns” în (x_2, y_2) și fiecare cetățean ce urmează să „ajungă” în (x_2, y_2) din (x_1, y_1). Constanta 2 arată că fiecare întâlnire se contorizează de două ori – o dată pentru gradele de fericire ale celor din (x_2, y_2) și o dată pentru ale celor din (x_1, y_1).

De menționat că, pentru o implementare mai facilă, se poate recurge la tablouri tridimensionale; pentru acest subtask, limita de memorie o permite. Astfel, în loc de două matrice, putem folosi un tablou în care prima dimensiune să fie egală cu k, deci în care $count[i]$ să reprezinte matricea aferentă zilei i.

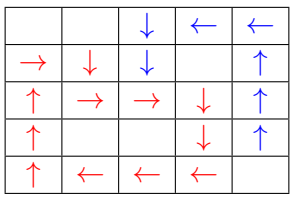
Complexitatea algoritmului este $\mathcal{O}(k \cdot n^2)$.

Cerința 2 – Subtask 2

În primul rând, observăm că, de la un punct încolo, traseul oricărui cetățean se va repeta, deoarece acesta va efectua o infinitate de pași, în timp ce orașul are un număr finit de cafenele. Mai mult, putem

fi siguri că, în a n^2-a zi, nu va mai exista niciun cetățean care să nu fi intrat deja în partea repetitivă (în continuare, o vom numi „ciclu") a traseului său, deoarece lungimea maximă a acestuia este n^2.

Spre exemplu, în matricea de mai jos, traseul cetățeanului $(4, 5)$ este compus dintr-un prefix nerepetitiv (albastru) și un ciclu (roșu).



Începând (cel târziu) cu a n^2-a zi, nu se va mai produce nicio întâlnire nouă între doi cetățeni, din moment ce amândoi se află fie în cicluri diferite, fie pe poziții diferite din același ciclu. Nu în ultimul rând, trebuie menționat că, odată ce doi cetățeni se întâlnesc într-o cafenea anume, aceștia nu se vor mai despărți niciodată, deoarece, din acel punct încolo, ei vor urma exact același traseu.

În concluzie, este de ajuns să analizăm configurația orașului în ziua n^2. Dacă, în acea zi, într-o cafenea anume se află k cetățeni, înseamnă că gradul de fericire al fiecăruia dintre ei este $k - 1$, deoarece s-a întâlnit, de-a lungul timpului, doar cu ceilalți $k - 1$ cetățeni. Așadar, cafeneaua respectivă contribuie cu $k \cdot (k - 1)$ la răspunsul final.

Restricția $n \leq 30$ ne permite să implementăm o soluție în complexitate $\mathcal{O}(n^4)$, în care simulăm, pentru fiecare cetățean în parte, traseul său pe parcursul celor n^2 zile, determinându-i poziția finală.

O soluție alternativă, tot în complexitate $\mathcal{O}(n^4)$, dar în practică mult mai rapidă, presupune simularea fiecărei zile ca la primul subtask, până când procesăm o zi în care nu s-a produs nicio întâlnire nouă. Corectitudinea soluției se bazează pe faptul că ultima zi în care se produce o astfel de întâlnire este cea în care, pentru ultima oară, un cetățean intră în partea ciclică a traseului său, unde sigur dă de altcineva. Această soluție obține 80 de puncte.

Cerința 2 – Subtask 3

La finalul simulării, singurele celule din matrice care ne interesează sunt cele care fac parte dintr-un ciclu. În ordinea descoperirii acestor cicluri, putem eticheta celulele respective cu numere strict pozitive, de la 1, pe rând, până la cel mult n^2.

Așadar, ne dorim să determinăm, pentru fiecare cetățean (x, y), care este eticheta $e(x, y)$ a cafenelei în care se va afla acesta în ziua n^2. După ce vom face rost de această informație, nu va mai trebui decât să calculăm frecvențele etichetelor în matricea e de $n \times n$ etichete.

Completarea matricei e se efectuează printr-un proces similar celui de la al doilea subtask, dar optimizat. În acest sens, nu vom simula traseul unui cetățean decât până în momentul în care vom ajunge într-o celulă deja vizitată, fie de către el însuși (caz în care am identificat un ciclu nou), fie de către cineva dinaintea lui.

În cazul în care am găsit un ciclu nou, cu etichete de la l până la r, vom determina mai întâi valorile $e(x, y)$ pentru celulele sale, printr-o simplă formulă bazată pe l, r și eticheta inițială a celulei (x, y).

Mai departe, parcurgem în sens invers prefixul nerepetitiv al traseului găsit. Dacă am ajuns (în acest sens *invers*) în (x_1, y_1) din (x_2, y_2), iar (x_2, y_2) face parte dintr-un traseu al cărui ciclu este etichetat cu valori de la l la r, atunci

$$e(x_1, y_1) = \begin{cases} r & \text{pentru } e(x_2, y_2) = l, \\ e(x_2, y_2) - 1 & \text{altfel.} \end{cases}$$

Complexitatea soluției devine $\mathcal{O}(n^2)$, căci fiecare celulă din matrice este vizitată o singură dată. De remarcat că, pentru punctajul maxim, răspunsul trebuie reținut pe 64 de biți.

6.9 Cod-sursă pentru problema Sim

```cpp
#include <bits/stdc++.h>
using namespace std;

ifstream fin("sim.in");
ofstream fout("sim.out");

const int NMAX = 1000;

int c, n, k;
int arrows[NMAX][NMAX];

int max_count;
int counts_second_day[NMAX][NMAX];

int64_t answer;
int labels[NMAX][NMAX];

int cycle_count;
int label_count;
int cycle_start_label[NMAX * NMAX + 1];
int cycle_final_label[NMAX * NMAX + 1];
int cycle_of_label[NMAX * NMAX + 1];
int count_of_label[NMAX * NMAX + 1];

int path_length;
int path_x[NMAX * NMAX];
int path_y[NMAX * NMAX];

int main() {
    fin >> c;
    fin >> n >> k;
    for (int x = 0; x < n; x++)
        for (int y = 0; y < n; y++)
            fin >> arrows[x][y];

    if (c == 1) {
        for (int x1 = 0; x1 < n; x1++)
            for (int y1 = 0; y1 < n; y1++) {
                int x2 = x1, y2 = y1;
                if (arrows[x1][y1] == 1) x2--; else
                if (arrows[x1][y1] == 2) x2++; else
                if (arrows[x1][y1] == 3) y2--; else
                if (arrows[x1][y1] == 4) y2++;
                counts_second_day[x2][y2]++;
            }
        for (int x = 0; x < n; x++)
            for (int y = 0; y < n; y++)
                max_count = max(max_count, counts_second_day[x][y]);
        fout << max_count << '\n';
        return 0;
    }

    for (int x = 0; x < n; x++)
        for (int y = 0; y < n; y++) {
            path_length = 0;
            int x1 = x, y1 = y;
            while (labels[x1][y1] == 0) {
                path_x[path_length] = x1;
                path_y[path_length] = y1;
```

```
                path_length++;
                labels[x1][y1] = -path_length;
                if (arrows[x1][y1] == 1) x1--; else
                if (arrows[x1][y1] == 2) x1++; else
                if (arrows[x1][y1] == 3) y1--; else
                if (arrows[x1][y1] == 4) y1++;
            }

            if (labels[x1][y1] < 0) {
                cycle_count++;
                const int cycle_start_index = -labels[x1][y1] - 1;
                const int cycle_length = path_length - cycle_start_index;
                cycle_start_label[cycle_count] = label_count + 1;
                cycle_final_label[cycle_count] = label_count + cycle_length;
                for (int i = 1; i <= cycle_length; i++)
                    cycle_of_label[label_count + i] = cycle_count;

                for (int i = cycle_start_index; i < path_length; i++) {
                    const int x = path_x[i];
                    const int y = path_y[i];
                    label_count++;

                    const int start_label = cycle_start_label[cycle_count];
                    const int label_index = label_count - start_label;
                    const int final_label = (label_index + n * n) % cycle_length + start_label;

                    labels[x][y] = final_label;
                    count_of_label[labels[x][y]]++;
                }
                path_length -= cycle_length;
            }

            for (int i = path_length - 1; i >= 0; i--) {
                const int x1 = path_x[i];
                const int y1 = path_y[i];
                int x2 = x1;
                int y2 = y1;
                if (arrows[x2][y2] == 1) x2--; else
                if (arrows[x2][y2] == 2) x2++; else
                if (arrows[x2][y2] == 3) y2--; else
                if (arrows[x2][y2] == 4) y2++;

                const int label = labels[x2][y2];
                const int cycle = cycle_of_label[label];
                const int start_label = cycle_start_label[cycle];
                const int final_label = cycle_final_label[cycle];
                const int previous_label = label == start_label ? final_label : label - 1;

                labels[x1][y1] = previous_label;
                count_of_label[labels[x1][y1]]++;
            }
        }

    for (int i = 1; i <= label_count; i++)
        answer += 1LL * count_of_label[i] * (count_of_label[i] - 1);
    fout << answer << '\n';
    return 0;
}
```

Capitolul 7

ONI 2024, clasa a VII-a

7.1 Problema Expresie

Propusă de: prof. Nistor Moț, Școala „Dr. Luca" Brăila

Se consideră un număr natural format din n cifre. Inserând între cifrele numărului dat p operatori $+$ și q operatori $-$ se obține o expresie aritmetică. Un operator poate fi inserat doar între două cifre, deci înaintea primei cifre a numărului nu se poate plasa un operator.

Cerințe

Scrieți un program care, pentru un număr dat, determină valoarea maximă a unei expresii aritmetice care se poate obține inserând p operatori $+$ și q operatori $-$ între cifrele numărului dat.

Date de intrare

Fișierul de intrare `expresie.in` conține pe prima linie numerele naturale n p q separate prin câte un spațiu, cu semnificația din enunț. Pe cea de-a doua linie se află un număr format din n cifre.

Date de ieșire

Fișierul de ieșire `expresie.out` va conține o singură linie, pe care va fi scrisă valoarea maximă a unei expresii aritmetice care se poate obține prin inserarea a p operatori $+$ și a q operatori $-$ între cifrele numărului dat.

Restricții

- $2 \leq n \leq 1\,000$
- $0 < p + q < n$
- Numărul citit nu începe cu 0.

#	Puncte	Restricții
1	6	$n \leq 18$ și $p + q = 1$
2	9	$n > 18$ și $p + q = 1$
3	25	Rezultatul are cel mult 18 cifre și $p + q > 1$
4	60	Nu există restricții suplimentare

Exemple

expresie.in	expresie.out	Explicații
5 1 2 54321	54	$54+3-2-1=54$

7.2 Rezolvarea problemei Expresie

Expresia va conține $p+q+1$ termeni, iar pentru a obține rezultatul maxim va trebui să facem termenii care se adună cât mai mari, iar termenii care se scad cât mai mici.
Prima idee clară este că vom forma cei q termeni care se scad din câte o cifră. Dar termenii care se adună? Mai puțin intuitiv este faptul că și la acești termeni vom urma aceeași idee: toți vor avea o singură cifră, exceptând unul care va fi cel mai mare termen posibil. Pentru justificare analizăm câteva cazuri particulare:

- Dacă $p+q=n-1$ toți termenii vor avea o singură cifră; vom pune semnul – în fața celor mai mici q cifre (exceptând prima cifră) și semnul + în fața celorlalte cifre.
- Dacă $p=0$ și $q=1$, vom scădea ultima cifră din numărul format de primele $n-1$ cifre.
- Asemănător procedăm pentru toate cazurile când $p=0$: scădem ultimele q cifre din numărul format din primele $n-q$ cifre.
- Dacă avem $p=1$ și $q=0$, rezultatul maxim se obține adunând prima sau ultima cifră la numărul format din celelalte cifre. Să justificăm pentru un număr de 6 cifre ($abcdef$), dar raționamentul este valabil pentru orice număr de cifre $sn>3$): $\overline{abcd}+\overline{ef} < \overline{abcde}+f$ (pentru că $\overline{abcd}+(10\cdot e+f) < 10\cdot\overline{abcd}+e+f$, sau $9\cdot e<9\cdot\overline{abcd}$).
- Asemănător procedăm pentru toate cazurile când $q=0$; adunăm la cel mai mare număr format din $n-p$ cifre consecutive celelalte cifre.

Algoritmul general: căutăm cel mai mare număr format din $n-p-q$ cifre situate pe poziții consecutive din care scădem cele mai mici q cifre dintre cele rămase (excluzând-o pe prima) și adunăm celelalte p cifre.
Dacă $n-p-q<18$ se pot face calcule cu numere întregi pe 64 biți, dacă nu, trebuie implementate operațiile de adunare și scădere pentru numere mari, deoarece limbajul nu are (încă) un tip de date pentru numere cu până la 1000 cifre.

7.3 Cod-sursă pentru problema Expresie

```
#include <fstream>
#include <algorithm>
#define LGMAX 1002

using namespace std;
ifstream fin("expresie.in");
ofstream fout("expresie.out");
int n, p, q, lg, suma, lgsum, lgrez;
char s[LGMAX];
char a[LGMAX];

typedef int NrMare[LGMAX];
NrMare secv, sum, rez;

void adun(NrMare x, int lgx, NrMare y, int lgy, NrMare z, int& lgz);
void dif(NrMare x, int lgx, NrMare y, int lgy, NrMare z, int& lgz);
void afisare(NrMare x, int lgx);
```

```
int compars(int i, int j);
int comparnr(NrMare x, int lgx, NrMare y, int lgy);

int main()
{int i, j, nr, imax, k, semn;
 fin>>n>>p>>q>>s;
 lg=n-p-q;
 //determin secventa de maxima lexicografic
 imax=0;
 if (p>0)
    {for (i=1; i<n-lg+1; i++)
         if (compars(i,imax)>0) imax=i;
    }
 //cifrele de secventa maxima sunt pe pozitiile imax..imax+lg-1
 nr=0;
 if (imax==0) //secventa este la inceput
    {for (j=imax+lg; j<n; j++) a[nr++]=s[j]; suma=0;}
    else
    {
     p--; //plus se pune inaintea acestei secvente
     suma=s[0]-'0';
     for (j=1; j<imax; j++) a[nr++]=s[j];
     for (j=imax+lg; j<n; j++) a[nr++]=s[j];
    }

 //sortez celelalte cifre
 sort(a,a+nr);
 //primele q vor fi cu minus, ultimele p cu plus
 for (i=0; i<q; i++) suma=suma-(a[i]-'0');
 for (i=q; i<nr; i++) suma=suma+(a[i]-'0');
 if (suma<0) {suma=-suma; semn=-1;}
    else semn=1;
 //adunam sum cu secventa maxima
 for (k=0, j=imax+lg-1; j>=imax; j--, k++) secv[k]=s[j]-'0';
 do
   {sum[lgsum++]=suma%10; suma/=10; }
 while (suma);
 if (semn==1)
    adun(secv, lg, sum, lgsum, rez, lgrez);
    else
    if (comparnr(secv,lg,sum,lgsum)>=0) {dif(secv,lg,sum,lgsum, rez, lgrez); semn=1;}
       else
       dif(sum,lgsum,secv,lg,rez,lgrez);
 if (semn==-1) fout<<'-';
 afisare(rez, lgrez);
 return 0;
}

void adun(NrMare x, int lgx, NrMare y, int lgy, NrMare z, int& lgz)
{int t, i, val;
 //cel mai scurt dintre numere il completez cu zerouri nesemnificative
 if (lgx<lgy) {lgz=lgy; for (i=lgx; i<lgy; i++) x[i]=0; }
    else {lgz=lgx; for (i=lgy; i<lgx; i++) y[i]=0; }
 //incep adunarea
 for (t=i=0; i<lgz; i++)
     {
      val=x[i]+y[i]+t;
      z[i]=val%10;
      t=val/10;
     }
 //daca a mai ramas o cifra de transport
```

```
 if (t) z[lgz++]=t;
}

void afisare(NrMare x, int lgx)
{int i;
 for (i=lgx-1; i>=0; i--) fout<<x[i];
 fout<<'\n';
}

int compars(int i, int j)
//returneaza 1 daca secventa care incepe la pozitia i este mai mare
//lexicografic decat secventa care incepe la pozitia j
//0 daca sunt egale
//-1 altfel
{int k;
 for (k=0; k<lg && s[i+k]==s[j+k]; k++);
 if (k==lg) return 0;
 if (s[i+k]>s[j+k]) return 1;
 return -1;
}

void dif(NrMare x, int lgx, NrMare y, int lgy, NrMare z, int& lgz)
//z=x-y (se garanteaza x>=y)
{int i, val, t;
  //cel mai scurt dintre numere il completez cu zerouri nesemnificative
 lgz=lgx; for (i=lgy; i<lgx; i++) y[i]=0;
 //incep scaderea
 for (t=i=0; i<lgz; i++)
     {
      val=x[i]-y[i]+t;
      if (val<0) {val+=10; t=-1;}
         else t=0;
      z[i]=val;
     }
 //daca am zerouri nesemnificative, le elimin
 if (lgz>1 && z[lgz-1]==0) lgz--;
}

int comparnr(NrMare x, int lgx, NrMare y, int lgy)
{int i;
 if (lgx>lgy) return 1;
 if (lgx<lgy) return -1;
 for (i=0; i<lgx && x[i]==y[i]; i++);
 if (i==lgx) return 0;
 if (x[i]<y[i]) return -1;
 return 1;
}
```

7.4 Problema Pictura

Propusă de: prof. Emanuela Cerchez, Colegiul Național „Emil Racoviță” Iași

Robotul Vasile se visează artist plastic. El creează picturi 3D, pe un suport pe care îl scoate la imprimantă. O pictură 3D are formă dreptunghiulară și poate fi împărțită în „pixeli 3D”, aranjați pe n linii și m coloane, fiecare pixel 3D având o anumită înălțime. Inițial pictura este albă.

Un pixel 3D este considerat „vârf” dacă el are înălțimea strict mai mare decât înălțimile tuturor vecinilor săi. Doi pixeli 3D se consideră *vecini* dacă ei se află pe aceeași linie, pe coloane consecutive, sau se află pe aceeași coloană, pe linii consecutive.

Robotul Vasile pictează algoritmic, în modul următor: parcurge pixelii în ordinea liniilor, iar pe fiecare linie în ordinea coloanelor și dacă pixelul curent este un vârf „toarnă” peste el o nouă culoare (o culoare diferită de alb și de toate culorile utilizate până la momentul respectiv). Vom denumi culori pure, culorile pe care robotul Vasile le toarnă peste pixelii vârf. Culoarea pură se va „scurge” și va vopsi toți pixelii 3D vecini care au înălțime strict mai mică decât a vârfului vopsit, apoi se va scurge în continuare la vecinii vecinilor cu înălțimi strict mai mici ș.a.m.d., până când scurgerea nu mai este posibilă. Atunci când culoarea pură ajunge peste un pixel alb, acesta se va colora în culoarea respectivă. Dacă însă culoarea pură ajunge peste un pixel care a fost colorat anterior, culorile se vor combina și se va crea o nouă culoare. Culorile care se formează prin combinare vor fi diferite de toate culorile pure pe care robotul Vasile le toarnă peste pixelii vârf.

Deși se visează artist, robotul Vasile nu are simț estetic, ca urmare a stabilit trei criterii de evaluare a calității artistice a unei picturi: numărul maxim de culori pure care se combină pe un pixel 3D, numărul de culori distincte care apar în pictură (pure sau obținute prin combinare), respectiv dimensiunea maximă a unei zone colorate în aceeași culoare, diferită de alb. O zonă este formată din pixeli 3D cu proprietatea că, pentru oricare doi pixeli p_1 și p_2 din zona respectivă există o succesiune de pixeli 3D care începe cu p_1, se termină cu p_2 și oricare doi pixeli consecutivi sunt *vecini*. Dimensiunea unei zone este egală cu numărul de pixeli 3D din zona respectivă.

Cerințe

Scrieți un program care, cunoscând n și m (dimensiunile picturii), respectiv înălțimile pixelilor 3D, rezolvă următoarele trei cerințe:

1. determină numărul maxim de culori pure care se combină pe un pixel 3D;
2. determină numărul de culori distincte care apar în pictura creată conform algoritmului aplicat de robotul Vasile;
3. determină dimensiunea maximă a unei zone formată din pixeli 3D de aceeași culoare, diferită de alb.

Date de intrare

Fișierul de intrare `pictura.in` conține pe prima linie trei numere naturale C n m, reprezentând cerința care trebuie să fie rezolvată (1, 2 sau 3), numărul de linii, respectiv numărul de coloane ale picturii. Pe fiecare dintre următoarele n linii se află câte m numere naturale reprezentând înălțimile pixelilor 3D care constituie suprafața pe care pictează robotul Vasile (respectând ordinea liniilor, respectiv a coloanelor). Valorile scrise pe aceeași linie sunt separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire `pictura.out` va conține o singură linie pe care va fi scris răspunsul la cerința C din fișierul de intrare.

Restricții

- $3 \leq n, m \leq 300$
- Înălțimile pixelilor 3D sunt numere naturale $\leq 30\,000$.
- Pentru toate datele de test se garantează că numărul de culori pure care se combină pe un același pixel 3D este < 200.

#	Puncte	Restricții
1	30	$C = 1$
2	50	$C = 2$
3	20	$C = 3$

Exemple

pictura.in	pictura.out
1 5 3 8 7 8 8 20 18 11 5 30 19 4 50 2 3 40	3
2 5 3 8 7 8 8 20 18 11 5 30 19 4 50 2 3 40	7
3 5 3 8 7 8 8 20 18 11 5 30 19 4 50 2 3 40	4

Explicații

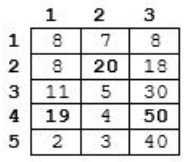
Înălțimile pixelilor 3D sunt ilustrate în figura 1, vârfurile fiind marcate îngroșat (înălțimile acestora fiind 20, 19, 50). Toți pixelii au inițial culoarea albă, marcată cu 0.

Vom colora succesiv cele 3 vârfuri. Colorând vârful cu înălțimea 20 în culoarea 1 obținem imaginea din figura 2.

Colorând vârful cu înălțimea 19 în culoarea 2 obținem imaginea din figura 3, unde am notat cu $(1, 2)$ combinația culorilor 1 și 2.

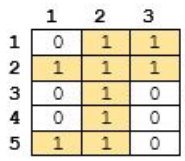
Colorând vârful cu înălțimea 50 în culoarea 3 obținem imaginea din figura 3, unde am notat cu $(1, 3)$ combinația culorilor 1 și 3, iar cu $(1, 2, 3)$ combinația culorilor 1, 2 și 3.

Sunt vizibile 7 culori distincte: 0, 1, 2, 3, $(1, 2)$, $(1, 3)$ și $(1, 2, 3)$. Numărul maxim de culori pure care se combină pe un același pixel 3D este 3. Dimensiunea maximă a unei zone formată din pixeli de aceeași culoare este 4 - zona colorată în culoarea $(1, 2, 3)$.



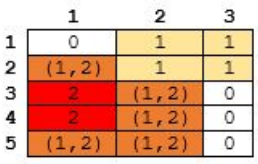
	1	2	3
1	8	7	8
2	8	**20**	18
3	11	5	30
4	**19**	4	**50**
5	2	3	40

Figura 1



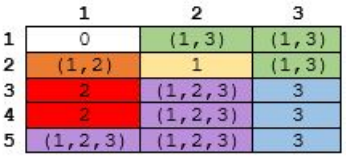
	1	2	3
1	0	1	1
2	1	1	1
3	0	1	0
4	0	1	0
5	1	1	0

Figura 2



	1	2	3
1	0	1	1
2	(1,2)	1	1
3	2	(1,2)	0
4	2	(1,2)	0
5	(1,2)	(1,2)	0

Figura 3



	1	2	3
1	0	(1,3)	(1,3)
2	(1,2)	1	(1,3)
3	2	(1,2,3)	3
4	2	(1,2,3)	3
5	(1,2,3)	(1,2,3)	3

Figura 4

7.5 Rezolvarea problemei Pictura

Parcurgem matricea care reprezintă pictura și identificăm elementele „vârf". Pentru fiecare vârf identificat adăugăm o nouă „culoare pură" și „vopsim" pictura în noua culoare adăugată
Culorile pure sunt reprezentate printr-o singură valoare numerică, dar culorile obținute prin amestecare sunt constituite dintr-o succesiune de valori numerice. Prin urmare vom reprezenta o culoare ca un vector în care reținem toate valorile numerice corespunzătoare culorilor pure care intră în componența acesteia. Memoria disponibilă permite utilizarea vectorilor alocați static, însă utilizarea vectorilor alocați dinamic (clasa `vector` din STL) ar conduce la o soluție cu un consum mic de memorie.

Pentru a vopsi pictura într-o nouă culoare c, începând din poziția vârfului curent, vom utiliza o stivă S pe care o inițializăm cu poziția vârfului curent.

Cât timp stiva S nu este vidă:

- Extragem elementul din vârful stivei (să-l notăm p).
- Parcurgem vecinii elementului p și verificăm pentru fiecare vecin dacă are o înălțime strict mai mică decât înălțimea poziției curente p (pentru a permite scurgerea culorii) și dacă nu a fost deja vopsit în culoarea c (pentru a nu vopsi un pixel în aceeași culoare de mai multe ori).
- Pentru fiecare vecin care îndeplinește condițiile de mai sus, adăugăm culoarea c în lista culorilor corespunzătoare acestei poziții din matrice și adăugăm în stivă acest vecin.

Algoritmul de vopsire este un algoritm asemănător cu un algoritm de *FILL*, numai că este posibilă parcurgerea aceleiași poziții de mai multe ori, pentru fiecare culoare pură adăugată.

Cerința 1

Pentru a rezolva cerința 1 este suficient să determinăm lungimea maximă a succesiunilor de culori obținute după vopsire.

Cerința 2

Pentru a rezolva cerința 2 trebuie să identificăm numărul de culori distincte. Pentru aceasta putem sorta succesiunile de culori și să numărăm apoi în timp liniar culorile distincte. Deoarece succesiunile de culori obținute sunt deja ordonate crescător, compararea acestora din punct de vedere lexicografic este facilă.

Cerința 3

Pentru a rezolva cerința 3, trebuie să identificăm fiecare zonă formată din pixeli de aceeași culoare, să determinăm aria fiecărei zone și apoi să determinăm maximul ariilor. Pentru a determina aria unei zone, aplicăm un algoritm de *FILL*.

7.6 Cod-sursă pentru problema Pictura

```
#include <fstream>
#include <algorithm>
```

```cpp
#include <vector>
#define DMAX 302
#define INF -1
#define ND 4

using namespace std;

ifstream fin("pictura.in");
ofstream fout("pictura.out");

struct pozitie {short int lin, col;};
int n, m, c;
short int H[DMAX][DMAX];
bool viz[DMAX][DMAX];
vector<int> C[DMAX][DMAX];
int dl[]={-1,0,1, 0};
int dc[]={ 0,1,0,-1};
int nrvf, nr, nrmaxim=1;
int vf;
pozitie S[DMAX*DMAX];
vector<int> a[DMAX*DMAX];

void citire();
bool varf(int i, int j);
void vopseste(int i, int j, int culoare);
void cerinta2();
void cerinta3();
void afisare();

int main()
{ int i, j;
  citire();
  for (i=1; i<=n; i++)
       for (j=1; j<=m; j++)
           if (varf(i,j))
              {nrvf++;
               vopseste(i, j, nrvf);
             }
 //afisare();
 for (i=1; i<=n; i++)
       for (j=1; j<=m; j++)
            if (C[i][j].size()>nrmaxim)
                nrmaxim=C[i][j].size();

 if (c==1) fout<<nrmaxim<<'\n';
    else
    if (c==2) cerinta2();
       else cerinta3();
 return 0;
}

void citire()
{int i, j;
 fin>>c>>n>>m;
 for (i=1; i<=n; i++)
     for (j=1; j<=m; j++) fin>>H[i][j];
 //bordare
 for (i=0; i<=m+1; i++) H[0][i]=H[n+1][i]=INF;
 for (i=0; i<=n+1; i++) H[i][0]=H[i][m+1]=INF;
}
```

```
bool varf(int i, int j)
{int k;
 for (k=0; k<ND; k++)
     if (H[i][j]<=H[i+dl[k]][j+dc[k]]) return 0;
 return 1;
}

void vopseste(int i, int j, int culoare)
{pozitie p, v;
 int k;
 p.lin=i; p.col=j; C[i][j].push_back(culoare);
 vf=1; S[vf]=p;
 while (vf>0)
       {p=S[vf--];
        for (k=0; k<ND; k++)
             {v.lin=p.lin+dl[k]; v.col=p.col+dc[k];
              if (H[v.lin][v.col]!=INF &&
                  H[p.lin][p.col]>H[v.lin][v.col] &&
                  (C[v.lin][v.col].size()==0 || C[v.lin][v.col].back()!=culoare))
                {
                 C[v.lin][v.col].push_back(culoare);
                 S[++vf]=v;
                }
             }
        }
}

void afisare()
{int i, j, k;
 for (i=1; i<=n; i++)
     {for (j=1; j<=m; j++)
         if (C[i][j].size()==0) fout<<0<<' ';
            else
            {fout<<"(";
             for (k=0; k<C[i][j].size(); k++)
                  fout<<C[i][j][k]<<',';
             fout<<") ";
            }
      fout<<"\n";
     }
 fout<<'\n';
}

bool compar(vector<int> a, vector<int> b)
{int i=0;
 while (i<a.size()&& i<b.size())
        {if (a[i]<b[i]) return 1;
         if (a[i]>b[i]) return 0;
         i++;}
 if (i==a.size() && i<b.size()) return 1;
 return 0;
}

int modul (int x)
{if (x<0) return -x;
 return x;
}

bool diferit(vector<int> a, vector<int> b)
{int i;
 if (a.size()!=b.size()) return 1;
```

```
 for (i=0; i<b.size(); i++)
     if (a[i]!=b[i]) return 1;
 return 0;
}
void cerinta2()
{int i, j;
 int rez=0;
 for (i=1; i<=n; i++)
     for (j=1; j<=m; j++)
          a[++nr]=C[i][j];
 sort(a+1,a+nr+1, compar);
 rez=1;
 for (i=2; i<=nr; i++)
     if (diferit(a[i-1],a[i])) rez++;
 fout<<rez<<'\n';
}

int zona(int i, int j)
{vector<int> c=C[i][j];
 int k, rez=1;
 pozitie p, v;
 vf=1; S[vf].lin=i; S[vf].col=j; viz[i][j]=1;
 while (vf>0)
       {p=S[vf--];
        for (k=0; k<ND; k++)
            {v.lin=p.lin+dl[k]; v.col=p.col+dc[k];
             if (!diferit(C[p.lin][p.col],C[v.lin][v.col])&&
                 !viz[v.lin][v.col]&&H[v.lin][v.col]!=INF)
                {viz[v.lin][v.col]=1;
                 S[++vf]=v; rez++;
                }
            }
        }
 return rez;
}

void cerinta3()
{int i, j, aria;
 int ariamax=0;
 for (i=1; i<=n; i++)
     for (j=1; j<=m; j++)
         if (C[i][j].size()>0)
            {//incepe o noua zona;
             aria=zona(i,j);
             if (aria>ariamax) {ariamax=aria;}
            }
 fout<<ariamax<<'\n';
}
```

7.7 Problema Secv9

Propusă de: stud. Ioan-Cristian Pop, Universitatea Politehnica București

Cifra de control a unui număr natural x se obține astfel:

- dacă numărul x are o singură cifră, atunci cifra de control a lui x este egală cu x;
- dacă numărul x are cel puțin două cifre, atunci se calculează suma cifrelor lui x (să o notăm cu s); cifra de control a lui x va fi egală cu cifra de control a lui s.

De exemplu, cifra de control a numărului 175 este egală cu cifra de control a numărului 13 $(1 + 7 + 5)$, care este egală cu 4 $(1 + 3)$.

Fie $x_1, x_2, \ldots, x_N$ un șir de N numere naturale. Două poziții i și j, cu $1 \leq i \leq j \leq N$, definesc secvența $[i, j]$ care va conține numerele $x_i, x_{i+1}, \ldots, x_j$.

O secvență $[i, j]$ cu proprietatea că suma tuturor elementelor din secvență are cifra de control egală cu 9 o vom denumi `secv9`.

Cerințe

Scrieți un program care, cunoscând N, numărul de elemente din șir, respectiv $x_1, x_2, \ldots, x_N$, elementele din șir, rezolvă următoarele două cerințe:

1. afișează lungimea maximă a unei secvențe `secv9`;
2. afișează numărul de secvențe `secv9` din șir.

Date de intrare

Fișierul de intrare `secv9.in` conține pe prima linie două numere naturale C și N, reprezentând cerința care trebuie rezolvată (1 sau 2), respectiv lungimea șirului. Următoarea linie conține N numere naturale $x_1, x_2, \ldots, x_N$, separate prin câte un spațiu, reprezentând elementele din șir.

Date de ieșire

Fișierul de ieșire `secv9.out` va conține pe prima linie un singur număr natural, reprezentând răspunsul la cerința C din fișierul de intrare.

Restricții

- $1 \leq N \leq 1\,000\,000$
- $0 \leq x_i \leq 1\,000$, pentru oricare $1 \leq i \leq N$.
- Se garantează pentru toate datele de test că există cel puțin o secvență `secv9`.

#	Puncte	Restricții
1	8	$C = 1$ și $1 \leq N \leq 1000$
2	10	$C = 1$ și $1001 \leq N \leq 5\,000$
3	22	$C = 1$ și $5001 \leq N \leq 1\,000\,000$
4	12	$C = 2$ și $1 \leq N \leq 1000$
5	15	$C = 2$ și $1001 \leq N \leq 5\,000$
6	33	$C = 2$ și $5001 \leq N \leq 1\,000\,000$

Exemple

secv9.in	secv9.out	Explicații
1 7 1 7 6 1 11 5 9	3	Sunt două secvențe secv9 în șirul dat: • secvența [3,5], alcătuită din numerele 6, 1 și 11, are suma termenilor 18=6+1+11, deci cifra de control este 9; • secvența [7,7], alcătuită din numărul 9, are suma termenilor 9, deci cifra de control este 9. Lungimea maximă a unei secvențe secv9 este 3.
2 7 1 7 6 1 11 5 9	2	Sunt două secvențe secv9 în șir: [3,5] și [7,7].

7.8 Rezolvarea problemei Secv9

Observație inițială

Cifra de control a unui număr nenul x se poate obține prin calcularea restului împărțirii lui x la 9, cu precizarea că, dacă restul este 0, atunci cifra de control este egală cu 9. Dacă x este egal cu 0, atunci cifra de control este egală cu 0.

De exemplu, pentru $x = 175$, cifra de control este egală cu 4. De asemenea, restul împărțirii la 9 este egal cu 4.

Sume parțiale

Pentru rezolvarea celor 2 cerințe, ne vom folosi de sume parțiale ($sp[i]$ reprezintă suma tuturor numerelor de la x_1 până la x_i, și se poate calcula astfel: $sp[i] = sp[i-1] + x_i$).

Astfel, pentru doi indici i și j cu $i \leq j$, dacă $sp[j] - sp[i] > 0$ și $(sp[j] - sp[i])\%9 = 0$, atunci secvența $x_{i+1}, x_{i+2}, \ldots, x_j$ este de tip secv9.

Cerința 1

este suficient să căutăm, pentru fiecare rest r la împărțirea cu 9 de la 0 la 8, cea mai din stânga apariție st, respectiv cea mai din dreapta apariție dr a unei sume parțiale cu restul r din vectorul sp. Dacă suma $sp[dr] - sp[st] > 0$, atunci secvența $x_{st+1}, x_{st+2}, \ldots, x_{dr}$, este un candidat. Reținem o astfel de secvență de lungime maximă.

Cerința 2

Construim un vector de frecvență f, unde $f[i]$ reprezintă câte sume parțiale au restul la împărțirea cu 9 egal cu i. Dacă avem $f[i]$ sume cu restul i, înseamnă ca vor exista $f[i] \times (f[i]-1)/2$ secvențe secv9 (grupând două câte două).

Vom număra numărul de secvențe pentru toate resturile de la 0 la 8 cu metoda precizată mai sus. Se observă însă, un caz particular: sumele cu restul 0 pot avea suma propriu-zisă egală cu 0, nefiind o secvență secv9. De aceea, este necesar să scădem din rezultat numărul de secvențe din șir cu suma egală cu 0.

Pentru a număra câte secvențe au suma egală cu 0, este suficient să identificam secvențele de 0 din șir. O secvență de k 0-uri consecutive conține $k \times (k-1)/2$ secvențe cu suma egală cu 0.

7.9 Cod-sursă pentru problema Secv9

```
#include <fstream>
#include <algorithm>
#include <iostream>
#include <string>

using namespace std;
const int NMAX = 1000000;

int freq[10], first[10], last[10];
int sp[NMAX + 5], v[NMAX + 5], n;
bool viz[10];

long long solve_task1() {
    int r;

    for (int i = 1; i <= n; ++i) {
        r = sp[i] % 9;
        if (viz[r] == 0) {
            viz[r] = 1;
            first[r] = i;
        }
        last[r] = i;
    }
    first[0] = 0;

    int ans = -1;
    for (int i = 0; i <= 9; ++i) {
        if (viz[i]) {
            int sum = sp[last[i]] - sp[first[i]];
            if (sum > 0) {
                ans = max(ans, last[i] - first[i]);
            }
        }
    }
    return ans;
}

long long solve_task2() {
    long long ans = 0;
    int r, nr_zero = 0;

    for (int i = 1; i <= n; ++i) {
        r = sp[i] % 9;
        ++freq[r];

        if (v[i] == 0) {
            ++nr_zero;
            ans -= 1LL * nr_zero;
        } else {
            nr_zero = 0;
        }
    }
    ++freq[0];

    for (int i = 0; i <= 9; ++i) {
```

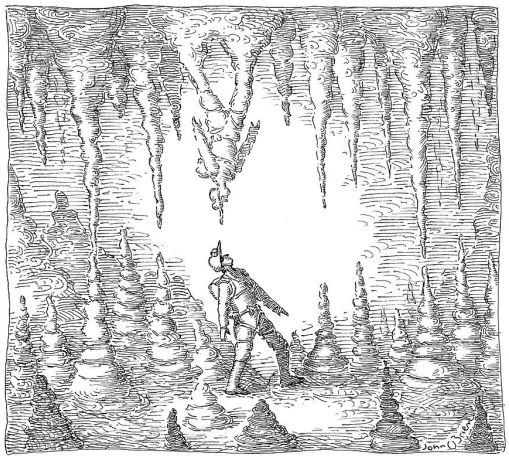
```
        ans = ans + 1LL * freq[i] * (freq[i] - 1) / 2;
    }
    return ans;
}

void solve_test(string in_file, string out_file) {
    ifstream fin(in_file);
    ofstream fout(out_file);
    long long ans;
    int caz;

    fin >> caz >> n;

    for (int i = 1; i <= n; ++i) {
        fin >> v[i];
        sp[i] = sp[i - 1] + v[i];
    }

    if (caz == 1) {
        ans = solve_task1();
    } else {
        ans = solve_task2();
    }
    fout << ans << endl;
}

int main() {
    solve_test("secv9.in", "secv9.out");
    return 0;
}
```

Capitolul 8

ONI 2024, clasa a VIII-a

8.1 Problema Geologie

Propusă de: stud. Livia Măgureanu, Facultatea de Limbi și Literaturi Străine, Universitatea București



Geologul George a descoperit recent în zona Iașiului o peșteră foarte frumoasă, ideală pentru studierea stalactitelor. În peșteră se află N stalactite, așezate în linie dreaptă. După săptămâni de muncă și analizare temeinică a peșterii, George a descoperit că fiecare stalactită este formată din straturi de depuneri cu diverse compoziții chimice.

Pentru a economisi timp, geologul a inventat un sistem prin care să noteze structura fiecărei stalactite. El a asociat fiecărui tip de strat diferit câte un număr prim, astfel încât, dacă două straturi au aceeași compoziție chimică, ele au asociate același număr și a calculat valoarea stalactitei, ca produs al numerelor prime ale straturilor ei.

Astfel, dacă are o stalactită cu 4 straturi, cărora le-a asociat numerele prime $2, 11, 2, 3$, atunci stalactita respectivă va avea valoarea $132 = 2 \cdot 2 \cdot 3 \cdot 11$.

George a obținut șirul A, de N numere, reprezentând șirul valorilor stalactitelor, în ordinea în care ele apar în peșteră.

Pentru a-și finanța mai departe cercetările, George i-a invitat pe membrii comisiei științifice a Muzeului de Mineralogie și Petrografie „Grigore Cobălcescu" din Iași să viziteze peștera. Pentru a-i impresiona, el plănuiește să aleagă o subsecvență de stalactite, aflate pe poziții consecutive în peșteră, pe care să le curețe, eliminând eventual unele straturi, astfel încât pentru fiecare stalactită să rămână exact un strat din componența acesteia (deci un singur număr prim), iar subsecvența obținută să fie formată numai din numere prime consecutive, în ordine strict crescătoare (gusturile matematice ale lui George...). Spunem că două numere prime x și y sunt consecutive dacă nu există niciun alt număr prim z cu $x < z < y$.

Cerințe

Aflați numărul maxim de stalactite care formează o subsecvență obținută prin curățare, care să fie pe gustul lui George.

Date de intrare

Pe prima linie a fișierului `geologie.in` se află numărul natural N. Pe a doua linie se află N numere naturale, reprezentând termenii șirului A, al valorilor stalactitelor în ordinea din peșteră.

Date de ieșire

Fișierul `geologie.out` trebuie să conțină pe prima linie numărul maxim cerut.

Restricții

- $1 \leq N \leq 1\,000\,000$.
- $2 \leq A_i \leq 1\,000\,000, \forall i, 1 \leq i \leq n$.

#	Puncte	Restricții
1	17	A_i este prim, $\forall i, 1 \leq i \leq n$
2	24	$N \leq 1\,000$
3	36	$A_i \leq 50\,000$
4	23	Nu există alte restricții

Exemple

geologie.in	geologie.out	Explicații
7 3 4 6 10 11 14 21	3	$A_1 = 3$ poate fi curățat astfel încât să rămână 3 $A_2 = 4$ poate fi curățat astfel încât să rămână 2 $A_3 = 6$ poate fi curățat astfel încât să rămână 2 sau cu 3 $A_4 = 10$ poate fi curățat astfel încât să rămână 2 sau 5 $A_5 = 11$ poate fi curățat astfel încât să rămână 11 $A_6 = 14$ poate fi curățat astfel încât să rămână 2 sau 7 $A_7 = 21$ poate fi curățat astfel încât să rămână 3 sau 7 Pentru soluția optimă A poate fi modificat în felul următor. 3 **2 3 5** 11 7 3 obținând astfel o subsecvență de 3 stalactite. A se observa că subsecvența 3 **2 3 5 11 7** 3 conține de asemenea numere prime consecutive, doar că acestea nu apar în ordine strict crescătoare

8.2 Rezolvarea problemei Geologie

Subtask 1

Pentru primul subtask, când toate numerele sunt prime, problema devine să determinăm pentru fiecare element al câtelea număr prim este și, apoi, să calculăm subsecvența de lungime maximă formată din numere prime consecutive.

Vom folosi Ciurul lui Eratostene pentru a determina numerele prime și numărul de ordine al acestora în lista numerelor prime. Odată ce avem această precalculare facută, putem găsi subsecvența de lungime

maximă cu o parcurgere și un contor pe care îl resetăm de fiecare dată când întâlnim două numere alăturate care nu sunt prime consecutive. Complexitatea acestei soluții este $O(V_{max} \cdot \log V_{max} + N)$, unde V_{max} este valoarea maximă din șir.

Subtask 2

Soluția se bazează pe cea de la subtask-ul anterior. Vom folosi în continuare Ciurul lui Eratostene pentru a determina numerele prime și pentru a descompune numerele din șirul dat în factori primi.

Se observă că fiecare număr din șir poate avea maxim 7 divizori primi diferiți. Vom considera, pe rând, fiecare poziție de început și fiecare număr prim cu care poate începe subsecvența pe care o căutăm. Astfel avem aproximativ $7 \cdot N$ variante de început posibile și pentru fiecare dintre ele vom calcula lungimea maximă a subsecvenței care pleacă din acel punct. Acest lucru se poate face parcurgând șirul în paralel cu șirul numerelor prime și verificând dacă numărul din șir la care am ajuns este divizibil cu numărul prim la care am ajuns.

Complexitatea finala a acestei abordări este $O(V_{max} \cdot \log V_{max} + N^2)$ sau $O(V_{max} \cdot \log V_{max} + N^2 \cdot \log V_{max}$ în funcție de implementarea folosită pentru descompunerea în factori primi.

Subtask 3 & 4

Soluția pentru ultimele două subtask-uri se bazează tot pe soluția pentru primul subtask. Diferența este că, pentru că acum numerele pot avea mai mulți divizori primi, vom calcula și reține soluția optimă pentru fiecare număr prim din descompunere în paralel.

Se observă că fiecare număr din șir poate avea maxim 7 divizori primi diferiți. Prin urmare determinăm, pentru fiecare poziție din șir, care sunt subsecvențele care se pot termina pe poziția respectivă și lungimile acestora. Să presupunem că avem un șir în care al $i-1$-lea număr din șir este 31, al i-lea număr din șir este $84 = 2^2 \cdot 3 \cdot 7$ și al $i+1$-lea număr este $231 = 3 \cdot 7 \cdot 11$. Când ajungem pe poziția $i+1$ vom prelungi două dintre subsecvențele care se terminau pe poziția i, cele care corespund numerelor prime 2 și, respectiv, 7. Astfel pentru poziția $i+1$ vom avea 3 subsecvențe care se pot termina la poziția respectivă: cea pentru 3 de lungime 2, cea pentru 3 de lungime 1 și cea pentru 11 de lungime 2. Putem implementa trecerea de la i la $i+1$ fie asemănător cu o interclasare, fie folosindu-ne de lista numerelor prime pe care am construit-o în precalculare.

Complexitatea acestei soluții și punctajul obținut diferă în funcție de cum se implementează descompunerea în factori primi, dar soluția oficială are complexitatea $O(V_{max} \cdot \log V_{max} + N)$.

8.3 Cod-sursă pentru problema Geologie

```
#include <fstream>
#include <vector>

using namespace std;

ifstream cin("geologie.in");
ofstream cout("geologie.out");

const int kMaxV = 1e6;

int ComputeCiur(vector<int>& id, vector<int>& prime) {
        id.resize(kMaxV + 1);

        int cnt = 0;
        for (int i = 2; i <= kMaxV; i++) {
                if (id[i] == 0) {
                        cnt++;
```

```
                        prime.push_back(i);
                        for (int j = i; j <= kMaxV; j += i) {
                                id[j] = cnt;
                        }
                }
        }

        return cnt;
}

void Read(int& n, vector<int>& v) {
        cin >> n;
        v.resize(n);
        for (int i = 0; i < n; i++) {
                cin >> v[i];
        }
}

void Solve() {
        // Ciur
        vector<int> id;
        vector<int> prime;
        int cnt_prime = ComputeCiur(id, prime);

        int n;
        vector<int> v;
        Read(n, v);

        // Finding max secv
        int max_secv = 0;
        vector<pair<int, int>> ls(cnt_prime + 1);

        for (int i = 0; i < v.size(); i++) {
                int x = v[i];
                while (x > 1) {
                        ls[id[x]] = {i, 1};
                        if (ls[id[x] - 1].first == i - 1) {
                                ls[id[x]].second = ls[id[x] - 1].second + 1;
                        }
                        max_secv = max(max_secv, ls[id[x]].second);
                        x = x / prime[id[x] - 1];
                }
        }

        cout << max_secv << "\n";
}

int main() {
        Solve();
        return 0;
}
```

8.4 Problema Kmajo

Propusă de: stud. Bogdan-Ioan Popa, Facultatea de Matematică-Informatică, Universitatea București

Se dă un șir A cu N elemente, numere naturale nenule, și un număr natural K.

O subsecvență a șirului este un șir format din unul sau mai multe elemente aflate pe poziții consecutive în șirul inițial.

Spunem că o valoare x se numește element majoritar al unei secvențe de lungime m, dacă ea apare în aceasta de cel puțin $\lfloor \frac{m}{2} \rfloor + 1$ ori.

Cerințe

Să se afișeze valorile care sunt elemente majoritare pentru cel puțin o subsecvență de lungime mai mare sau egală cu K a șirului A.

Date de intrare

Pe prima linie a fișierului `kmajo.in` se află numerele naturale N și K, cu semnificația din enunț, iar pe a doua linie se găsesc N numere naturale, reprezentând elementele șirului A. Valorile aflate pe aceeași linie sunt separate prin câte un spațiu.

Date de ieșire

Fișierul `kmajo.out` conține pe prima linie valorile cerute, în ordine strict crescătoare, separate prin câte un spațiu, sau valoarea -1, dacă nu există cel puțin o valoare care să respecte restricțiile impuse.

Restricții

- $1 \leq K \leq N \leq 1\,000\,000$.
- $1 \leq A_i \leq N, \forall i, 1 \leq i \leq n$.

#	Puncte	Restricții
1	11	$K = N$
2	16	$N \leq 1\,000$
3	22	$N \cdot K \leq 30\,000\,000$
4	30	$N \leq 100\,000$
5	21	Nu există alte restricții

Exemple

kmajo.in	kmajo.out	Explicații
12 3 2 2 1 3 4 2 2 3 3 3 4 4	2 3 4	2 este element majoritar în mai multe subsecvențe de lungime mai mare sau egală cu 3, un exemplu fiind: 2 2 1 3 **4 2 2** 3 3 3 4 4 3 este element majoritar în mai multe subsecvențe de lungime mai mare sau egală cu 3, un exemplu fiind: 2 2 1 3 4 **2 2 3 3 3** 4 4 4 este element majoritar în următoarea subsecvență evidențiată, de lungime mai mare sau egală cu 3. 2 2 1 3 4 2 2 3 3 **3 4 4**

8.5 Rezolvarea problemei Kmajo

Soluția de complexitate $O(N \cdot K)$ obține aproximativ 60 puncte

Se observă că este suficient să testăm subsecvențe cu lungimi cuprinse în intervalul $[K, 2 \cdot K]$ întrucât dacă există o subsecvență de lungime $L > 2 \cdot K$ care admite element majoritar, atunci cel puțin una dintre jumătățile sale va avea același element majoritar. Pentru fiecare L din intervalul $[K, 2 \cdot K]$ vom glisa o fereastră de lungime L peste întregul vector, menținând frecvențele tuturor elementelor. Atunci când găsim un element cu frecvența mai mare decât $[L/2]$ vom marca valoarea respectivă ca fiind parte din soluție.

Soluția de complexitate $O(N \cdot \sqrt{N})$ obține aproximativ 79 puncte

Dacă $K < 2 \cdot \sqrt{N}$, vom aplica soluția în $O(N \cdot K)$. Altfel se observă că vor fi maximum $O(\sqrt{N})$ candidați la a fi parte din soluție. Fiecare candidat va fi testat în $O(N)$. Pentru un candidat x fixat, vom construi un vector auxiliar B în care $B[i] = 1$ dacă $A[i] = x$ sau $B[i] = -1$ dacă $A[i] \neq x$. Tot ce rămâne de făcut este să găsim o subsecvență de lungime $\geq K$ care are suma > 0 în vectorul B.

Soluția de complexitate $O(N)$ obține 100 puncte

Vom testa fiecare valoare distinctă x dacă apare sau nu ca element majoritar într-o subsecvență de lungime cel puțin K. Fie $p_1 < p_2 < \cdots < p_T$ pozițiile pe care apare valoarea x în A, unde T reprezintă frecvența valorii x în A. Pentru ca x să fie element majoritar într-o subsecvență de lungime cel puțin K trebuie să existe doi indici $j \leq i$ astfel încât cel puțin una dintre cele două condiții să se respecte:

- $(i - j + 1) \cdot 2 > p_i - p_j + 1$, iar $p_i - p_j + 1 \geq K$
- $(i - j + 1) \cdot 2 > K$, iar $p_i - p_j + 1 < K$

Existența celor doi indici poate fi testată în $O(T)$ folosind tehnica Two-Pointers.

8.6 Cod-sursă pentru problema Kmajo

```
#include <bits/stdc++.h>
using namespace std;
const int N_MAX = 1e6;
const int INF = 1e9;
```

```cpp
int N, K;
vector <int> pos[N_MAX + 5];

bool check(vector <int> &pos, int N, int K) {
    if(pos.empty()) {
        return false;
    }

    int j = 0;
    pos.insert(pos.begin(), -N);

    int min_j = INF;
    for(int i = 1; i < pos.size(); i++) {
        while(j < i && pos[i] - pos[j + 1] + 1 >= K) {
            j++;
            min_j = min(min_j, 2 * j - pos[j]);
        }

        if(2 * (i - j) > K) {
            return true;
        }

        if(2 * i + 1 - pos[i] > min_j) {
            return true;
        }
    }
    return false;
}

int main() {
#ifndef LOCAL
    freopen("kmajo.in", "r", stdin);
    freopen("kmajo.out", "w", stdout);
#endif

    ios::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);
    cin >> N >> K;

    for(int i = 1; i <= N; i++) {
        int x;
        cin >> x;
        pos[x].push_back(i);
    }

    int cnt = 0;
    for(int i = 1; i <= N; i++) {
        if(check(pos[i], N, K)) {
            cout << i << " ";
            cnt++;
        }
    }

    if(cnt == 0) {
        cout << "-1";
    }
    cout << "\n";
    return 0;
}
```

8.7 Problema Mandms

Propusă de: stud. Mircea Măierean, Facultatea de Matematică și Informatică, Universitatea Babeș-Bolyai Cluj Napoca

Andra are un pachet cu n tipuri de buline de ciocolată, cu câte c_i buline de fiecare tip i. Numărul c_i depinde de patru factori, notați cu x, y, z, t, și se calculează astfel:

- $c_1 = x$
- $c_i = (c_{i-1} \cdot y)\%t + z$, $\forall i, 2 \leq i \leq n$

unde s-a notat cu a % b restul împărțirii numărului natural a la numărul natural nenul b.

Andra dorește să utilizeze toate bulinele pentru a construi piramide, fiecare fiind formată din unul sau mai multe rânduri, numerotate începând de la 1. Pentru fiecare piramidă în parte, pe rândul i, se află 2^{i-1} buline. Spre exemplu, pe rândul 8 al unei piramide, se află $2^7 = 128$ de buline de ciocolată. Pe fiecare rând al unei piramide se află unul sau mai multe tipuri de buline, iar același tip de buline se poate folosi pe oricâte rânduri. Dintre piramidele care se pot forma, cele **serioase** conțin pe fiecare rând doar un tip de buline.

Cerințe

Folosind toate bulinele, ajutați-o pe Andra să determine:

1) Numărul minim de piramide de ciocolată pe care le poate forma.

2) Numărul minim de piramide serioase de ciocolată pe care le poate forma, astfel încât toate cele obținute să fie de acest fel.

Date de intrare

Fișierul de intrare `mandms.in` conține pe prima linie numărul p, care poate fi doar 1 sau 2.

Pe a doua linie a fișierului, se află numărul natural n, cu semnificația din enunț.

Pe a treia linie se află numerele naturale x, y, z, t, în această ordine, cu semnificația din enunț.

Date de ieșire

Dacă $p = 1$, fișierul de ieșire `mandms.out` conține un număr natural, reprezentând valoarea precizată la cerința 1.

Dacă $p = 2$, fișierul de ieșire `mandms.out` conține un număr natural, reprezentând valoarea precizată la cerința 2.

Restricții

- $1 \leq n \leq 2\,000\,000$.
- $1 \leq x, y,\ t < 2^{64}$.
- $0 \leq z < 2^{64}$.
- $1 \leq c_i < 2^{64}$.
- $0 \leq c_i \cdot y < 2^{64}$, $\forall i, 2 \leq i \leq n$,

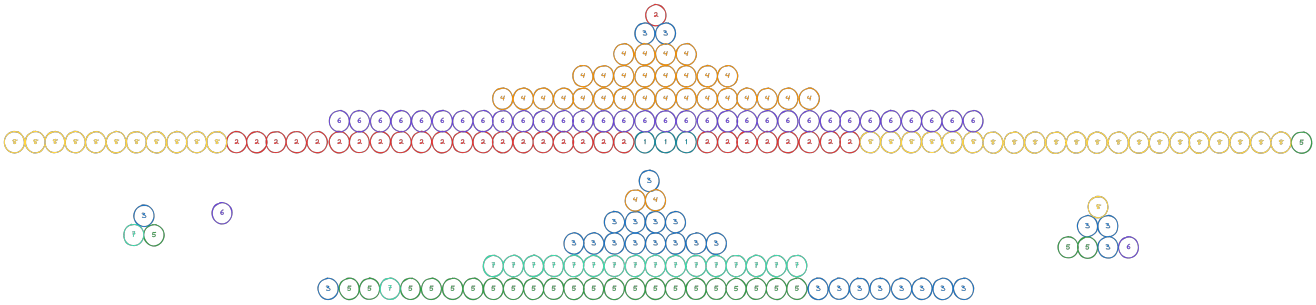
#	Puncte	Restricții
1	30	$p = 1, n \leq 1\,500\,000$, numărul total de buline este strict mai mic decât 2^{64}
2	10	$p = 1, n \leq 1\,500\,000$
3	40	$p = 2, n \leq 100\,000$
4	20	$p = 2$, nu există alte restricții suplimentare

Exemple

mandms.in	mandms.out	Explicații
1 8 3 15 18 17	5	$x = 3, y = 15, z = 18, t = 17$ Numărul de buline de cele 8 tipuri se calculează astfel: $c_1 = x = 3$ $c_2 = (3 \cdot 15)\%17 + 18 = 29$ $c_3 = (29 \cdot 15)\%17 + 18 = 28$ $c_4 = (28 \cdot 15)\%17 + 18 = 30$ $c_5 = (30 \cdot 15)\%17 + 18 = 26$ $c_6 = (26 \cdot 15)\%17 + 18 = 34$ $c_7 = (34 \cdot 15)\%17 + 18 = 18$ $c_8 = (18 \cdot 15)\%17 + 18 = 33$ Numărul minim de piramide care se pot forma este 5.
2 8 3 15 18 17	7	$x = 3, y = 15, z = 18, t = 17$ Numărul de buline de cele 8 tipuri se calculează astfel: $c_1 = x = 3$ $c_2 = (3 \cdot 15)\%17 + 18 = 29$ $c_3 = (29 \cdot 15)\%17 + 18 = 28$ $c_4 = (28 \cdot 15)\%17 + 18 = 30$ $c_5 = (30 \cdot 15)\%17 + 18 = 26$ $c_6 = (26 \cdot 15)\%17 + 18 = 34$ $c_7 = (34 \cdot 15)\%17 + 18 = 18$ $c_8 = (18 \cdot 15)\%17 + 18 = 33$ Numărul minim de piramide serioase care se pot forma, astfel încât toate cele obținute să fie serioase, este 7.

Posibile configurații ale piramidelor

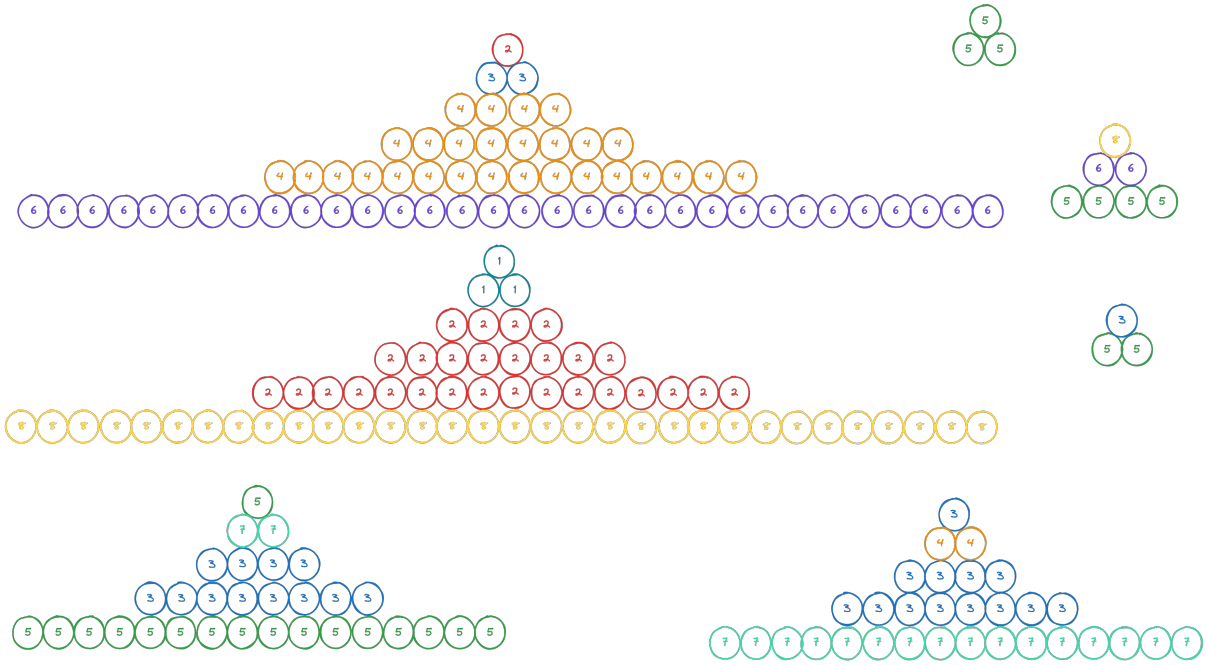
Exemplul 1



În această configurație, avem o piramidă cu 7 rânduri, una cu 6 rânduri, una cu 3 rânduri, una cu 2 rânduri, și una cu un singur rând.

Exemplul 2

Pentru acest exemplu, trebuie formate **doar** piramide serioase. Cele 7 piramide pot fi:



În această configurație, avem două piramide cu 6 rânduri, două cu 5 rânduri, una cu 3 rânduri, și două cu 2 rânduri.

8.8 Rezolvarea problemei Mandms

Rezolvarea problemei se bazează pe un algoritm de tip Greedy. Se observă faptul că, indiferent de numărul total de rânduri, o piramidă cu n rânduri va conține în totalitate $2^n - 1$ buline. Ca să minimizăm numărul total de piramide, încercăm la construcția fiecăreia să obținem o piramidă cu un număr cât mai mare de rânduri.

Cerința 1

Pentru subtaskul 1, calculăm numărul total de buline. La fiecare pas, selectăm cel mai apropiat număr de forma $2^p - 1$, care este mai mic sau egal cu totalul bulinelor. Scădem din total această valoare, și repetăm algoritmul până când numărul de buline devine 0. Răspunsul nostru este egal cu numărul total de repetări ale acestei proceduri.

La subtaskul 2, suma elementelor depășește limita maximă admisă de tipul de date unsigned long long, deci nu mai putem aplica direct procedeul descris la subtaskul 1. Pentru valorile maximale, suma totală a bulinelor nu depășește $n \cdot 2^{64} \approx 2^{21} \cdot 2^{64} = 2^{85}$. Ținem un vector de frecvență, care să poată reține frecvența fiecărei puteri. Pentru fiecare tip de buline existent, se descompune numărul în sumă de puteri ale lui doi, având termeni cât mai mari, și vom incrementa în vectorul de frecvență fiecare putere care apare în scrierea sumei (spre exemplu, $76 = 2^6 + 2^3 + 2^2$, deci vom incrementa vectorul pe pozițiile 2, 3, și 6). Pentru a putea forma o piramidă cu n rânduri, trebuie ca toate în vectorul de frecvență pozițiile de la 0 la k–1 să fie nenule, unde k este cea mai mare putere care apare. Primul pas este să mutăm frecvențele pe poziții cât mai mari: dacă pe poziția i avem frecvența x, atunci putem incrementa pe poziția $i + 1$ valoarea $x/2$, iar elementul de pe poziția i va deveni restul împărțirii lui x la 2. Spre exemplu, dacă am avea frecvența poziției 2 egală cu 5 și frecvența poziției 3 egală cu 8, după transformare,

vectorul va avea pe poziția 2 valoarea 1 și pe poziția 3 valoarea 10. La al doilea pas, iterăm de la 1 până la valoarea maximă găsită în descompunere. Pentru fiecare poziție i, dacă valoarea frecvenței este nenulă, ne asigurăm că toate pozițiile cu un index având valoarea mai mică, sunt diferite de 0. Folosim proprietatea că $2^i = 2^{i-1} + 2^{i-1}$. Scădem o unitate de pe poziția i, și adăugăm 2 la poziția anterioară în vector, iar apoi decrementăm valoarea i. Repetăm acest lucru, până când valoarea frecvenței de pe poziția de la care am început acest procedeu devine 0. După ce ne-am asigurat că avem o secvență continuă de valori nenule, adăugăm la totalul piramidelor valoarea nenulă minimă care se găsește în vectorul de frecvență.

Cerința 2

Pentru rezolvarea subtaskului 1, procedeul este exact ca cel descris anterior, însă primul pas nu mai este necesar, ci putem să începem direct cu pasul al doilea, după ce am descompus numerele în sume de puteri ale lui 2.

Pentru rezolvarea subtaskului 2, translatarea pas cu pas a valorilor de la o poziție la alta nu va intra în timp. Pentru generarea optimă a piramidelor serioase, frecvența rândurilor este în ordine descrescătoare. Rândul 1 va apărea de cele mai multe ori, după aceea rândul 2, și tot așa, până la cel mai mare rând posibil. Obiectivul final este să prelucrăm tabelul de frecvențe, astfel încât să avem toate valorile în ordine descrescătoare. Translatăm valorile frecvențelor, începând de la cea mai mare poziție. Dacă valoarea de pe poziția i este mai mare decât valoarea poziției $i-1$, vom încerca să le apropiem cât de mult posibil. Fie x numărul minim de buline pe care trebuie să le scădem din frecvența de pe poziția i. Prin mutarea a x buline de pe poziția i, vor apărea $2 \cdot x$ buline pe poziția $i-1$.

Pentru a-l determina pe x, avem ecuația:

$fr[i-1] + 2 \cdot x \geq fr[i] - x, 1 < i \leq k$, unde fr este vectorul de frecvență, și k este cea mai mare putere care apare în scrierea unei sume ca puteri ale lui 2.

$fr[i-1] + 2 \cdot x \geq fr[i] - x \Leftrightarrow 3 \cdot x \geq fr[i] - fr[i-1] \Leftrightarrow x \geq \frac{(fr[i]-fr[i-1])}{3}$

Din valoarea frecvenței aflate la poziția i scădem x, și la poziția $i-1$ creștem valoarea cu $2 \cdot x$. Executăm acest procedeu pentru fiecare pereche de indici care respectă această proprietate.

Există riscul ca o apropiere să afecteze pozițiile mai mari, și să nu mai respecte condiția impusă. Se repetă procedeul descris până când tot vectorul conține toate elementele în ordine descrescătoare. La fiecare executare a procedeului, diferențele dintre frecvențe devin tot mai mici, așa că acesta se va repeta de puține ori.

8.9 Cod-sursă pentru problema Mandms

```
#include <fstream>
#include <limits.h>
using namespace std;
ifstream fin("mandms.in");
ofstream fout("mandms.out");

int fr[100];

void decomposeNumber(unsigned long long a)
{ int cnt = 0;
  while (a)
        {
         if (a % 2) ++fr[cnt];
         ++cnt;
         a /= 2LL;
        }
}
```

```
void translatePositions()
{
    for (int i = 1; i < 90; ++i)
    {
        fr[i] += (fr[i - 1] / 2);
        fr[i - 1] %= 2;
    }
}

int main()
{
    int p, n;
    unsigned long long a, b, c, d;
    fin >> p >> n >> a >> b >> c >> d;
    decomposeNumber(a);

    unsigned long long limMax = ULLONG_MAX / b;

    for (int i = 2; i <= n; ++i)
    {
        a = (b * a) % d + c;
        decomposeNumber(a);
    }

    if (p == 1)
        translatePositions();

    int last = 0;

    for (int i = 0; i < 90; ++i)
        if (fr[i])
            last = i;

    bool ok = true;

    while (ok)
    {
        ok = false;
        for (int j = last; j > 0; --j)
        {
            if (fr[j] > fr[j - 1])
            {
                int diff = (fr[j] - fr[j - 1] + 2) / 3;
                fr[j] -= diff;
                fr[j - 1] += 2 * diff;
            }
            if (fr[j + 1] > fr[j])
                ok = true;
        }
    }
    fout << fr[0] << '\n';
    return 0;
}
```

Capitolul 9

Baraj selecție lot juniori ONI 2024

9.1 Problema Drei

Propusă de: prof. Ionel-Vasile Piț-Rada, Colegiul Național „Traian", Drobeta Turnu Severin

Arheologii au descoperit, printre alte vestigii ale unei civilizații dispărute, o reprezentare neobișnuită a numerelor pe care au denumit-o DREI. În reprezentarea DREI apar semne care au fost echivalate de arheologi cu literele mici din alfabetul englez. O *reprezentare DREI* este un șir de litere distincte care fie constă dintr-o singură literă, fie respectă următoarele două condiții:

1. litera maximă apare la unul dintre capete;
2. șirul obținut prin eliminarea literei maxime este o reprezentare DREI.

Primele douăsprezece reprezentări, corespunzătoare numerelor 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 sunt a, ab, b, ba, bac, bc, abc, ac, c, ca, cab, cb.
Să notăm cu $max(x)$ cea mai mare literă din reprezentarea DREI x.
Pentru a compara două reprezentări DREI notate cu x și respectiv y aplicăm, în ordine, următoarele reguli:

1. Dacă $max(x) > max(y)$, atunci $x > y$.
2. Reprezentarea formată dintr-o singură literă este mai mare decât orice altă reprezentare care are aceeași literă maximă în capătul drept și este mai mică decât orice altă reprezentare care are aceeași literă maximă în capătul stâng.
3. Dacă ambele reprezentări x și y au aceeași literă maximă poziționată la capătul drept, atunci se elimină litera maximă din ambele reprezentări și se compară reprezentările obținute (să le notăm x_{dr} și y_{dr}). Dacă $x_{dr} < y_{dr}$, atunci $x > y$, respectiv dacă $x_{dr} > y_{dr}$, atunci $x < y$.
4. Dacă ambele reprezentări x și y au aceeași literă maximă poziționată la capătul stâng, atunci se elimină litera maximă din ambele și se compară reprezentările obținute (să le notăm cu x_{st} și y_{st}). Dacă $x_{st} < y_{st}$, atunci $x < y$, respectiv dacă $x_{st} > y_{st}$, atunci $x > y$.

De exemplu:

- $edabc < edc$ (se aplică mai întâi regula 4, apoi se compară $dabc$ cu dc; se aplică din nou regula 4 și se compară abc cu c, apoi se aplică regula 2)
- $ab < b < ba$, conform regulii 2
- $abe < caf$, conform regulii 1
- $bac < cba$, conform regulii 2
- $bac < abc$ (se aplică mai întâi regula 3, apoi se compară ba cu ab, aplicând regula 2)

Cerințe

Scrieți un program care să determine răspunsurile pentru Q interogări de următoarele 4 tipuri:

Tip interogare	Răspuns
1 x	se afișează 1 dacă șirul x este o reprezentare DREI, respectiv 0 în caz contrar
2 x y	se compară reprezentările DREI x și y și se afișează -1 dacă $x < y$, 0 dacă $x = y$, respectiv 1 dacă $x > y$
3 N	se afișează cea de a N-a reprezentare DREI
4 x	se afișează numărul de ordine al reprezentării DREI x

Date de intrare

Fișierul de intrare `drei.in` conține pe prima linie numărul natural Q reprezentând numărul de interogări. Pe următoarele Q linii se află cele Q interogări, câte o interogare pe o linie, în formatul descris mai sus.

Date de ieșire

Fișierul de ieșire `drei.out` va conține Q linii. Pe linia i va fi scris răspunsul pentru cea de a i-a interogare din fișierul de intrare.

Restricții

- $1 \leq Q \leq 10^5$
- $1 \leq N \leq 10^{12}$
- Șirurile de caractere care apar în interogări au lungimea cel mult egală cu 26.
- Reprezentările DREI sunt numerotate în ordine crescătoare începând cu 1.

#	Puncte	Restricții
1	10	Toate interogările sunt de tip 1
2	13	Toate interogările sunt de tip 2
3	20	Toate interogările sunt de tip 3
4	20	Toate interogările sunt de tip 4
5	12	Șirurile din interogări sau cele obținute ca rezultat al interogărilor de tip 3 conțin cel mult primele 10 litere din alfabet și $Q < 10000$.
6	25	Nu există restricții suplimentare.

Exemple

drei.in	drei.out
6 1 acb 1 abcd 1 xabby 2 bac abc 3 12 4 cb	0 1 0 -1 cb 12

9.2 Rezolvarea problemei Drei

Observație inițială

fiecare cuvânt de o singură literă corespunde puterii 3^p, unde p este poziția literei în alfabet (începând de la 0): $p_a = 0, p_b = 1, \ldots, p_z = 25$.

Notăm cu literă(p) a p-a literă din alfabet (începând de la 0): literă(0) = a, literă(1) = b, ..., literă(25) = z. În plus, notăm cu || operația de concatenare între două șiruri de caractere (de exemplu: $a||b = ab$).

Termenul al n-lea, DREI(n), se determină unic astfel:

1. Dacă n este putere a lui 3 (n se poate scrie sub forma 3^p), atunci DREI(n) = literă(p).
2. Dacă n nu este putere a lui 3, atunci n se află între două puteri consecutive ($3^p < n < 3^{p+1}$) și, relativ la distanța față de puterea 3^{p+1}, avem două cazuri:
   (a) Dacă $2 \cdot n > 3^{p+1}$, atunci îl vom obține pe DREI(n) scăzând din 3^{p+1} diferența $d_{right} = 3^{p+1} - n$, adică vom așeza scrierea DREI(d_{right}) în stânga literei literă($p+1$), deci DREI(n) = DREI(d_{right}) || literă($p+1$)
   (b) Dacă $2 \cdot n < 3^{p+1}$, atunci îl vom obține pe DREI(n) adunând la 3^p diferența $d_{left} = n - 3^p$ și vom așeza scrierea DREI(d_{left}) la dreapta literei literă(p), deci DREI(n) = literă(p) || DREI(d_{left})

Exemplu: Vrem să aflăm DREI(59):

- Deoarece $27 < 59 < 81$ suntem în cazul 2-a). Obținem $d_{right} = 22$ și pentru DREI(59) așezăm scrierea DREI(22) în stânga literei literă(4), astfel:
  $$\text{DREI}(59) = \text{DREI}(22) \,||\, \text{literă}(4) = \text{DREI}(22) \,||\, e$$
- Deoarece $9 < 22 < 27$ suntem tot în cazul 2-a). Obținem $d_{right} = 5$ și pentru DREI(22) așezăm scrierea DREI(5) în stânga literei literă(3), astfel:
  $$\text{DREI}(22) = \text{DREI}(5) \,||\, \text{literă}(3) = \text{DREI}(5) \,||\, d$$
- Deoarece $3 < 5 < 9$ suntem tot în cazul 2-a). Obținem $d_{right} = 4$ și pentru DREI(5) așezăm scrierea DREI(4) în stânga literei literă(2), astfel:
  $$\text{DREI}(5) = \text{DREI}(4) \,||\, \text{literă}(2) = \text{DREI}(4) \,||\, c$$
- Deoarece $3 < 4 < 9$ suntem în cazul 2-b). Obținem $d_{left} = 1$ și pentru DREI(4) așezăm scrierea DREI(1) în dreapta literei literă(1), astfel:
  $$\text{DREI}(4) = \text{literă}(1) \,||\, \text{DREI}(1) = b \,||\, \text{DREI}(1)$$
- Deoarece $1 = 3^0$ suntem în cazul 1. Deci: DREI(1) = literă(0) = a. Astfel obținem:

$$\text{DREI}(1) = a$$
$$\text{DREI}(4) = b \,||\, \text{DREI}(1) = ba$$
$$\text{DREI}(5) = \text{DREI}(4) \,||\, c = bac$$
$$\text{DREI}(22) = \text{DREI}(5) \,||\, d = bacd$$
$$\text{DREI}(59) = \text{DREI}(22) \,||\, e = bacde$$

Interogare de tip 1 x

Să presupunem că șirul x are literele $x[0], x[1], x[2], \ldots, x[r-1]$.

Observăm că o secvență DREI nu poate conține aceeași literă de mai multe ori, iar litera maximă trebuie sa se afle într-unul dintre capete. Așa că, pornim o verificare simultan de la stânga și de la dreapta, verificând restricțiile de mai sus, iar apoi eliminând litera cea mai mare.

Algoritmul 1 Verificarea corectitudinii unui șir DREI

```
p ← 0
q ← r − 1
cât timp p + 1 ≤ q execută
    dacă x[p] ≤ x[p + 1] sau x[q − 1] ≥ x[q] atunci
        returnează false                          ▷ Litera maximă nu este la margini
    altfel dacă x[p] = x[q] atunci
        returnează false                          ▷ Literă duplicată
    altfel dacă x[p] > x[q] atunci
        p ← p + 1
    altfel
        q ← q + 1

returnează true
```

Interogare de tip 2 x y

Vom calcula în variabila rez rezultatul comparației dintre x și y.

- Dacă $rez = -1$, atunci $x < y$;
- Dacă $rez = 0$, atunci $x = y$;
- Dacă $rez = 1$, atunci $x > y$.

De asemenea, vom nota cu $max(x)$ cea mai mare literă din reprezentarea DREI x, iar cu $lungime(x)$ lungimea reprezentării DREI x.

Algoritmul 2 Compararea a două reprezentări DREI x și y

```
rez ← −∞                          ▷ Cât timp rez este setat pe −∞, nu știm rezultatul.
dacă x = y atunci
    rez ← 0
altfel
    cât timp rez = −∞ execută
        dacă max(x) < max(y) atunci
            rez ← −1                          ▷ x nu conține cea mai mare literă
        altfel dacă max(x) > max(y) atunci
            rez ← 1                           ▷ x conține cea mai mare literă
                ▷ Dacă ajungem în acest punct, înseamnă că x și y au aceeași literă maximă.
        lx ← lungime(x)
        ly ← lungime(y)
        dacă x[0] = max(x) și y[ly − 1] = max(y) atunci
            rez ← 1                           ▷ x are litera maximă la stânga, iar y nu
        altfel dacă x[lx − 1] = max(x) și y[0] = max(y) atunci
            rez ← −1                          ▷ x are litera maximă la dreapta, iar y nu
            ▷ Dacă ajungem în acest punct, înseamnă că litera maximă se află pe aceeași poziție
                ▷ (prima sau ultima) în x și y. Vom elimina litera din ambele reprezentări și
                                              ▷ comparăm ce rămâne.
        șterge max(x) din x
        șterge max(y) din y
```

Alternativ, se poate răspunde la această întrebare folosind algoritmul descris mai jos pentru interogările de tip 4, prin compararea numerelor de ordine asociate fiecărei reprezentări DREI.

Interogare de tip 3 N

Pe parcursul determinării literelor reprezentării DREI vom adăuga unui vector V perechi de forma (literă, direcție), unde direcție arată dacă litera va fi inserată la începutul (-1) sau la sfârșitul$(+1)$ șirului x.

Plecăm de la exemplul descris mai sus – DREI(59) = *bacde*:

$$\begin{gathered}\text{DREI}(59) = \text{DREI}(22) \mid\mid e \\ \text{DREI}(22) = \text{DREI}(5) \mid\mid d \\ \text{DREI}(5) = \text{DREI}(4) \mid\mid c \\ \text{DREI}(4) = b \mid\mid \text{DREI}(1) \\ \text{DREI}(1) = a\end{gathered}$$

Adăugăm în vector perechile: $(e, 1)$, $(d, 1)$, $(c, 1)$ și $(b, -1)$.

Algoritmul 3 Determinarea reprezentării DREI asociate numărului N

```
1: cât timp N nu este putere a lui 3 execută
2:     calculăm p                                   ▷ Avem 3^p < N < 3^(p+1)
3:     dacă 2 · N > 3^(p+1) atunci
4:         adăugăm în vectorul V perechea (p + 1, 1)
5:         înlocuim N cu 3^(p+1) − N
6:     altfel dacă 2 · N ≤ 3^(p+1) atunci
7:         adăugăm în vectorul V perechea (p, −1)
8:         înlocuim N cu N − 3^p
```

Avem $N = 3^p$. Inserăm în x litera literă(p) apoi parcurgem în ordinea inversă în care au fost adăugate perechile în V și dacă avem perechea (literă, direcție) în funcție de direcție adăugăm litera la începutul sau sfârșitul șirului x format până la momentul respectiv.

Alternativ, reconstituirea soluției se poate face folosind o coadă pentru literele din stânga, respectiv o stivă pentru literele din dreapta.

Interogare de tip 4 x

Vom folosi și aici un vector V pentru reconstituirea soluției.

Algoritmul 4 Determinarea reprezentării DREI asociate numărului N

```
1: p ← max(x) − 'a'
2: Nr ← 3^p
3: cât timp lungime(x) > 1 execută
4:     p ← max(x) − 'a'
5:     dacă x[0] = max(x) atunci
6:         adaugă în V perechea (3^p, 1)
7:     altfel
8:         adaugă în V perechea (3^p, −1)
9:     șterge max(x) din x;
```

Se parcurg în sens invers perechile din V și pentru fiecare pereche se adaugă sau se scade puterea curentă din puterea perechii anterioare, la final se adună/scade ultimul rezultat la/din Nr.

Alternativ, se poate implementa recursiv: (de exemplu, ID_DREI($abcd$) = 27 − ID_DREI(abc)).

9.3 Cod-sursă pentru problema Drei

```
#include <fstream>
#include <cstring>
using namespace std;
ifstream  fin("drei.in");
ofstream fout("drei.out");
int c,Q;
long long N;

struct drei{
    char x[30];
}x,y;

drei nr2drei(long long n){
    drei a;
    long long t=1,ss=1;
    int p=0;
    while(ss<n){
        p++;
        t=t*3;
        ss=ss+t;
    }
    ss=ss-t;
    ///t=3^p<=n
    ///sirul are litera maxima 'a'+p
    char c='a'+p;
    if(t==n){
        a.x[0]=c;
        a.x[1]=0;
        return a;
    }
    if(n<t){
        a=nr2drei(t-n);
        int l=strlen(a.x);
        a.x[l]=c; a.x[l+1]=0;
        return a;
    }
    a=nr2drei(n-t);
    char s[30];
    strcpy(s,a.x);
    strcpy(a.x+1,s);
    a.x[0]=c;
    return a;
}

long long drei2nr(drei x){
    char c=max(x.x[0],x.x[strlen(x.x)-1]);
    int k=c-'a';
    long long p=1,ss=1;
    for(int i=1;i<=k;i++){
        p=p*3;
        ss=ss+p;
    }
    if(strlen(x.x)==1)return p;
```

```
    if(x.x[0]==c){
        drei a;
        strcpy(a.x,x.x+1);
        return p+drei2nr(a);
    }
    else{
        drei a=x;
        a.x[strlen(x.x)-1]=0;
        return p-drei2nr(a);
    }
}

int main(){
    fin>>Q;
    for(int q=1;q<=Q;q++){
        fin>>c;
        if(c==1){
            fin>>x.x;
            int ok=1;
            int f[200]={0};
            for(int i=0;x.x[i] && ok;i++){
                f[x.x[i]]++;
                if(f[x.x[i]]==2){
                    ok=0;
                }
            }
            int p=0,r=strlen(x.x)-1;
            char cmax1=max(x.x[p],x.x[r]),cmax2;
            while(ok && p<r){
                if(x.x[p]==cmax1){
                    p++;
                }
                else{
                    r--;
                }
                cmax2=max(x.x[p],x.x[r]);
                if(cmax2>cmax1){
                    ok=0;
                    break;
                }
                cmax1=cmax2;
            }
            fout<<ok<<"\n";
        }
        if(c==2){
            fin>>x.x>>y.x;
            long long rx=drei2nr(x);
            long long ry=drei2nr(y);
            if(rx>ry)fout<<1<<"\n";
            else if(rx<ry)fout<<-1<<"\n";
            else fout<<0<<"\n";
        }
        if(c==3){
            fin>>N;
            x=nr2drei(N);
            fout<<x.x<<"\n";
        }
        if(c==4){
            fin>>x.x;
            long long z=drei2nr(x);
            fout<<z<<"\n";
```

```
        }
    }
    return 0;
}
```

9.4 Problema Soldați

Propusă de: prof. Marinel Șerban, Colegiul Național „Emil Racoviță" Iași
prof. Adrian Panaete, Colegiul Național „August Treboniu Laurian" Botoșani

Cei S soldați ai unei unități militare au un moment de relaxare și sunt răspândiți pe platou în poziții cunoscute. Evident, ei nu stau aliniați, în poziție de drepți. Platoul poate fi reprezentat printr-un tablou bidimensional cu n linii (numerotate de la 1 la n) și m coloane (numerotate de la 1 la m), o poziție în tablou fiind identificată prin indicele liniei și respectiv al coloanei. Când se anunță sosirea comandantului unității militare, cei S soldați trebuie să se alinieze. Alinierea presupune așezarea celor S soldații pe o aceeași linie L, pe coloane consecutive. Ca să nu se creeze haos, fiecare dintre ei se va deplasa numai pe orizontală și verticală. Dacă soldatul i ($1 \leq i \leq S$) se află inițial în poziția (L_i, C_i) iar la aliniere ajunge în poziția (Lin, Col), distanța pe care el o parcurge va fi $|L_i - Lin| + |C_i - Col|$.

Cerințe

Scrieți un program care să rezolve următoarele două cerințe:

1. determină o linie Lin, astfel încât pentru alinierea soldaților pe pozițiile 1, 2, $\cdots$, S de pe linia Lin, suma distanțelor parcurse de aceștia să fie minimă;
2. determină o linie Lin și o coloană Col, astfel încât pentru alinierea soldaților pe linia Lin pe coloanele $Col, Col + 1, \ldots, Col + S - 1$, suma distanțelor parcurse de aceștia să fie minimă.

Date de intrare

Fișierul de intrare `soldati.in` conține pe prima linie 4 valori naturale C n m S, separate prin câte un spațiu, reprezentând cerința care trebuie să fie rezolvată, numărul de linii, numărul de coloane, respectiv numărul de soldați. Următoarele S linii conțin pozițiile celor S soldați, câte un soldat pe o linie, pozițiile fiind descrise prin două numere naturale L C separate printr-un spațiu, reprezentând linia, respectiv coloana.

Date de ieșire

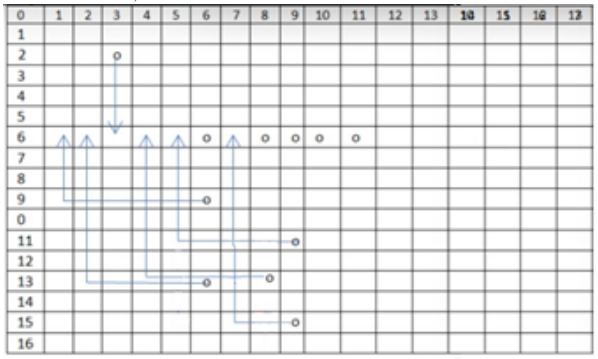
Fișierul de ieșire `soldati.out` va conține pe prima linie un număr natural $Dmin$ reprezentând distanța totală minimă parcursă pentru ca soldații să se alinieze conform cerinței C din fișierul de intrare. Dacă $C = 1$ pe cea de a doua linie va fi scris un număr natural Lin reprezentând linia pe care se face alinierea. Dacă $C = 2$, pe cea de-a doua linie se vor scrie două numere naturale separate prin spațiu Lin Col, reprezentând linia pe care se va face alinierea, respectiv coloana de la care soldații încep să se alinieze. În cazul în care există mai multe linii pe care soldații se pot alinia parcurgând distanță totală minimă, se va afișa cea mai mică dintre ele. Pentru $C = 2$, în cazul în care există mai multe coloane Col începând cu care soldații se pot alinia pe linia Lin parcurgând distanță totală minimă, se va afișa cea mai mică dintre ele.

Restricții

- $2 \leq n, m \leq 10^6$
- $1 \leq S \leq \min(m, 10^5)$
- Pozițiile inițiale ale soldaților sunt distincte.

#	Puncte	Restricții
1	5	Soldații se află inițial pe aceeași linie.
2	14	$2 \leq n, m \leq 1\,500$
3	12	$C = 1$, $1500 < n, m \leq 10\,000$
4	33	$C = 1$, fără alte restricții
5	12	$C = 2$, $1500 < n, m \leq 10\,000$
6	24	$C = 2$, fără alte restricții

Exemple

soldati.in	soldati.out	Explicații
1 16 17 11 2 3 6 6 6 8 6 9 6 10 6 11 9 6 11 9 13 8 13 6 15 9	54 6	Mai jos este reprezentat platoul și modul în care soldații se rearanjează 
2 16 17 11 2 3 6 6 6 8 6 9 6 10 6 11 9 6 11 9 13 8 13 6 15 9	46 6 3	2 3->6 3 4 pași pe coloana 3 6 6->6 4 2 pași pe linia 6 6 8->6 7 1 pas pe linia 6 6 9->6 9 0 pași (stă pe loc) 6 10->6 12 2 pași pe linia 6 6 11->6 13 2 pași pe linia 6 9 6->6 5 3 pași pe coloana 6 + 1 pas pe linia 6 11 9->6 10 5 pași pe coloana 9 + 1 pas pe linia 6 13 8->6 8 7 pași pe coloana 8 13 6->6 6 7 pași pe coloana 6 15 9->6 11 9 pași pe coloana 9 + 2 pași pe linia 6

9.5 Rezolvarea problemei Soldați

Prima observație

Pentru rezolvarea problemei putem trata separat deplasările celor S soldați pe cele două direcții obținând astfel două probleme.

Problema 1: Alinierea soldaților pe o linie LIN

Considerăm soldații sortați în ordine crescătoare după linie ($L_1 \leq L_2 \leq \cdots \leq L_s$). Formal, dorim să determinăm o linie LIN pentru care valoarea $|LIN - L_1| + |LIN - L_2| + \ldots |LIN - L_s|$ să fie minimă.

Problema 2: Alinierea soldaților pe linia LIN începând cu o coloană COL

Considerăm soldații sortați în ordine crescătoare după coloană $C_1 \leq C_2 \leq \cdots <= C_S$. Vrem să așezăm soldații pe coloanele $COL, COL+1, \ldots, COL+S-1$. Observăm inițial că așezarea Greedy (primul din stânga se deplasează pe coloana COL, al doilea pe coloana $COL+1$, ..., ultimul pe coloana $COL+S-1$) este optimă pentru o alegere fixată a valorii COL.

În acest context vrem să alegem COL astfel încât valoarea: $|COL - C_1| + |COL + 1 - C_2| + |COL + 2 - C_3| + \cdots + |COL + S - 1 - C_S|$ să fie minimă. Se observă că această problemă este similară cu Problema 1, ca și cum am vrea să aducem pe linia $LIN = COL$ soldați situați la liniile $C_1, C_2 - 1, \ldots, C_S - S + 1$.

Pentru cerința 1 se fixează $COL = 1$.

Pentru cerința 2 trebuie să avem grijă la două situații speciale:

- Dacă aplicând algoritmul de la problema 1 obținem $COL < 1$, trebuie să alegem $COL = 1$. În acest caz algoritmul ne-ar arăta că alinierea optimă s-ar realiza dacă șirul de soldați „ar ieși din cazarmă prin partea stângă".
- Dacă aplicând algoritmul de la problema 1 obținem $COL + S - 1 > m$ trebuie să alegem COL astfel încât $COL + S - 1 = m$, adică să alegem $COL = m - S + 1$. În acest caz, algoritmul ne-ar arăta că alinierea optimă s-ar realiza dacă șirul de soldați „ar ieși din cazarmă prin partea dreaptă".

Astfel, am redus problema 2 la problema 1. Ne ocupăm în continuare de rezolvarea problemei 1. Determinăm LIN astfel încât : $DIST(LIN) = |LIN - L_1| + |LIN - L_2| + \cdots + |LIN - L_S|$ are valoare minimă.

Rezultat ajutător: Fie $a \leq b$. Atunci $|X - a| + |X - b| \geq |(X - a) - (X - b)| = |b - a| = b - a$. Egalitatea are loc dacă $a \leq X \leq b$.

În $DIST(LIN)$ grupăm: primul termen cu ultimul, al doilea cu penultimul și așa mai departe.

$|LIN - L_1| + |LIN - L_S| >= L_S - L_1$ cu egalitate pentru $L_1 \leq LIN \leq L_S$

$|LIN - L_2| + |LIN - L_{S-1}| >= L_{S-1} - L_2$ cu egalitate pentru $L_2 \leq LIN \leq L_S - 1$

...

Și așa mai departe.

Dacă S este par ($S = 2 \cdot k$) atunci $DIST(LIN) \geq L_S - L_1 + L_{S-1} - L_2 + \cdots + L_{k+1} - L_k$ și egalitatea se obține când $L_k \leq LIN \leq L_{k+1}$. Deoarece vrem ca LIN să fie minimă se alege $LIN = L_k$.

Dacă S este impar ($S = 2 \cdot k + 1$) atunci $DIST(LIN) \geq L_S - L_1 + L_{S-1} - L_2 + \cdots + L_k - L_{k+1} + |L - L_{k+1}| \leq L_S - L_1 + L_{S-1} - L2 + \cdots + L_k - L_{k+1}$ cu egalitate dacă sunt îndeplinite simultan condițiile $L_k \leq LIN \leq L_{k+2}$ și $LIN = L_{k+1}$. Se observă că valoarea minimă se obține numai dacă $LIN = L_{k+1}$.

Cele două cazuri ne conduc la rezultatul unic $LIN = L_{(S+1)/2}$, adică LIN trebuie ales mediana șirului.

9.6 Cod-sursă pentru problema Soldați

```
#include <fstream>
#include <algorithm>
#define NMAX 100002

using namespace std;
ifstream fin("soldati.in");
ofstream fout("soldati.out");
int n, L, m, cerinta, C, S;
long long int rez;
int lin[NMAX];
pair<int,int> col[NMAX];
int modul (int x)
{if (x<0) return -x;
 return x;
}
bool compardif(pair<int,int> a, pair<int,int> b)
{
  return (a.first-a.second<b.first-b.second ||
          a.first-a.second==b.first-b.second && a.second<b.second);
}
int main()
{int i, coloana1, mij;
 fin>>cerinta>>n>>m>>S;
 for (i=1; i<=S; i++) {fin>>lin[i]>>col[i].first;}
 sort(lin+1, lin+S+1);

 mij=(S+1)/2;
 L=lin[mij];
 for (i=1; i<=mij; i++) rez+=L-lin[i];
 for (i=mij+1; i<=S; i++) rez+=lin[i]-L;
 sort(col+1,col+S+1);
 for (i=1; i<=S; i++) col[i].second=i;
 if (cerinta==1) coloana1=1;
     else
     {
      sort(col+1,col+S+1,compardif);
      coloana1 = max(1, col[mij].first - col[mij].second + 1);
      coloana1 = min(coloana1, m - S + 1);
      sort(col + 1, col + S + 1);
     }
 for (i=1; i<=S; i++) rez+=modul(col[i].first-(coloana1+i-1));
 fout<<rez<<'\n';
 if (cerinta==1) fout<<L<<'\n';
    else fout<<L<<' '<<coloana1<<'\n';
 fout.close();
 return 0;
}
```

9.7 Problema Tramvaie

Propusă de: stud. Dumitru Ilie, Facultatea de Matematică-Informatică, Universitatea București

Un grup de olimpici plănuiesc să viziteze orașul Ortogonal situat în RUDA (Regatul Unit al celor Două Axe). Ori de câte ori planifică o călătorie, ei studiază hotelurile și restaurantele din oraș. Din experiențele trecute, ei știu că vor mânca foarte multă pizza și că întoarcerea de la restaurant la hotel va fi dificilă. Prin urmare, pe harta orașului Ortogonal trasează un sistem de coordonate cartezian și identifică pe hartă hotelurile și restaurantele.
Străzile din orașul Ortogonal sunt fie paralele cu axa Ox, fie paralele cu axa Oy, distanța dintre oricare două străzi paralele consecutive fiind egală cu 1. Dacă ei se deplasează pe jos din punctul de coordonate (x_1,y_1) în punctul de coordonate (x_2,y_2) distanța parcursă va fi $|x_1 - x_2| + |y_1 - y_2|$. Însă atunci când ești ghiftuit cu pizza, vrei să mergi pe jos cât mai puțin. De aceea olimpicii noștri vor utiliza pentru deplasare tramvaiele.
Liniile de tramvai se află pe unele dintre străzi, ca urmare și ele sunt fie paralele cu Ox, fie paralele cu Oy. În orașul Ortogonal nu există stații, tramvaiul poate fi luat de oriunde de pe o linie de tramvai și se poate coborî din tramvai oriunde pe linia de tramvai.

Cerințe

Cunoscând amplasarea liniilor de tramvai în orașul Ortogonal, scrieți un program care să răspundă la Q întrebări de forma următoare: „Care este distanța minimă care trebuie să fie parcursă pe jos pentru a ne deplasa din punctul de coordonate (x_1,y_1) în punctul de coordonate (x_2,y_2)?”.

Date de intrare

Fișierul de intrare `tramvaie.in` conține pe prima linie numerele naturale N, M și Q, care reprezintă numărul de linii de tramvai paralele cu Ox, numărul de linii de tramvai paralele cu Oy, respectiv numărul de întrebări. Pe a doua linie se găsesc N numere întregi care reprezintă ordonatele la care se găsesc liniile de tramvai paralele cu Ox. Pe a treia linie se găsesc M numere întregi care reprezintă abscisele la care se găsesc liniile de tramvai paralele cu Oy. Pe următoarele Q linii se găsesc câte 4 numere întregi x_1 y_1 x_2 y_2, reprezentând coordonatele celor două puncte pentru care trebuie să determinăm distanța minimă care trebuie să fie parcursă pe jos pentru a ajunge dintr-un punct în celălalt. Valorile scrise pe aceeași linie sunt separate prin câte un spațiu.

Date de ieșire

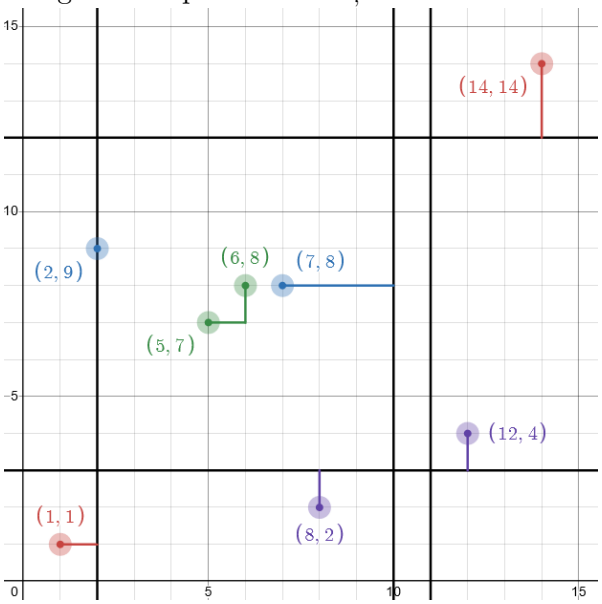
Fișierul de iesire `tramvaie.out` va conține Q linii. Pe linia i se află un singur număr natural, reprezentând răspunsul la cea de a i-a întrebare din fișierul de intrare.

Restricții

- $1 \leq N, M, Q \leq 100\,000$
- Coordonatele liniilor de tramvai și ale punctelor între care ne deplasăm sunt numere întregi cuprinse între 0 și 10^9.
- Nu vor exista două linii de tramvai paralele cu Ox cu aceeași ordonată.
- Nu vor exista două linii de tramvai paralele cu Oy cu aceeași abscisă.
- Este recomandat să nu utilizați x_1 sau y_1 ca nume de variabile.

#	Puncte	Restricții
1	12	Toate numerele din fișierul de intrare sunt cel mult 1000
2	24	$N, M, Q \leq 1000$
3	7	$N = M = 1$
4	57	Fără restricții suplimentare

Exemple

tramvaie.in	tramvaie.out	Explicații
2 3 4 3 12 10 11 2 1 1 14 14 2 9 7 8 8 2 12 4 6 8 5 7	3 3 2 2	În figură este prezentat orașul: 

Explicație

Între punctele roșii (prima pereche de puncte din fișierul de intrare) distanța minimă parcursă pe jos este 3. Aceasta se poate obține mergând spre est din punctul $(1, 1)$ până la linia de tramvai paralelă cu Oy situată la abscisa 2 (distanța parcursă pe jos fiind 1), apoi cu tramvaie până la punctul de coordonate $(14, 12)$ de unde se merge spre nord pe jos până la $(14, 14)$.
Între punctele albastre distanța minimă este tot 3. Aceasta se poate obține mergând cu tramvaiul din punctul $(2, 9)$ (care se află pe o linie de tramvai) până la punctul de coordonate $(10, 8)$ și apoi pe jos spre vest până la punctul de coordonate $(7, 8)$.
Pentru punctele mov distanța minimă este 2 și se poate obține mergând pe jos spre nord până la linia de tramvai paralelă cu Ox aflată la ordonata 3, apoi cu tramvaiul până la punctul de coordonate $(12, 3)$ și apoi pe jos spre nord până la punctul de coordonate $(12, 4)$.
Pentru punctele verzi, drumul în care se merge cel mai puțin pe jos este cel în care nu folosim niciun tramvai. Lungimea acestuia este $|6 - 5| + |8 - 7| = 2$.

9.8 Rezolvarea problemei Tramvaie

Pentru a determină distanța minimă care poate fi parcursă pe jos între două puncte de coordonate (x_0,y_0) și (x_1,y_1) analizăm două cazuri posibile, minimul dintre distanțele determinate pe cele două cazuri fiind rezultatul final.

Cazul 1.

Pentru deplasare nu se folosesc tramvaie. În acest caz răspunsul este $|x_1 - x_0| + |y_1 - y_0|$. Această formulă este cunoscută drept distanța Manhattan.

Cazul 2.

Pentru deplasare se folosesc tramvaie. În acest caz trebuie să determinăm distanța de la punctul de coordonate (x_0,y_0) la cel mai apropiat tramvai, respectiv distanța de la punctul de coordonate (x_1,y_1) la cel mai apropiat tramvai, suma acestor două distanțe fiind distanța totală parcursă pe jos între cele două puncte.

Pentru a determina distanța de la un punct la cel mai apropiat tramvai, determinăm distanța minimă de la punctul respectiv la o linie de tramvai paralelă cu Ox, respectiv distanța minimă de la punctul respectiv la o linie de tramvai paralelă cu Oy, rezultatul fiind minimul dintre cele două distanțe.

Pentru aceasta vom sorta liniile de tramvai paralele cu Ox, respectiv liniile de tramvai paralele cu Oy. Determinarea celei mai apropiate linii de tramvai (paralelă cu Ox, respectiv cu Oy) se poate face prin căutare binară. Complexitatea finală este $O((N+Q) \cdot \log_2 N + (M+Q) \cdot \log_2 M)$ datorată sortărilor și căutărilor binare.

O altă variantă posibilă ar fi să sortăm și punctele după abscisă, respectiv după ordonată și să determinăm determinăm cea mai apropiată linie de tramvai paralelă cu Ox, respectiv cu Oy printr-un algoritm similar cu algoritmul de interclasare. Complexitatea acestei soluții este $O(N \cdot \log_2 N + M \cdot \log_2 M + Q \cdot \log_2 Q)$, datorată sortărilor.

9.9 Cod-sursă pentru problema Tramvaie

```
#include <fstream>
#include <algorithm>
#define NMAX 100004

using namespace std;
ifstream fin("tramvaie.in");
ofstream fout("tramvaie.out");
int n, m, q;
int tv[NMAX];
int to[NMAX];
int modul (int x) {if (x>=0) return x; return -x;}
int dmin;
int cauta(int t[], int n, int v);
int main()
{int i, x1, y1,x2,y2,do1,do2,dv1,dv2;
 fin>>n>>m>>q;
 for (i=1; i<=n; i++) fin>>to[i];
 for (i=1; i<=m; i++) fin>>tv[i];
 sort(to+1,to+n+1);
 sort(tv+1,tv+m+1);
 for (i=0; i<q; i++)
     {fin>>x1>>y1>>x2>>y2;
      dmin=modul(x1-x2)+modul(y1-y2);
      do1=cauta(to,n,y1);
      dv1=cauta(tv,m,x1);
      do2=cauta(to,n,y2);
      dv2=cauta(tv,m,x2);
      if (dmin>min(do1,dv1)+min(do2,dv2))
         dmin=min(do1,dv1)+min(do2,dv2);
      fout<<dmin<<'\n';
```

```
 }
 return 0;
}

int cauta(int t[], int n, int v)
//cautare binara
{int st=0, dr=n+1, mij;
 while (dr-st>1)
       {
        mij=(st+dr)/2;
        //invariant t[st]<v<=t[dr]
        if (t[mij]<v) st=mij;
            else dr=mij;
       }
 if (dr==n+1) return v-t[st];
 if (st==0) return t[dr]-v;
 if (t[dr]-v<v-t[st]) return t[dr]-v;
 return v-t[st];
}
```

Partea a III-a

Tabăra de pregătire a lotului național de informatică juniori

Cluj, 10-15 mai 2024

Capitolul 10

Barajul 1

10.1 Problema Bug

Propusă de: prof. Emanuela Cerchez, Colegiul Național „Emil Racoviță" Iași

Robotul Vasile trebuie să lipească etichete pe produse care vin pe o bandă rulantă. Eticheta produsului va conține codul acestuia care este un număr natural. În mod normal, produsele ar trebui să fie codificate cu numerele naturale consecutive începând cu 1, în ordine crescătoare. Dar robotul Vasile are un bug: nu generează numere naturale care conțin ca subsecvență numărul natural X.

O subsecvență este formată din cifre situate pe poziții consecutive în număr. De exemplu, dacă $X = 213$, atunci numerele naturale 213, 1213, 2131, 2132 sau 721389 nu vor fi generate, deoarece conțin pe 213 ca subsecvență. Dar numărul 25136 va fi generat, deoarece cifrele 213 nu apar pe poziții consecutive.

Cerințe

Date fiind X și numărul de produse care vin pe bandă N, scrieți un program care să determine codul care va fi pe eticheta ultimului produs.

Date de intrare

Fișierul de intrare `bug.in` conține pe prima linie numerele naturale X N, separate prin spațiu, având semnificația din enunț.

Date de ieșire

Fișierul de ieșire `bug.out` va conține o singură linie pe care va fi scris codul de pe eticheta ultimului produs de pe bandă (cel de al N-lea).

Restricții

- $0 < X < 100\,000$
- Se garantează că cifrele numărului X sunt distincte.
- $0 < N \leq 10^{16}$

#	Puncte	Restricții
1	13	$1 \leq N \leq 10^6$
2	19	$10^6 < N$ și $X < 10$
3	24	$10^6 < N$ și $10 \leq X < 100$
4	44	$10^6 < N$ și $100 \leq X \leq 10^5$

Exemple

bug.in	bug.out	Explicații
3 13	15	$X = 3$ și $N = 13$. Codul de pe eticheta ultimului produs va fi 15, deoarece codurile generate sunt 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15.

10.2 Rezolvarea problemei Bug

Soluția 1 - prof. Emanuela Cerchez

Pasul 1.

Extragem cifrele numărului X și le plasăm în vectorul cx, de lungime lgx, începând cu unitățile la poziția 1.

Pasul 2.

Precalculăm următoarele valori:

- $s[i]$=numărul de numere de exact i cifre care nu conțin ca subsecvență pe X (unde i variază de la 1 la $LGMAX = 17$);
- $sum[i] = s[1]+s[2]+\cdots+s[i]$=numărul de numere de cel mult i cifre care nu conțin ca subsecvență pe X.

```
s[0]=1;  s[1]=9;
for (i=2; i <= lgx; i++) s[i]=s[i-1]*10;
s[lgx]--;
for (i=lgx+1; i < LGMAX; i++)
     {s[i]=10*s[i-1]-s[i-lgx];}
sum[0]=1;
for (i=1; i < LGMAX; i++) sum[i]=sum[i-1]+s[i];
```

Pasul 3.

Rezolvăm problema prin căutare binară pe rezultat.

```
st=0; dr=N*10+1;
while (dr-st > 1)
       {
        mij = (st + dr) / 2;
        nr = numar(mij);
        if (nr < N) st = mij;
           else dr=mij;
        }
```

Funcția $numar(n)$ determină numărul de valori $\leq n$ care nu conțin ca subsecvență pe X.

Pasul 4.

Funcția $numar(n)$ extrage cifrele numărului n și le plasează în vectorul $cifre$ de lungime lg; parcurge cifrele lui n începând cu cifra dominantă (de la dreapta către stânga) și analizează dacă secvența de cifre de lungime lgx care începe la poziția curentă este mai mare, mai mică sau egală cu X. În cazul în care este mai mică, pentru poziția curentă orice cifră mai mică decât cea curentă va conduce la o soluție corectă, deci actualizăm rezultatul astfel:

```
rez=rez+cifre[i]*sum[i-1];
```

În cazul în care este mai mare, trebuie să analizăm următoarele lgx cifre și pentru fiecare să actualizăm rezultatul adunând numărul de soluții care se obțin cu cifre mai mici decât cifra curentă, dar să scădem soluțiile generate cu prefixul lui X care începe la poziția curentă, astfel:

```
for (j=lgx; j>0; j--)
    {scad=egal(j-lgx+i);
     if (scad>0)
        {rez=rez+(cifre[j-lgx+i])* sum[j-lgx+i-1]-sum[j-lgx+i-lgx];}
        else
        rez=rez+cifre[j-lgx+i]* sum[j-lgx+i-1];
    }
    i=i-lgx+1;
```

În caz de egalitate procedăm similar ca în cazul mai mare, numai că încheiem numărarea la parcurgerea celor lgx cifre.

Pasul 5.

Din păcate, căutarea binară pe rezultat va genera cel mai mic număr natural pentru care numărul de numere mai mici sau egale cu el care nu conțin ca subsecvență pe X este N, dar nu este garantat că acest număr nu va avea ca subsecvență pe X. Vom corecta, eventual, rezultatul obținut în urma căutării binare, parcurgându-l începând cu unitățile și, atunci când identificăm o apariție a lui X, adunăm 1 la cifra unităților acestei apariții a lui X.

Soluția 2 - prof. Adrian Panaete

Se calculează răspunsul cifră cu cifră. În acest scop se precalculează:

1. $cnt[L][P]$, $1 \leq L \leq |N|+1$; $0 \leq P < |X|$, cu semnificația: câte numere valide au L cifre și au prefix de lungime P comun cu numărul invalid X.
   Observație: Dacă un număr valid are prefix P atunci el se numără și pentru prefixele $P-1$, $P-2$, ..., 1, 0.
   Astfel $cnt[L][0]$ poate fi interpretat drept numărul tuturor valorilor valide de L cifre.
2. $CNT[L][P][C]$, $0 \leq C \leq 9$, cu semnificația: câte dintre cele $cnt[L][P]$ numere valide descrise mai sus se obțin prin adăugarea la prefixul de lungime P a cifrei C

Se observă ca valorile $cnt[L][P]$ depind doar de $|X|$, nu și de valoarea cifrelor lui X. Astfel se obțin următoarele formule de calcul:

- pentru $L < |X|$:
  - $cnt[L][L] = 1$
  - $cnt[L][P] = 10 \cdot cnt[L][P+1]$

- pentru $L \geq |X|$ și $P > 0$:
  - $cnt[L][P] = 8 \cdot cnt[L-P-1][0] + cnt[L-P][1] + cnt[L][P+1]$
- pentru $L \geq |X|$ și $P = 0$:
  - $cnt[L][0] = 9 \cdot cnt[L-1][0] + cnt[L][1]$

Termenul $8 \cdot cnt[L-P-1][0]$ se referă la situația când după prefixul de lungime P adăugăm o cifră care nu este nici prima cifră din X și nici cifra care prelungește prefixul de lungime P la prefixul de lungime $P+1$.

Termenul $cnt[L-P][1]$ se referă la situația când după prefixul de lungime P adăugăm prima cifră din X.

Termenul $cnt[L][P+1]$ se referă la situația când după prefixul de lungime P adaugăm a $(P+1)$-a cifră din X.

Termenul $9 \cdot cnt[L-1][0]$ se referă la situația când numărul de lungime L începe cu orice cifră cu excepția primei cifre din X.

Termenul $cnt[L][1]$ se referă la situația când numărul de lungime L începe cu prima cifră din X.

Valorile $CNT[L][P][C]$ reprezintă termenii din descompunerea lui $cnt[L][P]$ în funcție de cifra C care se va adăuga prefixului de lungime P, considerat deja cunoscut. Aceste valori depind de cifrele numărului X și se vor calcula în funcție de P astfel:

- dacă $cnt[L][P] = 0$:
  - $CNT[L][P][C] = 0$ pentru orice cifră C
- dacă $P = 0$:
  - $CNT[L][P][C] = cnt[L][1]$ dacă C este prima cifră din X
  - $CNT[L][P][C] = cnt[L-1][0]$ dacă C nu este prima cifră din X
- dacă $P = |X| - 1$:
  - $CNT[L][P][C] = 0$ dacă C este ultima cifră din X
  - $CNT[L][P][C] = cnt[L-P][1]$ dacă C este prima cifră din X
  - $CNT[L][P][C] = cnt[L-P-1][0]$ pentru celelalte cifre
- dacă $0 < P < |X| - 1$:
  - $CNT[L][P][C] = cnt[L][P+1]$ dacă C este cifra de pe poziția $P+1$ din X
  - $CNT[L][P][C] = cnt[L-P][1]$ dacă C este prima cifra din X
  - $CNT[L][P][C] = cnt[L-P-1][0]$ pentru celelalte cifre

Odată ce am făcut precalculările vom determina soluția cifră cu cifră începând cu cifra de ordin maxim folosind următoarea strategie:

Detectăm pe rând cele cel mult 18 cifre. Între aceste cifre pot avea eventual și cifre 0 nesemnificative.

Să notăm cu R numărul de cifre care au mai rămas de detectat.

Știm deja prefixul PR de lungime $18 - R$ al soluției și vrem să detectăm următoarea cifră.

Pentru PR știm și numărul de cifre P din sufixul lui PR și care sunt prefix pentru X.

Rezultatul va fi un număr situat între două valori:

1. Cea inferioară LO = cele $18 - R$ cifre determinate urmate de R cifre de 0
2. Cea superioară HI = cele $18 - R$ cifre determinate urmate de R cifre de 9

Pentru acest context știm:

1. $offset < n$, unde $offset$ = numărul de valori valide strict mai mici decât LO
2. numărul căutat are valoarea intre LO și HI

Se consideră pe rând cele 10 cifre în ordine crescătoare și prelungim prefixul cu fiecare.

Numerele valide între LO și HI vor fi distribuite astfel:

- primele $NR0 = CNT[R+P][P][0]$ au prefixul PR continuat cu cifra 0
- următoarele $NR1 = CNT[R+P][P][1]$ au prefixul PR continuat cu cifra 1
- următoarele $NR2 = CNT[R+P][P][2]$ au prefixul PR continuat cu cifra 2
- ...
- ultimele $NR9 = CNT[R+P][P][9]$ au prefixul PR continuat cu cifra 9

Astfel putem poziționa valorile valide cu prefixul PR în funcție de cifra c pe care o adăugam la PR pe intervale $[STc, DRc]$:

- $c = 0 : [ST0, DR0]$, unde $ST0 = offset + 1$ și $DR0 = offset + NR0$
- $c = 1 : [ST1, DR1]$, unde $ST1 = DR0 + 1$ și $DR1 = DR0 + NR1$
- $c = 2 : [ST2, DR2]$, unde $ST2 = DR1 + 1$ și $DR2 = DR1 + NR2$
- ...
- $c = 9 : [ST9, DR9]$, unde $ST9 = DR8 + 1$ și $DR9 = DR8 + NR9$

Cifra corecta c este cea pentru care n este situat în intervalul $[STc, Drc]$. După detectarea fiecărei cifre c:

- Adaugăm cifra c la PR
- Actualizăm $offset$ la $STc - 1$
- Actualizăm valoarea lui P în funcție de cifra c adăugată.

Soluția 3 - instr. Cătălin Frâncu, Nerdvana București

Cea de a treia sursă comentată implementează o variantă mai generală pentru problema bug, care funcționează și pentru cazul în care cifrele lui X nu sunt distincte.

10.3 Cod-sursă pentru problema Bug

```
//Solutia 1
#include <fstream>
#define LGMAX 17
using namespace std;
ifstream fin("bug.in");
ofstream fout("bug.out");
int X, lgx, lg, p10=1;
long long int N;
long long int s[LGMAX];
long long int sum[LGMAX];
//s[i]=numarul de numere de i cifre care nu contin ca subsecventa pe X
int cx[LGMAX];
int cifre[LGMAX];
long long int bun(long long int c);
long long int numar(long long int n);

int main()
{int i, c;
 long long int st, dr, mij, nr, nrb;
 fin>>X>>N;

 c=X; lgx=0; while (c) {lgx++; p10*=10; cx[lgx]=c%10; c/=10;}
 s[0]=1; s[1]=9;
```

```
 for (i=2; i<=lgx; i++) s[i]=s[i-1]*10;
 s[lgx]--;
 for (i=lgx+1; i<LGMAX; i++)
      {s[i]=10*s[i-1]-s[i-lgx];}
 sum[0]=1;
 for (i=1; i<LGMAX; i++) sum[i]=sum[i-1]+s[i];

 //cautare binara pe rezultat
 st=0; dr=N*10+1;
 while (dr-st>1)
       {
        mij = (st + dr) / 2;
        nr = numar(mij);
        if (nr < N)
            st = mij;
            else
            dr=mij;
        }
 //cel mai mic dr pentru care se obtine N, dar nu este sigur corect, poate contine cifra X
 dr=bun(dr);
 fout<<dr<<'\n';
 return 0;
}

bool egalst(int i)
{int j;
 for (j=1; j<=lgx && i+j-1<=lg; j++)
     if (cx[j]!=cifre[i+j-1]) return 0;
 if (j>lgx) return 1;
 return 0;
}

long long int bun(long long int c)
{long long int rez=0;
 int i, t;
 lg=0;
 while (c)
       {cifre[++lg]=c%10;
        c/=10;
       }
 for (t=0, i=1; i<=lg; i++)
     {cifre[i]=cifre[i]+t; t=0;
     if (egalst(i))
         cifre[i]++;
     if (cifre[i]>9) {t=1; cifre[i]-=10;}
     }
if (t) {cifre[++lg]=t;}
for (i=lg; i>0; i--) rez=rez*10+cifre[i];
return rez;
}

int egal (int i)
//verific daca cifrele incepand cu pozitia i sunt egale cu X
{int j;
 int rez=0;
 for (j=lgx; j>0 && j-lgx+i>0; j--)
      rez=rez*10+cifre[j-lgx+i];
 if (rez==X) return 0;
 if (rez<X) return -1;
 return 1;
}
```

```
long long int numar(long long int n)
//numarul de valori <=n care nu contin ca subsecventa pe X
{long long int rez=0;
 int i, j, scad, c;
 lg=0;
 while (n)
       {cifre[++lg]=n%10;
        n/=10;
       }
 for (i=lg; i>0; i--)
       {scad=egal(i);
        if (scad<0)
            rez=rez+cifre[i]*sum[i-1];
            else
            {if (scad==0)
               {rez=rez+cifre[i]*sum[i-1];
                for (j=lgx-1; j>0; j--)
                     if (cifre[j-lgx+i]<=cifre[i]) rez=rez+cifre[j-lgx+i]* sum[j-lgx+i-1];
                        else
                        if (j-lgx+i-lgx>=0)
                           rez=rez+cifre[j-lgx+i]*sum[j-lgx+i-1]-sum[j-lgx+i-lgx];
                           else
                           rez=rez+cifre[j-lgx+i]*sum[j-lgx+i-1];

                return rez-1;
               }
            for (j=lgx; j>0; j--)
                {scad=egal(j-lgx+i);
                if (scad>0)
                    {rez=rez+(cifre[j-lgx+i])* sum[j-lgx+i-1]-sum[j-lgx+i-lgx];}
                    else
                    rez=rez+cifre[j-lgx+i]* sum[j-lgx+i-1];
                }
            i=i-lgx+1;
           }
        }
  return rez;
}
```

```
//Solutia 2
#include <bits/stdc++.h>

using namespace std;
ifstream f("bug.in");
ofstream g("bug.out");
const int DEBUG=1;
int m,used[20][10];
int64_t n,x,val,sol,nr[20][10],X[20];
int64_t P[18];
int64_t getNr(int C,int P)
{
 if(used[C][P])return nr[C][P];
 used[C][P]=1;
 nr[C][P]=(C<P||P==m)?0:C==P?1:P?getNr(C-P-1,0)*8+getNr(C-P,1)+getNr(C,P+1):
                                getNr(C-1,0)*9+getNr(C,1);
 // calculez cate numere valide au C cifre dintre care cel putin P sunt din prefixul lui X
 // C<P nr[C][P]=0 pentru ca nu pot sa am prefix mai lung decat numarul de cifre
 // P==m nr[C][P]=0 pentru ca nu am vois sa ma prefix de lungime m
 // C==P nr[C][P]=1
 // P==0 nr[C][P]=getNr(C-1,0)*9+getNr(C,1);return nr[C][P]
```

```
 // pentru ca pot continua cu o cifra diferita de prima 9 posibilitati
 // sau pot continua cu prima cifra
 // if(P==m-1){nr[C][P]=getNr(C-P-1,0)*8+getNr(C-P,1);return nr[C][P];}
 // P>0 nr[C][P]=getNr(C-P-1,0)*8+getNr(C-P,1)+getNr(C,P+1);
 // pot continua cu o cifra diferita de prima din X si de cea care urmeaza in X
 // pot continua cu prima cifra din X
 // pot continua cu cifra care urmeaza in X
 return nr[C][P];
}
void adaugaCifra(int L,int P,int C)
{
 int64_t cnt;
 if (L==P){g<<sol<<'\n'; exit(0);}
 // nu ma mai intereseaza decat ultimele L cifre ale rezultatului
 // dintre acestea primele P sunt deja adaugate si formeaza un prefix de lungime P din
 // stiu sigur ca ce am adaugat pana aici este corect
 // vreau sa vad daca e corect sa adaug cifra C
 if (P==0) cnt=C==X[1]?nr[L][1]:nr[L-1][0];
     else
     cnt=C==X[1]?nr[L-P][1]:C==X[P+1]?nr[L][P+1]:nr[L-P-1][0];
 if (cnt<n) {n-=cnt;adaugaCifra(L,P,C+1);}
 sol=10*sol+C;
 if (P==0)
    {
     if (C==X[1]) adaugaCifra(L,1,0);
         else adaugaCifra(L-1,0,0);
    }
    else
    {
     if (C==X[1]) adaugaCifra(L-P,1,0);
        else
        if (C==X[P+1]) adaugaCifra(L,P+1,0);
           else adaugaCifra(L-P-1,0,0);
    }
}
int main()
{
 f>>x>>n;n++;
 for (int val=x;val;val/=10) X[++m]=val%10;
 reverse(X+1,X+m+1);
 getNr(17,0);
//precalculez nr[L][P]=cate secvente valide de L cifre au prefix comun cu X de lungime cel mult P
 adaugaCifra(17,0,0);
 return 0;
}
```

```
//Solutia 3
#include <stdio.h>
#define MAX_DIGITS_X 5
#define MAX_DIGITS_ANSWER 17
#define BASE 10

// δ[S][C] = Cîte cifre de la începutul lui X se potrivesc la finalul șirului
// obținut din primele S cifre din X concatenate cu cifra C?
// Exemplu: X = 67678. Atunci δ[3][7] = 2, pentru că dacă luăm primele 3 cifre
// ale lui X (676) și le concatenăm cu cifra 7, atunci obținem 6767, care se
// termină cu 67, deci primele două cifre din X se potrivesc.
int delta[MAX_DIGITS_X + 1][BASE];

// c[L][S] = Cîte numere care NU îl conțin pe X există care au L cifre, știind
// că anterior acestor L cifre am plasat primele S cifre din X?
```

```
//
// Exemplu: X = 1234. Atunci c[7][2] este numărul de numere de 7 cifre
// (0000000-9999999) care nu încep cu 34 și nu conțin 1234.
long long c[MAX_DIGITS_ANSWER + 1][MAX_DIGITS_X + 1];

int x, digits_x[MAX_DIGITS_X + 1], len_x;
long long n;

void read_input_data() {
  FILE* f = fopen("bug.in", "r");
  fscanf(f, "%d %lld", &x, &n);
  fclose(f);
}

void reverse_array(int* v, int size) {
  for (int i = 1, j = size; i < j; i++, j--) {
    int tmp = v[i]; v[i] = v[j]; v[j] = tmp;
  }
}

void split_digits_of_x() {
  len_x = 0;

  while (x) {
    digits_x[++len_x] = x % BASE;
    x /= BASE;
  }

  reverse_array(digits_x, len_x);
}

// Testează dacă primele @state cifre din X, concatenate cu @digit, se termină
// cu primele @len cifre din X.
bool prefix_matches_suffix(int state, int digit, int len) {
  int i = 1;
  while ((i < len) && (digits_x[i] == digits_x[i + (state + 1 - len)])) {
    i++;
  }

  return (i == len) && (digits_x[len] == digit);
}

int longest_prefix_which_is_also_suffix(int state, int digit) {
  int len = (state == len_x) ? state : (state + 1);

  while (len && !prefix_matches_suffix(state, digit, len)) {
    len--;
  }

  return len;
}

// Calculează matricea δ naiv, în len_x^3 * BASE. Cormen prezintă și o
// implementare în O(len_x * BASE). Un algoritm și mai bun (KMP) nu mai
// folosește deloc matricea, ci doar un vector calculabil în O(len_x).
void compute_delta_matrix() {
  for (int state = 0; state <= len_x; state++) {
    for (int digit = 0; digit < BASE; digit++) {
      delta[state][digit] = longest_prefix_which_is_also_suffix(state, digit);
    }
  }
```

```
}

long long sum_of_possible_continuations(int len, int state) {
  if (state == len_x) {
    return 0;
  } else if (len == 0) {
    return 1;
  } else {
    long long result = 0;
    for (int digit = 0; digit < BASE; digit++) {
      int next_state = delta[state][digit];
      result += c[len - 1][next_state];
    }
    return result;
  }
}

void compute_c_matrix() {
  for (int len = 0; len <= MAX_DIGITS_ANSWER; len++) {
    for (int state = 0; state <= len_x; state++) {
      c[len][state] = sum_of_possible_continuations(len, state);
    }
  }
}

long long find_nth_number_not_containing_mask() {
  long long result = 0;
  long long need_to_skip = n;
  int state = 0;

  for (int remaining = MAX_DIGITS_ANSWER; remaining > 0; remaining--) {
    int digit = 0;
    bool can_advance = true;

    while (can_advance) {
      int next_state = delta[state][digit];
      long long would_skip = c[remaining - 1][next_state];

      if (would_skip <= need_to_skip) {
        need_to_skip -= would_skip;
        digit++;
      } else {
        can_advance = false;
      }
    }

    state = delta[state][digit];
    result = BASE * result + digit;
  }
  return result;
}

void write_answer(long long answer) {
  FILE* f = fopen("bug.out", "w");
  fprintf(f, "%lld\n", answer);
  fclose(f);
}

int main() {
  read_input_data();
  split_digits_of_x();
```

```
  compute_delta_matrix();
  compute_c_matrix();
  long long answer = find_nth_number_not_containing_mask();
  write_answer(answer);
  return 0;
}
```

10.4 Problema Nooverlap

Propusă de: prof. Dan Pracsiu, Liceul Teoretic „Emil Racoviță" Vaslui

Fie o matrice cu 5 linii și N coloane ce memorează numere întregi și un număr natural nenul K.

Cerințe

Să se determine suma maximă care se poate obține alegând exact K submatrici 2×2 care nu se suprapun și însumând elementele din cele K submatrici alese.

Date de intrare

Fișierul de intrare `nooverlap.in` conține pe prima linie numerele naturale N și K separate prin spațiu. Următoarele 5 linii conțin fiecare câte N numere întregi separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire `nooverlap.out` va conține pe prima linie un singur număr natural reprezentând suma maximă cerută.

Restricții

- $2 \leq N \leq 10\,000$
- $1 \leq K \leq 100$
- $-1000 \leq$ valorile din matrice ≤ 1000
- O matrice 2×2 aleasă nu out poate suprapune cu o alta aleasă anterior și de asemenea trebuie să fie complet în interiorul matricii $5 \times N$.
- Se garantează că pentru toate testele se pot alege K submatrici care nu se suprapun.

#	Puncte	Restricții
1	15	$2 \leq N \leq 20$ și $1 \leq K \leq 5$
2	15	$N > 20$ și pentru a obține suma maximă se pot selecta K matrici de pe primele două linii.
3	70	Fără restricții suplimentare

Exemple

nooverlap.in	nooverlap.out	Explicații
6 3 9 3 1 1 2 4 2 8 1 8 9 -1 0 0 0 -1 8 0 6 7 1 1 2 3 4 5 1 1 1 1	68	Matricea dată are 5 linii și 6 coloane și trebuie să alegem 3 submatrici 2×2 care nu se suprapun. Prima submatrice aleasă: 9 3 2 8 A doua: 6 7 4 5 A treia: 8 9 -1 8 Suma totală va fi $22 + 22 + 24 = 68$.

10.5 Rezolvarea problemei Nooverlap

Să presupunem că în matricea a de dimensiuni $5 \times n$ de mai jos am ales patru submatrici care sunt marcate cu 1, iar restul componentelor sunt zero:

$$a = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Observăm din acest exemplu că pe orice coloană submatricile 2×2 pot ocupa zero, două sau patru elemente. Avem deci următoarele posibilități de ocupare a unor elemente pe o coloană:

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Mai sus, prima coloană (să o numim configurație de tip 0) este plină de 0, deci nu este utilizat niciun element de pe ea. A doua coloană (să o numim configurație de tip 1) are primele două elemente ocupate. Ultima coloană (o numim configurație de tip 7) are ocupate ultimele patru elemente.

În continuare, construim rezolvarea cu programare dinamică. Vom utiliza o matrice dp tridimensională $k \times n \times 8$, în care:

$dp[i][j][p]$ = suma maximă obținută din primele j coloane, s-au ales deja i submatrici 2×2, iar pe ultima linie este configurația dată de valoarea lui p ($p = 0 \ldots 7$ fiind tipul de coloană)

Relațiile de recurență

Cum calculăm $dp[i][j][0]$? Pentru că $p = 0$, atunci de pe coloana curentă j a matricii a nu alegem niciun element, deci $dp[i][j][0]$ va depinde de valorile obținute deja până pe coloana $j - 1$, deci:

$dp[i][j][0] = \max(dp[i][j-1][p],\ p \in \{0, 1, 2, 3, 4, 5, 6, 7\})$

Cum calculăm pe $dp[i][j][1]$? Pentru că $p = 1$, atunci înseamnă că primele două elemente de pe coloana j sunt ocupate de o jumătate de matrice 2×2, iar cealaltă jumătate de submatrice se află pe coloana anterioară $j - 1$. Acest lucru înseamnă că această submatrice 2×2 ocupă componentele $a[0][j-1]$, $a[1][j-1]$, $a[0][j]$, $a[1][j]$, deci $dp[i][j][1]$ depinde de maximul dintre $dp[i-1][j-1][0]$, $dp[i-1][j-1][3]$, $dp[i-1][j-1][4]$, la care se adaugă suma submatricii 2×2.

Similar se determină și $dp[i][j][2]$, $dp[i][j][3]$, $dp[i][j][4]$.

Să vedem cum se determină $dp[i][j][5]$, când avem primele patru valori de pe coloana j ocupate de două submatrice 2×2, care ocupă în consecință și primele patru poziții de pe coloana $j - 1$. Deci $dp[i][j][5]$ depinde doar de $dp[i-2][j-1][0]$ (pentru că pentru a putea alege cele două submatrici 2×2 este necesar să fie disponibilă coloana $j - 1$), la care se adaugă suma elementelor din cele două submatrici 2×2.

Asemănător cu $dp[i][j][5]$ se tratează și $dp[i][j][6]$ și $dp[i][j][7]$.

Inițializarea matricei dp se poate face cu o valoare foarte mică, sau, crescând toate valorile din matricea inițială a cu 1000, dp se poate inițializa cu 0.

Soluția problemei va fi în $\max(dp[k][n][p],\ p \in \{0, 1, 2, 3, 4, 5, 6, 7\})$

Complexitatea este $O(n \times k \times \Sigma)$, unde $\Sigma = 8$.

O soluție alternativă care nu obține decât punctaj parțial este un algoritm de tip Greedy. Luăm toate submatricile 2×2, fiecare submatrice fiind dată prin coordonatele colțului stânga-sus și suma elementelor submatricii. Sortăm descrescător submatricile după sumă, apoi luăm primele k submatrici care nu se suprapun. Când alegem o submatrice, într-o matrice marcăm cu 1 cele 2×2 poziții ocupate.

Soluția Greedy nu furnizează rezultatul corect în multe situații. De exemplu dacă avem $k = 4$ și matricea $5 \times n$ este de forma:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 4 & 4 & 4 & 4 & 1 & 1 & 1 \\ 1 & 1 & 4 & 5 & 5 & 4 & 1 & 1 & 1 \\ 1 & 1 & 4 & 5 & 5 & 4 & 1 & 1 & 1 \\ 1 & 1 & 4 & 4 & 4 & 4 & 1 & 1 & 1 \end{bmatrix}$$

atunci dacă alegem submatricea formată din cele patru valori de 5, atunci nu mai putem alege cele patru submatrice care au câte trei de 4 și un 5.

Pentru 15 puncte se poate implementa un algoritm de tip backtracking.

O altă soluție de punctaj maxim fructifică observația că valoarea k este foarte mică față de valoarea n. Astfel, vom păstra din matricea inițială doar coloanele din care putem selecta cele k configurații care asigură alegerea sumei maxime. Vor fi selectate cel mult câte $2 \times k$ coloane pentru fiecare tip de configurație. Pentru a face această selecție se poate construi câte o listă ordonată descrescător care conține $n - 1$ perechi de forma (coloană, sumă configurație) din care selectăm primele $2 \times k$ perechi sau putem utiliza Σ cozi cu prioritate. Apoi reconstruim o parte din matricea inițială folosind doar coloanele selectate și pe noua construcție aplicăm algoritmul de programare dinamică prezentat anterior. Complexitatea va fi acum $O((n \times log + k^2) \times \Sigma)$

10.6 Cod-sursă pentru problema Nooverlap

```
#include <fstream>
#define NMAX 10002
#define KMAX 102
#define CFMAX 8
#define INF -1000000000
using namespace std;
ifstream fin("nooverlap.in");
```

```
ofstream fout("nooverlap.out");
int n, k;
int a[6][NMAX];
int smax[NMAX][KMAX][CFMAX];
//smax[i][t][cf]=suma maxima pe care o pot obtine selectand t submatrice 2x2 de pe
//liniile i, i+1, ...n, pe linia i configuratia fiind cf
/* 01234567
   01000110
   01100111
   00110101
   00011111
   00001011
*/

int c[CFMAX][CFMAX+1]={{8,0,1,2,3,4,5,6,7},{3,0,3,4},{2,0,4},{2,0,1},{3,0,1,2},{1,0},{1,0},{1,0}};
int nr[CFMAX]={0,1,1,1,1,2,2,2};

int suma(int i, int cf)
{int rez=0, j;
 if (cf==0) return 0;
 if (cf<5)
    {rez=a[cf][i]+a[cf+1][i]+a[cf][i+1]+a[cf+1][i+1]; return rez;}
 for (j=1; j<=5; j++) rez+=a[j][i]+a[j][i+1];
 if (cf==5)
    return rez-a[5][i]-a[5][i+1];
 if (cf==6)
    return rez-a[3][i]-a[3][i+1];
 return rez-a[1][i]-a[1][i+1];
}

int main()
{int i, j, h, t, cf, maxim,cfmax,s;
 fin>>n>>k;
 for (i=1; i<=5; i++)
     for (j=1; j<=n; j++) fin>>a[i][j];
 for (t=1; t<=k; t++)
     for (cf=0; cf<CFMAX; cf++)
          smax[n][t][cf]=smax[n-1][t][cf]=INF;
 for (cf=1; cf<8; cf++) smax[n-1][0][cf]=INF;
 for (cf=1; cf<5; cf++) smax[n-1][1][cf]=suma(n-1,cf);
 for (cf=5; cf<8; cf++) smax[n-1][2][cf]=suma(n-1,cf);
 for (i=n-2; i>=1; i--)
     for (t=1; t<=k; t++)
         for (cf=0; cf<CFMAX; cf++) //determin smax[i][t][cf]
               {maxim=INF; s=suma(i,cf);
                for (j=1; j<=c[cf][0]; j++)
                    if (t-nr[cf]>=0 && smax[i+1][t-nr[cf]][c[cf][j]]!=INF &&
                        s+smax[i+1][t-nr[cf]][c[cf][j]]>maxim)
                        maxim=s+smax[i+1][t-nr[cf]][c[cf][j]];
                smax[i][t][cf]=maxim;
               }
maxim=INF;
for (cf=0; cf<CFMAX; cf++)
    if (maxim<smax[1][k][cf])
        {maxim=smax[1][k][cf]; cfmax=cf;}
fout<<maxim<<'\n';
fout.close();
return 0;
}

#include <bits/stdc++.h>
```

```
#define NEW A[nou]
#define OLD A[vechi]
using namespace std;
ifstream f("nooverlap.in");
ofstream g("nooverlap.out");
const int mInf = -500000;
const int N = 10010;
int A[2][6][105],n,k,b,vechi,nou=1,v[5][N],a[6],sol;
void getCol(int j)
{
 for (int i=1;i<=4;i++) a[i]=v[i-1][j-1]+v[i][j-1]+v[i-1][j]+v[i][j];
 a[5]=max(a[1]+a[3],max(a[1]+a[4],a[2]+a[4]));
}
int main()
{
  f>>n>>k;
  for (int i=0;i<5;i++)
      for (int j=1;j<=n;j++) f>>v[i][j];
  for (int i=0;i<6;i++)
       for (int j=0;j<=k+1;j++) OLD[i][j]=NEW[i][j]=mInf;
  getCol(2);
  OLD[0][0]=0;
  for (int i=1;i<=4;i++)OLD[i][1]=a[i];OLD[5][2]=a[5];
  for (int col=3;col<=n;col++,vechi^=1,nou^=1)
      {
       getCol(col);
       for (int lg=0;lg<=k;lg++)
           {
            NEW[0][lg]=OLD[0][lg];
            for (int i=1;i<=4;i++) NEW[i][lg+1]=OLD[0][lg]+a[i];
            NEW[5][lg+2]=OLD[0][lg]+a[5];
            }
        for (int i=1;i<=5;i++)
            for (int lg=0;lg<=k;lg++)
                NEW[0][lg]=max(NEW[0][lg],OLD[i][lg]);
        for (int i=1;i<=2;i++)
            for (int j=i+2;j<=4;j++)
                for (int lg=1;lg<=k;lg++)
                    {
                    NEW[i][lg+1]=max(NEW[i][lg+1],OLD[j][lg]+a[i]);
                    NEW[j][lg+1]=max(NEW[j][lg+1],OLD[i][lg]+a[j]);
                    }
       }
   sol=OLD[0][k];
   for (int i=1;i<=5;i++) sol=max(sol,OLD[i][k]);
   g<<sol;
   return 0;
}
```

10.7 Problema Pase

Propusă de: prof. Ciprian Chește, Liceul Tehnologic „Grigore C. Moisil" Buzău

Fotbaliștii echipei naționale se antrenează pe un teren de formă dreptunghiulară. Pe acest teren au fost trasate L linii orizontale și C linii verticale, astfel încât între oricare două linii distanța să fie egală cu 1 metru. Liniile au fost numerotate de sus în jos cu valori de la 1 la L, iar coloanele au fost numerotate de la stânga la dreapta cu valori de la 1 la C și tot terenul este format din celule pătratice de latura 1. Intersecția dintre o linie și o coloană o vom denumi punct. Un punct este specificat prin linia și coloana la intersecția cărora se află.

Fotbaliștii pot trimite pase doar dintr-un punct către un alt punct. O pasă poate trece prin **unul sau mai multe puncte**. Un punct prin care trece o pasă se va numi **punct pasabil**.

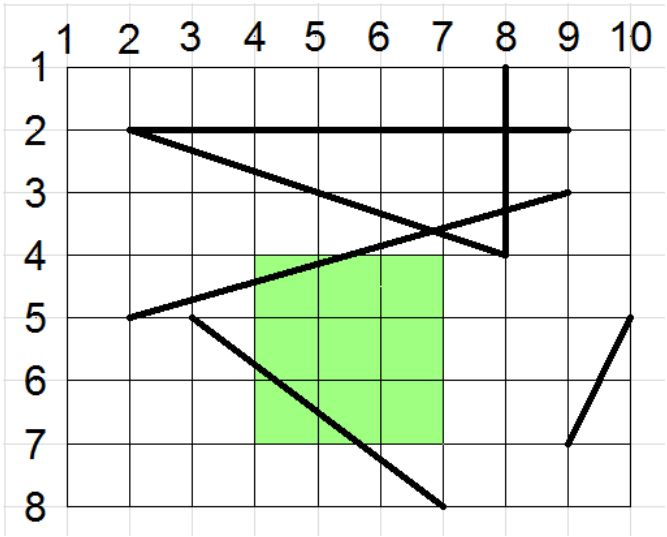
Antrenorul echipei de fotbal dorește realizarea unei aplicații care să analizeze pasele jucătorilor dintr-un meci pentru a îmbunătăți calitatea paselor date de fotbaliști de la un meci la altul.

Cerințe

Cunoscând P pase, specificate fiecare prin punctul de start și punctul de final, să se determine zonele **pătratice** din teren, de **arie maximă și nenulă**, care nu conțin în interior sau pe margini niciun punct pasabil.

Exemplu: $L = 8, C = 10, P = 6$ și pasele:

$(2,2) \to (2,9); (2,2) \to (4,8); (5,3) \to (8,7); (1,8) \to (4,8); (5,2) \to (3,9); (7,9) \to (5,10)$ atunci există o singură zonă pătratică de arie maximă $= 9$, care nu conține niciun punct pasabil, colorată în figură.



Date de intrare

Fișierul de intrare `pase.in` conține:

- pe prima linie numerele naturale L și C, cu semnificația din enunț;
- pe a doua linie numărul natural P, reprezentând numărul de pase;
- pe următoarele P linii sunt descrise cele P pase, câte o pasă pe o linie, fiecare pasă fiind descrisă prin patru numere naturale a b c d, unde a b reprezintă linia, respectiv coloana punctului de start, iar c d linia, respectiv coloana punctului de final ($1 \leq a, c \leq L$, $1 \leq b, d \leq C$).

Numerele aflate pe aceeași linie a fișierului sunt separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire `pase.out` va conține, pe prima linie, numărul Z de zone de arie pătratică maximă determinate și pe a doua linie aria A unei astfel de zone. Pe următoarele Z linii zonele determinate, câte o zonă pe o linie; o zonă va fi specificată prin 4 numere naturale separate prin spațiu, reprezentând coordonatele punctului (linie, coloană) din colțul stânga-sus al zonei și coordonatele punctului (linie, coloană) din colțul dreapta-jos al zonei. Afișarea se va face în ordinea crescătoare a liniilor colțului din stânga-sus al zonei pătratice, iar pentru două linii egale în ordinea crescătoare a coloanei.

Restricții

- $1 \leq L, C \leq 1\,000$
- $1 \leq P \leq 1\,000$
- Se pot trimite pase de la un punct la același punct (cu mult efect!)
- Se garantează că pentru datele de test există soluție de arie nenulă.

#	Puncte	Restricții
1	12	Fișierul de intrare conține doar pase orizontale.
2	8	Fișierul de intrare conține doar pase verticale.
3	4	Fișierul de intrare conține doar pase pentru care punctul de start și cel final coincid.
4	16	$1 \leq L, C \leq 100$; fișierul de intrare conține și pase oblice
5	60	Fără restricții suplimentare

Exemple

pase.in	pase.out	Explicații
8 10 6 2 2 2 9 2 2 4 8 5 3 8 7 1 8 4 8 5 2 3 9 7 9 5 10	1 9 4 4 7 7	Datele din fișierul de intrare corespund imaginii precedente. Există o singură zonă pătratică de arie maximă egală cu 9, având colțul din stânga-sus $(4, 4)$ și colțul din dreapta-jos $(7, 7)$

10.8 Rezolvarea problemei Pase

Cuvinte cheie:

- Programare dinamică
- Puncte laticiale
- Cel mai mare divizor comun
- Asemănarea figurilor geometrice
- Sume parțiale în matrice

Soluția I (sume parțiale în matrice) $O(L \cdot C \cdot \min(L, C))$

- În prima parte a rezolvării problemei se analizează fiecare pasă și într-o matrice cu L linii și C coloane, pe care s-o denumim matricea paselor, având toate valorile inițiale nule, se marchează cu 1 toate punctele laticiale prin care trece o pasă.

Pentru a rezolva această subproblemă se calculează numărul de puncte laticiale prin care trece o pasă cu expresia $np = cmmdc(|l2 - l1|, |c2 - c1|) + 1$, unde $(l1, c1)$ și $(l2, c2)$ sunt coordonatele între care se trimite pasa. Atenție că există și un caz particular și anume situația când pasa se dă „pe loc" adică $l1 = l2$ și $c1 = c2$.
Apoi se determină punctele laticiale prin care trece pasa rezolvând mai multe cazuri diferite în funcție de așezarea punctelor $(l1, c1)$ și $(l2, c2)$, cu alte cuvinte direcția pasei poate fi către unul dintre cele patru puncte cardinale sau pe diagonale. Problema revine acum la a afla care este cea mai mare suprafață pătratică de elemente de zero din matricea paselor.

- În a doua etapă se calculează matricea sumelor parțiale a matricei paselor.
- În a treia etapă se parcurge matricea sumelor parțiale și pentru fiecare punct laticial de coordonate (l,c) se calculează suma elementelor dintr-o suprafață pătratică. Dacă această sumă este zero, adică zona pătratică este formată doar din elemente de 0, atunci se actualizează lungimea laturii maxime dacă este cazul.
  În funcție de implementare această etapă poate fi realizată în $O(LC)^2)$ sau $O(LCmin(L, C))$
- În ultima etapă se parcurge încă odată matricea sumelor parțiale, de această dată cunoscând lungimea laturii maxime se memorează într-o structură corespunzătoare toate zonele cu lungimea laturii maxime determinate anterior, ale cărei elemente, la final se afișează în fișierul de ieșire.

Observații:

- Are mare importanță cum se construiește matricea sumelor parțiale. Acest subalgoritm se poate realiza în $O(L^2C^2)$ sau $O(LC(L + C))$ sau $O(LC)$.
- O modalitate eficientă de construcție a acestei matrice se realizează pe baza valorilor din pozițiile $(i - 1, j)$,$(i, j - 1)$ și $(i - 1, j - 1)$ cât și a valorii din matricea inițială de pe poziția (i, j).

Soluția II (programare dinamică) $O(L \cdot C)$

La fel ca la soluția anterioară se analizează fiecare pasă și într-o matrice având L linii și C coloane, pe care s-o denumim matricea paselor (MP), având toate valorile inițiale egale cu 1, se marchează cu 0 toate punctele laticiale prin care trece o pasă.

Se ține desigur cont de direcția din care vine pasa și de cazul particular al „paselor pe loc".

Pentru a determina cea mai mare zonă pătratică având doar elemente de 1 se procedează astfel:

- Se utilizează o matrice auxiliară $aux[L][C]$ care se construiește folosind algoritmul:

Algoritmul 5 Construcția matricei aux

```
1: pentru i ← 1...L execută
2:     pentru j ← 1...C execută
3:         dacă MP[i][j] = 1 atunci
4:             aux[i][j] ← min(MP[i][j − 1], MP[i − 1][j], MP[i − 1][j − 1]) + 1
5:         altfel
6:             aux[i][j] ← 0
```

- Se determină maximul dintre elementele lui aux.
- Se parcurge încă odată matricea aux și cu maximul determinat anterior se rețin toate zonele pătratice de arie maxima și coordonate lor într-o structură de date, care ulterior se afișează.

Soluție alternativă - prof. Ionel Vasile Pițˇ Rada

Se construiește matricea cu valoarea 1 la pozițiile pasabile și valoarea zero în rest. Se construiește matricea sumelor parțiale. Apoi se caută binar cea mai mare latură pentru care avem o zonă pătrată cu suma valorilor egală cu zero, adică să nu conțină poziții pasabile.

10.9 Cod-sursă pentru problema Pase

```
// programare dinamica O(LC)
#include <bits/stdc++.h>
#define nmax 1000
using namespace std;

struct punct
{
    int l,c;
};

struct drept
{
    int l1,c1,l2,c2;
};

ifstream fin("pase.in");
ofstream fout("pase.out");

int teren[nmax + 1][nmax + 1];
int spmat[nmax + 1][nmax + 1];

drept v[nmax*nmax + 1];

int cmmdc(int a,int b)
{
    if (a==0 && b==0) return 1;
    else
        if (b==0) return a;
            else return cmmdc(b,a%b);
}

void afisare(int L, int C, int x[nmax + 1][nmax + 1])
{
    for(int i = 1;i <= L; i++)
        {
        for(int j = 1; j <= C; j++)
            fout << x[i][j] <<" ";
        fout << "\n";
        }
    fout <<"\n";
}

int main()
{
    int P;              // numarul de pase
    punct p1,p2;        // punctul de unde pleaca pasa, punctul unde ajunge pasa
    int L,C;            // dimensiuni teren
    int i,j,nrp,nri;    // nrp=nr de puncte laticiale prin care trece o pasa, nri=nr de intervale
    int nrmax;          // nr maxim de zero dintr-o zona
    int dl,dc;          // deplasarea pe verticala sau orizontala
    drept a;            // variabila necesara adaugarii zonelor gasite intr-un vector de drept.
    int k = 0;          // indicatorul vectorului de zone patratice determinate
    // citire date de intrare
    fin >> L >> C;
    fin >> P;
    for(i = 1;i <= L;i++)
        for(j = 1;j <= C;j++)
            teren[i][j] = 1;
    for(i = 1; i <= P; i++)
```

```
    {
        fin >> p1.l >> p1.c >> p2.l >> p2.c;
        // marchez pe teren punctele laticiale prin care trece o pasa
        nrp = 1 + cmmdc(abs(p2.l - p1.l),abs(p2.c - p1.c));
        nri = nrp - 1;
        dl = abs(p2.l - p1.l)/nri;
        dc = abs(p2.c - p1.c)/nri;
        if (p1.l <= p2.l && p1.c <= p2.c) {dl = dl; dc = dc;}
        else if (p1.l <= p2.l && p1.c > p2.c) {dl = dl; dc = -dc;}
                else if (p1.l > p2.l && p1.c <= p2.c) {dl = -dl; dc = dc;}
                    else {dl = -dl;dc = -dc;}

        for(j = 0; j < nrp; j++)
               teren[p1.l+j*dl][p1.c+j*dc] = 0;
    }

    // determin latura celui mai mare patrat de elemente de 1 din matrice
    for(i = 1;i <= L;i++) spmat[i][1] = teren[i][1];
    for(j = 1;j <= C;j++) spmat[1][j] = teren[1][j];

    nrmax = 0;
    for(i = 2; i <= L; i++)
       for(j = 2; j <= C; j++)
         if(teren[i][j] == 1)
                {
                spmat[i][j] = min(spmat[i-1][j-1], min(spmat[i-1][j], spmat[i][j-1])) + 1;
                if(spmat[i][j] > nrmax)
                            nrmax = spmat[i][j];
                }
    // parcurg din nou matricea si retin zonele de arie maxima
    for(i = 1; i <= L; i++)
       for(j = 1; j <= C; j++)
            if(spmat[i][j] == nrmax && nrmax > 1)
                    {
                    a.l1 = i - nrmax + 1;a.c1 = j - nrmax + 1;
                    a.l2 = i;a.c2 = j;
                    v[++k] = a;
                    }

    // afisare rezultate
    if (k == 0) fout << 0 <<"\n";
    else
        {
        fout << k <<"\n";
        fout << (nrmax - 1)*(nrmax - 1)<<"\n";
        for(i = 1;i <= k;i++)
            fout << v[i].l1 <<" "<<v[i].c1 << " " <<v[i].l2 <<" " <<v[i].c2 << "\n";
        }
    return 0;
}
```

```
//prof. Ionel Vasile Pit-Rada
#include <fstream>
using namespace std;
ifstream  fin("pase.in");
ofstream fout("pase.out");
int L,C,P,A[1002][1002],B[1002][1002],a1,b1,c1,d1;
struct quadruplu{
    int a1,b1,c1,d1;
}v[1000002];
int cmmdc(int a, int b){
```

```
    if(b==0)return a;
    return cmmdc(b,a%b);
}
int main(){
    fin>>L>>C>>P;
    for(int p=1;p<=P;p++){
        fin>>a1>>b1>>c1>>d1;
        if(a1>c1){
            swap(a1,c1);
            swap(b1,d1);
        }
        if(b1<=d1){
            int dif1=c1-a1,dif2=d1-b1;
            int c=cmmdc(dif1,dif2);
            if(c==0){
                A[a1][b1]=1;
            }
            else{
                dif1/=c;
                dif2/=c;
                int lin=a1,col=b1;
                A[lin][col]=1;
                while(!(lin==c1 && col==d1)){
                    lin+=dif1;
                    col+=dif2;
                    A[lin][col]=1;
                }
            }
        }
        else{
            int dif1=c1-a1,dif2=b1-d1;
            int c=cmmdc(dif1,dif2);
            if(c==0){
                A[a1][b1]=1;
            }
            else{
                dif1/=c;
                dif2/=c;
                int lin=a1,col=b1;
                A[lin][col]=1;
                while(!(lin==c1 && col==d1)){
                    lin+=dif1;
                    col-=dif2;
                    A[lin][col]=1;
                }
            }
        }
    }
    ///calculam sume partiale
    for(int i=1;i<=L;i++){
        for(int j=1;j<=C;j++){
            B[i][j]=B[i-1][j]+B[i][j-1]-B[i-1][j-1]+A[i][j];
        }
    }
    ///trebuie determinata aria maxima patratica cu zerouri
    ///cautam binar latura maxima
    int lat1=1,lat2=min(L,C),lat=0,lmij;
    while(lat1<=lat2){
        lmij=(lat1+lat2)/2;
        int ok=0;
        for(int i=1;i+lmij-1<=L && !ok;i++){
```

```
            int i1=i+lmij-1;
            for(int j=1;j+lmij-1<=C && !ok;j++){
                int j1=j+lmij-1;
                int s=B[i1][j1]-B[i-1][j1]-B[i1][j-1]+B[i-1][j-1];
                if(s==0){
                    ok=1;
                }
            }
        }
        if(ok==0){
            lat2=lmij-1;
        }
        else{
            lat=lmij;
            lat1=lmij+1;
        }
    }
    if(lat==0){
        ///nu exista zone patratice neatinsa de pase
        fout<<0;
    }
    else{
        int nr=0;
        for(int i=1;i+lat-1<=L;i++){
            int i1=i+lat-1;
            for(int j=1;j+lat-1<=C;j++){
                int j1=j+lat-1;
                int s=B[i1][j1]-B[i-1][j1]-B[i1][j-1]+B[i-1][j-1];
                if(s==0){
                    nr++;
                    v[nr]={i,j,i1,j1};
                }
            }
        }
        fout<<nr<<"\n";
        fout<<(lat-1)*(lat-1)<<"\n";
        for(int i=1;i<=nr;i++){
            fout<<v[i].a1<<" "<<v[i].b1<<" "<<v[i].c1<<" "<<v[i].d1<<"\n";
        }
    }
    return 0;
}
```

Capitolul 11

Barajul 2

11.1 Problema Copaci

Propusă de: lector dr. Paul Diac, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza" Iași

Pentru a se relaxa, Dan se plimbă printr-o pădure virtuală formată din $N \times N$ copaci, așezați sub forma unei matrici pătratice cu N linii și N coloane. Pe fiecare copac este scrisă o cifră zecimală.

Plimbarea lui Dan începe de la un copac aflat într-o poziție de coordonate necunoscute și constă dintr-o succesiune de pași. La fiecare pas, Dan se va deplasa la unul dintre copacii vecini. Doi copaci sunt vecini dacă se află pe aceeași linie pe coloane consecutive, sau se află pe aceeași coloană, pe linii consecutive. Pe parcursul plimbării, Dan memorează într-un șir S cifrele scrise pe copacii aflați în pozițiile prin care a trecut. Fiind însă foarte obosit, este posibil ca la un moment dat să fi memorat greșit cifrele în șirul S.

Cerințe

Date fiind matricea care reprezintă pădurea virtuală și șirul S în care au fost memorate cifrele de pe copacii situați în pozițiile vizitate de Dan pe parcursul plimbării, să se determine numărul maxim de poziții vizitate pe parcursul plimbării, până la prima cifră memorată greșit în șirul S.

Date de intrare

Fișierul de intrare `copaci.in` conține pe prima linie numărul natural N. Pe următoarele N linii se află câte N cifre, reprezentând cifrele scrise pe copacii din pădurea virtuală, în ordinea liniilor, iar pentru copacii de pe aceeași linie, în ordinea coloanelor. Pe ultima linie se află cifrele memorate în șirul S. În fișierul de intrare nu sunt spații.

Date de ieșire

Fișierul de ieșire `copaci.out` va conține o singură linie pe care va fi scris un singur număr natural, reprezentând răspunsul la cerința dată.

Restricții

- $1 \leq N \leq 100$
- $1 \leq |S| \leq 10^5$

#	Puncte	Restricții
1	20	$N \leq 10$ și $\|S\| \leq 100$.
2	80	Fără restricții suplimentare.

Exemple

copaci.in	copaci.out	Explicații
3 769 617 502 97205562	5	Numărul maxim de poziții parcurse până la prima cifră memorată greșit în S este 5. O plimbare posibilă este: 769 617 502
3 125 452 803 32541	5	Șirul S este corect în întregime. O plimbare posibilă este: 125 452 803
5 36231 92928 08044 51868 43301 6281864292913	11	Numărul maxim de poziții parcurse până la prima cifră memorată greșit în S este 11. O plimbare posibilă este: 36231 92928 08044 51868 43301 Observați că pe parcursul plimbării Dan poate trece de mai multe ori prin aceeași poziție.
2 30 47 121	0	Numărul maxim de poziții parcurse până la prima cifră memorată greșit în S este 0, pentru că nu există niciun copac pe care să fie scrisă cifra 1.

11.2 Rezolvarea problemei Copaci

Soluție $O(N^2 \times |S|)$.

Pentru primul pas al plimbării lui Dan, putem construi o matrice de poziții accesibile $a[i][j]$ în care să reținem valoarea 1 pentru coordonate la care ar putea fi primul copac de la care pleacă Dan, și 0 în rest. Valorile 1 vor fi doar la coordonate la care copacii au cifra scrisă egală cu prima cifră $S[0]$. Pentru pasul al doilea, putem construi o nouă matrice similară, cu valori 1 la coordonatele la care Dan ar putea fi la al doilea pas. La aceste coordonate trebuie să fie copaci cu cifra scrisă următoare, adică $S[1]$, dar coordonatele să fie și vecine cu coordonate în care matricea $a[][]$ are valoarea 1. Astfel se completează plimbarea lui Dan cu al doilea pas în continuarea copacilor care ar fi putut fi primii în plimbare. Mai departe, pentru fiecare pas, putem construi astfel o matrice de poziții accesibile, dar fiecare pas i se bazează doar pe matricea de la pasul anterior $i-1$ și cifra $S[i]$. Astfel, putem reține doar ultimele două matrici. Pentru simplificarea implementării putem construi o matrice $a[0...1][1...N][1...N]$ unde prima coordonată reprezintă paritatea pasului curent iar următoarele două corespund liniilor și coloanelor pădurii. Paritatea se schimbă de la un pas la altul, astfel la pasul curent i construim accesibilitatea în matricea $a[i\%2][][]$ folosind matricea de la pasul anterior $a[!(i\%2)][][]$. Rezultatul va fi cea mai mare

valoare i pentru care avem măcar un element nenul în matricea $a[i\%2][][]$. Parcurgând matricea la fiecare pas și iterând fiecare din cei 4 vecini pentru fiecare element, obținem o complexitate de $O(N^2 \times |S|)$ și aproximativ 60-70 puncte.

Soluție $O(N^2 \times |S| \ / \ 64)$.

Putem compresa informația din matricea $a[][][]$ folosind numere întregi ai căror biți au valorile 0 și 1 cu aceeași semnificație ca în soluția de bază. Folosind tipul *long long*, putem grupa într-un singur element 64 de valori consecutive de pe o aceeași linie. Cifrele scrise pentru fiecare copac în parte sunt relevante și ele pentru fiecare coloană, dar le putem reprezenta tot prin mulțimi de biți, precalculate pentru fiecare cifră posibilă în parte. Pentru că avem doar zece cifre, memoria este suficientă. Vecinii de deasupra și de dedesubt se pot procesa printr-un *sau* pe mulțimi de biți: *or* adică operatorul |. Vecinii din stânga și dreapta strict din interiorul unei mulțimi de 64 biți se pot procesa folosind operatorii de shift-are pe biți $a[\ldots][\ldots][\ldots] << 1$ și $a[\ldots][\ldots][\ldots] >> 1$, în combinație cu *si* adică & pentru mulțimile de biți ale cifrei $S[i]$. Vecinii din capetele unei mulțimi de 64 biți trebuie tratați separat, este posibil ca un drum să se continue dintr-o coloană care este inclusă în gruparea din stânga sau dreapta grupării curente. Dar având doar două capete, sunt doar două operații în plus. Implementări care folosesc grupări de mai puțini biți sau *bitset* <> din STL ar putea obține punctaje parțiale sau maxime în funcție și de alte detalii de implementare.

11.3 Cod-sursă pentru problema Copaci

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define NMax 105
#define SMax 100005
#define K 64
#define lld unsigned long long
int N;
lld b[10][NMax][NMax/K+5]; // bitset of each digit
lld a[2][NMax][NMax/K+5]; // a[phase][line][bit set] = if accessible
char P[NMax][NMax], S[SMax];

int main() {
freopen("copaci.in", "r", stdin);
freopen("copaci.out", "w", stdout);
scanf("%d\n", &N);
for (int i = 0; i < N; i++)
        fgets(P[i], SMax, stdin);
fgets(S, SMax, stdin);
S[strlen(S)-1] = 'x';
int l = 0, ok = 0, y, l2;
for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
                a[0][i][j/K] |= ((lld)(S[0] == P[i][j])) << (j%K);
                b[P[i][j]-'0'][i][j/K] |= ((lld)1 << (j%K));
                ok |= (a[0][i][j/K] != 0);
        }
}

while (ok) {
        l++; l2 = l%2;
        ok = 0;
        for (int i = 0; i < N; i++) {
                for (int j = 0; j <= (N+1)/K+1; j++) { a[l2][i][j] = 0; }
    }
```

```
        for (int i = 0; i < N; i++) {
                for (int j = 0; j <= (N+1)/K; j++) {
                        a[l2][i][j]|=(b[S[l]-'0'][i][j] & ((a[!l2][i][j] << 1)|(a[!l2][i][j]>>1)));
                        if (i > 0)
                   { a[l2][i][j] |= (b[S[l]-'0'][i][j] & a[!l2][i-1][j]); }
               if (i < N)
                   { a[l2][i][j] |= (b[S[l]-'0'][i][j] & a[!l2][i+1][j]); }
               if (j > 0)
                   {a[l2][i][j]|=(((lld)(P[i][j*K]==S[l])) &&
                                 (a[!l2][i][j-1] & ((lld)1<<(K-1))));}
               if ((j+1)*K-1 < N)
                   {a[l2][i][j] |= ((lld)(((lld)(P[i][(j+1)*K-1] == S[l])) &&
                                   (a[!l2][i][j+1] & (lld)1))) << (K-1); }
                        ok |= (a[l2][i][j] != 0);
                        }
                }
        }
printf("%d\n", l);
fclose(stdout);
return 0;
}
```

11.4 Problema Pmo

Propusă de: prof. Ciprian Chește, Liceul Tehnologic „Grigore C. Moisil" Buzău
stud. Dumitru Ilie, Facultatea de Matematică-Informatică, Universitatea București

Fie N un număr natural.

Definim o *partiție multiplicativă ordonată* a numărului N ca fiind o scriere a lui N ca produs de unul sau mai mulți divizori diferiți de 1 ai lui N.

Exemple :

- Cele 4 partiții ale numărului $N = 8$ sunt: $8, 2 \cdot 4, 4 \cdot 2, 2 \cdot 2 \cdot 2$
- Cele 8 partiții ale numărului $N = 12$ sunt: $12, 2 \cdot 6, 6 \cdot 2, 3 \cdot 4, 4 \cdot 3, 2 \cdot 2 \cdot 3, 2 \cdot 3 \cdot 2, 3 \cdot 2 \cdot 2$

Cerințe

Să se scrie un program care citește T numere naturale X_i , $1 \leq i \leq T$, și determină numărul de partiții multiplicative ordonate ale fiecărui număr X_i , $1 \leq i \leq T$.

Date de intrare

Fișierul de intrare `pmo.in` conține:

- pe prima linie numărul natural T
- pe a doua linie T numere naturale X_i, $1 \leq i \leq T$, separate prin câte un spațiu

Date de ieșire

Fișierul de ieșire `pmo.out` va conține, pe linia i, numărul de partiții multiplicative ordonate ale numărului X_i , $1 \leq i \leq T$.

Restricții

- $1 \leq T \leq 30\,000$
- $2 \leq X_i \leq 10^9$ pentru $1 \leq i \leq T$

#	Puncte	Restricții
1	12	$T \leq 1000$ și toate numerele sunt puteri ale unui număr prim
2	20	$T \leq 100$ și $2 \leq X_i \leq 15\,000$ pentru $1 \leq i \leq T$
3	68	Fără restricții suplimentare

Exemple

pmo.in	pmo.out	Explicații
4 2 8 12 10	1 4 8 3	Numerele $2, 8, 12, 10$ au 1, 4, 8, respectiv 3 partiții multiplicative ordonate.
2 123456 987654	2496 75	Numerele $123456, 987654$ au 2496, respectiv 75 partiții multiplicative ordonate.

11.5 Rezolvarea problemei Pmo

Soluția 1 - stud. Dumitru Ilie

Vom prezenta soluția pentru un singur X_i. Vom începe prin a factoriza numărul. Pentru a face asta rapid ne vom folosi de ciurul lui Eratostene pentru a găsi numerele prime mai mici decât $\sqrt{VAL\_MAX} \approx 32000$. Avem aproximativ 3400 de numere prime ce ne interesează.

O observație importantă este faptul că 2 se comportă la fel ca 3, 5, 7, 11, și orice alt număr prim, având o singură partiție multiplicativ ordonată. Mai mult, $2 \cdot 3 = 6$ se comportă la fel ca $2 \cdot 5 = 10$, $2 \cdot 7 = 14$, $3 \cdot 5 = 15$, și orice alt produs de două numere prime, având 3 partiții multiplicativ ordonate. Acest fapt se poate extinde. În general pentru un număr $p_1^{e_1} \cdot p_2^{e_2} \cdot p_3^{e_3} \cdot \ldots \cdot p_k^{e_k}$ (unde p_i este număr prim) nu ne interesează decât exponenții numerelor prime, e_1, e_2, e_3, ..., e_k. Mai mult, orice rearanjare a acestor exponenți oferă același număr de partiții, deci le putem rearanja într-un mod convenabil, mai exact descrescător. De exemplu, numărul $2^1 \cdot 7^5 \cdot 11^2 = 4067294$ va fi redus la o listă de 3 numere, mai exact $[5, 2, 1]$.

Datorită observației precedente putem calcula o singură dată răspunsul pentru toate configurațiile de exponenți și apoi obține în timp foarte bun numărul de partiții folosindu-ne de procesarea anterioară. Trebuie sa avem grijă ca numărul de configurații să nu fie prea mare, încât să nu depășim limita de timp doar preprocesând sau limita de memorie din cauza numărului mare de date necesare. Într-adevăr numărul configurațiilor este scăzut, aproximativ 1300 (Determinarea numărului exact este un exercițiu interesant pe care vă sugerăm să îl încercați). Pentru a obține prima dată numărul de partiții al unei configurații putem folosi metoda backtracking pentru a itera prin toate configurațiile „incluse" în configurația curentă și să adăugăm numărul de partiții obținut prin eliminarea acelor elemente. De exemplu, configurația $[5, 2, 1]$ poate fi „ruptă" în:

- $[5, 2, 1] - [1, 0, 0] = [4, 2, 1]$;
- $[5, 2, 1] - [2, 0, 0] = [3, 2, 1]$;
- $[5, 2, 1] - [3, 0, 0] = [2, 2, 1]$;
- $[5, 2, 1] - [4, 0, 0] = [1, 2, 1] = [2, 1, 1]$ (prin convenție le ordonăm descrescător);
- $[5, 2, 1] - [5, 0, 0] = [2, 1, 0] = [2, 1]$ (zerourile nu ne interesează);
- $[5, 2, 1] - [0, 1, 0] = [5, 1, 1]$;
- $[5, 2, 1] - [1, 1, 0] = [4, 1, 1]$;
- $[5, 2, 1] - [2, 1, 0] = [3, 1, 1]$;
- $[5, 2, 1] - [3, 1, 0] = [2, 1, 1]$;
- $[5, 2, 1] - [4, 1, 0] = [1, 1, 1]$;
- $[5, 2, 1] - [5, 1, 0] = [1, 1]$;
- $[5, 2, 1] - [0, 2, 0] = [5, 1]$;
- $[5, 2, 1] - [1, 2, 0] = [4, 1]$;
- $[5, 2, 1] - [2, 2, 0] = [3, 1]$;
- $[5, 2, 1] - [3, 2, 0] = [2, 1]$;
- $[5, 2, 1] - [4, 2, 0] = [1, 1]$;
- $[5, 2, 1] - [5, 2, 0] = [1]$;
- $[5, 2, 1] - [0, 0, 1] = [5, 2]$;
- $[5, 2, 1] - [1, 0, 1] = [4, 2]$;
- $[5, 2, 1] - [2, 0, 1] = [3, 2]$;
- $[5, 2, 1] - [3, 0, 1] = [2, 2]$;
- $[5, 2, 1] - [4, 0, 1] = [2, 1]$;
- $[5, 2, 1] - [5, 0, 1] = [2]$;
- $[5, 2, 1] - [0, 1, 1] = [5, 1]$;
- $[5, 2, 1] - [1, 1, 1] = [4, 1]$;
- $[5, 2, 1] - [2, 1, 1] = [3, 1]$;
- $[5, 2, 1] - [3, 1, 1] = [2, 1]$;
- $[5, 2, 1] - [4, 1, 1] = [1, 1]$;

- $[5,2,1]-[5,1,1]=[1]$;
- $[5,2,1]-[0,2,1]=[5]$;
- $[5,2,1]-[1,2,1]=[4]$;
- $[5,2,1]-[2,2,1]=[3]$;
- $[5,2,1]-[3,2,1]=[2]$;
- $[5,2,1]-[4,2,1]=[1]$;
- $[5,2,1]-[5,2,1]=[]$;

Soluția 2 - prof. Ciprian Chește

Se știe că numărul de soluții ordonate ale ecuației $x_1 \cdot x_2 \cdot \cdots \cdot x_n = T$ este egal cu

$$\prod_{i=1}^{k} \binom{\alpha_i+n-1}{n-1} \textbf{ (1)}$$

unde $\alpha_i, 1 \leq i \leq k$ reprezintă exponenții descompunerii numărului T, în factori primi.

Un număr T poate fi descompus ca partiție multiplicativă ordonată în produs de $1, 2, 3, \ldots, \alpha_1+\alpha_2+\cdots+\alpha_k$ termeni. Așadar putem aplica relația (1) pentru toate valorile lui n cuprinse între 1 și $\alpha_1+\alpha_2+\cdots+\alpha_k$. Trebuie însă avut în vedere că în relația (1) sunt incluse și soluțiile care-l conțin pe 1 ca factor, ceea ce înseamnă că acestea trebuie scăzute din numărul total calculat. Soluțiile cel conțin pe 1 ca factor și au k termeni pot fi calculate combinatoric în funcție de soluțiile anterioare, adică cele cu $k-1$ termeni.

Exemplu: $T = 12 = 2^2 3^1$.

Sunt 8 partiții multiplicative ordonate ale lui 12:

$12, 2 \cdot 6, 6 \cdot 2, 3 \cdot 4, 4 \cdot 3, 2 \cdot 2 \cdot 3, 2 \cdot 3 \cdot 2, 3 \cdot 2 \cdot 2$.

- Dacă 12 ar fi scris ca produs de 1 termen atunci relația (1) furnizează $\binom{2+1-1}{1-1} \cdot \binom{1+1-1}{1-1} = \binom{2}{0} \cdot \binom{1}{0} = 1$ soluție.
- Dacă 12 ar fi scris ca produs de 2 termeni atunci relația (1) furnizează $\binom{2+2-1}{2-1} \cdot \binom{1+2-1}{2-1} = \binom{3}{1} \cdot \binom{2}{1} = 6$ soluții. Din acest număr trebuie scăzute $1 \cdot \binom{2}{1} = 2$ deoarece există o singură soluție și 1 poate fi așezat la această soluție în $\binom{2}{1}$ poziții. Deci pentru acest caz avem $6-2=4$ soluții.
- Dacă 12 ar fi scris ca produs de 3 termeni atunci relația (1) furnizează $\binom{2+3-1}{3-1} \cdot \binom{1+3-1}{3-1} = \binom{4}{2} \cdot \binom{3}{2} = 18$ soluții. Din acest număr trebuie scăzute $1 \cdot \binom{3}{2}$ deoarece există o singură soluție cu un singur factor și 2 de 1 pot fi așezați în poziții și să mai scădem $4 \cdot \binom{3}{1}$ deoarece există 4 soluții scrise ca produs de 2 factori și un 1 poate fi așezat în $\binom{3}{1}$ poziții. Deci pentru acest caz avem $18-3-12=3$ soluții.

În total avem $1+4+3=8$ soluții.

Soluția 3 - prof. Adrian Panaete

Spunem că două numere sunt echivalente dacă au același număr de divizori primi și multiseturile exponenților pentru cele două numere coincid.

Observație: Dacă două numere A și B sunt echivalente atunci acestea vor avea exact același număr de descompuneri distincte.

Demonstrație: Într-adevăr, dacă $A = a_1^{E_1} \cdot a_2^{E_2} \cdot \cdots \cdot a_m^{E_m}$ cu a_i numere prime distincte putem scrie $B = b_1^{E_1} \cdot b_2^{E_2} \cdot \cdots \cdot b_k^{E_m}$ unde b_i sunt eventual alte numere prime.

Considerând o descompunere oarecare $A = A_1 \cdot A_2 \cdot \cdots \cdot A_p$ și fiecare factor A_j se poate scrie că $A_j = a_1^{e_1} \cdot a_2^{e_2} \cdot \cdots \cdot a_m^{e_m}$. Corespunzător acelui factor putem construi pentru B factorul $B_j = b_1^{e_1} \cdot b_2^{e_2} \cdot \cdots \cdot b_m^{e_m}$ și astfel avem o descompunere $B = B_1 \cdot B_2 \cdot \cdots \cdot B_p$ care corespunde în mod unic descompunerii alese pentru A.

În particular pentru un număr oarecare A putem asocia cea mai mică valoare echivalentă cu A $CA = Canonic(A)$ astfel : fie $E_1, E_2, \ldots, E_m$ aleși în ordine descrescătoare $CA = p_1^{E_1} \cdot p_2^{E_2} \cdot \cdots \cdot p_m^{E_m}$ unde $p_1, p_2, p_3 \cdots = 2, 3, 5, \ldots =$ șirul numerelor prime.

Observație: $m \leq 9$ deci echivalentul minim va avea cel mult factorii primi 2, 3, 5, 7, 11, 13, 17, 19, 23 și mai mult se poate dovedi că exponenții corespunzători sunt respectiv cel mult 29, 11, 6, 3, 2, 2, 1, 1, 1.

În final se poate astfel dovedi că nu există decât un număr relativ mic de numere canonice (puțin peste 1000) și orice alt număr va fi echivalent cu unul dintre aceste numere.

Acest lucru ne conduce spre următorul algoritm. - se factorizează numărul pe care vrem să îl rezolvăm - se ordonează descrescător exponenții - se construiește numărul canonic minim echivalent cu numărul inițial - se rezolvă numărul canonic.

Cum rezolvăm valorile canonice? Formula de rezolvare este valabilă pentru orice număr (nu numai pentru valorile canonice).

Fie A un număr. Notăm $sol[A]$ = numărul de descompuneri pentru A. În primul rând, dacă A are un singur factor prim atunci avem caz particular și se poate observă că $sol[A]$ este o putere de 2. De asemenea, dacă A este număr liber de pătrate putem trata particular problema cu un backtracking (există de fapt doar 9 astfel de numere canonice). Ca o observăție - soluția pentru numerele libere de pătrate sunt numere cunoscute sub denumirea de numere Fubini(irelevant pentru problema noastră)

Fie o descompunere $A = A_1 \cdot A_2 \cdot \cdots \cdot A_m$. Pentru A_1 fixat se realizează toate descompunerile numărului $A_2 \cdot A_3 \cdot \cdots \cdot A_m = \frac{A}{A_1}$ în număr de $sol[\frac{A}{A_1}]$. Se observă că astfel $sol[A]$ se obține ca sumă din $sol[\frac{A}{A_1}]$ pentru orice A_1 divizor diferit de 1 al lui A în particular pentru că formula să funcționeze se va lua $sol[1] = 1$ deoarece pentru $A_1 = A$ descompunerea va conține doar un singur termen. Această idee ne va conduce la un algoritm în care pentru a avea soluția pentru un număr A avem nevoie de soluțiile pentru fiecare divizor al lui A.

Cum putem optimiza acest algoritm ? În primul rând când rezolvăm un număr A vom avea de rezolvat toți divizorii lui A ceea ce ne arată că de fapt ar trebui să recalculăm același lucru de mai multe ori. Pentru a rezolva această situație vom memoiza orice valoare pe care o rezolvăm la un moment dat astfel încât când ajungem în rezolvare la un număr canonic care a fost rezolvat deja să nu fim nevoiți să recalculăm. Memoizarea ne reduce mult din operațiile pe care le avem de efectuat, dar tot trebuie să parcurgem toți divizorii numărului de rezolvat pentru fiecare număr canonic care nu a fost încă rezolvat.

Putem totuși să reducem numărul de divizori pe care trebuie să îl accesăm dacă urmărim cu atenție cum se formează mulțimea divizorului unui număr. Astfel, dacă alegem un divizor foarte mare (obținut de exemplu prin împărțirea lui a cu un factor prim A_i) mulțimea divizorilor acestui număr se regăsește în mulțimea divizorilor lui A, astfel dacă în loc să adunăm la $sol[A]$ valoarea $sol[\frac{A}{A_i}]$ adunăm $2 \cdot sol[\frac{A}{A_i}]$ vom adună de fapt nu numai pentru divizorul $\frac{A}{A_i}$ ci pentru toți divizorii acestuia care la rândul lor sunt divizori pentru A, deci trebuie să apară în soluție.(Ideea este aproximativă... calculul trebuie făcut cu atenție). În final, ținând cont că același lucru de va întâmplă pentru toți factorii primi A_i, dacă adunăm $2 \cdot sol[\frac{A}{A_i}]$ avem $sol[A]$ la care se adaugă de mai multe ori valori $sol[D]$, unde D pentru divizorii $\frac{A}{A_i}$. Analizând cu atenție cum putem elimina ce avem în plus se observă că după ce adunăm termenii $2 \cdot sol[\frac{A}{A_i}]$ va trebui să scădem $2 \cdot sol[\frac{A}{A_i \cdot A_j}]$ unde A_i, A_j sunt doi factori primi ai lui A, apoi să adunăm $2 \cdot sol[\frac{A}{\text{produs de 3 factori primi distincți ai lui A}}]$, apoi să scădem $2 \cdot sol[\frac{A}{\text{produs de 3 factori primi distincți ai lui A}}]$ și așa mai departe.

Cu alte cuvinte, pentru a calcula $sol[A]$:

- se adună $2 \cdot sol[\frac{A}{2}]$, $2 \cdot sol[\frac{A}{3}] \ldots 2 \cdot sol[\frac{A}{23}]$ (numai pentru factorii primi ai lui A)

- se scade $2 \cdot sol[\frac{A}{6}]$, $2 \cdot sol[\frac{A}{15}] \ldots 2 \cdot sol[\frac{A}{19 \cdot 23}]$ ($sol[A/(p \cdot q)]$ dar numai pentru p și q factori primi pentru A)

Și așa mai departe.

Această soluție va trebui să acceseze pentru un număr A cu M factori primi distincți doar 2^M divizori (ceea ce înseamnă mult mai puțin decât numărul total de divizori ai lui A)

Am obținut astfel un algoritm similar ca idee cu principiul includerii și excluderii (PINEX).

În final mai putem aminti de o optimizare a factorizarii numerelor care poate fi aplicată nu doar la această soluție ci în general la orice factorizare a numerelor (inclusiv la soluțiile anterioare). Optimizarea nu trebuie aplicată pentru a obține punctajul maxim dar poate ameliora foarte mult timpul de execuție.

Factorizare rapidă pentru numere până la $1\,000\,000\,000$

Se aplică Ciurul lui Eratostene până la $R = \sqrt{1\,000\,000\,000} = 32622$.

La marcajul numerelor în ciur vom folosi marcajul: Un factori prim al lui i (în loc să marcăm cu 1).

Vom construi în paralel șirul numerelor prime până la R (de fapt vom vedea ulterior că nu ne sunt necesare decât cele până la 1000).

Odată făcute precalculările putem factoriza numerele folosind succesiv următoarele idei:

- **Cazul 1**: Factorizarea numerelor $X \leq R$
  Deoarece pentru fiecare valoare $X \leq R$ cunoaștem un factor prim $P[X]$ determinăm exponentul lui $P[X]$ și simultan simplificăm numărul prin $P[X]$.
  Repetăm cu valoarea simplificată a lui X până când X ajunge la valoarea 1. Numărul de pași în acest caz este egal cu suma exponenților factorilor primi ai lui X (supraestimat 29 + 11 + 6 + 3 + 2 + 2 + 1 + 1 + 1).
- **Cazul 2**: Factorizarea numerelor X > R
  Pasul 1: Folosind numerele prime ≤ 1000 în total 162 numere) se calculează factorii primi mai mici decât 1000 cu exponenții lor și evident se simplifică X prin acești factori primi.
  Pasul 2: Se aplică dacă numărul avea factori primi ≥ 1000 ceea ce e echivalent cu faptul că X nu a devenit 1 în pasul anterior. Evident că valoarea rămasă X va mai conține doar factori primi > 1000 deci numărul factorilor primi distincți sau nedistincți este cel mult 2 adică $X = p$ sau $X = p^2$ sau $X = p \cdot q$ cu p, q numere prime și $p, q > 1000$
  - subcazul 2.1: dacă $X < 1000000$ atunci X nu poate să aibă doi factori < 1000 deci mai avem un singur factor prim cu exponentul 1
  - subcazul 2.2: dacă $X \geq 100000$ și X este pătrat perfect -> $X = p^2$ deci mai avem un factor prim cu exponentul 2
  - subcazul 2.3: se aplică dacă numărul X nu a fost factorizat încă. Se observă că acum suntem strict în una din două situații - $X = p$ sau $X = p \cdot q$ cu p și q numere prime $p, q > 1000$ (în cazul $X = p$, $p > 1000000$). Pentru a distinge între cele două situații aplicăm testul de primalitate Rabin-Miller. Deoarece $X \leq 1000000000$ este suficient să aplicăm testul doar pentru bazele 2, 3, 5, 7 pentru că testul devine determinist pentru numere ≤ 1000000:
    * dacă testul validează că X este număr prim atunci mai avem un factor prim cu exponentul 1
    * dacă testul spune că X este compozit atunci mai avem încă doi factori primi cu exponentul 1.

În cazul problemei noastre nu avem nevoie efectiv de factorizare ci doar de exponenții fiecărui factor prim. Astfel în ultimul caz analizat nu avem efectiv nevoie de valorile p și q din scrierea $X = p \cdot q$ ci doar de informația că avem doi exponenți 1.

Dacă în alt context (la o altă problema) chiar am dori factorizarea completă, observăm că singura nedeterminare a algoritmului este necunoașterea celor doi factori p și q de la cazul menționat. În acest caz putem determina unul dintre factori aplicând numărului X algoritmul Pollard's RHO.

O ultimă optimizare se referă la divizorii numerelor canonice pe care le întâlnim pe parcursul rezolvării. Acești divizori conțin sigur factori primi ≤ 23 doar că nu știm dacă exponenții sunt ordonați descrescător.

Pentru a avea formă canonică la aceste numere este suficient să iterăm printre cele 9 valori prime de la 2 la 23 și să determinăm exponenții acestora. Prin reordonarea descrescătoare a acestor exponenți putem canoniza și acești divizori astfel că în memoizarea aplicată în mod efectiv acel număr apropiat de 1000 de valori de valori distincte ale soluției.

11.6 Cod-sursă pentru problema Pmo

```
// Ilie Dumitru
#include<cstdio>
#include<bitset>
#include<vector>
#include<algorithm>
#include<map>
const int VMAX = 1000000001, SQRT_VMAX = 32000;

struct fact
{
        int v[9], cnt;
        fact() : v(), cnt(0) {}
        friend bool operator<(const fact& a, const fact& b)
        {
                if(a.cnt != b.cnt)
                        return a.cnt < b.cnt;

                int i;

                for(i = 0;i < a.cnt;++i)
                        if(a.v[i] != b.v[i])
                                return a.v[i] < b.v[i];

                return 0;
        }
};

std::bitset<SQRT_VMAX> ciur;
std::vector<int> prime;
std::map<fact, long long> memo;

void precalc()
{
        int i, j;
        memo[fact()] = 1;
        prime.push_back(2);
        for(i = 3;i < SQRT_VMAX;i += 2)
                if(!ciur[i])
                {
                        prime.push_back(i);
                        for(j = i * i;j < SQRT_VMAX;j += i * 2)
                                ciur[j] = 1;
                }
}

fact factorizare(int x)
{
        int i;
        fact f;
        for(i = 0;i < (int)prime.size() && prime[i] * prime[i] <= x;++i)
        {
                if(x % prime[i] == 0)
                {
```

```
                        do
                        {
                                x /= prime[i];
                                ++f.v[f.cnt];
                        }while(x % prime[i] == 0);
                        ++f.cnt;
                }
        }

        if(x != 1)
                ++f.v[f.cnt++];
        return f;
}

long long count(fact f);

long long bkt(int k, fact& f, bool sub = 0)
{
        if(k == f.cnt)
        {
                if(sub)
                        return count(f);
                return 0;
        }
        int i;
        long long ans = 0;
        for(i = 0;i <= f.v[k];++i)
        {
                f.v[k] -= i;
                ans += bkt(k + 1, f, sub || i);
                f.v[k] += i;
        }
        return ans;
}

long long count(fact f)
{
    std::sort(f.v, f.v + f.cnt, std::greater<int>());
    while(f.cnt && f.v[f.cnt - 1] == 0)
                --f.cnt;
    std::map<fact, long long>::iterator it;

    if((it = memo.find(f)) != memo.end())
                return it->second;

        memo[f] = 0;
        it = memo.find(f);
        return it->second = bkt(0, f);
}

int main()
{
        FILE* f = fopen("pmo.in", "r"), *g = fopen("pmo.out", "w");
        int _, x;

        precalc();
        fscanf(f, "%d", &_);
        do
        {
                fscanf(f, "%d", &x);
                fprintf(g, "%lld\n", count(factorizare(x)));
```

```
        }while(--_);

        fclose(f);
        fclose(g);
        return 0;
}
```

```
// prof. Chesca Ciprian
#include <bits/stdc++.h>
#define fmax 200
#define nmax 33000

using namespace std;

long long comb[51][51];
int prim[nmax], prime[nmax];

int main()
{
    int N, w, i, j, s, T, z;
    long long cnt_sol, produs_cnk, total;
    long long total_sol[fmax], total_sol_no1[fmax];
    int e[fmax];
    int fact, k, x = 0;
    int nr_prime = 0;

    ifstream fin("pmo.in");
    ofstream fout("pmo.out");

    // precalculez combinarile
    comb[0][0] = 1;
    comb[1][0] = 1; comb[1][1] = 1;
    for(i = 2;i <= 50;i++)
    {
        comb[i][0] = 1;
        for(j = 1;j <= i; j++)
            comb[i][j] = comb[i-1][j] + comb[i-1][j-1];
    }

    // ciurul lui Eratostene (cca 276 - cca 195 î.Hr.)
    prim[1] = 1;
    for (i = 2; i <= nmax; i++)
        if (prim[i] == 0)
            for (j = i + i; j <= nmax; j += i)
                      prim[j] = 1;

    for (i = 2; i <= nmax; i++)
        if (prim[i] == 0) prime[++nr_prime] = i;

    fin >> T;
    for(z = 1; z <= T; z++)
    {
    // citesc un numar din fisierul de intrare
    fin >> N;

    k = 0;

    //descompunere in factori primi a numarului N
    w = N;
    // descompun t in factori primi
    i = 1;
```

```
    while (prime[i]*prime[i]<=w)
    {
        if (w%prime[i]==0)
            {
            fact=0;
            while (w%prime[i]==0) {w /= prime[i];fact++;}
            if (fact) e[++k]=fact;
            }
        i++;
    }

    if (w > 1) e[++k] = 1;

    // calculez functia Prime Omega (suma exponentilor)
    s = 0;
    for(i = 1;i <= k;i++)
        s += e[i];

    // calculez numarul total de solutii,
    // inclusiv cele ce-l contin pe 1, si le retin intr-un vector
    x = 0;
    for(i = 1; i <= s; i++)
    {
        produs_cnk = 1;
        cnt_sol = 0;
        for(j = 1; j <= k; j++)
            produs_cnk *= comb[e[j] + i - 1][i - 1];

        cnt_sol += produs_cnk;

        total_sol[++x] = cnt_sol;
    }
    // scad numarul solutiilor cel contin pe 1
    total_sol_no1[1] = 1;
    for(i = 2; i <= s;i++)
    {
        total = 0;
        for(j = 1; j <= i - 1; j++)
            total += (total_sol_no1[j] * comb[i][j]);
        total_sol_no1[i] = total_sol[i] - total;
    }
    // calculez numarul final de solutii
    cnt_sol = 0;
    for(i = 1; i <= s; i++)
        cnt_sol += total_sol_no1[i];
    fout << cnt_sol <<"\n";
    }
    return 0;
}
```

```
//prof. Adrian Panaete
#include <bits/stdc++.h>

using namespace std;
ifstream f("pmo.in");
ofstream g("pmo.out");
const int vMax=1000000000;
const int SQRT=sqrt(vMax);
int np;
int sgn[512],lp[512],P[SQRT],myP[SQRT];
int canonic(int);
```

```
int millerCanonic(int);
inline int canonic2(int);
int64_t sol(int);
map<int,int64_t> SOL;
void precalc();//,bkt(int,int);
void afiseaza(int);
bool isMillerPrime(int);
int main()
{
    precalc();
    int n;
    f>>n;
    for(;n;n--)
    {
        int x;
        f>>x;x=millerCanonic(x);
        g<<sol(x)<<'\n';
    }
    return 0;
}
void precalc()
{
    for(int i=2;i<SQRT;i++)
        if(myP[i]==0)
        {
            P[++np]=i;
            for(int j=i;j<=SQRT;j+=i)
                myP[j]=i;
        }

    int fub[10]={1, 1, 3, 13, 75, 541, 4683, 47293, 545835, 7087261};
    sgn[0]=-1;lp[0]=1;
    for(int i=1,lg=1,LG=2;i<=9;i++,lg*=2,LG*=2)
        for(int j=0,k=lg;j<lg;j++,k++)
        {
            lp[k]=lp[j]*P[i];
            sgn[k]=-sgn[j];
        }
    for(int i=1;i<512;i++)
    {
        SOL[lp[i]]=fub[__builtin_popcount(i)];
    }
    for(int i=1;i<=29;i++)
        SOL[1<<i]=1<<(i-1);
}
int64_t sol(int x)
{

    if(SOL.find(x)!=SOL.end())
        return SOL[x];

    int y=canonic2(x);

    if(SOL.find(y)!=SOL.end())
    {
        SOL[x]=SOL[y];
        return SOL[y];
    }
    int64_t solNow=0;
    for(int i=1;i<512&&y%lp[i]==0;i++)
        solNow+=sgn[i]*sol(y/lp[i]);
```

```
    solNow*=2LL;
    SOL[x]=solNow;
    SOL[y]=solNow;
    return solNow;
}
int canonic(int x)
{
    int expo[10],nd=0;
    for(int i=1;i<=np && P[i]*P[i]<=x;i++)
        if(x%P[i]==0)
        {
            nd++;expo[nd]=0;
            while(x%P[i]==0)
            {
                x/=P[i];
                expo[nd]++;
            }
            for(int j=nd;j>1&&expo[j]>expo[j-1];j--)
                swap(expo[j],expo[j-1]);

        }
    if(x>1)
    {
        expo[++nd]=1;
        x=1;
    }
    for(int i=1;i<=nd;i++)
        for(int j=expo[i];j;j--)
            x*=P[i];
    return x;
}
inline int canonic2(int x)
{
    int expo[10],nd=0;
    for(int i=1;i<=9;i++)
        if(x%P[i]==0)
        {
            nd++;
            expo[nd]=0;
            while(x%P[i]==0)
            {
                expo[nd]++;
                x/=P[i];
            }
            for(int j=nd;j>1&&expo[j]>expo[j-1];j--)
                swap(expo[j],expo[j-1]);
        }
    for(int i=1;i<=nd;i++)
        for(int j=expo[i];j;j--)
            x*=P[i];
    return x;
}
int millerCanonic(int x)
{
    /// cazul 1 : numere pana la SQRT
    int expo[10],nd=0,pr;
    if(x<SQRT)
    {
        while(x>1)
        {
            nd++;expo[nd]=0;pr=myP[x];
```

```
            while(x%pr==0)
            {
                expo[nd]++;
                x/=pr;
            }
            for(int j=nd;j>1&&expo[j]>expo[j-1];j--)
                swap(expo[j],expo[j-1]);

        }
    }
    /// cazul 2 : numere peste SQRT - factorizare pana la 1000
    for(int i=1;i<=168&&P[i]*P[i]<=x;i++)
        if(x%P[i]==0)
        {
            nd++;expo[nd]=0;pr=P[i];
            while(x%pr==0)
            {
                expo[nd]++;
                x/=pr;
            }
            for(int j=nd;j>1&&expo[j]>expo[j-1];j--)
                swap(expo[j],expo[j-1]);
        }
    /// cazul 3 : mai am inca factori primi?
    if(x>1)
    {
        /// atunci sigur sunt > 1000
        if(x<=1000000) /// aici nu pot sa am decat inca unul
            expo[++nd]=1;
        else /// ori mai am 1 mare ori doi mai mici
        {
            int r=sqrt(1.0*x+0.00000001);
            if(r*r==x)
            {
                expo[++nd]=2;
                for(int j=nd;j>1&&expo[j]>expo[j-1];j--)
                    swap(expo[j],expo[j-1]);
            }
            else
            {
                expo[++nd]=1;
                if(!isMillerPrime(x))
                    expo[++nd]=1;
            }
        }
    }
    x=1;
    for(int i=1;i<=nd;i++)
        for(int j=expo[i];j;j--)
            x*=P[i];
    return x;

}
bool isMillerPrime(int N)
{
    int D=N-1,S,B,X;
            /// se scrie N-1 = 2^K * D  unde D=impar
    while(D%2==0)
        D/=2;
    for(int i=1; i<=4; i++)/// B = 2,3,5,7,11,13,17
    {
```

```
        B=P[i];X=1;S=D;
        for(int E=D;E;E>>=1){if(E&1)X=1LL*X*B%N;B=1LL*B*B%N;}
        while(S!=N-1&&X!=1&&X!=N-1)
        {
            X=1LL*X*X%N;
            S*=2;
        }
        if(X!=N-1 && S%2==0)
            return false;
    }
    return true;
}
```

11.7 Problema Pokemoni

Propusă de: prof. Gheorghe-Eugen Nodea, Centrul Județean de Excelență Gorj

Zona de joacă a celor doi pokemoni, **Fire** și **Water**, poate fi reprezentată ca o hartă bidimensională pe care sunt marcate punctele de coordonate (x, y), unde $0 \leq x \leq N$ și $0 \leq y \leq M$. Dintre aceste $(N + 1) \times (M + 1)$ puncte marcate pe hartă, P puncte sunt *puncte speciale* unde pokemonii se pot ascunde (hidden-points).

Inițial pokemonul **Fire** se află în punctul de coordonate $(0, 0)$ și dorește să ajungă în punctul de coordonate (N, M). Acesta se poate deplasa la un moment dat din punctul de coordonate (x, y) în unul dintre punctele vecine $(x, y+1)$ sau $(x+1, y)$. Pokemonul **Water** se află inițial în punctul de coordonate (N, M) și dorește să ajungă în punctul de coordonate $(0, 0)$. Acesta se poate deplasa la un moment dat din punctul de coordonate (x, y) în unul dintre punctele vecine $(x, y - 1)$ sau $(x - 1, y)$.

Ca în orice joc, pokemonii au stabilit reguli, după cum urmează:

1. Pokemonul **Fire**, începe jocul și își alege primul traseul pe care îl parcurge.
2. Pokemonul **Water**, își alege traseul pe care îl parcurge, după ce **Fire** parcurge traseul său.
3. Fiecare pokemon parcurge un traseu care vizitează **un număr maxim de puncte speciale dintre cele disponibile**.
4. Dacă sunt mai multe trasee cu număr maxim de puncte speciale, va fi ales traseul **minim lexicografic**.
5. Un punct special odată vizitat își va pierde proprietatea de punct special.

Cerințe

Date fiind N, M și lista punctelor speciale, scrieți un program care să determine numărul de puncte speciale rămase pe hartă la finalul jocului.

Date de intrare

Fișierul de intrare `pokemoni.in` conține pe prima linie numerele naturale N, M și P cu semnificația din enunț. Pe următoarele P linii se află punctele speciale, câte un punct pe o linie; un punct special este specificat prin coordonatele sale x y. Numerele aflate pe aceeași linie a fișierului de intrare sunt separate prin câte un spațiu.

Date de ieșire

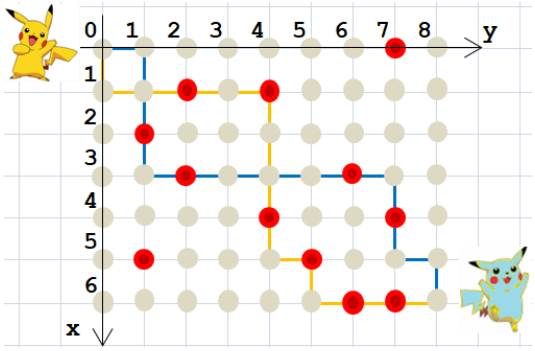
Fișierul de ieșire `pokemoni.out` va conține o singură linie pe care va fi scris un număr natural ce reprezintă numărul de puncte speciale rămase pe hartă la finalul jocului.

Restricții

- $2 < N, M \leq 10\,000$
- $1 < P \leq 100\,000$
- $0 \leq x \leq N$, $0 \leq y \leq M$
- Considerăm că un punct (x_1, y_1) este mai mic decât un punct (x_2, y_2) dacă $x_1 < x_2$ sau $x_1 = x_2$ și $y_1 < y_2$.
- Un traseu $(t_1, t_2, \ldots t_{lg})$ este mai mic din punct de vedere lexicografic decât un alt traseu $(s_1, s_2, \ldots, s_{lg})$ dacă există $i \geq 0$, astfel încât $t_j = s_j$ pentru orice $0 \leq j < i$ și $t_i < s_i$.

#	Puncte	Restricții
1	30	$2 < N, M \leq 100$, $1 < P \leq 1\,000$
2	20	$100 < N, M \leq 1\,500$, $1000 < P \leq 10\,000$
3	20	$N = 1000$, $M = 1000$, $10\,000 < P \leq 100\,000$
4	15	$1000 < N, M \leq 1\,500$, $P = 100\,000$
5	15	$N = 10\,000$, $M = 10\,000$, $P = 100\,000$

Exemple

pokemoni.in	pokemoni.out	Explicații
6 8 12 5 5 1 4 3 6 4 4 6 7 4 7 0 7 1 2 5 1 6 6 2 1 3 2	2	Cele două posibile trasee alese de pokemoni sunt marcate în imaginea de mai jos.  Numărul maxim de puncte speciale ce pot vizitate de pokemonul **Fire** este 6. Există mai multe trasee pentru a vizita cele 6 puncte speciale: 1. $(1,2)$–$(1,4)$–$(4,4)$–$(5,5)$–$(6,6)$ – $(6,7)$ 2. $(1,2)$–$(3,2)$–$(4,4)$–$(5,5)$–$(6,6)$ – $(6,7)$ Dintre aceste trasee el a ales primul traseu, acesta fiind minim lexicografic. Cele 6 puncte speciale vizitate de pokemonul **Fire** își pierd calitatea de punct special. Din punctele speciale rămase pokemonul **Water** poate vizita maximum 4. Rămân la final pe hartă 2 puncte speciale.

11.8 Rezolvarea problemei Pokemoni

Analizăm inițial traseul ales de pokemonul **Fire**, traseu care influențează traseul pokemonului **Water**.

Soluție 1. $O(P^2)$ - 50p

Reținem cele P coordonate ale punctelor speciale în vectorul Ps. Sortăm crescător după coordonata x, iar în caz de egalitate după coordonata y punctele speciale.

Problema se reduce la determinarea lungimii subșirului crescător maximal (**LIS**).

Pentru a determina lungimea maximă a unui subșir crescător al lui Ps, vom construi un șir suplimentar L, cu proprietatea că $L[i]$ este lungimea maximă a unui subșir care începe în $Ps[i]$. Atunci lungimea maximă LEN a unui subșir crescător va fi cel mai mare element din vectorul L.

Vom reține în vectorii $st[]$, respectiv $dr[]$ cea mai din stânga, respectiv dreapta poziție a fiecărei lungimii $L[i]$ găsită în șirul Ps, cu $1 \leq i \leq LEN$.

Determinarea traseului minim lexicografic ales de **Fire**, notat cu T, presupune parcurgem vectorul L, de la stânga la dreapta, plecând de la poziția $st[LEN]$ către poziția $dr[LEN]$.

Dacă $L[i] = LEN - 1$ iar $Ps[i].x \geq T[LEN].x$ și $Ps[i].y \geq T[LEN].y$ atunci adăugăm la traseu punctul $Ps[i]$, iar $LEN = LEN - 1$. Această parcurgere se efectuează în $O(N)$ amortizat.

La final, se elimină traseul pokemonului **Fire** din vectorul punctelor speciale Ps.

Se aplică același algoritm pentru traseul pokemonului **Water**.

Soluție 2. $O(N \times M)$ - 65-85p

Pentru a determina traseul parcurs de **Fire** putem reține într-o matrice de tip `bool` punctele speciale de pe hartă: $Sp[x][y] = 0$ sau 1, după cum punctul (x, y) este punct special sau nu.

$best[x][y]$ = numărul maxim de puncte ce se vizitează pe un traseu optim de la $(0, 0)$ la (x, y)

$best[0][0] = Sp[0][0]$

$best[x][0] = Sp[x][0] + best[x-1][0]$, pentru $x > 0$

$best[0][y] = Sp[0][y] + best[0][y-1]$, pentru $y > 0$

$best[x][y] = Sp[x][y] + max(best[x-1][y], best[x][y-1])$, pentru $x, y > 0$

Rezultatul va fi furnizat de $best[x][y]$. Din modul cum se construiește matricea $best$ se obține traseul minim lexicografic prin marcarea punctelor speciale vizitate.

Evident, se poate comprima matricea c astfel încât să se rețină liniile și coloanele care conțin puncte speciale. Această observație aduce puncte în plus.

Soluție 3. $O(P \times \log P)$ - 100p

Sortăm punctele ca în Soluția 1. Determinăm subșirul crescător maximal, folosindu-ne de căutare binară. Pentru aceasta, definim doi vectori auxiliari b și dp, unde:

- $b[i]$ - reprezintă un element al șirului de puncte în care se termină un subșir crescător maximal de lungime i
- $dp[i]$ - lungimea celui mai lung subșir crescător maximal care se termină cu elementul din șirul de puncte aflat la poziția i

Din moment ce șirul b va fi întotdeauna crescător, căutăm binar valoarea lui $dp[i]$, și vom actualiza valoarea lui $b[dp[i]]$ cu punctul curent. În funcția de comparare, va trebui ca ambele coordonate să fie mai mici decât cele ale punctului curent.

Pentru reconstituirea traseului, iterăm de la finalul vectorului dp. Dacă cumva valoarea la un anumit indice i este egală cu lungimea maximă a unui traseu obținut, vom lua numărul de pe poziția respectivă, scădem 1 din lungimea traseului, și continuă iterația.

După determinarea traseului lui Fire, se elimină punctele din șirul dat, și se repetă algoritmul pentru Water.

Însă, prin această metodă, obținem șirul maxim lexicografic, din cauza reconstituirii acestuia. Pentru a evita această problemă, vom sorta inițial șirul în ordine descrescătoare din punct de vedere lexicografic, și vom determina subșirul descrescător maximal.

Soluție 4. $O(P \times \log P)$ - 100p

Sortăm punctele în ordine lexicografică. Apoi, pentru fiecare punct cu coordonatele (x_i, y_i), vrem să determinăm care este cel mai bun punct (x_p, y_p) cu $x_p \leq x_i$ și $y_p \leq y_i$ cu lungimea drumului maximă. Diferit față de soluția precedentă, putem să căutăm valoarea maximă a unei lungimi de drum de pe intervalul $[1, y_i]$ întrucât procesăm punctele în ordine lexicografică (deci liniile nu pot fi restricții).

Ne dorim să construim un vector v unde $v[j]$ reprezintă lungimea maximă a unui drum care are ultimul punct situat pe coloana j.

Așadar, fiecare punct (x_i, y_i) va avea două tipuri de operații:

1. Query: care punct are drumul maxim cu $y_p \leq y_i$. (deci maximul dintre $v[1], v[2], \ldots, v[j]$
2. Update: toate punctele de pe intervalul $[y_p, m]$ se pot lipi de punctul curent, deci se va modifica $v[j]$. Cu alte cuvinte, maximul de pe intervalul $v[j], v[j+1], \ldots, v[m]$ va fi influențat de $v[j]$

Observăm ca această problemă se poate rezolva cu arbori de intervale, dar și cu arbori indexați binari (întrucât query-urile sunt strict pe intervale cu capătul stânga 1).

Pentru a genera drumul minim lexicografic, în arbore vom ține și indicele punctului cu lungimea maximă și comparăm pe caz de egalitate.

Acestă soluție funcționează numai pentru valori scăzute ale lui N.

11.9 Cod-sursă pentru problema Pokemoni

```
#include <bits/stdc++.h>
using namespace std;
ifstream f("pokemoni.in");
ofstream g("pokemoni.out");
const int NM = 100003;
struct puncts
{
    int x, y;
};
int cmp(puncts a, puncts b)
{
    return (a.x < b.x || a.x == b.x && a.y < b.y);
}
puncts P[NM], sir[NM];
int L[NM], pz[NM];
int N, M, n, m;

int LIS()
{   int i, j, m, k = 0, l, r, poz;
    for (i=1; i<=n; ++i)
    {
        if (P[i].y >= sir[k].y) sir[++k] = P[i], L[i] = k, pz[k] = i;
        else
        {
            l = 1; r = k; poz = k + 1;
            while (l <= r)
            {
                m = (l+r)/2;
                if (sir[m].y > P[i].y)
                    poz = m, r = m-1;
                else
                    l = m+1;
            }
```

```
            sir[poz] = P[i];
            pz[poz] = max(pz[poz], poz);
            L[i] = poz;
        }
    }
    return k;
}

int LIS1()
{   int i, j, pm, len, res;
    puncts q = {N+1, M+1};
    len = res = LIS();
    i = n;
    while (len)
    {
        pm = 0;
        for(j=i; j>=pz[len]; --j)
            if (L[j] == len)
                if (P[j].x <= q.x && P[j].y <= q.y)
                    if (pm == 0) pm = j;
                    else if (P[j].x < P[pm].x || P[j].x == P[pm].x && P[j].y < P[pm].y)
                    {
                        pm = j;
                    }

        i = pm-1; len--;
        q = P[pm];
        P[pm] = {0, 0};
    }
    return res;
}

int main()
{   int i, j, n1, n2;
    f >> N >> M >> n;
    m = n;
    for(i=1; i<=n; ++i)
        f >> P[i].x >> P[i].y;
    sort(P+1, P+n+1, cmp);
    n1 = LIS1();
    j = 0;
    for(i=1; i<=n; i++)
        if (P[i].x + P[i].y > 0)
            P[++j] = P[i];
    n = j;
    n2 = LIS();
    g << m - n1 - n2;
    return 0;
}
```

Partea a IV-a

Tabăra de pregătire a lotului național de informatică juniori

Orăștie, 24-29 mai 2024

Capitolul 12

Barajul 3

12.1 Problema Drumuri

Propusă de: prof. Mihai Bunget, Colegiul Național „Tudor Vladimirescu" Târgu Jiu

Fie N un număr natural și A un șir indexat de la 1, cu N^2 elemente, ce constituie o permutare a numerelor $1, 2, 3, \ldots, N^2$. Dintr-o poziție i, cu $1 \leq i \leq N^2$, ne putem deplasa în poziția:

- $i - 1$, dacă $i - 1$ nu se divide cu N
- $i + 1$, dacă i nu se divide cu N
- $i - N$, dacă $i - N \geq 1$
- $i + N$, dacă $i + N \leq N^2$

Un element A_i se numește *start* dacă toate elementele în care ne putem deplasa din poziția i au valoarea mai mare decât A_i. Un *drum* este o succesiune de elemente, $A_{i_1}, A_{i_2}, \ldots, A_{i_k}$, cu $k \geq 1$, astfel încât $A_{i_1} < A_{i_2} < \ldots < A_{i_k}$, cu proprietatea că elementul A_{i_1} este start și că ne putem deplasa din poziția i_1 în i_2, din i_2 în i_3,..., din i_{k-1} în i_k.

Cerințe

Să se afișeze un șir A astfel încât numărul drumurilor să fie cât mai mic (și să nu depășească 10^9), precum și numărul drumurilor din șirul afișat.

Date de intrare

Fișierul de intrare `drumuri.in` conține pe prima linie numărul natural N.

Date de ieșire

Fișierul de ieșire `drumuri.out` va conține pe prima linie N^2 numere naturale distincte cuprinse între 1 și N^2, separate prin câte un spațiu, reprezentând elementele șirului A. Pe a doua linie va fi scris numărul drumurilor din șirul A scris pe prima linie.

Restricții

- $1 \leq N \leq 1\,000$
- Numărul drumurilor din șirul A, notat $nrdrum$, trebuie să fie mai mic decât $1\,000\,000\,000$.
- Dacă numărul drumurilor este corect determinat pentru șirul A afișat, se acordă 40% din punctaj.

- În plus, se acordă $\left(\frac{nrmin}{nrdrum}\right)^4 \cdot 60\%$ din punctaj, unde $nrmin$ este numărul minim de drumuri care se poate obține pentru o permutare a șirului $1, 2, 3, \ldots, N^2$, dacă studiem toate permutările posibile.

#	Puncte	Restricții
1	45	$1 \leq N \leq 16$
2	36	$17 \leq N \leq 100$
3	19	Fără restricții suplimentare

Exemple

drumuri.in	drumuri.out	Explicații
2	1 2 3 4 5	Pentru primul exemplu drumurile sunt $(1), (1,2), (1,2,4), (1,3), (1,3,4)$, iar cum 5 este numărul minim de drumuri pentru toate permutările șirului $1, 2, 3, 4$, se vor acorda 100 de puncte.
2	1 3 4 2 6	Pentru al doilea exemplu drumurile sunt $(1), (1,3), (1,4), (2), (2,3), (2,4)$ și se vor acorda $40 + \left(\frac{5}{6}\right)^4 \cdot 60 = 68,93$ puncte.
3	7 6 9 1 3 5 4 8 2 15	Pentru al treilea exemplu drumurile sunt $(1), (1,4), (1,4,8), (1,3), (1,3,5), (1,3,5,9), (1,3,6), (1,3,6,7), (1,3,8), (1,7), (1,3,6,9), (2), (2,5), (2,5,9), (2,8)$ și se vor acorda $73,85$ puncte.

12.2 Rezolvarea problemei Drumuri

În primul rând vom studia cum se calculează numărul drumurilor pentru un șir A dat, ce reprezintă o permutare a numerelor $1, 2, 3, \ldots, N^2$.

Se reține mai întâi, pentru fiecare număr de la 1 la N^2, poziția din șirul A în care se află acesta. Se procesează apoi elementele șirului A în ordine crescătoare, calculând câte drumuri ajung în fiecare element. Elementul de valoare 1 este un punct de start, deci în el ajunge un singur drum. Pentru fiecare element de valoare de la 2 la N^2, numărul drumurilor care ajung în acest element este suma numărului de drumuri care ajung în elementele mai mici decât acesta (care au fost procesate deja) și din care ne putem deplasa în elementul curent. În final se calculează suma numărului de drumuri care ajung în fiecare element.

În continuare vom arăta că numărul drumurilor pentru orice permutare este cel puțin $2 \cdot N^2 - 2 \cdot N + 1$. Numim predecesor al unui element, un element din șirul A din care ne putem deplasa în elementul curent.

În șirul A există cel puțin un element de start, de exemplu cel cu valoarea 1. Împărțim șirul A în N secvențe a câte N elemente și observăm că în fiecare secvență $A_{N\cdot(i-1)+1}, A_{N\cdot i+2}, \ldots, A_{N\cdot i}$, unde $1 \leq i \leq N$, sunt posibile $N-1$ deplasări, câte o deplasare între oricare două elemente alăturate (de la elementul mai mic spre cel mai mare). Deci în fiecare element mai mare, dintre două elemente alăturate, va ajunge un drum venind dinspre elementul mai mic. Dacă elementul mai mic dintr-o pereche este element de start, atunci drumul pornește din acesta, altfel elementul mai mic are un predecesor (care evident e mai mic decât el) și continuând inductiv ajungem la un element care e start. Cum există N secvențe și în fiecare secvență vor fi cel puțin $N-1$ drumuri, obținem că vor exista cel puțin $N \cdot (N-1)$ drumuri care ajung în elemente ale șirului A, venind dinspre elemente alăturate, din aceeași secvență.

Considerând acum subșirurile de lungime N de forma $A_i, A_{N+i}, A_{2\cdot N+i}, \ldots, A_{(N-1)\cdot N+i}$, unde $1 \leq i \leq N$, observăm că în fiecare subșir se pot face $N-1$ deplasări între oricare două elemente alăturate, de la elementul mai mic spre cel mai mare. Printr-un raționament asemănător celui pentru secvențe se deduce că vor exista cel puțin $N \cdot (N-1)$ drumuri care ajung în elemente ale șirului, venind dinspre elemente alăturate, din același subșir.

În total vor fi cel puțin $N \cdot (N-1) + N \cdot (N-1) + 1 = 2 \cdot N^2 - 2 \cdot N + 1$ drumuri.

În final trebuie să dăm exemplu de permutare a numerelor $1, 2, 3, \ldots, N^2$ pentru care există exact $2 \cdot N^2 - 2 \cdot N + 1$ drumuri. Pentru a înțelege mai ușor acest exemplu, vom folosi o formă de reprezentare a șirului A mai sugestivă. Astfel, vom împărți șirul în secvențe de lungime N, și scriind aceste secvențe pe linii diferite vom obține o matrice pătratică de dimensiuni $N \times N$. Mai exact șirul A se transformă în matricea M astfel încât $M_{ij} = A_{N\cdot(i-1)+j}$, pentru orice $1 \leq i, j \leq N$. Deplasarea dintr-o poziție k în pozițiile $k-1, k+1, k-N, k+N$ în șirul A, unde $k = N \cdot (i-1) + j$ corespunde deplasării în matricea M din poziția (i, j) în pozițiile $(i, j-1), (i, j+1), (i-1, j), (i+1, j)$, respectiv.

Pentru N multiplu de 3 vom bloca cu $*$ anumite poziții din matricea M, apoi vom completa pozițiile rămase libere cu valori începând de la 1, consecutive, punând fiecare nouă valoare într-o poziție învecinată cu una deja completată (aceste valori le vom numi valori *mici*). Blocarea pozițiilor se poate face după următorul algoritm: se pune $*$ pe coloanele 1 și 3 începând cu linia 2, din doi în doi, apoi pe coloanele 4 și 6 începem de pe linia 1 din doi în doi ș.a.m.d. Dacă pe coloanele i și $i+2$ trebuie să punem $*$ pe ultima linie, atunci punem $*$ pe coloana $i+1$ și ultima linie. De exemplu, pentru $N = 6$, matricea se va completa astfel:

$$\begin{pmatrix} 1 & 2 & 3 & * & 24 & * \\ * & 4 & * & 23 & 22 & 25 \\ 6 & 5 & 7 & * & 21 & * \\ * & 8 & * & 19 & 18 & 20 \\ 10 & 9 & 12 & * & 17 & * \\ 11 & * & 13 & 14 & 15 & 16 \end{pmatrix}$$

Pozițiile marcate cu $*$ se vor completa aleatoriu cu numerele rămase (pe care le vom numi valori *mari*) și obținem:

$$\begin{pmatrix} 1 & 2 & 3 & 26 & 24 & 27 \\ 28 & 4 & 29 & 23 & 22 & 25 \\ 6 & 5 & 7 & 30 & 21 & 31 \\ 32 & 8 & 33 & 19 & 18 & 20 \\ 10 & 9 & 12 & 34 & 17 & 35 \\ 11 & 36 & 13 & 14 & 15 & 16 \end{pmatrix}$$

Pentru a reconstitui șirul A se parcurge matricea pe linii. Observăm că va exista un singur element de start, în fiecare element mic ajunge un singur drum, iar fiecare element mare fiind învecinat doar cu elemente mici, în el va ajunge un singur drum de la fiecare element mic învecinat. Astfel fiecare deplasare orizontală sau verticală posibilă adaugă un singur drum elementului în care se ajunge, obținând exact numărul minim de drumuri calculat anterior. Pentru N de forma $3 \cdot k + 2$ se completează în mod similar matricea M de dimensiune $3 \cdot k + 3$ apoi se șterg prima linie și prima coloană, iar pentru N de forma $3 \cdot k + 1$ se completează matricea de dimensiune $3 \cdot k + 3$ și se șterg primele două linii, prima și ultima coloană.

Pentru a obține punctaje parțiale, pentru $N \leq 16$ se poate afișa orice permutare, numărul drumurilor fiind sigur mai mic decât 10^9, însă acest număr trebuie determinat corect. Pentru $N > 16$ se poate alege de exemplu o permutare A pentru care matricea corespunzătoare M să aibă aspectul unei table de șah, punând numerele mici pe pătrățelele negre iar cele mari pe pătrățelele albe, obținând astfel un punctaj de aproximativ 64p. De asemenea, numărul drumurilor se poate determina și recursiv.

12.3 Cod-sursă pentru problema Drumuri

```
#include <fstream>

using namespace std;

ifstream f("drumuri.in");
ofstream g("drumuri.out");

int a[1005][1005], d[1005][1005];
pair<int,int> poz[1000003];
int i, j, n, nrdrum, val, m;

bool start(int l, int c)
{
 bool rez = true;
 if (l > 1) if (a[l-1][c] < a[l][c]) rez = false;
 if (l < n) if (a[l+1][c] < a[l][c]) rez = false;
 if (c > 1) if (a[l][c-1] < a[l][c]) rez = false;
 if (c < n) if (a[l][c+1] < a[l][c]) rez = false;
 return rez;
}

void solve(int h)
{int l, c;
 l = poz[h].first;
 c = poz[h].second;
 if (start(l,c)) d[l][c] = 1;
     else
     {
      if (l > 1) if (a[l-1][c] < a[l][c]) d[l][c] += d[l-1][c];
      if (l < n) if (a[l+1][c] < a[l][c]) d[l][c] += d[l+1][c];
      if (c > 1) if (a[l][c-1] < a[l][c]) d[l][c] += d[l][c-1];
      if (c < n) if (a[l][c+1] < a[l][c]) d[l][c] += d[l][c+1];
    }
}

void add(int l, int c)
{
 a[l][c] = val;
 poz[val] = {l,c};
 val++;
 if ((l > 1) and (a[l-1][c]==0)) add(l-1,c);
 if ((l < n) and (a[l+1][c]==0)) add(l+1,c);
 if ((c > 1) and (a[l][c-1]==0)) add(l,c-1);
 if ((c < n) and (a[l][c+1]==0)) add(l,c+1);
}

void mat(int n)
{
  int i, j;
  for (j = 1; j <= n; j+=3)
      if (j%2==1)
         {
          for (i = 2; i < n; i+=2)
              {
                a[i][j] = -1;
                a[i][j+2] = -1;
              }
          if (n%2==0) a[n][j+1] = -1;
         }
```

```
        else
        {
         for (i = 1; i < n; i+=2)
             {
              a[i][j] = -1;
              a[i][j+2] = -1;
             }
         if (n%2==1) a[n][j+1] = -1;
        }
}

int main()
{
  f >> n;
  if (n%3==0) mat(n);
     else
     {
      mat(n+1);
      for (i = 2; i <= n+1; i++)
          for (j = 2; j <= n+1; j++)
              a[i-1][j-1] = a[i][j];
     }
  val = 1;
  add(1,1);
  val = n*n;
  for (i = 1; i <= n; i++)
       for (j = 1; j <= n; j++)
            if ((a[i][j]==-1)or(a[i][j]==0))
                {a[i][j] = val; poz[val] = {i,j}; val--;}
   m = n*n;
   for (i = 1; i <= m; i++)
        solve(i);
   nrdrum = 0;
   for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            {
             g << a[i][j] << " ";
             nrdrum += d[i][j];
            }
   g << "\n" << nrdrum;
   return 0;
}
```

12.4 Problema Gimigpt

Propusă de: stud. Ioan-Cristian Pop, Universitatea Politehnica București

În urma rezultatelor bune obținute, Gimi a devenit foarte popular printre colegii lui. Însă, nu în modul pe care și-l dorea. Acum, majoritatea colegilor au început să-i pună diverse întrebări teoretice.

Gimi a primit o listă de N întrebări de la colegii săi, și își propune să răspundă în ordine la ele. A i-a întrebare q_i constă dintr-un șir de caractere alcătuit numai din litere mici ale alfabetului englez. Ce a observat însa este că el poate primi aceeași întrebare, formulată diferit – așa că, dacă întrebarea q_i *se aseamănă* cu cel puțin o întrebare q_j (cu $1 \leq j < i$) la care deja a răspuns, atunci el nu va mai răspunde la întrebarea q_i.

Gimi și-a definit următoarele criterii de *asemănare* între întrebarea q_j și întrebarea q_i:

- Egalitate (E): Dacă întrebarea q_i este egală cu întrebarea q_j, atunci q_i și q_j se aseamănă prin criteriul E.
- Inserare (I); Dacă prin inserarea unui singur caracter în q_i, aceasta devine egală cu q_j, atunci q_i și q_j se aseamănă prin criteriul I.
- Ștergere (S): Dacă prin ștergerea unui singur caracter din q_i, aceasta devine egală cu q_j, atunci q_i și q_j se aseamănă prin criteriul S.
- Modificare (M): Dacă prin modificarea unui singur caracter din q_i, aceasta devine egală cu q_j, atunci q_i și q_j se aseamănă prin criteriul M.

Notăm cu T mulțimea de criterii de asemănare pe care el le utilizează. De exemplu, dacă T = EI, el va folosi numai criteriile E și I pentru a determina dacă întrebările seamănă sau nu. Dacă T = EISM, atunci el va folosi toate cele 4 criterii.

Gimi pornește cu prima întrebare din listă și va răspunde la ea. După aceea, începând de la a doua întrebare până la ultima, procedează astfel:

- Dacă întrebarea q_i seamănă cu cel puțin o întrebare q_j pe baza a cel puțin unuia dintre criteriile din T, atunci **NU** va răspunde la întrebarea q_i.
- Altfel, va răspunde la întrebarea q_i.

Fiind un set foarte mare de întrebări, Gimi și-ar dori să determine: pentru o mulțime de criterii T, la care întrebări va răspunde (și la care nu).

Cerințe

Scrieți un program care, cunoscând N, numărul de întrebări, T, criteriile de asemănare folosite, respectiv cele N întrebări q_1, q_2, ..., q_N, afișează pentru fiecare dintre aceste întrebări dacă se va răspunde la ea sau nu.

Date de intrare

Fișierul de intrare `gimigpt.in` conține pe prima linie un număr natural N, reprezentând numărul de întrebări, și un șir de caractere T, care conține criteriile de asemănare, separate printr-un spațiu.
Următoarele N linii conțin, în ordine, cele N întrebări, câte o întrebare pe o linie.

Date de ieșire

Fișierul de ieșire `gimigpt.out` va conține N linii. Linia i va conține valoarea 1 dacă se va răspunde la cea de a i-a întrebare din fișierul de intrare, respectiv valoarea 0 în caz contrar.

Restricții

- $T \in$ {E, EI, ES, EM, EIS, EIM, ESM, EISM}
- $1 \leq N \leq 80\,000$
- $2 \leq \text{LEN}(q_i) \leq 26$, pentru oricare $1 \leq i \leq N$, unde LEN reprezintă lungimea șirului.
- Testele sunt grupate. În cadrul fiecărui subtask, sunt precizate și grupele de teste asociate

#	Puncte	Restricții
1	5+6+6+6 (gr. 1, 2, 3, 4)	$N = 800$, $T \in$ {E, EI, ES, EM}
2	3+3+3+3 (gr. 5, 6, 7, 8)	$N = 10\,000$, $T \in$ {E, EI, ES, EM}
3	5 (gr. 9)	$N = 800$, $T =$ EISM
4	7 (gr. 10)	$N = 4\,000$, $T =$ EISM
5	12 (gr. 11)	$N = 10\,000$, $T =$ EISM
6	12 (gr. 12)	$N = 30\,000$, $T =$ EISM
7	2+2+4 (gr. 13, 14, 15)	$N \in \{800, 10\,000, 80\,000\}$, $T =$ EISM, $2 \leq len(q_i) \leq 10$
8	2+2+4 (gr. 16, 17, 18)	$N \in \{800, 10\,000, 80\,000\}$, $T =$ EISM, q_i conține numai a și b
9	13 (gr. 19, 20)	$N = 80\,000$, $T =$ EISM

Exemple

gimigpt.in	gimigpt.out	Explicații
5 EISM abc abb abbx abc bbb	 1 0 1 0 1	Se va răspunde la prima întrebare *abc*. A doua întrebare *abb* se aseamănă cu întrebarea *abc* pe baza criteriului M (se modifica ultima literă din *b* în *c*), deci nu se va răspunde. Se va răspunde la a treia întrebare *abbx*. Se observă că, dacă s-ar fi răspuns la întrebarea *abb*, atunci nu s-ar mai fi răspuns la întrebarea curentă. A patra întrebare *abc* se aseamănă cu întrebarea *abc* pe baza criteriului E (șirurile sunt egale), deci nu se va răspunde la ea. Se va răspunde la întrebarea *bbb*.
4 EI abc abb ac abbx	 1 1 0 1	Se va răspunde la prima întrebare *abc*. Se va răspunde la a doua întrebare *abb*. Ea nu se aseamănă cu întrebarea *abc* pe baza criteriilor E și I. Nu se va răspunde la a treia întrebare *ac*. Ea se aseamănă cu întrebarea *abc* pe baza criteriului I (prin inserarea literei *b*). Se va răspunde la întrebarea *abbx*. Ea se aseamănă cu întrebarea *abx* pe baza criteriului S (prin ștergerea literei *x*), care nu face parte din T. Criteriul I presupune strict inserarea unei litere în șirul curent (*abbx*) și nu inserarea unei litere într-o întrebare la care s-a răspuns deja.

12.5 Rezolvarea problemei Gimigpt

Plecăm mai întâi de la soluția care obține în jur de 67 de puncte. Ideea acestei soluții este să verificăm, pentru fiecare întrebare, dacă se aseamănă cu oricare dintre întrebările precedente. Dar, pentru a verifica dacă două șiruri q_i și q_j se aseamănă, facem următorii pași:

1. Dacă diferența de lungime dintre șiruri este cel puțin 2, atunci nu se aseamănă.
2. Determinăm cel mai lung prefix comun și cel mai lung sufix comun (fie x capătul dreapta al prefixului, și y capătul stânga al sufixului, raportate la întrebarea q_i)
3. În funcție de valorile lui x și y, avem următoarele cazuri:
   - Dacă $y = 0$, atunci cele două șiruri sunt identice (deci se aseamănă).
   - Dacă $x + 1 < y$, atunci cele două șiruri diferă prin cel puțin două transformări, deci nu se aseamănă.
   - Dacă $|y - x| <= 1$, atunci înseamnă că unul dintre caractere trebuie șters, inserat sau modificat. În funcție de criteriile de asemănare selectate, șirurile se aseamănă sau nu.

Această soluție are o complexitate mare, deoarece verifică șir cu șir. Observăm că prin operațiile de modificare noi putem genera aproximativ 26^2 șiruri care se aseamănă, dar tot vom trece prin lista de N cuvinte.

Optimizarea acestei soluții constă în a genera o parte dintre șirurile care se aseamănă cu q_j odată ce răspundem la întrebare, în mulțimi separate în funcție de tipul de transformare.

Acestea sunt de forma:

- Set original (SO): adăugăm întrebarea q_j.
- Set cu operații de ștergere (SS): adăugăm fiecare variație posibilă (exemplu: din *abbx* se vor adăuga *bbx*, *abx* și *abbx*)
- Set cu operații de modificare (SM): adăugăm fiecare variație posibilă, însă pentru fiecare poziție, plasăm un „Joker" # în loc să punem toate cele 26 de litere de la a la z (exemplu: din *abbx* se vor adăuga *#bbx*, *a#bx*, *ab#x*, *abb#*).

Apoi, pentru fiecare întrebare q_i, se vor face următoarele verificări, în funcție de criteriile de asemănare:

- E: Verificăm dacă există șirul curent (căutăm în setul SO)
- I: Prin inserarea unei litere – verificăm dacă apare șirul curent în setul SS (decât să inserăm în șirul curent, e mai simplu să verificăm prin ștergerea unei litere dacă se poate ajunge în șirul curent).
- S: Prin ștergerea unui singur caracter – pentru fiecare poziție, „tăiem" litera și verificăm dacă apare șirul transformat prin căutarea în setul SO. Pentru $q_i = abyb$, s-ar putea obține întrebările *byb*, *abb* și *aby*.
- Prin modificarea unei litere – pentru fiecare poziție, modificăm litera în „Joker" # și verificăm dacă apare șirul transformat prin căutarea în setul SM. Pentru $q_i = abyb$, s-ar putea obține șirurile *#byb*, *a#yb*, *ab#b*, *aby#*;

Observăm că dacă $q_j = abbx$ iar $q_i = abyb$, cele două nu se aseamănă, pe baza niciunui criteriu de asemănare.

Pentru a verifica dacă există șirul într-un set, pentru 100 de puncte, se pot folosi hash-uri, de exemplu, algoritmul lui Rabin-Karp.

12.6 Cod-sursă pentru problema Gimigpt

```
/*
    100p - Rabin Karp algorithm with a single modulo
```

```
    MOD - used for position in hash table
    Hash value is calculated modulo (2^63 - 1) using unsigned long long (uint_64t)
    Assumption: 2 strings match if hashes match
*/

#include <fstream>
#include <string>
#include <vector>

using namespace std;

const unsigned int MOD = 10007;
const unsigned int SIGMA = 27;
const int MAXLEN = 26;

ifstream fin("gimigpt.in");
ofstream fout("gimigpt.out");

// Hash for original words (each new word will insert exactly 1 word)
vector <uint64_t> h_orig[MAXLEN + 1][MOD];
// Hash for words with modified letters (each new word may insert MAXLEN words, 1 per K)
vector <uint64_t> h_modif[MAXLEN + 1][MAXLEN + 1][MOD];
// Hash for words with deleted letters (each new word may insert at most MAXLEN words)
vector <uint64_t> h_remv[MAXLEN + 1][MOD];

// Powers of sigma
uint64_t p_sigma[MAXLEN + 1];
// Prefixes and suffixes of strings
uint64_t prefix[MAXLEN + 1];
uint64_t suffix[MAXLEN + 1];
// Hashes of word, without a specific letter
uint64_t word_hash[MAXLEN + 1];

// Type of inputs
int type;

// Add words to original hash -- collisions should *not* occur
void addOriginalWord(const int &n, const int &h1, const uint64_t &h2) {
    h_orig[n][h1].push_back(h2);
}

bool checkOriginalWord(const int &n, const int &h1, const uint64_t &h2) {
    for (const auto &x : h_orig[n][h1])
        if (x == h2)
            return true;
    return false;
}

// Add words to moified hash -- collisions should *not* occur
void addModifiedWord(const int &n, const int &k, const int &h1, const uint64_t &h2) {
    h_modif[n][k][h1].push_back(h2);
}

bool checkModifiedWord(const int &n, const int &k, const int &h1, const uint64_t &h2) {
    for (const auto &x: h_modif[n][k][h1])
        if (x == h2)
            return true;
    return false;
}

// Add words to removed hash -- collisions *may* occur
```

```
void addShorterWord(const int &n, const int &h1, const uint64_t &h2) {
    for (const auto &x : h_remv[n][h1])
        if (x == h2)
            return;
    h_remv[n][h1].push_back(h2);
}

bool checkShorterWord(const int &n, const int &h1, const uint64_t &h2) {
    for (const auto &x : h_remv[n][h1])
        if (x == h2)
            return true;
    return false;
}

// Build base SIGMA powers -- 1 for each modulo
void buildPowersOfSigma() {
    p_sigma[0] = p_sigma[0] = 1;
    for (int i = 1; i <= MAXLEN; ++i) {
        p_sigma[i] = 1LL * p_sigma[i - 1] * SIGMA;
    }
}

// Gets id of letter
unsigned int getLetterID(char ch) {
    return ch - 'a' + 1;
}

// Builds hashes for prefixes and suffixes
void buildPrefixesAndSuffixes(const string &s, const int &n) {
    prefix[0] = getLetterID(s[0]);
    for (int i = 1; i < n; ++i) {
        prefix[i] = prefix[i - 1] * SIGMA + getLetterID(s[i]);
    }

    suffix[n - 1] = getLetterID(s[n - 1]);
    for (int i = n - 2, k = 1; i >= 0; --i, ++k) {
        suffix[i] = p_sigma[k] * getLetterID(s[i]) + suffix[i + 1];
    }
}

// word_hash[k] = hash of word without the letter k
void buildHashes(const string &s, const int &n) {
    word_hash[n - 1] = prefix[n - 2];

    for (int i = n - 2, k = 1; i > 0; --i, k++) {
        word_hash[i] = prefix[i - 1] * p_sigma[k] + suffix[i + 1];
    }

    word_hash[0] = suffix[1];
}

bool addWord(const string &s, const int &n) {
    int mod;
    // Add original word.
    mod = suffix[0] % MOD;
    addOriginalWord(n, mod, suffix[0]);

    for (int i = 0; i < n; ++i) {
        int mod = word_hash[i] % MOD;
        // Add the word by removing the i-th letter
        addShorterWord(n - 1, mod, word_hash[i]);
```

```
        // Add the word by modifying the i-th letter (leaving it as a joker)
        addModifiedWord(n, i, mod, word_hash[i]);
    }

    // No need to add the words by inserting a letter (a + 1 = b <=> b - 1 = a)

    // Finish!
    return true;
}

// Checks if word exists, and if not, inserts it
bool solve(const string &s, const int &n) {
    buildPrefixesAndSuffixes(s, n);
    buildHashes(s, n);

    // Check if original word matches perfectly.
    int mod = suffix[0] % MOD;
    if (checkOriginalWord(n, mod, suffix[0]))
        return false;
    // Check if, by adding a letter, the word exists.
    // In other words, check if an existing word with a removed letter matches this one
    if (type & 2)
        if (checkShorterWord(n, mod, suffix[0]))
            return false;

    for (int i = 0; i < n; ++i) {
        mod = word_hash[i] % MOD;
        // Check if, by removing i-th letter, the word exists.
        if (type & 4)
            if (checkOriginalWord(n - 1, mod, word_hash[i]))
                return false;
        // Check if, by modifying i-th letter, the word exists. We are
        // looking for a match with another word of same length where the i-th
        // letter is removed
        if (type & 8)
            if (checkModifiedWord(n, i, mod, word_hash[i]))
                return false;
    }
    return addWord(s, n);
}

int main() {
    int q, n, ans = 0;
    string s, type_str;

    buildPowersOfSigma();
    fin >> q >> type_str;
    for (const char &ch : type_str) {
        if (ch == 'E')
            type = (type | 1);
        else if (ch == 'I')
            type = (type | 2);
        else if (ch == 'S')
            type = (type | 4);
        else
            type = (type | 8);
    }

    for ( ; q > 0; --q) {
        fin >> s; n = s.size();
```

```
        fout << solve(s, n) << "\n";
    }
    return 0;
}
```

12.7 Problema P2

Propusă de: prof. Ionel-Vasile Piț-Rada, Colegiul Național Traian Drobeta Turnu Severin

Considerând două numere naturale X și Y, spunem că X este prefix pentru Y, dacă X se poate obține din Y prin ștergerea a 0, 1 sau mai multor cifre de la sfârșitul lui Y.

Exemple:

- 1024 este prefix pentru 1024789056
- 526 este prefix pentru 526

Se dau numerele naturale Q, L și apoi o succesiune de Q interogări de următoarele 3 tipuri:

Tip interogare	Semnificație
1 M	Să se determine N minim, $N \leq L$, astfel încât 2^N să aibă prefix pe M (se garantează că pentru interogările date există un astfel de număr).
2 M N	Să se verifice dacă M este prefix pentru 2^N (răspunsul este 1 în caz afirmativ, respectiv 0 în caz contrar).
3 M A B	Să se determine numărul de valori N ($A \leq N \leq B$), astfel încât M este prefix pentru 2^N.

unde M, N, A și B sunt numere naturale.

Cerințe

Date fiind Q, L și o succesiune de Q interogări, să determine răspunsurile pentru interogările date.

Date de intrare

Fișierul de intrare `p2.in` conține pe prima linie numerele naturale Q și L, separate prin spațiu.
Pe următoarele Q linii sunt scrise cele Q interogări, câte o interogare pe o linie, în forma descrisă în tabel.

Date de ieșire

Fișierul de ieșire `p2.out` va conține Q linii.
Pe linia i va fi scris răspunsul pentru cea de a i-a interogare din fișierul de intrare.

Restricții

- $1 \leq Q \leq 100\,000$
- $1 \leq L \leq 1\,000\,000$
- $0 \leq M \leq 999\,999$
- $0 \leq N \leq L$
- $0 \leq A \leq B \leq L$

#	Puncte	Restricții
1	24	Fișierul de intrare conține numai interogări de tipul 1 și $1 \leq L, Q \leq 100\,000$.
2	24	Fișierul de intrare conține numai interogări de tipul 2 și $1 \leq L, Q \leq 100\,000$.
3	24	Fișierul de intrare conține numai interogări de tipul 3 și $1 \leq L, Q \leq 100\,000$.
4	16	$1 \leq L, Q \leq 100\,000$.
5	12	Fără restricții suplimentare

Exemple

p2.in	p2.out	Explicații
4 13 1 8 2 81 13 2 64 7 3 1 0 13	3 1 0 4	Valorile pentru $2^0, 2^1, \ldots, 2^{13}$ sunt 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192. Pentru prima interogare avem $M = 8$, iar cel mai mic $N \leq 13$ cu prefix egal cu 8 pentru 2^N este 3, $2^3 = 8$. Pentru a doua interogare avem $M = 81$ și $N = 13$ și deoarece $2^{13} = 8192$ avem rezultatul 1. Pentru a treia interogare avem $M = 64$ și $N = 7$ și deoarece $2^7 = 128$, avem că rezultatul este 0. Pentru a patra interogare avem $M = 1$, $A = 0$ și $B = 13$, iar valorile lui N pentru care 1 apare ca prefix în 2^N sunt 0, 4, 7, 10 deci răspunsul este 4.

12.8 Rezolvarea problemei P2

Pentru fiecare valoare $0 \leq N \leq L$ vom calcula prefixele de cel mult 6 cifre ale lui 2^N. Pentru fiecare prefix obținut vom reține lista valorilor N pentru care se obține prefixul respectiv, în ordine crescătoare.

Prefixele se vor calcula folosind operații cu numere mari. Se poate lucra cu baze de reprezentare egale cu diverse puteri ale lui 10.

Pentru interogările de tipul 1 M este suficient să afișăm prima valoare din lista corespunzătoare prefixului M.

Pentru interogările de tipul 2 M N verificăm dacă N apare în lista de valori memorate pentru prefixul M (sau reținem pentru fiecare N prefixul de lungime 6 al lui 2^N și apoi verificăm în cel mult 6 pași dacă M se poate obține ca prefix al lui 2^N).

Pentru interogările de tipul 3 M A B vom căuta binar în lista valorilor pentru care se obține prefixul M pentru a determina numărul de valori incluse în intervalul dat $[A; B]$.

Observații:

- Prefixele pentru puterile 2^N se pot determina și utilizând reprezentarea în virgulă mobilă long double și având grijă doar la păstrarea mantisei, păstrând exponentului o valoare mică.
- Se poate verifica și că pentru a obține prefixul de cel mult 6 cifre este suficient ca în reprezentarea cu numere mari să păstrăm cel mult, aproximativ 20 cifre.

12.9 Cod-sursă pentru problema P2

```
#include <fstream>
#define BAZA 1000000000
#define NMAX 10000000
#define MMAX 10000000
using namespace std;
ifstream  fin("p2.in");
ofstream fout("p2.out");
int Q, L, C, M, N, A, B, na, n;
```

```
int a[10000002], v[NMAX+2];
int nr[MMAX+2], poz2[MMAX+2], poz1[MMAX+2],T[MMAX+2];
long long int z;
int main()
{
    fin>>Q>>L;
    v[0]=1;
    nr[1]++;
    na=1; a[0]=1;
    for (n=1; n<=L; n++)
    {
        long long int t=0;
        for (int i=max(0,na-30); i<=na-1; i++)
        {
            t=t+2*a[i];
            a[i]=t%BAZA;
            t=t/BAZA;
        }
        while (t)
        {
            a[na]=t%BAZA;
            na++;
            t=t/BAZA;
        }

        z=0;
        for (int i=na-1; z<999999 && i>=0; i--)
            z=z*BAZA+a[i];
        while (z>999999)
              z=z/10;
        v[n]=z;
        while (z>0)
        {
            nr[z]++;
            z=z/10;
        }
    }
    poz1[0]=0; poz2[0]=0;
    for (int i=1; i<=1000000; i++)
    {
        poz1[i]=poz1[i-1]+nr[i-1];
        poz2[i]=poz1[i];
    }
    for (int i=0; i<=L; i++)
    {
        z=v[i];
        while (z>0)
        {
            poz2[z]++;
            T[poz2[z]]=i;
            z=z/10;
        }
    }
    for (int q=1; q<=Q; q++)
    {
        fin>>C>>M;
        if (C==1)
            fout<<T[poz1[M]+1]<<"\n";
        if (C==2)
        {
            fin>>N;
```

```
            int x=v[N];
            while (x>M) x=x/10;
            fout<<(x==M)<<"\n";
        }
        if (C==3)
        {
            fin>>A>>B;
            if (A>B) swap(A,B);
            if (nr[M]==0)
                fout<<"0\n";
            else
            {
                int p1=poz1[M]+1, p2=poz2[M], n1=-1, n2=-1;
                while (p1<=p2)
                {
                    int p=(p1+p2)/2;
                    if (T[p]>=A)
                        n1=p, p2=p-1;
                    else p1=p+1;
                }
                if (n1!=-1)
                {
                    p1=n1; p2=poz2[M];
                    while (p1<=p2)
                    {
                        int p=(p1+p2)/2;
                        if (T[p]<=B)
                            n2=p, p1=p+1;
                        else p2=p-1;
                    }
                }
                if (n1==-1 || n2==-1)
                   fout<<0<<"\n";
                else
                    fout<<n2-n1+1<<"\n";
            }
        }
    }
    return 0;
}
```

Capitolul 13

Barajul 4

13.1 Problema Farfurii

Propusă de: prof. Zoltan Szabo, Inspectoratul Școlar Județean Mureș

Gigel este un pasionat al ordinii. Când se duce în vizită la bunica, întotdeauna face ordine în dulapul de bucătărie al bunicii, unde farfuriile stau în 3 teancuri verticale, fără să fie, uneori, aranjate după mărimea lor. Ieri, când a vizitat-o pe bunica, s-a hotărât să aranjeze toate farfuriile într-un singur teanc, ordonate descrescător de la bază spre vârf, după diametru. Și pentru a da și o nuanță de joc acestei activități, s-a hotărât, că la fiecare mișcare va lua doar o farfurie din vârful unui teanc și va așeza în vârful unui alt teanc, și niciodată nu va pune o farfurie mai mare peste o farfurie mai mică.

Pentru a putea codifica mutările, cele trei teancuri de farfurii le vom denumi coloane și le vom numerota cu 1, 2 și 3. În orice moment, pe oricare dintre coloane pot exista sau nu farfurii. Dacă o coloană nu conține nicio farfurie, vom spune că este goală. Pe o coloană goală se poate pune orice farfurie ce poate fi accesată. Dacă o coloană conține cel puțin o farfurie, atunci în vârful acestei coloane se va putea așeza o nouă farfurie numai dacă diametrul noii farfurii este mai mic sau egal cu diametrul farfuriei din vârful coloanei.

Rezolvarea problemei va consta dintr-o succesiune de operații. O operație constă în extragerea unei farfurii de pe o coloană C_1 și plasarea acesteia pe o altă coloană C_2, respectând restricțiile de mai sus. Dată fiind o coloană C, spunem că succesiunea de operații este corectă dacă, după executarea tuturor operațiilor, toate farfuriile se vor afla pe coloana C, în ordinea descrescătoare a diametrelor de la bază spre vârf.

Cerințe

Cunoscând configurația celor trei teancuri inițiale de farfurii, precum și numărul C al coloanei pe care trebuie să se afle la final farfuriile, scrieți un program care să determine o succesiune corectă de operații.

Date de intrare

Fișierul de intrare `farfurii.in` conține 4 linii. Pe linia i ($1 \leq i \leq 3$) se află numărul natural N_i, ce reprezintă numărul de farfurii din coloana i, urmat de N_i numere naturale separate prin câte un spațiu, reprezentând dimensiunile diametrelor farfuriilor de pe coloana i de la bază spre vârf (dacă $N_i = 0$, atunci nu există farfurii pe coloana i). Pe ultima linie se află numărul C (1, 2 sau 3), reprezentând coloana pe care vor fi plasate toate farfuriile în final.

Date de ieșire

Fișierul de ieșire `farfurii.out` va conține operațiile, în ordinea efectuării lor, câte o operație pe o linie. O operație va fi descrisă prin două numere separate prin spațiu C_1 C_2 cu semnificația: „farfuria din vârful coloanei C_1 se mută la vârful coloanei C_2".
Pe ultima linie se vor scrie numerele 0 0, separate prin spațiu.

Restricții

- $1 \leq C_1, C_2, C \leq 3$
- $1 \leq$ dimensiunile diametrelor farfuriilor ≤ 100
- $2 \leq N_1 + N_2 + N_3 \leq 18$
- Soluția nu este unică. Se acceptă orice soluție corectă.
- Nu se cere optimizarea numărului de operații, dar rezolvarea problemei trebuie să se încadreze în timpul de execuție specificat.

#	Puncte	Restricții
1	4	Toate farfuriile sunt de aceeași dimensiune.
2	11	Există doar două dimensiuni diferite de farfurii.
3	12	Există doar trei dimensiuni diferite de farfurii.
4	22	Toate farfuriile sunt ordonate pe o singură coloană.
5	14	Toate farfuriile formează două coloane crescătoare de la bază spre vârf.
6	18	Toate farfuriile au dimensiuni diferite.
7	19	Fără restricții suplimentare

Exemple

farfurii.in	farfurii.out
2 1 7 3 5 1 9 0 3	2 3 1 3 2 1 2 3 1 2 1 3 2 3 0 0

Explicație

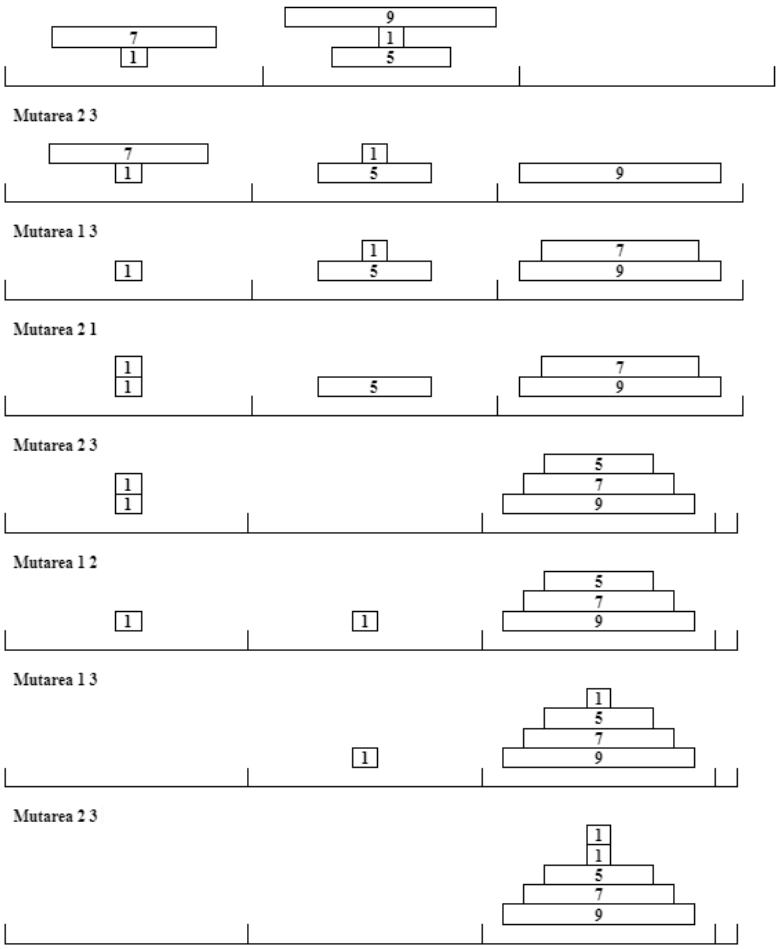
Farfuriile trebuie să fie plasate pe coloana $C = 3$. Efectul executării operațiilor este ilustrat în figură:

13.2 Rezolvarea problemei Farfurii

Soluția 1 - prof. Zoltan Szabo

Problema *farfurii* este o adaptare a problemei *turnurilor din Hanoi*. În problema clasică toate discurile sunt pe o singură tijă ordonate strict descrescător. La această variantă avem trei deosebiri majore:

- farfuriile nu formează un singur teanc;
- farfuriile nu sunt ordonate;



- farfuriile pot avea dimensiuni egale.

Cazul general se obține prin includerea completă a celor trei condiții de mai sus, însă prin păstrarea doar a câtorva dintre condiții, se obțin câteva probleme particulare cu un nivel de dificultate mai scăzut, aducând un punctaj parțial corespunzător. Menționăm, că pentru fiecare caz, pot exista mai multe moduri de rezolvare, soluția nu este unică. Avem următoarele cazuri:

1. **Toate farfuriile au dimensiuni egale (4 puncte)** În acest caz toate farfuriile de pe celelalte două coloane diferite de coloana C se vor muta pe coloana C.
2. **Există doar două dimensiuni diferite de farfurii (11 puncte)**
   (a) **Toate farfuriile sunt pe o singură coloană (5 puncte)** Avem la dispoziție două coloane libere, deci toate elementele le vom separa după dimensiunea lor pe cele două coloane: cele *mici* pe o coloană și cele *mari* pe o altă coloană. Apoi vom construi coloana ordonată de farfurii pe coloana C. În funcție de coloana de destinație, se vor obține trei moduri diferite de mutare a farfuriilor, însă fără a avea un grad de dificultate ridicat.
   (b) **Farfuriile sunt dispersate pe mai multe coloane (6 puncte)** În acest caz, coloana 1 va fi punctul de colectare a farfuriilor *mici*, iar coloana 2 va fi punctul de colectare a farfuriilor *mari*. În primul pas, toate farfuriile *mici* din vârful coloanei 2 se vor muta pe coloana 1. Astfel am obținut proprietatea, că prima coloană are în vârf o valoare de farfurie *mică*, iar coloana 2 are în vârf o farfurie *mare*. Coloanele 1 și 2 nu sunt ordonate, dar vârful lor permite plasarea tuturor farfuriilor de pe coloana 3 sortate pe cele două mărimi. Astfel putem elibera coloana 3. Urmează să sortăm elementele de pe coloana 2, mutând toate mărimile *mari* pe coloana 3, iar cele mici pe coloana 1. Elementele sunt separate: cele *mici* pe coloana 2 și cele *mari* pe coloana 3. Mutarea lor pe coloana C se va realiza ca la cazul precedent.
3. **Există doar trei dimensiuni de farfurii (12 puncte)** În acest caz, coloana 1 va colecta farfuriile *mici*, coloana 2 farfuriile *mijlocii*, iar coloana 3 farfuriile *mari*. În primul pas vom

muta eventualele farfurii *mici* din vârful coloanei 2 pe coloana 1. În pasul al doilea vom muta din vârful coloanei 3, toate farfuriile *mici* pe coloana 1, iar farfuriile *mijlocii* pe coloana 2. Am obținut proprietatea ca fiecare dintre cele trei coloane are în vârf: coloana 1 farfurie *mică*, coloana 2 farfurie *mijlocie*, coloana 3 farfurie *mare*. În continuare vom separa elementele neordonate de sub vârf pe cele trei coloane. Începem cu coloana 1. Verificăm care este prima valoare de sub coloana constantă *mică*. Dacă această valoare este *mijlocie*, atunci coloana de valori *mici* se va muta pe coloana 3, iar valoarea *mijlocie* pe coloana 2, iar dacă valoarea este *mare*, atunci coloana de valori *mici* se va muta pe coloana 2, iar valoarea *mare* pe coloana 3. După care coloana de valori *mici* se mută înapoi pe coloana 1. Acest procedeu se va relua până când toate valorile diferite de valoare *mică* vor ajunge pe coloanele 2 sau 3. Cu un algoritm asemănător vom muta valorile *mici* și *mari* de pe coloana 2, apoi vom relua un algoritm asemănător și pentru coloana 3, având în vedere ca niciodată să nu punem farfurii mai mari peste farfurii mai mici.

4. **Toate farfuriile sunt ordonate pe o singură coloană (22 puncte)**
   (a) **Farfuriile sunt ordonate strict crescător și se vor muta pe o altă coloană (2 puncte)** În acest caz, toate elementele, printr-o simplă parcurgere se vor muta pe coloana C.
   (b) **Farfuriile sunt ordonate strict crescător și se vor muta pe aceeași coloană (4 puncte)** Toate elementele, printr-o simplă parcurgere se vor muta pe o altă coloană, apoi cu algoritmul turnurilor din Hanoi se vor remuta farfuriile pe coloana inițială.
   (c) **Farfuriile sunt ordonate strict descrescător, și se vor muta pe o altă coloană (4 puncte)** În acest caz, se aplică algoritmul turnurilor din Hanoi pentru a muta farfuriile pe coloana C.
   (d) **Farfuriile sunt ordonate crescător și se vor muta pe o altă coloană (2 puncte)** Există și elemente egale. În acest caz, la fel ca la subpunctul 4.*a*., toate elementele, printr-o simplă parcurgere se vor muta pe coloana C.
   (e) **Farfuriile sunt ordonate crescător și se vor muta pe aceeași coloană (5 puncte)** Există și elemente egale. Toate elementele, printr-o simplă parcurgere se vor muta pe o altă coloană, apoi cu algoritmul turnurilor din Hanoi modificat să fie funcționabil și pentru numere egale se vor remuta farfuriile pe coloana inițială.
   (f) **Farfuriile sunt ordonate descrescător, și se vor muta pe o altă coloană (5 puncte)** În acest caz, se aplică algoritmul turnurilor din Hanoi modificat să fie funcționabil și pentru dimensiuni egale pentru a muta farfuriile pe coloana C. Modificarea adusă agoritmului se poate realiza în mai multe feluri, de exemplu, pentru fiecare element folosim o pereche pentru valoare și frecvență, astfel obținem doar elemente învecinate în șir cu valori distincte. O altă modalitate este de a folosi algoritmul turnurilor din Hanoi în care tipărirea se realizează în mod repetat cât timp elementul din vârful stivei este aceeași valoare.

5. **Toate farfuriile formează două coloane crescătoare de la bază spre vârf (14 puncte)**
   (a) **Farfuriile sunt ordonate strict crescător pe două coloane și se vor muta pe a treia coloană (2 puncte)** În acest caz se realizează o interclasare a celor două șiruri.
   (b) **Farfuriile sunt ordonate strict crescător pe două coloane și se vor muta pe una dintre coloanele inițiale (4 puncte)** Teancul de farfurii se obține prin metoda interclasării pe a treia coloană, apoi cu algoritmul turnurilor din Hanoi se vor remuta farfuriile pe coloana dorită.
   (c) **Farfuriile sunt ordonate crescător pe două coloane și se vor muta pe a treia coloană (3 puncte)** În acest caz se realizează o interclasare a celor două șiruri, ținând cont că există și elemente egale.
   (d) **Farfuriile sunt ordonate crescător pe două coloane și se vor muta pe una dintre coloanele inițiale (5 puncte)** Prin metoda interclasării se obține teancul de farfurii pe a treia coloană, apoi cu algoritmul turnurilor din Hanoi adaptat și pentru numere egale se vor remuta farfuriile pe coloana dorită.

6. **Toate farfuriile au dimensiuni diferite (18 puncte)** Se referă la cazul general cu toate dimensiunile de farfurii distincte două câte două. Studiem valorile elementelor din vârf. Toate elementele fiind distincte, vom avea câte un element minim, maxim și mijlociu. Știind că cel mai mic element se potrivește peste celelalte două, încercăm să maximizăm adâncimea coloanei

care conține vârful minim, având toate elementele mai mici decât elementul mijlociu. O astfel de coloană respectă proprietățile jocului Hanoi clasic, și vom aplica astfel: - coloana cu vârful minim, **fără elementul de la bază** se mută pe coloana care conține maximul - **elementul de la baza** coloanei minimului se mută pe coloana care continue elementul mijlociu. După ce s-au efectuat pașii de mai sus, configurația nouă devine mai „bună" decât configurația anterioară, eliberând un nou element de sub coloana ce a fost mutată, iar elementele mutate păstrează proprietatea Hanoi în continuare. Aceste mutări sunt ireversibile, deci nu putem ajunge la ciclu infinit. Astfel pas cu pas se reconstruiește toată coloana. În cazul în care coloana cu toate elementele s-a format pe o altă tijă decât cea din problemă, mai apelăm încă un Hanoi clasic și problema va fi rezolvată complet.
După cum se știe, algoritmul turnurilor din Hanoi cu n discuri necesită $2^n - 1$ mutări, deci turnul nostru se va construi în cel mai nefavorabil caz în $2^1 + 2^2 + 2^3 + \cdots + 2^n = 2^{n+1} - 1$ mutări, deci complexitatea algoritmului este $O(2^n)$.

7. **Fără restricții suplimentare (19 puncte)** Se referă la cazul general când există dimensiunile de farfurii egale. Algoritmul este similar cu cel descris la punctul 6, aplicând modificarea algoritmului turnurilor din Hanoi pentru numere egale, descrisă la punctul $4.f$.

Soluția 2 - stud. Dumitru Ilie

Vom pleca de la explicația soluției optime a problemei turnurilor din Hanoi. Vrem să mutăm cele N discuri de pe turnul A pe turnul C într-un număr minim de mutări. Pentru a face asta va trebui mai întâi să mutăm cele $N - 1$ discuri mai mici de pe turnul A pe turnul B, apoi vom muta cel mai mare disc pe turnul C și apoi toate discurile de pe turnul B pe turnul C. Pentru a muta cele $N - 1$ discuri vom folosi același raționament, recursiv, mai puțin în cazul în care nu trebuie să mutăm nici un disc ($N = 0$).

Acum să ne întoarcem la problema noastră. Pentru a muta toate discurile, în ordine crescătoare pe un anume turn, să îl numim *target*, vrem să poziționăm cel mai mare disc la baza lui. Întâi trebuie să știm unde se află cel mai mare disc. Avem 2 cazuri:

1. Discul cel mai mare este pe un alt turn decât *target*. În acest caz vrem să mutăm toate discurile de deasupra lui, și discurile de pe turnul *target* pe cel de-al treilea turn. Apoi vrem să mutăm discul cel mai mare pe turnul *target* și apoi vrem să mutăm toate discurile înapoi pe *target*. Observăm un tipar recursiv.
2. Discul cel mai mare este pe turnul *target*. În acest caz avem două subcazuri. Fie este deja la baza turnului *target*, fie nu.
   (a) Dacă este la bază putem să îl ignorăm complet (fiind cel mai mare putem pune orice alt disc deasupra lui) și putem rezolva problema pentru $N - 1$ discuri.
   (b) Dacă însă nu este la bază, avem sub el discuri mai mici. Ce vom face este să îl mutăm pe unul dintre turnurile auxiliare, urmat de a scoate toate discurile de pe turnul *target* și a le muta pe alt disc auxiliar. Acum putem muta cel mai mare disc pe *target* și apoi toate celelalte peste el. Am presupus totuși că putem muta discul cel mai mare pe un turn auxiliar. Dacă nu îl putem muta instant, vom muta tot ce e deasupra lui și tot ce e pe primul turn auxiliar pe cel de-al doilea turn auxiliar iar acum acesta poate fi mutat.

Avem deci un algoritm recursiv care încearcă să mute mereu discul maxim. Se poate demonstra că în cazul problemei originale, acest algoritm este chiar optim (din punctul de vedere al numărului de mutări). În cazul problemei noastre avem de făcut decizii legate de care turn auxiliar este folosit cu ce scop. Aceste decizii influențează numărul de mutări făcute, dar pentru restricțiile mici ale problemei nu este nevoie să le facem într-un mod inteligent, orice alegere este suficient de bună.

Ce facem dacă avem valori egale?

Aproape nimic diferit („If it works don't fix it" - citat din popor). Singura diferență constă în alegerea discului maxim, deoarece acum există mai multe „discuri maxime". O soluție care merge foarte bine este să alegem primul disc maxim găsit, în ordine din vârful turnului spre bază.

Complexitatea finală a acestei soluții este de aproximativ la $O(N \cdot 2^N)$. Probabil că implementări mai rapide există, dar aceasta este suficient de bună.

Pentru cei doritori de provocări puteți încerca să rezolvați optim problema (din punctul de vedere al numărului de mutări) pentru orice caz.

13.3 Cod-sursă pentru problema Farfurii

```
//prof. Zoltan Szabo
#include <fstream>
using namespace std;
int n[3], C, a[3][22][2], k=0, nmax, nrsol=0,sol[2000000][2];
const int val=1, frecv=0;
ifstream fin("farfurii.in");
ofstream fout("farfurii.out");

bool ok(int C)
// e ok daca avem o coloana descresc. pe coloana d si celelalte coloane sunt goale
{
    if(n[(C+1)%3]!=0 or n[(C+2)%3]!=0)
        return false; // nu e ok daca sunt elemente pe alte coloane
    else
        for(int i=0;i<n[C]-1;i++)
            if(a[C][i][val]<a[C][i+1][val])
                return false; // nu e ok daca coloana nu e descrescatoare de la baza spre varf
    return true;
}

int minim()
// returneaza coloana ce are farfuria cea mai mica in varf
{
    if (a[0][n[0]][val]<=a[1][n[1]][val] and a[0][n[0]][val]<=a[2][n[2]][val])
        return 0;
    if (a[1][n[1]][val]<=a[0][n[0]][val] and a[1][n[1]][val]<=a[2][n[2]][val])
        return 1;
    return 2;
}

int maxim()
// returneaza coloana ce are farfuria cel mai mare in varf
{
    if (a[2][n[2]][val]>=a[1][n[1]][val] and a[2][n[2]][val]>=a[0][n[0]][val])
        return 2;
    if (a[1][n[1]][val]>=a[2][n[2]][val] and a[1][n[1]][val]>=a[0][n[0]][val])
        return 1;
    return 0;
}

int adancime(int x, int y)
// numarul maxim de farfurii ordonate corect
// pe coloana x cu toate elementele mai mici sau egale cu farfuria din varful coloanei y
{
    int d=1;
    int k=n[x];
    while (k>1 and a[x][k-1][val]>a[x][k][val] and a[x][k-1][val]<=a[y][n[y]][val])
    {
        --k;
        ++d;
    }
    return d;
```

```
}

void hanoi(int deep,int x,int y, int z)
// generam formal mutarile pentru coloana mutata cu functia mutare(deep,x,y);
{
    if (deep)
    {
        if (deep==1)
        {
            for (int i=1;i<=a[x][n[x]][frecv];i++)
                if (nrsol==0 or (nrsol>0 and sol[nrsol][1]!=x+1))
                {
                    sol[++nrsol][0]=x+1;
                    sol[nrsol][1]=y+1;
                }
                else
                {
                    sol[nrsol][1]=y+1;
                    if (sol[nrsol][1]==sol[nrsol][0])
                        nrsol--;
                }
            if (a[x][n[x]][val]==a[y][n[y]][val])
            {
                a[y][n[y]][frecv]+=a[x][n[x]][frecv];
                nmax--;
                n[x]--;
            }
            else
            {
                n[y]++;
                a[y][n[y]][val]=a[x][n[x]][val];
                a[y][n[y]][frecv]=a[x][n[x]][frecv];
                n[x]--;
            }
        }
        else
        {
            hanoi(deep-1,x,z,y);
            for (int i=1;i<=a[x][n[x]][frecv];i++)
                if (nrsol==0 or (nrsol>0 and sol[nrsol][1]!=x+1))
                {
                    sol[++nrsol][0]=x+1;
                    sol[nrsol][1]=y+1;
                }
                else
                {
                    sol[nrsol][1]=y+1;
                    if(sol[nrsol][1]==sol[nrsol][0])
                        nrsol--;
                }
            if (a[x][n[x]][val]==a[y][n[y]][val])
            {
                a[y][n[y]][frecv]+=a[x][n[x]][frecv];
                nmax--;
                n[x]--;
            }
            else
            {
                n[y]++;
                a[y][n[y]][val]=a[x][n[x]][val];
                a[y][n[y]][frecv]=a[x][n[x]][frecv];
```

```
                n[x]--;
            }
            hanoi(deep-1,z,y,x);
        }
    }
}

int main()
{ int deep,i,j,x,y,z;
  a[1][0][val]=999998; a[1][0][frecv]=1;
  a[2][0][val]=999999; a[2][0][frecv]=1;
  a[0][0][val]=999997; a[3][0][frecv]=1;
  for (int i=0;i<3;i++)
      {
        fin>>n[i];
        deep=0;
        for (j=1;j<=n[i];++j)
            {
            fin>>x;
            if (x==a[i][deep][val])
               a[i][deep][frecv]++;
            //farfuriile invecinate cu dimensiuni egale le memoram impreuna, marim frecventa cu 1
               else
               {
                ++deep;               // creste inaltimea coloanei
                a[i][deep][val]=x;    // memoram dimensiunea farfuriei
                a[i][deep][frecv]=1;  // frecventa e 1
               }
            }
        n[i]=deep;  // numarul de elemente
       }
    nmax=n[0]+n[1]+n[2];
    fin>>C;
    while (!ok(C-1))
        // pozitiile coloanelor din problema sunt 1,2,3, iar in tablou 0, 1, 2
       {
        x=minim(); // numarul coloanei cu farfuria cea mai mica in varf
        z=maxim(); // numarul coloanei cu farfuria cea mai mare in varf
        y=3-x-z;   // numarul tijei care are in varf discul mijlociu
        deep=adancime(x,y);
        // numarul de elemente din vf tijei x, ce sunt mai mic decat discul din varful lui y
        if (deep!=nmax)
          {
            hanoi(deep-1,x,z,y); // se aplica hanoi pentru a genera mutarile necesare infisier
            hanoi(1,x,y,z); // se aplica hanoi pentru a genera mutarile necesare infisier
          }
          else
          hanoi(deep,x,C-1,4-x-C); // se aplica hanoi pentru a genera mutarile necesare infisier
       }
    for (int i=1;i<=nrsol;i++)
         fout<<sol[i][0]<<" "<<sol[i][1]<<endl;
    fout<<"0 0"<<endl;
    fout.close();
    return 0;
}
```

```
// stud. Ilie Dumitru
#include<cstdio>

FILE* f = fopen("farfurii.in", "r"), *g = fopen("farfurii.out", "w");
int n[4];
```

```
int st[4][20];

void move(int start, int end)
{
 fprintf(g, "%d %d\n", start, end);
 st[end][n[end]++] = st[start][--n[start]];
}

void solve(int* min, int target)
{
 int i, j, maxI = -1, maxJ, max = -1;
 for (j = 1; j < 4; ++j)
      for (i = n[j] - 1; i >= min[j]; --i)
          {
           if (st[j][i] > max)
              {
                max = st[j][i];
                maxI = i;
                maxJ = j;
              }
          }
    if (maxI == -1)
        return;
    if (min[target] < n[target] && max == st[target][min[target]])
       {
        ++min[target];
        solve(min, target);
       }
       else
       if (maxJ == target)
          {
           int next[4] = {0, min[1], min[2], min[3]};
           int ids[3] = {target, target % 3 + 1, (target + 1) % 3 + 1};
           next[target] = maxI + 1;
           solve(next, ids[1]);
           move(ids[0], ids[2]);
           solve(min, target);
          }
          else
          {
           int next[4] = {0, min[1], min[2], min[3]};
           int lastId = 6 - target - maxJ;
           next[maxJ] = maxI + 1;
           solve(next, lastId);
           move(maxJ, target);
           solve(min, target);
          }
}

int main()
{
 int k, i, min[4] = {0, 0, 0, 0};
 for (k = 1; k < 4; ++k)
     {
      fscanf(f, "%d", n + k);
      for (i = 0; i < n[k]; ++i)
          fscanf(f, "%d", st[k] + i);
     }
 fscanf(f, "%d", &k);
 solve(min, k);
 fprintf(g, "0 0\n");
```

```
 fclose(f); fclose(g);
 return 0;
}
```

13.4 Problema Numbers

Propusă de: prof. Dan Pracsiu, Liceul Teoretic „Emil Racoviță" Vaslui

Pentru un număr natural nenul x, definim operația de rotire cu k cifre ca fiind mutarea la final a primelor k cifre ale lui x, în aceeași ordine. Operația de rotire cu k cifre este posibilă numai dacă numărul de cifre ale lui x este strict mai mare decât k.

Exemple:

- aplicând operația de rotire cu 3 cifre numărului 123456 obținem 456123;
- aplicând operația de rotire cu o cifră numărului 222 obținem 222;
- aplicând operația de rotire cu două cifre numărului 1000 obținem 10.

Se dă un șir de n numere naturale nenule. Se pot aplica operații de rotire unor numere din șir astfel încât să nu existe două numere rotite cu același număr de cifre.

Cerințe

Să se determine suma maximă care se poate obține însumând numerele din șirul dat, după rotirea convenabilă a unor numere din șir, conform specificațiilor din enunț.

Date de intrare

Fișierul de intrare `numbers.in` conține pe prima linie numărul n, iar pe a doua linie un șir de n numere naturale nenule, separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire `numbers.out` va conține o singură linie pe care va fi scrisă suma maximă obținută.

Restricții

- $1 \leq n \leq 10\,000$
- Numerele din șir sunt numere naturale nenule și au cel mult 15 cifre.

#	Puncte	Restricții
1	20	$1 \leq n \leq 8$
2	60	$10 \leq n \leq 200$
3	20	Fără restricții suplimentare

Exemple

numbers.in	numbers.out	Explicații
3 2 15 31	84	Se rotește 15 cu o cifră și se obține 51. $2 + 51 + 31 = 84$.
4 17 15 204 1	507	Se rotește 17 cu o cifră și 204 cu două cifre. Se obține suma $71 + 15 + 420 + 1 = 507$.
4 21 31 42 87	181	Nu se rotește niciun număr. Se obține suma maximă $21 + 31 + 42 + 87 = 181$.

4 23 113 45719 55588	183469	Se rotește 23 cu o cifră, 113 cu două cifre, 45719 cu 4 cifre și 55588 cu 3 cifre. Se obține suma maximă $32 + 311 + 94571 + 88555 = 183469$.

13.5 Rezolvarea problemei Numbers

Subtask 1 - $O(n!)$ - 20p

Deoarece n este foarte mic, atunci punctajul maxim pentru acest subtask se poate obține cu metoda backtracking.

Subtask 2 - $O(n \cdot 2^{15})$ - 80p

Deoarece n este mai mic sau egal cu 200, atunci vom utiliza metoda programării dinamice pe stări exponențiale. Construim pentru aceasta matricea dp cu n linii și 2^{15} coloane în care $dp[i][mask]$ = suma maximă care se poate obține din primele i numere, unele dintre ele fiind rotite cu k cifre, unde valorile k folosite sunt marcate cu 1 în $mask$.

Parcurgem fiecare număr din șirul a și pentru fiecare $a[i]$ avem opțiunea să nu-l rotim, deci îl adunăm la sumă chiar pe $a[i]$, sau să-l rotim cu j poziții, unde j este setat cu 0 în $mask$, caz în care la sumă putem adăuga valoarea lui $a[i]$ rotită la stânga cu j cifre, iar bitul j se setează cu 1 în noua mască. Deci

$dp[i][mask] = max(dp[i-1][mask] + a[i], dp[i-1][mask|2^j] + RotateLeft(a[i], j))$

unde prin | se înțelege operatorul OR pe biți, iar $RotateLeft(a[i], j)$ este funcția care returnează valoarea lui $a[i]$ rotită de j ori.

Soluția problemei se găsește în $\max(dp[n][i], i = 1 \ldots 2^{15} - 1)$

Subtask 3 - 100p

Se observă că sunt în total cel mult 14 numere care se vor roti. În consecință, pentru fiecare $j = 1 \ldots 14$, vom reține în $L[j]$ cele mai mari 14 numere care se pot obține din șirul de numere prin rotirea cu exact j cifre la stânga. În acest fel, vom putea lua în considerare numai $14 \times 14 = 196$ de numere. Rețineți faptul că listele $L[j]$, cu $j = 1 \ldots 14$, nu sunt neapărat disjuncte.

În continuare se poate aplica pentru cele cel mult 196 de numere ideea de la subtaskul 2.

13.6 Cod-sursă pentru problema Numbers

```
#include <bits/stdc++.h>
using namespace std;
using i64 = long long;
const int N_MAX = 100000;
const int SHIFT_MAX = 14;

int N;
i64 A[N_MAX + 5][SHIFT_MAX + 5];
i64 dp[SHIFT_MAX * SHIFT_MAX + 5][1 << SHIFT_MAX];
vector <int> candidates;

int num_of_digs(i64 x) {
    int ret = 0;
    do {
```

```
        ret++;
        x /= 10;
    }while(x);
    return ret;
}

i64 shift_left(i64 x, int shift) {
    int d = num_of_digs(x);
    i64 p10 = 1;
    for(int i = 1; i < d; i++) {
        p10 *= 10;
    }
    while(shift--) {
        x = (x % p10) * 10 + x / p10;
    }
    return x;
}

vector <int> get_top_SHIFT_MAX(int shift) {
    vector <int> ret;
    vector < pair<i64, int> > nums;
    for(int i = 1; i <= N; i++) {
        if(num_of_digs(A[i][0]) > shift) {
            nums.push_back(make_pair(A[i][shift] - A[i][0], i));
        }
    }
    sort(nums.begin(), nums.end(), greater<pair<i64, int>>());
    for(int i = 0; i < SHIFT_MAX && i < nums.size(); i++) {
        ret.push_back(nums[i].second);
    }
    return ret;
}

int main() {
#ifndef LOCAL
    freopen("numbers.in", "r", stdin);
    freopen("numbers.out", "w", stdout);
#endif
ios::sync_with_stdio(false);
cin.tie(0); cout.tie(0);

cin >> N;
i64 sum = 0;
for (int i = 1; i <= N; i++) {
     cin >> A[i][0];
     sum += A[i][0];
     int digs = num_of_digs(A[i][0]);
     for (int d = 1; d < digs; d++) {
          A[i][d] = shift_left(A[i][0], d);
     }
}

for (int i = 1; i <= SHIFT_MAX; i++) {
     auto ret = get_top_SHIFT_MAX(i);
     candidates.insert(candidates.end(), ret.begin(), ret.end());
}

sort(candidates.begin(), candidates.end());
candidates.erase(unique(candidates.begin(), candidates.end()), candidates.end());
candidates.insert(candidates.begin(), 0);
```

```
for (int i = 1; i < candidates.size(); i++) {
    int digs = num_of_digs(A[candidates[i]][0]);
    for (int conf = 0; conf < (1 << SHIFT_MAX); conf++) {
        dp[i][conf] = dp[i - 1][conf];
        for (int shift = 0; shift < SHIFT_MAX && shift + 1 < digs; shift++) {
            if (conf & (1 << shift))
              dp[i][conf]=max(dp[i][conf],
                        dp[i-1][conf^(1<<shift)]+A[candidates[i]][shift+1]-A[candidates[i]][0]);
         }
     }
}

cout << sum + dp[candidates.size() - 1][(1 << SHIFT_MAX) - 1] << "\n";
return 0;
}
```

13.7 Problema Nyse

Propusă de: stud. Andrei Onuț, Yale University, S.U.A.

Pe Bursa de Valori din New York, cunoscută și sub numele de NYSE – New York Stock Exchange, există un singur tip de acțiune ce se poate tranzacționa. Se cunoaște prețul zilnic al unei acțiuni de acest tip pentru o perioadă de N zile consecutive: p_i reprezintă prețul unei acțiuni în ziua i ($1 \leq i \leq N$), exprimat în dolari.

William urmărește ce se întâmplă pe bursă, iar în fiecare zi i poate tranzacționa, în total, cel mult L_i acțiuni. Prin urmare, dacă în ziua i vinde S acțiuni (la prețul de p_i dolari fiecare) și cumpără B acțiuni (la prețul de p_i dolari fiecare), trebuie respectate inegalitățile: $0 \leq S$, $0 \leq B$ și $0 \leq S + B \leq L_i$. De asemenea, numerele S și B trebuie să fie numere naturale. **Atenție: nu este nevoie ca William să dețină cel puțin o acțiune pentru a putea vinde una.** De exemplu, este posibil ca prima tranzacție pe care William o efectuează să fie vânzarea unei (sau a mai multor) acțiuni.

Pentru orice i, spunem că, la finalul zilei i, William *și-a închis pozițiile*, dacă numărul de acțiuni cumpărate în primele i zile este egal cu numărul de acțiuni vândute în primele i zile (incluzând și eventualele tranzacții efectuate în ziua i). În urma unei serii de tranzacții, *profitul* este egal cu diferența dintre suma totală de dolari obținută din vânzarea acțiunilor și respectiv suma totală de dolari utilizată pentru cumpărarea acțiunilor.

William are nevoie să răspundă, în ordine, la Q întrebări de forma: considerându-se o valoare naturală x, să se determine indicele minim al unei zile j (unde $1 \leq j \leq N$), astfel încât după o serie de tranzacții în zilele $1, 2, \ldots, j$, profitul obținut să fie de cel puțin x dolari, iar el să își fi închis pozițiile la finalul zilei j. Dacă nu există niciun astfel de indice j, atunci se consideră că $j = -1$.

William știe încă de la începutul primei zile prețurile pentru întreaga perioadă de N zile, astfel că își poate planifica optim tranzacțiile, în funcție de fiecare întrebare primită.

Cerințe

Scrieți un program care, cunoscând N, prețul zilnic al unei acțiuni pentru cele N zile, precum și limitele zilnice de tranzacționare, răspunde corect la Q întrebări de forma descrisă.

Cele Q întrebări sunt independente între ele, în sensul că pentru fiecare întrebare se poate alege o altă serie de tranzacții din care să rezulte cea mai mică valoare a lui j, în cazul în care aceasta există (când $j \neq -1$).

Date de intrare

Fișierul de intrare `nyse.in` conține pe prima linie numărul natural N. Pe următoarea linie se află N numere naturale, reprezentând, în ordine: p_1, p_2, ..., p_N. Pe următoarea linie se află N numere naturale, reprezentând, în ordine: L_1, L_2, ..., L_N. Pe următoarea linie se află numărul natural Q. Pe următoarele Q linii se află întrebările, câte o întrebare pe fiecare linie. O linie care reprezintă o întrebare conține un singur număr natural x, cu semnificația de mai sus. Numerele aflate pe aceeași linie a fișierului de intrare sunt separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire `nyse.out` conține Q linii. Pe cea de a k-a linie se află răspunsul pentru cea de a k-a întrebare descrisă în fișierul de intrare (unde $1 \leq k \leq Q$).

Restricții

- $1 \leq N \leq 900\,000$ și $1 \leq Q \leq 100\,000$.

- $1 \leq p_i \leq 1\,000\,000\,000$ și $0 \leq L_i \leq 1\,000$, pentru fiecare i: $1 \leq i \leq N$.
- $0 \leq x \leq 10^{18}$, pentru fiecare dintre cele Q întrebări.
- În fiecare zi, WILLIAM are și opțiunea de a nu face tranzacții.

#	Puncte	Restricții
1	4	$p_1 = p_2 = \ldots = p_N$
2	12	$N \leq 1\,000$ și $L_1 = L_2 = \ldots = L_N = 1$
3	17	$N \leq 1\,000$
4	11	$\max(p_1, p_2, \ldots, p_N) - \min(p_1, p_2, \ldots, p_N) \leq 25$
5	18	$99\,000 \leq N \leq 100\,000$
6	14	$N > 100\,000$ și $L_1 = L_2 = \ldots = L_N = 1$
7	24	Fără restricții suplimentare

Exemple

nyse.in	nyse.out
2 10 5 3 3 1 14	2
5 10 1 20 21 25 1 1 1 1 1 1 20	4
5 10 12 5 113 343 1 2 3 2 1 4 0 1000 345 21	1 -1 5 4

Explicații

Primul exemplu: Se cunoaște prețul acțiunii pentru o perioadă de două zile consecutive ($N = 2$). În prima zi, o acțiune valorează $p_1 = 10$ dolari, iar în cea de a doua zi, o acțiune valorează $p_2 = 5$ dolari. Atât în prima zi, cât și în cea de a doua, se pot tranzacționa cel mult trei acțiuni ($L_1 = L_2 = 3$).

Avem de răspuns la o singură întrebare ($Q = 1$), pentru care $x = 14$.

În cadrul întrebării, WILLIAM alege să vândă trei acțiuni (la prețul de 10 dolari fiecare) în prima zi și apoi să cumpere trei acțiuni (la prețul de 5 dolari fiecare) în a doua zi. Astfel, la finalul celei de a doua zile, numărul de acțiuni cumpărate este egal cu numărul de acțiuni vândute, adică trei, deci WILLIAM și-a închis pozițiile. Profitul maxim obținut după primele două zile este egal cu: $3 \cdot 10 - 3 \cdot 5 = 15$ dolari.

Al doilea exemplu: Profitul maxim care se poate obține la finalul celei de a cincea zile este de 35 de dolari.

Al treilea exemplu: Profitul maxim care se poate obține la finalul celei de a cincea zile este de 556 de dolari.

13.8 Rezolvarea problemei Nyse

Prima observație

Putem să cumpărăm și să vindem (în orice ordine) o acțiune dacă: $X \leq Y$, unde X este prețul la care cumpărăm și Y este prețul la care vindem. În urma acestei tranzacții, se obține un profit (net) de $(Y - X) \geq 0$ dolari. Prin urmare, dacă avem două prețuri, notate cu a și b, alegem să cumpărăm la $\min(a, b)$ și să vindem la $\max(a, b)$ – profitul, în această situație, este egal cu $\max(a, b) - \min(a, b)$ dolari.

Soluție pentru primul subtask

Cum prețul acțiunii este constant pentru întreaga perioadă de N zile ($p_1 = p_2 = \ldots = p_N$), înseamnă că nu se poate obține un profit pozitiv (adică mai mare strict decât 0). Prin urmare, profitul maxim care se poate obține la finalul fiecărei zile i este de 0 (dolari). De vreme ce $i \in \{1, 2, \ldots, n\}$, înseamnă că profitul maxim care se poate obține la finalul primei zile este tot de 0 dolari.

În cadrul fiecărei întrebări, dacă valoarea x citită este egală cu 0, deducem că $j = 1$. Altfel (când $x > 0$), $j = -1$.

A doua observație

Să presupunem că prețurile unei acțiuni pentru patru zile distincte sunt: a, b, c, respectiv d și că limita de tranzacționare din fiecare zi este egală cu 1. Fără a pierde din generalitate, putem asuma că $a \leq b \leq c \leq d$. Astfel, strategia care rezultă în profitul maxim în acest scenariu este să cumpărăm (câte o acțiune) la prețurile a și b și să vindem (câte o acțiune) la prețurile c și d. Prin urmare, obținem un profit de $(c + d) - (a + b)$ dolari – se cumpără două acțiuni și se vând tot două acțiuni, deci ne-am închis pozițiile.

Prin contradicție, să presupunem că nu aceasta ar fi strategia optimă. Astfel, să presupunem, de exemplu, că strategia optimă ar fi să cumpărăm la prețurile a și c și să vindem la prețurile b și d. Astfel, se obține un profit egal cu $(b+d)-(a+c)$. Am presupus că aceasta este soluția optimă, astfel că $b+d-a-c > c+d-a-b$, dacă folosim informațiile din paragraful precedent. Această inegalitate este echivalentă cu $b > c$, care contrazice $b \leq c$. În mod similar, se pot analiza și restul de patru strategii rămase, obținând, din nou, contradicții pentru fiecare caz analizat. Așadar, strategia descrisă în paragraful anterior este într-adevăr optimă.

Soluție pentru al doilea subtask

Să notăm cu S_i șirul nevid, indexat începând cu poziția 1, cu i elemente, care reprezintă sortarea în ordine non-descrescătoare a primelor i elemente din șirul p, pentru fiecare zi i: $1 \leq i \leq N$. Astfel: $S_{i,1} \leq S_{i,2} \leq \ldots \leq S_{i,i}$. Având în vedere că, la finalul seriei de tranzacții, dorim să avem un număr egal de acțiuni cumpărate și vândute, să notăm cu $k = \lfloor \frac{i}{2} \rfloor$ – alegem să cumpărăm și să vindem (în orice ordine) de câte k ori.

Aplicând în mod inductiv observația anterioară, este optim să cumpărăm la prețurile: $S_{i,1}, \ldots, S_{i,k}$ și să vindem la prețurile: $S_{i,i-k+1}, \ldots, S_{i,i}$. Să notăm sumele cu $l_{sum}[i] = S_{i,1} + \ldots + S_{i,k}$ și $u_{sum}[i] = S_{i,i-k+1} + \ldots + S_{i,i}$. Prin urmare, profitul maxim $best[i]$ care se poate obține după primele i zile, închizând pozițiile, este egal cu $(u_{sum}[i] - l_{sum}[i])$.

Fiecare element $best[i]$ se poate calcula în complexitatea $O(i \cdot \log(i))$, prin sortarea primelor i elemente. Astfel, complexitatea totală pentru calcularea șirului $best$ este de $O(N^2 \cdot \log(N))$.

Soluție pentru al patrulea subtask

Întrucât diferența $max(p_i) - min(p_i) \leq 25$, înseamnă că șirul p conține cel mult 26 de valori distincte. Prin urmare, calcularea șirului *best* poate fi efectuată mai rapid, într-o complexitate liniară de $O(n)$, folosind o tehnică similară, de exemplu, cu *Counting sort* – se menține un vector de frecvență.

De vreme ce fiecare pereche de tranzacții cumpărare-vânzare, sau vânzare-cumpărare, rezultă într-un profit de cel puțin 0 dolari, înseamnă că $best[1] \leq best[2] \leq \ldots \leq best[N]$. Astfel, în cadrul unei întrebări, determinarea indicelui minim j al unei zile, în cazul în care există, se poate face prin căutare binară pe șirul *best*.

Complexitatea totală este de $O(N + Q \cdot \log(N))$.

Soluție pentru al cincilea subtask

Similar cu soluția de la subtask-ul anterior, numărul maxim de elemente distincte din șirul p este de 10^5. Așadar, în loc să folosim un vector de frecvență, putem alege să calculăm șirul *best* folosind *Sqrt Decomposition*, în complexitatea de timp $O(N \cdot \sqrt{M}) \leq O(N \cdot \sqrt{N})$, după ce am atribuit, în prealabil, fiecărui element din șirul p o valoare din mulțimea $\{1, 2, \ldots, M\}$ – unde M reprezintă numărul de valori distincte din p.

Soluție completă

Ideea pentru soluția de 100 de puncte este să utilizăm o structură de date ce permite determinarea poziției minime y pentru care suma elementelor pe prefixul $[1...y]$ este mai mare sau egală decât o valoare fixată k, într-o complexitate de timp logaritmică, $O(\log(N))$; în cazul problemei noastre, pentru fiecare i, putem seta $k = \lfloor \frac{L_1+L_2+\ldots+L_i}{2} \rfloor$, iar $y \in \{1, 2, \ldots, M\}$. Astfel, putem utiliza un Arbore de intervale. Căutarea binară pentru găsirea lui y și interogările (de sumă) în cadrul arborelui de intervale pot fi efectuate simultan, vizitând $\Theta(\log(N))$ noduri din arbore, per interogare (cu parametrul k).

Deci, complexitatea totală de timp este de $O(N \cdot \log(N) + Q \cdot \log(N)) \leq O((N + Q) \cdot \log(N))$.

Având în vedere numărul relativ mare de elemente N din șirul p, din motive practice, recomandăm sortarea utilizând o metodă pseudo-liniară, cum ar fi Radix sort.

13.9 Cod-sursă pentru problema Nyse

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
using PII = pair<int, int>;
using ll = long long;
static constexpr int NMAX = ((int)(9e5) + 4), GROUPMAX = (260);
int n = (0);
PII p[NMAX], _temp[NMAX];
int limit[NMAX];
int m = (0), p_prime[NMAX];
ll best[NMAX];
int counter[NMAX];
bool FLAG = (false);
int delta = (0);
vector<PII> G[GROUPMAX];

static inline void read_normalize()
{
    ios_base ::sync_with_stdio(false);
    cin.tie(nullptr);
```

```
    freopen("nyse.in", "r", stdin);
    freopen("nyse.out", "w", stdout);
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i)
        scanf("%d", &p[i].first), _temp[i] = {(p[i].first), (i)};
    if (n > 1)
    {
        int count = 0;
        for (int i = 0; i < 4; ++i)
        {
            for (int x = 1; x <= n; ++x)
                G[((_temp[x].first >> (8 * i)) & 255)].push_back(_temp[x]);
            count = 0;
            for (int x = 0; x < 256; ++x)
                if (!G[x].empty())
                {
                    for (PII &v : G[x])
                        _temp[(++count)] = v;
                    G[x].clear();
                }
        }
    }
    p_prime[(m++)] = _temp[1].first,
    p[_temp[1].second].second = 1;
    for (int i = 2; i <= n; ++i)
    {
        if (_temp[i].first != _temp[(i - 1)].first)
            p_prime[(m++)] = _temp[i].first;

        p[_temp[i].second].second = m;
    }
    for (int i = 1; i <= n; ++i)
        scanf("%d", &limit[i]);
    return;
}

pair<int, ll> V[(NMAX << 2)];

inline void update(const int pos, const int x, const int y)
{
    int node = 1, a = 1, b = m,
        mid = 0;
    const ll to_add_second = (1LL * x * y);
    while (a != b)
    {
        V[node].first += (x), V[node].second += (to_add_second);
        mid = ((a + b) >> 1);
        if (pos <= mid)
            node = (node << 1), b = mid;
        else
            node = ((node << 1) + 1), a = (mid + 1);
    }
    V[node].first += (x), V[node].second += (to_add_second);
    return;
}

inline int find(int target)
{
    int node = 1, a = 1, b = m,
        mid = 0;
    int have_left = 0;
```

```
    while (a != b)
    {
        mid = ((a + b) >> 1);
        have_left = (V[(node << 1)].first);
        if (have_left >= target)
            node = (node << 1), b = mid;
        else
            target -= (have_left), node = ((node << 1) + 1), a = (mid + 1);
    }
    return a;
}

inline ll lower_tail(const int k)
{
    int left = 1, right = m, node = 1, ans = (-1);
    pair<int, ll> keep = {0, 0LL};
    while (left <= right)
    {
        if (left == right)
        {
            ans = left;
            keep.first += (V[node].first);
            keep.second += (V[node].second);
            break;
        }
        int mid = ((left + right) >> 1);
        if ((keep.first + V[(node << 1)].first) >= k)
            right = mid, node = (node << 1);
        else
            {left = (mid+1);
             keep.first += (V[(node<<1)].first);
             keep.second += (V[(node << 1)].second);
             node = ((node << 1) + 1);
            }
    }

    int take_from_ans = (k - (keep.first - counter[ans]));
    if (FLAG)
        if (take_from_ans < counter[ans])
            delta = p_prime[(ans - 1)];
    return (keep.second+(1LL*(take_from_ans-counter[ans])*p_prime[(ans-1)]));
}

int main()
{
    read_normalize();
    int sum_limits = 0;
    ll sum_total = 0,
       sum_lower_tail = 0,
       sum_upper_tail = 0;
    for (int i = 1; i <= n; ++i)
    {
        update(p[i].second, limit[i], p[i].first);
        sum_limits += (limit[i]), counter[p[i].second] += (limit[i]);
        sum_total += (1LL * p[i].first * limit[i]);
        if (i == 1)
            continue;
        if (sum_limits & 1)
            FLAG = 1;
        else
            FLAG = 0;
```

```
        delta = 0;
        sum_lower_tail = lower_tail((sum_limits >> 1));
        if ((FLAG) & (!delta))
            delta = p_prime[(find(((sum_limits) + 1) >> 1) - 1)];
        sum_upper_tail = ((sum_total - sum_lower_tail) - 1LL * delta);
        best[i] = (sum_upper_tail - sum_lower_tail);
    }

    int q = 0;
    scanf("%d", &q);
    ll x = 0;
    int left = 0, right = 0, ans = 0, mid = 0;
    for (int i = 1; i <= q; ++i)
    {
        scanf("%lld", &x);
        left = 1, right = n, ans = (-1);
        if (best[n] >= x)
        {
            while (left <= right)
            {
                mid = ((left + right) >> 1);
                if (best[mid] >= x)
                    ans = mid, right = (mid - 1);
                else
                    left = (mid + 1);
            }
        }
        printf("%d\n", ans);
    }
    return 0;
}
```