2023

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

CUPA SEPI

Organizate în parteneriat de

SEPI

Societatea pentru Excelență și Performanță în Informatică

și Ministerul Educației și Cercetării

Infobits Academy

2023
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
CUPA SEPI

Enunțuri, soluții, surse

Tehnoredactare

Emanuela Cerchez, Cătălin Frâncu

Coperta

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

ISBN

978-630-6559-24-4

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 2023, secțiunea Gimnaziu

Etapa județeană și etapa națională

Clasa a V-a

- Prof. Adrian Doru Pintea, Inspectoratul Școlar Județean Cluj - coordonator
- Stud. Ilie-Daniel Apostol, Facultatea de Matematică-Informatică, Universitatea București
- Stud. Ioana Gabor, Universitatea Babeș-Bolyai Cluj-Napoca
- Stud. Dumitru Ilie, Facultatea de Matematică-Informatică, Universitatea București
- Prof. Alina Gabriela Boca, Colegiul Național de Informatică „Tudor Vianu" București
- Prof. Eugenia-Cristiana Iordaiche, Liceul Teoretic „Grigore Moisil" Timișoara
- Prof. Victor-Claudiu Manz, Colegiul Național de Informatică „Tudor Vianu" București
- Prof. Marius Nicoli, Colegiul Național „Frații Buzești" Craiova
- Prof. Alina Pintescu, Colegiul Național „Gheorghe Șincai" Baia Mare
- Prof. Carmen Popescu, Colegiul Național „Gheorghe Lazăr" Sibiu
- Prof. Roxana Tîmplaru, Liceul Tehnologic „Ștefan Odobleja" Craiova

Clasa a VI-a

- Prof. Raluca Costineanu, Colegiul Național „Ștefan cel Mare" Suceava - coordonator
- Prof. Ana Maria Arișanu, Colegiul Național „Mircea cel Bătrân" Râmnicu Vâlcea
- Stud. Denis Andrei Banu, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza" Iași
- Stud. Vlad-Mihai Bogdan, Facultatea de Matematică-Informatică, Universitatea București
- Stud. Mihnea-Vicențiu Bucă, Facultatea de Matematică-Informatică, Universitatea București
- Prof. Alin Burța, Colegiul Național „B.P. Hașdeu" Buzău
- Prof. Delia Dabelea, Colegiul Național „Spiru Haret" Târgu Jiu
- Prof. Vlad Laurențiu Nicu, Liceul Teoretic „Mihail Kogălniceanu" Vaslui
- Prof. Daniel Popa, Liceul Teoretic „Aurel Vlaicu" Orăștie
- Prof. Liliana Șchiopu, Colegiul Național „Frații Buzești" Craiova
- Prof. Florentina Ungureanu, Colegiul Național de Informatică Piatra-Neamț

Clasa a VII-a

- Prof. Emanuela Cerchez, Colegiul Național „Emil Racoviță" Iași - coordonator
- Stud. Ștefan-Cosmin Dăscălescu, Universitatea București
- Instr. Cristian Frâncu, Clubul Nerdvana București
- Prof. Ionel-Vasile Piț-Rada, Colegiul Național „Traian" Drobeta Turnu-Severin
- Prof. Adrian Panaete, Colegiul Național „A.T. Laurian" Botoșani
- Prof. Filonela Balașa, Colegiul Național „Grigore Moisil" București
- Prof. Nistor-Eugen Moț, Școala „Dr. Luca" Brăila
- Prof. Flavius Boian, Colegiul Național „Spiru Haret" Târgu Jiu
- Stud. Ioan-Cristian Pop, Universitatea Politehnica București
- Prof. Dan Octavian Dumitrașcu, Colegiul Național „Dinicu Golescu" Câmpulung
- Stud. Theodor-Gabriel Tulba-Lecu, Universitatea Politehnica București

Clasa a VIII-a

- Prof. Daniela Lica, Centrul Județean de Excelență Prahova - coordonator
- Prof. Marinel Șerban, Colegiul Național „Emil Racoviță” Iași
- Prof. Stelian Ciurea, Colegiul Național „Samuel von Brukenthal”, Sibiu
- Prof. Claudiu-Cristian Gorea-Zamfir, Inspectoratul Școlar Județean Iași
- Prof. Isabela Coman, Colegiul Național de Informatică „Tudor Vianu” București
- Instr. Cătălin Frâncu, Clubul Nerdvana București
- Prof. Gheorghe Manolache, Colegiul Național de Informatică Piatra Neamț
- Prof. Cristina Anton, Colegiul Național „Gheorghe Munteanu Murgoci” Brăila
- Stud. Bogdan Ioan Popa, Facultatea de Matematică-Informatică, Universitatea București
- Stud. Sebastian Popa, Facultatea de Matematică-Informatică, Universitatea București
- Stud. Andrei Onuț, Universitatea Yale, S.U.A.

Barajul de selecție a lotului național de juniori

- Prof. Marius Nicoli, Colegiul Național „Frații Buzești” Craiova - coordonator
- Prof. Emanuela Cerchez, Colegiul Național „Emil Racoviță” Iași
- Prof. Adrian Panaete, Colegiul Național „August Treboniu Laurian”, Botoșani
- Prof. Marinel Șerban, Colegiul Național „Emil Racoviță”, Iași
- Stud. Bogdan Ioan Popa, Facultatea de Matematică și Informatică, Universitatea București
- Prof. Ionel-Vasile Piț-Rada, Colegiul Național „Traian”, Drobeta Turnu Severin
- Prof. Daniela Lica, Centrul Județean de Excelență Prahova
- Prof. Flavius Boian, Colegiul Național „Spiru Haret”, Târgu-Jiu
- Stud. Andrei Onuț, Universitatea Yale, S.U.A.
- Prof. Raluca Costineanu, Colegiul Național „Ștefan cel Mare” Suceava
- Stud. Ștefan-Cosmin Dăscălescu, Universitatea București
- Stud. Vlad-Mihai Bogdan, Facultatea de Matematică-Informatică, Universitatea București
- Prof. Daniel Popa, Liceul Teoretic „Aurel Vlaicu” Orăștie

Tabăra de pregătire a lotului național de juniori și de selecție a echipelor reprezentative ale României pentru competițiile internaționale Slatina, 7-14 mai 2023

- Prof. Adrian Panaete, Colegiul Național „A.T. Laurian” Botoșani - coordonator
- Prof. Marinel Șerban, Colegiul Național „Emil Racoviță” Iași
- Prof. Ciprian Chește, Liceul Tehnologic „Grigore C. Moisil” Buzău
- Prof. Gheorghe Eugen Nodea, Colegiul Național „Tudor Vladimirescu” Târgu Jiu
- Prof. Ionel-Vasile Piț-Rada, Colegiul Național „Traian” Drobeta-Turnu Severin
- Prof. Daniela Lica, Centrul Județean de Excelență Prahova
- Prof. Raluca Costineanu, Colegiul Național „Ștefan cel Mare” Suceava
- Prof. Emanuela Cerchez, Colegiul Național „Emil Racoviță” Iași
- Prof. Mihai Bunget, Colegiul Național „Tudor Vladimirescu” Târgu Jiu
- Prof. Flavius Boian, Colegiul Național „Spiru Haret” Târgu Jiu
- Prof. Zoltan Szabo, Inspectoratul Școlar Mureș
- Instr. Cătălin Frâncu, Clubul Nerdvana București
- Stud. Theodor Gabriel Tulba-Lecu, Universitatea Politehnică București
- Stud. Vlad Mihai Bogdan, Facultatea de Matematică-Informatică, Universitatea București
- Stud. Ioan Cristian Pop, Universitatea Politehnică București
- Stud. Andrei Onuț, Universitatea Yale, S.U.A.

CUPA SEPI
Câmpulung Muscel, 26-31 iulie 2023

- Prof. Marius Nicoli, Colegiul Național „Frații Buzești”, Syncro Soft Craiova - coordonator
- Prof. Daniela Lica, Centrul Județean de Excelență Prahova
- Prof. Victor Manz, Colegiul Național de Informatică „Tudor Vianu” București
- Stud. Ștefan-Cosmin Dăscălescu, Facultatea de Matematică-Informatică, Universitatea București
- Prof. Livia Măgureanu, Colegiul Național de Informatică „Tudor Vianu” București
- Prof. Ciprian Chește, Liceul Tehnologic „Grigore C. Moisil” Buzău
- Flaviu-Cristian Verde, Colegiul Național de Informatică „Tudor Vianu” București
- Stud. Andrei Onuț, Universitatea Yale, S.U.A.
- Stud. Bogdan Ioan Popa, Facultatea de Matematică-Informatică, Universitatea București
- Stud. Luca Perju Verzotti, Colegiul Național de Informatică „Tudor Vianu” București
- Prof. Adrian Panaete, Colegiul Național „A.T. Laurian” Botoșani
- Prof. Emanuela Cerchez, Colegiul Național „Emil Racoviță” Iași
- Prof. Marinel Șerban, Colegiul Național „Emil Racoviță” Iași
- Prof. Ionel-Vasile Piț-Rada, Colegiul Național „Traian” Drobeta-Turnu Severin
- Prof. Raluca Costineanu, Colegiul Național „Ștefan cel Mare” Suceava
- Stud. Vlad Mihai Bogdan, Facultatea de Matematică-Informatică, Universitatea București

Cuprins

I Olimpiada Județeană de Informatică 2023 **5**

1. OJI 2023, clasa a V-a **7**
1.1. Problema Aeriana . . . 7
1.2. Rezolvarea problemei Aeriana . . . 8
1.3. Cod-sursă pentru problema Aeriana . . . 9
1.4. Problema Castel . . . 11
1.5. Rezolvarea problemei Castel . . . 12
1.6. Cod-sursă pentru problema Castel . . . 13

2. OJI 2023, clasa a VI-a **14**
2.1. Problema Ciocolata . . . 14
2.2. Rezolvarea problemei Ciocolata . . . 15
2.3. Cod-sursă pentru problema Ciocolata . . . 16
2.4. Problema Unificare . . . 18
2.5. Rezolvarea problemei Unificare . . . 19
2.6. Cod-sursă pentru problema Unificare . . . 20

3. OJI 2023, clasa a VII-a **22**
3.1. Problema Palindrom . . . 22
3.2. Rezolvarea problemei Palindrom . . . 23
3.3. Cod-sursă pentru problema Palindrom . . . 24
3.4. Problema Primprim . . . 26
3.5. Rezolvarea problemei Primprim . . . 27
3.6. Cod-sursă pentru problema Primprim . . . 28

4. OJI 2023, clasa a VIII-a **30**
4.1. Problema Hibrid . . . 30
4.2. Rezolvarea problemei Hibrid . . . 32
4.3. Cod-sursă pentru problema Hibrid . . . 35
4.4. Problema Tema . . . 37
4.5. Rezolvarea problemei Tema . . . 38
4.6. Cod-sursă pentru problema Tema . . . 40

II Olimpiada Națională de Informatică 2023 **43**

5. ONI 2023, clasa a V-a **45**
5.1. Problema Cadouri . . . 45
5.2. Rezolvarea problemei Cadouri . . . 46
5.3. Cod-sursă pentru problema Cadouri . . . 47
5.4. Problema Legos . . . 49
5.5. Rezolvarea problemei Legos . . . 51

5.6. Cod-sursă pentru problema Legos . . . 51
5.7. Problema Patinaj . . . 52
5.8. Rezolvarea problemei Patinaj . . . 53
5.9. Cod-sursă pentru problema Patinaj . . . 54

6. ONI 2023, clasa a VI-a **56**
6.1. Problema Balon . . . 56
6.2. Rezolvarea problemei Balon . . . 58
6.3. Cod-sursă pentru problema Balon . . . 59
6.4. Problema Magictrick . . . 61
6.5. Rezolvarea problemei Magictrick . . . 62
6.6. Cod-sursă pentru problema Magictrick . . . 62
6.7. Problema Puteri3 . . . 64
6.8. Rezolvarea problemei Puteri3 . . . 65
6.9. Cod-sursă pentru problema Puteri3 . . . 65

7. ONI 2023, clasa a VII-a **67**
7.1. Problema Dominew . . . 67
7.2. Rezolvarea problemei Dominew . . . 68
7.3. Cod-sursă pentru problema Dominew . . . 69
7.4. Problema Pix . . . 70
7.5. Rezolvarea problemei Pix . . . 71
7.6. Cod-sursă pentru problema Pix . . . 72
7.7. Problema Secvmin . . . 73
7.8. Rezolvarea problemei Secvmin . . . 74
7.9. Cod-sursă pentru problema Secvmin . . . 74

8. ONI 2023, clasa a VIII-a **76**
8.1. Problema Castel . . . 76
8.2. Rezolvarea problemei Castel . . . 78
8.3. Cod-sursă pentru problema Castel . . . 79
8.4. Problema Kth . . . 81
8.5. Rezolvarea problemei Kth . . . 82
8.6. Cod-sursă pentru problema Kth . . . 84
8.7. Problema Struguri . . . 87
8.8. Rezolvarea problemei Struguri . . . 88
8.9. Cod-sursă pentru problema Struguri . . . 89

9. Baraj selecție lot juniori ONI 2023 **92**
9.1. Problema Extrapare . . . 92
9.2. Rezolvarea problemei Extrapare . . . 93
9.3. Cod-sursă pentru problema Extrapare . . . 94
9.4. Problema Fuziune . . . 95
9.5. Rezolvarea problemei Fuziune . . . 96
9.6. Cod-sursă pentru problema Fuziune . . . 97
9.7. Problema Udp . . . 101
9.8. Rezolvarea problemei Udp . . . 102
9.9. Cod-sursă pentru problema Udp . . . 104

III Tabăra de pregătire a lotului național de informatică juniori, Slatina, 7-14 mai 2023 **107**

10. Barajul 1 **109**
10.1. Problema Excursie . . . 109
10.2. Rezolvarea problemei Excursie . . . 111
10.3. Cod-sursă pentru problema Excursie . . . 112
10.4. Problema F1 . . . 114
10.5. Rezolvarea problemei F1 . . . 115
10.6. Cod-sursă pentru problema F1 . . . 117
10.7. Problema Pcp . . . 122
10.8. Rezolvarea problemei Pcp . . . 123
10.9. Cod-sursă pentru problema Pcp . . . 124

11. Barajul 2 **129**
11.1. Problema Ashima . . . 129
11.2. Rezolvarea problemei Ashima . . . 131
11.3. Cod-sursă pentru problema Ashima . . . 132
11.4. Problema Bossfight . . . 137
11.5. Rezolvarea problemei Bossfight . . . 138
11.6. Cod-sursă pentru problema Bossfight . . . 139
11.7. Problema Verevițe . . . 141
11.8. Rezolvarea problemei Verevițe . . . 142
11.9. Cod-sursă pentru problema Verevițe . . . 144

12. Barajul 3 **148**
12.1. Problema Countall . . . 148
12.2. Rezolvarea problemei Countall . . . 149
12.3. Cod-sursă pentru problema Countall . . . 150
12.4. Problema Culegeri . . . 153
12.5. Rezolvarea problemei Culegeri . . . 154
12.6. Cod-sursă pentru problema Culegeri . . . 156
12.7. Problema Teatru . . . 158
12.8. Rezolvarea problemei Teatru . . . 159
12.9. Cod-sursă pentru problema Teatru . . . 161

IV Cupa SEPI, Câmpulung Muscel, 26-31 iulie 2023 **165**

13. Cupa SEPI **167**
13.1. Problema All Numbers . . . 167
13.2. Rezolvarea problemei All Numbers . . . 168
13.3. Cod-sursă pentru problema All Numbers . . . 169
13.4. Problema Kpartit . . . 173
13.5. Rezolvarea problemei Kpartit . . . 174
13.6. Cod-sursă pentru problema Kpartit . . . 175
13.7. Problema Semnal . . . 179
13.8. Rezolvarea problemei Semnal . . . 180
13.9. Cod-sursă pentru problema Semnal . . . 182
13.10. Problema Circles . . . 184
13.11. Rezolvarea problemei Circles . . . 186
13.12. Cod-sursă pentru problema Circles . . . 189

Partea I

Olimpiada Națională de Informatică
- etapa județeană -

19 martie 2023

Capitolul 1

OJI 2023, clasa a V-a

1.1 Problema Aeriana

Propusă de: stud. Ioana Gabor, Universitatea Babeș-Bolyai Cluj-Napoca

O companie aeriană are planificate N zboruri. Fiecare zbor are asociate câte șase numere naturale cu următoarea semnificație:

- primul număr $A1$ identifică aeroportul de decolare,
- cel de-al doilea număr $A2$ identifică aeroportul de aterizare,
- următoarele patru numere naturale $H1$, $M1$, $H2$ și $M2$, reprezintă în ordine ora și minutul decolării, respectiv ora și minutul aterizării.

Aterizarea poate să fie în ziua curentă sau în ziua următoare. Un zbor poate să dureze maximum 23 de ore și 59 de minute. De exemplu, pentru $H1 = 10$, $M1 = 5$, $H2 = 15$, $M2 = 20$ aterizarea are loc în aceeași zi cu decolarea (zborul durează 5 ore și 15 minute), iar pentru $H1 = 23$, $M1 = 5$, $H2 = 1$, $M2 = 15$ aterizarea are loc în ziua următoare (zborul durează 2 ore și 10 minute).

Un virus informatic s-a infiltrat în sistemele de calcul ale companiei și a inversat momentul de decolare cu cel de aterizare al zborurilor pe care le consideră speciale. Un zbor este considerat special de către acest virus în cazul în care codul aeroportului de decolare, $A1$, este un număr prim, iar codul aeroportului de aterizare, $A2$, se divide cu suma cifrelor lui $A1$.

Cerințe

Cunoscându-se numărul de zboruri N și datele fiecăruia, **înainte de intervenția virusului**, să se determine:

1. Care este durata maximă a unui zbor, înainte de intervenția virusului.
2. Care este durata maximă a unui zbor, după intervenția virusului. Se iau în calcul atât duratele zborurilor inversate (speciale), cât și duratele zborurilor neinversate (nespeciale).

Date de intrare

Fișierul de intrare `aeriana.in` conține pe prima linie valoarea C (numărul cerinței, poate fi 1 sau 2), pe a doua linie valoarea N (numărul de zboruri). Pe fiecare dintre următoarele N linii sunt câte șase numere naturale $A1$, $A2$, $H1$, $M1$, $H2$, $M2$, în această ordine, despărțite prin câte un spațiu, cu semnificația din enunț.

Date de ieșire

Fișierul de ieșire `aeriana.out` va conține pe prima linie două numere naturale separate printr-un spațiu, reprezentând numărul de ore și respectiv numărul de minute ale zborului de durată maximă, în condițiile cerinței specificate.

Restricții

- $1 \leq N \leq 1\,000$;
- $0 \leq H1, H2 \leq 23$;
- $0 \leq M1, M2 \leq 59$;
- $0 \leq A1, A2 \leq 1\,000\,000\,000$;
- Un zbor va dura cel puțin un minut și cel mult 23 de ore și 59 de minute.

#	Puncte	Restricții
1	19	$C = 1$ și toate zborurile se desfășoară în aceeași zi
2	17	$C = 1$, $M1 = 0$, $M2 = 0$ pentru toate zborurile
3	17	$C = 1$, fără alte precizări
4	47	$C = 2$

Exemple

aeriana.in	aeriana.out	Explicații
1 3 47 55 0 0 23 59 1 437 23 43 10 34 11 457 10 43 10 23	23 59	$C = 1$, $N = 3$. Duratele acestor zboruri sunt, în ordine, 23 de ore și 59 de minute, 10 ore și 51 de minute, iar pentru ultimul zbor, 23 de ore și 40 de minute.
2 3 47 55 0 0 23 59 1 437 23 43 10 34 11 457 10 43 10 23	23 40	$C = 2$, $N = 3$. Pentru primul zbor $A1 = 47$ este număr prim, suma cifrelor sale este egală cu 11 și $A2 = 55$ se divide cu 11, deci primul zbor devine $23 : 59 - 00 : 00$ și are o durata de 0 ore și 1 minut. Al doilea zbor rămâne nemodificat, deoarece 1 nu e prim. Al treilea zbor rămâne nemodificat. Chiar dacă 11 este prim, 457 nu se divide cu 2 (suma cifrelor lui 11). Zborul de durata maximă, dupa interventia virusului, este cel de-al treilea.

1.2 Rezolvarea problemei Aeriana

Cerința 1

Se calculează toate duratele zborurilor, iar pe urmă trebuie afișat maximul dintre acestea. Pentru calculul duratei unui zbor, atât momentul decolării, cât și momentul aterizării „se convertesc în minute", adică se află, pentru fiecare moment, „distanța în minute" dintre momentul 00:00 al primei zile și momentul respectiv, presupunând că ambele momente sunt în ziua 1, cu formula $total\_minute = H \cdot 60 + M$. În cazul în care decolarea și aterizarea au loc în aceeași zi, durata

zborului (în minute) este dată de diferența dintre $total\_minute\_2$ și $total\_minute\_1$. În cazul în care decolarea și aterizarea au loc în zile diferite, diferența anterioara ar fi negativă. În acest caz, se adună „încă o zi”, adică $24 \cdot 60$ minute la $total\_minute\_2$, deoarece inițial am presupus că momentul aterizării este în ziua 1.

Cerința 2

Se verifică pentru fiecare zbor dacă este „zbor special”. În primul rând, primul cod (notat în enunț $A1$) trebuie să fie un număr prim (deci obligatoriu mai mare sau egal cu 2). Pentru a căuta mai rapid dacă are și alt divizor în afară de 1 și el însuși, ne folosim de faptul că pentru orice divizor d al lui $A1$, automat și $A1/d$ ar fi divizor al lui $A1$. Astfel căutăm divizori posibili d ai lui $A1$ pornind cu d de la 2 și mergând cât timp $d \leq A1/d$. Dacă primul cod este prim, se calculează suma cifrelor acestuia și se verifică dacă al doilea cod e divizibil cu suma calculată. În cazul în care un zbor este „special”, se interschimbă $h1$ cu $h2$ și $m1$ cu $m2$. Se calculează duratele zborurilor și maximul dintre acestea, precum în cazul cerinței 1.

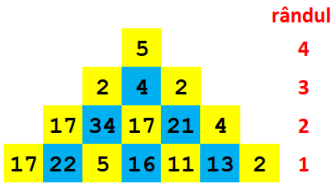
1.3 Cod-sursă pentru problema Aeriana

```
//Complexitate O(n*sqrt(a1)) timp, O(1) memorie
#include <fstream>
#define DURATA_ORA 60
#define NR_ORE_PE_ZI 24
using namespace std;
ifstream fi("aeriana.in");
ofstream fo("aeriana.out");

int main(){
int C, N, a1, a2, h1, m1, h2, m2;
int durata_maxima_in_minute = 0; // este garantat ca orice zbor va dura cel putin un minut
int durata_curenta_in_minute;
int momentul_1_in_minute;
int momentul_2_in_minute;
bool interschimbam = false;
bool prim;
int suma;

fi>>C>>N;

for (int i=1;i<=N;i++){
     fi>>a1>>a2>>h1>>m1>>h2>>m2;
     // pentru cerinta 1, a1 si a2 pot fi ignorate

     if (C == 2) {
        //pentru cerinta 2, avem urmatorul algoritm care verifica daca un zbor e "special"
        if (a1 < 2) prim = false;
           else
           {
            prim = true;
            for (int d=2; prim && d*d<=a1; d++){
                if (a1 % d == 0)
                    prim = false;
                }
            }
        if (prim) {
            // in cazul in care a1 e prim, aflam suma cifrelor acestuia
            suma = 0;
            while (a1 > 0){
```

```
                    suma += a1%10;
                    a1 /= 10;
                }
            if (a2 % suma== 0)
               interschimbam = true;
               else
               interschimbam = false;
            }
           else
           interschimbam = false;
        if (interschimbam) {
            swap(h1,h2); //efectuam interschimbarea si aflam maximul intocmai precum la cerinta 1
            swap(m1,m2);
            }
        }
    momentul_1_in_minute=h1*DURATA_ORA+m1; //momentele 1 si 2 se convertesc in numarul
    momentul_2_in_minute=h2*DURATA_ORA+m2; //de minute ce au trecut incepand cu 00:00, prima zi
    durata_curenta_in_minute = momentul_2_in_minute - momentul_1_in_minute;
    if (durata_curenta_in_minute < 0){ //suntem in cazul cand zborul aterizeaza a doua zi
       durata_curenta_in_minute += NR_ORE_PE_ZI *DURATA_ORA;
       //deci adunam 24*60 pentru conversia celui de-al doilea moment
       }
    durata_maxima_in_minute = max(durata_maxima_in_minute, durata_curenta_in_minute);
}
int durata_ore = durata_maxima_in_minute / DURATA_ORA;
int durata_minute = durata_maxima_in_minute % DURATA_ORA;
fo<<durata_ore<<' '<<durata_minute<<'\n';
fi.close(); fo.close();
return 0;
}
```

1.4 Problema Castel

Propusă de: prof. Eugenia-Cristiana Iordaiche, Liceul Teoretic „Grigore Moisil" Timișoara



Un joc dispune de N cuburi galbene și N cuburi albastre, de dimensiuni identice; pe fiecare cub galben este scris un număr natural nenul, de cel mult 9 cifre. Jocul urmărește construirea unui castel alcătuit din mai multe rânduri de cuburi, în care rândul de sus este format dintr-un singur cub, de culoare galbenă, iar fiecare dintre celelalte rânduri încep și se termină cu câte un cub de culoare galbenă. Oricare două cuburi vecine pe același rând au câte o latură comună și fiecare cub, cu excepția celor galbene de pe margine, are o latură comună cu un cub care aparține rândului de deasupra. Oricare două cuburi cu o latură comună au culori diferite. Rândurile de cuburi sunt numerotate de jos în sus, începând de la 1. Pentru construcția castelului se preiau cuburile galbene în ordinea în care acestea sunt date, iar cele albastre într-o ordine oarecare, și sunt plasate pe rânduri, de jos în sus, și pe fiecare rând de la stânga la dreapta, astfel: primul cub se plasează pe rândul de la bază (numerotat cu 1), apoi fiecare cub (galben sau albastru) se plasează fie în continuare, pe rândul curent, la dreapta, fie pe un rând nou, peste un cub al rândului curent. După plasarea cubului din vârful castelului, pe fiecare cub albastru se scrie un număr egal cu suma numerelor scrise pe cei doi vecini galbeni situați pe același rând, în stânga și în dreapta sa. Pentru a câștiga jocul, castelul obținut trebuie să aibă un număr maxim de rânduri, chiar dacă poate nu folosește toate cuburile date.

Cerințe

Cunoscând numerele scrise pe cele N cuburi galbene, în ordinea dată, scrieți un program care să determine:

1. numărul cuburilor galbene, dintre cele N date, pe care sunt scrise valori de o singură cifră;
2. rândul pe care se află cubul din vârful castelului și numărul scris pe acest cub;
3. numărul cuburilor albastre din care este alcătuit castelul și suma tuturor numerelor de pe acestea.

Date de intrare

Fișierul de intrare `castel.in` conține:

- pe prima linie două numere naturale C și N, în această ordine, despărțite printr-un spațiu, unde C reprezintă numărul cerinței și poate avea valorile 1, 2 sau 3, iar N are semnificația din enunț;
- pe a doua linie, N numere naturale despărțite prin câte un spațiu, reprezentând numerele scrise pe cuburile galbene, în ordinea în care sunt preluate.

Date de ieșire

Fișierul de ieșire `castel.out` conține pe prima linie:

- un singur număr natural pentru rezolvarea cerinței 1, reprezentând valoarea determinată conform acestei cerințe;
- două numere naturale despărțite printr-un spațiu, în cazul cerințelor 2 și 3. Pentru cerința 2, primul număr reprezintă rândul pe care se află cubul din vârful castelului iar cel de-al doilea număr reprezintă valoarea scrisă pe acest cub. Pentru cerința 3, prima valoare

reprezintă numărul de cuburi albastre care alcătuiesc castelul, iar a doua valoare reprezintă suma tuturor numerelor scrise pe aceste cuburi.

Restricții

- $3 \leq N \leq 5\,000$.

#	Puncte	Restricții
1	25	$C = 1$
2	30	$C = 2$
3	45	$C = 3$

Exemple

castel.in	castel.out	Explicații
1 12 17 5 11 2 17 17 4 2 2 5 34 88	6	C=1 și sunt 6 cuburi pe care sunt scrise numere de o singură cifră.
2 12 17 5 11 2 17 17 4 2 2 5 34 88	4 5	Exemplul corespunde imaginii din enunț și C=2. Cubul din vârful castelului este pe rândul 4 și pe el este scris 5.
3 12 17 5 11 2 17 17 4 2 2 5 34 88	6 110	Exemplul corespunde imaginii din enunț și C=3. Sunt 6 cuburi albastre în castel și suma numerelor scrise pe acestea este 110.

1.5 Rezolvarea problemei Castel

Cerința 1

Se contorizează toate valorile citite care au o singură cifră, ele sunt strict mai mari decât 0 și mai mici sau egale cu 9.

Cerința 2

Se calculează, pentru fiecare rând R al castelului, numărul cuburilor galbene aflate pe acesta. Știm că pe ultimul rând este un singur cub galben, pe penultimul rând sunt 2 cuburi galbene, pe antepenultimul 3 cuburi galbene, și, procedând în acest fel, pe primul rând al castelului (numerotat cu 1) sunt K cuburi galbene. Astfel, castelul construit, are în total $1+2+3+4+\cdots+K$ cuburi galbene, aranjate pe cele K rânduri. Determinăm cea mai mare valoare a lui K pentru care suma $1+2+3+4+\cdots+K$ nu depășește valoarea lui N. Rândul pe care se află cubul din vârful castelului este K iar numărul scris pe acesta este cel de pe poziția p în șirul de intrare, unde $p = 1+2+\cdots+K$. Alternativ p poate fi determinat și cu formula $K \cdot (K+1)/2$.

Cerința 3

Observăm că pe fiecare rând al castelului numărul cuburilor albastre este cu 1 mai mic decât numărul cuburilor galbene. Păstrând aceleași notații ca la cerința anterioară, numărul cuburilor albastre din castel este egal cu $1+2+3+\cdots+(K-1)$. Pentru fiecare cub albastru, calculăm numărul scris pe acesta ca fiind suma a două valori preluate succesiv din fișierul de intrare. Acestea

reprezintă numerele scrise pe cuburile galbene situate pe același rând, în stânga și dreapta fiecărui cub albastru. Calculăm suma tuturor valorilor scrise pe cuburile albastre din castel.

1.6 Cod-sursă pentru problema Castel

```
#include <fstream>
using namespace std;
ifstream fin("castel.in");
ofstream fout("castel.out");
int C, N, H, x, maxi, Ha, y, k;
long long int suma=0;
int main()
{
    fin>>C>>N;
    H=0;
    while(H*(H+1)/2<=N)
        H++;
    H=H-1;
    if(C==1)
    {
        for(int i=1; i<=N; i++)
        {
            fin>>x;
            if(x>=1&& x<=9)
            k++;
        }
        fout<<k<<"\n";
    }
    else if (C==2)
    {
        for(int i=1; i<=H*(H+1)/2; i++)
            fin>>x;
        fout<<H<<" "<<x<<"\n";
    }
    else if(C==3)
    {
        Ha=H;
        for(int i=1; i<=(Ha+1); i++)
        {
            fin>>x;
            for(int j=2; j<=H; j++)
            {
                fin>>y;
                suma=suma+x+y;
                x=y;
            }
            H--;
        }
        fout<<Ha*(Ha-1)/2<<" "<<suma<<"\n";
    }
    return 0;
}
```

Capitolul 2

OJI 2023, clasa a VI-a

2.1 Problema Ciocolata

Propusă de: stud. Vlad-Mihai Bogdan, Facultatea de Matematică-Informatică, Universitatea București și prof. Daniel Popa, Liceul Teoretic „Aurel Vlaicu" Orăștie

Irina și Mihaela sunt surori. Într-o zi, mama lor le aduce N tablete de ciocolată, numerotate de la 1 la N, pe care le așază, în această ordine, pe o poliță a unui raft. Pentru fiecare tabletă se cunoaște gramajul (numărul de grame pe care le cântărește). **Cantitatea totală** de ciocolată consumată de o fată este egală cu suma gramajelor tuturor tabletelor consumate de ea. Pentru a consuma ciocolată, fetele trebuie să respecte următoarele reguli:

- cantitatea totală de ciocolată consumată de Irina trebuie să fie mai mare sau egală cu cantitatea totală de ciocolată consumată de sora sa;
- diferența dintre cantitatea totală de ciocolată consumată de Irina și cantitatea totală de ciocolată consumată de Mihaela trebuie să fie cât mai mică;
- fiecare fată trebuie să consume cel puțin o tabletă de ciocolată;
- fiecare fată consumă tablete de ciocolată de pe raft: Irina începe de la cea numerotată cu 1 și continuă, în ordine, de la stânga la dreapta, iar Mihaela începe cu cea numerotată cu N și continuă, în ordine, de la dreapta la stânga;
- fiecare fată poate întrerupe oricând consumul tabletelor de ciocolată, iar cele rămase fie sunt abandonate pe raft, fie sunt consumate de fata cealaltă, dacă ajunge la ele;
- fiecare tabletă de ciocolată fie este consumată complet de una dintre fete, fie rămâne pe raft, dar fetele NU pot sări peste nicio tabletă de ciocolată.

Cerințe

Determinați și afișați:

1. cel mai des întâlnit gramaj în șirul de tablete așezate inițial pe poliță, iar dacă sunt mai multe gramaje care apar de un număr maxim de ori, se alege cel mai mic dintre acestea;
2. diferența minimă dintre cantitatea totală de ciocolată consumată de Irina și cantitatea totală de ciocolată consumată de Mihaela.

Date de intrare

Pe prima linie din fișierul `ciocolata.in` se găsește numărul C, reprezentând cerința ce trebuie rezolvată (1 sau 2), urmat de numărul N, cu semnificația din enunț. Pe a doua linie se află N

numere naturale, reprezentând gramajele celor N tablete de ciocolată, în ordinea numerotării lor. Numerele aflate pe aceeași linie a fișierului sunt separate prin câte un spațiu.

Date de ieșire

Pe prima linie a fișierului de ieșire `ciocolata.out` se va afla un singur număr reprezentând răspunsul cerința C.

Restricții

- $C \in \{1, 2\}$
- $1 \leq N \leq 100\,000$
- gramajul fiecărei tablete este un număr natural nenul mai mic sau egal cu 10 000
- se garantează că există întotdeauna soluție.

#	Puncte	Restricții
1	30	$C = 1$
2	5	$C = 2$ și $N = 2$
3	10	$C = 2$ și $1 \leq N \leq 100$
4	25	$C = 2$ și $1 \leq N \leq 1\,000$
5	30	$C = 2$, fără restricții suplimentare

Exemple

ciocolata.in	ciocolata.out	Explicații
1 6 1 4 3 3 5 4	3	$C = 1$, $N = 6$, iar cele mai frecvente gramaje de ciocolată dintre cele 6 sunt 3 și 4, fiecare apărând de câte două ori. Se va alege gramajul 3.
2 5 14 4 25 2 9	3	$C = 2$, $N = 5$, iar Irina a consumat prima tabletă de ciocolată (în cantitate totală de 14 grame), iar Mihaela ultimele două tablete (în cantitate totală de 11 grame), deci diferența de cantitate este de 3 grame.
2 11 3 7 3 12 4 9 4 2 6 5 17	1	$C = 2$, $N = 11$, Irina va consuma primele cinci tablete de ciocolată (în cantitate totală de 29 grame), iar Mihaela ultimele trei tablete (în cantitate totală de 28 grame).

2.2 Rezolvarea problemei Ciocolata

Cerința 1

Pentru rezolvarea primei cerințe, este suficient să introducem toate numerele într-un vector de frecvența, iar mai apoi să îl parcurgem și să determinăm primul indice la care se găsește valoarea maximă.

Cerința 2

Pentru rezolvarea acestei cerințe putem folosi următoarea abordare: vom menține patru variabile: i - indicele până la care va mânca Irina ciocolată, sum_i - cantitatea totală de ciocolată de la poziția 1 până la poziția i, j - indicele până la care va mânca Mihaela ciocolată, sum_j - cantitatea totală de ciocolată de la poziția j până la poziția N. În momentul în care modificăm valorile i și j, va trebui să actualizăm și variabilele sum_i și sum_j. Așadar, pentru fiecare poziție i (și pentru fiecare valoare sum_i), va trebui să determinăm cea mai din stânga poziție j pentru care $sum_j \leq sum_i$. Observația cheie este că în momentul în care trecem de la poziția i la poziția $i + 1$, valoarea j corespunzătoare indicelui i este mai mare sau egală decât valoarea corespunzătoare indicelui $i+1$. Observăm, așadar, că în momentul în care indicele i crește, indicele j scade sau rămâne pe loc. Prin urmare, folosind un astfel de algoritm, se obține complexitatea $O(N)$.

Cerința 2 - soluție alternativă

Pentru fiecare poziție fixată i (până la care va mânca Irina ciocolată), va trebui să determinăm poziția cea mai din stânga j, astfel încât cantitatea totală de ciocolată de la poziția 1 până la poziția i să fie cel mult la fel de mare că și cantitatea totală de ciocolată de la poziția j la poziția N. Putem să căutăm binar pentru fiecare poziție i, indicele j corespunzător, iar pentru determinarea sumei numerelor de la poziția j până la poziția N, vom folosi sume parțiale.

Complexitatea unei astfel de soluții este $O(N \cdot \log_2 N)$.

2.3 Cod-sursă pentru problema Ciocolata

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

ifstream in("ciocolata.in");
ofstream out("ciocolata.out");

const int MAXN = (int)1e5;
const int MAXF = (int)1e4;

int N, T;
int V[MAXN], sum[MAXN], fr[1 + MAXF];

int main() {
    in >> T >> N;
    for (int i = 0; i < N; i++) {
        in >> V[i];
    }
    if (T == 2) {
        int i = 0, j = N - 1, answer = INT_MAX, sumLeft = V[0], sumRight = V[N - 1];
        while (i < j) {
            if (sumLeft >= sumRight) {
                answer = min(answer, sumLeft - sumRight);
                j--;
                sumRight += V[j];
            } else {
                i++;
                sumLeft += V[i];
            }
        }
        out << answer << "\n";
    }
    else {
```

```
        int answer = 0;
        for (int i = 0; i < N; i++) {
            fr[V[i]]++;
            if (fr[V[i]]>fr[answer] || (fr[V[i]]==fr[answer] && V[i]<answer)) {
                answer = V[i];
            }
        }
        out << answer << "\n";
    }
    return 0;
}
```

2.4 Problema Unificare

Propusă de: prof. Raluca Costineanu, Colegiul Național „Ștefan cel Mare" Suceava

Prin operația de **unificare** a două numere naturale a și b înțelegem obținerea celui mai mare număr care se poate forma din cifrele distincte din scrierea numărului a și cifrele distincte din scrierea numărului b. De exemplu, unificând $a = 727952$ cu $b = 92868$ vom obține numărul 99876522, deoarece din a vom utiliza cifrele 2, 5, 7, 9, iar din b cifrele 2, 6, 8, 9. Cel mai mare număr pe care îl putem forma cu aceste cifre este 99876522.

Operația de unificare poate fi aplicată și pentru k numere, respectând aceeași regulă: pentru fiecare număr din cele k identificăm cifrele distincte care apar în scrierea lui, apoi determinăm cel mai mare număr care se poate forma utilizând toate aceste cifre. De exemplu, unificând numerele 112, 223 și 12334 vom obține 43322211.

Se dau două numere naturale, n și k, și un șir de n numere naturale $a_1, a_2, \ldots, a_n$.

Cerințe

Determinați și afișați:

1. cel mai mare număr de exact k cifre din șirul dat;
2. cel mai mare număr care poate fi obținut prin unificarea a două valori aflate pe poziții alăturate în șirul dat;
3. cel mai mare număr care se poate obține prin unificarea a k valori aflate pe poziții consecutive în șirul dat.

Date de intrare

Fișierul de intrare `unificare.in` conține pe prima linie un număr natural C, reprezentând cerința ce trebuie rezolvată (1, 2 sau 3), pe a doua linie n și k, cu semnificația din enunț, iar pe a treia linie cei n termeni ai șirului precizat, în ordinea din șir. Numerele aflate pe aceeași linie a fișierului sunt separate prin câte un spațiu.

Date de ieșire

În fișierul de ieșire `unificare.out`:

- dacă $C = 1$, se va afișa pe prima linie cel mai mare număr de k cifre din șirul dat;
- dacă $C = 2$, se va afișa pe prima linie cel mai mare număr obținut prin unificarea a două numere alăturate în șir;
- dacă $C = 3$, se va afișa pe prima linie valoarea maximă obținută prin unificarea a k valori aflate pe poziții consecutive.

Restricții

- $C \in \{1, 2, 3\}$; $1 \leq n \leq 100\,000$; $1 \leq k \leq n/2$;
- $0 \leq a_i \leq 100\,000\,000$, pentru oricare $1 \leq i \leq n$

#	Puncte	Restricții
1	20	$C = 1$ și $k \leq 8$
2	5	$C = 2$ și $n = 2$
3	10	$C = 2$ și $0 \leq a_i \leq 9$, pentru oricare $1 \leq i \leq n$
4	35	$C = 2$, fără restricții suplimentare
5	15	$C = 3$ și $k \leq 8$
6	15	$C = 3$ și $k \leq n/2$

Exemple

unificare.in	unificare.out	Explicații
1 5 3 112 223 12334 561 289	561	$C = 1$, $n = 5$ și $k = 3$. În șir sunt 4 numere care au exact 3 cifre: 112, 223, 561 și 289, cel mai mare dintre ele fiind 561.
2 5 3 112 223 12334 561 289	6543211	$C = 2$, $n = 5$ și $k = 3$, nu utilizăm valoarea lui k și unificând a_3 cu a_4 vom obține cea mai mare valoare: 6543211
3 5 3 112 223 12334 561 289	9865432211	$C = 3$, $n = 5$ și $k = 3$. Cea mai mare valoare care se poate obține este 9865432211 și o obținem unificând a_3 cu a_4 și cu a_5.

2.5 Rezolvarea problemei Unificare

Cerința 1

Pentru fiecare număr din cele n citite determinăm numărul de cifre, dacă acesta este egal cu k atunci comparăm numărul cu maximul determinat până atunci, pentru a identifica cel mai mare număr.

Cerința 2

Pentru fiecare pereche de valori aflate pe poziții alăturate în șir determinăm valoarea obținută prin unificarea celor două numere în vederea determinării maximului posibil.

O variantă pentru a determina valoarea unificată pentru două numere este să considerăm toate cifrele în ordine descrescătoare și să adăugăm la numărul pe care îl formăm acele cifre care apar în scrierea fiecărui număr.

Cerința 3

Trebuie să unificăm câte k valori aflate pe poziții consecutive în șirul dat, iar valoarea obținută ar fi prea mare pentru a putea fi reținută într-o variabilă simplă de memorie. Putem utiliza un vector de numărare în care să reținem, pentru fiecare cifră, în câte dintre numerele dintr-o secvență de k valori aflate pe poziții consecutive în șir, apare cifra respectivă. Pentru a compara numărul curent cu maximul pe care îl determinăm va trebui mai întâi să verificăm dacă am obținut un număr cu mai multe cifre, caz în care ar fi mai mare, sau dacă avem același număr de cifre, verificăm dacă

avem o cifră mai mare cu un număr mai mare de apariții. Maximul obținut îl vom afișa, cifră cu cifră, în fișierul de ieșire.

2.6 Cod-sursă pentru problema Unificare

```
#include <fstream>
using namespace std;
ifstream f("unificare.in");
ofstream g("unificare.out");
const int N = 100005;
int main()
{   int C, n, k, i, x;
    f >> C >> n >> k;
    if (C == 1)
    {
        int mx = 0;
        for (i = 1; i <= n; i++)
        {
            f >> x;
            int cx = x, nrcif = 0;
            do
                nrcif++, cx /= 10;
            while (cx);
            if (x > mx && nrcif == k)
                mx = x;
        }
        g << mx << '\n';
    }
    else if (C == 2)
    {
        int y, cx, cy;
        long long int nr, mx = 0;
        f >> x;
        for (int i = 2; i <= n; i++)
        {
            f >> y;
            nr = 0;
            for (int c = 9; c >= 0; c--)
            {
                bool okX = 0, okY = 0;
                cx = x; cy = y;
                do
                {
                    if (cx % 10 == c) { okX = 1; break; }
                    cx /= 10;
                } while (cx);
                do
                {
                    if (cy % 10 == c) { okY = 1; break; }
                    cy /= 10;
                } while (cy);
                if (okX) nr = nr * 10 + c;
                if (okY) nr = nr * 10 + c;
            }
            if (nr > mx) mx = nr;
            x = y;
        }
        g << mx << '\n';
    }
    else if (C == 3)
```

```
    {
        int v[N], frMx[10] = {}, frK[10] = {};
        for (i = 1; i <= n; i++) f >> v[i];
        for (i = 1; i <= k; i++)
        {
            int fr[10] = {}, x = v[i];
            do
                fr[x % 10] = 1, x /= 10;
            while (x);
            for (int c = 0; c <= 9; c++)
                frK[c] += fr[c];
        }
        for (int c = 0; c <= 9; c++) frMx[c] = frK[c];
        for (i = k + 1; i <= n; i++)
        {
            int fr[10] = {}, x = v[i];
            do
                fr[x % 10] = 1, x /= 10;
            while (x);
            for (int c = 0; c <= 9; c++)
                frK[c] += fr[c];
            int fri[10] = {}, y = v[i - k];
            do
                fri[y % 10] = 1, y /= 10;
            while (y);
            for (int c = 0; c <= 9; c++)
                frK[c] -= fri[c];
            int sMX = 0, sK = 0, cmp = 0;
            for (int c = 0; c <= 9; c++)
                sMX += frMx[c], sK += frK[c];
            if (sK > sMX) cmp = 1;
            else if (sK < sMX) cmp = -1;
            else
            {
                for (int c = 9; c >= 0; c--)
                    if (frK[c] > frMx[c])
                    {
                        cmp = 1; break;
                    }
                    else if (frK[c] < frMx[c])
                    {
                        cmp = -1; break;
                    }
            }
            if (cmp > 0)
                for (int c = 0; c <= 9; c++)
                    frMx[c] = frK[c];
        }
        for (int c = 9; c >= 0; c--)
            while (frMx[c])
                g << c, frMx[c]--;
        g<<'\n';
    }
    return 0;
}
```

Capitolul 3

OJI 2023, clasa a VII-a

3.1 Problema Palindrom

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

Un număr se numește *palindrom* dacă citit de la stânga la dreapta este identic cu numărul citit de la dreapta la stânga. De exemplu, numerele 131 și 15677651 sunt palindromuri. Un număr care nu este palindrom poate fi transformat în palindrom adăugând la dreapta sa una sau mai multe cifre.

Cerințe

Dat fiind un șir de n numere naturale, scrieți un program care să rezolve următoarele două cerințe:

1. să se determine numărul minim total de cifre care trebuie să fie adăugate, astfel încât fiecare valoare din șir să fie palindrom;
2. considerând că putem adăuga cel mult S cifre, să se determine numărul maxim de termeni palindrom aflați pe poziții consecutive în șirul obținut.

Date de intrare

Fișierul de intrare `palindrom.in` conține pe prima linie numărul C, reprezentând cerința care trebuie să fie rezolvată (1 sau 2). Pe cea de a doua linie se află un număr natural n, reprezentând numărul de valori din șir. Pe următoarele n linii se află cele n numere din șir, câte un număr pe o linie. Dacă $C = 2$, pe ultima linie a fișierului de intrare se va afla numărul natural S reprezentând numărul maxim de cifre ce pot fi adăugate.

Date de ieșire

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

Restricții

- $1 \leq n \leq 50\,000$; $0 \leq S \leq 500\,000$
- Numerele din șir au cel mult 50 de cifre.

#	Puncte	Restricții
1	15	$C = 1$ și $n = 1$.
2	10	$C = 2$, $S = 0$, $1 < n \leq 100$ și numerele din șir au cel mult 18 cifre.
3	14	$C = 1$, $1 < n \leq 1\,000$ și numerele din șir au cel mult 18 cifre.
4	15	$C = 2$, $S > 0$, $1 < n \leq 1\,000$ și numerele din șir au cel mult 18 cifre.
5	16	$C = 2$, $1\,000 < n \leq 50\,000$ și numerele din șir au cel mult 18 cifre.
6	13	$C = 1$, $1\,000 < n \leq 50\,000$ și numerele din șir au între 19 și 50 de cifre.
7	17	$C = 2$, $1\,000 < n \leq 50\,000$ și numerele din șir au între 19 și 50 de cifre.

Exemple

palindrom.in	palindrom.out	Explicații
1 5 12232 131 12345 0 7717	7	$C = 1$, $n = 5$. Pentru a transforma 12232 în palindrom trebuie să adăugăm minimum două cifre (1223221), pentru 12345 trebuie să adăugăm minimum 4 cifre (123454321), pentru 7717 trebuie să adăugăm minimum o cifră (77177), iar numerele 131 și 0 sunt deja palindromuri. În total 2+4+1=7.
2 7 12232 131 12345 0 7717 1244 215809 4	3	$C = 2$, $n = 7$, $S = 4$, deci se pot adăuga maximum 4 cifre. Putem adăuga cele 4 cifre numărului 12345 și obținem o secvență de lungime 3 formată numai din palindromuri (131 123454321 0). O altă variantă este de a adăuga o cifră la 7717 și două cifre la 1244 și obținem tot o secvență de lungime 3 formată numai din palindromuri (0 77177 124421). Pentru orice altă variantă, secvența de palindromuri obținută are mai puțini termeni.

3.2 Rezolvarea problemei Palindrom

Cerința 1

Vom citi succesiv numerele și vom determina pentru fiecare număr citit numărul minim de cifre care trebuie să fie adăugate pentru a transforma numărul respectiv în palindrom. Pentru numere mari (de maximum 50 de cifre), vom citi fiecare număr caracter cu caracter, până la întâlnirea marcajului de sfârșit de linie și vom reține cifrele numărului într-un vector. Pentru punctaj parțial (numere de maximum 18 cifre) se va citi numărul într-o variabilă de tip *longlongint*, apoi se vor extrage cifrele numărului și se vor plasa într-un vector. Pentru a determina numărul minim de cifre care trebuie adăugate la finalul numărului pentru a transforma acest număr în palindrom putem determina lungimea celui mai lung sufix al numărului care are proprietatea de a fi palindrom. Să notăm această lungime cu lgs, iar lungimea numărului cu lg. Numărul minim de cifre care trebuie să fie adăugate este $nr = lg - lgs$ (se adaugă la final primele nr cifre ale numărului în ordine inversată). Desigur, pentru punctaj parțial este posibilă și o abordare „prin încercări”:

- dacă numărul este deja palindrom, $nr = 0$;
- dacă numărul nu este palindrom, adăugăm la sfârșitul lui o cifră (prima cifră a numărului) și verificăm dacă se obține un palindrom (în acest caz nr va fi 1);
- apoi încercăm cu două cifre, trei cifre, ș.a.m.d.
- În cel mai defavorabil caz adăugăm $nr = lg - 1$ cifre (primele $lg - 1$ cifre în ordine inversă).

Nu este necesar să reținem toate numerele citite, vom reține într-un vector nr cu n elemente numărul minim de cifre care trebuie să fie adăugate pentru a transforma fiecare număr din șir în palindrom. Rezultatul la cerința 1 este suma valorilor memorate în vectorul nr.

Cerința 2

Trebuie să determinăm cea mai lungă subsecvență a vectorului nr, construit la cerința anterioară, care are suma elementelor mai mică sau egală cu S. Pentru aceasta vom parcurge vectorul nr cât timp suma elementelor din secvența curentă (să o notăm sum) este $\leq S$. Când am ajuns la o poziție i pentru care $sum + nr[i] > S$, elementul curent nu mai poate fi „înghițit" în soluție. Prin urmare:

- comparăm lungimea secvenței curente cu lungimea maximă și o reținem dacă este mai mare;
- eliminăm elementele de la începutul secvenței curente, actualizând corespunzător sum, până când este posibil să „înghițim" valoarea $nr[i]$ în soluție ($sum + nr[i] \leq S$).

Când parcurgerea vectorului nr s-a încheiat, trebuie să comparăm și lungimea ultimei secvențe cu lungimea maximă.
Pentru punctaje parțiale sunt posibile și abordări de complexitate $O(n^2)$ sau $O(n^3)$.

3.3 Cod-sursă pentru problema Palindrom

```
#include <fstream>
#define NMAX 50002
#define LGMAX 102
using namespace std;
ifstream fin("palindrom.in");
ofstream fout("palindrom.out");
int n, c, S, lg, lgmax;
long long int sum;
int nr[NMAX];
int v[LGMAX];
int main()
{int i, j, st, dr, inceput;
 char ch;
 fin>>c>>n; fin.get(ch);
 for (i=1; i<=n; i++)
      {//citeste un numar
       ch=0; lg=0;
       while (1)
             {fin.get(ch);
              if (ch=='\n') break;
              v[++lg]=ch-'0';
             }
       //determin numarul minim de cifre care tb adaugate la finalul lui v pt a deveni palindrom
       for (j=1; j<lg; j++)
           {
            for (st=j, dr=lg; st<dr && v[st]==v[dr]; st++, dr--);
            if (st>=dr) break;
           }
       nr[i]=j-1;
```

```
      }
   if (c==1)
      { sum=0;
        for (i=1; i<=n; i++) sum+=nr[i];
        fout<<sum<<'\n';
      }
      else
      {
       fin>>S;
       //determinam in nr o secventa de lungime maxima cu suma elementelor <=S
       sum=0; inceput=1;
       for (i=1; i<=n; i++)
           if (sum+nr[i]<=S)
               sum+=nr[i];
               else
               {
                if (i-inceput>lgmax) lgmax=i-inceput;
                while (inceput<=i && sum+nr[i]>S)
                       sum-=nr[inceput++];
                if (sum+nr[i]<=S) sum+=nr[i];
                }
        if (i-inceput>lgmax) lgmax=i-inceput;
        fout<<lgmax<<'\n';
      }
  return 0;
}
```

3.4 Problema Primprim

Propusă de: stud. Ștefan-Cosmin Dăscălescu, Universitatea București

Pentru un număr natural a definim *costul* ca fiind valoarea absolută (modulul) diferenței dintre a și numărul prim cel mai apropiat de a. Asupra unui șir de n numere naturale, situate pe poziții numerotate de la 1 la n, se aplică, în ordine, o succesiune de q operații. O operație constă dintr-o înlocuire și o afișare și este descrisă sub forma i x p, cu semnificația:

- mai întâi înlocuim cu x elementul din șir de pe poziția i;
- apoi afișăm suma minimă totală a costurilor unor elemente convenabil selectate de pe p poziții distincte din șir.

Cerințe

Cunoscând n și cele n elemente ale șirului, scrieți un program care să determine:

1. suma costurilor tuturor elementelor din șirul dat;
2. rezultatele afișate în urma aplicării fiecăreia dintre cele q operații, date în forma precizată.

Date de intrare

Fișierul de intrare `primprim.in` va conține pe prima linie un număr natural C, reprezentând cerința care trebuie să fie rezolvată (1 sau 2), pe a doua linie numărul natural n, cu semnificația din enunț, iar pe a treia linie cele n elemente din șir, în ordinea din șir. Dacă $C = 2$, pe a patra linie se află numărul natural q, reprezentând numărul de operații, iar pe următoarele q linii se află cele q operații, câte o operație pe linie, în forma descrisă în enunț. Numerele scrise pe aceeași linie sunt separate prin câte un spațiu.

Date de ieșire

Dacă $C = 1$, fișierul de ieșire `primprim.out` va conține o singură linie pe care va fi afișată suma costurilor tuturor elementelor din șir. Dacă $C = 2$, fișierul de ieșire *primprim.out* va conține q linii, pe linia i fiind scris rezultatul afișat după executarea celei de a i-a operații din fișierul de intrare.

Restricții

- $1 \leq q \leq 2 \cdot 10^5$
- $1 \leq i, p \leq n \leq 10^6$; $1 \leq x \leq 10^6$
- Elementele șirului sunt numere naturale nenule $\leq 10^6$.

#	Puncte	Restricții
1	20	$C = 1$, $n = 1$
2	22	$C = 1$, $1 < n \leq 1\,000$
3	28	$C = 2$, $n \leq 1\,000$, $q \leq 10$
4	30	$C = 2$, nu există restricții suplimentare

Exemple

primprim.in	primprim.out	Explicații
1 5 8 1 3 5 9	4	$C = 1$, $n = 5$, iar șirul este $8, 1, 3, 5, 9$. Costurile elementelor sunt, în ordine, 1, 1, 0, 0, 2, deci suma este 4.
2 5 8 1 3 5 9 3 2 6 4 3 5 2 5 12 5	2 0 3	$C = 2$, $n = 5$, iar șirul inițial este $8, 1, 3, 5, 9$. Se aplică șirului 3 operații. După prima operație, pentru care $i = 2$, $x = 6$ și $p = 4$, șirul devine $8, 6, 3, 5, 9$. Suma minimă totală se obține dacă selectăm valorile de pe pozițiile 1, 2, 3 și 4, costurile fiind 1+1+0+0=2. După a II-a operație, pentru care $i = 3$, $x = 5$ și $p = 2$, șirul devine $8, 6, 5, 5, 9$. Selectăm valorile de pe pozițiile 3 și 4 (acestea având costul 0). După a III-a operație, pentru care $i = 5$, $x = 12$ și $p = 5$, șirul devine $8, 6, 5, 5, 12$. Selectăm toate valorile, deci suma este $1 + 1 + 0 + 0 + 1 = 3$.

3.5 Rezolvarea problemei Primprim

Pentru ambele cerințe va fi necesar să determinăm cât mai rapid pentru un număr dat distanța față de cel mai apropiat număr prim.
O primă abordare ar fi ca pentru fiecare număr să verificăm mai întâi dacă este prim (în acest caz, costul ar fi 0), iar în caz contrar ne deplasăm la stânga și la dreapta sa până când identificăm un număr prim, calculând apoi costul folosind formula dată din enunț. Totuși, o asemenea abordare ar avea complexitatea $O(x \cdot \sqrt{valmax})$, unde x reprezintă distanța maximă față de un număr prim. Deoarece x este cel mult 57, o asemenea abordare nu obține punctaj maxim.
Pentru a optimiza această abordare, vom precalcula costurile pentru toate numerele de la 1 la 10^6. Pentru aceasta, vom utiliza ciurul lui Eratostene, pentru a genera numerele prime $\leq 10^6$, urmând ca mai apoi costul să fie calculat în $O(1)$ pentru fiecare număr. Complexitatea precalculării este $O(n \log \log n)$.

Cerința 1.

Vom citi succesiv numerele și vom afla succesiv costul pentru fiecare număr de la intrare, reținând într-o variabilă suma costurilor. În funcție de abordarea folosită pentru calcularea costurilor, se pot obține diverse punctaje parțiale, dar punctajul maxim pe cerință se poate obține doar folosind metoda bazată pe ciurul lui Eratostene, abordare explicată mai sus.

Cerința 2.

Pentru această cerință vom citi succesiv operațiile și le vom executa.
Pentru a obține un cost total minim trebuie să adunăm cele mai mici p costuri. O abordare eficientă se bazează pe observația că pentru orice număr costul este cel mult 57, fapt ce ne permite să utilizăm un vector de frecvență fr, unde $fr[i]$ = numărul de elemente din vectorul v care au costul i.
Pentru fiecare operație, la modificarea unei valori din vector, vom decrementa frecvența costului

pentru vechea valoare și vom incrementa frecvența costului pentru noua valoare. Pentru a determina costul total minim pentru a obține cel puțin p numere prime în vector (valoarea afișată după executarea operației), vom parcurge vectorul de frecvență de la stânga la dreapta ($i = 0, \ldots, 57$). La fiecare pas i, pentru a calcula costul total minim adunăm la o variabilă *cmin* produsul dintre $fr[i]$ și i (există $fr[i]$ numere care pot fi transformate în numere prime cu costul i), iar într-o variabilă nr reținem câte numere prime au fost deja obținute. În momentul în care $nr + fr[i] > p$, parcurgerea se oprește și adunăm la *cmin* doar costul obținerii celor $p - nr$ numere prime care mai sunt necesare (adică adunăm $fr[i] \cdot (p - nr)$).

În funcție de cum selectăm cele mai mici p costuri și în funcție de cum calculăm costurile, se pot obține diverse punctaje parțiale.

3.6 Cod-sursă pentru problema Primprim

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

int vals[1000002];
int primes[200002], cnt;
int ans[1100002];
bool pr[1100002];

int fr[202];

int main()
{
 int c;
 fin >> c;
 int n, a = 1100000, i, j;
 fin >> n;
 for (i = 1; i <= n; i++) fin >> vals[i];
 //precalculam raspunsul optim pentru fiecare numar de la 1 la a folosind ciurul lui Eratostene
 for (i = 2; i <= a; i++)
     {
      if (pr[i] == 0)
         {
                   primes[cnt++] = i;
           for (j = i+i; j <= a; j += i)
                pr[j] = 1;
         }
     }
 int poz = -1;
 //parcurg valorile de la 1 la a pentru a afla raspunsul optim dupa ce am aflat numerele prime
 ans[1] = 1;
 for (i = 2; i <= a; i++)
     {
      if (pr[i] == 0)
          poz++;
      ans[i] = abs(i - primes[poz]);
      if (poz + 1 < cnt)
         ans[i] = min(ans[i], abs(i - primes[poz+1]));
     }
 //folosim vector de frecventa pentru a tine aceste diferente, care sunt destul de mici
 for (i = 1; i <= n; i++)
     fr[ans[vals[i]]]++;
```

```
if (c == 1)
    {
     int total = 0;
     for (i = 1; i <= n; i++)
          total += ans[vals[i]];
     fout << total;
     return 0;
    }
int q;
fin >> q;
for (i = 1; i <= q; i++)
    {
     int a, b, p;
     fin >> a >> b >> p;
     fr[ans[vals[a]]]--;
     vals[a] = b;
     fr[ans[vals[a]]]++;
     int dif = 0;
     int sol = 0;
     //la fiecare pas parcurg vectorul de frecventa pana cand dau de p diferente
     while (p)
           {
            sol += min(p, fr[dif]) * dif;
            p -= min(p, fr[dif]);
            dif++;
           }
     fout << sol << '\n';
   }
return 0;
}
```

Capitolul 4

OJI 2023, clasa a VIII-a

4.1 Problema Hibrid

Propusă de: stud. Andrei Onuț, Universitatea Yale, S.U.A.

O mașină hibrid se deplasează pe o șosea rectilinie folosind, alternativ, fie motorul termic (pe benzină), fie motorul electric. Axa numerelor întregi poate fi folosită pentru a descrie coordonatele de pe șosea. Deplasarea mașinii folosind motorul electric se efectuează fără taxă, dar unele porțiuni din șosea necesită folosirea motorului termic, ceea ce impune plata anumitor taxe. Se cunoaște lista celor P porțiuni taxabile de șosea, numerotate de la 1 la P, **oricare două dintre ele fiind disjuncte**. Fiecare porțiune este descrisă de trei numere întregi: st_i, dr_i și c_i ($1 \leq i \leq P$), cu semnificația că pe porțiunea de șosea situată între coordonatele st_i și dr_i (inclusiv la capetele st_i și dr_i) se va folosi motorul termic și se va achita taxa în valoare de c_i lei. Această taxă se va plăti la fiecare traversare a porțiunii, indiferent de sensul deplasării.

Traseul pe care mașina hibrid îl are de străbătut conține N borne amplasate pe șosea, numerotate de la 1 la N, în ordinea în care acestea trebuie vizitate. Pentru fiecare dintre cele N borne se cunoaște coordonata poziției sale pe șosea: $x_1, x_2, x_3, \ldots, x_N$. Deplasarea între două borne consecutive de pe traseu, adică între borna i și borna $(i+1)$ (pentru fiecare i: $1 \leq i < N$), se face pe drumul cel mai scurt, respectiv pe segmentul de dreaptă ce unește punctele cu coordonatele x_i și x_{i+1} de pe șosea. **Mașina hibrid va începe traseul din dreptul bornei cu numărul de ordine 1, adică de la coordonata x_1 de pe șosea.**

De asemenea, se știe că nicio bornă de pe traseu nu se află în interiorul sau la capetele porțiunilor taxabile, unde se folosește deplasarea cu motorul termic.

Cerințe

Să se determine:

1. numărul de ordine al porțiunii taxabile peste care se va trece de cele mai multe ori în călătorie (folosind motorul termic). Dacă există mai multe astfel de porțiuni, se va alege cea cu indicele minim, în funcție de ordinea dată în fișierul de intrare. De asemenea, în caz că nu se va trece peste nicio porțiune taxabilă, acest număr va fi egal cu -1.
2. suma totală, exprimată în lei, care trebuie plătită pentru a parcurge traseul stabilit. În caz că nu se va trece peste nicio porțiune taxabilă, atunci această sumă va fi egală cu 0.

Date de intrare

Pe prima linie a fișierului de intrare `hibrid.in` se află, separate între ele prin câte un spațiu, trei numere naturale, C, P și N, cu semnificațiile din enunț. C poate avea fie valoarea 1, fie valoarea 2, în funcție de cerința care trebuie rezolvată. Pe următoarele P linii se află, separate între ele prin câte un spațiu, câte trei numere întregi: st_i, dr_i și c_i, cu semnificația de mai sus. Pe următoarea linie se află N numere întregi, separate între ele prin câte un spațiu, reprezentând, în ordine, coordonatele bornelor ce trebuie vizitate: $x_1, x_2, x_3, \ldots, x_N$.

Date de ieșire

Fișierul de ieșire `hibrid.out` va conține, pe prima linie, un singur număr întreg, în funcție de cerința care trebuie rezolvată. Dacă $C = 1$, atunci se va rezolva cerința 1, altfel, se va rezolva cerința 2.

Restricții

- $2 \leq P \leq 100\,000$ și $2 \leq N \leq 200\,000$.
- $-300\,000 \leq st_i < dr_i \leq 300\,000$ și $1 \leq c_i \leq 100\,000$, pentru fiecare i: $1 \leq i \leq P$.
- $-1\,000\,000 \leq x_i \leq 1\,000\,000$, pentru fiecare i: $1 \leq i \leq N$.
- Întrucât au dimensiuni neglijabile, pot exista și două sau mai multe borne situate la aceeași coordonată pe șosea.
- Pe durata întregului traseu, motorul termic (pe benzină) este utilizat doar pentru parcurgerea porțiunilor taxabile peste care mașina hibrid trebuie să treacă. În rest, se folosește doar motorul electric, pentru a reduce poluarea.
- Pentru teste în valoare de 49 de puncte, $C = 1$, iar pentru restul de teste, $C = 2$.

#	Puncte	Restricții
1	11	Pentru efectuarea traseului nu se va trece peste nicio porțiune taxabilă
2	8	$0 \leq st_i$, x_j și dr_i, $x_j \leq 70$, pentru fiecare i și j: $1 \leq i \leq P$, $1 \leq j \leq N$
3	12	$\lvert x_{i+1} - x_i \rvert \leq 70$, pentru $1 \leq i < N$ și $\lvert x_i \rvert \leq 300\,000$, pentru $1 \leq i \leq N$
4	40	$P, N \leq 3\,000$
5	29	Nu există alte restricții suplimentare

Exemple

hibrid.in	hibrid.out
1 2 4 4 8 10 -10 -9 22 -14 20 -14 0	2
2 2 4 4 8 10 -10 -9 22 -14 20 -14 0	86

Explicații

Primul exemplu ($C = 1$): Există două porțiuni taxabile ($P = 2$):

- porțiunea 1 cuprinde coordonatele: 4, 5, 6, 7, 8 și are taxa de 10 lei la fiecare trecere;
- porțiunea 2 cuprinde coordonatele: -10, -9 și are taxa de 22 de lei la fiecare trecere.

Traseul pe care mașina hibrid îl are de parcurs este alcătuit din $N = 4$ borne, situate la coordonatele: -14 (prima bornă, **din dreptul căreia se începe traseul**), 20 (a doua bornă), -14 (a treia bornă; de remarcat că este situată la aceeași coordonată ca și prima bornă!), respectiv 0 (a patra bornă). Peste prima porțiune taxabilă se va trece de două ori, iar peste cea de a doua se va trece de trei ori. Prin urmare, se va afișa 2.

Al doilea exemplu ($C = 2$): Conform explicației de mai sus, se va afișa 86 (2 treceri $\times$ 10 lei/trecere + 3 treceri $\times$ 22 lei/trecere = 86 de lei, adică suma totală plătită pentru efectuarea traseului).

4.2 Rezolvarea problemei Hibrid

Observație inițială:

Pentru că cele P porțiuni taxabile sunt disjuncte două câte două, iar nicio bornă dintre cele N nu se poate afla la capetele sau în interiorul segmentelor ce descriu porțiunile taxabile (mai formal, nu există nicio pereche de indici (i, j): $1 \leq i \leq P$ și $1 \leq j \leq N$ pentru care $st_i \leq x_j \leq dr_i$), înseamnă că dr_i **ar putea fi *ignorat* pentru fiecare i:** $1 \leq i \leq P$. Astfel, dacă ar fi nevoie să verificăm, de exemplu, de câte ori s-a trecut peste o porțiune taxabilă, ar trebui doar verificat de câte ori punctul de coordonată st_i a fost *vizitat* în timp ce se efectua deplasarea între două borne consecutive de pe traseu; a număra de câte ori s-a trecut peste o porțiune taxabilă i ($1 \leq i \leq P$) este echivalent cu a număra numărul de indici j ($1 \leq j < N$) pentru care: $\min(x_j, x_{j+1}) < st_i < \max(x_j, x_{j+1})$; dr_i, observăm, nu mai influențează această abordare.

Notații:

$$\min(a, b) = \begin{cases} a & \text{dacă } a < b; \\ b & \text{altfel.} \end{cases}$$

$$\max(a, b) = \begin{cases} a & \text{dacă } a > b; \\ b & \text{altfel.} \end{cases}$$

Subtask 1:

Întrucât se cunoaște faptul că pentru efectuarea traseului nu se va trece peste niciuna dintre cele P porțiuni taxabile (unde este impusă folosirea motorului termic), în fișierul de ieșire se va afișa -1 (în cazul în care $C = 1$) sau 0 (în cazul în care $C = 2$).

Subtask 2:

Întrucât $0 \leq x_i \leq 70$, pentru fiecare i: $1 \leq i \leq N$, înseamnă că lungimea fiecărui segment de dreaptă între două borne consecutive de pe traseu va fi de cel mult 70 ($|x_{i+1} - x_i| \leq 70$, pentru fiecare i: $1 \leq i < N$).

Să considerăm introducerea următorului tablou unidimensional $fr[0, \ldots, 70]$ ce conține exact 71 de elemente, cu următoarea semnificație: $fr[x]$ ($0 \leq x \leq 70$) = de câte ori s-a trecut, în timpul

efectuării traseului, prin coordonata x (de pe axa numerelor). Observăm că $fr[x] = 0$ în cazul în care nu s-a trecut niciodată prin dreptul coordonatei x.

Pentru a calcula într-o complexitate de $O(71 \times N)$ care vor fi valorile elementelor din vectorul $fr[0, \ldots, 70]$, putem utiliza următoarea metodă, de tip *brute-force*:

```
for(int i = 1; i < N; ++i) /// iterăm prin lista coordonatelor de borne
{
  int p1 = min(x[i], x[i + 1]); /// capăt „stânga” pe axa numerelor întregi
  int p2 = max(x[i], x[i + 1]); /// capăt „dreapta” pe axa numerelor întregi

  for(int j = p1; j <= p2; ++j) /// iterăm prin coordonatele de pe axa numerelor
    ++fr[j]; /// incrementăm numărul de „vizite” prin dreptul coordonatei j
}
```

Apoi, pentru cerința $C = 1$ se determină care este indicele minim i ($1 \leq i \leq P$) pentru care valoarea $fr[st_i]$ este maximă, iar pentru cerința $C = 2$ se calculează suma:

$$\sum_{i=1}^{P} c_i \times fr[st_i].$$

Complexitate temporală totală: $O(P + N \times 71)$.

De remarcat! Pentru acest subtask se mai știe, în plus, și că $0 \leq st_i < dr_i \leq 70$, pentru fiecare i: $1 \leq i \leq P$. De asemenea, de vreme ce porțiunile taxabile sunt disjuncte două câte două, înseamnă că numărul P nu poate fi prea mare. Mai exact, acesta este cel mult egal cu 35, și se poate obține, spre exemplu, pentru: $st_i = (i-1) \times 2$ și $dr_i = st_i + 1$, pentru fiecare i: $1 \leq i \leq P$, rezultând în intervalele: $[0, 1], [2, 3], \ldots, [68, 69]$. Astfel, înseamnă că și o soluție cu o complexitate de $O(P \times N) = O(35 \times N)$ va trece toate testele aferente acestui subtask.

Subtask 3:

Ca și în cazul precedent, avem în continuare că $|x_{i+1} - x_i| \leq 70$, pentru fiecare i: $1 \leq i < N$, ceea ce înseamnă că am putea *simula* traseul descris, iterând prin fiecare coordonată prin care mașina hibrid se deplasează. De asemenea, valoarea absolută a coordonatelor bornelor este cel mult egală cu 300 000; prin urmare, pentru fiecare deplasare între două borne consecutive am putea aplica același algoritm ca și mai sus. Trebuie avut grijă la faptul că indicii ale căror valori dorim să le modificăm într-un tablou unidimensional de frecvență/numărare ar putea deveni și negativi. Așadar, vom aplica indicilor o *translație spre dreapta* cu 300 000 poziții: mai exact, poziția $-300\,000$ va fi translatată în poziția 0, poziția $-299\,999$ va fi translatată în poziția 1, …, poziția 0 va fi translatată în poziția 300 000, …, poziția 300 000 va fi translatată în poziția 600 000.

Secvența de cod ar putea deveni:

```
for(int i = 1; i < N; ++i) //iterăm prin lista coordonatelor de borne
{
  int p1 = min(x[i], x[i + 1]);//capăt „stânga” pe axa numerelor întregi
  int p2 = max(x[i], x[i + 1]);//capăt „dreapta” pe axa numerelor întregi

  for(int j=p1; j<=p2; ++j)//iterăm prin coordonatele de pe axa numerelor
    ++fr[j + 300000]; //incrementăm numărul de „vizite” prin dreptul coordonatei j
}
```

→ unde: $fr[0, \ldots, 600\,000]$ este tabloul unidimensional cu ajutorul căruia numărăm de câte ori s-a trecut prin dreptul fiecărei coordonate. Așadar, dacă dorim să știm, de exemplu, de câte ori am trecut prin coordonata x ($-300\,000 \leq x \leq 300\,000$), accesăm valoarea $fr[x + 300\,000]$.

Ca și în cazul precedent, complexitatea temporală totală rămâne: $O(P + N \times 71)$.

Subtask 4:

Vom *simula*, în continuare, traseul mașinii pe șosea, însă, pentru a ne încadra în limita de timp dată, nu vom mai parcurge coordonate de pe axa numerelor, ci vom parcurge direct lista de intervale ce descriu cele P porțiuni taxabile.

Mai exact spus, vom parcurge fiecare *segment* $(i, i+1)$ $(1 \leq i < N)$ al traseului. Vom itera, apoi, prin lista celor P porțiuni, un indice j $(1 \leq j \leq P)$ și dacă se întâmplă să avem: $\min(x_i, x_{i+1}) < st_j < \max(x_i, x_{i+1})$, în lumina observației inițiale, înseamnă că porțiunea taxabilă cu indicele j a fost traversată în timp ce se făcea deplasarea de la coordonata bornei i la cea a bornei $(i+1)$. Astfel, tot într-un tablou unidimensional de frecvență (ce conține P elemente), se va contoriza de câte ori s-a trecut peste fiecare porțiune taxabilă.

Astfel, am putea avea următoarea abordare:

```
for(int i = 1; i < N; ++i) /// iterăm prin lista coordonatelor de borne
{
  int p1 = min(x[i], x[i + 1]); //capăt „stânga” pe axa numerelor întregi
  int p2 = max(x[i], x[i + 1]); //capăt „dreapta” pe axa numerelor întregi

  for(int j = 1; j <= P; ++j) //iterăm prin lista de porțiuni taxabile
    if(p1 < st[j] && st[j] < p2)
      ++fr2[j]; //marcăm trecerea peste porțiunea j
}
```

În mod similar, ca în soluțiile descrise anterior, pentru cerința $C = 1$ se determină care este indicele minim i $(1 \leq i \leq P)$ pentru care valoarea $fr2[i]$ este maximă, iar pentru cerința $C = 2$ se calculează suma:

$$\sum_{i=1}^{P} c_i \times fr2[i].$$

Complexitatea totală este: $O(P \times N)$.

Subtask 5:

Remarcăm că cele P porțiuni de șosea sunt incluse complet în porțiunea reprezentată de segmentul $[-300\,000, 300\,000]$ de pe șosea. Astfel, o observație este că, dacă mașina hibrid trebuie să parcurgă un segment $[l, r]$ $(l \leq r)$ de pe șosea, se poate modifica acest interval astfel încât să reprezinte intersecția: $[l, r] \cap [-300\,000, 300\,000]$; în cazul în care intersecția este vidă, se poate *ignora* segmentul $[l, r]$ de pe traseu, din moment ce parcurgerea acestuia nu va influența niciuna dintre cele P porțiuni taxabile.

De această dată, este nevoie ca *marcarea* unui interval $[l, r]$ să fie efectuată în timp constant, $O(1)$. Astfel, poate fi folosită, de exemplu, metoda *Vectorului de Diferențe* (*Difference Array* în Engleză); în România, această tehnică mai este cunoscută și sub numele de *Șmenul lui Mars* și puteți citi mai multe pe infoarena.ro sau pbinfo.ro. Întrucât l sau r pot avea valori negative, se va alege din nou o *translație spre dreapta* cu 300 000 de poziții. În momentul actualizării intervalului, valoarea de pe poziția $(l + 300\,000)$ în tabloul unidimensional folosit pentru *Șmenul lui Mars* va crește cu o unitate, iar cea de pe poziția $(r + 300\,000 + 1)$ va scădea cu o unitate.

Complexitatea totală a acestei soluții este: $O(P + N + max_d)$ și garantează trecerea cu succes a tuturor testelor, unde max_d reprezintă diferența maximă dintre două coordonate vizitate în timpul efectuării traseului (considerând intersecțiile cu segmentul $[-300\,000, 300\,000]$): $max_d \leq 600\,000$.

4.3 Cod-sursă pentru problema Hibrid

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

ifstream f("hibrid.in");
ofstream g("hibrid.out");

using ll = long long int;

typedef pair<int, int> Segment;
typedef pair<Segment, int> Road;

const int PMAX = 1e5;
const int VMAX = 3e5;

int C, P, N;
Road A[(PMAX + 10)];

int V[((VMAX + 1 + VMAX) + 10)];

void Read()
{
    f >> C >> P >> N;
    for (int i = 1; i <= P; ++i)
        f >> A[i].first.first >> A[i].first.second >> A[i].second;
    return;
}

int my_min(int a, int b)
{
    return ((a < b) ? a : b);
}

int my_max(int a, int b)
{
    return ((a > b) ? a : b);
}

void Solve()
{
    int last_X = 0, X = 0;
    for (int i = 1; i <= N; ++i)
    {
        f >> X;
        if (i > 1)
        {
            int l = my_min(last_X, X);
            int r = my_max(last_X, X);
            l = my_max(-VMAX, l), r = my_min(VMAX, r);
            if (l <= r)
                ++V[(l + VMAX)], --V[((r + 1) + VMAX)];
        }
        last_X = X;
    }
    for (int i = (-VMAX + 1); i <= VMAX; ++i)
        V[(i + VMAX)] += V[((i - 1) + VMAX)];
    return;
}

int main()
```

```
{
    Read();
    Solve();
    int ans_1 = -1;
    ll ans_2 = 0;
    for (int i = 1; i <= P; ++i)
    {
        int X = (A[i].first.first + VMAX);
        if (V[X] > 0)
        {
            if (ans_1 == -1)  ans_1 = i;
                else
                if (V[X] > V[(A[ans_1].first.first + VMAX)]) ans_1 = i;
            ans_2 += 1LL * V[X] * A[i].second;
        }
    }
    if (C == 1) g << ans_1;
        else    g << ans_2;
    g << '\n';
    return 0;
}
```

4.4 Problema Tema

Propusă de: prof. Daniela Lica, Centrul Județean de Excelență Prahova

Macarie a primit ca temă la Informatică următorul enunț de problemă: *„Se consideră un șir A cu N numere naturale nenule, numerotate începând de la 1 până la N. Numim **secvență** o succesiune de termeni situați pe **poziții consecutive** în șir, iar **lungimea secvenței** o reprezintă numărul de termeni din care este formată. **Costul unei secvențe** este egal cu produsul dintre suma valorilor prime din secvență și suma celor compuse. Numărul compus este un număr care are cel puțin un divizor natural diferit de 1 și de el însuși, iar un număr este prim dacă are exact doi divizori naturali distincți, pe 1 și pe el însuși.”.*

Știm că numărul 1 nu este nici număr prim, nici compus, deci nu influențează costul niciunei secvențe în care se găsește. Evident, costul unei secvențe care nu conține niciun număr prim sau al unei secvențe care nu conține niciun număr compus este egal cu 0. De asemenea, suma valorilor prime dintr-o secvență care conține un singur număr prim X este egală cu X; în mod similar, suma valorilor compuse dintr-o secvență care conține un singur număr compus Y este egală cu Y.

Cerințe

Ajutați-l pe Macarie să rezolve următoarele două cerințe ale temei:

1. Să se determine lungimea maximă a unei secvențe din șirul A pentru care costul ei este mai mic sau egal decât un număr natural nenul K.
2. Presupunem că fiecare număr **compus** din șirul A este înlocuit cu produsul dintre **cel mai mic** factor prim al său și **cel mai mare** factor prim al său. Să se determine secvența de lungime maximă din șirul nou obținut, pentru care cel mai mare divizor comun al numerelor din care este formată este diferit de 1. Se vor afișa pozițiile primului și ultimului element din secvență. Dacă sunt mai multe astfel de secvențe de lungime maximă, se va afișa cea pentru care poziția primului său element este maximă.

Date de intrare

Pe prima linie a fișierului de intrare `tema.in` se află trei numere naturale nenule C, N și K, în această ordine, separate prin câte un spațiu, unde C este numărul cerinței care trebuie rezolvată (1 sau 2), iar N și K au semnificația din enunț. Pe a doua linie se află N numere naturale nenule, separate între ele prin câte un spațiu, reprezentând, în ordine, termenii șirului A.

Date de ieșire

Pe prima linie a fișierului de ieșire `tema.out`:

- se scrie un număr natural nenul, reprezentând lungimea maximă determinată pentru prima cerință, dacă $C = 1$;
- se scriu două numere naturale nenule, separate printr-un spațiu, reprezentând, în ordine, pozițiile primului, respectiv ultimului element din secvența de lungime maximă, determinată conform celei de a doua cerințe, dacă $C = 2$.

Restricții

- $2 \leq N \leq 100\,000$.
- $1 \leq K \leq 10^{18}$; **Numărul K nu are niciun rol pentru cerința 2.**
- $1 \leq A_i \leq 1\,000\,000$, pentru fiecare i: $1 \leq i \leq N$.

- **În cazul ambelor cerințe, există o secvență soluție ce are lungimea cel puțin egală cu 2.**
- Există cel puțin un element diferit de 1 în șirul A.

#	Puncte	Restricții
1	10	$C = 1$ și $N = 2$
2	25	$C = 1$ și $N \leq 4\,000$
3	15	$C = 1$ și $5\,000 < N$
4	10	$C = 2$ și $N = 2$
5	25	$C = 2$ și $N \leq 4\,000$
6	15	$C = 2$ și $5\,000 < N$

Exemple

tema.in	tema.out	Explicații
1 10 45 10 2 3 1 4 5 8 2 6 3	5	$C = 1$, $N = 10$ și $K = 45$. Secvența $(2, 3, 1, 4, 5) = (A_2, A_3, A_4, A_5, A_6)$ are costul egal cu $(2 + 3 + 5) \times 4 = 40$. Nu există, în șirul A, o secvență de lungime mai mare și de cost mai mic sau egal decât 45.
2 10 20 1 2 32 4 42 49 7 21 1 63	5 8	$C = 2$, $N = 10$ și $K = 20$. După modificări, șirul A devine: $(1, 2, 4, 4, 14, 49, 7, 21, 1, 21)$. Există două secvențe de lungime maximă pentru care cel mai mare divizor comun („c.m.m.d.c.”) al numerelor din care sunt formate este diferit de 1: $(2, 4, 4, 14)$ (poziția în șir a primului element este 2, iar c.m.m.d.c.-ul elementelor sale este 2), respectiv $(14, 49, 7, 21)$ (poziția în șir a primului element este 5, iar c.m.m.d.c.-ul elementelor sale este 7). Pentru că sunt două secvențe de lungime maximă, în enunț este precizat că se va alege cea pentru care poziția primului său element este maximă, adică, în acest caz, $(14, 49, 7, 21) = (A_5, A_6, A_7, A_8)$.

4.5 Rezolvarea problemei Tema

Pentru cerința 1, soluția în complexitate $O(ValMax \times \log ValMax + N)$ obține punctaj maxim, adică 50 de puncte. Folosind **Ciurul lui Eratostene**, în complexitate $O(ValMax \times \log ValMax)$, se vor determina toate numerele prime $\leq ValMax$ și, pentru fiecare număr compus, reținem cel mai mic și cel mai mare factor prim al său.

Algoritmul de *ciur* generează toate numerele prime. În momentul găsirii unui număr prim p, marcăm toți multiplii lui de la $2 \times p$ până la cel mult $ValMax$ ca fiind neprimi. Pentru fiecare

multiplu (notat cu x), p devine cel mai mare factor prim al lui x (iar dacă x avea deja un factor prim, p îl suprascrie). Acest lucru este natural, căci p este cel mai mare număr prim întâlnit până la acest moment. În plus, dacă la momentul curent x era considerat prim, atunci p devine și cel mai mic factor prim al acestuia.

Determinarea secvenței de lungime maximă, pentru care costul acesteia este mai mic sau egală cu K, se realizează în complexitate $O(N)$, folosind *doi indici*. Pentru fiecare indice curent i, considerat capăt dreapta al secvenței curente, determinăm indicele de început (din stânga) al secvenței pentru care produsul nu depășește K. Dacă produsul este mai mare, se renunță la $A[st]$, incrementând st și actualizând valoarea produsului. Se va reține lungimea maximă a secvenței ce respectă cerința. Odată determinat st pentru poziția i, dorim să calculăm valoarea st' pentru poziția $i+1$. Încorporăm $A[i+1]$ în costul curent. Cât timp costul depășește K, este clar că pentru niciun i viitor $A[st]$ nu va mai fi parte din soluție (căci, cu cât vom deplasa i spre dreapta, cu atât mai mult vom depăși costul K). Deci, eliminăm $A[st]$ și încercăm pe rând valorile $st+1, st+2, \ldots$ până revenim sub costul K. Prima valoare acceptabilă este capătul stâng st' corespunzător lui $i+1$.

Pentru o soluție care determină în $O(N^3)$ secvența, se obțin aproximativ 20 de puncte.

Pentru o soluție care determină în $O(N^2)$ secvența, se obțin aproximativ 35 de puncte.

Pentru cerința 2, soluția în complexitate $O(N)$ obține punctaj maxim, adică 50 de puncte.

Considerând șirul obținut după transformarea elementelor compuse, vom determina, pentru fiecare poziție i din șir, cât de mult ne putem *extinde* spre stânga și spre dreapta cu o secvență care conține cel mai mic factor prim al lui $A[i]$ (notat *fpmin[A[i]]*), respectiv cel mai mare factor prim a lui $A[i]$ (notat *fpmax[A[i]]*), astfel:

- $st[0][i]$ - cel mai mic indice din stânga unde regăsim *fpmin[A[i]]* ca factor în toate elementele dintre acel indice și i;
- $st[1][i]$ - cel mai mic indice din stânga unde regăsim *fpmax[A[i]]* ca factor în toate elementele dintre acel indice și i;
- $dr[0][i]$ - cel mai mare indice din dreapta unde regăsim *fpmin[A[i]]* ca factor în toate elementele de la i până la acea poziție;
- $dr[1][i]$ - cel mai mare indice din dreapta unde regăsim *fpmax[A[i]]* ca factor în toate elementele de la i până la acea poziție.

Pentru fiecare element $A[i]$, lungimea maximă a secvenței care îl conține și care are cel mai mare divizor comun diferit de 1 este: $\max((dr[0][i] - st[0][i] + 1), (dr[1][i] - st[1][i] + 1))$.

Pentru o soluție care determină în $O(N^3)$ secvența, se obțin aproximativ 20 de puncte.

Pentru o soluție care determină în $O(N^2)$ secvența, se obțin aproximativ 35 de puncte.

O altă abordare în $O(N)$ folosește aceeași tehnică a celor *doi indici* de la cerința 1. Pentru un capăt dreapta i, presupunem că am calculat capătul stâng st care produce cea mai lungă secvență cu c.m.m.d.c. diferit de 1. Cum actualizăm secvența pentru $i+1$? Putem menține un vector de frecvențe al tuturor factorilor primi din secvența curentă. Când adăugăm sau eliminăm un element în/din secvență, incrementăm sau decrementăm frecvențele celor doi factori (dacă factorii unui element sunt egali, modificăm frecvențele doar cu 1, nu cu 2). Atunci, o secvență cu c.m.m.d.c. diferit de 1 are proprietatea că frecvența cel puțin a unui factor de la capătul secvenței este egală cu lungimea secvenței. Când avansăm de la i la $i+1$, verificăm dacă secvența $[st, i+1]$ mai are această proprietate. Cât timp nu o are, eliminăm capătul stâng al secvenței.

4.6 Cod-sursă pentru problema Tema

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

const int NMAX = 1e5 + 5;
const int VALMAX = 1e6;
const long long KMAX = 1LL*1e18;

ifstream f ("tema.in");
ofstream g ("tema.out");

int A[NMAX];
int Task, N;
long long K;
int fpmax[VALMAX+5]; //fpmax[i] - Cel mai mare factor prim a lui i
int fpmin[VALMAX+5]; //fpmin[i] Cel mai mic factor prim a lui i
bool compus[VALMAX+5];

void Ciur () {
// determinam pentru fiecare numar cel mai mic si cel mai mare
// factor prim al in timp ce folosind Ciurul lui Eratostene
// determinam numerele prime <= VALMAX
// numarul x este prim atunci compus[x] = false;
for (int i = 2; i <= VALMAX; i += 2) {
     fpmax[i] = fpmin[i] = 2;
     compus[i] = true;
    }
compus[1] = compus[0] = true;
compus[2] = false;
for (int i = 3; i <= VALMAX; i += 2) {
    if (!compus[i]) {
       fpmin[i] = i;
       fpmax[i] = i;
       for (int j = 2*i; j <= VALMAX; j += i) {
           if (fpmin[j] == 0) fpmin[j] = i;
           fpmax[j] = i;
           compus[j] = true;
        }
     }
 }
}

void Read () {
f >> Task >> N >> K;
int S = 0;
for (int i = 1; i <= N; ++ i )
     { f >> A[i];
       S += (A[i]==1? 1: 0);
     }
}

void Solve_1 () {
// pentru fiecare indice curent i, considerat capat dreapta
// al secventei curente determinam indicele de inceput al secventei
// pentru care produsul nu depaseste K.
// Daca produsul este mai mare se renunta la A[st], incrementand st
// si actualizand valoarea produsului
// se va retine lungimea maxima a secventei ce respecta cerinta
long long S_prim = 0;
long long S_compus = 0;
```

```
int ans_st = 0, ans_dr = -1;
int st = 1;
for (int i = 1; i <= N; ++ i ) {
    if (compus[A[i]] && A[i] > 1) S_compus += 1LL * A[i];
    if (!compus[A[i]] && A[i] > 1)S_prim += 1LL * A[i];
    while (st <= i && S_prim * S_compus > K) {
          if (compus[A[st]] && A[st] > 1) S_compus -= A[st];
          if (!compus[A[st]] && A[st] > 1) S_prim -= A[st];
          ++ st;
        }
    if (i - st + 1 > ans_dr - ans_st + 1) {
            ans_st = st;
            ans_dr = i;
        }
    }
g << ans_dr - ans_st + 1 << '\n';
}

int st[2][NMAX];
int dr[2][NMAX];

void Solve_2 () {
//Considerand transformarea numerelor deja efectuata determinam pentru fiecare i
// st[0][i] - cel mai mic indice din stanga unde regasim fpmin[A[i]] ca factor
            //in toate elementele dintre acel indice si i
// st[1][i] - cel mai mic indice din stanga unde regasim fpmax[A[i]] ca factor
            //in toate elementele dintre acel indice si i
// dr[0][i] - cel mai mare indice din dreapta unde regasim fpmin[A[i]] ca factor
            //in toate elementele de la i pana la acea pozitie
// dr[1][i] - cel mai mare indice din dreapta unde regasim fpmax[A[i]] ca factor
            //in toate elementele de la i pana la acea pozitie
// Pentru fiecare A[i] lungimea maxima a secventei care il contine si are cmmdc > 1
// este = max{dr[0][i] - st[0][i] + 1, dr[1][i] - st[1][i] + 1}
for (int i = 0; i < 2; ++ i )
        for (int j = 1; j <= N; ++ j )
            st[i][j] = dr[i][j] = j;

for (int i = 2; i <= N; ++ i ) {
    if (A[i] == 1) {
        st[0][i] = st[1][i] = i+1;
        continue;
      }
    if (fpmin[A[i]] == fpmin[A[i-1]])
        st[0][i] = st[0][i-1];
        else if (fpmin[A[i]] == fpmax[A[i-1]])
            st[0][i] = st[1][i-1];
        else st[0][i] = i;
    if (fpmax[A[i]] == fpmin[A[i-1]])
        st[1][i] = st[0][i-1];
        else if (fpmax[A[i]] == fpmax[A[i-1]])
            st[1][i] = st[1][i-1];
        else st[1][i] = i;
    }

for (int i = N-1; i >= 1; -- i ) {
     if (A[i] == 1) {
        dr[0][i] = dr[1][i] = i-1;
        continue;
        }
    if (fpmin[A[i]] == fpmin[A[i+1]])
        dr[0][i] = dr[0][i+1];
```

```
        else if (fpmin[A[i]] == fpmax[A[i+1]])
            dr[0][i] = dr[1][i+1];
        else dr[0][i] = i;
    if (fpmax[A[i]] == fpmin[A[i+1]])
        dr[1][i] = dr[0][i+1];
        else if (fpmax[A[i]] == fpmax[A[i+1]])
            dr[1][i] = dr[1][i+1];
        else dr[1][i] = i;
    }

int ans_left = 1, ans_right = 0;
for (int i = 1; i <= N; ++ i ) {
    int lg = dr[0][i] - st[0][i] + 1;
    if (lg > (ans_right - ans_left + 1)) {
        ans_left = st[0][i];
        ans_right = dr[0][i];
       }
       else
       if (lg == (ans_right - ans_left + 1) && ans_left < st[0][i]) {
          ans_left = st[0][i];
          ans_right = dr[0][i];
        }
    lg = dr[1][i] - st[1][i] + 1;
    if (lg > (ans_right - ans_left + 1)) {
        ans_left = st[1][i];
        ans_right = dr[1][i];
        }
       else
       if (lg == (ans_right - ans_left + 1) && ans_left < st[1][i]) {
          ans_left = st[1][i];
          ans_right = dr[1][i];
       }
    }
g << ans_left << " " << ans_right << '\n';
}

int main () {
Ciur();
Read();
if (Task == 1) Solve_1();
    else Solve_2();
return 0;
}
```

Partea a II-a

Olimpiada Națională de Informatică - etapa națională -

Târgu Mureș, 1-5 aprilie 2023

Capitolul 5

ONI 2023, clasa a V-a

5.1 Problema Cadouri

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

Înaintea vacanței de Paște, la școală, s-au primit cadouri pentru elevii din clasa a V-a. Sunt N cutii cu bomboane și se cunoaște numărul de bomboane din fiecare cutie. **Numărul de cutii de bomboane primite de fiecare copil trebuie să fie același. Acest număr trebuie să fie mai mare sau egal cu** 2. Cutiile cu bomboane vor fi oferite în ordinea primirii, primele cutii primului copil, următoarele cutii celui de al doilea copil, următoarele cutii celui de al treilea copil etc.

Se dorește să se împartă cutiile cu bomboane unui număr cât mai mare de copii. De asemenea, mai este o condiție: să se împartă copiilor toate cutiile, sau cel mult una dintre cutii să rămână neoferită. În cazul că se ia decizia ca o cutie să nu fie dată copiilor, aceasta se păstrează de către doamna dirigintă pentru a-i servi pe aceștia la întoarcerea la școală, iar restul cutiilor cu bomboane se pun pe catedră în ordinea în care au fost primite, fără ca elevii să știe despre cea păstrată. Alegerea acestei cutii trebuie făcută astfel încât **numărul total de bomboane care se împart să fie cât mai mare**.

Cerințe

1. Care este numărul maxim de copii care vor primi cadouri?
2. Care este numărul maxim posibil de bomboane pe care le poate primi un copil în condițiile descrise mai sus?

Pentru ambele cerințe trebuie determinat și numărul de bomboane din cutia care eventual se păstrează.

Date de intrare

Fișierul de intrare `cadouri.in` conține pe prima linie un număr C, indicând cerința. Pe linia a doua se află un număr N, reprezentând numărul de cutii de bomboane primite la școală. Pe linia a treia se află N numere, separate prin câte un spațiu, reprezentând numărul de bomboane din fiecare cutie, în ordinea în care acestea au fost primite.

Date de ieșire

Fișierul de ieșire `cadouri.out` conține două numere naturale, separate printr-un singur spațiu liber, cu următoarea semnificație: pentru $C = 1$ prima valoare este numărul maxim de copii care primesc cadouri iar a doua este numărul de bomboane din cutia păstrată; pentru $C = 2$ prima valoare este numărul maxim de bomboane primite de un copil iar a doua reprezintă numărul de bomboane din cutia păstrată. Dacă nu se păstrează nicio cutie, în fișierul de ieșire a doua valoare scrisă va fi 0 (atât în cazul cerinței 1 cât și în cazul cerinței 2).

Restricții

- $2 \leq N \leq 100\,000$.
- Numerele de pe linia a treia sunt naturale, nenule, formate din cel mult 9 cifre.

#	Puncte	Restricții
1	23	$C = 1$
2	77	$C = 2$

Exemple

cadouri.in	cadouri.out	Explicații
1 5 2 7 4 1 2	2 1	Se rezolvă cerința 1. Doi copii primesc cadouri. Cutia cu o bomboană nu este dată nici unui copil.
2 5 2 7 4 1 2	9 1	Se rezolvă cerința 2. Doi copii primesc cadouri, primul copil primește cutiile cu 2 și 7 bomboane, iar al doilea copil primește cutiile cu 4 și 2 bomboane. Deci primul copil primește număr maxim de bomboane, 9. Cutia cu o bomboană nu este dată nici unui copil.

5.2 Rezolvarea problemei Cadouri

Notăm cu n numărul de cutii primite la școală. Pentru a oferi cadouri la un număr maxim de elevi se vor distribui câte două cutii fiecăruia. Dacă numărul de cutii este impar este necesar să se păstreze una dintre ele. Pentru a asigura distribuirea unui număr total maxim de bomboane, cutia păstrată trebuie să fie una cu număr minim de bomboane.

Cerința 1

Este necesară determinarea valorii minime din șirul dat și se afișează valorile $n/2$ și acest minim.

Cerința 2

Dacă n este par se vor distribui toate cutiile, iar soluția este maximul din șirul de sume de câte două elemente consecutive, $v[1] + v[2]$, $v[3] + v[4] \ldots$, (am notat cu v șirul dat la intrare). Pentru cazul cu n impar apar două situații care trebuie analizate. Dacă valoarea minimă din șir este pe poziție impară, maximul care poate fi dat unui copil se poate obține fie dintre valorile $v[1] + v[2]$,

$v[3]+v[4]\ldots$, formate înainte de poziția apariției minimului fie dintre valorile $v[n]+v[n-1]$, $v[n-2]+v[n-3]$ formate după poziția aparației minimului. Dacă minimul este pe o poziție pară (să o notăm în continuare cu p o poziție unde se află minimul), trebuie să mai luăm în calcul și valoarea $v[p-1]+v[p+1]$.

Întrucât elementele șirului dat nu sunt distincte, se observă că și minimul poate apărea de mai multe ori. Astfel, trebuie realizate cele prezentate mai sus pentru fiecare poziție unde se găsește un element cu valoarea minimă (notăm mai departe cu p o astfel de poziție).

O abordare care șterge în mod repetat elementul de pe o poziție p și aplică strategia descrisă pentru cazul cu n par nu se încadrează în timp pe toate testele.

Pentru a obține un program care rulează mai rapid vom calcula un șir (notat ms) care la poziția i curentă memorează cea mai mare sumă a unei valori de forma $v[1]+v[2]$, $v[3]+v[4]\ldots$, cu indici mai mici decât i precum și un alt șir (notat md) care memorează rezultatele unui calcul similar din dreapta: la poziția curentă se află maximul sumelor de câte două elemente din perechi de forma $v[n]+v[n-1]$, $v[n-2]+v[n-3]$, cu indici mai mari ca poziția curentă. Astfel, la fiecare poziție p unde se află un element cu valoare minimă este suficient să luăm în calcul valorile $ms[p-1]$, $md[p+1]$ (dacă p este impar), respectiv $ms[p-2]$, $md[p+2]$ și $v[p-1]+v[p+1]$ (dacă p este par).

5.3 Cod-sursă pentru problema Cadouri

```
#include <fstream>
using namespace std;
int c, n, minim, i, s, d, sol;
int v[100010], ms[100010], md[100010];

int main () {
    ifstream fin ("cadouri.in");
    ofstream fout("cadouri.out");
    fin>>c>>n;
    minim = 2000000000;
    for (i=1;i<=n;i++) {
        fin>>v[i];
        if (v[i] < minim) minim = v[i];
    }
    for (i=2; i<=n; i+=2) {
        s = v[i] + v[i-1];
        ms[i] = max(ms[i-2], s);
    }
    for (i=n-1; i>=1; i-=2) {
        d = v[i] + v[i+1];
        md[i] = max(md[i+2], d);
    }
    if (c == 1) fout<<n/2<<" ";
       else
       {
        if (n%2 == 0) fout<<ms[n]<<" ";
           else
           {// n impar
            for (i=1; i<=n; i++) {
                if (v[i] == minim) {
                    if (i%2 == 1) {
                        // minim pe pozitie impara
                        sol = max(sol, ms[i-1]);
                        sol = max(sol, md[i+1]);
                    } else {
```

```
                        sol = max(sol, v[i-1]+v[i+1]);
                        sol = max(sol, ms[i-2]);
                        sol = max(sol, md[i+2]);
                    }
                }
            }
            fout<<sol<<" ";
        }
       }
    if (n%2 == 1) fout<<minim<<"\n";
       else fout<<"0\n";
    return 0;
}
```

5.4 Problema Legos

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

Un joc de lego are P piese care sunt cuburi identice. Dorel se joacă cu ele pentru a construi diverse jucării, dar pentru aceasta are nevoie de ajutorul vostru.

Cerințe

Cunoscându-se numărul de piese P pe care le are, Dorel vrea să știe:

1. Numărul de piese din care poate să construiască cea mai mare fundație. O fundație are forma unui pătrat și are latura formată din cel puțin 3 piese (ca în figura 1).
2. Numărul de piese din cel mai înalt turn care se poate construi. Un turn din piese de lego Dorel îl construiește astfel: la început va face un pătrat pe care îl numește parter (sau etajul 0). Peste acesta va pune 4 piese în colțuri pe care le numește piloni. Apoi, peste piloni, va pune un nou pătrat pe care îl numește etaj 1. Peste acesta va pune din nou piloni, peste care va pune etajul 2. Și va continua, până la ultimul etaj. Peste ultimul etaj **nu** pune piloni. Toate etajele construite au același număr de piese și au forma de pătrat cu latura de cel puțin 3 piese. Înălțimea unui turn este dată de numărul de etaje. Pilonii **nu** sunt considerați etaje, aceștia fac parte din structura turnului. Dacă se pot construi mai multe turnuri având aceeași înălțime, atunci Dorel vrea să știe numărul de piese al turnului cu cele mai multe piese.(Vezi figura 2).
3. Numărul de terenuri de legoball care se pot construi folosind **toate** piesele de lego. Un teren de legoball are forma unui dreptunghi în care fiecare latură este formată din cel puțin 3 piese (ca în figura 3).

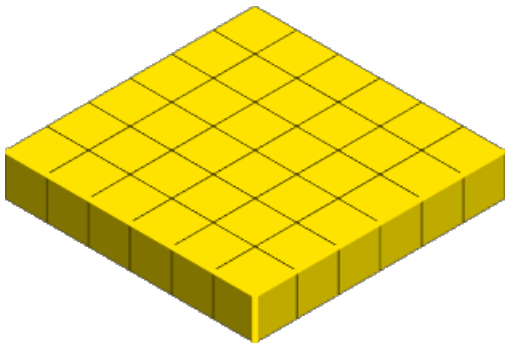


Figura 1. O fundație de mărime 6 x 6

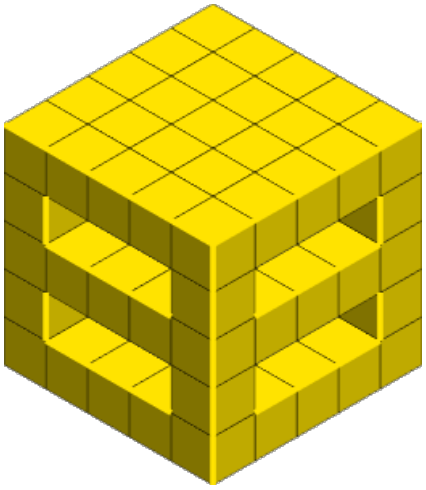


Figura 2. Un turn de înălțime 3, fiecare etaj are mărimea 5 x 5

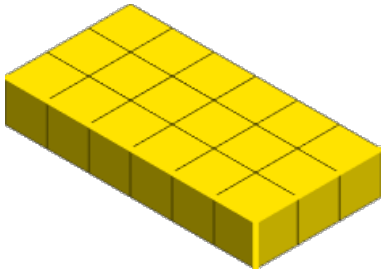


Figura 3. Un teren de legoball de mărime 6 x 3.

Date de intrare

Fișierul de intrare `legos.in` conține două numere naturale nenule C și P, separate printr-un singur spațiu liber, reprezentând cerința respectiv numărul de piese de lego pe care le are Dorel.

Date de ieșire

Pentru fiecare din cele 3 cerințe fișierul de ieșire `legos.out` va conține un singur număr care reprezintă răspunsul la acea cerință.

Restricții

- $1 \leq C \leq 3$
- $1 \leq P \leq 1\,000\,000\,000$
- Pentru cerința 2 un turn poate fi format doar din parter, dar nu poate fi format din parter și piloni (deoarece ar avea piloni peste ultimul etaj).

#	Puncte	Restricții
1	31	$C = 1$
2	33	$C = 2$
3	36	$C = 3$

Exemple

legos.in	legos.out	Explicații
1 29	25	Se rezolvă cerința 1. Sunt 29 piese de lego. Cea mai mare fundație ce poate fi construită are dimensiunea 5 x 5, este formată din 25 de piese.
2 19	16	Se rezolvă cerința 2. Sunt 19 piese de lego. Cel mai înalt turn care poate fi făcut este format doar din parter. Există două astfel de turnuri, unul are 9 piese iar celălalt 16. Dintre acestea mai multe sunt în turnul de 16 piese.
3 18	2	Se rezolvă cerința 3. Sunt două moduri de a construi un teren de legoball. Acestea au dimensiunile 3 x 6, respectiv 6 x 3.

5.5 Rezolvarea problemei Legos

Cerința 1

Se determină cel mai mare pătrat perfect mai mic decât P. Pentru a face asta vom itera până când pătratul numărului iterat depășește P. Acesta este primul pătrat perfect mai mare decât P, ceea ce înseamnă că precedentul este răspunsul.

Cerința 2

Se observă că cel mai înalt turn se obține atunci când fiecare etaj are dimensiunea 3 x 3 sau atunci când fiecare etaj are dimensiunea 4 x 4. O soluție este să iterăm numărul de etaje și să ne oprim atunci când numărul de piese depășește P. Pentru a optimiza această soluție se face observația adițională că numărul de piese este de forma $9+13\cdot(nrEtaje-1)$. Din aceasta se obține formula generală a numărului de etaje din cel mai înalt turn și apoi formula pentru numărul de piese din cel mai înalt turn cu etaje de mărime 3 x 3. Se repetă raționamentul pentru 4 x 4.

Cerința 3

Trebuie să numărăm în câte moduri poate fi scris P ca produs de două numere cu mențiunea că nici unul dintre numere să nu fie mai mic decât 3. Optimizarea este similară cu cea pentru verificarea primalității unui număr, vom itera până când $d\times d$ depășește P. Pentru fiecare divizor vom adăuga 2 la răspuns(corespunzător pentru d x $\frac{P}{d}$ și $\frac{P}{d}$ x d), cu mențiunea că dacă divizorul ridicat la puterea a doua este P se va adăuga doar 1 la răspuns ($\frac{P}{d}=d$ deci cele două moduri sunt la fel, d x d și d x d).

5.6 Cod-sursă pentru problema Legos

```
#include<cstdio>
int main()
{FILE* f=fopen("legos.in", "r");
 FILE* g=fopen("legos.out", "w");
 int c, P, i, ans=0;
 fscanf(f, "%d%d", &c, &P);
 if (c==1)
         {for (i=3; i*i<=P; ++i);
     fprintf(g, "%d\n", (i-1)*(i-1));
         }
     else
     if (c==2)
            {if (P<16) fprintf(g, "%d\n", 9);
                     else
                     if (P<22) fprintf(g, "%d\n", 16);
                         else  fprintf(g, "%d\n", (P-9)/13*13+9);
            }
        else
        {for (i=3; i*i<P; ++i)
                         if (P%i==0) ans+=2;
                 if (i*i==P) ++ans;
                 fprintf(g, "%d\n", ans);
        }
 fclose(f); fclose(g);
 return 0;
}
```

5.7 Problema Patinaj

Propusă de: prof. Victor-Claudiu Manz, Colegiul Național de Informatică „Tudor Vianu” București

Clubul Sportiv SEPI are și o secție de patinaj artistic. Conducerea clubului și-a propus să participe la proba de perechi a următoarei olimpiade și are de luat unele decizii privind echipele pe care le poate înscrie.

Fiecare echipă participantă la olimpiadă trebuie să fie formată dintr-o pereche de patinatori (o fată și un băiat) și un antrenor. În plus, valorile membrilor unei echipe trebuie să fie cât mai apropiate. Valoarea unui sportiv și respectiv a unui antrenor este calculată pe baza rezultatelor obținute la competițiile anterioare. Acestea sunt codificate sub forma unui singur număr cu cel mult 9 cifre. Fiecare cifră a numărului reprezintă un rezultat anterior, iar **suma cifrelor reprezintă valoarea sportivului, respectiv antrenorului**. De exemplu, numărul 18305 codifică rezultatele 1, 8, 3, 0, 5, obținute la ultimele 5 concursuri, ceea ce corespunde valorii $17 (= 1 + 8 + 3 + 0 + 5)$.

La olimpiadă fiecare sportiv și fiecare antrenor poate să facă parte din cel mult o echipă înscrisă. În plus, pentru fiecare echipă, dacă notăm cu V_M maximul dintre valorile antrenorului, fetei și băiatului și cu V_m minimul dintre valorile antrenorului, fetei și băiatului, înscrierea în concurs este permisă doar dacă $V_M - V_m \leq 1$.

Cerințe

Cunoscând numerele care codifică rezultatele antrenorilor, fetelor și băieților, scrieți un program care să determine:

1. Numărul maxim de echipe, N_P, pe care le poate înscrie Clubul Sportiv SEPI la olimpiadă astfel încât acestea să respecte regulile de mai sus.
2. Valoarea maximă, V, a unui antrenor al clubului care poate antrena o pereche de patinatori (fată, băiat), ce poate fi înscrisă la olimpiadă conform regulilor de mai sus și numărul de variante N_V în care se poate alege o echipă care poate fi pregătită de un antrenor de valoare V.

Date de intrare

Fișierul de intrare `patinaj.in` conține:

- pe prima linie numărul natural C care reprezintă numărul cerinței și poate avea una dintre valorile 1 sau 2;
- pe cea de-a doua linie, un număr natural N, care reprezintă atât numărul antrenorilor angajați, cât și al fetelor și al băieților legitimați la club;
- pe fiecare dintre următoarele trei linii câte N valori, despărțite prin câte un spațiu. Pe cea de-a treia linie, acestea reprezintă codificările rezultatelor anterioare ale celor N antrenori, pe cea de-a patra linie ele reprezintă codificările rezultatelor anterioare ale celor N fete, iar valorile de pe cea de-a cincea linie reprezintă codificările rezultatelor anterioare ale celor N băieți.

Date de ieșire

Fișierul de ieșire `patinaj.out` va conține:

- pentru cerința 1: numărul maxim de echipe N_P care pot fi înscrise la olimpiadă conform regulilor precizate mai sus;

- pentru cerința 2: două numere naturale, V și N_V, separate printr-un spațiu, reprezentând valoarea maximă a unui antrenor al clubului **pentru care există cel puțin o pereche pe care o poate antrena** și respectiv numărul variantelor în care clubul poate alege o echipă care poate fi pregătită de un antrenor cu valoarea V, dacă se rezolvă cerința 2. În cazul în care clubul nu poate înscrie nicio pereche, se va afișa **un singur număr**: -1.

Restricții

- $1 \leq N \leq 100\,000$
- fiecare dintre numerele citite de pe a treia, a patra și a cincea linie e fișierului este un **număr natural cu cel mult** 9 **cifre**.

#	Puncte	Restricții
1	51	$C = 1$
2	49	$C = 2$

Exemple

patinaj.in	patinaj.out	Explicații
1 4 8093 18305 20009 188 1803 3303331 909 91995 8017 20009 0 8017	2	Se pot forma cel mult 2 echipe. Prima ar putea fi formată din fata cu codificarea 1803 și băiatul cu codificarea 20009 și pregătită de antrenorul cu codificarea 20009. A doua poate fi formată din fata 3303331, băiatul 8017 și pregătită de antrenorul 18305.
2 4 8093 18305 20009 188 1803 3303331 909 91995 8017 20009 0 8017	17 4	Clubul are 4 antrenori cu valorile $20 = 8 + 0 + 9 + 3$, $17 = 1 + 8 + 3 + 0 + 5$, $11 = 2 + 0 + 0 + 0 + 9$ și $17 = 1 + 8 + 8$. Valoarea maximă este 20, dar nu există o pereche pe care să o poată pregăti antrenorul cu valoarea 20 conform regulilor impuse. În schimb, un antrenor cu valoarea 17 ar putea pregăti o pereche înscrisă la olimpiadă. Sunt 4 variante de alegere a unei echipe pregătite de un antrenor cu valoarea 17. Acestea ar putea avea în componență fata 3303331 și unul dintre cei doi băieți cu codificarea rezultatelor anterioare 8017. O astfel de pereche ar putea fi pregătită de antrenorul 18305 sau de 188.

5.8 Rezolvarea problemei Patinaj

Prima observație importantă este că, deși codificările rezultatelor anterioare pot fi numere destul de mari, valorile asociate sunt cuprinse între 0 (suma cifrelor lui 0 este 0) și 81 (suma cifrelor lui 999999999 este 81). Prin urmare, putem construi vectorii de frecvență nra asociat valorilor antrenorilor, nrf asociat valorilor fetelor și nrb asociat valorilor băieților.

Cerința 1

Pentru fiecare i cuprins între 0 și 81 vom aduna la rezultat numărul echipelor cu valoarea membrilor i sau $i+1$. Pentru i cuprins între 0 și 80 acest număr este $\min(nra[i]+nra[i+1], nrf[i]+nrf[i+1], nrb[i]+nrb[i+1])$. Pentru fiecare i este necesară actualizarea valorilor $nra[i+1], nrf[i+1], nrb[i+1]$, deoarece atunci când numărăm echipele cu valoarea membrilor $i+1$ sau $i+2$ trebuie să excludem antrenorii și sportivii pe care i-am folosit deja în echipe luate în considerare la pasul i. Pentru echipele cu valoarea tuturor membrilor 81 mai trebuie adunat $\min(nra[81], nrf[81], nrb[81])$.

Cerința 2

Vom parcurge valorile posibile ale antrenorilor și vom determina cel mai mare i cu proprietatea că există cel puțin un antrenor cu valoarea i și există atât băieți, cât și fete cu valori din mulțimea $\{i-1, i\}$ sau există atât băieți, cât și fete cu valori din mulțimea $\{i, i+1\}$. Dacă există un astfel de i, pe care îl notăm cu $i0$, atunci putem forma echipe cu antrenori de valoarea $i0$ luând (în toate modurile posibile) antrenori cu valoarea $i0$ și sportivi cu valori din mulțimea $\{i0-1, i0\}$ sau luând antrenori cu valoarea $i0$ și sportivi cu valori din mulțimea $\{i0, i0+1\}$. E nevoie de atenție pentru a nu număra de două ori echipele în care toți membrii au valoarea exact $i0$.

5.9 Cod-sursă pentru problema Patinaj

```
#include <fstream>
using namespace std;
const int VMAX = 81;
int nr_a[1 + VMAX], nr_f[1 + VMAX], nr_b[1 + VMAX];
int main()
{   ifstream in("patinaj.in");
    ofstream out("patinaj.out");
    int c, n, x, sc;
    in >> c >> n;
    for (int i = 0; i < n; i++)
    {
        in >> x;
        sc = 0;
        do
        {
            sc += x % 10;
            x /= 10;
        }
        while (x != 0);
        nr_a[sc]++;
    }
    for (int i = 0; i < n; i++)
    {
        in >> x;
        sc = 0;
        do
        {
            sc += x % 10;
            x /= 10;
        }
        while (x != 0);
        nr_b[sc]++;
    }
    for (int i = 0; i < n; i++)
    {
```

```
        in >> x;
        sc = 0;
        do
        {
            sc += x % 10;
            x /= 10;
        }
        while (x != 0);
        nr_f[sc]++;
    }
    if (c == 1)
    {   int nr = 0;
        for (int i = 0; i < VMAX; i++)
        {
            int minim = min(nr_a[i] + nr_a[i+1], nr_f[i] + nr_f[i+1]);
            minim = min(minim, nr_b[i] + nr_b[i+1]);
            nr += minim;
            if (minim > nr_a[i])
                nr_a[i+1] -= min(nr_a[i+1], minim - nr_a[i]);
            if (minim > nr_f[i])
                nr_f[i+1] -= min(nr_f[i+1], minim - nr_f[i]);
            if (minim > nr_b[i])
                nr_b[i+1] -= min(nr_b[i+1], minim - nr_b[i]);
        }
        nr += min(nr_a[VMAX], min(nr_f[VMAX], nr_b[VMAX]));
        out << nr << "\n";
    }
    else
    {   int v_max_a = -1;
        long long nr_var = 0;
        for (int i = 0; i <= VMAX; i++)
        {
            if (nr_a[i] == 0) continue;
            bool exista_st = false, exista_dr = false;
            long long nr_st = 0, nr_dr = 0;
            if (i > 0 && nr_f[i] + nr_f[i-1] != 0 && nr_b[i] + nr_b[i-1] != 0)
            {
                exista_st = true;
                nr_st = (long long)(nr_f[i] + nr_f[i-1]) * (nr_b[i] + nr_b[i-1]);
            }
            if (i < VMAX && nr_f[i] + nr_f[i+1] != 0 && nr_b[i] + nr_b[i+1] != 0)
            {
                exista_dr = true;
                nr_dr = (long long)(nr_f[i] + nr_f[i+1]) * (nr_b[i] + nr_b[i+1]);
            }
            if (exista_st || exista_dr || (nr_f[i] != 0 && nr_b[i] != 0))
            {
                v_max_a = i;
                nr_var = (nr_st + nr_dr) * nr_a[i] - (long long)nr_a[i] * nr_f[i] * nr_b[i];
            }
        }
        out << v_max_a;
        if (v_max_a != -1) out << " " << nr_var;
        out << "\n";
    }
    in.close();   out.close();
    return 0;
}
```

Capitolul 6

ONI 2023, clasa a VI-a

6.1 Problema Balon

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

Firma TgM produce plăci de umflat baloane. O placă de dimensiuni $n \times m$ este formată din n linii cu câte m celule pătrate de latură 1, fiecare celulă conținând un dispozitiv de care poate fi prins un balon pentru a fi umflat. Un balon are un nivel de umplere b_{ij} cuprins între 1 (dezumflat) și nivelul de umplere maxim posibil k. Introducerea unui nou volum de aer într-un balon umplut la nivelul maxim k, conduce la spargerea lui (nivel $k+1$). Fiecare balon spart este înlocuit automat cu un balon nou aflat la nivelul de umplere 1 înainte de a introduce un nou volum de aer în oricare dintre baloanele de pe placă.

Introducerea aerului în anumite baloane se face printr-o acționare care constă în următorii pași:

- se conectează un sistem cilindru-piston la dispozitivul dintr-o celulă aflată pe linia x și coloana y;
- se selectează o valoare naturală nenulă d;
- se apasă butonul Air aflat pe mânerul pistonului.

În urma acționării butonului Air, fiecare balon situat în pătratul cu colțul din stânga sus (x,y) cu latura d trece de la nivelul de umplere curent la nivelul de umplere imediat următor. Dacă pătratul de latură d depășește una sau mai multe din marginile plăcii, se transmite aer doar în baloanele aflate în interiorul acestuia. La acționarea butonului se consumă un număr de unități de volum de aer egal cu numărul de baloane aflate în interiorul pătratului.

Cerințe

Dându-se dimensiunile unei plăci n și m, nivelul maxim posibil de umplere a unui balon k, numărul p de acționări ale butonului Air, nivelul inițial de umplere al fiecărui balon de pe placă și pentru fiecare dintre acționările pistonului cele trei valori x, y și d corespunzătoare, scrieți un program care determină și afișează:

1. numărul de unități de aer consumate după cele p acționări ale butonului Air;
2. numărul de baloane sparte după cele p acționări ale butonului Air;
3. nivelul maxim de umplere a unui balon după cele p acționări ale butonului Air și numărul de baloane aflate la acest nivel de umplere.

Date de intrare

Fișierul de intrare `balon.in` conține pe prima linie un număr natural C, reprezentând cerința ce trebuie rezolvată (1, 2 sau 3), pe a doua linie patru numere naturale nenule n, m, k și p, cu semnificația din enunț, pe fiecare din următoarele n linii câte m valori reprezentând nivelul inițial de umplere a baloanelor de pe linia respectivă, iar pe fiecare din ultimele p linii câte trei numere naturale x, y și d corespunzătoare unei acționări a pistonului. Numerele aflate pe aceeași linie a fișierului sunt separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire `balon.out` va conține pe prima linie:

- dacă $C = 1$, numărul de unități de aer consumate după cele p acționări ale butonului Air;
- dacă $C = 2$, numărul de baloane sparte după cele p acționări ale butonului Air;
- dacă $C = 3$, două numere separate printr-un spațiu reprezentând nivelul maxim de umplere a unui balon după cele p acționări ale butonului Air și numărul de baloane aflate la acest nivel de umplere.

Restricții

- $C \in \{1, 2, 3\}$;
- $1 \leq n, m, d \leq 1\,000$;
- $1 \leq x \leq n$ și $1 \leq y \leq m$;
- $3 \leq k, p \leq 1\,000\,000$;
- $1 \leq b_{ij} \leq k$, pentru oricare $1 \leq i \leq n$ și $1 \leq j \leq m$;

#	Puncte	Restricții
1	30	$C = 1$
2	35	$C = 2$
3	35	$C = 3$

Exemple

balon.in	balon.out	Explicații
1 4 6 5 2 1 2 3 3 2 1 1 1 4 3 2 2 3 1 1 1 5 3 2 4 2 4 2 2 2 1 3 3 5 3	13	C = 1, n = 4, m = 6, k = 5 și p = 2. La prima acționare a pistonului se consumă 9 unități de volum de aer corespunzătoare elementelor marcate cu galben în figura din enunț. La a doua acționare a pistonului se consumă 4 unități de volum de aer corespunzătoare elementelor marcate cu roz în figura din enunț, deoarece doar patru dintre baloane sunt în interiorul pătratului de latură 3 cu colțul din stânga sus în poziția (3, 5).

2 4 6 5 3 1 2 3 3 2 1 1 1 4 3 2 2 3 1 1 1 5 3 2 4 2 4 2 2 2 1 3 3 5 3 3 2 4	2	C = 2, n = 4, m = 6, k = 5 și p = 3. După prima acționare corespunzătoare tripletei x=2, y=1, d=3 nivelurile de umplere devin: 1 2 3 3 2 1 2 2 5 3 2 2 4 2 2 1 5 3 3 5 3 4 2 2 și nu s-a spart niciun balon. După a doua acționare corespunzătoare tripletei x=3, y=5, d=3 nivelurile de umplere devin: 1 2 3 3 2 1 2 2 5 3 2 2 4 2 2 1 1 4 3 5 3 4 3 3 și s-a spart balonul din poziția x=3, y=5, fiind înlocuit automat cu un balon nou. După a treia acționare corespunzătoare tripletei x=3, y=2, d=4 nivelurile de umplere devin: 1 2 3 3 2 1 2 2 5 3 2 2 4 3 3 2 2 4 3 1 4 5 4 3 și s-a spart balonul din poziția x=4, y=2, fiind înlocuit automat cu un balon nou. În total s-au spart 2 baloane.
3 4 6 5 3 1 2 3 3 2 1 1 1 4 3 2 2 3 1 1 1 5 3 2 4 2 4 2 2 2 1 3 3 5 3 3 2 4	5 2	C = 3, n = 4, m = 6, k = 5 și p = 3. După cele trei acționări, nivelurile de umplere au devenit: 2 2 5 3 2 2 4 3 3 2 2 4 3 1 4 5 4 3 Nivelul maxim de umplere a unui balon este 5 și sunt două baloane cu acest nivel de umplere.

6.2 Rezolvarea problemei Balon

Cerința 1

Pentru rezolvarea primei cerințe determinăm pentru fiecare acționare a butonului Air descrisă de tripleta de valori (x,y,d) numărul de baloane în care intră câte o unitate de volum de aer utilizând formula:

$$(\min(x+d-1,n)-x+1)\cdot(\min(y+d-1,m)-y+1)$$

și îl adunăm la volumul total.

Cerința 2

Pentru fiecare triplet de valori (x,y,d) marcăm într-o matrice nouă (a) colțurile pătratului/ dreptunghiului care include baloanele în care se introduce aer. În matricea nou creată determinăm, utilizând sume parțiale, fiecare valoare a_{ij} ca fiind numărul total de unități de aer care au fost introduse prin dispozitivul aflat în celula din linia i și coloana j de pe placă. Adunăm la numărul de baloane sparte valoarea $[a_{ij}/k]$. Dacă adunând la nivelul inițial de umplere (b_{ij}) restul împărțirii lui a_{ij} la k se depășește nivelul maxim de umplere, numărul de baloane sparte crește cu o unitate.

Cerința 3

Construim matricea (a) conform descrierii de la cerința 2. Actualizăm nivelul de umplere al fiecărui balon b_{ij} adunând restul împărțirii lui a_{ij} la k, iar dacă b_{ij} depășește valoarea maximă de umplere, vom scădea din b_{ij} valoarea k. Determinăm apoi frecvența fiecărui nivel de umplere posibil și afișăm cel mai mare nivel de umplere și frecvența acestuia.

6.3 Cod-sursă pentru problema Balon

```
#include <bits/stdc++.h>
#define ull unsigned long long
#define nmaxi 1002
#define Nmaxi 1000020
using namespace std;
ifstream in("balon.in");
ofstream out("balon.out");
int c,n,m,k,d,x,y,p,b[nmaxi][nmaxi],a[nmaxi][nmaxi],s[Nmaxi];
ull vol,bb;
int main()
{
 in>>c;
 in>>n>>m>>k>>p;
 for (int i=1; i<=n; ++i)
     for (int j=1; j<=m; ++j)
          in>>b[i][j];
 switch(c)
 {
    case 1:
    {
        for (int i=1; i<=p; ++i)
        {   int xx, yy;
            in>>x>>y>>d;
            xx=x+d-1;
            yy=y+d-1;
            if(xx>n)xx=n;
            if(yy>m)yy=m;
            vol+=1ULL*(xx-x+1)*(yy-y+1);
        }
        out<<vol<<'\n';
        break;
    }
    case 2:
    {
        for (int l=1; l<=p; ++l)
        {   int xx, yy;
            in>>x>>y>>d;
            a[x][y]++;
```

```
            xx=x+d-1;
            yy=y+d-1;
            if (xx>n) xx=n;
            if (yy>m) yy=m;
            a[x][yy+1]--;
            a[xx+1][y]--;
            a[xx+1][yy+1]++;
        }
        for (int i=1; i<=n; ++i)
            for (int j=1; j<=m; ++j)
                a[i][j]+=a[i-1][j]+a[i][j-1]-a[i-1][j-1];
        for (int i=1; i<=n; ++i)
            for (int j=1; j<=m; ++j)
            {
                bb+=a[i][j]/k;
                if (b[i][j]+a[i][j]%k>k) bb++;
            }
        out<<bb<<'\n';
        break;
    }
    case 3:
    {
        for (int l=1; l<=p; ++l)
        {
            int xx, yy;
            in>>x>>y>>d;
            a[x][y]++;
            xx=x+d-1;
            yy=y+d-1;
            if (xx>n) xx=n;
            if (yy>m) yy=m;
            a[x][yy+1]--;
            a[xx+1][y]--;
            a[xx+1][yy+1]++;
        }
        for (int i=1; i<=n; ++i)
            for (int j=1; j<=m; ++j)
                a[i][j]+=a[i-1][j]+a[i][j-1]-a[i-1][j-1];
        for (int i=1; i<=n; ++i)
            for (int j=1; j<=m; ++j)
            {
                b[i][j]+=a[i][j]%k;
                if (b[i][j]>k) b[i][j]-=k;
                s[b[i][j]]++;
            }
        int nivMax=k;
        while (nivMax&&!s[nivMax]) nivMax--;
        out<<nivMax<<' '<<s[nivMax]<<'\n';
    }
 }
 return 0;
}
```

6.4 Problema Magictrick

Propusă de: stud. Mihnea-Vicențiu Bucă, Facultatea de Matematică-Informatică, Universitatea București

Richard a pregătit un truc magic pentru a o impresiona pe Dara. Pentru a pune acest truc magic în practică Richard a cumpărat un pachet de N cărți pe spatele cărora este scris câte un număr natural nenul.

Totuși Richard consideră că pachetul de cărți nu este suficient de bun pentru trucul lui magic. El se decide să aleagă un set, de cel puțin două cărți, din pachetul cumpărat astfel încât acesta să aibă *coeficientul magic* cât mai mare.

Coeficientul magic al unui set de cărți reprezintă produsul dintre suma numerelor scrise pe cărțile respective și cel mai mare divizor comun al acestor numere. De exemplu, pentru setul de cărți care au inscripționate numerele $\{2,\ 3,\ 6,\ 7,\ 8\}$ *coeficientul magic* maxim este 32 și se obține pentru setul de cărți având numerele $\{2, 6, 8\}$ (Vezi exemplul 2).

Cerințe

Fiind date numerele scrise pe cele N cărți din pachet, să se determine:

1. *coeficientul magic* al întregului pachet de cărți;
2. *coeficientul magic* maxim, alegând din pachet un set de cel puțin două cărți.

Date de intrare

Fișierul de intrare `magictrick.in`, va conține pe prima linie numerele naturale C și N, unde C reprezintă cerința care trebuie rezolvată (1 sau 2), iar N reprezintă numărul de cărți cumpărate de Richard. Pe următoarea linie fișierul conține N valori, reprezentând numerele ce sunt scrise pe spatele fiecărei cărți. Numerele care se găsesc pe aceeași linie a fișierului sunt separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire `magictrick.out` conține fie doar răspunsul pentru cerința 1 (dacă $C = 1$), fie doar răspunsul pentru cerința 2 (dacă $C = 2$).

Restricții

- $C \in \{1, 2\}$
- $2 \leq N \leq 100\,000$
- numerele scrise pe cele N cărți au valori cuprinse între $[1,\ 500\,000]$.

#	Puncte	Restricții
1	20	$C = 1$
2	9	$C = 2$, $N \leq 1\,000$, iar fiecare număr scris pe cele N cărți aparține $[1,\ 6]$
3	11	$C = 2$, $N \leq 1\,000$, iar fiecare număr scris pe cele N cărți aparține $[1, 1\,000]$
4	13	$C = 2$, iar numerele scrise pe cele N cărți sunt puteri ale aceluiași număr din intervalul $[2, 6]$
5	47	fără restricții suplimentare.

Exemple

magictrick.in	magictrick.out	Explicații
1 5 1 2 3 4 5	15	*Coeficientul magic* al pachetului este: $(1+2+3+4+5) \cdot cmmdc(1,2,3,4,5) = 15 \cdot 1 = 15$
2 5 2 3 6 7 8	32	*Coeficientul magic maxim* este: $cmmdc(2,6,8) \cdot (2+6+8) = 2 \cdot 16 = 32$

6.5 Rezolvarea problemei Magictrick

Cerința 1

Pentru rezolvarea primei cerințe calculăm *cmmdc*-ul și suma celor N numere și afișăm produsul celor două valori obținute.

Cerința 2

Soluția 1 - complexitate $O(N \cdot V_{MAX})$ - 20 de puncte:

Având în vedere cea de-a treia restricție a problemei, pentru fiecare i de la 1 la V_{MAX}, unde V_{MAX} reprezintă valoare maximă inscripționată pe spatele unei cărți, determinăm suma și numărul valorilor care sunt divizibile cu i. Răspunsul va fi produsul maxim dintre i și suma valorilor divizibile cu i, dacă există cel puțin doi multipli de i.

Soluția 2 - complexitate $O(N \cdot \sqrt{V_{MAX}})$ - 63 de puncte.

Se observă că singurele numere care pot contribui la răspuns sunt divizorii valorilor inscripționate pe cărțile din pachet. Vom parcurge fiecare valoare din pachetul de cărți, îi aflăm divizorii, iar pentru fiecare divizor vom contoriza câte valori din pachet sunt multipli ai acestui divizor, determinând și suma acestor valori.

Soluția 3 - complexitate $O(V_{MAX} \log(V_{MAX}))$ - 80 de puncte.

Pentru fiecare număr de la 1 la V_{MAX} vom număra și însuma multiplii acestuia inscripționați pe cărțile din pachetul inițial.

6.6 Cod-sursă pentru problema Magictrick

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

int cnt[500005], H[500005];
int c, n, x, d;

long long sumv[500005];
long long rez, sum, cmmdc = -1;

int main() {
    fin >> c >> n;
    /* cerinta 1 */
    if (c == 1) {
        for (int i = 0; i < n; ++i) {
```

```
            fin >> x;
            sum += x;
            if (cmmdc == -1) cmmdc = x;
            else {
                /* algoritmul lui euclid */
                int a = cmmdc, b = x, c;
                while (b) {
                    c = a % b;
                    a = b, b = c;
                }
                cmmdc = a;
            }
        }
        fout << cmmdc * sum;
        return 0;
    }
    /* cerinta 2 */
    for (int i = 0; i < n; ++i) {
        fin >> x;
        ++H[x];
    }
    for (int i = 1; i <= 500000; ++i)
        for (int j = i; j <= 500000; j += i) {
            cnt[i] += H[j];
            sumv[i] += (long long)H[j] * j;
        }
    /* gaseste rezultat */
    for (int i = 1; i <= 500000; ++i) {
        if (cnt[i] >= 2 and sumv[i] * i > rez)  {
            rez = sumv[i] * i;
        }
    }
    fout << rez<<'\n';
    return 0;
}
```

6.7 Problema Puteri3

Propusă de: prof. Daniel Popa, Liceul Teoretic „Aurel Vlaicu" Orăștie

Lui Scortzy îi plac foarte mult bilele și puterile lui 3, astfel și-a organizat colecția de bile în cutii, după următoarea regulă: în prima cutie a pus o bilă, în a doua cutie 3 bile, în a treia cutie 9 bile, apoi $27, 81, 243$ ...ș.a.m.d. Privind linia lungă de cutii Scortzy și-a pus întrebarea: *Ce număr de bile poate obține folosind bilele din cutii, fără a le scoate din cutie?*
Pentru a răspunde întrebării a început să formeze numerele: 0 (nici o cutie), 1 (cutia 1), 3 (cutia 2), 4 (cutiile 1 și 2), 9 (cutia 3) ... ș.a.m.d., obținând șirul lui Scortzy, primii termeni ai acestui șir fiind: $0, 1, 3, 4, 9, 10, 12, 13, 27, 28, 30, 31, 36, 37$.

Plăcându-i noul șir obținut Scortzy dorește să rezolve următoarele probleme:

Cerințe

1. Citind un număr natural n, determină câte cutii au mai puțin de n bile în ele.
2. Citind un număr natural n urmat de n valori naturale $x_1, x_2, \ldots, x_n$, determină câte bile sunt, în fiecare dintre cutiile utilizate, pentru a obține cel de-al x_i-lea număr din șirul lui Scortzy.

Date de intrare

Pe prima linie a fișierului `puteri3.in` se află numerele naturale c și n, separate printr-un spațiu. Dacă $c = 2$ atunci pe următoarele n linii se vor găsi n valori naturale $x_1, x_2, \ldots, x_n$, câte una pe linie, ce reprezintă pozițiile din șirul lui Scortzy.

Date de ieșire

Fișierul de ieșire `puteri3.out` va conține

- dacă $c = 1$, un singur număr care reprezintă soluția cerinței 1;
- dacă $c = 2$, pe fiecare dintre cele n linii ale sale unul sau mai multe numere. Pe linia i a fișierului se vor afla unul sau mai multe numere, separate prin câte un spațiu, în ordine crescătoare, ce reprezintă numărul de bile din fiecare cutie folosită pentru a obține numărul de pe poziția x_i din șirul lui Scortzy.

Restricții

- $C \in \{1, 2\}$;
- $1 \leq x_1, x_2, \ldots, x_n \leq 10^{18}$
- Pentru $c = 1, 1 \leq n \leq 10^{18}$
- Pentru $c = 2, 1 \leq n \leq 1\,000$, numărul de bile dintr-o cutie nu are mai mult de 80 cifre.

#	Puncte	Restricții
1	20	$c = 1$
2	30	$c = 2$, $1 \leq x_1, x_2, \ldots, x_n \leq 1\,000$
3	35	$c = 2$, numărul de bile dintr-o cutie nu este mai mare decât 10^{18}
4	15	$c = 2$, fără restricții suplimentare

Exemple

puteri3.in	puteri3.out	Explicații
1 100	5	Cutiile cu 1, 3, 9, 27 și 81 bile au mai puțin de 100 de bile în ele
2 3 4 14 9	1 3 1 9 27 27	Primii termeni ai șirului lui Scortzy sunt: 0, 1, 3, 4, 9, 10, 12, 13, 27, 28, 30, 31, 36, 37. Termenul de pe poziția 4 are valoarea 4 și se obține din suma 1+3. Termenul de pe poziția 14 are valoarea 37 și se obține din suma 1+9+27. Termenul de pe poziția 9 are valoarea 27 și se obține folosind cutia ce conține 27 bile

6.8 Rezolvarea problemei Puteri3

Cerința 1

Rezultatul se obține prin numărarea puterilor lui 3 mai mici decât n-ul citit.

Cerința 2

Se observă că pentru a forma numerele din șirul lui Scortzy se folosesc valorile (puteri ale lui 3) conform tabelului de mai jos unde x-ul marchează termenii puteri ale lui 3 care sunt folosiți.

Poziție:	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Șir:	0	1	3	4	9	10	12	13	27	28	30	31	36	37
1		x		x		x		x		x		x		x
3			x	x			x	x			x	x		
9					x	x	x	x					x	x
27									x	x	x	x	x	x
binar	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101

Dacă șirurile de x-uri, pe verticală, sunt considerate ca fiind numere în baza 2 se observă că diferența între poziția elementului în șirul lui Scortzy și numărul transformat din binar în zecimal este 1. De aici rezultă și soluția: pentru a obține un elementul de pe o *pozitie* dată din șirul lui Scortzy se scade din *pozitie* valoarea 1 și se transformă în baza 2 numărul obținut. Dacă numărul are cifra i din reprezentarea în baza 2 are valoarea 1 atunci se folosește numărul 3^i în determinarea termenului de pe poziția *pozitie*. Pentru a mări viteza de calcul se calculează la început toate puterile lui 3 până la valoarea maximă acceptată.

6.9 Cod-sursă pentru problema Puteri3

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
ifstream fin("puteri3.in");
ofstream fout("puteri3.out");
int a[100][100]; // a[i]-3^i, a[i][0]-cate cifre are numarul i
unsigned long long n,i,x;
int c, k, j, t;
```

```
int main()
{
    /// precalcul
    a[0][0]=a[0][1]=1;// 3^0 e 1 ai are o cifra
    for(i=1; i<85; i++)/// calculez 3^i
    {
        t = 0; ///transportul e 0
        a[i][0] = a[i-1][0]; /// va avea cel putin acelasi numar de cifre
        for(j=1; j<=a[i-1][0]; j++)
        {
            a[i][j]=a[i-1][j]*3+t;
            t=a[i][j]/10;           // calculez transportul
            a[i][j]%=10;            // pe loc ramane ultima cifra
        }
        if(t>0) // inseamna ca va creste numarul de cifre
        {
            a[i][++a[i][0]]=t;
        }
    }
    fin >> c >> n;
    if(c==1)
    {
        k=0;
        for(i=1; i<n; i*=3)k++;
        fout << k << "\n";
    }
    else
    {
        for(i=1; i<=n; i++)
        {
            fin>>x;
            x--;
            if(x==0)fout<<"0";
            else
            {
                for(j=0; x>0; j++, x/=2)
                    if(x%2==1)
                      {
                          for(k=a[j][0];k>0;k--)fout<<a[j][k];
                          fout << " ";
                      }
            }
            fout<<"\n";
        }
    }
 return 0;
}
```

Capitolul 7

ONI 2023, clasa a VII-a

7.1 Problema Dominew

Propusă de: prof. Flavius Boian, Colegiul Național „Spiru Haret" Târgu Jiu

Pentru că se plictisește și este foarte inteligent, Radu l-a rugat pe prietenul lui, savantul Feder, să creeze o activitate care să-i pună mintea la încercare. Savantul Feder a adus N piese dreptunghiulare pe care sunt scrise numere naturale și le-a așezat pe masă în ordinea crescătoare a valorilor scrise pe ele, pe poziții consecutive, una lângă cealaltă. Apoi îi dă lui Radu, una câte una, alte M piese dreptunghiulare, pe care sunt scrise numere naturale, într-o ordine oarecare. Când Radu primește o piesă el trebuie să o așeze în șirul de pe masă pe **cea mai mică poziție posibilă**, astfel încât piesele din șir să rămână în ordine crescătoare. Evident, șirul de pe masă se modifică pe măsură ce Radu așază piesele în șir.

Cerințe

Cunoscând șirul pieselor de pe masă, în ordinea în care sunt așezate, precum și cele M piese pe care le primește succesiv Radu, scrieți un program care să afișeze pentru fiecare dintre cele M piese poziția pe care aceasta este așezată în șir.

Date de intrare

Fișierul de intrare `dominew.in` conține pe prima linie numărul natural N. Pe a doua linie se află N numere naturale, în ordine crescătoare, reprezentând valorile pieselor din șirul aflat inițial pe masă. Pe a treia linie se află numărul natural M. Pe cea de-a patra linie sunt M numere naturale, reprezentând valorile pieselor pe care le primește Radu, în ordinea în care acesta le primește. Numerele scrise pe aceeași linie sunt separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire `dominew.out` va conține o singură linie pe care vor fi scrise M valori separate prin câte un spațiu, cea de a i-a valoare fiind poziția pe care este așezată în șirul de pe masă cea de a i-a piesă primită de Radu ($1 \leq i \leq M$).

Restricții

- $2 \leq N \leq 1\,000\,000$
- $1 \leq M \leq 8\,000$

- Valorile scrise pe piese sunt numere naturale $\leq 10^9$.
- Pozițiile elementelor din șirul de pe masă sunt numerotate începând cu 1.

#	Puncte	Restricții
1	8	$M = 1$
2	14	$1 < N, M \leq 100$
3	8	$100 < N \leq 50\,000$, $100 < M \leq 250$
4	24	$N \leq 100\,000$, $250 < M \leq 1\,000$
5	46	$N \leq 1\,000\,000$, $1\,000 < M \leq 8\,000$

Exemple

dominew.in	dominew.out
6 2 5 5 9 10 11 3 5 1 12	2 1 9
14 2 2 2 4 7 8 9 10 12 16 20 21 23 24 7 18 7 20 1 16 25 23	11 5 13 1 12 20 18

Explicații

Exemplul 1
Inițial pe masă se află $N = 6$ piese, în ordine crescătoare. 2 5 5 9 10 11 Radu primește $M = 3$ piese. Prima piesă are valoarea 5 și va fi așezată în șirul de pe masă pe poziția 2. Șirul va deveni 2 5 5 5 9 10 11. A doua piesă are valoarea 1, va fi așezată în șirul de pe masă pe poziția 1. Șirul va deveni 1 2 5 5 5 9 10 11. A treia piesă are valoarea 12 și va fi așezată pe poziția 9 în șirul de pe masă. Șirul va deveni 1 2 5 5 5 9 10 11 12.

Exemplul 2
După primele 3 inserări, șirul va fi 2 2 2 4 7 7 8 9 10 12 16 18 20 20 21 23 24, valoarea 20 ajungând pe poziția 13.

7.2 Rezolvarea problemei Dominew

La citire se rețin elementele în 2 vectori distincți (primul va avea N elemente, iar cel de-al doilea M elemente).

În primul rând, se poate observa că numărul maxim de actualizări este mult mai mic decât numărul de valori inițiale ($M \leq 8\,000$), mult mai mic decât valoarea maximă a lui N, ceea ce ne duce cu gândul la a procesa separat cei doi vectori pentru a obține răspunsul într-un timp optim.

Pe măsură ce se citesc elementele din cel de-al doilea vector se identifică poziția pe care ar trebui inserate acestea în vectorul sortat până la acel moment. Vom nota valoarea pe care o verificăm la un pas cu x. Pentru fiecare element din șirul de M valori, va trebui mai întâi să căutăm binar în vectorul cu valorile inițiale poziția în care se află cea mai mare valoare mai mică decât x. Să notăm această poziție cu a. Apoi, vom afla și câte valori din vectorul de lungime M situate

înaintea valorii curente sunt mai mici decât x. Datorită valorii mici a lui M, putem face acest lucru verificând fiecare poziție anterioară poziției curente. Să notăm acest răspuns cu b.

Astfel, răspunsul corespunzător pentru o valoare citită va fi $a + b + 1$.

Soluția are complexitatea totală $O(M^2 + M \cdot \log N)$.

Soluție alternativă

Se vor citi mai întâi valorile și se vor sorta valorile din vectorul de lungime M. Apoi, vom aplica un algoritm de interclasare pentru a răspunde în manieră offline la toate întrebările, memorând mai întâi poziția unde se găseau acele actualizări în vectorul al doilea, având grijă să prioritizăm valoarea din vectorul al doilea în caz de egalitate, pentru a respecta restricția din enunț.

Această soluție are complexitatea totală $O(M^2 + M + N)$.

7.3 Cod-sursă pentru problema Dominew

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

ifstream f("dominew.in");
ofstream g("dominew.out");

int n, m, i, j, x, st, dr, mij, poz, t;
int a[1000001], b[8001];

int main()
{
    f>>n;
    for (i=1; i<=n; i++) f>>a[i];
    f>>m;
    for (i=1; i<=m; i++)
    {
        f>>b[i];
        st=1; dr=n;
        poz=0;
        while (st<=dr)
        {
            mij=(st+dr)/2;
            if (a[mij]<b[i])
            {
                poz=mij;
                st=mij+1;
            }
            else
                dr=mij-1;
        }
        ///caut liniar pozitia pe care trebuie sa inserez in noul vector
        for (t=1; t<i; t++)
            if (b[i]>b[t])
                poz++;
        g<<poz+1<<" ";
    }
    return 0;
}
```

7.4 Problema Pix

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

Robotul Vasile s-a angajat la un depozit de pixuri. Aici pixurile sunt ambalate în cutii. Există N tipuri de cutii; într-o cutie de tipul i ($1 \leq i \leq N$) sunt ambalate exact nr_i pixuri ($nr_1 \leq nr_2 \leq \cdots \leq nr_N$). În depozit există un număr atât de mare de cutii de fiecare tip încât Vasile poate utiliza oricâte cutii dorește, de orice tip. Sarcina robotului Vasile este să livreze pixurile comandate de diferite firme de birotică. El nu știe câte pixuri va avea de livrat la următoarea comandă, dar știe că vor fi cel mult $Vmax$ pixuri. Ca urmare, pentru a fi eficient, robotul Vasile vrea să își pregătească în camera de livrare un număr minim de cutii de pixuri astfel încât să poată livra orice număr de pixuri cuprins între 1 și $Vmax$ folosind cutiile pregătite, evident, **fără a deschide cutiile**.

Cerințe

Scrieți un program care citește valorile N, $nr_1, nr_2, \ldots nr_N$ și $Vmax$ și determină numărul minim de cutii pe care robotul Vasile trebuie să le pregătească în camera de livrare astfel încât să poată livra orice număr de pixuri cuprins între 1 și $Vmax$, fără a deschide nicio cutie.

Date de intrare

Fișierul de intrare `pix.in` conține pe prima linie numărul natural N reprezentând numărul de tipuri de cutii. Pe a doua linie se află N numere naturale în ordine crescătoare, separate prin câte un spațiu, $nr_1, nr_2, \ldots nr_N$ reprezentând numărul pixuri ambalate în fiecare tip de cutie. Pe a treia linie se află numărul natural $Vmax$ cu semnificația din enunț.

Date de ieșire

Fișierul de ieșire `pix.out` va conține o singură linie pe care va fi scris un număr natural reprezentând numărul minim de cutii pe care robotul Vasile trebuie să le pregătească în camera de livrare astfel încât să poată livra orice număr de pixuri cuprins între 1 și $Vmax$.

Restricții

- $1 \leq N \leq 100\,000$
- $1 \leq Vmax, nr_i \leq 10^{12}$, pentru $1 \leq i \leq N$
- Se garantează că pentru toate datele de test există soluție.

#	Puncte	Restricții
1	20	$1 \leq N < 15$
2	10	$15 \leq N < 600$
3	40	$Vmax \leq 100\,000$

Exemple

pix.in	pix.out	Explicații
4 1 2 3 5 15	5	Numărul minim de cutii pe care trebuie să le pregătească Vasile este 5 (o cutie de tipul 1, două de tipul 2 și două de tipul 4, numărul de pixuri din aceste cutii fiind 1, 2, 2, 5, 5) El poate astfel livra orice număr de pixuri între 1 și 15, o modalitate posibilă fiind: 1: cutia de tip 1 2: o cutie de tip 2 3: o cutie de tip 1 și o cutie de tip 2 4: două cutii de tip 2 5: o cutie de tip 4 6: o cutie de tip 1 și o cutie de tip 4 7: o cutie de tip 2 și o cutie de tip 4 8: câte o cutie de tipurile 1, 2, 4 9: o cutie de tip 4 și două cutii de tip 2 10: două cutii de tip 4 11: două cutii de tip 4 și o cutie de tip 1 12: două cutii de tip 4 și o cutie de tip 2 13: două cutii de tip 4, o cutie de tip 2 și o cutie de tip 1 14: două cutii de tip 4 și două cutii de tip 2 15: toate cutiile Aceasta nu este singura posibilitate de a alege un număr minim de cutii pentru a obține toate valorile de la 1 la 15.

7.5 Rezolvarea problemei Pix

Reținem într-un vector nr capacitățile distincte $\leq Vmax$.

Deoarece se garantează că există soluție, obligatoriu $nr[1] = 1$.

Vom lua o cutie de tipul 1.

Analizăm apoi $nr[2]$. Dacă $nr[2] > nr[1]$ atunci comenzile pentru capacitatea x din mulțimea $\{1, 2, ..., nr[2] - 1\}$ nu pot fi obținute decât din mai multe cutii cu capacitatea $nr[1] = 1$. Deci vom lua $nr[2] - 1$ cutii cu capacitatea $nr[1] = 1$.

Pentru a vedea câte cutii cu capacitatea $nr[2]$ luăm trebuie să analizăm $nr[3]$, pentru ca toate valorile până la $nr[3] - 1$ inclusiv să fi obținute.

Observăm că dacă am lua j cutii de capacitate $nr[2]$ am obține toate comenzile cu capacitatea x din mulțimea $\{1, 2, \ldots (j + 1) \cdot nr[2] - 1\}$. Să generalizăm acum.

Să notăm cu sum suma capacităților cutiilor luate până la pasul i (acestea reprezintă valoarea maximă pentru care avem soluție până la acest pas).

Câte cutii cu capacitatea $nr[i]$ vom lua?

Dacă $nr[i + 1] - 1 \leq sum$ nu are rost să luăm cutii cu capacitatea $nr[i]$.

Determinăm câte de cutii cu capacitatea $nr[i]$ sunt necesare astfel încât $nr[i+1]-1 \geq cate \cdot nr[i]+sum$.

Pentru ca formula să funcționeze și pentru $i=n$ vom inițializa componenta $n+1$ a vectorului nr cu $Vmax+1$, astfel că la pasul n vom asigura toate comenzile pentru capacitățile $x \leq Vmax$.

7.6 Cod-sursă pentru problema Pix

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

#define NMAX 100002

using namespace std;
ifstream fin("pix.in");
ofstream fout("pix.out");
int n, lg;
long long int nr[NMAX];
long long int a[NMAX];
long long int vmax, rez, sum, ultima, cate;

int main()
{int i;
 fin>>n;
 for (i=1; i<=n; i++) fin>>a[i];
 fin>>vmax;
 //sort(a+1,a+n+1);
 ///retin in nr doar valorile distincte <=vmax
 nr[1]=a[1]; lg=1;
 for (i=2; i<=n; i++)
      {if (a[i]==a[i-1]) continue;
       if (a[i]>vmax) break;
       nr[++lg]=a[i];
      }
 n=lg; nr[n+1]=vmax+1;
 rez=nr[2]-1; sum=nr[2]-1;
 for (i=2; i<=n; i++)
     {
       if (nr[i+1]-1<=sum) continue;
       cate=(nr[i+1]-1-sum)/nr[i];
       if (cate*nr[i]+sum<nr[i+1]-1) cate++;
       rez+=cate;
       sum+=cate*nr[i];
     }
 fout<<rez<<'\n';
 fout.close();
 return 0;
 }
```

7.7 Problema Secvmin

Propusă de: stud. Theodor-Gabriel Tulba-Lecu și Ioan-Cristian Pop, Universitatea Politehnica București

Fie un șir de n numere naturale v_1, v_2, ..., v_n, unde v_i reprezintă al i-lea număr din șir. O subsecvență $[x, y]$ a șirului v (cu $1 \leq x \leq y \leq n$) conține toate elementele $v_x, v_{x+1}, \ldots v_{y-1}, v_y$.

Cerințe

Fiind date două numere naturale n și k și un șir v de n numere naturale, scrieți un program care să răspundă la următoarea întrebare: câte subsecvențe conțin simultan cele mai mici k valori distincte din șir?

Date de intrare

Fișierul de intrare `secvmin.in` conține pe prima linie numerele n și k, iar pe cea de a doua linie se vor afla numerele naturale v_1, v_2, ...v_n, cu semnificația din enunț. Numerele scrise pe aceeași linie sunt separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire `secvmin.out` va conține o singură linie pe care va fi scris răspunsul la cerința problemei.

Restricții

- $1 \leq n \leq 1\,000\,000$
- $1 \leq k \leq 5$
- $1 \leq v_i \leq 1\,000\,000\,000$, pentru $1 \leq i \leq n$
- Se garantează că șirul va conține cel puțin k valori distincte.

#	Puncte	Restricții
1	23	$k = 1$
2	32	$k = 2$
3	45	$3 \leq k \leq 5$

Exemple

secvmin.in	secvmin.out	Explicații
7 1 1 3 2 2 1 3 2	19	$k = 1$. Cea mai mică valoare din șir este egală cu 1. Subsecvențele care conțin valoarea 1 sunt: [1 1], [1 2], [1 3], [1 4], [1 5], [1 6], [1 7], [2 5], [2 6], [2 7], [3 5], [3 6], [3 7], [4 5], [4 6], [4 7], [5 5], [5 6], [5 7].

7 2 1 5 2 2 1 5 2	15	$k = 2$. Cea mai mică valoare din șir este egală cu 1, iar a doua cea mai mică valoare din șir este egală cu 2. Subsecvențele care conțin valorile 1 și 2 sunt: [1 3], [1 4], [1 5], [1 6], [1 7], [2 5], [2 6], [2 7], [3 5], [3 6], [3 7], [4 5], [4 6], [4 7], [5 7].

7.8 Rezolvarea problemei Secvmin

Să considerăm cele mai mici valori distincte din șir $y_1, y_2, \ldots y_k$.

Pentru fiecare poziție i din șir vom calcula și vom actualiza valorile $last_1, last_2, \ldots last_k$ cu semnificația $last_j$ este cea mai mare poziție mai mică sau egală decât i în care a apărut valoarea y_j.

Se observă că o secvență care are capătul dreapta în poziția i va conține toate cele k valori minime dacă și numai dacă capătul stânga al secvenței este mai mic sau egal decât toate valorile $last_j$.

Pentru a obține soluția, la fiecare i actualizăm șirul $last$ și adunăm la soluție valoarea minimă a acestui șir.

Pentru a forma și actualiza șirul $last$ este necesar ca în prealabil să avem calculate cele k valori minime, dar aceasta se poate realiza încă de la citire prin inserarea în șirul y a celor mai mici k valori atunci când ele apar.

Șirul $last$ se inițializează cu 0 și se actualizează de fiecare dată când la o poziție i apare una dintre cele k valori minime. Soluția are complexitatea $N \cdot K$.

7.9 Cod-sursă pentru problema Secvmin

```
// prof. Adrian Panaete
#include <bits/stdc++.h>
using namespace std;
ifstream f("secvmin.in");
ofstream g("secvmin.out");
const int N = 1000010;
int n, k, x[N], y[10], last[10];
int64_t sol;
int main()
{
    f>>n>>k;
    for (int i=1; i<=k; i++) y[i]=1000000010;
    for (int i=1; i<=n; i++)
    {
        int val;
        f>>val;
        if (val<y[k])
        {
            int p=1;
            while (val>y[p]) p++;
            if (val<y[p])
```

```
            {
                int  q=k;
                while (q>p)
                {
                    y[q]=y[q-1];
                    q--;
                }
                y[p]=val;
            }
        }
        x[i]=val;
    }
    for (int i=1; i<=n; i++)
    {
        int cnt=n+1;
        for (int j=1; j<=k; j++)
        {
            if (x[i]==y[j]) last[j]=i;
            cnt=min(cnt,last[j]);
        }
        sol+=cnt;
    }
    g<<sol;
    return 0;
}
```

Capitolul 8

ONI 2023, clasa a VIII-a

8.1 Problema Castel

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

După ce a scăpat de SPÂN și a devenit împărat, HARAP-ALB a decis să-și construiască un nou castel în împărăția sa ce poate fi reprezentată cu ajutorul sistemului de coordonate carteziene. El știe că ROȘ-ÎMPĂRAT a construit $N + 1$ garduri dreptunghiulare, însă știe și că acesta este cam zgârcit și nu a folosit cele mai bune materiale.

HARAP-ALB a învățat din greșeli, iar acum încearcă să se ferească de pericole cât de mult poate. De aceea, el vrea să își amplaseze castelul într-un punct din sistemul cartezian care să se afle în interiorul a cel puțin N dintre cele $N + 1$ garduri.

Cerințe

Fiind date numărul natural nenul N și coordonatele celor $N + 1$ garduri (perechi de colțuri stânga-sus și dreapta-jos), să se determine (în cazul în care există) punctul **cel mai apropiat de originea sistemului de coordonate** unde HARAP-ALB își poate amplasa castelul astfel încât acesta să se afle în interiorul a cel puțin N garduri.

Date de intrare

Fișierul de intrare `castel.in` conține pe prima linie numărul natural nenul N, cu semnificația de mai sus. Următoarele $N + 1$ linii conțin câte patru numere naturale a_i, b_i, c_i, d_i ($1 \leq i \leq N + 1$), separate între ele prin câte un spațiu, reprezentând coordonatele gardurilor. Colțul din stânga-sus al gardului i se află în punctul de abscisă a_i și ordonată b_i, iar cel din dreapta-jos în punctul de abscisă c_i și ordonată d_i.

Date de ieșire

Fișierul de ieșire `castel.out` conține pe prima linie două numere naturale, reprezentând abscisa, respectiv ordonata punctului determinat conform cerinței. Dacă sunt mai multe astfel de puncte, se va alege **cel cu abscisa minimă**. Dacă nu există niciun astfel de punct, se va afișa mesajul **NU**.

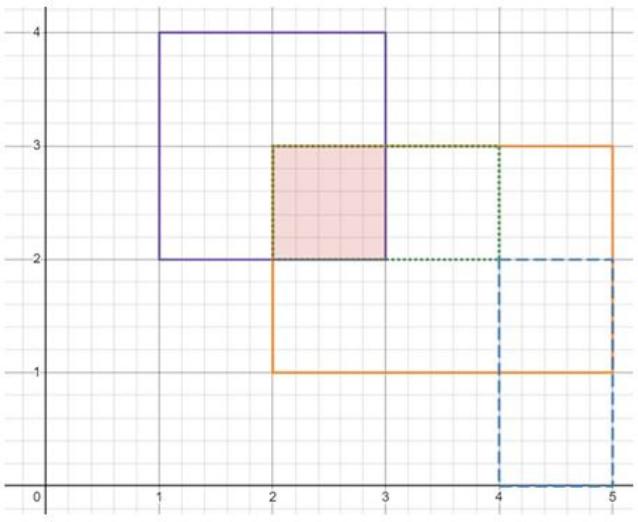
Restricții

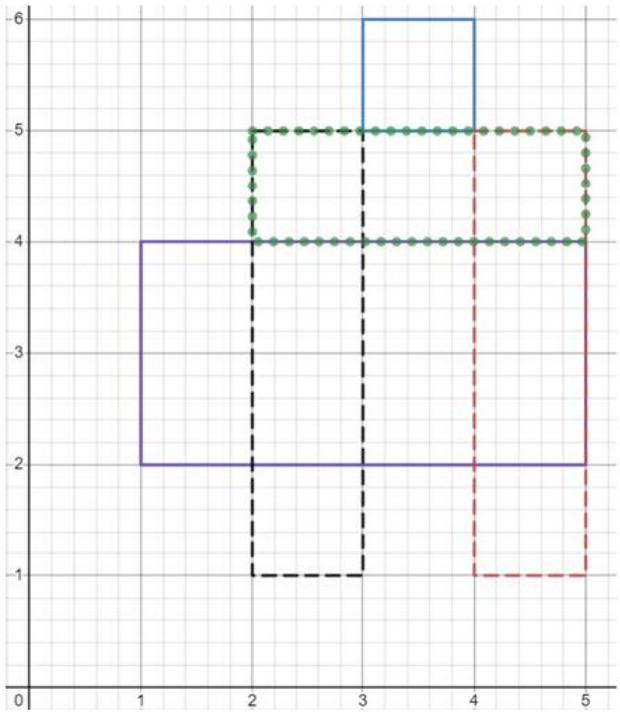
- $1 \leq N < 200\,000$.
- $0 \leq a_i, b_i, c_i, d_i < 100\,000$ și $a_i \leq c_i$, $d_i \leq b_i$, pentru fiecare i: $1 \leq i \leq N + 1$.
- **Laturile tuturor dreptunghiurilor sunt paralele cu axele de coordonate ale sistemului cartezian.**
- Originea sistemului de coordonate se află în punctul $(0, 0)$.
- Interiorul unui gard conține și gardul (conturul).
- Pot exista dreptunghiuri degenerate, adică segmente (paralele cu axele de coordonate) sau puncte.
- Pot exista dreptunghiuri identice.
- Distanța dintre două puncte situate la coordonatele (a, b), respectiv (c, d) este egală cu $\sqrt{(c-a)^2 + (d-b)^2}$.

#	Puncte	Restricții
1	19	$N < 200$ și $a_i, b_i, c_i, d_i < 200$, pentru fiecare i: $1 \leq i \leq N + 1$.
2	30	$N < 2\,000$
3	51	Nu există alte restricții suplimentare

Exemple

castel.in	castel.out	Explicații
3 1 4 3 2 4 2 5 0 2 3 4 2 2 3 5 1	2 2	Zona hașurată cu roșu reprezintă punctele ce se află în interiorul a cel puțin 3 dreptunghiuri. Punctul $(4, 2)$ respectă, de asemenea, cerința. Dintre toate acestea, $(2, 2)$ este cel mai apropiat de origine.
4 1 4 5 2 2 5 3 1 4 5 5 1 3 6 4 5 2 5 5 4	NU	Niciun punct nu se află în interiorul a cel puțin 4 garduri, așa cum se poate observa din cea de a doua diagramă de mai jos.





8.2 Rezolvarea problemei Castel

Problema se rezumă la a afla dacă, din $N + 1$ dreptunghiuri date, putem alege N care să aibă intersecția nevidă.

Subtask 1

Se folosește o matrice pentru a simula intersecțiile dreptunghiurilor. Astfel obținem complexitatea $\mathcal{O}(N \cdot VAL\_MAX^2)$, unde VAL_MAX este maximul coordonatelor dreptunghiurilor.

Subtask 2

Se elimină, pe rând, fiecare dreptunghi, și se face intersecția celor rămase. Complexitatea obținută este $\mathcal{O}(N^2)$.

Soluția completă

Se observă că intersecția a două dreptunghiuri ce au laturile paralele cu axele de coordonate este fie un dreptunghi, fie mulțimea vidă. Astfel, intersecția a două dreptunghiuri (x_1, y_1, x_2, y_2) și (x_3, y_3, x_4, y_4) este un dreptunghi (x_5, y_5, x_6, y_6) cu $x_5 = \max(x_1, x_3)$, $y_5 = \min(y_1, y_3)$, $x_6 = \min(x_2, x_4)$, $y_6 = \max(y_2, y_4)$. Dacă $x_5 > x_6$ sau $y_6 > y_5$, intersecția celor două dreptunghiuri este, de fapt, mulțimea vidă.

Observăm că intersecția dreptunghiurilor este comutativă și asociativă. Construim doi vectori auxiliari pre și suf astfel: $pre[i]$ este egal cu intersecția dreptunghiurilor $1, 2, \ldots i$, iar $suf[i]$ este egal cu intersecția dreptunghiurilor $i, i+1, \ldots N+1$. Dacă v_i este al i-lea dreptunghi, $pre[1] = v_1$ și $pre[i]$ este egal cu intersecția dintre $pre[i-1]$ și v_i pentru $i > 1$. Analog se calculează suf.

Așadar, pentru a găsi punctele ce aparțin intersecției a cel puțin N dreptunghiuri excludem pe rând câte un dreptunghi și calculăm, în timp constant, intersecția celor rămase. Pentru $1 < i < N + 1$, intersecția obținută prin eliminarea dreptunghiului v_i este egală cu intersecția dintre $pre[i-1]$ și $suf[i+1]$. Se procedează asemănător și pentru $i = 1$ și $i = N + 1$. Pentru

fiecare intersecție nevidă găsită alegem punctul din stânga-jos al dreptunghiului obținut (deoarece toate coordonatele sunt pozitive, deci acela este cel mai apropiat de origine) și comparăm cu un minim global.

Complexitatea obținută este $\mathcal{O}(N)$.

Soluție alternativă

O altă soluție, tot în $\mathcal{O}(N)$, pornește de la următorul raționament. Intersecția tuturor dreptunghiurilor, dacă există, este un dreptunghi de coordonate $(a, b) - (c, d)$. În particular,

$$a = \max\{a_k | 1 \leq k \leq N + 1\}$$

Cum se modifică această coordonată dacă eliminăm al i-lea dreptunghi? Dacă $a_i \neq a$, atunci a nu se modifică. În schimb, dacă $a_i = a$, atunci a capătă valoarea celui de-al doilea maxim al mulțimii $\{a_k\}$. Similar raționăm și pentru ceilalți trei parametri ai intersecției, b, c și d.

Algoritmul calculează primele două maxime sau minime pentru fiecare dintre cei patru parametri: $\max\{a_k\}$, $\min\{b_k\}$, $\min\{c_k\}$, $\max\{d_k\}$. Apoi elimină fiecare dreptunghi pe rând și determină în $\mathcal{O}(1)$ intersecția (vidă sau nu).

8.3 Cod-sursă pentru problema Castel

```
#include <bits/stdc++.h>
const int nmax = 5 + 1e6;
using namespace std;
ifstream f("castel.in");
ofstream g("castel.out");
long long result = LLONG_MAX;
int n, rx, ry;

void check(int x, int y)
{
    static long long d;
    d = 1ll * x * x + 1ll * y * y;
    if (d < result || (d == result && x < rx))
        rx = x, ry = y, result = d;
}

struct rectangle
{
    int x1, y1, x2, y2;
} v[nmax], pre[nmax], suf[nmax], e{-2, -2, -2, -2}, z{-1, -1, -1, -1};

rectangle intersection(const rectangle &a, const rectangle &b)
{
    if (a.x1 + a.x2 + a.y1 + a.y2 == -8)
        return b;
    if (b.x1 + b.x2 + b.y1 + b.y2 == -8)
        return a;
    static int nx1, ny1, nx2, ny2;
    static rectangle r;
    nx1 = max(a.x1, b.x1);
    ny1 = min(a.y1, b.y1);
    nx2 = min(a.x2, b.x2);
    ny2 = max(a.y2, b.y2);
    if (nx1 > nx2 || ny2 > ny1)
```

```
        return z; // multimea vida
    r = {nx1, ny1, nx2, ny2};
    return r;
}

int main()
{
    f >> n;
    pre[0] = pre[n + 2] = suf[0] = suf[n + 2] = {-2, -2, -2, -2}; // element neutru
    for (int i = 1; i <= n + 1; i++)
    {
        f >> v[i].x1 >> v[i].y1 >> v[i].x2 >> v[i].y2;
        pre[i] = intersection(pre[i - 1], v[i]);
    }
    for (int i = n + 1; i >= 1; i--)
        suf[i] = intersection(suf[i + 1], v[i]);

    for (int i = 1; i <= n + 1; i++)
    {
        static rectangle r;
        r = intersection(pre[i - 1], suf[i + 1]);
        if (r.x1 + r.y1 + r.x2 + r.y2 != -4) // daca obtinem o multime nevida
            check(r.x1, r.y2);
    }
    if (result != LLONG_MAX)
        g << rx << " " << ry;
    else
        g << "NU";
    return 0;
}
```

8.4 Problema Kth

Propusă de: stud. Andrei Onuț, Universitatea Yale, S.U.A.

Se dă un șir V ce conține N numere întregi numerotate începând de la 1: $V_1, V_2, \ldots, V_N$ și două numere naturale nenule K și L, cu proprietatea că: $1 \leq K \leq L \leq N$. MIHAI studiază doar secvențele de lungime L, adică secvențele formate din exact L elemente situate pe poziții alăturate în acest șir V.

El își poate pune următoarea întrebare: „Dacă aș rearanja, în ordine **crescătoare**, elementele secvenței de lungime L care începe la poziția *poz* în șirul V, ce **valoare** s-ar afla pe poziția a K-a în cadrul secvenței rezultate?". Pentru secvența din șir care începe la poziția *poz* și are L elemente, adică $V_{poz}, V_{poz+1}, \ldots, V_{poz+L-1}$, valoarea elementului de pe poziția a K-a în cadrul secvenței este $V_{poz+K-1}$.

Cerințe

Ajutați-l pe MIHAI să afle care este răspunsul corect pentru Q întrebări de tipul descris mai sus!

Date de intrare

Pe prima linie a fișierului de intrare `kth.in` se află trei numere naturale nenule N, K și L, separate între ele prin câte un spațiu, cu semnificațiile de mai sus. Pe următoarea linie se află, separate între ele prin câte un spațiu, N numere întregi, reprezentând, în ordine, elementele șirului V. Pe următoarea linie se află numărul natural nenul Q, reprezentând numărul de întrebări formulate de către MIHAI. Pe fiecare dintre următoarele Q linii se află câte un număr natural nenul *poz*, reprezentând poziția de început a secvenței de L elemente pentru care se pune întrebarea respectivă.

Date de ieșire

Fișierul de ieșire `kth.out` va conține Q linii. Pe linia i se va afla un număr întreg ce reprezintă răspunsul la întrebarea i, în ordinea dată în fișierul de intrare, pentru fiecare i: $1 \leq i \leq Q$.

Restricții

- $2 \leq N \leq 300\,000$ și $1 \leq Q \leq 300\,000$.
- $-50\,000 \leq V_i \leq 50\,000$, pentru fiecare i: $1 \leq i \leq N$.
- $1 \leq poz \leq N - L + 1$, pentru fiecare dintre cele Q întrebări.
- Valorile *poz* din cadrul celor Q întrebări nu sunt neapărat distincte între ele oricare două.

#	Puncte	Restricții
1	7	$Q = 1$ și $V_1 = V_2 = \ldots = V_N$, adică elementele din șirul V au aceeași valoare
2	11	$Q, N \leq 200$
3	12	$Q, N \leq 1\,000$ și $1 \leq V_i \leq 2\,000$, pentru fiecare i: $1 \leq i \leq N$
4	14	$Q, N \leq 1\,000$
5	27	$1 \leq V_i \leq 23$, pentru fiecare i: $1 \leq i \leq N$
6	29	Nu există alte restricții suplimentare

Exemple

kth.in	kth.out	Explicații
5 2 3 4 -5 2 1 4 2 2 1	1 2	Sunt $N = 5$ elemente în șirul $V = (4, -5, 2, 1, 4)$. Pentru prima întrebare (pentru care $poz = 2$), dacă secvența formată din $L = 3$ elemente: (V_2, V_3, V_4) ar fi ordonată crescător, aceasta ar deveni: $(-5, 1, 2)$, ceea ce înseamnă că pe cea de a doua ($K = 2$) poziție în cadrul ei s-ar afla valoarea 1.
5 2 3 1 5 2 4 3 2 2 1	4 2	Sunt $N = 5$ elemente în șirul $V = (1, 5, 2, 4, 3)$, $K = 2$ și $L = 3$. $Q = 2$ întrebări formulate.

8.5 Rezolvarea problemei Kth

Subtask 3

Fie: Max = valoarea cea mai mare a unui element din șirul V dat = $\max(V_i)$, unde: $1 \leq i \leq N$. Din restricția $1 \leq V_i \leq 2\,000$, pentru fiecare i: $1 \leq i \leq N$, deducem că $1 \leq Max \leq 2\,000$.

Fiecare întrebare dintre cele Q va fi rezolvată imediat după citirea valorii poz corespunzătoare, astfel:

1. Se inițializează un tablou unidimensional de frecvență/numărare $cnt[1, \ldots, Max]$. La început, $cnt[x] = 0$, pentru fiecare x: $1 \leq x \leq Max$.
2. Se iterează cu ajutorul unui indice j (unde: $poz \leq j \leq poz + L - 1$) prin secvența de L elemente pentru care trebuie să dăm răspunsul la întrebare. Când se ajunge în dreptul valorii V_j, vom marca corespunzător o nouă apariție în tabloul cnt, astfel: $cnt[V_j] = cnt[V_j] + 1$.
3. După această traversare (prin exact L elemente): $cnt[x]$ = [de câte ori se găsește valoarea x în secvența $V_{poz}, \ldots, V_{poz+L-1}$]. Acest număr este egal cu 0 în cazul în care nu există niciun indice y: $poz \leq y \leq poz + L - 1$, pentru care $V_y = x$.
4. Pentru a găsi răspunsul la întrebare, dorim să găsim cea mai mică valoare val, pentru care se întâmplă: $cnt[val] > 0$ (ne asigurăm că val există în secvența de lungime L corespunzătoare întrebării curente) și: $cnt[1] + cnt[2] + \ldots + cnt[val] \geq K$. Acest pas poate fi efectuat printr-o parcurgere liniară a tabloului de frecvență cnt.

Complexitatea totală a algoritmului este: $O(N + Q \times (L + Max))$.

Subtask 5

Întrucât $1 \leq V_i \leq 23$, pentru fiecare i: $1 \leq i \leq N$, înseamnă că, pentru fiecare dintre cele Q întrebări răspunsul este un număr din mulțimea: $\{1, 2, 3, \ldots, 22, 23\}$.

Astfel, putem considera următorul tablou bidimensional (*de sume parțiale*): $num[x][i]$ = [câte elemente din mulțimea $\{V_1, V_2, \ldots, V_{i-1}, V_i\}$, adică din prefixul $[1, i]$ din șir, sunt egale cu numărul întreg x], pentru fiecare x: $1 \leq x \leq 23$ și i: $1 \leq i \leq N$. (Se consideră: $num[x][0] = 0$.)

Prin urmare, pentru a răspunde la o întrebare, trebuie, din nou, să determinăm cea mai mică

valoare val ($1 \leq val \leq 23$), pentru care: $(num[val][poz + L - 1] - num[val][poz - 1]) > 0$ și:

$$\sum_{i=1}^{val} (num[i][poz + L - 1] - num[i][poz - 1]) \geq K.$$

Această metodă poate fi implementată *testând*, pe rând, fiecare valoare posibilă pe care val o poate lua; sunt cel mult 23 de astfel de valori.

Complexitatea totală a algoritmului este: $O(N \times 23 + Q \times 23) = O(23 \times (N + Q))$.

Subtask 6

O soluție care garantează trecerea cu succes a tuturor testelor, de exemplu, se poate baza pe metoda *Împărțirii în bucăți de mărime sqrt(n)* (*Sqrt Decomposition* în Engleză); în România, această tehnică mai este cunoscută și sub numele de *Șmenul lui Bogdan Batog* și puteți citi mai multe pe infoarena.ro. Tehnica aceasta va fi utilizată pentru o mai eficientă procesare a, în esență, unor tablouri unidimensionale de frecvență/numărare.

De asemenea, vom alege să *pre-procesăm* (înainte de a citi numărul Q din fișierul de intrare) răspunsul pentru fiecare poziție de început posibilă poz a unei secvențe de lungime L. Sunt exact $N - L + 1$ astfel de poziții de început.

De remarcat! Pentru a evita procesarea de numere/valori negative, putem crește valoarea fiecărui element din șirul V cu un număr întreg constant; de exemplu, putem crește cu 50 001 fiecare element: $V_i = V_i + 50\,001$, pentru fiecare i: $1 \leq i \leq N$. Acum, elementele din șirul inițial pot avea doar valori întregi pozitive cuprinse în intervalul $[1, 100\,001]$. Când se va efectua afișarea unui răspuns, trebuie să avem grijă, la final, să scădem din el această constantă 50 001.

Să notăm: $vMax$ = valoare maximă din șirul V, după ce fiecare element a fost crescut cu valoarea 50 001. De vreme ce $\lceil \sqrt{vMax} \rceil \leq \lceil \sqrt{100\,001} \rceil = 317$, putem considera următorul tablou unidimensional cu 316 elemente, numerotate începând de la 1: $t[i]$ = [câte valori cuprinse între $((i-1) \times 317 + 1)$ și $\min((i \times 317), vMax)$ există în șirul V în cadrul secvenței de lungime L: $V_{poz}, \ldots, V_{poz+L-1}$], pentru fiecare i: $1 \leq i \leq 316$; cu alte cuvinte, elementul $t[i]$ (sau *bucket-ul* i) reține informație despre valorile: $((i-1) \times 317 + 1), \ldots, \min((i \times 317), vMax)$. Variabila poz reprezintă *indicele* cu ajutorul căruia efectuăm *pre-procesarea*: $1 \leq poz \leq N - L + 1$.

Când se realizează trecerea de la poz la $poz + 1$, avem grijă să **ștergem** *o copie* a valorii V_{poz} și să **adăugăm** *o copie* a valorii V_{poz+L} în structura reprezentată de t. Ambele operații, atât de ștergere, cât și de adăugare a unei valori x, pot fi realizate în $O(1)$, prin modificarea elementului corespunzător: $t[\lfloor \frac{x+316}{317} \rfloor]$.

Din nou, pentru a afla răspunsul pentru secvența curentă (din cadrul *pre-procesării*) trebuie determinată cea mai mică valoare val ($1 \leq val \leq 100\,001$), pentru care $t[j] > 0$ și: $t[1] + t[2] + \ldots + t[j-1] + t[j] \geq K$, unde: $j = \lfloor \frac{val+316}{317} \rfloor$. Această operație poate fi efectuată în $O(\sqrt{vMax})$.

Complexitatea totală a algoritmului este: $O(N \times \sqrt{vMax} + Q)$.

Soluție alternativă - prof. Stelian Ciurea

Iată și o soluție în $O(N \times \log L + Q)$. Ea depinde de existența unei structuri de date care să ne ofere operații de inserare, de ștergere și de aflare a maximului și a minimului în timp logaritmic. Precalculăm într-un tablou unidimensional B răspunsurile la toate întrebările posibile. Astfel, $B[i]$ va reține răspunsul pentru secvența $V[i \ldots i + L - 1]$ (pentru i: $1 \leq i \leq N - L + 1$).

Vom procesa toate secvențele de lungime L de la stânga la dreapta. Vom menține elementele secvenței curente în două grupe: în grupa 1 cele mai mici K elemente, iar în grupa 2 cele mai

mari $L - K$ elemente. În acest caz, răspunsul la întrebare este valoarea maximă din grupa 1. Acum, presupunând că am calculat aceste două grupe pentru secvența care începe la poziția i, vom determina cele două grupe pentru secvența care începe la poziția $i + 1$. Pentru aceasta, observăm că secvența care începe la $i + 1$ conține în locul valorii $V[i]$ valoarea $V[i + L]$. În concluzie, vom șterge valoarea $V[i]$ din grupa în care se află și vom insera în grupa potrivită valoarea $V[i + L]$. Pentru a determina grupele în care facem ștergerea, respectiv inserarea, vom compara fiecare dintre cele două valori cu maximul din grupa 1; dacă sunt mai mici, ștergerea și inserarea se vor face în grupa 1, altfel în grupa 2. Observație: dacă ștergerea și inserarea se fac în grupe diferite, atunci mărimile celor două seturi vor devia de la cele dorite (K și $L - K$). Putem reechilibra grupele transferând, după caz, maximul grupei 1 în grupa 2 sau minimul grupei 2 în grupa 1.

Pentru prima secvență, cele două grupe trebuie construite în mod diferit. O variantă este să inserăm primele K valori în grupa 1, apoi pe celelalte $L - K$ să le inserăm cu reechilibrarea grupelor. O altă variantă este să inserăm pur și simplu toate valorile în grupa 1 și să facem reechilibrarea grupelor doar în momentele în care notăm răspunsurile la întrebări.

O structură de date care oferă aceste operații este `multiset`, declarată în biblioteca `<set>`. O altă structură este un *heap*, care însă trebuie implementat cu grijă, căci varianta de bază nu oferă ștergerea unui element arbitrar, ci numai a maximului/minimului.

8.6 Cod-sursă pentru problema Kth

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

const int NMAX = 3e5 + 1;
const int absVMAX = 5e4;
const int ADD = absVMAX + 1;
const int DIM = 317;

int N, K, L;
int V[NMAX], Max = (int)(-1e5);
int ans[NMAX];
int fr[((absVMAX << 1) + 10)];
int M, cnt_bucket[(DIM + 5)];

int my_max(int a, int b)
{
    return ((a > b) ? a : b);
}

void Read()
{
    f >> N >> K >> L;
    for (int i = 1; i <= N; ++i)
        f >> V[i], V[i] += ADD, Max = my_max(Max, V[i]);
    return;
}

int Find(int pos)
{
    return (int)((pos + DIM - 1) / DIM);
}
```

```
void Modify(int X, int Val)
{
    fr[X] += Val;
    cnt_bucket[(int)(Find(X))] += Val;
    return;
}

int Find_Kth()
{
    int sum = 0, ans = 0;
    for (int i = 1; i <= M; ++i)
    {
        sum += cnt_bucket[i];
        if (sum >= K)
        {
            int sum_prime = sum - cnt_bucket[i];
            int need = K - sum_prime;
            int left = ((i - 1) * DIM + 1), right = (i * DIM);

            for (int x = left; x <= right; ++x)
                if (fr[x] > 0)
                {
                    need -= fr[x];
                    if (need <= 0)
                    {
                        ans = x;
                        break;
                    }
                }
            break;
        }
    }
    return ans;
}

void Load()
{
    for (int i = 1; i <= L; ++i)
        Modify(V[i], +1);
    ans[1] = Find_Kth();
    return;
}

void Precalculation()
{
    M = Find(Max);
    Load();

    for (int i = 2; i <= (N - L + 1); ++i)
    {
        Modify(V[i - 1], -1);
        Modify(V[(i + L - 1)], +1);
        ans[i] = Find_Kth();
    }

    for (int i = 1; i <= (N - L + 1); ++i)
        ans[i] -= ADD;
    return;
}

void Test_Case()
```

```
{
    int pos = 0;
    f >> pos;
    g << ans[pos] << '\n';
    return;
}

void Solve()
{
    int Q = 0;
    f >> Q;
    for (int q = 1; q <= Q; ++q)
        Test_Case();
    return;
}

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

8.7 Problema Struguri

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

În 29 septembrie 1474 Ștefan cel Mare a cerut ajutor papei Sixtus al IV-lea în vederea apropiatului război care bătea la ușă.

Apropiindu-se toamna și fiind în criză de timp din cauza războiului iminent, Ștefan a hotărât să supravegheze personal recoltarea strugurilor de la viile Huși, din apropiere de Vaslui, vie la care Ștefan ținea foarte mult. Strugurii recoltați au fost depozitați în grămezi la marginea fiecărui rând de vie. Se cunoaște, pentru fiecare dintre cele N rânduri, cantitatea (în ocale) recoltată de pe rândul respectiv. Ștefan a hotărât ca transportul strugurilor la cramă să se facă cu ajutorul unor căruțe, o căruță putând transporta orice cantitate de struguri. Recolta fiind foarte bogată, transportul se va face în una sau mai multe ture. Ștefan, care supraveghea atent activitatea, a constatat că la fiecare tură dispune de **exact atâtea** căruțe câte grămezi au mai rămas de transportat. Ștefan era un conducător corect astfel încât a hotărât ca toate căruțele disponibile la un moment dat să transporte **aceeași** cantitate de struguri.

Cerințe

1. Determinați cea mai lungă **secvență** de grămezi pe care le pot transporta cele N căruțe **în prima tură**.
2. Determinați **o modalitate** de transport a tuturor strugurilor la cramă, conform cerințelor lui Ștefan.

Date de intrare

Fișierul de intrare `struguri.in` conține pe prima linie două numere naturale C și N, separate printr-un spațiu, reprezentând cerința (1 sau 2) și numărul de grămezi de struguri care trebuie transportate. Linia a doua conține N numere naturale nenule, separate prin câte un spațiu, reprezentând cele N cantități de struguri.

Date de ieșire

Dacă cerința este 1, în fișierul de ieșire `struguri.out` se vor afișa pe prima linie trei numere naturale, separate prin câte un spațiu, reprezentând lungimea secvenței de grămezi, numărul de ordine al primei grămezi din secvență, respectiv numărul de ordine al ultimei grămezi din secvență. Dacă cerința este 2, în fișierul de ieșire se vor afișa: pe prima linie numărul T de ture; fiecare dintre următoarele T linii are aceeași structură:

- primele trei valori reprezintă numărul de căruțe care transportă struguri în tura respectivă, cantitatea de struguri pe care o transportă fiecare căruță, respectiv numărul de grămezi transportate;
- următoarele valori de pe linia respectivă reprezintă **numerele de ordine** ale grămezilor transportate în tura respectivă; valorile vor fi afișate ordonate crescător și separate prin câte un spațiu.

Restricții

- $1 \leq N \leq 20\,000$
- Cantitățile sunt mai mici ca $100\,000$

- Pentru ambele cerințe orice soluție corectă este acceptată
- Pentru cerința 1 se acordă `46` puncte, iar pentru cerința 2 se acordă `54` puncte

#	Puncte	Restricții
1	12	$1 \leq N \leq 10$, din care 5 puncte se acordă pentru cerința 1
2	30	$11 \leq N \leq 100$, din care 12 puncte se acordă pentru cerința 1
3	14	$101 \leq N \leq 1\,000$, din care 8 puncte se acordă pentru cerința 1
4	27	$1\,001 \leq N \leq 10\,000$, din care 21 de puncte se acordă pentru cerința 1
5	17	$10\,001 \leq N \leq 20\,000$

Exemple

struguri.in	struguri.out	Explicații
`1 10` `2 1 3 4 5 10 6 8 9 7`	`5 6 10`	De la grămada a șasea până la a zecea sunt în total 40 ocale; fiecare dintre cele 10 căruțe va transporta câte 4 ocale.

`2 5`
`7 8 11 2 5`
	`3`
	`5 4 3 1 2 5`
	`2 1 1 4`
	`1 11 1 3`
		O soluție posibilă poate fi următoarea:
		Tura 1: există 5 grămezi, deci sunt 5 căruțe (fiecare căruță transportă câte 4 oca din 3 grămezi (1 2 5)
		Tura 2: au mai rămas 2 grămezi deci sunt 2 căruțe, fiecare transportă câte 1 oca din grămada 4
		Tura 3: a mai rămas o grămadă deci dispunem de 1 căruță, ce va transporta 11 ocale din grămada 3
		O altă soluție corectă poate fi descrisă în fișierul de ieșire astfel:
		2
		5 3 2 1 2
		3 6 3 3 4 5
		În această situație sunt 2 ture, în prima se transportă cu fiecare din cele 5 căruțe, câte 3 ocale, din grămezile 1 și 2. La a doua tură se vor folosi 3 căruțe care transportă fiecare câte 6 ocale din grămezile 3, 4 și 5

8.8 Rezolvarea problemei Struguri

Se utilizează principiul **cutiei lui Dirichlet** la fiecare tură (pentru grămezile rămase). O demonstrație a acestuia se găsește, de exemplu, pe Wikipedia.

Pentru a determina o secvență cu proprietatea din enunț vom considera sumele parțiale:

$S_1 = a_1$
$S_2 = a_1 + a_2$
...
$S_i = a_1 + a_2 + \cdots + a_i (1 \leq i \leq n)$

...
$S_n = a_1 + a_2 + \cdots + a_n$.

Avem două cazuri:

- există k cu S_k divizibil cu n, caz în care secvența căutată este $a_1, \ldots a_k$;
- nu există sume care să fie divizibile cu n. În acest caz sumele dau la împărțirea la n resturi ce fac parte din mulțimea $\{1, 2, \ldots n-1\}$. Cum sunt n sume și $n-1$ resturi posibile, conform principiului cutiei lui Dirichlet, rezultă faptul că există două sume S_p și S_q $(p < q)$ care la împărțirea la n dau același rest. Deci $S_q - S_p$ este divizibil cu n și putem lua secvența $a_{p+1}, \ldots a_q$

În funcție de implementare se pot obține timpi diferiți:

1. implementat cu Dirichlet deștept (cel mai rapid) – se calculează sumele la citire/refacere și la detectarea unui rest 0 sau a unui rest care a mai apărut se oprește procesul.
2. implementat cu căutare completă 0 în resturi apoi, dacă nu există un rest 0, căutarea a două resturi egale.
3. implementat cu cea mai lungă secvență detectată la fiecare pas (cele mai puține ture).
4. implementat cu cea mai scurtă secvență detectată la fiecare pas (cele mai multe ture).

Desigur, și aceste implementări pot fi îmbunătățite. De exemplu, la implementările (2), (3), (4), după detectarea a două resturi egale procesul poate fi oprit.

Având în vedere că la cerința 1 se cere **secvența** cea mai lungă, se poate deduce, dacă nu se cunoaște principiul cutiei lui Dirichlet, că **există o secvență** de N numere a cărei sumă este divizibilă cu N. Utilizând această idee se obțin pe rând secvențe cu această proprietate (pentru valori diferite ale lui N). Căutarea unei secvențe se poate face elementar, luând pe rând intervale $[st, dr]$ de lungimi $N, N-1, N-2, \ldots$. În funcție de implementare, se obțin punctaje diferite.

8.9 Cod-sursă pentru problema Struguri

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

struct gramada
{
   unsigned int nrG;
   unsigned int oca;
   bool luat;
} G[20001], G1[20001];

struct solutie
{
   unsigned int nr_carute;
   unsigned int cate;
   unsigned int oca_pe_caruta;
   unsigned int care[20001];
} sol[501];

int N, i, ramase, caz, nrsol, C;

void citire()
{  int i;
   fin >> C >> N;
   for (i = 1; i <= N; i++)
```

```
   {
      fin >> G1[i].oca;
      G1[i].nrG = i;
   }
   ramase = N;
}

void refa()
{  int j = 0;
   for (i = 1; i <= N; i++)
      if (!G1[i].luat)
         G[++j] = G1[i];
   N = j;
   memcpy (G1, G, sizeof (G));
}

void Dirichlet_lung()
{  unsigned int Rest[20001] = {0};
   int S = 0, i, j, deunde, pana_unde, lmax = 0;
   caz = 0;
   for (i = 1; i <= N; i++)  //calculez sumele si resturile
   {
      S = S + G1[i].oca;
      Rest[i] = S % N;
   }
   //caut 0 in resturi
   i = N;
   while (i >= 1 && Rest[i]) i--;
   if (i)
   {
      lmax = i;          //am carutele 1..i
      caz = 1;           //adica am un rest 0
      deunde = 1;
      pana_unde = i;
          caz = 1;
   }

   for (i = 1; i < N; i++)  //acum caut resturi egale
      for (j = N; j > i; j--)
        if (Rest[i] == Rest[j])  // am gasit 2 resturi egale la distanta maxima
           if (j - i > lmax)
               {
              lmax = j - i;
              deunde = i + 1;
                            pana_unde = j;
               }
   if (ramase == N && C == 1)
   {
           fout << lmax << ' ' << deunde << ' ' << pana_unde << '\n';
           exit(0);
   }
   nrsol++;
   sol[nrsol].nr_carute = ramase;
   sol[nrsol].cate = lmax;
   S = 0;
   for (i = 1; i <= lmax; i++)
   {
      sol[nrsol].care[i] = G1[deunde].nrG;
      S += G1[deunde].oca;
      G1[deunde].luat = true;
      deunde++;
```

```
   }
   sol[nrsol].oca_pe_caruta = S / ramase;
   ramase = ramase - lmax;
}

void afisare()
{
   int i, j;
   fout << nrsol << '\n';
   for (i = 1; i <= nrsol; i++)
   {
      fout << sol[i].nr_carute << ' ';
      fout << sol[i].oca_pe_caruta << ' ';
      fout << sol[i].cate << ' ';
      for (j = 1; j <= sol[i].cate; j++)
         fout << sol[i].care[j] << ' ';
      fout << '\n';
   }
}

int main()
{
   citire();
   while (ramase)
   {
      refa();
      Dirichlet_lung();
   }
   afisare();
   return 0;
}
```

Capitolul 9

Baraj selecție lot juniori ONI 2023

9.1 Problema Extrapare

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

Un număr natural se numește *extrapar* dacă poate fi scris ca sumă de puteri distincte ale lui 2, puteri care au exponent par. Numărul 0 este considerat, de asemenea, extrapar. Considerând reprezentarea în baza 2 pentru un număr natural, se numerotează pozițiile cifrelor din reprezentare, de la dreapta către stânga, începând cu 0. Asupra reprezentării în baza 2 trebuie să se efectueze o singură operație. Operația constă din eliminarea a exact K cifre situate pe poziții consecutive.

Cerințe

Fiind date reprezentările în baza 2 pentru N numere naturale, să se determine pentru fiecare dintre ele dacă se poate obține un număr *extrapar* în condițiile de mai sus.

Date de intrare

Fișierul de intrare `extrapare.in` conține pe prima linie două numere naturale N K, separate printr-un spațiu. Pe fiecare dintre următoarele N linii se află reprezentarea în baza 2 a unui număr natural.

Date de ieșire

Fișierul de ieșire `extrapare.out` va conține N linii. Pe cea de a i-a linie ($1 \leq i \leq N$) se va afișa reprezentarea în baza 2 a numărului extrapar obținut prin efectuarea unei singure operații asupra celei de a i-a reprezentări din fișierul de intrare, sau valoarea -1 dacă obținerea unui număr extrapar nu este posibilă în acest mod.

Restricții

- $0 < N \leq 10$
- $0 \leq K <$ numărul de cifre din oricare reprezentare din fișierul de intrare
- Orice reprezentare din fișierul de intrare are cel mult 1 000 000 de cifre.
- Dacă există mai multe modalități de a efectua o operație astfel încât rezultatul să fie un număr extrapar, se va afișa rezultatul pentru acea operație în care poziția primei cifre eliminate este cea mai mare (cea mai din stânga poziție).

- Se garantează că reprezentările în baza 2 din fișierul de intrare au cifra cea mai din stânga egală cu 1.
- Reprezentarea în baza 2 a numărului rezultat în urma efectuării operației se va afișa fără zerourile nesemnificative ce se pot forma la stânga lui.

#	Puncte	Restricții
1	19	$K = 0$
2	32	$K > 0$, lungimea șirurilor $\leq 1\,000$
3	49	$K > 0$, lungimea șirurilor $\leq 1\,000\,000$

Exemple

extrapare.in	extrapare.out	Explicații
9 3 1001101 1010000010 101010001 111010100 100100 100010100 101000001 11110 101000	-1 1010000 10001 10100 100 10100 1 -1 0	Trebuie să eliminăm 3 cifre pentru a forma numere extrapare. - Numerele pentru care nu se poate obține un număr extrapar prin eliminarea a trei cifre de pe poziții consecutive sunt primul și al optulea. - Din 1010000010 tăiem cifrele de pe pozițiile 0, 1 și 2 și obținem 1010000. - 101010001 este deja extrapar și vom elimina cele mai din stânga trei cifre. - Observăm că dacă rezultatul este format doar din cifre egale cu 0 se va afișa un singur 0. Și așa mai departe.

9.2 Rezolvarea problemei Extrapare

Subtask k = 0

Parcurgem reprezentările binare și, pentru fiecare reprezentare, verificăm dacă există vreo cifră 1 care este pe o poziție impară. În acest caz, se afișează -1, în caz contrar se afișează reprezentarea respectivă.

Soluție completă

Pentru fiecare reprezentare binară, vom afla mai întâi cea mai din dreapta poziție care nu se potrivește cu șablonul unui număr extrapar, această poziție fiind cea mai din dreapta poziție impară pe care se află o cifră egală cu 1. Să notăm această poziție cu x. Dacă nu există o asemenea poziție, putem tăia primele k valori urmând să avem grijă la scoaterea zerourilor nesemnificative de la începutul numărului.

Altfel, vom tăia o subsecvență de lungime k ce se termină la poziția x (intervalul pozițiilor tăiate va fi $[x - k + 1, x]$ sau $[1, k]$ dacă $x < k$) și vom verifica dacă șirul rămas este extrapar. În cazul în care șirul este extrapar, vom tăia zerourile nesemnificative și vom afișa răspunsul cerut.

Soluție alternativă

O altă abordare constă în a precalcula pentru fiecare reprezentare lungimea sufixului valid (care respectă șablonul de extrapar), precum și lungimea prefixului valid atât care presupune prima poziție drept o poziție pară, cât și cel care presupune prima poziție drept o poziție impară. Vom încerca să tăiem pe rând fiecare subsecvență de lungime k, verificarea fiind posibilă în $O(1)$ folosindu-ne de precalculările făcute anterior. Dacă găsim o poziție validă (prefixul și sufixul corespunzător sunt ambele valide), atunci vom afișa șirul fără secvența de lungime k, având grijă din nou la zerourile nesemnificative.

9.3 Cod-sursă pentru problema Extrapare

```
#include <bits/stdc++.h>
using namespace std;
ifstream f("extrapare.in");
ofstream g("extrapare.out");
int n,k;
void solveTest()
{
    string s,t;
    f>>s;
    int lg=s.size(),R,L;
    R=lg-2;
    while(R>=0&&s[R]=='0')R-=2;
    if(R<k)
        t=s.substr(k,1000001);
    else
        t=s.substr(0,R-k+1)+s.substr(R+1,1000001);
    lg=t.size();L=0;
    while(L<lg-1&&t[L]=='0')L++;
    t=t.substr(L,1000010);
    lg=t.size();
    if(lg%2==0)
    {
        g<<"-1\n";
        return;
    }
    for(int i=1;i<lg;i+=2)
        if(t[i]=='1')
        {
            g<<"-1\n";
            return;
        }
    g<<t<<'\n';
}
int main()
{
    f>>n>>k;
    for(int i=1;i<=n;i++)
        solveTest();
    return 0;
}
```

9.4 Problema Fuziune

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

Se consideră un șir de n numere naturale nenule $a = (a_1\ a_2\ \ldots\ a_n)$. Două numere situate pe poziții consecutive în șir (a_i și a_{i+1}, unde $1 \leq i < n$) pot *fuziona* dacă ele au cel puțin un divizor comun strict mai mare decât 1. În urma fuziunii ele vor fi înlocuite de cel mai mic număr care se divide cu toți divizorii lui a_i și ai lui a_{i+1}. Operația de fuziune se poate repeta, pe noul șir obținut, până când în șir nu va exista nicio pereche de numere situate pe poziții consecutive care să poată fuziona. Să notăm cu b șirul obținut după efectuarea tuturor operațiilor de fuzionare.

Numim *coeficient de fuziune* al șirului b și îl notăm cu $cf(b)$ un număr nenul care are proprietatea că orice termen al șirului b are cel puțin un divizor comun cu $cf(b)$, strict mai mare decât 1. În plus, $cf(b)$ trebuie să respecte următoarele condiții:

1. factorii săi primi sunt distincți (de exemplu $30 = 2 \cdot 3 \cdot 5$ are doar factori primi distincți, dar $18 = 2 \cdot 3 \cdot 3$ nu are doar factori primi distincți);
2. orice factor prim al lui $cf(b)$ este factor prim pentru cel puțin un număr din șirul b;
3. lista factorilor primi distincți ai lui $cf(b)$ ordonată strict crescător este minimă din punct de vedere lexicografic.

Cerințe

Dat fiind un șir de numere naturale nenule, scrieți un program care să rezolve următoarele două cerințe:

1. să se determine lungimea minimă a șirului b obținut după efectuarea tuturor operațiilor de fuziune posibile;
2. să se determine $cf(b)$.

Date de intrare

Fișierul de intrare `fuziune.in` conține pe prima linie cerința C care trebuie să fie rezolvată (1 sau 2). Pe cea de-a doua linie se află un număr natural n, reprezentând numărul de valori din șir. Pe următoarele n linii se află cele n numere din șir, câte un număr pe o linie.

Date de ieșire

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

Restricții

- $1 \leq n \leq 100\,000$
- $2 \leq a_i \leq 2 \cdot 10^9$, pentru orice $1 \leq i \leq n$
- Se garantează că numărul de factori primi distincți pentru numerele din șirul b (pentru cerința 1) și pentru $cf(b)$ (pentru cerința 2) este cel mult egal cu 250.
- Fie $d = (d_1 < d_2 < \cdots < d_k)$ și $f = (f_1 < f_2 < \cdots < f_h)$ două liste de factori primi distincți ordonate strict crescător. Spunem că d este mai mică din punct de vedere lexicografic decât f dacă există o poziție i astfel încât $d_j = f_j$, pentru orice $1 \leq j < i$ și ($d_i < f_i$ sau $i = k+1$).

#	Puncte	Restricții
1	15	$C = 1$, $1 \leq n \leq 1\,000$ și numerele din șir sunt $\leq 10^4$.
2	15	$C = 1$, $10\,000 \leq n \leq 100\,000$ și numerele din șir sunt $\leq 10^4$.
3	10	$C = 1$, $1 \leq n \leq 1\,000$ și numerele din șir sunt $\leq 2 \cdot 10^9$.
4	30	$C = 1$, $10\,000 \leq n \leq 100\,000$ și numerele din șir sunt $\leq 2 \cdot 10^9$.
5	12	$C = 2$ și rezultatul va fi un număr de maximum 18 cifre.
6	18	$C = 2$ și rezultatul va fi un număr cu maximum 1500 cifre.

Exemple

fuziune.in	fuziune.out	Explicații
1 8 30 18 997 121 625 5 55 101	4	$C = 1$. Numerele $30 = 2 \cdot 3 \cdot 5$ și $18 = 2 \cdot 3^2$ vor fuziona și se obține valoarea $90 = 2 \cdot 3^2 \cdot 5$. Numerele 625, 5 și 55 vor fuziona de asemenea, apoi rezultatul va fuziona cu 121 și se obține valoarea $75625 = 5^4 \cdot 11^2$. În șirul rezultat vor rămâne, după efectuarea tuturor fuzionărilor 4 numere (90, 997, 75625 și 101).
2 3 16 18 25	30	$C = 2$. În urma efectuării tuturor operațiilor de fuziune posibile se obține șirul $144 = 2^4 \cdot 3^2$ $25 = 5^2$ $cf(b) = 2 \cdot 3 \cdot 5 = 30$.
2 8 30 18 97 121 625 5 55 10403	3233010	$C = 2$. În urma efectuării tuturor operațiilor de fuziune posibile se obține șirul $90 = 2 \cdot 3^2 \cdot 5$ 97 $75625 = 5^4 \cdot 11^2$ $10403 = 101 \cdot 103$ $cf(b)$=$2 \cdot 3 \cdot 5 \cdot 11 \cdot 97 \cdot 101 = 3233010$.

9.5 Rezolvarea problemei Fuziune

O primă observație constă în faptul că pentru a verifica dacă fuziunea este posibilă este suficient să reținem lista factorilor primi distincți pentru orice număr (și pentru cele pe care le citim și pentru cele obținute ca rezultat al unei fuziuni). Vom reține factorii primi distincți în ordine crescătoare, pentru a optimiza operațiile de fuziune.
Pentru a obține eficient descompunerea în factori primi a fiecărui număr va trebui să ne folosim de faptul că există maximum 250 de factori primi distincți. Pe măsură ce descompunem numerele în factori primi, vom ține o listă (vector) cu numerele prime pe care le-am descoperit până acum. Astfel, pentru a descompune un număr în factori primi, vom parcurge lista de factori primi descoperiți până acum și pe fiecare îl vom împărți repetat la numărul nostru. Dacă la

final rămânem cu un număr diferit de 1, vom continua cu o descompunere în factori primi clasică și fiecare factor prim nou întâlnit va fi adăugat în listă. Astfel, vom face maximum 250 de descompuneri în factori primi clasice, reducând astfel timpul de execuție semnificativ.

Definim un tip *lista* în care reținem lista factorilor primi distincți ai unui număr și lungimea acesteia.

```
struct lista {int d[LGMAX];
              int lg;};
```

Vom utiliza un vector b cu elemente de tip *lista* în care reținem termenii șirului b, obținut după efectuarea tuturor operațiilor de fuziune. Să notăm cu lgb lungimea lui b.

Citim succesiv numerele din șirul dat și descompunem numărul curent în factori primi și reținem doar factorii primi distincți (în ordine crescătoare) în variabila *crt* de tip *lista*. Cât timp ultimul element din b fuzionează cu *crt*, *crt* va fi înlocuit cu reuniunea dintre *crt* și $b[lgb]$, iar $b[lgb]$ este eliminat din b. Când *crt* nu mai fuzionează cu $b[lgb]$ (sau $lgb == 0$) adăugăm pe *crt* în b. Pentru a verifica dacă două liste fuzionează și pentru a determina reuniunea, vom utiliza un algoritm similar cu interclasarea, ținând cont de faptul că divizorii sunt în ordine crescătoare.

Dacă cerința este 1, vom afișa lgb.

Dacă cerința este 2, vom determina produsul divizorilor primi distincți ai numerelor din b, până când condiția ca fiecare număr din b să aibă un divizor în comun cu $cf(b)$ este asigurată. Pentru determinarea coeficientului de fuziune $cf(b)$ este necesară implementarea înmulțirii dintre un număr mare și un număr de tip *int*.

Observație

Punctaje parțiale se pot obține pentru testele pentru care valorile din șirul dat sunt mici, folosind o reprezentare a listei divizorilor prin vector caracteristic. De asemenea punctaje parțiale la cerința 2 se pot obține dacă se lucrează pentru $cf(b)$ cu tipul *long long int*.

Soluție alternativă

O altă variantă pentru a efectua fuzionările este următoarea: Pornim de la o soluție ineficientă care fuzionează maximal elementele de la stânga la dreapta, cât timp există fuzionări. Astfel pentru șirul $7, 6, 2, 3, 2, 3, 2, 3$. O tură de fuzionare ar aduce șirul în forma $7, 6$. 6 fuzionează cu 2, apoi rezultatul cu 3 și așa mai departe. Se observă că pentru șirul $7, 2, 3, 2, 3, 2, 3, 6$, după o tură de fuzionare am obține șirul $7, 2, 3, 2, 3, 2, 6$. O astfel de soluție ar avea o complexitate pătratică în raport cu N. La fel și una care fuzionează maximal elementele de la dreapta la stânga. Totuși, dacă vom fuziona maximal elementele de la stânga la dreapta, apoi de la dreapta la stânga, apoi de la stânga la dreapta etc vom obține o complexitate liniară, cu o constantă 250.

9.6 Cod-sursă pentru problema Fuziune

```
#include <bits/stdc++.h>
#define N_MAX 100000
#define PRIME_MAX 250
#define DIG_MAX 1500
using namespace std;
ifstream fin("fuziune.in");
ofstream fout("fuziune.out");

struct prime_list {
    int len;
```

```
    int primes[PRIME_MAX + 5];
};

int C, N;
prime_list primes;
prime_list prime_d[N_MAX + 5];
int parent[N_MAX + 5];
bool used[N_MAX + 5];
int ans[DIG_MAX + 5];
int len_ans;

prime_list prime_decomp(int X) {
    int last = primes.len;
    prime_list ret;
    ret.len = 0;
    for(int d = 1; d <= primes.len; d++) {
        if(X % primes.primes[d] == 0) {
            ret.primes[++ret.len] = primes.primes[d];
            while(X % primes.primes[d] == 0) {
                X /= primes.primes[d];
            }
        }
    }

    for(int d = 2; 1ll * d * d <= X; d++) {
        if(X % d == 0) {
            primes.primes[++primes.len] = d;
            ret.primes[++ret.len] = d;
            while(X % d == 0) {
                X /= d;
            }
        }
    }

    if(X > 1) {
        ret.primes[++ret.len] = X;
        primes.primes[++primes.len] = X;
    }

    if(last != primes.len) {
        sort(primes.primes + 1, primes.primes + primes.len + 1);
    }

    sort(ret.primes + 1, ret.primes + ret.len + 1);
    return ret;
}

bool intersect(int a, int b) {
    int j = 1;
    for(int i = 1; i <= prime_d[a].len && j <= prime_d[b].len; i++) {
        while(j <= prime_d[b].len && prime_d[b].primes[j] < prime_d[a].primes[i]) {
            j++;
        }
        if(j <= prime_d[b].len && prime_d[b].primes[j] == prime_d[a].primes[i]) {
            return true;
        }
    }
    return false;
}

int unite(int a, int b) {
```

```
    int j = 1;

    prime_list ret;
    ret.len = 0;

    for(int i = 1; i <= prime_d[a].len; i++) {
        while(j <= prime_d[b].len && prime_d[b].primes[j] < prime_d[a].primes[i]) {
            ret.primes[++ret.len] = prime_d[b].primes[j++];
        }
        if(j <= prime_d[b].len && prime_d[b].primes[j] == prime_d[a].primes[i]) {
            j++;
        }
        ret.primes[++ret.len] = prime_d[a].primes[i];
    }
    for(; j <= prime_d[b].len; j++) {
        ret.primes[++ret.len] = prime_d[b].primes[j];
    }
    prime_d[a] = ret;
    return a;
}

int main() {
    fin >> C >> N;
    int last_N = N;
    for(int i = 1; i <= N; i++) {
        int x;
        fin >> x;
        prime_d[i] = prime_decomp(x);
        parent[i] = i;
    }

    int L = 0;
    for(int i = 1; i <= N; i++) {
        int idx = i;
        while(L > 0 && intersect(parent[L], idx)) {
            idx = unite(parent[L--], idx);
        }
        parent[++L] = idx;
    }
    N = L;
    if(C == 1) {
        fout << N << "\n";
        return 0;
    }

    for(int i = 1; i <= last_N; i++) {
        reverse(prime_d[i].primes + 1, prime_d[i].primes + prime_d[i].len + 1);
    }

    ans[1] = 1; len_ans = 1;
    int cnt = 0;
    while(cnt < N) {
        int mi = INT_MAX;
        for(int i = 1; i <= N; i++) {
            if(prime_d[parent[i]].len == 0) {
                continue;
            }
            mi = min(mi, prime_d[parent[i]].primes[prime_d[parent[i]].len]);
        }
        int L = 0;
        for(int i = 1; i <= N; i++) {
```

```
            if(prime_d[parent[i]].len == 0) {
                continue;
            }
            if(mi == prime_d[parent[i]].primes[prime_d[parent[i]].len]) {
                cnt += used[i] == false;
                used[i] = true;
                prime_d[parent[i]].len--;
            }
        }

        long long t = 0;
        for(int i = 1; i <= len_ans; i++, t /= 10) {
            ans[i] = (t += 1ll * ans[i] * mi) % 10;
        }
        while(t) {
            ans[++len_ans] = t % 10;
            t /= 10;
        }
    }

    for(int i = len_ans; i >= 1; i--) {
        fout << ans[i];
    }
    fout << "\n";
    return 0;
}
```

9.7 Problema Udp

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

Chris a scris pe un caiet foarte multe numere de două sau trei cifre, toate divizibile cu 7. Interesant este că aceste numere conțineau doar cifrele 1, 2 sau 4. Mădălina a făcut curățenie în casă și a aruncat caietul. Acum Chris este foarte supărat că și-a pierdut numerele. Ca să îl înveselească, Mădălina i-a spus lui Chris:

„Numerele tale conțineau exact U cifre de 1, D cifre de 2 și P cifre de 4". Bucuros, Chris a venit la concurenții de la ONIGim și i-a rugat să îi regăsească numerele. Experți în programare, concurenții s-au apucat de treabă, dar imediat și-au dat seama că este posibil să existe mai multe moduri de a reconstitui numerele. Din fericire Chris se mulțumește cu oricare reconstituire și a promis 100 de puncte pentru fiecare concurent care îi va oferi o soluție validă.

Cerințe

Cunoscând cele trei numere U, D și P cu semnificația din enunț, să se determine numere de două sau trei cifre, divizibile cu 7 astfel încât în numerele determinate să se regăsească exact U cifre de 1, D cifre de 2 și P cifre de 4.

Date de intrare

Fișierul de intrare `udp.in` conține trei numere naturale U, D și P, separate prin câte un spațiu, având semnificația că pe caietul lui Chris erau scrise exact U cifre de 1, D cifre de 2 și P cifre de 4.

Date de ieșire

Fișierul de ieșire `udp.out` va conține o soluție validă afișată după cum urmează: pe prima linie un număr K, reprezentând numărul de valori distincte divizibile cu 7, formate din două sau trei cifre care nu pot fi decât 1, 2 sau 4. Pe următoarele K linii vor fi afișate numerele din soluția validă. Astfel pe linia $i+1$ ($1 \leq i \leq K$) se vor afișa câte două numere separate prin spațiu val_i și cnt_i cu semnificația că val_i este un număr de două sau de trei cifre, divizibil cu 7, conținând doar cifre 1, 2 sau 4 și acest număr apare în soluție de cnt_i ori.

Dacă nu există soluții valide, afișați o singură linie cu numărul -1.

Restricții

- $0 \leq U, D, P \leq 10^{15}$.
- Cel puțin una dintre valorile U, D și P este nenulă.

#	Puncte	Restricții
1	25	$0 \leq U, D, P \leq 20$
2	25	$20 < U, D, P \leq 10\,000$
3	25	$10\,000 < U, D, P < 2^{31}$
4	25	$2^{31} \leq U, D, P \leq 10^{15}$

Exemple

udp.in	udp.out	Explicații
0 13 11	2 42 9 224 2	Soluția conține două numere divizibile cu 7, mai precis 42 și 224. Acestea au cel mult trei cifre și sunt divizibile cu 7. Considerând 9 numere cu valoarea 42 și două cu valoarea 224 aceste numere vor conține 0 cifre de 1, 13 cifre de 2 și 11 cifre de 4.
71234 41125 62112	4 14 46110 21 25122 42 16002 112 1	O soluție posibilă ar putea fi: 46110 numere de 14, 25122 numere de 21, 16002 numere de 42 și un număr 112. În total toate aceste numere vor conține 71234 de 1, 41125 de 2 și 62112 de 4.

9.8 Rezolvarea problemei Udp

Numerele de cel mult 3 cifre formate cu cifrele 1,2 și 4 sunt: 14, 21, 42, 112, 224, 441.

Să presupunem că deja avem o soluție. Se poate observă că se poate modifica soluția în altă soluție după următoarele reguli:

Transformarea de tip 1:

- 112 și 224 se transformă în 21, 21, 42
- 112 și 441 se transformă în 14, 14, 21
- 224 și 441 se transformă în 14, 42, 42

Dacă aplicăm de câte ori este posibil transformarea de tip 1 vom obține o soluție care nu va conține decât cel mult una dintre valorile 112, 224, 441.

Transformarea de tip 2: Sa presupunem că au mai rămas valori 112. Aplicăm ori de câte ori este posibil transformarea 112, 112, 42 se transformă în 14, 21, 21, 21.

În urma acestei transformări rezultă una dintre următoarele situații:

- nu mai avem deloc 42 $\Rightarrow$ avem o soluție formată doar numerele 14, 21, 112.
- nu mai avem deloc 112 $\Rightarrow$ avem o soluție formată doar din numerele 14, 21, 42
- mai rămâne o singură valoare 112 $\Rightarrow$ avem o soluție în care folosim exact o dată 112 și în rest mai apar doar numerele 14, 21, 42.

Analog interpretăm efectul aplicării transformărilor de tip 2:

- 224,224,14 se transformă în 21,42,42,42
- 441,441,21 se transformă în 14,14,14,42

Concluzia este următoarea: în urma aplicării transformărilor de tip 1 și 2 se poate afirma că dacă problema are soluție atunci putem forma o soluție având una dintre următoarele 7 forme:

1. conține numai numerele 14, 21, 42
2. conține doar numerele 14, 21, 42 și exact o singură dată numărul 112
3. conține doar numerele 14, 21, 42 și exact o singură dată numărul 224
4. conține doar numerele 14,21,42 și exact o singură dată numărul 441
5. conține doar numerele 14, 21, 112

6. conține doar numerele 21, 42, 224
7. conține doar numerele 14,42,441

Se încearcă determinarea unei soluții în una dintre cele 7 forme și dacă nu se obține o soluție se trage concluzia că problema nu are soluție.

Metoda 1

Se poate observa că, dacă avem o soluție de forma 1, atunci sînt îndeplinite următoarele condiții:

$$\begin{cases} U \leq D + P \\ D \leq U + P \\ P \leq U + D \\ U + D + P \text{ este număr par} \end{cases} \tag{9.1}$$

Vom obține o soluție cu:

- $(U + P - D)/2$ de 14
- $(D + U - P)/2$ de 21
- $(P + D - U)/2$ de 42

Soluțiile de forma 2, 3 sau 4 sunt similare și se obțin astfel (exemplific pentru soluția de forma 2). Deoarece avem o dată valoarea 112, după ce scădem U cu 2 și D cu 1, valorile U, D și P astfel modificate vor verifica condițiile (9.1) deci din acel moment am redus problema la determinarea unei soluții de forma 1.

Soluțiile de forma 5, 6 sau 7 sunt soluții care corespund situațiilor în care una (și numai una) dintre valorile U, D și P „domină" celelalte două valori. Tot pentru exemplificare vom trata forma 5. Soluția conține doar 14, 21, 112 $\Rightarrow U \geq D + P$. Dacă $U > 2D + P$ cu siguranță nu pot obține soluție pentru ca numărul de 1 este prea mare. Dacă caz putem forma $U - D - P$ valori de 112. După ce formăm aceste numere, numărul de valori U, D, P modificate vor respecta relația $U = D + P$. Din acel moment se formează cu cifrele rămase D de 21 și P de 14. Concluzie:
Cazul 1: $U \leq D + P$, $D \leq U + P$, $P \leq U + D$
Dacă $U + D + P$ este par formăm unica soluție posibilă formată doar din numerele 14, 21, 42
Dacă $U + D + P$ impar - încercăm una dintre situațiile în care formând o singură dată unul și numai unul dintre numerele de 3 cifre din cele ramase se formează (în mod unic) soluție formată doar cu numerele 14, 21, 42.
Cazul 2: $U > D + P$ sau $D > U + P$ sau $P > U + D$ sunt similare.
Dacă $U > 2D + P$ sau $D > 2P + U$ sau $P > 2U + P$ problema nu are soluție. Altfel problema are soluție obținută după următoarea strategie (exemplific pentru cazul $U > D + P$).
Dacă $U > D + P$ formăm valori 112 până când obținem $U = D + P$, formăm valori 14 până terminăm valorile de 4 și din cifrele rămase formăm numere 21.
O rezolvare prin această metodă se regăsește într-una dintre sursele oficiale.

Metoda 2

Se poate folosi o strategie prin care ne propunem ca plecând de la cifrele inițiale să aplicăm în mod repetat de un număr maxim posibil de ori modificări prin care formăm numerele propuse combinând una dintre cifre cu alte cifre sau numere de care dispunem cu scopul final de a elimina cifrele și de a rămâne doar cu cele 6 numere (14, 21, 42, 112, 224 și 441). Sunt mai multe astfel de transformări. Le încercăm pe toate, în mod repetat până în momentul în care ajungem în una dintre următoarele două situații:

1. Nu mai avem cifre ⇒ am obținut o soluție.
2. Nu mai putem niciuna dintre transformările posibile și ne mai rămân cifre ⇒ nu există soluție.

Enumerăm o parte dintre transformări lăsând găsirea tuturor ca exercițiu

- 1, 2 se transformă în 21
- 1, 4 se transformă în 14
- 2, 4 se transformă în 42
- 1, 21 se transformă în 112
- ...

9.9 Cod-sursă pentru problema Udp

```
//Metoda 1
#include <bits/stdc++.h>
using namespace std;
ifstream f("udp.in");
ofstream g("udp.out");
int64_t U,D,P,UP,DU,PD,UUD,DDP,PPU;
void CHECK();
int main()
{   f>>U>>D>>P;
    UUD=U-D-P; DU=2*D+P-U; UP=P; PD=DDP=PPU=0; CHECK();
    DDP=D-P-U; PD=2*P+U-D; DU=U; UP=UUD=PPU=0; CHECK();
    PPU=P-U-D; UP=2*U+D-P; PD=D; DU=UUD=DDP=0; CHECK();
    UP=(U+P-D)/2; DU=(D+U-P)/2; PD=(P+D-U)/2; UUD=DDP=PPU=0; CHECK();
    UUD=1; DDP=PPU=0; UP=(U+P-D-1)/2; DU=(U+D-P-3)/2; PD=(P+D-U+1)/2; CHECK();
    DDP=1; PPU=UUD=0; UP=(U+P-D+1)/2; DU=(U+D-P-1)/2; PD=(P+D-U-3)/2; CHECK();
    PPU=1; UUD=DDP=0; UP=(U+P-D-3)/2; DU=(U+D-P+1)/2; PD=(D+P-U-1)/2; CHECK();
    g<<"-1\n";
    return 0;
}
void CHECK()
{   if (UP<0||DU<0||PD<0||UUD<0||DDP<0||PPU<0) return;
    if (UP+DU+2*UUD+PPU!=U) return;
    if (DU+PD+2*DDP+UUD!=D) return;
    if (UP+PD+2*PPU+DDP!=P) return;
    g<<(UP>0)+(DU>0)+(PD>0)+(UUD>0)+(DDP>0)+(PPU>0)<<'\n';
    if (UP)  g<<"14 "<<UP<<'\n';
    if (DU)  g<<"21 "<<DU<<'\n';
    if (PD)  g<<"42 "<<PD<<'\n';
    if (UUD) g<<"112 "<<UUD<<'\n';
    if (DDP) g<<"224 "<<DDP<<'\n';
    if (PPU) g<<"441 "<<PPU<<'\n';
    exit(0);
}
```

```
//Metoda 2
#include <bits/stdc++.h>
using namespace std;
ifstream f("udp.in");
ofstream g("udp.out");
int64_t U,D,P,UD,UP,DP,UUD,DDP,UPP;
void printSol()
{   if (U+D+P) {g<<"-1\n"; exit(0);}
    int64_t k=(UD>0)+(UP>0)+(DP>0)+(UUD>0)+(DDP>0)+(UPP>0);
    g<<k<<'\n';
    if (UP>0)  g<<"14 "<<UP<<'\n';
```

```
    if (UD>0)  g<<"21 "<<UD<<'\n';
    if (DP>0)  g<<"42 "<<DP<<'\n';
    if (UUD>0) g<<"112 "<<UUD<<'\n';
    if (DDP>0) g<<"224 "<<DDP<<'\n';
    if (UPP>0) g<<"441 "<<UPP<<'\n';
}
void solve2()
{   f>>U>>D>>P;
    while (U+D+P) //solutie cu eliminarea pe orice cale a unei cifre
    {   /// 1+2=12'
        if(U>=1&&D>=1){int64_t x=min(U,D);U-=x;D-=x;UD+=x;}
        /// 1+4=14
        else if(U>=1&&P>=1){int64_t x=min(U,P);U-=x;P-=x;UP+=x;}
        /// 2+4=24
        else if(D>=1&&P>=1){int64_t x=min(D,P);D-=x;P-=x;DP+=x;}
        /// 1+12=112
        else if(U>=1&&UD>=1){int64_t x=min(U,UD);U-=x;UD-=x;UUD+=x;}
        /// 2+24=224
        else if(D>=1&&DP>=1){int64_t x=min(D,DP);D-=x;DP-=x;DDP+=x;}
        /// 4+14=144
        else if(P>=1&&UP>=1){int64_t x=min(P,UP);P-=x;UP-=x;UPP+=x;}
        /// 1+224=12+24
        else if(U>=1&&DDP>=1){int64_t x=min(U,DDP);U-=x;DDP-=x;UD+=x;DP+=x;}
        /// 1+144=14+14
        else if(U>=1&&UPP>=1){int64_t x=min(U,UPP);U-=x;UPP-=x;UP+=2*x;}
        /// 2+112=12+11
        else if(D>=1&&UUD>=1){int64_t x=min(D,UUD);D-=x;UUD-=x;UD+=2*x;}
        /// 2+144=14+24
        else if(D>=1&&UPP>=1){int64_t x=min(D,UPP);D-=x;UPP-=x;UP+=x;DP+=x;}
        /// 4+112=14+12
        else if(P>=1&&UUD>=1){int64_t x=min(P,UUD);P-=x;UUD-=x;UP+=x;UD+=x;}
        /// 4+224=24+24
        else if(P>=1&&DDP>=1){int64_t x=min(P,DDP);P-=x;DDP-=x;DP+=2*x;}
        /// 1+1+24=12+14
        else if(U>=2&&DP>=1){int64_t x=min(U/2,DP);U-=2*x;DP-=x;UD+=x;UP+=x;}
        /// 4+4+12=14+24
        else if(P>=2&&UD>=1){int64_t x=min(P/2,UD);P-=2*x;UD-=x;UP+=x;DP+=x;}
        /// 2+2+14=12+24
        else if(D>=2&&UP>=1){int64_t x=min(D/2,UP);D-=2*x;UP-=x;UD+=x;DP+=x;}
        /// 12+12+4=112+24
        else if(UD>=2&&P>=1){int64_t x=min(UD/2,P);UD-=2*x;P-=x;UUD+=x;DP+=x;}
        /// 14+14+2=144+12
        else if(UP>=2&&D>=1){int64_t x=min(UP/2,D);UP-=2*x;D-=x;UPP+=x;UD+=x;}
        /// 24+24+1=224+14
        else if(DP>=2&&U>=1){int64_t x=min(DP/2,U);DP-=2*x;U-=x;DDP+=x;UP+=x;}
        /// 12+14+2=112+24
        else if(UD>=1&&UP>=1&&D>=1){int64_t x=min(min(UD,UP),D);UD-=x;UP-=x;D-=x;UUD+=x;DP+=x;}
        /// 12+24+4=224+14
        else if(UD>=1&&DP>=1&&P>=1){int64_t x=min(min(UD,DP),P);UD-=x;DP-=x;P-=x;DDP+=x;UP+=x;}
        /// 14+24+1=144+12
        else if(UP>=1&&DP>=1&&U>=1){int64_t x=min(min(UP,DP),U);UP-=x;DP-=x;U-=x;UPP+=x;UD+=x;}
        /// nu avem solutie (probabil)
        else break;
    }
    printSol();
}
int main()
{   solve2();
    return 0;
}
```

Partea a III-a

Tabăra de pregătire a lotului național de informatică juniori

Slatina, 7-14 mai 2023

Capitolul 10

Barajul 1

10.1 Problema Excursie

Propusă de: stud. Andrei Onuț, Universitatea Yale, S.U.A.,
prof. Daniela Lica, Centrul Județean de Excelență Prahova

Pe insula *Kauai*, aflată în statul Hawaii, se găsesc N sate așezate în linie dreaptă, numerotate în ordine crescătoare, începând de la 1: $1, 2, 3, \ldots, N$. În dreptul fiecărui sat i ($1 \leq i \leq N$) se află un indicator pe care poate scrie fie **R**, cu semnificația că din satul i se poate ajunge în satul $(i+1)$, fie **L**, cu semnificația că din satul i se poate ajunge în satul $(i-1)$. Se știe faptul că pe indicatorul satului 1 este scris **R**, iar pe cel al satului N este scris **L**.

Ajunși în *Kauai* pentru Q zile, tinerii JOHN și MARY își propun să facă câte o excursie în fiecare zi. În cadrul fiecărei excursii, tinerii pornesc din sate diferite, urmărind să se întâlnească, eventual, într-unul dintre cele N sate. Fiecare excursie va fi reprezentată sub forma (i, j) (unde $1 \leq i, j \leq N$ și $i \neq j$), cu semnificația: JOHN se află inițial în satul i, MARY în satul j, iar ei încep să se deplaseze între sate, respectând sensul de deplasare de pe indicatoarele satelor vizitate.

Interesant este că cei doi au posibilitatea să înlocuiască oricât de multe indicatoare întâlnesc pe parcursul traseului și să urmeze astfel noul sens de deplasare; adică, pot înlocui un indicator **R** întâlnit într-unul **L** sau pot înlocui un indicator **L** întâlnit într-unul **R** și apoi să continue deplasarea între sate. *Fiecare înlocuire a unui indicator poate fi efectuată dacă se achită o taxă în valoare de* 1 *dolar.* Pe durata excursiei, din cauza căldurii, tinerii pot alege și să staționeze în satele în care se găsesc la un moment dat (inclusiv în satele din care pornesc), adică nu este nevoie ca cei doi să se deplaseze simultan între sate pe durata întregului traseu. Important este ca cei doi, într-un final, să se întâlnească într-un sat.

Cerințe

Pentru fiecare dintre cele Q excursii, se cere să se determine **costul total minim**, exprimat în dolari, care va fi plătit pentru ca cei doi tineri să se poată întâlni într-unul dintre cele N sate, eventual după modificarea a 0 sau mai multe indicatoare din satele vizitate.

Date de intrare

Pe prima linie a fișierului de intrare `excursie.in` se află numărul natural nenul N, cu semnificația de mai sus. Pe următoarea linie se află, fără niciun spațiu între ele, N caractere ce pot fi fie **'R'**, fie **'L'**, al i-lea caracter reprezentând ce este scris inițial pe indicatorul din dreptul satului i. Pe cea de a treia linie se află numărul natural nenul Q, reprezentând numărul de excursii. Pe următoarele

Q linii se află, separate prin câte un spațiu, câte două numere naturale i și j, descriind, în ordine, excursiile tinerilor, adică satele din care pornesc cei doi în fiecare dintre cele Q zile.

Date de ieșire

Fișierul de ieșire `excursie.out` va conține Q linii, pe cea de a i-a linie aflându-se **costul total minim** necesar efectuării excursiei din ziua i, pentru fiecare i: $1 \leq i \leq Q$.

Restricții

- $2 \leq N \leq 200\,000$ și $1 \leq Q \leq 200\,000$.
- **Indicatoarele din satele** 1 **și** N **nu pot fi modificate.**
- Chiar dacă pentru efectuarea unei excursii dintre cele Q se pot modifica indicatoarele anumitor sate, în timpul nopții, înainte de începerea următoarei excursii, acestea vor reveni la starea inițială, așa cum au fost descrise în fișierul de intrare.
- Dacă pentru efectuarea unei excursii nu este necesar să fie schimbat niciun indicator, atunci costul (total) aferent acesteia va fi egal cu 0 dolari. De asemenea, în cadrul unei excursii, este permis ca indicația de pe un indicator întâlnit pe parcursul traseului să fie schimbată de mai multe ori.
- Este posibil ca pentru efectuarea unei excursii să existe mai multe modalități de a modifica indicatoarele astfel încât costul total să fie minim.

#	Puncte	Restricții
1	9	$\|i - j\| = 1$, pentru fiecare dintre cele Q excursii
2	7	Toate indicatoarele arată inițial **R**, cu excepția celui din dreptul satului N
3	6	Toate indicatoarele arată inițial **L**, cu excepția celui din dreptul satului 1
4	11	$N \leq 200$ și $Q \leq 300$
5	29	$N, Q \leq 3\,000$
6	22	$N, Q \leq 70\,000$
7	16	Nu există alte restricții suplimentare

Exemple

excursie.in	excursie.out
7 RRRLLLL 1 2 6	0
5 RRLRL 3 1 3 4 1 2 5	0 1 1

8 RLRRRLRL 4 2 4 1 6 2 8 6 2	1 1 2 1

Explicații

Primul exemplu:

- Pe insulă există $N = 7$ sate, iar inițial, conform indicatoarelor din dreptul satelor:
  - din satul 1 se poate ajunge în satul 2;
  - din satul 2 se poate ajunge în satul 3;
  - din satul 3 se poate ajunge în satul 4;
  - din satul 4 se poate ajunge în satul 3;
  - din satul 5 se poate ajunge în satul 4;
  - din satul 6 se poate ajunge în satul 5;
  - din satul 7 se poate ajunge în satul 6.
- Va fi efectuată o singură excursie ($Q = 1$): JOHN va începe din satul 2, iar MARY din satul 6. Fără a schimba niciun indicator (cost total: 0 dolari), cei doi se pot întâlni, de exemplu, în satul 3, respectând indicațiile de pe indicatoarele întâlnite. Astfel, JOHN se va deplasa din satul 2 în satul 3. De asemenea, MARY se va deplasa, pornind din satul 6, în satul 5, apoi în satul 4, iar în cele din urmă în satul 3, unde se va întâlni cu JOHN.

Al doilea exemplu:

- Pentru cea de a doua excursie $(4, 1)$, este necesar ca cel puțin un indicator să fie schimbat, cu costul de 1 dolar. De exemplu, indicatorul din dreptul satului 3 poate fi transformat din **L** în **R**.

Al treilea exemplu:

- Pentru cea de a treia excursie $(2, 8)$, este nevoie ca cel puțin două indicatoare să fie schimbate, cu costul total de 2 dolari. De exemplu, indicatoarele din dreptul satelor 2 și 6 pot fi transformate din **L** în **R**. JOHN și MARY nu se pot întâlni dacă nu schimbă cel puțin două indicatoare în cadrul acestei excursii.

10.2 Rezolvarea problemei Excursie

Observație inițială:

Fără a pierde din generalitatea rezolvării, considerăm că $x < y$, pentru fiecare dintre cele Q excursii. Fie k satul pentru care se va obține un cost total minim (în cadrul unei excursii); pot exista mai multe astfel de sate k, însă vom considera doar unul dintre ele, din moment ce toate vor rezulta în același cost total.

Observăm că: $x \leq k \leq y$ (orice sat k, cu $k < x$, va genera un cost mai mare sau egal decât satul x, de vreme ce, de exemplu, JOHN ar putea rămâne pe întreaga durată a traseului în satul x; similar raționăm și pentru orice sat k, cu $k > y$.).

Idee:

Pentru fiecare excursie ($x < y$), considerăm fiecare punct posibil de întâlnire k și calculăm costul pentru a ajunge în punctul k din x și y. Pentru a ajunge dintr-o poziție x în poziția k, unde $x < k$, trebuie ca toate caracterele din secvența $x \dots k-1$ să fie **R**. Pentru a ajunge dintr-o poziție y în poziția k, unde $y > k$, trebuie ca toate caracterele din secvența $k+1 \dots y$ să fie **L**.

Vom precalcula următorii vectori:

- $Pref[i]$- numărul de caractere **L** din secvența $1 \dots i$
- $Suff[i]$- numărul de caractere **R** din secvența $i \dots N$

Costul pentru ca o excursie $x < y$ să se întâlnească în punctul k ($x \leq k \leq y$) este:

$$cost(k) = Pref[k-1] - Pref[x-1] + Suff[k+1] - Suff[y+1]$$
$$cost(k) = (Pref[k-1] + Suff[k+1]) - (Pref[x-1] + Suff[y+1])$$

Pentru claritate, notăm cu $Value(k) = Pref[k-1] + Suff[k+1]$. Deci, $cost(k) = Value(k) - (Pref[x-1] + Suff[y+1])$.

Facem următoarele 2 observații:

1. $Value(k)$ nu depinde de valorile lui x și y.
2. $(Pref[x-1] + Suff[y+1])$ nu depinde de valoarea lui k.

Mai concret, $cost(k)$ este diferența dintre o valoare prestabilită ($Value$ nu își modifică valoarea da la o excursie la alta) și o constantă. Cum $Value$ diferă în funcție de k, răspunsul nostru este dat de valoarea minimă a lui $Value(k)$.

Astfel, problema noastră s-a redus la a afla k între x și y, pentru care $Value(k)$ este *minim*.

Există mai multe abordări pentru a afla minimul dintr-un interval.

Pentru determinarea optimă a minimului pe un interval, vom construi o matrice RMQ (Range Minimum Query), care reține minimul pe intervale de lungimi puteri de 2, după care, aflarea minimului pe orice interval (x, y) se obține în $O(1)$.

Soluție alternativă - Cătălin Frâncu

O altă variantă de calcul a RMQ, răspunde celor Q interogări în timp total $\mathcal{O}(Q \log N)$, folosind memorie suplimentară doar $\mathcal{O}(N)$ în loc de $\mathcal{O}(N \log N)$. Procedăm astfel:

1. Ordonăm excursiile descrescător după capătul stâng.
2. Trecem prin vector de la dreapta la stânga și, la fiecare poziție x, adăugăm $Value(x)$ într-o stivă de maxime ordonată descrescător. Așadar, $Value(x)$ va elimina din vârful stivei toate valorile mai mici decât ea însăși.
3. După adăugarea lui $Value(x)$ în stivă, răspundem la toate interogările care încep la poziția x. Pentru o interogare (x, y), răspunsul este cea mai mare valoare aflată în stivă pe o poziție mai mică sau egală cu y. Întrucât stiva este ordonată, putem căuta binar acea valoare.

10.3 Cod-sursă pentru problema Excursie

```
#include <bits/stdc++.h>
using namespace std;
ifstream f ("excursie.in");
```

```
ofstream g ("excursie.out");

const int NMAX = 200000;
const int QMAX = 200000;
const int LOGMAX = 20;

int N, Q;
char S[NMAX+5];
int Pref[NMAX+5];
int Suff[NMAX+5];
int Value[NMAX+5];
int Log[NMAX+5];
int RMQ[LOGMAX][NMAX+5];

void Read () {
    f >> N;
    for (int i = 1; i <= N; ++ i ) f >> S[i];
}

void Precalculare () {
    for (int i = 1; i <= N; ++ i )
        Pref[i] = Pref[i-1] + (S[i] == 'L');
    for (int i = N; i >= 1; -- i )
        Suff[i] = Suff[i+1] + (S[i] == 'R');
    for (int i = 1; i <= N; ++ i )
        Value[i] = (Pref[i-1] + Suff[i+1]);

    Log[1] = 0;
    for (int i = 2; i <= N; ++ i )
        Log[i] = Log[i/2] + 1;
    for (int i = 1; i <= N; ++ i )
        RMQ[0][i] = Value[i];

    for (int l = 1; (1<<l) <= N; ++ l )
        for (int i = 1; i + (1<<l) - 1 <= N; ++ i )
            RMQ[l][i] = min(RMQ[l-1][i], RMQ[l-1][i + (1<<(l-1))]);
}

int GasesteMinim (int x, int y) {
    int lg = Log[y - x + 1];
    return min(RMQ[lg][x], RMQ[lg][y - (1<<lg) + 1]);
}

void Solve () {
    int Q;
    f >> Q;
    for (int i = 1; i <= Q; ++ i ) {
        int x, y;
        f >> x >> y;
        if (x > y) swap(x, y);
        g << GasesteMinim(x, y) - (Pref[x-1] + Suff[y+1]) << '\n';
    }
}

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

10.4 Problema F1

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

Se dă un tablou bidimensional cu N linii și N coloane. Există Q poziții distincte, etichetate cu numere naturale distincte de la 1 la Q, unde în tablou există valoarea 1, la toate celelalte poziții din tablou există valoarea 0. Pentru o poziție oarecare, dintre cele Q date, numim „forța" acelei poziții numărul subtablourilor din tabloul dat care conțin doar o singură valoare 1, cea aflată la acea poziție, restul elementelor din subtablouri fiind egale cu 0.

Cerințe

Pentru un șir format din P etichete distincte, dintre cele corespunzătoare celor Q poziții date, se cere să se calculeze suma „forțelor" acestora.

Date de intrare

Pe prima linie a fișierului de intrare `f1.in` se găsesc numerele naturale N, Q și P, separate prin spațiu. Pe următoarele Q linii se află câte două numere naturale, separate prin spațiu, reprezentând linia și coloana pentru fiecare dintre cele Q poziții unde se află valoarea 1, în ordinea etichetelor lor. Pe următoarele P linii se află câte un număr natural, reprezentând cele P etichete ale pozițiilor pentru care trebuie calculată suma „forțelor".

Date de ieșire

Fișierul de ieșire `f1.out` va conține pe prima linie un număr natural reprezentând suma „forțelor" cerută.

Restricții

- $1 \leq P \leq Q \leq 1\,000$.
- $1 \leq N \leq 5\,000$.
- Liniile și coloanele tabloului sunt numerotate cu $1, 2, \ldots, N$.

#	Puncte	Restricții
1	2	$Q = 1$
2	18	$Q \leq 5$
3	7	Cele $Q \leq 200$ poziții unde există valoarea 1 sunt ordonate strict crescător după linii și după coloane
4	7	Cele $Q \leq 200$ poziții unde există valoarea 1 sunt situate pe aceeași linie
5	7	$N \leq 10, Q \leq 200$
6	12	$N \leq 300, Q \leq 200$
7	23	$N \leq 500, Q \leq 200$
8	8	$Q \leq 200$
9	16	Nu există alte restricții suplimentare

Exemple

f1.in	f1.out
4 1 1 2 2 1	36
4 3 2 2 2 3 4 4 1 3 1	33

Explicații

În primul exemplu avem un tablou de dimensiune 4x4 și valoarea 1 la poziția (2,2). Subtablourile care conțin poziția (2,2) au colțul stânga-sus în zona delimitată de liniile 1 și 2, respectiv coloanele 1 și 2, iar colțul dreapta-jos în zona delimitată de liniile 2 și 4, respectiv coloanele 2 și 4. În total 36 subtablouri.

În al doilea exemplu avem un tablou de dimensiune 4x4 și valoarea 1 la pozițiile (2,2), (3,4) și (4,1). Trebuie calculate forțele pozițiilor cu etichetele 3 și 1, deci ale pozițiilor (4,1) și (2,2). Subtablourile care conțin doar poziția (4,1) sunt: cele cu colțul stânga sus în pozițiile (1,1) sau (2,1) și colțul dreapta jos în poziția (4,1); cele cu colțul stânga sus în (3,1) sau (4,1) și colțul dreapta jos în (4,1), (4,2) sau (4,3); cea cu colțul stânga sus în (4,1) și colțul dreapta jos în (4,4). Astfel forța poziției (4,1) este 2+6+1=9. Subtablourile care conțin doar poziția (2,2) sunt: cele cu colțul stânga sus în pozițiile (1,1), (1,2), (2,1) sau (2,2) și colțul dreapta jos în pozițiile (2,2), (2,3), (2,4), (3,2) sau (3,3); cele cu colțul stânga sus în (1,2) sau (2,2) și colțul dreapta jos în (4,2) sau (4,3). Forța poziției (2,2) este egală cu 20+4=24. Suma celor două forțe este 33.

10.5 Rezolvarea problemei F1

Soluție în $\mathcal{O}(N^6)$:

Un algoritm forță brută stochează efectiv matricea de $N \times N$ elemente. Apoi iterează prin toate cele $\mathcal{O}(N^4)$ combinații de colțuri stânga-sus și dreapta-jos ale unei submatrice. Pentru o submatrice fixată, algoritmul calculează în $\mathcal{O}(N^2)$ suma submatricii. Dacă suma este 1, atunci matricea contribuie la puterea punctului pe care îl conține.

Dar trebuie să răspundem la întrebarea: despre care punct este vorba? Pentru a determina în $\mathcal{O}(1)$ punctul conținut, putem folosi un artificiu. În loc să stocăm în matrice valori 1, stocăm valori distincte pentru fiecare punct: 1 000, 1 001, 1 002... Atunci suma unei submatrice va fi $S \in [1\,000, 1\,999]$ dacă și numai dacă ea conține exact punctul $S - 1\,000$.

Soluție în $\mathcal{O}(N^4)$:

Putem îmbunătăți soluția de mai sus dacă stocăm sume parțiale în locul matricei propriu-zise. Atunci putem calcula suma oricărei submatrice în $\mathcal{O}(1)$.

Soluție în $\mathcal{O}(N^3)$:

Continuăm să îmbunătățim soluția de mai sus. Orice submatrice are o primă și o ultimă coloană. Să iterăm prin toate perechile de coloane (S, D) și să aflăm toate submatricile care se întind de

la coloana S și până la coloana D și care conțin exact un punct. Colectăm, pentru fiecare dintre cele N linii, numărul de puncte pe acea linie între coloanele S și D. Putem face acest lucru în $\mathcal{O}(1)$ per linie, fie folosind matricea de sume parțiale, fie actualizând aceste valori pe măsură ce coloana dreaptă D avansează spre dreapta.

Odată obținută lista de linii care conțin puncte, pentru fiecare punct X care este singur pe linie și care se numără printre cele P puncte de interes, puterea lui X crește cu $L_1 \times L_2$, unde L_1 și L_2 reprezintă distanțele până la precedenta, respectiv următoarea linie care conține puncte. Aceasta deoarece un dreptunghi care îl conține pe X poate să înceapă oriunde între linia anterioară care conține puncte (exclusiv) și linia lui X (inclusiv).

Soluție în $\mathcal{O}(Q \cdot N^2)$:

Soluția de mai sus poate fi îmbunătățită. Pentru o pereche de coloane (S, D) nu este nevoie să iterăm prin toate liniile matricei. Putem să iterăm prin cele Q puncte și să le colectăm pe cele a căror coloană este cuprinsă între S și D. Având în vedere că la fiecare colectare dorim punctele sortate după linie, le sortăm o singură dată imediat după citirea datelor. Apoi, ca mai sus, puterea fiecărui punct crește cu $L_1 \times L_2$, unde L_1 și L_2 sunt distanțele pe linie până la punctele anterior și următor.

Remarcăm că acest algoritm tratează corect, fără cod suplimentar, cazul în care mai multe puncte se află pe aceeași linie. Ele vor fi consecutive în lista ordonată și toate vor avea putere 0, căci fie L_1, fie L_2 vor fi 0.

Un beneficiu suplimentar al acestei soluții este că nu mai construiește efectiv matricea, deci consumul de memorie este $\mathcal{O}(Q)$.

Soluție în $\mathcal{O}(Q^2 \cdot N)$:

Față de algoritmul anterior facem următoarea observație. Pentru o coloană S fixată, când avansăm de la coloana D la $D+1$, dacă pe coloana $D+1$ nu se află niciun punct, atunci puterea totală pe intervalul $[S, D+1]$ va fi egală cu puterea pe intervalul $[S, D]$. De ce? Să ne amintim că algoritmul colectează punctele de pe coloane aflate în acel interval. Dacă coloana $D+1$ este goală, atunci lista de puncte colectate nu se modifică, deci implicit nici puterea totală nu se modifică.

Rezultă că D nu trebuie să itereze prin toate cele (maximum) N coloane, ci doar prin cele Q care conțin puncte. Odată calculată puterea pentru o pereche de coloane (S, D), o putem multiplica cu C, unde C este distanța de la D până la următoarea coloană care conține puncte.

Soluție în $\mathcal{O}(Q^3)$:

Putem proceda pentru coloana S exact cum procedăm în soluția anterioară pentru coloana D. Așadar, S și D iterează prin cele Q puncte. Odată calculată puterea unui interval $[S, D]$, o multiplicăm cu $C_1 \times C_2$, unde C_1 și C_2 sunt distanțele până la punctul anterior lui S, respectiv următor lui D.

Soluție în $\mathcal{O}(P \cdot Q^2)$ (Cătălin Frâncu):

Putem calcula explicit puterea fiecărui punct în $\mathcal{O}(Q^2)$ per punct. Grupăm punctele în benzi, după coloană. Apoi fixăm un punct X și iterăm prin perechi de benzi (i, j). Pentru fiecare pereche ne întrebăm: câte submatrice conțin doar punctul X, au colțul stânga-sus între benzile i și $i+1$ și au colțul dreapta-jos între benzile j și $j+1$?

Putem răspunde în $\mathcal{O}(1)$ la fiecare întrebare, calculând răspunsul pentru perechea $(i, j+1)$ din răspunsul la perechea (i, j). Menținem întinderea pe verticală a acestor dreptunghiuri. Inițial, pentru $i = j =$ banda din care face parte X, întinderea este dată de punctul anterior și următor lui X în banda sa (sau 0, respectiv $N+1$ dacă punctul X este primul, respectiv ultimul în banda sa).

Cînd trecem la următorul j, evaluăm toate punctele din noua bandă și, dacă ele sînt cuprinse în întinderea curentă, restrîngem această întindere.

Soluție în $\mathcal{O}(Q^2 \log Q)$ (Cătălin Frâncu):

Soluția în $\mathcal{O}(Q^3)$ este ineficientă deoarece, pentru fiecare pereche de puncte (S, D), ea colectează naiv, în $\mathcal{O}(Q)$, punctele dintre acele coloane. Dar dacă am putea să menținem interactiv această colecție, cu efort doar $\mathcal{O}(\log Q)$ pentru a trece de la punctul D la $D+1$? Dorim să efectuăm eficient operațiile:

1. Actualizare: Adaugă un punct la colecție.
2. Interogare: Care este puterea colecției?

(**Notă**: Pentru brevitate spunem „puterea colecției". Formularea completă ar fi „contribuția acestei colecții, dintre coloanele S și D, la puterile punctelor pe care le conține și care sunt și printre cele P puncte de interes".)

Cum recalculăm puterea? Să considerăm o colecție cu 4 puncte pe liniile A, B, C și D. Atunci contribuția lui B este $(B-A)(C-B)$, iar contribuția lui C este $(C-B)(D-C)$. Acum să considerăm că adăugăm un punct pe linia X, iar colecția devine A, B, X, C, D. Trebuie să scădem vechile contribuții ale lui B și C și să adăugăm noile contribuții ale lui B, X și C. Puterile celorlalte puncte din colecție nu se modifică, deci putem recalcula puterea în timp constant.

Rezultă că trebuie să putem accesa în $\mathcal{O}(\log Q)$ punctele anterior și următor unui punct dat. Putem implementa această colecție cu diverse structuri de date. Soluția propusă sortează punctele după linie. Apoi menține un arbore Fenwick (AIB) de 0 și 1 în care bifează cu 1 indicii punctelor din colecția prezentă.

La adăugarea unui punct în colecție, să zicem cel cu indicele 17, vom interoga AIB-ul pentru suma până la poziția 17. Să presupunem că primim răspunsul 8, deci punctul tocmai inserat este al 8-lea în ordine în colecția curentă. Atunci putem căuta binar în AIB poziția pe care se ating sumele 7 sau 9 pentru a afla pozițiile elementului anterior sau următor.

Remarcăm că o căutare binară în AIB, implementată naiv, va face $\log Q$ calculări de sumă, fiecare cu costul $\mathcal{O}(\log Q)$, deci va avea complexitatea $\mathcal{O}(\log^2 Q)$. Dar ea poate fi implementată și în $\mathcal{O}(\log Q)$. Detalii aici.

10.6 Cod-sursă pentru problema F1

```
// Complexitate: O(Q^2 log Q). Catalin Francu
#include <algorithm>
#include <stdio.h>
#define MAX_POINTS 4000
#define NONE -1

typedef struct {
  int row, col;
  bool is_query;
} point;
```

```
typedef struct {
  int fen[MAX_POINTS + 1];
  int max_p2;
  int size;
  long long power;
} collection;

point p[MAX_POINTS + 2];
int ord[MAX_POINTS + 2];
collection c;
int side, num_points;

void read_input_data() {
  int num_queries;
  FILE* f = fopen("f1.in", "r");

  fscanf(f, "%d %d %d", &side, &num_points, &num_queries);
  for (int i = 1; i <= num_points; i++) {
    fscanf(f, "%d %d", &p[i].row, &p[i].col);
    p[i].is_query = false;
  }

  while (num_queries--) {
    int pos;
    fscanf(f, "%d", &pos);
    p[pos].is_query = true;
  }

  fclose(f);
}

void sort_points_by_line() {
  std::sort(p + 1, p + num_points + 1, [] (point& a, point& b) {
    return (a.row < b.row);
  });
}

void place_sentinels() {
  p[0].col = 0;
  p[num_points + 1].col = side + 1;
}

void sort_points_by_column() {
  for (int i = 0; i <= num_points + 1; i++) {
    ord[i] = i;
  }
  place_sentinels();
  std::sort(ord, ord + num_points + 2,
            [] (int a, int b) {
              return (p[a].col < p[b].col);
            });
}

void collection_compute_max_p2() {
  c.max_p2 = num_points;
  while (c.max_p2 & (c.max_p2 - 1)) {
    c.max_p2 &= c.max_p2 - 1;
  }
}

void collection_fenwick_inc(int pos) {
```

```
  do {
    c.fen[pos]++;
    pos += pos & -pos;
  } while (pos <= num_points);
}

int collection_fenwick_sum(int pos) {
  int s = 0;
  while (pos) {
    s += c.fen[pos];
    pos &= pos - 1;
  }
  return s;
}

// Returnează poziția la care suma parțială >= @sum.
// Contract: sum > 0.
int collection_fenwick_bin_search(int sum) {
  int pos = 0;

  for (int interval = c.max_p2; interval; interval >>= 1) {
    if ((pos + interval <= num_points) && (c.fen[pos + interval] < sum)) {
      sum -= c.fen[pos + interval];
      pos += interval;
    }
  }

  return pos + 1;
}

void collection_fenwick_wipe() {
  for (int i = 1; i <= num_points; i++) {
    c.fen[i] = 0;
  }
}

void collection_init() {
  c.size = c.power = 0;
  collection_compute_max_p2();
}

void collection_wipe() {
  c.size = c.power = 0;
  collection_fenwick_wipe();
}

int collection_get_neighbor(int self, int delta, int limit) {
  int sum = collection_fenwick_sum(self);

  if (sum == limit) {
    return NONE;
  } else {
    return collection_fenwick_bin_search(sum + delta);
  }
}

int collection_get_previous_point(int self) {
  return collection_get_neighbor(self, -1, 1);
}

int collection_get_next_point(int self) {
```

```
  return collection_get_neighbor(self, +1, c.size);
}

void collection_update_power(int prev, int self, int next, int sign) {
  if (p[self].is_query) {
    int x = (prev == NONE) ? 0 : p[prev].row;
    int y = p[self].row;
    int z = (next == NONE) ? (side + 1) : p[next].row;

    c.power += sign * (y - x) * (z - y);
  }
}

void collection_add_power(int prev, int self, int next) {
  collection_update_power(prev, self, next, +1);
}

void collection_subtract_power(int prev, int self, int next) {
  collection_update_power(prev, self, next, -1);
}

void collection_add_point(int self) {
  collection_fenwick_inc(self);
  c.size++;
  int prev = collection_get_previous_point(self);
  int next = collection_get_next_point(self);

  if (prev != NONE) {
    int prev2 = collection_get_previous_point(prev);
    collection_subtract_power(prev2, prev, next);
    collection_add_power(prev2, prev, self);
  }
  if (next != NONE) {
    int next2 = collection_get_next_point(next);
    collection_subtract_power(prev, next, next2);
    collection_add_power(self, next, next2);
  }
  collection_add_power(prev, self, next);
}

long long process_all_column_ranges() {
  long long total_power = 0;
  for (int i = 1; i <= num_points; i++) {
    int left = ord[i], prev = ord[i - 1];
    if (p[left].col != p[prev].col) {
      for (int j = i; j <= num_points; j++) {
        int right = ord[j], next = ord[j + 1];
        collection_add_point(right);
        if (p[right].col != p[next].col) {
          total_power +=
            c.power *
            (p[left].col - p[prev].col) *
            (p[next].col - p[right].col);
        }
      }
      collection_wipe();
    }
  }
  return total_power;
}
```

```c
void write_answer(long long answer) {
  FILE* f = fopen("f1.out", "w");
  fprintf(f, "%lld\n", answer);
  fclose(f);
}

int main() {
  read_input_data();
  sort_points_by_line();
  sort_points_by_column();
  collection_init();

  long long total_power = process_all_column_ranges();
  write_answer(total_power);
  return 0;
}
```

10.7 Problema Pcp

Propusă de: prof. Ciprian Chește, Liceul Tehnologic „Grigore C. Moisil" Buzău

Se consideră un număr natural nenul N.

Cerințe

Să se determine toate pătratele perfecte distincte care se obțin prin permutarea cifrelor numărului N.
Exemple:

- Pentru $N = 691$ prin permutarea cifrelor sale se obțin numerele $169, 196, 619, 691, 916, 961$ din care 3 sunt pătrate perfecte: $169 = 13^2, 196 = 14^2$ și $961 = 31^2$.
- Pentru $N = 1044$ prin permutarea cifrelor sale se obțin numerele 0144, 0414, 0441, 1044, 1404, 1440, 4014, 4041, 4104, 4140, 4401, 4410 din care 2 sunt pătrate perfecte: $144 = 12^2, 441 = 21^2$.

Date de intrare

Fișierul de intrare `pcp.in` conține pe primul rând numărul natural N.

Date de ieșire

Fișierul de ieșire `pcp.out` va conține pe prima linie numărul P reprezentând numărul de pătrate perfecte care se pot obține prin permutarea cifrelor numărului N și apoi pe următoarele P linii cele P pătrate perfecte în ordine crescătoare.
Dacă nu există niciun pătrat perfect ce se poate obține prin permutarea cifrelor lui N se va afișa mesajul *nu exista*.

Restricții

- $0 < N < 10^{14}$
- Cifrele de 0 care prin permutarea cifrelor lui N apar la începutul unui număr, nu se iau în considerare.

#	Puncte	Restricții
1	20	$1 \leq N < 10^4$
2	68	$10^4 \leq N < 10^{13}$
3	12	$10^{13} \leq N < 10^{14}$

Exemple

pcp.in	pcp.out
691	3 169 196 961
1044	2 144 441

202050	4 225 2025 22500 202500
2023	nu exista

10.8 Rezolvarea problemei Pcp

Brute force

O primă abordare, nu tocmai eficientă, ar putea consta în construirea tuturor permutărilor cifrelor numărului N și verificarea proprietății ca numărul obținut să fie pătrat perfect. Această abordare presupune utilizarea unui mecanism de tip backtracking sau un algoritm echivalent pentru generarea tuturor permutărilor și care poate fi optimizat prin vectori de frecvență a cifrelor.

Soluție eficientă

O soluție eficientă constă în generarea tuturor pătratelor perfecte mai mici decât N. Pentru fiecare pătrat perfect generat se va verifica dacă cifrele sale coincid cu cifrele numărului N cu excepția cifrelor de 0. Având în vedere că se cere și numărul acestora, numerele cu proprietatea cerută trebuie memorate într-o structură de date adecvată(vector) și afișate ulterior fără a fi sortate deoarece au fost generate în ordine crescătoare.
Ordinul de complexitate al soluției este $O(\sqrt{N} \cdot \log N)$.

Soluție eficientă optimizată - prof. Adrian Panaete, stud. Gabriel Theodor Tulba-Lecu

Se pot realiza următoarele optimizări:

- Se va construi un vector de frecvențe, pe care-l vom nota cu FR pentru cifrele nenule disponibile generând succesiv toate pătratele $\leq VALMAX$ și verificăm la fiecare pas dacă vectorul de frecvențe al pătratului la care am ajuns coincide cu vectorul FR, caz în care am descoperit o soluție.
- În locul efectuării înmulțirilor, pentru a genera pătratele perfecte se pot utiliza adunări folosind ideea ca $n^2 = 1 + 3 + 5 + \cdots + (2n - 1)$
- Deoarece frecvența unei cifre este cel mult 14, putem simula vectorul de frecvențe utilizând un număr pe 64 de biți.
  Astfel fiecărei cifre i se alocă câte 4 biți:
  pentru cifra 1 - primii 4 biți,
  pentru cifra 2 - următorii 4 biți,
  ...
  pentru cifra 9 - ultimii 4 biți.
  În total 36 de biți.
  Incrementarea frecvenței cifrei 1 se va realiza prin adunare cu 16^0,
  Incrementarea frecvenței cifrei 2 se va realiza prin adunare cu 16^1,
  ...
  Incrementarea frecvenței cifrei 9 se va realiza prin adunare cu 16^8,
  Decrementarea frecvenței cifrelor se va face similar (frecvența cifrei C se realizează scăzând din numărul pe 64 de biți valoarea 16^{C-1}

- Cât timp verificăm un pătrat generat cu valoarea mai mică decât $VALMAX$ ($\leq$ numărul maxim ce se poate forma cu cifrele disponibile), frecvența cifrei 0 nu ne interesează.
- Vectorul de frecvente al unei valori $(k+1)^2$ poate fi „recalculat” din vectorul de frecvențe al lui k^2 astfel: Parcurgem succesiv și simultan cifrele celor două pătrate începând de la cifra unităților și continuând cu cifra zecilor, a sutelor, etc.
  La fiecare ordin scădem frecvența cifrei la care am ajuns în k^2 și o creștem pe cea de la $(k+1)^2$ și eliminăm cifrele respective. În acest mod, în momentul în care cele două valori obținute coincid am realizat transformarea vectorului de frecvențe deoarece cifrele rămase neprocesate din cele două valori k^2 și $(k+1)^2$ coincid.

Soluție de 100 puncte - prof. Mihai Bunget

Se determină cifrele numărului N și se află cel mai mic număr (A) și cel mai mare număr (B) care se pot forma cu cifrele lui N. Se parcurg valorile de la $[\sqrt{A}]$ la $[\sqrt{B}]$, iar pentru fiecare valoare x se determină cifrele lui $x \cdot x$ și, folosind vectori de frecvență, se verifică dacă acestea coincid cu cifrele lui N. Optimizarea se produce dacă observăm că în funcție de restul împărțirii lui N la 9 putem alege să verificăm numai anumite valori ale lui x. Astfel, dacă restul R al împărțirii lui N la 9 este 0 trebuie să parcurgem doar numerele x divizibile cu 3, dacă R este 1 parcurgem numerele x ce dau restul 1 sau 8 la împărțirea cu 9, dacă R este 4 parcurgem numerele x ce dau restul 2 sau 7 la împărțirea cu 9, iar dacă R este 7 parcurgem numerele x ce dau restul 4 sau 5 la împărțirea cu 9. Aceasta se explică prin faptul că restul împărțirii unui pătrat perfect la 9 poate fi doar 0,1,4 sau 7, iar acest pătrat perfect trebuie să aibă aceleași cifre cu N deci ambele vor da același rest la împărțirea cu 9. Matematic restul împărțirii unui număr la 9 este egal cu restul împărțirii sumei cifrelor sale la 9. Pentru afișarea soluțiilor în ordine crescătoare se sortează vectorul soluțiilor.

Restul împărțirii unui pătrat perfect la 9 poate fi doar 0,1,4 sau 7. Deci dacă N nu dă unul din aceste resturi la împărțirea cu 9 se va afișa nu există. Altfel, se va verifica dacă un pătrat perfect are aceleași cifre cu N dar și același rest la împărțirea cu 9.

10.9 Cod-sursă pentru problema Pcp

```cpp
//prof. Adrian Panaete, student Gabriel Theodor Tulba-Lecu
#include <chrono>
#include <cstdio>
#include <cstring>
#include <random>
#include <vector>
using namespace std;

long long n, max_n, cnt0 = 0, residue;
long long cnt;
long long ans[500];
char fr_n[10];
char fr_k[10];

vector<vector<int>> visited_residues = {{0, 3, 6}, {1, 8}, {}, {}, {2, 7}, {}, {}, {4, 5}, {}};

void compute_fr(long long n, char *fr) {
    while (n) {
        fr[n % 10]++;
        n /= 10;
    }
}

long long compute_max_n(char *fr) {
```

```
    long long r = 0;
    for (int i = 9; i >= 0; i--) {
        for (int j = 0; j < fr[i]; j++) {
            r = r * 10 + i;
        }
    }
    return r;
}

void clear_fr(char *fr) { memset(fr, 0, 10 * sizeof(char)); }

bool compare_fr(char *fr1, char *fr2) {return memcmp(fr1+1, fr2+1, 9*sizeof(char))==0;}

int main() {
    FILE *file_in = fopen("pcp.in", "r");
    FILE *file_out = fopen("pcp.out", "w");
    fscanf(file_in, "%lld\n", &n);
    residue = n % 9;
    if (visited_residues[residue].size() == 0) {
        fprintf(file_out, "nu exista");
        return 0;
    }

    compute_fr(n, fr_n);
    max_n = compute_max_n(fr_n);
    for (long long j = 0; j * j <= max_n; j += 9) {
        for (auto &r : visited_residues[residue]) {
            clear_fr(fr_k);
            compute_fr((j + r) * (j + r), fr_k);
            if (compare_fr(fr_n, fr_k)) {
                ans[cnt++] = (j + r) * (j + r);
            }
        }
    }

    if (cnt == 0) {
        fprintf(file_out, "nu exista\n");
        return 0;
    }
    fprintf(file_out, "%lld\n", cnt);
    for (int i = 0; i < cnt; i++) {
        fprintf(file_out, "%lld\n", ans[i]);
    }
    return 0;
}
```

```
//prof. Mihai Bunget
#include <fstream>
#include <algorithm>
#include <math.h>
using namespace std;
ifstream f("pcp.in");
ofstream g("pcp.out");

long long n,m,pp,x,i,j,a,ok,sol,r,u,b[100000],a1,m1,rest,pas;
int v[10],w[10],z[10];

void calcul()
{
    for(i = a1; i <= a; i = i+pas)
    {
        for(u = 0; u <= 9; u++)
```

```
            w[u] = 0;
        x = i*i;
        pp = x;
        while(x != 0)
        {
            r = x%10;
            w[r]++;
            x = x/10;
        }
        ok = 1;
        for(u = 1; u <= 9; u++)
            if(w[u]!=v[u]) ok = 0;
        if(w[0]>v[0]) ok = 0;
        if(ok==1)
        {
            sol++;
            b[sol] = pp;
        }
    }
}

int main()
{   f >> n;
    while(n!=0)
    {
        r = n%10;
        v[r]++;
        w[r]++;
        z[r]++;
        rest += r;
        n = n/10;
    }
    rest = rest%9;
    m = 0;
    for(i = 9; i >= 0; i--)
        while(w[i]>0)
        {
            m = m*10+i;
            w[i]--;
        }
    m1 = 0;
    for(i = 1; i <= 9; i++)
        while(z[i]>0)
        {
            m1 = m1*10+i;
            z[i]--;
        }
    a = floor(sqrt(m));
    a1 = floor(sqrt(m1));
    sol = 0;
    if(rest==0)
    {
        if(a1%3!=0)
            a1 = a1+(3-a1%3);
        pas = 3;
        calcul();
    }
    else
    if(rest==4)
    {
        if(a1%9==2)
```

```
        {
            pas = 9;
            calcul();
            a1 += 5;
            calcul();
        }
        else
        if(a1%9==7)
        {
            pas = 9;
            calcul();
            a1 += 4;
            calcul();
        }
        else
        {
            if(a1%9 < 2)
            {
                a1 += 2-a1%9;
                pas = 9;
                calcul();
                a1 += 5;
                calcul();
            }
            else
            if(a1%9 < 7)
            {
                a1 += 7-a1%9;
                pas = 9;
                calcul();
                a1 += 4;
                calcul();
            }
            else
            {
                a1 += 3;
                pas = 9;
                calcul();
                a1 += 5;
                calcul();
            }
        }
    }
    else
    if(rest==7)
    {
         if(a1%9==4)
        {
            pas = 9;
            calcul();
            a1++;
            calcul();
        }
        else
        if(a1%9==5)
        {
            pas = 9;
            calcul();
            a1 += 8;
            calcul();
        }
```

```
        else
        {
            while(a1%9!=4)a1++;
            pas = 9;
            calcul();
            a1++;
            calcul();
        }
    }
    else
    {
       if(a1%9==0)
       {
           a1++;
           pas = 9;
           calcul();
           a1 += 7;
           calcul();
       }
       else
       if(a1%9==1)
       {
           pas = 9;
           calcul();
           a1 += 7;
           calcul();
       }
       else
       if(a1%9 < 8)
       {
           a1 += 8-a1%9;
           pas = 9;
           calcul();
           a1 += 2;
           calcul();
       }
       else
       {
           pas = 9;
           calcul();
           a1 += 2;
           calcul();
       }
    }
    if(sol==0) g << "nu exista\n";
    else{
    sort(b+1,b+sol+1);
    g << sol << "\n";
    for(i = 1; i <= sol; i++)  g << b[i] << "\n";
    }
    return 0;
}
```

Capitolul 11

Barajul 2

11.1 Problema Ashima

Propusă de: prof. Bunget Mihai, Colegiul Național „Tudor Vladimirescu" Târgu Jiu

Se dă un șir A, ale cărui elemente sunt definite prin relația $A_i = i^K \cdot 2^i$, pentru orice $1 \leq i$, unde K este un număr natural dat. Elementele acestui șir se așează într-o matrice M, formată din L linii și C coloane, astfel: $M_{11} = A_1, M_{21} = A_2, M_{12} = A_3, M_{31} = A_4, M_{22} = A_5, M_{13} = A_6, M_{41} = A_7, M_{32} = A_8, \ldots$, adică parcurgând matricea pe diagonale din stânga-jos spre dreapta-sus.

De exemplu, pentru $K = 0$, $L = 3$ și $C = 4$, șirul A este format din elementele $2, 4, 8, 16, 32, 64, \ldots$, iar matricea M va fi completată astfel:

2	8	64	512
4	32	256	2048
16	128	1024	4096

Cerințe

Ashima vă cere să răspundeți la Q cerințe de forma:

- $l_1\ l_2\ c_1\ c_2$: care este suma elementelor $M_{i,j}$ din matricea M astfel încât $l_1 \leq i \leq l_2$ și $c_1 \leq j \leq c_2$?

Date de intrare

Pe prima linie a fișierului de intrare `ashima.in` se află numerele K, L, C și Q, iar pe următoarele Q linii se află câte patru numere $l_1\ l_2\ c_1\ c_2$.

Date de ieșire

În fișierul de ieșire `ashima.out` se vor afișa Q linii. Pe linia i se va afișa rezultatul celei de-a i-a cerințe, modulo $1\,000\,000\,007$.

Restricții

- $0 \leq K \leq 3$
- $1 \leq L, C \leq 100\,000$

- $1 \leq Q \leq 200$
- $1 \leq l_1 \leq l_2 \leq L$
- $1 \leq c_1 \leq c_2 \leq C$
- $0 \leq l_2 - l_1,\ c_2 - c_1 \leq 1\,000$

#	Puncte	Restricții
1	16	$1 \leq L, C \leq 100$
2	21	$1 \leq L, C \leq 1\,000$
3	27	K=0
4	15	K=1
5	12	K=2
6	9	K=3

Exemple

ashima.in	ashima.out
0 3 4 3 1 1 2 4 1 2 1 3 1 3 2 3	584 366 1512
1 2 5 2 1 1 2 4 1 2 1 3	1080 642

Explicații

- Pentru primul exemplu matricea M se completează ca în exemplul dat în enunț.
  La prima cerință trebuie calculată suma elementelor aflate între liniile 1 și 1 și coloanele 2 și 4. Suma acestor elemente este $8 + 64 + 512 = 584$.
  La a doua cerință trebuie calculată suma elementelor aflate între liniile 1 și 2 și coloanele 1 și 3. Suma acestor elemente este $2 + 8 + 64 + 4 + 32 + 256 = 366$.
  La a treia cerință trebuie calculată suma elementelor aflate între liniile 1 și 3 și coloanele 2 și 3. Suma acestor elemente este $8 + 64 + 32 + 256 + 128 + 1024 = 1512$.
- Pentru al doilea exemplu avem $K = 1$, deci șirul A are elementele $1 \cdot 2^1,\ 2 \cdot 2^2,\ 3 \cdot 2^3,\ \ldots,\ 10 \cdot 2^{10}$, iar matricea M, cu 2 linii și 5 coloane, se completează astfel:

2	24	160	896	4608
8	64	384	2048	10240

  La prima cerință trebuie calculată suma elementelor aflate între liniile 1 și 1 și coloanele 2 și 4. Suma acestor elemente este $24 + 160 + 896 = 1080$.
  La a doua cerință trebuie calculată suma elementelor aflate între liniile 1 și 2 și coloanele 1 și 3. Suma acestor elemente este $2 + 24 + 160 + 8 + 64 + 384 = 642$.

Dacă ați reușit să răspundeți la cerințe, șirul ***A shi*** matricea ***Ma*** vă mulțumesc pentru ajutor!

11.2 Rezolvarea problemei Ashima

Soluție brută 16p.

Se parcurge matricea M pe diagonale, din stânga-jos spre dreapta-sus, completând-o cu elementele șirului A. Apoi, pentru fiecare cerință, se calculează suma elementelor submatricii determinate de liniile l_1 și l_2, respectiv coloanele c_1 și c_2.

Complexitate $O(L \cdot C + Q \cdot L \cdot C)$

Soluție brută cu sume parțiale 37p.

Se parcurge matricea M pe diagonale, din stânga-jos spre dreapta-sus, completând-o cu elementele șirului A. Apoi se calculează sumele parțiale pentru matricea M folosind recurențele $SC_j = SC_j + M_{i,j},\ M_{i,j} = M_{i,j-1} + SC_j$, pentru fiecare $i \in \{1, 2, \ldots, L\},\ j \in \{1, 2, \ldots, C\}$, unde SC_j este suma elementelor pe coloana j până la poziția curentă. Calculul sumei elementelor submatricii determinate de liniile l_1 și l_2, respectiv coloanele c_1 și c_2, se face cu formula $M_{l_2,c_2} - M_{l_2,c_1-1} - M_{l_1-1,c_2} + M_{l_1-1,c_1-1}$.

Complexitate $O(L \cdot C + Q)$

Soluție optimă 100p.

Pentru a răspunde la o cerință, se determină pentru fiecare diagonală stânga-jos dreapta-sus a submatricii determinate de liniile l_1 și l_2, respectiv coloanele c_1 și c_2, suma elementelor de pe această diagonală. Pentru aceasta se determină poziția în șirul A a primului și ultimului element de pe diagonală. Pentru a determina poziția elementului $M_{i,j}$ în șirul A, se află câte elemente sunt situate pe diagonalele anterioare ale matricei până la poziția acestui element. Să notăm cu h, respectiv u, pozițiile primului, respectiv ultimului element de pe o diagonală în șirul A. Atunci suma elementelor de pe această diagonală a submatricii va fi $S_u - S_{h-1}$, unde am notat cu S_n suma primelor n elemente din șirul A.

În funcție de valoarea exponentului K, valoarea lui S_n este:

- pentru $K = 0$ avem $S_n = 2^{n+1} - 2$;
- pentru $K = 1$ avem $S_n = 2^{n+1} \cdot (n-1) + 2$;
- pentru $K = 2$ avem $S_n = 2^{n+1} \cdot (n^2 - 2 \cdot n + 3) - 6$;
- pentru $K = 3$ avem $S_n = 2^{n+1} \cdot (n^3 - 3 \cdot n^2 + 9 \cdot n - 13) + 26$.

Calculul acestor sume se poate face astfel:

- Pentru $K = 0$ avem $S_n = 2 + 2^2 + 2^3 + \cdots + 2^n$. Înmulțind această relație cu 2 obținem $2 \cdot S_n = 2^2 + 2^3 + \cdots + 2^n + 2^{n+1}$. Scăzând din aceasta relația inițială obținem $S_n = 2^{n+1} - 2$.
- Pentru $K = 1$ avem $S_n = 1 \cdot 2 + 2 \cdot 2^2 + 3 \cdot 2^3 + \cdots + n \cdot 2^n$. Înmulțind această relație cu 2 obținem $2 \cdot S_n = 1 \cdot 2^2 + 2 \cdot 2^3 + \cdots + (n-1) \cdot 2^n + n \cdot 2^{n+1}$. Scăzând din aceasta relația inițială obținem $S_n = n \cdot 2^{n+1} - 2 - 2^2 - \cdots - 2^n = n \cdot 2^{n+1} - (2^{n+1} - 2) = 2^{n+1} \cdot (n-1) + 2$.
- Pentru $K = 2$ avem $S_n = 1^2 \cdot 2 + 2^2 \cdot 2^2 + 3^2 \cdot 2^3 + \cdots + n^2 \cdot 2^n$. Înmulțind această relație cu 2 obținem $2 \cdot S_n = 1^2 \cdot 2^2 + 2^2 \cdot 2^3 + \cdots + (n-1)^2 \cdot 2^n + n^2 \cdot 2^{n+1}$. Scăzând din aceasta relația inițială obținem $S_n = n^2 \cdot 2^{n+1} - 1 \cdot 2 - 3 \cdot 2^2 - \cdots - (2 \cdot n - 1) \cdot 2^n = n^2 \cdot 2^{n+1} - 2 \cdot (1 \cdot 2 + 2 \cdot 2^2 + \cdots + n \cdot 2^n) + 2 + 2^2 + \cdots + 2^n = n^2 \cdot 2^{n+1} - 2 \cdot (2^{n+1} \cdot (n-1) + 2) + (2^{n+1} - 2) = 2^{n+1} \cdot (n^2 - 2 \cdot n + 3) - 6$.
- Pentru $K = 3$ avem $S_n = 1^3 \cdot 2 + 2^3 \cdot 2^2 + 3^3 \cdot 2^3 + \cdots + n^3 \cdot 2^n$. Putem scrie $S_n = 2 \cdot S_n - S_n = 1^3 \cdot 2^2 + 2^3 \cdot 2^3 + 3^3 \cdot 2^4 + \cdots + n^3 \cdot 2^{n+1} - (1^3 \cdot 2 + 2^3 \cdot 2^2 + 3^3 \cdot 2^3 + \cdots + n^3 \cdot 2^n) = 2^{n+1} \cdot n^3 - 3 \cdot (1^2 \cdot 2 + 2^2 \cdot 2^2 + 3^2 \cdot 2^3 + \cdots + n^2 \cdot 2^n) + 3 \cdot (1 \cdot 2 + 2 \cdot 2^2 + 3 \cdot 2^3 + \cdots + n \cdot 2^n) - (2 + 2^2 + 2^3 + \cdots + 2^n)$ și se folosesc formulele anterioare.

Complexitate $O(Q \cdot \max(l_2 - l_1 + c_2 - c_1) \cdot \log(L \cdot C))$

Optimizare (prof. Adrian Panaete, Cătălin Frâncu)

O optimizare elegantă, dar care nu este necesară pentru a obține 100p, calculează puterile lui 2 în timp constant, eliminând factorul logaritmic. Pentru fiecare dintre cele $L + C - 1$ diagonale precalculăm puterea lui 2 necesară pentru primul termen de pe acea diagonală (așadar pe prima coloană și pe ultima linie din matrice). Când avem nevoie de puterea 2^k pentru cel de-al k-lea termen, înmulțim puterea precalculată pe diagonala lui k cu 2^d, unde d este distanța dintre elementul k și primul element de pe acea diagonală. Această optimizare necesită doi vectori de câte 200 000 de elemente, deci 1,6 MB de memorie.

Această soluție poate fi împinsă mai departe pornind de la Mica Teoremă a lui Fermat, care ne spune că $2^{MOD-1} = 1 \pmod{MOD}$. Așadar, pentru a calcula 2^k, putem începe prin a calcula $k' = k \mod (MOD - 1)$. Astfel exponentul maxim va fi $1\,000\,000\,005 < 2^{30}$. Acum construim doi vectori de câte $2^{15} = 32\,768$ de elemente fiecare: $a[i] = 2^{32\,768 \times i}$ și $b[i] = 2^i$. Atunci $2^k = a[k'/32\,768] \times b[k' \mod 32\,768]$. Această optimizare necesită $2 \times 32\,768 \times 4 = 256$ KB de memorie.

11.3 Cod-sursă pentru problema Ashima

```
//prof. Mihai Bunget
#include <bits/stdc++.h>
#define ll long long
using namespace std;
ifstream f("ashima.in");
ofstream g("ashima.out");
const long long mod = 1000000007;
ll K,L,C,Q;
ll la,lb,ca,cb;
ll doi = 2;

void date()
{
    f >> K >> L >> C >> Q;
}

ll poz(ll l, ll c)
{
    ll r;
    if(L <= C){
        if(l+c <= L+1) r = (l+c-2)*(l+c-1)/2+c;
        else if(l+c <= C+1) r = L*(L+1)/2+L*(l+c-L-2)+L-l+1;
             else r = L*(L+1)/2+L*(C-L)+(L-1)*L/2-(L+C-l-c+1)*(L+C-l-c+2)/2+L-l+1;
    }
    else{
        if(l+c <= C+1) r = (l+c-2)*(l+c-1)/2+c;
        else if(l+c <= L+1) r = C*(C+1)/2+C*(l+c-C-2)+c;
             else r = C*(C+1)/2+C*(L-C)+C*(C-1)/2-(L+C-l-c+1)*(L+C-l-c+2)/2+L-l+1;
    }
    return r;
}

ll pwr(ll e)
{
    if(e==0) return 1;
    if(e%2==0)
```

```
    {
        ll z = pwr(e/2);
        return (z*z)%mod;
    }
    return (doi*pwr(e-1))%mod;
}

ll suma(ll n)
{
    ll r,x,y,m;
    if(K==0)
    {
        r = (pwr(n+1)-2+mod)%mod;
    }
    else
    if(K==1)
    {
        r = pwr(n+1);
        r = ((n-1)%mod*r+2)%mod;
    }
    else
    if(K==2)
    {
        r = pwr(n+1);
        m = n%mod;
        x = (m*m)%mod;
        x = (x-2*m+3+2*mod)%mod;
        r = (r*x%mod-6+mod)%mod;
    }
    else
    {
        r = pwr(n+1);
        m = n%mod;
        x = (m*m)%mod;
        y = (m*x)%mod;
        x = (y-3*x%mod+9*m%mod-13+4*mod)%mod;
        r = (r*x+26)%mod;
    }
    return r;
}

void solsoft()
{
    ll i,j,h,u,sol;

    for(i = 1; i <= Q; i++)
    {
        f >> la >> lb >> ca >> cb;
        sol = 0;
        for(j = la; j <= lb; j++)
        {
            h = poz(j,ca);
            if(j+ca-la<=cb) u = poz(la,j+ca-la);
            else u = poz(j+ca-cb,cb);
            sol = (sol+suma(u)-suma(h-1)+mod)%mod;
        }
        for(j = ca+1; j <= cb; j++)
        {
            h = poz(lb,j);
            if(j+lb-la<=cb) u = poz(la,j+lb-la);
            else u = poz(j+lb-cb,cb);
```

```
            sol = (sol+suma(u)-suma(h-1)+mod)%mod;
        }
        g << sol << "\n";
    }
}

int main()
{
    date();
    solsoft();
    return 0;
}
```

```
//Catalin Francu
#include <stdio.h>

#define MOD 1000000007
#define ROOT_MOD 32768
#define ROOT_BITS 15

typedef unsigned long long u64;
typedef __int128 i128;

int k, l, c;

// pow2: 2^0, 2^1, 2^2, ..., 2^32767
// pow2_large: 2^0, 2^32768, 2^65536, 2^98304, ... 2^(32768*32767)
int pow2[ROOT_MOD];
int pow2_large[ROOT_MOD];

void precompute_powers_of_2() {
  pow2[0] = 1;
  for (int i = 1; i < ROOT_MOD; i++) {
    pow2[i] = pow2[i - 1] * 2 % MOD;
  }

  pow2_large[0] = 1;
  pow2_large[1] = (2 * pow2[ROOT_MOD - 1]) % MOD;
  for (int i = 2; i < ROOT_MOD; i++) {
    pow2_large[i] = (u64)pow2_large[i - 1] * pow2_large[1] % MOD;
  }
}

int power_of_2(u64 e) {
  e %= (MOD - 1); // Mica teoremă a lui Fermat.

  long long small = pow2_large[e >> ROOT_BITS];
  long long large = pow2[e & (ROOT_MOD - 1)];
  int result = small * large % MOD;

  return result;
}

// Pentru a deduce aceste formule, fie S_kn = \sum_1^n i^k*2^i.
// Atunci S_3n se deduce astfel:
// S_3n = sum i^3*2^i =
// sum [(i-1) + 1]^3*2^i =
// sum (i-1)^3*2^i + 3 sum (i-1)^2*2^i + 3 sum (i-1)*2^i + sum 2^i =
// sum (i-1)^3*2^(i-1) + 3 sum (i-1)^2*2^(i-1) + 3 sum (i-1)*2^(i-1) + sum 2^i =
// Apoi facem schimbarea de variabilă i-1 -> j și regăsim sumele
// S_3n - n^3*2^n, S_2n - n^2*2^n, S_1n - n*2^n.
int sum_i_k_2_i(int k, i128 n) {
```

```
  i128 polynomial = 0;
  int free_term = 0;

  switch (k) {
    case 3:
      polynomial = ((n - 3) * n + 9) * n - 13;
      free_term = 26;
      break;
    case 2:
      polynomial = (n - 2) * n + 3;
      free_term = -6;
      break;
    case 1:
      polynomial = n - 1;
      free_term = 2;
      break;
    case 0:
      polynomial = 1;
      free_term = -2;
      break;
  }

  polynomial %= MOD;
  int result = (polynomial * power_of_2(n + 1) + free_term) % MOD;

  return result;
}

int min(int x, int y) {
  return (x < y) ? x : y;
}

u64 get_square_number(int row, int col) {
  int m = min(l, c);
  int diag = row + col - 1;
  u64 result;

  if (diag <= m) {
    result = (u64)diag * (diag + 1) / 2 - (row - 1);
  } else if (diag >= l + c - m) {
    // simetrie
    result = (u64)l * c + 1 - get_square_number(l + 1 - row, c + 1 - col);
  } else if (l < c) {
    result = (u64)(diag - m) * m + (u64)m * (m + 1) / 2
      - (row - 1);
  } else {
    result = (u64)(diag - m) * m + (u64)m * (m + 1) / 2
      - (c - col);
  }
  return result;
}

u64 get_partial_sum_at(int row, int col) {
  u64 square_number = get_square_number(row, col);
  return sum_i_k_2_i(k, square_number);
}

int rectangle_query(int row1, int col1, int row2, int col2) {

  // Luăm laturile sus și dreapta cu +, cele de jos și stînga cu -.
  i128 positive = 0;
```

```
  for (int col = col1; col < col2; col++) {
    positive += get_partial_sum_at(row1, col);
  }
  for (int row = row1; row <= row2; row++) {
    positive += get_partial_sum_at(row, col2);
  }

  i128 negative = 0;
  for (int row = row1 + 1; row <= row2 + 1; row++) {
    negative += get_partial_sum_at(row, col1 - 1);
  }
  for (int col = col1; col < col2; col++) {
    negative += get_partial_sum_at(row2 + 1, col);
  }

  return ((positive % MOD) + MOD - (negative % MOD)) % MOD;
}

int main() {
  int num_queries;
  FILE* fin = fopen("ashima.in", "r");
  FILE* fout = fopen("ashima.out", "w");

  fscanf(fin, "%d %d %d %d", &k, &l, &c, &num_queries);
  precompute_powers_of_2();

  while (num_queries--) {
    int row1, col1, row2, col2;
    fscanf(fin, "%d %d %d %d", &row1, &row2, &col1, &col2);
    int answer = rectangle_query(row1, col1, row2, col2);
    fprintf(fout, "%d\n", answer);
  }

  fclose(fin);  fclose(fout);
  return 0;
}
```

11.4 Problema Bossfight

Propusă de: stud. Theodor Gabriel Tulba-Lecu, Universitatea Politehnică din București

Now we can devour gods, TOGETHER!

Gimi a găsit un nou joc, faimos pentru dificultatea sa ridicată. Jocul este alcătuit din N camere, numerotate de la 1 la N. Fiecare cameră i conține un monstru a cărui putere este un număr natural P_i. Gimi trece, în ordine, prin toate camerele. În fiecare cameră el poate alege să se lupte cu monstrul sau nu.

Gimi pornește cu o sabie de durabilitate S. El învinge un monstru doar dacă puterea acestuia este mai mică sau egală cu durabilitatea sabiei. După luptă, durabilitatea sabiei scade cu puterea monstrului. De exemplu, dacă Gimi are o sabie de durabilitate 10 și se luptă cu un monstru de putere 4, atunci durabilitatea sabiei sale va scădea la 6.

Ținând la reputația sa, Gimi dorește să se lupte cu exact **3 monștri** din ce în ce mai puternici. Cu alte cuvinte, dacă Gimi a învins un monstru de putere x, el se va lupta în continuare numai cu monștri de putere strict mai mare decât x.

Cerințe

Gimi se întreabă în câte moduri poate să aleagă **3 monștri** cu care să se lupte. Două mulțimi de 3 monștri se consideră diferite dacă există cel puțin un monstru în prima mulțime care nu aparține celei de-a doua mulțimi.

Formal, se cere numărul de tripleți $i < j < k$ pentru care $P_i < P_j < P_k$ și $P_i + P_j + P_k \leq S$.

Date de intrare

Fișierul de intrare `bossfight.in` conține pe prima linie două numere naturale N și S, reprezentând numărul de camere ale jocului și durabilitatea inițială a sabiei lui Gimi, iar pe a doua linie N numere naturale P_i separate prin câte un spațiu, reprezentând puterile celor N monștri.

Date de ieșire

În fișierul de ieșire `bossfight.out` se va afișa un singur număr ce reprezintă numărul total de moduri în care Gimi poate alege monștrii.

Restricții

- $1 \leq N \leq 10\,000$
- $1 \leq P_i, S \leq 1\,000\,000\,000$, pentru oricare $1 \leq i \leq N$

#	Puncte	Restricții
1	11	$1 \leq N \leq 100$
2	27	$1 \leq N \leq 2\,000$
3	62	Nu există alte restricții.

Exemple

bossfight.in	bossfight.out	Explicații
5 9 1 2 3 4 3	5	Tripletele de poziții sunt: $(1, 2, 3), (1, 2, 4), (1, 2, 5), (1, 3, 4), (2, 3, 4)$. Un exemplu de triplet incorect este $(1, 4, 5)$, deoarece puterea monstrului din camera 4 este mai mare decât puterea monstrului din camera 5.
8 16 4 2 1 6 5 7 9 8	13	Tripletele de poziții sunt: $(1, 5, 6), (2, 4, 6), (2, 4, 8), (2, 5, 6), (2, 5, 7), (2, 5, 8), (3, 4, 6), (3, 4, 7), (3, 4, 8), (3, 5, 6), (3, 5, 7), (3, 5, 8), (3, 6, 8)$

11.5 Rezolvarea problemei Bossfight

Notăm cu x o poziție din șir și cu P_x valoarea de la poziția x.

Soluție în $\mathcal{O}(N^3)$:

O prima soluție constă în verificarea tuturor tripletelor de numere posibile și numărarea celor valide. Această soluție ar trebui sa obțină în jur de 11 puncte.

Soluție în $\mathcal{O}(N^2 \log N)$:

Plecăm de la soluția precedentă, în care selectăm toate tripletele de numere (x, y, z) cu proprietatea că $x < y < z$, $P_x < P_y < P_z$ și $P_x + P_y + P_z \leq S$. Observăm că odată cu selectarea pozițiilor x și y, putem alege orice valoare P_z din dreapta lui y cuprinsă între $[P_y + 1, S - P_x - P_y]$ (sau similar pentru y și z fixate, iar x determinat). O soluție bazată pe această idee poate obține de la 38 de puncte în sus, în funcție de implementare.

Arbore Fenwick (AIB)

Iterăm prin fiecare boss central, de la stânga la dreapta. Pe măsură ce facem asta, menținem o *colecție* cu boșii din stânga centrului. Pentru un centru fixat y, iterăm prin fiecare boss din dreapta, z. Pentru fiecare pereche (y, z) cu $P_y < P_z$, un boss corect din stânga trebuie să aibă puterea cel mult egală cu $(P_y - 1)$ și cel mult egală cu $(S - P_y - P_z)$. După ce terminăm de procesat centrul y, adăugăm P_y la colecție și trecem la centrul $y + 1$.

Cum implementăm colecția? Avem nevoie de o structură care să admită în $\mathcal{O}(\log N)$ operațiile (1) adaugă o putere la colecție și (2) raportează numărul de puteri mai mici sau egale cu un P dat. Un AIB atinge acest scop, cu observația că puterile pot fi 10^9, deci trebuie normalizate. O tehnică simplă de normalizare le ține sortate într-un vector. Prin diferența $(S - P_y - P_z)$ pot lua naștere și valori care nu sunt printre cele N puteri date la intrare. Pe acestea le căutăm binar în vectorul sortat de puteri.

O optimizare care scade timpul de rulare cu circa 40% pornește de la următoarea observație. Dacă $P_y - 1 < S - P_y - P_z$, atunci știm că valoarea care ne limitează este $P_y - 1$ și nu mai este nevoie să căutăm binar poziția lui $(S - P_y - P_z)$ printre puteri.

Căutare binară

Putem mai întâi să selectăm valoarea de la mijloc (y), iar apoi sortăm crescător secvența de valori din dreapta lui y. Pentru orice x din stânga, putem căuta binar cea mai mică (P_{z1}), respectiv cea mai mare (P_{z2}) valoare pe care o poate lua P_z. Numărul de soluții pentru z este egal cu numărul de valori din șir cuprinse între z_1 și z_2.

Diverse optimizări se pot face la această soluție. Odată cu mutarea poziției y, valorile din dreapta nu trebuiesc sortate din nou. În schimb, se poate adăuga / șterge direct valoarea care este afectată de modificarea poziției y în complexitate $\mathcal{O}(N)$. Această soluție ar trebui sa obțină până la 78 de puncte.

Soluție în $\mathcal{O}(N^2)$:

Plecăm de la soluția precedentă, în care mai întâi setăm valoarea y, apoi x, iar la final căutam binar valorile posibile ale lui P_z. Motivul pentru care este nevoie să căutăm binar este acela că valorile lui P_x alternează. Dar dacă am sorta crescător valorile posibile ale lui P_x, respectiv valorile posibile ale lui z, atunci am putea identifica tripletele folosind o tehnică de tipul „two-pointers”. Pe măsură ce creștem valoarea lui P_x, vom scădea valoarea lui P_z.

Pornim inițial cu vectorul de valori posibile ale lui x care va conține strict prima valoare din șir, în timp ce vectorul de valori posibile ale lui z va conține ultimele $n-2$ valori din șir, sortate crescător. Plecăm cu y de pe poziția a doua, și calculăm toate tripletele posibile cu tehnica prezentată mai sus. Apoi, când trecem la y pe poziția a treia, ștergem această valoare din vectorul de valori posibile ale lui z, și inserăm valoarea de pe poziția a doua în vectorul de valori posibile ale lui x. Putem face ștergerea și inserarea naiv, în $\mathcal{O}(n)$. Nu este nevoie să căutăm soluții mai eficiente, întrucât tehnica „two-pointers” necesită oricum $\mathcal{O}(n)$. Repetăm acest proces pentru fiecare valoare posibilă a lui y.

Această soluție, cât timp este implementată eficient, ar trebui să obțină 100 de puncte.

11.6 Cod-sursă pentru problema Bossfight

```cpp
// Autor: Gabi Tulba-Lecu
#include <bits/stdc++.h>
using namespace std;
ifstream fin("bossfight.in");
ofstream fout("bossfight.out");
long long ret = 0;
int n, t, v[10000] = {};
int main() {
    fin >> n >> t;
    for (int i = 0; i < n; ++i) fin >> v[i];

    vector<int> rhs, lhs(v, v + n);
    sort(begin(lhs), end(lhs));

    for (int i = n - 1; i >= 0; --i) {
        lhs.erase(lower_bound(begin(lhs), end(lhs), v[i]));
        for (int x = 0, y = lhs.size(); x < rhs.size(); ++x) {
            while (y > 0 && (lhs[y - 1] >= v[i] || lhs[y - 1] + v[i] > t - rhs[x])) --y;
            if (v[i] < rhs[x]) ret += y;
        }
        rhs.insert(lower_bound(begin(rhs), end(rhs), v[i]), v[i]);
    }
    fout << ret << endl;
```

```
    return 0;
}
```

```
// Autor: Ionel-Vasile Piț-Rada
// O(N*log(N)+N*N)
#include <fstream>
#include <algorithm>
using namespace std;
ifstream  fin("bossfight.in");
ofstream fout("bossfight.out");
int N,S,r,n1,n2,v1[10002],v2[10002];
long long s;
struct pereche{
    int val,p;
}v[10002];
int cmp(pereche a, pereche b){
    if(a.val<b.val)return 1;
    if(a.val==b.val && a.p<b.p)return 1;
    return 0;
}
int main(){
    fin>>N>>S;
    for(int i=1;i<=N;i++){
        fin>>v[i].val;
        v[i].p=i;
    }
    sort(v+1,v+1+N,cmp);
    s=0;
    for(int i=2;i<=N-1;i++){
        n1=0;
        for(int j=1;j<=i-1;j++){
            if(v[j].val<v[i].val && v[j].p<v[i].p){
                n1++;
                v1[n1]=v[j].val;
            }
        }
        n2=0;
        for(int j=i+1;j<=N;j++){
            if(v[i].val<v[j].val && v[i].p<v[j].p){
                n2++;
                v2[n2]=v[j].val;
            }
        }
        int si=S-v[i].val;
        for(int j=1;j<=n1;j++){
            while(n2>=1 && v1[j]+v2[n2]>si){
                n2--;
            }
            if(n2==0)break;
            s=s+n2;
        }
    }
    fout<<s;
    return 0;
}
```

11.7 Problema Veverițe

Propusă de: prof. Adrian Panaete, Colegiul Național „August Treboniu Laurian" Botoșani

Două veverițe gemene au descoperit un depozit de alune care are o formă foarte ciudată. Mai precis, depozitul are forma unei matrice $N \times N$ cu N impar. Fiecare poziție din matrice este o cameră și în fiecare cameră se află câte o alună. Camerele sunt numerotate cu numărul de linie și numărul de coloană. Prima veveriță, pe numele ei Chip, se găsește inițial în camera $(1, 1)$, iar a doua, pe numele ei Dale, în camera (N, N). Veverițele vor să culeagă toate alunele și să se întâlnească în camera din centrul depozitului. Pentru aceasta, ele vor parcurge camerele trecând din una în alta fără să treacă vreodată printr-o cameră prin care a mai trecut una dintre ele. Ele se pot deplasa dintr-o cameră în alta (evident, fără să iasă din depozit și fără să intre într-o camera deja vizitată). Când trece dintr-o cameră într-alta, Chip își notează traseul astfel:

- Dacă merge spre camera din dreapta, din camera (L, C) în camera $(L, C + 1)$, notează trecerea cu 0.
- Dacă merge spre camera din stânga, din camera (L, C) în camera $(L, C-1)$, notează trecerea cu 1.
- Dacă merge spre camera de sus, din camera (L, C) în camera $(L - 1, C)$, notează trecerea cu 2.
- Dacă merge spre camera de jos, din camera (L, C) în camera $(L + 1, C)$, notează trecerea cu 3.

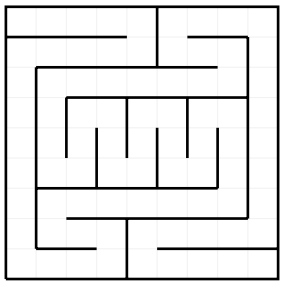
Interesant este că la orice trecere a lui Chip dintr-o cameră în alta, Dale se va muta exact în sens opus, adică:

- Dacă Chip merge spre dreapta, Dale merge spre stânga.
- Dacă Chip merge spre stânga, Dale merge spre dreapta.
- Dacă Chip merge în sus, Dale merge în jos.
- Dacă Chip merge în jos, Dale merge în sus.

Meticulos, Chip își calculează și își notează în ordine lexicografică toate traseele posibile prin care pot fi culese toate alunele din depozit. De exemplu, pentru $N = 9$, traseul de pe poziția $P = 12345$ este notat astfel:

```
000031111333333300021120000022213312213
```

și corespunde schemei de mai jos:



Cerințe

Cunoscând N, să se răspundă la Q întrebări de forma: „Ce traseu a notat Chip pe poziția P?"

Date de intrare

Fișierul de intrare `veverite.in` conține pe prima linie numerele N și Q cu semnificația de mai sus. Pe următoarele Q linii se găsesc întrebările, câte o întrebare pe o linie. Astfel, pe linia $i + 1$

$(1 \leq i \leq Q)$ se găsește un singur număr P_i.

Date de ieșire

Fișierul de ieșire `veverite.out` conține Q linii. Fiecare linie conține un șir de caractere 0, 1, 2 sau 3, neseparate prin spațiu. Șirul de pe linia i $(1 \leq i \leq Q)$ va conține codificarea traseului lui Chip corespunzător poziției P_i.

Restricții

- $3 \leq N \leq 9$
- N este impar.
- $1 \leq Q \leq 1\,000$
- $1 \leq P_i \leq maxP$, unde $maxP$ este numărul maxim de trasee posibile pentru N.

#	Puncte	Restricții
1	4	$N = 3$
2	7	$N = 5$
3	8	$N = 7$, $P_i \leq 10$ $(1 \leq i \leq Q)$
4	13	$N = 7$, $P_i \leq 564$ $(1 \leq i \leq Q)$
5	19	$N = 9$, $P_i \leq 10$ $(1 \leq i \leq Q)$
6	21	$N = 9$, $P_i \leq 1\,000$ $(1 \leq i \leq Q)$
7	28	$N = 9$, $P_i \leq 93\,866$ $(1 \leq i \leq Q)$

Exemple

veverite.in	veverite.out
5 3 1 2 3	000031111300 000031303112 000033311120
9 1 12345	000031111333333300021120000022213312213

11.8 Rezolvarea problemei Veverițe

- Pentru $N = 3$ avem 2 trasee.
- Pentru $N = 5$ avem 16 trasee.
- Pentru $N = 7$ avem 564 trasee.
- Pentru $N = 9$ avem 93866 trasee.

Soluția pentru toate subtask-urile este backtracking. Se generează prin backtracking toate traseele lui Chip în ordine lexicografică având grijă ca pentru orice mutare face Chip să consemnăm și mutarea simetrică față de centrul matricei pe care o face Dale. Punctajele obținute depind de modul în care se implementează backtracking-ul după cum urmează.

Soluție backtracking neoptimizat

Se consideră o matrice în care vom marca pozițiile prin care a trecut una dintre veverițe. La fiecare moment verificăm ce poziții vecine celei în care se găsește Chip nu au fost vizitate și îl mutăm pe Chip acolo. Avem grijă sa marcăm drept vizitate poziția lui Chip și poziția simetrică în care se găsește Dale. Deoarece numărul de soluții este foarte mic pentru $N \leq 7$ cu această abordare se vor obține punctele corespunzătoare subtaskurilor 1, 2, 3 și 4.

Soluție backtracking neoptimizat, oprit la poziția maximă din query-uri

Este exact aceeași abordare de mai sus cu observația suplimentară că la fiecare soluție validă obținută incrementăm un contor și la fiecare apel al funcției recursive de backtracking verificăm dacă acel contor a atins poziția maximă din toate query-urile (situație în care returnăm).

În acest fel nu vor fi generate toate soluțiile, ci doar primele în ordine lexicografică de care avem nevoie pentru a răspunde la întrebările din test.

Această abordare rezolvă subtask-urile 5 și 6.

Backtracking optimizat

Pentru a rezolva subtask-ul maxim vom implementa în alt mod backtracking-ul. În loc să folosim o „matrice de vizitare", vom construi inițial o matrice în care vom consemna numărul de vecini nevizitați. Astfel pozițiile din colț vor avea 2 vecini, celelalte poziții de pe liniile și coloanele exterioare 3 vecini, iar restul pozițiilor vor avea 4 vecini. În momentul în care una dintre veverițe va pleca dintr-o poziție, numărul vecinilor acelei poziții va deveni 0 iar numărul vecinilor pentru fiecare poziție vecină nevizitată va descreste cu 1.

În acest mod vom putea controla dacă o poziție a fost vizitată (are 0 vecini) sau nu.

O primă optimizare ar fi aceea că, dacă o poziție are la intrarea în ea un vecin marcat cu 2, din poziția curentă va trebui să trecem obligatoriu în acel vecin (altfel nu s-ar mai putea ajunge mai târziu la acel vecin). Implementată îngrijit, această optimizare este suficientă pentru a obține 100p.

A doua optimizare este mai subtilă și se referă la situația când traseul nu va fi valid pentru că separă matricea în două zone "neconexe" și astfel, dacă la un moment dat intrăm în una dintre ele, cealaltă nu va mai putea fi parcursă.

Pentru a surprinde astfel de situații, trebuie să fim foarte atenți atunci când traseul lui Chip se găsește la un moment dat pe o linie sau coloană exterioară. Pentru a nu „deconecta" pozițiile nevizitate în zone inaccesibile una alteia, se poate observa că pe pozițiile de pe prima linie și în continuare de pe ultima coloană trebuie să apară (dacă apar) în traseu strict în ordinea de la Chip spre Dale. Similar pentru pozițiile primei coloane urmate apoi de ultima linie. Altfel se va crea o zonă nevizitată în care Chip nu ar trebui să intre, iar Dale nu mai are cum să intre după cum urmează:

- Pe de o parte, dacă Chip ar ajunge acolo, atunci el însuși ar avea traseul spre centrul matricei blocat de propriul să traseu.
- Pe de altă parte, Dale are deja blocat traseul spre acea poziție de traseul curent al lui Chip.

11.9 Cod-sursă pentru problema Verevițe

```
// Autor: prof. Adrian Panaete
#include <bits/stdc++.h>
using namespace std;
ifstream f("veverite.in");
ofstream g("veverite.out");
string trSol, trConj;
int n, nrVec[13][13], nrSol=0, nrConj, q;
int totSol[10]={0,0,0,3,0,17,0,565,0,93867}, haveQ[100000], Q[1001];
        ///   R  L  U  D
int di[4]=  { 0, 0,-1, 1};
int dj[4]=  { 1,-1, 0, 0};
int ONLY[4]={ 1, 2, 4,8};
int NO[4]  ={14,13,11,7};
int M,K;
void bkt(int,int,int);
string Solutii[100000];
int main()
{
    f>>n>>q;
    for(int i=1;i<=q;i++)
    {
        f>>Q[i];
        haveQ[Q[i]]=1;
    }
    /// generez matricea numarului de vecini
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            nrVec[i][j]=4;
    for(int i=1;i<=n;i++)
    {
        nrVec[1][i]--;
        nrVec[n][i]--;
        nrVec[i][1]--;
        nrVec[i][n]--;
    }
    M=(n+1)/2;nrVec[M][M]=6;
    K=(n*n-1)/2;
    nrConj=totSol[n];
    trSol=string(K,'0');
    trConj=string(K,'0');
    bkt(1,1,0);
    for(int i=1;i<=q;i++)
        g<<Solutii[Q[i]]<<'\n';
    return 0;
}
void bkt(int i,int j,int k)
{
    if(nrSol==nrConj-1)return;
    if(i==M&&j==M)
    {
        if(k==K)
        {
            nrSol++;
            nrConj--;
            if(haveQ[nrSol])Solutii[nrSol]=trSol;
            if(haveQ[nrConj])Solutii[nrConj]=trConj;
        }
        return;
    }
```

```
    if(i==1&&nrVec[i][j-1])return;
    if(j==1&&nrVec[i-1][j])return;
    if(i==n&&nrVec[i][j-1])return;
    if(j==n&&nrVec[i-1][j])return;
    int PTH=15;
    int memV=0;
    for(int d=0,b=1;d<4;d++,b<<=1)
    {
        int I=i+di[d];
        int J=j+dj[d];
        if(nrVec[I][J]==0)
            PTH&=NO[d];
        else
        {
            memV|=ONLY[d];
            nrVec[i][j]--;nrVec[n+1-i][n+1-j]--;
            nrVec[I][J]--;nrVec[n+1-I][n+1-J]--;
            if(nrVec[I][J]==1)
                PTH&=ONLY[d];
        }
    }
    while(PTH)
    {
        int d=__builtin_ctz(PTH);
        PTH-=PTH&(-PTH);
        trSol[k]+=d;trConj[k]+=3-d;
        bkt(i+di[d],j+dj[d],k+1);
        trSol[k]-=d;trConj[k]-=3-d;
    }
    for(;memV;memV-=memV&(-memV))
    {
        int d=__builtin_ctz(memV);
        int I=i+di[d],J=j+dj[d];
        nrVec[i][j]++;nrVec[n+1-i][n+1-j]++;
        nrVec[I][J]++;nrVec[n+1-I][n+1-J]++;
    }
}
```

```
// Autor: Cătălin Frâncu
// Backtracking cu euristica fundăturii: dacă există un vecin care are exact
// un vecin liber, vizitează-l doar pe acela.
#include <stdio.h>
#include <stdlib.h>

#define MAX_N 9
#define MAX_SOLUTIONS 94'000

#define NUM_DIRECTIONS 4
#define RIGHT 0
#define LEFT 1
#define UP 2
#define DOWN 3
#define NONE -1

#define BIT_MASK 3
#define BIT_WIDTH 2

int DIR[NUM_DIRECTIONS][2] = { { 0, 1 }, { 0, -1 }, { -1, 0 }, { 1, 0 } };

bool visited[MAX_N + 2][MAX_N + 2];
char num_freedoms[MAX_N + 1][MAX_N + 1];
__int128 sol[MAX_SOLUTIONS + 1], current_sol;
```

```
int size, target_depth, center, num_solutions;

inline void process_solution(int row, int col) {
  sol[++num_solutions] = current_sol;
}

inline void set_cell(int row, int col, bool val) {
  visited[row][col] = val;
  int delta = 1 - 2 * val; // true --> -1, false --> +1
  num_freedoms[row][col - 1] += delta;
  num_freedoms[row][col + 1] += delta;
  num_freedoms[row - 1][col] += delta;
  num_freedoms[row + 1][col] += delta;
}

inline void set_both_cells(int row, int col, bool val) {
  set_cell(row, col, val);
  set_cell(size - row + 1, size - col + 1, val);
}

void compute_freedoms() {
  for (int row = 1; row <= size; row++) {
    for (int col = 1; col <= size; col++) {
      num_freedoms[row][col] =
        !visited[row - 1][col] + !visited[row + 1][col] +
        !visited[row][col - 1] + !visited[row][col + 1];
    }
  }
}

void prepare_matrices() {
  target_depth = (size * size - 1) / 2;
  center = (size + 1) / 2;

  for (int i = 1; i <= size; i++) {
    visited[i][0] = visited[i][size + 1] = true;
    visited[0][i] = visited[size + 1][i] = true;
  }

  set_both_cells(1, 1, true);

  compute_freedoms();
}

inline int get_forced_dir(int row, int col) {
  for (int dir = 0; dir < NUM_DIRECTIONS; dir++) {
    int r = row + DIR[dir][0];
    int c = col + DIR[dir][1];
    if (!visited[r][c] && (num_freedoms[r][c] == 1)) {
      return dir;
    }
  }
  return NONE;
}

inline void push_bits(int val) {
  current_sol = (current_sol << BIT_WIDTH) + val;
}

inline void pop_bits() {
  current_sol >>= BIT_WIDTH;
```

```
}

void backtrack(int depth, int row, int col) {
  if (depth == target_depth) {
    process_solution(row, col);
    return;
  }

  if ((row == center) && (col == center)) {
    return; // prea devreme în centru
  }

  int start = 0, end = NUM_DIRECTIONS - 1;
  int forced_dir = get_forced_dir(row, col);
  if (forced_dir != NONE) {
    start = end = forced_dir;
  }

  for (int dir = start; dir <= end; dir++) {
    int r = row + DIR[dir][0];
    int c = col + DIR[dir][1];
    if (!visited[r][c]) {
      push_bits(dir);
      set_both_cells(r, c, true);
      backtrack(depth + 1, r, c);
      set_both_cells(r, c, false);
      pop_bits();
    }
  }
}

void print_solution(int pos, FILE* f) {
  for (int d = target_depth - 1; d >= 0; d--) {
    int val = (sol[pos] >> (BIT_WIDTH * d)) & BIT_MASK;
    fputc('0' + val, f);
  }
  fputc('\n', f);
}

int main() {
  int num_queries;
  FILE* fin = fopen("veverite.in", "r");
  FILE* fout = fopen("veverite.out", "w");

  fscanf(fin, "%d %d", &size, &num_queries);

  prepare_matrices();
  backtrack(0, 1, 1);

  while (num_queries--) {
    int pos;
    fscanf(fin, "%d", &pos);
    print_solution(pos, fout);
  }
  fclose(fin);  fclose(fout);
  return 0;
}
```

Capitolul 12

Barajul 3

12.1 Problema Countall

Propusă de: stud. Vlad-Mihai Bogdan, Facultatea de Matematică și Informatică, Universitatea București

Kida vă oferă două numere N și M. Ea vă mai oferă și un șir, A, de N numere naturale cuprinse între 0 și M inclusiv. Șirul A conține două tipuri de valori: valori cuprinse între 1 și M, care nu pot fi schimbate, respectiv valori de 0, care pot fi înlocuite cu orice număr cuprins între 1 și M.

Pentru un șir V, cu valori între 1 și M, vom nota cu $count(V)$ numărul de perechi (V_i, V_j) de pe pozițiile i și j astfel încât $i < j$ și cmmdc$(V_i, V_j) = 1$.

Cerință

Se cere suma $count(V)$ pentru toate șirurile distincte V care se pot obține din șirul A, înlocuind toate valorile de 0 cu numere cuprinse între 1 și M. Deoarece acest număr poate să fie foarte mare, se cere restul împărțirii sale la $10^9 + 9$.

Date de intrare

Pe prima linie din fișierul de intrare `countall.in` se află două numere N și M, iar pe a doua linie valorile din șirul A, separate prin câte un spațiu.

Date de ieșire

Pe singura linie din fișierul de ieșire `countall.out` se află numărul S, reprezentând restul împărțirii la $10^9 + 9$ a sumei valorilor *count* pentru toate șirurile care se pot obține din șirul A.

Restricții

- $1 \leq N, M \leq 100\,000$
- $0 \leq A_i \leq M$, pentru orice i de la 1 la N

#	Puncte	Restricții
1	13	$1 \leq N, M \leq 7$
2	8	$M = 2$
3	21	Șirul A nu conține nicio valoare de 0.
4	23	Șirul A conține doar valori de 0.
5	17	$1 \leq N, M \leq 1\,000$
6	18	Fără restricții suplimentare.

Exemple

countall.in	countall.out	Explicații
3 4 2 0 3	9	Șirurile care se pot obține sunt: $[2, 1, 3]$, pentru care valoarea *count* este 3; $[2, 2, 3]$, pentru care valoarea *count* este 2; $[2, 3, 3]$, pentru care valoarea *count* este 2; $[2, 4, 3]$, pentru care valoarea *count* este 2. Suma valorilor *count* este $3 + 2 + 2 + 2 = 9$.
3 2 0 0 2	7	Șirurile care se pot obține sunt: $[1, 1, 2]$, pentru care valoarea *count* este 3; $[1, 2, 2]$, pentru care valoarea *count* este 2; $[2, 1, 2]$, pentru care valoarea *count* este 2; $[2, 2, 2]$, pentru care valoarea *count* este 0. Suma valorilor *count* este $3+2+2+0 = 7$.

12.2 Rezolvarea problemei Countall

Pentru a rezolva problema, vom împărți perechile de numere prime între ele în trei categorii:

- perechi formate din două numere care deja sunt fixate în șir;
- perechi formate din două numere care înlocuiesc două valori de 0;
- perechi formate cu un număr fixat și un număr care înlocuiește un 0.

O primă observație importantă este că fiecare pereche de poziții apare de același număr de ori în toate șirurile care se pot forma. Pentru simplitate, vom nota cu Z numărul de valori de 0 din șir și cu R numărul de valori fixate din șir.

Soluție în $\mathcal{O}(N^2 \cdot \log M + M^2 \cdot \log M)$

Vom calcula numărul de perechi de valori prime între ele, unde ambele numere sunt fixate în șir. Acest număr se poate calcula ușor dacă fixăm fiecare două numere din șir și facem cel mai mare divizor comun al lor. Vom nota acest număr cu $fixedPairs$. Aceste perechi vor apărea în fiecare șir care se poate obține, deci, trebuie să înmulțim $fixedPairs$ cu M^Z.

Notăm cu $freePairs$ numărul de perechi de numere prime între ele, unde ambele numere înlocuiesc două valori de zero din șir. Acest număr se poate calcula ușor în $O(M^2 \cdot \log M)$. Aceste perechi contribuie la toate șirurile care au $Z-2$ valori de zero și restul de valori fixate. Trebuie să ținem cont și de pozițiile pe care așezăm numerele din pereche. Prin urmare, $freePairs$ trebuie înmulțit cu $Z \cdot (Z-1)/2 \cdot M^{Z-2}$.

Notăm cu $combinedPairs$ numărul de perechi formate dintr-un număr fixat și o valoare de zero din șir. Acestea apar în toate șirurile cu $Z - 1$ valori de zero, deci, acest număr trebuie înmulțit cu $M^{Z-1} \cdot Z$.

Soluție în $\mathcal{O}(M \cdot 2^F \cdot F + M \cdot \log M)$

Pentru soluția completă, trebuie să calculăm valorile $fixedPairs$, $freePairs$ și $combinedPairs$ mai rapid.

Pentru început, vom calcula phi_i ca fiind numărul de valori mai mici sau egale decât i care sunt prime cu i. Acest șir se poate calcula folosind un ciur în $O(M \cdot \log M)$. Acum, $freePairs$ este egal cu $1 + \sum_{i=2}^{M} 2 \cdot phi_i$.

Pentru a calcula $fixedPairs$ și $combinedPairs$ vom folosi principiul includerii și al excluderii. Ne vom concentra pe calcularea $fixedPairs$, de vreme ce $combinedPairs$ se calculează într-un mod asemănător.

Pentru un număr fixat x, vom reține o listă cu numerele prime distincte care apar în descompunerea sa în factori primi. Observăm că dimensiunea aceste liste este foarte mică; mai exact, ea poate să aibă cel mult 7 elemente. Vom încerca să calculăm pentru fiecare număr din șir, numărul de numere din stânga sa cu care **nu** este prim (și vom nota acest număr cu $badPairs$).

Așadar, pentru fiecare submulțime nevidă a listei de factori primi distincți, va trebui să adăugăm, respectiv să scădem din $badPairs$ (în funcție de paritatea cardinalului submulțimii) fr_{subset}, unde fr_{subset} este numărul de numere din șir din stânga numărului pe care îl procesăm care conțin numerele din $subset$ în descompunerea lor în factori primi. Conform principiului includerii și al excluderii, trebuie să scădem fr_{subset} dacă subset are dimensiune pară, respectiv să adăugăm fr_{subset} dacă subset are dimensiune impară. La final, trebuie să adăugăm $i - badPairs$ la $fixedPairs$.

Detaliile în ceea ce privește calcularea răspunsului (după calcularea $fixedPairs$, $combinedPairs$ și $freePairs$) sunt aceleași cu cele de la subtask-ul anterior.

Complexitatea finală este $\mathcal{O}(M \cdot 2^F \cdot F + M \cdot \log M)$.

12.3 Cod-sursă pentru problema Countall

```cpp
// Autor: Vlad Bogdan
#include <iostream>
#include <vector>
#include <assert.h>
#include <algorithm>
#include <numeric>

const std::vector<int> primes = {
  2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71,
  73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151,
  157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233,
  239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317
};

const int MOD = 1e9 + 9;

int lgpow(int a, int b) {
    int answer = (b >= 0);
    while (b > 0) {
        if (b % 2 == 1) {
```

```
            answer = (int64_t)answer * a % MOD;
        }
        b /= 2;
        a = (int64_t)a * a % MOD;
    }
    return answer;
}

int solveBothUnknown(int N, int M, const std::vector<int> &phi) {
    int answer = 1;
    for (int i = 2; i <= M; i++) {
        answer += 2 * phi[i];
        if (answer >= MOD) {
            answer -= MOD;
        }
    }
    answer = (int64_t)answer * N % MOD
      * (N - 1) % MOD
      * lgpow(2, MOD - 2) % MOD
      * lgpow(M, N - 2) % MOD;
    return answer;
}

int solveRest(int R, int M, const std::vector<int> &V, const std::vector<int> &phi) {
    if (V.empty()) {
        return 0;
    }

    int answerFixed = 0, answerPaired = 0;
    std::vector<int> fr(*max_element(V.begin(), V.end()) + 1);
    for (int i = 0; i < (int)V.size(); i++) {
        std::vector<int> primeFactors;
        int tmp = V[i];
        for (const int &prime : primes) {
            if (tmp % prime == 0) {
                primeFactors.push_back(prime);
                while (tmp % prime == 0) {
                    tmp /= prime;
                }
            }
            if (tmp == 1) {
                break;
            }
        }
        if (tmp > 1) {
            primeFactors.push_back(tmp);
        }

        int count = 0, countCoprime = 0;
        for (int mask = 1; mask < (1 << (int)primeFactors.size()); mask++) {
            int value = 1, card = 0;
            for (int i = 0; i < (int)primeFactors.size(); i++) {
                if ((mask & (1 << i)) > 0) {
                    value = value * primeFactors[i];
                    card++;
                }
            }

            count += fr[value] % MOD * (card % 2 == 1 ? 1 : -1);
            if (count >= MOD) {
                count -= MOD;
```

```
            }
            countCoprime += (M / value) * (card % 2 == 1 ? 1 : -1);
            if (countCoprime >= MOD) {
                countCoprime -= MOD;
            }
            fr[value]++;
        }

        answerPaired += M - countCoprime;
        if (answerPaired >= MOD) {
            answerPaired -= MOD;
        }
        answerFixed += i - count;
        if (answerFixed >= MOD) {
            answerFixed -= MOD;
        }
    }

    answerFixed = (int64_t)answerFixed * lgpow(M, R) % MOD;
    answerPaired = (int64_t)answerPaired * lgpow(M, R - 1) % MOD * R % MOD;
    return (answerFixed + answerPaired) % MOD;
}

int main() {
    assert(freopen("countall.in", "r", stdin));
    assert(freopen("countall.out", "w", stdout));

    int N, M; std::cin >> N >> M;
    std::vector<int> V(N), fixedValues;
    for (int &x : V) {
        std::cin >> x;
        if (x > 0) {
            fixedValues.push_back(x);
        }
    }

    std::vector<int> phi(M + 1);
    std::iota(phi.begin(), phi.end(), -1);
    for (int i = 2; i <= M; i++) {
        for (int j = 2 * i; j <= M; j += i) {
            phi[j] -= phi[i];
        }
    }

    int R = N - (int)fixedValues.size();
    std::cout << (solveBothUnknown(R, M, phi) + solveRest(R, M, fixedValues, phi)) % MOD << "\n";
    return 0;
}
```

12.4 Problema Culegeri

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

Ultima atracție a lotului național de informatică este șahul. Așa că Gimi s-a decis să-și deschidă o afacere, aceea fiind să vândă culegeri de șah. Aceste culegeri conțin tactici nemaivăzute, deci cererea este ridicată.

În avans, el a primit lista de comenzi. Zilnic, timp de N zile, el va trebui să livreze c_i culegeri la finalul zilei i. Gimi și-a început afacerea prin cumpărarea unei fabrici ce poate produce K culegeri pe zi, stocul inițial de culegeri fiind $S = 0$.

Zilnic, el poate alege una din următoarele două opțiuni:

- Gimi oprește producția în această zi și modernizează fabrica. În urma îmbunătățirilor, capacitatea de producție a fabricii va crește cu 1 ($K \leftarrow K + 1$).
- Gimi urmărește cu atenție procesul de printare. Astfel, stocul de culegeri va crește cu K ($S \leftarrow S + K$).

La finalul zilei i, el va livra c_i culegeri ($S \leftarrow S - c_i$). Dacă nu are cel puțin c_i culegeri în stoc, atunci lipsa lui de punctualitate va fi evidențiată, iar activitatea nu va mai putea continua.

Cerință

Cunoscând numărul de zile în care se desfășoară activitatea, producția zilnică inițială, respectiv lista cu cele N comenzi, să se rezolve următoarele cerințe:

- dacă $T = 1$, atunci Gimi vrea să afle numărul maxim de culegeri pe care le poate avea în stoc la finalul ultimei zile (a N-a zi).
- dacă $T = 2$, atunci Gimi vrea să afle numărul maxim de culegeri pe care le poate avea în stoc la finalul celei de-a i-a zi, pentru fiecare i de la 1 la N.

Date de intrare

Prima linie din fișierul de intrare conține trei numere T, N și K, separate prin câte un spațiu. A doua linie va conține N numere, unde al i-lea număr reprezintă c_i, comanda de culegeri din a i-a zi.

Date de ieșire

- Dacă $T = 1$, atunci fișierul de ieșire va conține un singur număr natural, reprezentând numărul maxim posibil de culegeri rămase în stoc la finalul celei de-a N-a zi.
- Dacă $T = 2$, atunci fișierul de ieșire va conține, pe o singură linie, N numere naturale separate prin câte un spațiu, unde al i-lea număr reprezintă numărul maxim posibil de culegeri rămase în stoc la finalul celei de-a i-a zi.

Restricții

- $1 \leq N \leq 500\,000$
- $0 \leq K \leq N$
- $0 \leq c_i \leq N \cdot K$
- Se garantează că există o modalitate prin care Gimi își va putea desfășura activitatea până în ultima zi inclusiv.

- Strategia utilizată pentru ziua i nu trebuie neapărat să fie continuată în ziua $i+1$. Se poate folosi o altă strategie, diferită de cea pentru ziua precedentă.

#	Puncte	Restricții
1	7	$T = 1$ și $1 \leq N \leq 15$
2	14	$T = 1$ și $15 < N \leq 2\,000$
3	19	$T = 1$ și $2\,000 < N \leq 500\,000$
4	8	$T = 2$ și $1 \leq N \leq 15$
5	21	$T = 2$ și $15 < N \leq 2\,000$
6	31	$T = 2$ și $2\,000 < N \leq 500\,000$

Exemple

culegeri.in	culegeri.out
2 5 2 1 1 3 1 3	1 2 1 2 2
1 5 2 1 1 3 1 3	2

Explicații

Pentru primul exemplu, $T = 2$, deci se va rezolva a doua cerință. Notăm cu „P x” zilele în care vom produce x culegeri, iar cu „I 1” zilele în care vom crește producția cu 1. Strategiile optime, pentru fiecare zi, ar putea fi:

- ziua 1: P 2
- zilele 1-2: P 2 → P 2
- zilele 1-3: P 2 → P 2 → P 2
- zilele 1-4: P 2 → I 1 → P 3 → P 3
- zilele 1-5: P 2 → I 1 → P 3 → P 3 → P 3

Se observă că strategiile diferă în funcție de ziua în care se vor termina experimentele. Pentru zilele 1-4, capacitatea de producție va crește în a doua zi, dar pentru zilele 1-3, capacitatea de producție nu se va modifica.

În plus, este imposibilă creșterea capacității de producție în prima zi, deoarece stocul ar fi zero, iar Gimi trebuie să livreze o culegere.

Pentru al doilea exemplu, $T = 1$, deci se va rezolva prima cerință. Se va afișa doar stocul maxim posibil la finalul ultimei zile.

12.5 Rezolvarea problemei Culegeri

În acest editorial vom analiza numai cerința $T = 2$, încât prima cerință este inclusă în aceasta.

Soluție în $\mathcal{O}(2^N \cdot N)$

Această soluție presupune generarea tuturor scenariilor posibile. În fiecare zi, avem două opțiuni: să creștem producția sau să printăm culegeri. Fiind N zile, vor fi $\mathcal{O}(2^N)$ scenarii, iar apoi în $\mathcal{O}(\mathcal{N})$

putem simula fiecare scenariu. Putem reduce complexitatea la $\mathcal{O}(2^N)$ dacă menținem incremental stocul pe parcursul recursivității. Această soluție va obține în jur de 15 puncte.

Soluție în $\mathcal{O}(N^2)$

Putem construi următoarea dinamică:

$dp[i][j]$ = stocul maxim posibil de la finalul celei de-a i-a zi, dacă în j dintre zile am crescut capacitatea de producție.

Relația de recurență va fi:

$$dp[i][j] = \max(dp[i-1][j] + (k+j), dp[i-1][j-1]) - c[i]$$

$dp[i-1][j]+(k+j)$ reprezintă soluția în care printăm culegeri. Plecăm de la o soluție cu capacitatea de producție $k+j$ (deci în j dintre zile am crescut producția) și printăm $k+j$ culegeri.

$dp[i-1][j-1]$ reprezintă soluția în care decidem să creștem capacitatea de producție, fără să modificăm stocul. Numărul de culegeri din stoc va fi egal cu cel de la ziua precedentă.

La final, va trebui să scădem $c[i]$ din această sumă, fiind obligați să livrăm $c[i]$ culegeri. Se poate obține o valoare negativă. În acest caz, este imposibil să obținem o soluție validă pentru perechea (i, j), și nu va fi folosită în crearea altor soluții.

Stocul maxim de la finalul celei de-a i-a zi va fi

$$S_i = \max(dp[i][0],\ dp[i][1],\ \ldots,\ dp[i][i])$$

O optimizare care se poate aduce acestei soluții este aceea că nu este necesară crearea unei matrici, ci doar păstrarea a două linii. Observăm ca linia $dp[i]$ este calculată strict pe baza valorilor de pe linia $dp[i-1]$, deci putem reduce memoria utilizată. O astfel de soluție va obține în jur de 50 de puncte.

Soluție în $\mathcal{O}(N)$

Una dintre observațiile de la care pleacă această soluție este aceea că investițiile (adică creșterea capacității de producție) trebuiesc făcute cât mai devreme posibil.

O altă observație este că nu toate investițiile pot fi profitabile. De exemplu, este inutil să investim în ultima zi. Pe caz general, dacă putem produce K culegeri, nu are sens să investim de la ziua $N-K$ încolo.

O ultimă observație este aceea că noi putem simula un scenariu în care decidem să investim la ziua i. Dacă stocul nu a ajuns niciodată negativ, atunci investiția este bună. Altfel, suntem obligați să renunțăm la ea (și să privim progresul fabricii ca și când această investiție nu s-a făcut niciodată).

Având în vedere cele de mai sus, pornim cu următoarea soluție: construim o stivă cu zilele în care investim. Presupunem că vom face orice investiție care are potențialul să fie profitabilă – deci la ziua i vom investi dacă este posibil și vom adăuga ziua i pe stivă.

Dacă stocul ajunge negativ, suntem obligați să renunțăm la investiții. Atunci, vom elimina investițiile una câte una de pe stivă și vom modifica stocul. Dacă ne aflăm la ziua i, și eliminăm o investiție făcută în ziua j, vom pierde $i-j$ zile în care am avut producția $k+1$, iar în ziua j vom produce k. Deci, vom câștiga $(i-j+1)\cdot k-(i-j)\cdot(k+1)$ culegeri, adică $k-(i-j)$.

Dacă o investiție s-a dovedit a fi profitabilă, nu are cum să fie scoasă de pe stivă - deoarece se garantează că există o soluție. Astfel, putem face acest proces zilnic (presupunem că putem investi și renunțăm la investiții dacă este nevoie), iar la final vom avea stocul maxim pentru ziua N.

Pentru a obține stocul maxim pentru celelalte zile, putem merge înapoi - de la ziua N la ziua 1. Știm pe baza soluției pentru ziua N că avem pe stivă singurele investiții care merită făcute (o soluție pentru ziua i nu va avea investiții diferite de soluția pentru ziua N). Așa că, pe măsură ce ne „întoarcem" în timp (iterăm cu i de la ziua $N-1$ spre ziua 1), renunțăm la investițiile care devin neprofitabile (în loc să avem ultima zi N, acum ultima zi este i - potențialul profit va scădea).

Această soluție va obține 100 de puncte. Alte implementări pot construi un vector cu investițiile făcute, iar apoi căuta ternar începând cu ce investiție să renunțe, obținând punctaje asemănătoare.

12.6 Cod-sursă pentru problema Culegeri

```cpp
// Autor: Ioan-Cristian Pop
#include <fstream>
#include <stack>

using namespace std;

const int NMAX = 500000;

long long c[NMAX + 1];
long long ans[NMAX + 1];
stack <int> st;

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

int main() {
    int n, task;
    long long k;
    fin >> task >> n >> k;

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

    long long s = 0;
    for (int i = 1; i <= n; ++i) {
        if (s >= c[i] && n - i > k) { // Presupunem ca putem investi, iar investitia se merita
            k = k + 1;
            s = s - c[i];
            st.push(i);
        } else if (s + k >= c[i]) { // Nu putem investi, dar e suficient sa producem
            s = s + k - c[i];
        } else { // Renuntam la o parte din investitii
            s = s + k;
            while (!st.empty() && s < c[i]) {
                int j = st.top();
                st.pop();

                // Renuntam la investitia din ziua j
                k--;
```

```
                s = s - (i - j) + k; // In i - j zile, am avut productia k + 1.
            }

            if (s < c[i]) {
                fout << "IMPOSIBIL\n";
                return 0;
            }

            s -= c[i];
        }
    }

    ans[n] = s;
    if (task == 1) {
        fout << ans[n] << "\n";
        return 0;
    }

    for (int i = n - 1; i >= 1; --i) {
        // Trecerea de la ziua i+1 la ziua i
        s = s - k + c[i + 1];
        // Renuntam la investitiile care devin neprofitabile la momentul de fata
        while (!st.empty() && i - st.top() < k) {
            int j = st.top();
            st.pop();

            k--;
            s = s - (i - j) + k;
        }
        ans[i] = s;
    }

    for (int i = 1; i <= n; ++i) {
        fout << ans[i] << " ";
    }
    fout << endl;
    return 0;
}
```

12.7 Problema Teatru

Propusă de: Instr. Cătălin Frâncu, Clubul Nerdvana, București

Marele dramaturg BONIFACCI a uimit lumea teatrului cu noua sa capodoperă, *Dada?!*.

Pentru că teatrul modern este un pic minimalist, piesa are un singur actor. Pentru că teatrul modern este un pic absurd, Actul 1 constă doar din replica 'A', iar Actul 2 constă doar din replica 'D'. După cum vedeți, în această piesă de teatru, **replica** este descrisă printr-un singur caracter. Și, pentru că teatrul modern este un pic repetitiv, fiecare act K, începând cu al treilea inclusiv, este obținut prin concatenarea actelor $K-2$ și $K-1$, în această ordine. Piesa are N acte, numerotate începând de la 1 până la N, toate construite după această regulă. În cadrul fiecărui act, replicile sunt numerotate începând tot de la 1.

De exemplu, primele 7 acte sunt (vă rugăm să nu aplaudați până la final):

Actul	**Textul**
1	A
2	D
3	AD
4	DAD
5	ADDAD
6	DADADDAD
7	ADDADDADADDAD

Cerință

La repetiție, actorul și-a propus să recite, cu intonație, L replici alăturate din actul al N-lea, începând cu replica numărul P, până la replica cu numărul $P+L-1$. Determinați secvența de caractere rostite de actor la repetiție.

Date de intrare

Fișierul de intrare `teatru.in` conține pe prima linie trei numere naturale nenule N, P și L, separate între ele prin câte un spațiu, având semnificațiile de mai sus.

Date de ieșire

Fișierul de ieșire `teatru.out` va conține, pe prima linie, L caractere 'A' sau 'D', reprezentând, în ordine, replicile spuse de actor la repetiție.

Restricții

- Fie S_N numărul de replici din actul al N-lea. Se garantează că $S_N \leq 10^{18}$.
- $1 \leq L \leq 2\,000\,000$.
- $1 \leq P \leq S_N - L + 1$ (cu alte cuvinte, există L replici în actul N începând cu replica numărul P).

#	Puncte	Restricții
1	7	$N \leq 35$, $L = 1$
2	13	$N \leq 35$, $L \leq 200\,000$
3	17	$N \leq 40$, $L \leq 200\,000$
4	19	$N \leq 46$, $L \leq 200\,000$
5	21	$L \leq 200\,000$
6	23	Nu există restricții suplimentare.

Exemple

teatru.in	teatru.out	Explicații
7 3 9	DADDADADD	Actul al 7-lea are textul complet ADDADDADADDAD. Trebuie afișate caracterele de pe pozițiile 3-11.
12 80 8	ADDADADD	

12.8 Rezolvarea problemei Teatru

Fie F_N al N-lea număr al lui Fibonacci. Observăm că actul N are lungime F_N. Precalculăm primele N numere Fibonacci.

Soluție în memorie F_{N+2}

Pentru primele două subtaskuri, putem pur și simplu crea prin concatenare toate actele de la 1 la N. Apoi tipărim subsecvența de la P la $P + L - 1$ din actul N. Această soluție necesită timp și memorie:

$$F_N + F_{N-1} + F_{N-2} + \cdots + F_2 + F_1 = F_{N+2} - 1$$

și se va încadra în 128 MB de memorie doar pentru $N \leq 38$. La limită, putem folosi și un singur bit pentru fiecare literă (având doar două litere distincte); astfel această versiune poate rezolva și subtaskul 3.

Soluție în memorie F_N

Observăm că actele $2, 3, \ldots, N - 1$ sunt sufixe ale actului N. De aceea, putem folosi o singură zonă de memorie de F_N caractere în care construim, de la dreapta spre stânga, actele $2, 3, \ldots, N$. Astfel memoria necesară este doar de F_N octeți, suficientă până la $N \leq 40$.

Soluție în $\mathcal{O}(N \cdot L)$

$P \leq K < P + L$ și calculăm replica de pe fiecare poziție. Fie $C(K, N)$ replica de pe poziția K din actul N. Putem defini aceste valori prin cazurile de bază $C(1, 1) =$ 'A', $C(1, 2) =$ 'D' și recurența:

$$C(K,N) = \begin{cases} C(K, N-2) & \text{dacă } K \leq F_{n-2} \\ C(K - F_{N-2}, N-1) & \text{dacă } K > F_{n-2} \end{cases}$$

Diferența între subtaskurile 4 și 5 este că până la $F_{46} = 1.836.311.903$ putem opera pe `int`, iar mai departe este necesar tipul `long long`.

Soluție în $\mathcal{O}(N + L)$

Cât timp intervalul $[P, P + L)$ este complet inclus în actul $N - 2$ sau $N - 1$, reducem problema la acel act. Cînd P și $P + L$ se află respectiv în actele $N - 2$ și $N - 1$, reducem problema la a tipări un sufix al actului $N - 2$ și un prefix al actului $N - 1$.

Prefixul unui act N, dacă este mai scurt decât actul $N - 2$, se reduce la un prefix al acelui act. Altfel prefixul constă din actul $N - 2$ în totalitate și un prefix al actului $N - 1$. Similar, sufixul unui act N fie se reduce la un sufix al actului $N - 1$, fie la un sufix al actului $N - 2$ și actul $N - 1$ în totalitate. Cazul de bază, dacă coborâm suficient (eventual până la actul 1), este un act în totalitate (dacă sufixele și prefixele devin vide).

Întrucât la fiecare nivel al recurenței există cel mult un prefix și un sufix, vor exista cel mult N din fiecare. Pentru a tipări actele acoperite în întregime, le putem descompune naiv (recursiv) până la lungime 1, cu un efort total $\mathcal{O}(L)$. Pentru un plus de viteză, putem precalcula primele 10-20 de acte.

Altă soluție în $\mathcal{O}(N + L)$

Să inserăm (ipotetic) în fiecare act o bară verticală la locul concatenării. Acum, să analizăm replica 5 din actul 8 și să o regăsim în actele anterioare. Am notat-o cu X în loc de D, pentru ușurință.

```
Actul 8: DADAXDAD|ADDADDADADDAD
Actul 6: DAD|AXDAD
Actul 5: AX|DAD
Actul 3: A|X
Actul 2: X
```

Pornind din actul 8, am coborât de partea stângă a barei și am ajuns la actul 6. De acolo am coborât de partea dreaptă a barei până la actul 5, etc. Notăm asta ca: `8 s 6 d 5 s 3 d 2`. Tipărim A sau D după acum actul final este 1 sau 2. Iată traseele replicilor 5-9:

```
Replica 5: 8 s 6 d 5 s 3 d 2      --> D
Replica 6: 8 s 6 d 5 d 4 s 2      --> D
Replica 7: 8 s 6 d 5 d 4 d 3 s 1  --> A
Replica 8: 8 s 6 d 5 d 4 d 3 d 2  --> D
Replica 9: 8 d 7 s 5 s 3 s 1      --> A
```

Ca mod de construcție, observăm că pe stânga coborâm două niveluri, iar pe dreapta doar unul. Mai observăm că traseele a două replici consecutive diferă la ultimul moment unde prima replică coboară pe stânga, fix lângă bară, iar a doua coboară pe dreapta. Așadar, algoritmul construiește traseul primei replici ca pe o stivă cu actul cel mai mic la vârf. Apoi, de L ori,

- tipărește caracterul corect;
- coboară în stivă până la primul `s`;
- schimbă `s`-ul în `d`;

- urcă în stivă, pe stânga, din două în două niveluri, până la 1 sau 2.

La prima vedere, pare că algoritmul este $\mathcal{O}(NL)$. Totuși, să analizăm comportamentul stivei. Dacă la un moment dat ultimele direcții sunt `s d ... d`, după trecerea la replica următoare direcțiile vor fi `d s ... s`. Majoritatea trecerilor vor modifica doar ultimul element din stivă din `s` în `d`. Recunoaștem aici incrementarea unui contor binar, unde `s` $= 0$ și `d` $= 1$, care are cost amortizat (mediu) de două operații per incrementare. Așadar, complexitatea totală este $\mathcal{O}(N)$ pentru construcția stivei inițiale, apoi $\mathcal{O}(L)$ amortizat.

Soluție alternativă (Prof. Stelian Ciurea)

O soluție frumoasă din punct de vedere matematic pornește de la observația că, pentru a k-a poziție numărând de la dreapta, replica este 'D' dacă și numai dacă

$$\lfloor (k+1)\phi \rfloor - \lfloor k\phi \rfloor = 2,$$

unde $\phi = (1+\sqrt{5})/2$ este numărul de aur. Din păcate, implementarea directă, chiar și cu `long double`, greșește ocazional din cauza erorilor de precizie. Ea poate fi dusă până la 100 de puncte cu gestionarea proprie a zecimalelor.

12.9 Cod-sursă pentru problema Teatru

```
// Autor: Cătălin Frâncu
// Algoritmul cu prefixe și sufixe.
#include <stdio.h>
#define MAX_FIB 87 // Suficient, căci F_88 > 10^18.
const char acts12[4] = ".AD"; // Cazurile de bază.

long long fib[MAX_FIB + 1] = { 0, 1 };

void precompute(int n) {
  for (int i = 2; i < n; i++) {
    fib[i] = fib[i - 1] + fib[i - 2];
  }
}

// Tipărește actul n în întregime.
void whole(FILE* f, int n) {
  if (!n) {
    // nimic
  } else if (n <= 2) {
    fputc(acts12[n], f);
  } else {
    whole(f, n - 2);
    whole(f, n - 1);
  }
}

// Tipărește replicile din actul n începînd cu poziția p.
void suffix(FILE* f, int n, long long p) {
  if (!p) {
    whole(f, n);
  } else if (p < fib[n - 2]) {
    suffix(f, n - 2, p);
    whole(f, n - 1);
  } else {
    suffix(f, n - 1, p - fib[n - 2]);
```

```
  }
}

// Tipărește primele p replici din actul n.
void prefix(FILE* f, int n, long long p) {
  if (!p) {
    // Nimic.
  } else if (p < fib[n - 2]) {
    prefix(f, n - 2, p);
  } else {
    whole(f, n - 2);
    prefix(f, n - 1, p - fib[n - 2]);
  }
}

int main() {
  int n, l;
  long long p;
  FILE* f = fopen("teatru.in", "r");
  fscanf(f, "%d %lld %d", &n, &p, &l);
  p--;
  fclose(f);

  precompute(n);

  // Cît timp ambele capete încap în același act, coboară în acel act.
  while ((n > 2) && (p >= fib[n - 2] || p + l < fib[n - 2])) {
    if (p >= fib[n - 2]) {
      p -= fib[n - 2];
      n--;
    } else {
      n -= 2;
    }
  }

  // Acum că p și p + l sînt în acte diferite, tipărește prefixul și sufixul.
  f = fopen("teatru.out", "w");
  if (n <= 2) {
    whole(f, n);
  } else {
    suffix(f, n - 2, p);
    prefix(f, n - 1, p + l - fib[n - 2]);
  }
  fputc('\n', f);
  fclose(f);
  return 0;
}
```

```
// Autor: Cătălin Frâncu
// Algoritmul cu stivă.
#include <stdio.h>
#define MAX_FIB 87 // Suficient, căci F_88 > 10^18.

struct {
  bool right; // false = stînga, true = dreapta
  int level;
} st[MAX_FIB + 1];

long long fib[MAX_FIB + 1] = { 0, 1 };

int main() {
  int n, l;
```

```
  long long p;
  FILE* f = fopen("teatru.in", "r");
  fscanf(f, "%d %lld %d", &n, &p, &l);
  p--;
  fclose(f);

  for (int i = 2; i < n; i++) {
    fib[i] = fib[i - 1] + fib[i - 2];
  }

  f = fopen("teatru.out", "w");

  // Construiește stiva de coborîri stînga-dreapta.
  st[0] = { false, n };
  int ss = 1;
  do {
    if (p < fib[n - 2]) {
      n -= 2;
      st[ss++] = { false, n };
    } else {
      p -= fib[n - 2];
      n--;
      st[ss++] = { true, n };
    }
  } while (n > 2);

  while (l--) {
    // Tipărește caracterul curent.
    fputc(st[ss - 1].level == 1 ? 'A' : 'D', f);

    // Urcă în stivă pînă la primul s
    while (st[ss - 1].right) {
      ss--;
    }

    // Avansează
    st[ss - 1].right = true;
    st[ss - 1].level++;

    // Coboară pe stînga pînă la un nivel terminal.
    while (st[ss - 1].level > 2) {
      st[ss] = { false, st[ss - 1].level - 2 };
      ss++;
    }
  }

  fputc('\n', f);
  fclose(f);
  return 0;
}
```

Partea a IV-a

Cupa SEPI

Câmpulung Muscel, 26-31 iulie 2023

Capitolul 13

Cupa SEPI

13.1 Problema All Numbers

Propusă de: prof. Chește Ciprian, Liceul Tehnologic „Grigore C. Moisil" Buzău

Fie N un număr natural.

Cerințe

Să se determine suma tuturor numerele naturale cu următoarele proprietăți:

- descompunerea acestora în factori primi conține aceiași factori primi ca și N;
- suma exponenților descompunerii în factori primi a acestora este aceeași ca a lui N;
- au un număr maxim de divizori.

Date de intrare

Pe prima linie se va găsi numărul natural N.

Date de ieșire

Pe prima linie se va găsi un singur număr natural, reprezentând restul împărțirii sumei determinate la numărul $1\,000\,000\,007$.

Restricții

- $1 \leq N \leq 10^{12}$

#	Puncte	Restricții
1	24	$1 \leq N < 10^6$
2	28	$10^6 \leq N < 10^{10}$
3	48	$10^{10} \leq N \leq 10^{12}$

Exemple

stdin	stdout
20	70
945	7455
99999999	833333155

Explicații

- $20 = 2^2 \cdot 5^1$, $50 = 2^1 \cdot 5^2$ și ambele numere au un număr maxim de divizori, egal cu 6. Restul împărțirii sumei lor la numărul 1 000 000 007 este egal cu 70.
- $1575 = 3^2 \cdot 5^2 \cdot 7^1$, $2205 = 3^2 \cdot 5^1 \cdot 7^2$, $3675 = 3^1 \cdot 5^2 \cdot 7^2$ și toate cele trei numere au un număr maxim de divizori, egal cu 18. Restul împărțirii sumei lor la numărul 1 000 000 007 este egal cu 7455.
- Există cinci numere naturale care respectă proprietățile impuse: 566 666 593, 366 666 612, 433 333 295, 366 666 663 și 99 999 999. Suma acestor numere este egală cu 1 833 333 162, iar restul împărțirii acestui număr la 1 000 000 007 este egal cu 833 333 155.

13.2 Rezolvarea problemei All Numbers

Cunoștințe necesare

- Descompunere în factori primi
- Numărul divizorilor
- Combinatorică

Fie $N = p_1^{e_1} p_2^{e_2} \ldots . p_k^{e_k}$ cu $e_i \neq 0, 1 \leq i \leq K$ și $S = e_1 + e_2 + \cdots + e_k$

Vom descompune numărul N în factori primi și vom reține într-o structură de date factorii primi împreună cu exponenții acestora. Se cunoaște că numărul de divizori ai lui N poate fi calculat cu formula $(e_1 + 1)(e_2 + 1) \ldots (e_k + 1)$ Pentru a obține un număr maxim de divizori cu aceeași sumă a exponenților S, trebuie ca exponenții să fie cât mai apropiați, adică să difere cât mai puțin. Doar în acest mod se poate ajunge la un produs maxim pentru că și $S + K$ este constantă. Vom calcula câtul C și restul R al împărțirii dintre suma exponenților și numărul exponenților. Vom calcula apoi numărul T care are aceeași factori primi ca și N ridicați la C, urmând ca restul R să fie distribuit între cei K factori primi. De exemplu pentru $N = 945 = 3^3 5^1 7^1$ avem $S = 5, K = 3$ deci $C = 1$ și $R = 2$. Restul $R = 2$ poate fi distribuit la cei 3 factori în 3 moduri diferite așadar se pot obține numerele $3^2 5^2 7^1$, $3^2 5^1 7^2$ și $3^1 5^2 7^2$ adică 1575, 2205 și 36754 care au fiecare 18 divizori.

Se observă de aici că trebuie generate toate combinările de K luate câte R printr-un mecanism de tip backtracking sau un algoritm asemănător și apoi generate numerele căutate plecând de la numărul T. Aceste numere vor fi apoi însumate modulo $10^9 + 7$.

Soluție alternativă

Propusă de: stud. Ștefan-Cosmin Dăscălescu, Universitatea din București

Mai întâi, descompunem N în factori primi și calculăm exponenții, împreună cu suma lor asemănător cu prima soluție.

Dacă numărul exponenților este x și suma lor este y, vrem ca fiecare exponent să apară cel puțin la puterea y/x ori și cel mult la puterea $y/x + 1$ ori, numărul exponenților ce apar la puterea $y/x + 1$ fiind dat de $y \,\%\, x$.

Astfel, vom putea folosi o abordare bazată pe metoda programării dinamice, $dp[i][j][k][l]$ fiind suma tuturor numerelor optime care au folosit primii i exponenți, numărul total de exponenți folosit este j, mai putem folosi k exponenți cu puterea q și l exponenți cu puterea $q+1$, unde q este y/x de mai sus.

Pentru a face actualizarea de la o stare la următoarea stare, avem două cazuri în funcție de valorile din k și l, putând fie să adăugăm numărul prim curent la puterea q, fie același număr prim la puterea $q+1$. Trebuie avut grijă la calcularea puterilor numerelor prime din descompunerea lui N pentru evitarea calculelor ce nu sunt necesare.

Complexitatea finală a soluției va fi $O(\sqrt{N}+x^3 \cdot y)$, unde x și y sunt numerele de mai sus, x fiind cel mult egal cu 12 (numărul factorilor primi distincți ai lui N) iar y fiind cel mult egal cu 40. Totuși, un x mare nu permite și un y mare și viceversa.

13.3 Cod-sursă pentru problema All Numbers

```
// prof. Chesca Ciprian - sursa 100 p
#include <iostream>
#include <cassert>
#define ll long long
#define MOD 1000000007

using namespace std;

ll p[100],e[100],st[100];           // p[] - vectorul factorilor primi
ll bo, so, n;                       // e[] - vectorul exponentilor
ll H = 1, W = 0;

void desc(ll n, ll &k,ll p[],ll e[])
{
ll i, fact;

k = 0;
// extrag factorii de 2
fact = 0;
while (n%2 == 0)
   {
    n /= 2;
    fact++;
    }

if (fact) {p[++k] = 2; e[k] = fact;}

// descompun w in factori primi
i = 3;
while (i*i <= n)
    {
     if (n%i==0)
        {
        fact = 0;
        while (n%i == 0) { n /= i;fact++;}
        if (fact) {p[++k] = i; e[k]=fact;}
        }
        i += 2;
    }

if (n > 1) {p[++k] = n; e[k] = 1;}
}
```

```
// Subprograme pentru generarea combinarilor cu backtracking
void afiscomb(int k)
{
    int i;
    ll  A = 1;

    // adaug restul exponentilor
    for(i = 1; i < k;i++)
          A = (A*p[st[i]])%MOD;

    // insumez modulo
    W = (W + A)%MOD;
}

void comb(int n, int t, int k)
{
    if (k == t + 1) afiscomb(k);
    else
        for(int i = st[k-1] + 1; i <= n;i++)
            {
            st[k] = i;
            comb(n, t, k + 1);
            }
}

int main()
{
    ll k, i, j, cat, rest;

    cin >> n;
    // descompun numarul in factori primi
    desc(n,k,p,e);

    // calculez suma exponentilor
    // bo este notatia pentru functia big omega si reprezinta suma
    // exponentilor din descompunerea in factori primi
    // so este notatia pentru little omega si reprezinta numarul factorilor
    // primi din descompunerea in factori primi
    bo = 0; so = k;
    for(i = 1;i <= k;i++)
        bo += e[i];

    // calculez catul si restul lui bo la so
    cat = bo / so;
    rest = bo % so;

    // formez numarul care are aceeasi factori primi si toti exponentii egali cu cat
    H = 1;
    for(i = 1; i <= so; i++)
      for(j = 1; j <= cat; j++)
                H = (H * p[i])%MOD;

    // generez combinari de so luate cate rest
    comb(so, rest, 1);

    cout << (H*W)%MOD << "\n";
    return 0;
}
```

```
/*
    alln - Stefan Dascalescu
```

```cpp
/*
Se poate observa ca daca avem x exponenti si suma lor este y, vrem sa-i
impartim cat mai echilibrat (y/x pentru toate, eventual pentru unele avem
y/x+1)

Apoi, putem rula o dinamica pe 4 dimensiuni care se foloseste de descompunerea
echilibrata pentru a afla suma tuturor numerelor, divizor cu divizor.
*/
#include<bits/stdc++.h>
using namespace std;

const int mod = 1000000007;

// suma tuturor numerelor optime care au folosit primii i exponenti, nr total
// folosit este j, mai sunt k exponenti cu puterea q si l exponenti cu puterea
// q+1
long long dp[12][52][12][12];
int main()
{

  long long n;
  cin >> n;

  int sum = 0; // suma exponentilor

  vector<pair<long long, int> > exponenti; // exponentii

  for(long long i = 2; i * i <= n; i++)
    if(n % i == 0)
    {
      exponenti.push_back({i, 0});
      while(n % i == 0)
      {
        exponenti.back().second++;
        sum++;
        n /= i;
      }
    }

  if(n > 1)
  {
    exponenti.push_back({n, 1});
    sum++;
  }


  int q = sum / exponenti.size();
  int mare = sum % exponenti.size();
  int mic = exponenti.size() - mare;

  long long put = 1;
  for(int i = 1; i <= q; i++)
    put = (put * exponenti[0].first) % mod;


  if(mare)
    dp[0][q+1][mic][mare-1] = (put * exponenti[0].first) % mod;
  if(mic)
    dp[0][q][mic-1][mare] = put;

  for(int i = 0; i + 1 < (int) exponenti.size(); i++)
  {
```

```
    long long put = 1;
    for(int p = 1; p <= q; p++)
      put = (put * exponenti[i+1].first) % mod;
    for(int j = 0; j <= sum; j++)
    {
      for(int nrmic = 0; nrmic < (int) exponenti.size(); nrmic++)
      {
        for(int nrmare = 0; nrmare < (int) exponenti.size(); nrmare++)
        {
          if(dp[i][j][nrmic][nrmare] == 0)
            continue;

          if(nrmic)
          {
            dp[i+1][j + q][nrmic - 1][nrmare] += (dp[i][j][nrmic][nrmare] * put);
            dp[i+1][j + q][nrmic - 1][nrmare] %= mod;
          }

          if(nrmare)
          {
            long long val = dp[i][j][nrmic][nrmare] * put;
            val %= mod;
            val = val * exponenti[i+1].first;
            val %= mod;
            dp[i+1][j + q + 1][nrmic][nrmare - 1] += val;
            dp[i+1][j + q + 1][nrmic][nrmare - 1] %= mod;
          }
        }
      }
    }
  }
  cout << dp[exponenti.size() - 1][sum][0][0] << '\n';
  return 0;
}
```

13.4 Problema Kpartit

Propusă de: prof. Victor Manz, Colegiul Național de Informatică „Tudor Vianu" București

Un șir x format din literele $x_0, x_1, \ldots, x_{n-1}$ cu $n > 0$ se numește *kpartit de ordin k* dacă există k și $p_1, p_2, \ldots, p_{k-1}$ cu proprietățile:

- $0 < k \leq n$
- $0 < p_1 < p_2 < \cdots < p_{k-1} < n$
- $x_i = x_j$ pentru orice $0 \leq i < j < p_1$
- $x_i = x_j$ pentru orice $p_1 \leq i < j < p_2$
- ...
- $x_i = x_j$ pentru orice $p_{k-1} \leq i < j < n$
- $x_{p_1-1} \neq x_{p_1}, x_{p_2-1} \neq x_{p_2}, \ldots, x_{p_k-1} \neq x_{p_k}$

De exemplu șirurile de lungime 8 *bbbawwww* și *daaaaddd* sunt kpartite de ordinul 3, șirul de litere de lungime 10 *eeeeezzzzz* este kpartit de ordinul 2, iar șirul de lungime 5 *yyyyy* este kpartit de ordinul 1.

Cerință

Scrieți un program care, pentru un șir de litere $x = (x_0, x_1, \ldots, x_{n-1})$ de lungime n și un număr natural k, $0 < k \leq n$ determină numărul minim de litere care trebuie eliminate din x astfel încât acesta să devină kpartit de ordin k. În cazul în care acest lucru nu e posibil, se va afișa -1.

Date de intrare

Pe prima linie se află numerele n și k, separate printr-un spațiu, cu semnificația de mai sus iar pe cea de-a doua linie termenii șirului dat $x_0, x_1, \ldots, x_{n-1}$. Aceștia nu vor fi separați prin spații.

Date de ieșire

Pe prima linie se află un singur număr natural, reprezentând rezultatul determinat.

Restricții

- $1 \leq k \leq n \leq 5\,000$
- șirul x este format doar din litere mici ale alfabetului englez

#	Puncte	Restricții
1	21	$n \leq 20$
2	36	$n \leq 100$
3	24	$n \leq 1\,000$
4	19	Nu există alte restricții

Exemple

stdin	stdout	Explicații
8 3 bbbawwww	0	Nu este necesară eliminarea niciunei litere.

8 2 bbbawwww	1	Este suficientă eliminarea unei litere a astfel încât șirul dat să devină kpartit de ordin 2.
10 5 eeeeezzzzz	-1	Nu este posibilă obținerea unui șir kpartit de ordinul 5 prin eliminarea unor litere ale șirului dat.
10 2 bcbbxxbccc	3	Este suficientă eliminarea unei litere c și a celor două litere x astfel încât șirul dat să devină kpartit de ordin 2.

13.5 Rezolvarea problemei Kpartit

Problema poate fi rezolvată folosind metoda programării dinamice.

Prima soluție, mai puțin eficientă, dar mai ușor de intuit constă în actualizarea în n pași a unei matrice cu k linii și $NL = 26$ coloane. Un element al acesteia, $lmax[nrs][p]$, are semnificația: lungimea maximă a unui subșir *kpartit* de ordinul nrs care se termină cu cea de-a p-a literă din alfabet.

La fiecare pas $i, 1 \leq i \leq n$, $lmax[nrs][p]$ se obține ca fiind maximul dintre $lmax[nrs][p]$ de la pasul $i-1$, la care se adaugă 1 și $\max\{1 + lmax[nrs-1][j], 0 \leq j < NL, j \neq p\}$.

Rezultatul cerut este diferența dintre n și maximul elementelor de pe linia k a matricei de la pasul n (care corespunde întregului șir s). Cazul particular în care trebuie afișat -1 corespunde situației în care toate elementele liniei k sunt nule.

Se observă că matricea este actualizată în $k \cdot NL$ pași, pentru fiecare dintre cele n prefixe ale șirului s citit. Rezultă astfel că algoritmul are complexitatea timp $O(n \cdot k \cdot NL)$ (NL reprezentând dimensiunea alfabetului, adică 26 în cazul nostru). O astfel de soluție va obține 81 de puncte.

Pentru a obține punctajul maxim trebuie făcută observația că nu este necesară parcurgerea tuturor celor $NL = 26$ de litere, ci este suficient să reținem doar pozițiile din alfabet pentru care se obțin cele mai mari două valori de pe fiecare linie nrs a matricei.

Notăm cu $indicemax1[nrs]$ = indicele (poziția în alfabet a) unei litere care poate fi ultima într-un subșir *kpartit* de ordinul nrs de lungime maximă și cu $indicemax2[nrs]$ = indicele unei litere care poate fi ultima într-un subșir *kpartit* de ordinul nrs de lungime maximă **dacă nu îl luăm în considerare** pe $indicemax1[nrs]$.

Cu ajutorul acestora, la fiecare pas $i, 1 \leq i \leq n$, $lmax[nrs][p]$ se obține ca fiind maximul dintre $lmax[nrs][p]$ de la pasul $i-1$, la care se adaugă 1 și:

- $1 + lmax[nrs-1][indicemax1[nrs-1]]$, dacă $p \neq indicemax1[nrs-1]$
- $1 + lmax[nrs-1][indicemax2[nrs-1]]$, dacă $p = indicemax1[nrs-1]$

După calculul fiecărui element $lmax[nrs][p]$, trebuie actualizate (dacă este cazul) $indicemax1[nrs]$ și $indicemax2[nrs]$. Trebuie avut grijă ca de fiecare dată să se păstreze relația $indicemax1[nrs] \neq indicemax2[nrs]$.

O astfel de soluție, având complexitatea timp $O(n \cdot k)$ și complexitatea spațiu $O(n \cdot NL)$ obține punctajul maxim.

13.6 Cod-sursă pentru problema Kpartit

```cpp
#include <bits/stdc++.h>

using namespace std;

const int N_MAX = 5000;
int N, K;
char C[N_MAX + 5], C2[N_MAX + 5];
int dp[N_MAX + 5][N_MAX + 5];
int max_dp[N_MAX + 5];
int freq[N_MAX + 5];

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);  cout.tie(0);

    cin >> N >> K;
    int N_init = N;
    cin >> C2;
    int L = 1;
    C[1] = C2[0];
    freq[1] = 1;
    for(int i = 1; i < N; i++) {
        if(C2[i] == C[L]) {
            freq[L]++;
        }
        else {
            C[++L] = C2[i];
            freq[L] = 1;
        }
    }
    N = L;

    for(int k = 1; k <= K; k++) {
        if(k > 1) {
            fill(max_dp, max_dp + N_MAX + 1, -1);
        }

        pair<int, int> mx1, mx2;
        for(int i = 1; i <= N; i++) {
            if(max_dp[C[i]] != -1) {
                dp[k][i] = freq[i] + max_dp[C[i]];
            }

            pair<int, int> mx;
            if(mx1.first == C[i]) {
                mx = mx2;
            }
            else {
                mx = mx1;
            }

            if(mx.second > 0) {
                dp[k][i] = max(dp[k][i], freq[i] + mx.second);
            }

            if(dp[k][i] > 0) {
                max_dp[C[i]] = dp[k][i];
            }
            if(dp[k - 1][i] > 0) {
```

```
                if(mx1.first == C[i]) {
                    mx1.second = max(mx1.second, dp[k - 1][i]);
                }
                else if(mx2.first == C[i]) {
                    mx2.second = max(mx2.second, dp[k - 1][i]);
                }
                else {
                    if(mx1.second < dp[k - 1][i]) {
                        mx2 = mx1;
                        mx1 = make_pair(C[i], dp[k - 1][i]);
                    }
                    else if(mx2.second < dp[k - 1][i]) {
                        mx2 = make_pair(C[i], dp[k - 1][i]);
                    }
                }

                if(mx2.second > mx1.second) {
                    swap(mx2, mx1);
                }
            }
        }
    }
    int ans = 0;
    for(int i = 1; i <= N; i++) {
        ans = max(ans, dp[K][i]);
    }
    if(ans == 0) {
        cout << "-1\n";
    }
    else {
        cout << N_init - ans << "\n";
    }
    return 0;
}
```

```
//O(n*k)
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
const int N = 5e3;
const int NL = 26;

char s[N+1];
int n, k, l_max[N+1][NL], indice_max_1[N+1], indice_max_2[N+1];

//l_max[nr_s][lit] = lung. max a unui subsir format din nr_s secv. care se
//termina cu litera lit
//
//indice_max_1[nr_s] = indicele unei litere care poate fi ultima intr-un
//subsir de lungime maxima format din nr_s secvente
//
//indice_max_2[nr_s] = indicele unei litere care poate fi ultima intr-un
//subsir de lungime al doilea maxim format din nr_s secvente
//
//ma asigur ca indice_max_2[nr_s] != indice_max_1[nr_s]

int main()
{
    cin >> n >> k >> ws;
    int nr = 0;
    char c = cin.get();
```

```
    while (!cin.eof() && 'a' <= c && c <= 'z')
    {
        s[nr++] = c;
        c = cin.get();
    }
    fill(indice_max_1, indice_max_1 + N, -1);
    fill(indice_max_2, indice_max_2 + N, -1);
    int nr_max_secv = 1;
    indice_max_1[nr_max_secv] = (int)(s[0] - 'a');
    l_max[nr_max_secv][(int)(s[0] - 'a')] = 1;
    for (int i = 1; i < n; i++)
    {
        if (s[i] != s[i-1])
        {
            nr_max_secv++;
        }
        int p = (int)(s[i] - 'a');
        int lim = min(nr_max_secv, k);
        for (int nr_s = 1; nr_s <= lim; nr_s++)//n_s = nr. de secvente
        {
            l_max[nr_s][p]++;
            if (p != indice_max_1[nr_s-1])
            {
                //pot adauga s[i] la sfarsitul sirului si am o secventa in plus
                l_max[nr_s][p] = max(l_max[nr_s][p],
                                     1 + l_max[nr_s-1][indice_max_1[nr_s - 1]]);
            }
            else
            {   //daca nu il pot folosi pe indice_max_1, adaug la sirul care
                //se termina cu indice_max_2_
                if (indice_max_2[nr_s - 1] != -1)
                {
                    l_max[nr_s][p] = max(l_max[nr_s][p],
                                         1 + l_max[nr_s-1][indice_max_2[nr_s - 1]]);
                }
            }
            if (indice_max_1[nr_s] == -1 ||
                l_max[nr_s][p] > l_max[nr_s][indice_max_1[nr_s]])
            {
                indice_max_2[nr_s] = indice_max_1[nr_s];
                indice_max_1[nr_s] = p;
            }
            else if (p != indice_max_1[nr_s] &&
                     (indice_max_2[nr_s] == - 1 ||
                      l_max[nr_s][p] > l_max[nr_s][indice_max_2[nr_s]]))
            {
                //indicii trebuie sa fie diferiti
                indice_max_2[nr_s] = p;
            }
        }
    }
    int rez = n - l_max[k][indice_max_1[k]];
    if (rez == n) rez = -1;
    cout << rez << "\n";
    return 0;
}
```

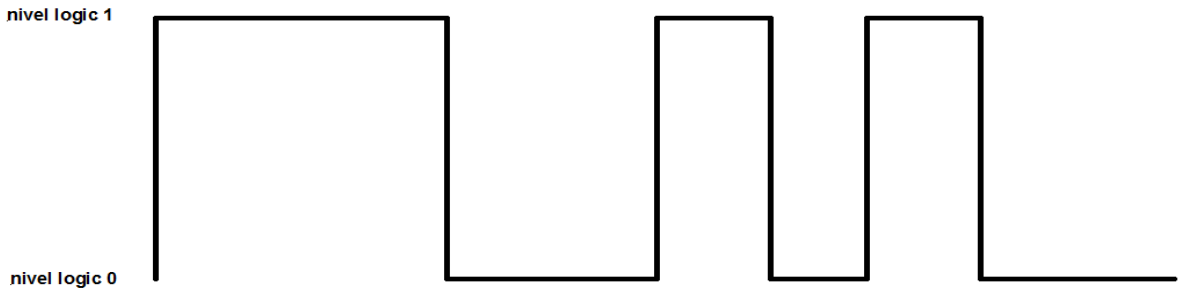
```
#include <iostream>
#include <string>
#include <vector>
#include <utility>
#include <assert.h>
```

```cpp
#include <algorithm>
const int SIGMA = 26;

int main() {
  int N, K;
  std::string S;
  std::cin >> N >> K >> S;

  std::vector<std::vector<int>> max_len(K + 1, std::vector<int>(SIGMA));
  max_len[1][(int)(S[0] - 'a')] = 1;
  std::vector<std::pair<int, int>> best(K + 1, std::make_pair(-1, -1));
  best[1] = std::make_pair((int)(S[0] - 'a'), -1);
  for (int i = 1, max_k = 1; i < N; i++) {
    int curr_letter = (int)(S[i] - 'a');
    if (S[i] != S[i - 1]) {
      max_k++;
    }

    for (int j = 1; j <= std::min(K, max_k); j++) {
      max_len[j][curr_letter]++;
      for (const int &op : {best[j - 1].first, best[j - 1].second}) {
        if (op != curr_letter && op != -1) {
          max_len[j][curr_letter] = std::max(max_len[j][curr_letter], 1 + max_len[j - 1][op]);
        }
      }

      if (best[j].first == -1 || max_len[j][best[j].first] < max_len[j][curr_letter]) {
        if (best[j].second != curr_letter) {
          best[j].second = best[j].first;
        }
        best[j].first = curr_letter;
      } else if ((best[j].second == -1 || max_len[j][best[j].second] < max_len[j][curr_letter])
                 && best[j].first != curr_letter) {
        best[j].second = curr_letter;
      }
    }
  }
  int answer = *std::max_element(max_len[K].begin(), max_len[K].end());
  std::cout << (answer > 0 ? N - answer : -1) << "\n";
  return 0;
}
```

13.7 Problema Semnal

Propusă de: prof. Chește Ciprian, Liceul Tehnologic „Grigore C. Moisil" Buzău

Un semnal digital reprezintă o succesiune de date reprezentate cu ajutorul a două niveluri de tensiune electrică denumite 0 logic și 1 logic și sunt prezente în toate electronicele digitale, în special în echipamentele de calcul și în cele utilizate în transmisia de date. Reprezentarea grafică sau așa numita formă de undă a unui semnal digital este o succesiune neîntreruptă de linii orizontale de nivel 0 sau de nivel 1, unite prin linii verticale, conform imaginii alăturate.



Cerință

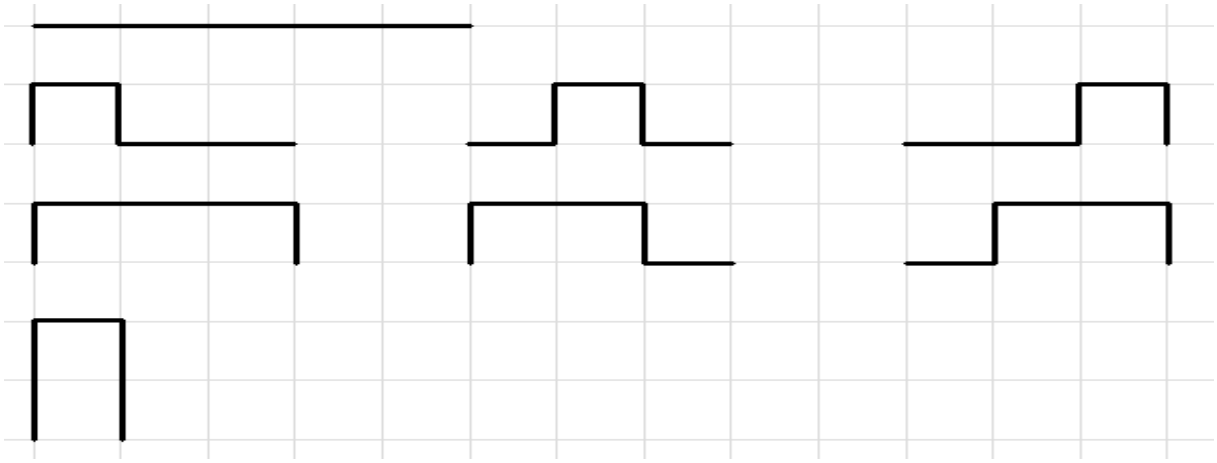
Fie L un număr natural nenul. Se consideră un **fir neîntrerupt** de lungime L. Să se determine numărul de forme de undă diferite notat cu U care se pot realiza folosind acest fir respectând următoarele proprietăți:

- O formă de undă începe și se termină pe nivelul logic 0;
- O formă de undă folosește cel mult două niveluri logice;
- Lungimile liniilor orizontale și verticale sunt doar numere naturale nenule;

Două forme de undă se consideră diferite dacă diferă prin cel puțin o linie.

Exemplu:

Pentru $L = 5$ se pot realiza $U = 8$ forme de undă diferite, conform figurii următoare:



Date de intrare

Pe prima linie se va găsi numărul natural L.

Date de ieșire

Pe prima linie se va găsi restul împărțirii numărului natural U la 3 000 017.

Restricții

- $1 \leq L \leq 10^6$

#	Puncte	Restricții
1	16	$1 \leq L \leq 34$
2	20	$35 \leq L \leq 20\,000$
3	64	$20\,001 \leq L \leq 10^6$

Exemple

stdin	stdout
5	8
2022	177722
470926	19116

Explicații

- Pentru primul exemplu: Se pot forma $U = 8$ forme de undă diferite folosind un fir de lungime $L = 5$ respectând proprietățile cerute, conform exemplului de mai sus.
- Pentru al doilea exemplu : Restul împărțirii numărului U ce reprezintă numărul de forme de undă diferite ce se pot forma folosind un fir de lungime $L = 2\,022$ la 3 000 017 este 177 722.
- Pentru al treilea exemplu : Restul împărțirii numărului U ce reprezintă numărul de forme de undă diferite ce se pot forma folosind un fir de lungime $L = 470\,926$ la 3 000 017 este 19 116.

13.8 Rezolvarea problemei Semnal

Etapa de raționament

Solutia problemei se încadrează în metoda *stars and bars* care este o abordare de rezolvare a problemelor de combinatorică. Se poate folosi când trebuie să determinăm numărul de modalități de a grupa un număr dat de obiecte identice.

Mai multe detalii despre această metodă puteți găsi aici: Stars and bars

Vom face o analiză în funcție de amplitudinea semnalului digital pe care o vom nota cu H sau cu alte cuvinte de diferența dintre nivelul logic 0 și nivelul logic 1.

- Pentru $H = 0$ se poate genera un singur semnal (o simplă linie dreaptă de lungime L). Acest rezultat este $comb(L + 1, 0)$ pe care îl vom explica în continuare.
- Pentru $H = 1$ distingem subcazurile:
  1. Dacă avem o singură trecere de la nivelul 0 la nivelul 1 și desigur o singură revenire atunci din lungimea L se scade 2, corespunzător celor două treceri de la un nivel la altul și avem apoi de așezat 2 linii verticale în $L–2 + 1$ poziții, adică $comb(L–1, 2)$
  2. Dacă avem două treceri de la un nivel la altul atunci din L scădem 4 și vom avea de așezat 4 linii verticale în $L–4 + 1$ poziții adică $comb(L - 3, 4)$

3. Ș.a.m.d. de unde vom găsi următoarea sumă de combinări $comb(L+1,0)+comb(L-1,2)+comb(L-3,4)+\ldots$

- Pentru $H=2$ raționamentul este asemănător și va conduce la suma de combinări $comb(L–3,2)+comb(L-7,4)+\ldots$

Sumele se vor calcula pentru toate valorile H care generează forme de undă, mai precis dacă L este număr impar atunci H poate fi $L/2$, iar dacă L este număr par, H poate fi $L/2–1$.

Exemplu

Pentru L = 10 avem:

- $H=0$, $comb(11,0)=1$;
- $H=1$, $comb(9,2)+comb(7,4)=36+35=71$
- $H=2$, $comb(7,2)=21$
- $H=3$, $comb(5,2)=10$
- $H=4$, $comb(3,2)=3$

În total sunt 106 semnale digitale distincte care se pot forma!

Etapa de implementare

În funcție de modul de calcul al combinărilor se pot distinge acum mai multe abordări de implementare:

- Combinări precalculate dinamic cu triunghiul lui Pascal dar cu valori limită de maxim 5 000 – soluție de aproximativ $20-30$ puncte
- Combinări calculate clasic în care fiecare produs care apare în formula combinărilor se calculează de fiecare dată împreună cu inversul modular necesar – soluție de aproximativ $30-40$ puncte
- Combinări în care fiecare produs este precalculat – soluție de aproximativ $40-50$ puncte.
- Combinări calculate cu teorema lui Lucas – soluție de $70-80$ puncte. Mai multe despre teorema lui Lucas puteți găsi aici: Teorema lui Lucas
- Combinări calculate în $O(1)$ după ce în prealabil au fost precalculate atât factoriale necesare cât și inversele modulare necesare – soluție de 100 puncte.

Soluție alternativă

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

Vom folosi următoarele notații:

- $f_0[n]$ = numărul semnalelor de lungime n care se termină cu _ (orizontal jos)
- $f_1[n]$ = numărul semnalelor de lungime n care se termină cu | (vertical urcare)
- $f_2[n]$ = numărul semnalelor de lungime n care se termină cu − (orizontal sus)
- $f_3[n]$ = numărul semnalelor de lungime n care se termină cu | (vertical coborâre)

Ne interesează $f_0[L]+f_3[L]$

Se observă relațiile:

- $f_0[n]=f_0[n-1]+f_3[n-1]$, deoarece putem prelungi cu 1 pas orizontal în jos orice semnal care se termină cu orizontal jos sau coborâre verticală
- $f_1[n]=f_0[n-h]$, deoarece putem prelungi un semnal care se termină cu orizontal jos cu urcarea verticală de lungimea h

- $f_2[n] = f_1[n-1] + f_2[n-1]$, deoarece putem prelungi cu 1 pas orizontal sus orice semnal care se termină cu orizontal sus sau urcare verticală
- $f_3[n] = f_2[n-h]$, deoarece putem prelungi un semnal care se termină cu orizontal sus cu coborâre verticală de lungimea h

Inițializarea trebuie făcută cu atenție: $f_0[0] = 1, f_1[0] = f_2[0] = f_3[0] = 0$

13.9 Cod-sursă pentru problema Semnal

```
#include <iostream>
#define MOD 3000017
using namespace std;

int L,s,t,inv[1000005],fact[1000005],ifact[1000005];
int main(){
    cin>>L;

    inv[1] = 1; fact[0] = 1; fact[1] = 1; ifact[0] = 1; ifact[1] = 1;
    for (int i = 2; i <= 1000000; i++) {
        inv[i]   = 1LL * (MOD - MOD/i) * inv[MOD % i] % MOD;
        fact[i]  = 1LL * fact[i - 1] * i % MOD;
        ifact[i] = 1LL * ifact[i - 1] * inv[i] % MOD;
    }

    s = 1;
    for(int h=1;h<=(L+1)/2-1;h++){
        int t=0;
        for(int k=1;k<=(L+1)/(2*h+2);k++){
            ///comb(L-2*k*h+1,2*k)
            t = (t + (long long)fact[L-2*k*h+1]*ifact[L-2*k*h-2*k+1]%MOD*ifact[2*k]%MOD)%MOD;
        }
        s = (s + t)%MOD;
    }
    cout<<s;
    return 0;
}
```

```
/// Author: proflaurian
/// pentru fiecare h fixat
/// stabilesc un numar par k de segmente verticale de inaltime h
/// asigur conectarea pe orizontala intre oricare doua consecutive
/// partitionez ce a ramas ( daca am >=0 ) in k+1 si
/// distribui pe cele k+1 segmente orizontale delimitate de inceput,starsit si cele k verticale
/// folosesc ideea Partitionari(n,k) = combinari(n+k,k-1)
#include <bits/stdc++.h>

using namespace std;

const int N = 1000010;
const int MOD = 3000017;
inline int prod(int x,int y){x=1LL*x*y%MOD;return x;}
inline int suma(int x,int y){x=(x+y)%MOD;return x;}
inline int expo(int b,int e)
{
    if(e==0)return 1;
    int r=expo(b,e/2);
    r=prod(r,r);
    if(e%2)r=prod(r,b);
    return r;
}
```

```
inline int invMod(int x){return expo(x,MOD-2);}
int L,F[N],I[N],sol=1;

void precalc()
{
    F[0]=1;
    for(int i=1;i<=L;i++)F[i]=prod(F[i-1],i);
    I[L]=invMod(F[L]);
    for(int i=L;i>0;i--)I[i-1]=prod(I[i],i);
}
int comb(int n,int k){return prod(F[n],prod(I[k],I[n-k]));}
int main()
{
    cin >> L;
    precalc();
    for(int h=2;L-h>0;h+=2)
        for(int k=2,R=L-h-1;R>=0;k+=2,R-=h+2)
            sol=suma(sol,comb(R+k,k));
    cout << sol;
    return 0;
}
```

13.10 Problema Circles

Propusă de: stud. Andrei Onuț, Universitatea Yale, S.U.A.

Pe un teren agricol de lângă orașul Austin, din statul Texas, există N traiectorii, numerotate: 1, 2, ..., N, pe care se pot utiliza tractoare. Traiectoriile sunt disjuncte între ele oricare două și au forma unor **cercuri** de raze ce au lungimile valori numere naturale nenule. Sensul deplasării pe fiecare traiectorie este descris prin trei puncte **distincte** (din sistemul de coordonate carteziene) de pe traiectorie, în ordinea în care acestea sunt vizitate de către tractor. Astfel, dacă un tractor urmează traiectoria i, atunci tractorul este pornit din punctul $(x_{i,1}, y_{i,1})$, ajunge în $(x_{i,2}, y_{i,2})$, apoi în $(x_{i,3}, y_{i,3})$ și, în final, se întoarce în $(x_{i,1}, y_{i,1})$, unde tractorul este oprit, efectuându-se **exact o rotație completă pe cerc**.

Fiecare traiectorie i dintre cele N are asociată o valoare $V(i)$ egală cu: $V(i) = r(i) \cdot r(i) \cdot w(i)$, unde $r(i)$ reprezintă lungimea razei cercului ce descrie traiectoria i, iar $w(i) = 1$, dacă deplasarea pe traiectoria i se face în sens antiorar (adică invers acelor de ceas) sau $w(i) = -1$, dacă deplasarea se face în sens orar.

Aflați în vacanță la casele din Austin ale bunicilor, SAM și JOHN vor să ajute la îngrijirea terenului agricol. Astfel, SAM își alege o mulțime **nevidă** $A = \{a_1, a_2, \ldots, a_k\}$ alcătuită din k traiectorii distincte dintre cele N ($1 \leq k < N$), iar lui JOHN îi rămâne mulțimea **nevidă** $B = \{b_1, b_2, \ldots, b_{N-k}\}$ alcătuită din restul de $(N - k)$ traiectorii nealese de SAM.

Bineînțeles, munca lor este răsplătită în funcție de performanță, astfel că după ce fiecare termină de utilizat tractoarele pe traiectoriile alese, SAM primește: $k \cdot (V(a_1) + V(a_2) + \ldots + V(a_k))$ puncte la sala de bowling, iar JOHN primește: $(N - k) \cdot (V(b_1) + V(b_2) + \ldots + V(b_{N-k}))$ puncte. **Este posibil ca punctajul obținut să fie mai mic sau egal decât** 0**.**

Cerință

Pentru a asigura echilibrul între cele două punctaje obținute, bunicii aleg un număr natural L. Determinați în câte moduri distincte își pot împărți cei doi tineri cele două mulțimi de traiectorii A și B, astfel încât **valoarea absolută** a diferenței dintre punctajul obținut de SAM și respectiv punctajul obținut de JOHN să fie mai mică sau egală decât numărul L.

Date de intrare

Pe prima linie se află numărul natural N, cu semnificația de mai sus. Pe fiecare dintre următoarele N linii se află, separate între ele prin câte un spațiu, câte șase numere naturale; astfel, pe a $(i+1)$-a linie se află, în ordine, valorile $x_{i,1}$, $y_{i,1}$, $x_{i,2}$, $y_{i,2}$, respectiv $x_{i,3}$, $y_{i,3}$, pentru fiecare i: $1 \leq i \leq N$. Pe următoarea linie se află numărul natural L.

Date de ieșire

Pe prima linie se află un singur număr întreg, reprezentând numărul de moduri distincte determinat. În cazul în care nu există niciun astfel de mod, atunci valoarea acestui număr va fi egală cu -1.

Restricții

- $2 \leq N \leq 34$ și $0 \leq L \leq 10^{17}$.
- $0 \leq x_{i,j}, y_{i,j} \leq 1\,000\,000$ și $x_{i,j}, y_{i,j}$ sunt numere naturale, pentru fiecare (i, j): $1 \leq i \leq N$ și $1 \leq j \leq 3$.

- $r(i) > 0$ și $|V(i)| \leq 10^{12}$, pentru fiecare i: $1 \leq i \leq N$.
- Cercurile ce descriu fiecare dintre cele N traiectorii au coordonatele (abscisa și ordonata) centrului valori numere naturale mai mici sau egale decât 1 000 000.
- O traiectorie circulară este alcătuită doar din punctele de pe conturul cercului ce descrie traiectoria, iar două traiectorii sunt disjuncte dacă cercurile ce le descriu nu au niciun punct în comun.
- Pentru un cerc C de rază r ($r > 0$), ce are centrul în punctul de coordonate (x_C, y_C), **toate** punctele de pe cerc se află la aceeași distanță față de centru, egală cu lungimea razei. Astfel, $(x - x_C)^2 + (y - y_C)^2 = r^2$, pentru **fiecare** $(x, y) \in C$ (unde x și y sunt numere reale).
- **În cadrul deplasării unui tractor pe traiectoria i, fiecare punct de coordonate reale de pe cercul ce descrie traiectoria este vizitat exact o singură dată, cu excepția punctului $(x_{i,1}, y_{i,1})$, care este vizitat de două ori: o dată la începutul deplasării și o dată la finalul deplasării.**
- *Intrarea la sala de bowling se plătește în puncte, nu în dolari.*

#	Puncte	Restricții
1	4	$N = 2$
2	7	$N = 3$
3	8	$N \leq 15$ și $(x_{i,1}, y_{i,1})$, $(x_{i,2}, y_{i,2})$ și $(x_{i,3}, y_{i,3})$ pot descrie un triunghi dreptunghic, pentru fiecare i
4	34	$N \leq 15$
5	8	Toate cele N traiectorii sunt în sens antiorar
6	39	Nu există alte restricții suplimentare

Exemple

stdin	stdout
3 0 1 1 0 2 1 7 14 13 6 10 5 14 10 12 12 12 8 30	-1
3 0 1 1 0 2 1 7 14 13 6 10 5 14 10 12 12 12 8 43	4

Explicații

Centrele cercurilor ce descriu cele $N = 3$ traiectorii se află în punctele $(1, 1)$, $(10, 10)$, respectiv $(12, 10)$. Cercurile au lungimile razelor egale cu 1, 5, respectiv 2.

Prima traiectorie este descrisă de un cerc cu raza $r(1) = 1$ și este este în sens antiorar, astfel că $w(1) = 1$. Prin urmare, $V(1) = 1$. În mod similar, $V(2) = -25$ și $V(3) = 4$.

Există, **în total**, șase moduri distincte în care SAM și JOHN își pot împărți cele două mulțimi nevide A și B:

- Dacă $A = \{1\}$, iar $B = \{2, 3\}$, SAM primește un punct și JOHN (-42) de puncte. Valoarea absolută a diferenței $1 - (-42)$ este egală cu 43.

- Dacă $A = \{1, 2\}$, iar $B = \{3\}$, SAM primește (-48) de puncte și JOHN 4 puncte. Valoarea absolută a diferenței $(-48) - 4$ este egală cu 52.
- Dacă $A = \{1, 3\}$, iar $B = \{2\}$, SAM primește 10 puncte și JOHN (-25) de puncte. Valoarea absolută a diferenței $10 - (-25)$ este egală cu 35.
- Dacă $A = \{2\}$, iar $B = \{1, 3\}$, SAM primește (-25) de puncte și JOHN 10 puncte. Valoarea absolută a diferenței $(-25) - 10$ este egală cu 35.
- Dacă $A = \{2, 3\}$, iar $B = \{1\}$, SAM primește (-42) de puncte și JOHN un punct. Valoarea absolută a diferenței $(-42) - 1$ este egală cu 43.
- Dacă $A = \{3\}$, iar $B = \{1, 2\}$, SAM primește 4 puncte și JOHN (-48) de puncte. Valoarea absolută a diferenței $4 - (-48)$ este egală cu 52.

Primul exemplu: Nu există niciun mod de a alege mulțimile A și B, astfel încât valoarea absolută a diferenței dintre cele două punctaje să fie mai mică sau egală decât $L = 30$.

Al doilea exemplu: Există patru moduri distincte de a alege mulțimile A și B, astfel încât valoarea absolută a diferenței dintre cele două punctaje să fie mai mică sau egală decât $L = 43$, după cum urmează:

- $A = \{1\}$ și $B = \{2, 3\}$;
- $A = \{1, 3\}$ și $B = \{2\}$;
- $A = \{2\}$ și $B = \{1, 3\}$;
- $A = \{2, 3\}$ și $B = \{1\}$.

13.11 Rezolvarea problemei Circles

Partea de Geometrie

Pentru fiecare traiectorie i: $1 \leq i \leq N$, va fi nevoie să calculăm valoarea $V(i)$. Să introducem, pentru simplitate, următoarele notații în cadrul traiectoriei i:

- $(x_{i,j}, y_{i,j}) = (x_j, y_j)$, pentru $j \in \{1, 2, 3\}$,
- $x_{C,i} =$ [abscisa centrului cercului ce descrie traiectoria i] $= x_C$,
- $y_{C,i} =$ [ordonata centrului cercului ce descrie traiectoria i] $= y_C$,
- $r(i) =$ [lungimea razei cercului ce descrie traiectoria i] $= r$.

Astfel, cu notațiile de mai sus, $V(i) = r^2 \cdot w(i)$. Presupunând că am avea o metodă prin care să determinăm valorile x_C și y_C, am putea calcula și valoarea r apoi.

Scriind de trei ori *ecuația carteziană a cercului*, care este listată și în cadrul secțiunii **Restricții** din enunț, obținem:

$$(x_1 - x_C)^2 + (y_1 - y_C)^2 = r^2, \tag{13.1}$$

$$(x_2 - x_C)^2 + (y_2 - y_C)^2 = r^2, \tag{13.2}$$

$$(x_3 - x_C)^2 + (y_3 - y_C)^2 = r^2. \tag{13.3}$$

Prin urmare, folosind, de exemplu, ecuația (13.1), obținem că $r = \sqrt{(x_1 - x_C)^2 + (y_1 - y_C)^2}$.

Acum, să ne concentrăm pe găsirea valorilor x_C și y_C, folosind, în continuare, cele trei ecuații de mai sus.

În cazul în care $x_1 = x_2$, obținem, în primul rând, că: $y_C = \frac{y_1+y_2}{2}$ (din primele două ecuații). Apoi, folosind prima și cea de a treia ecuație, obținem că: $x_C = \frac{x_1^2+y_1^2-x_3^2-y_3^2-2\cdot y_1\cdot y_C+2\cdot y_3\cdot y_C}{2\cdot(x_1-x_3)}$. De remarcat că $x_1 \neq x_3$, deoarece cele trei puncte ce descriu traiectoria i sunt distincte și deja am afirmat că $x_1 = x_2$. Similar, se tratează cazurile $x_1 = x_3$ sau $x_2 = x_3$. De asemenea, folosind exact același raționament, putem rezolva și dacă $y_1 = y_2$, $y_1 = y_3$ sau $y_2 = y_3$, interschimbând valorile x cu valorile y ale celor trei puncte.

Dacă se întâmplă simultan ca: $x_1 \neq x_2$, $x_1 \neq x_3$, $x_2 \neq x_3$, $y_1 \neq y_2$, $y_1 \neq y_3$ și $y_2 \neq y_3$, putem rezolva următorul sistem de două ecuații liniare pentru a afla cele două necunoscute x_C și y_C:

$$a \cdot x_C + b \cdot y_C = T,$$
$$c \cdot x_C + d \cdot y_C = Q,$$

unde:

- $T = x_1^2 + y_1^2 - x_2^2 - y_2^2$,
- $a = 2 \cdot (x_1 - x_2) \neq 0$,
- $b = 2 \cdot (y_1 - y_2) \neq 0$,
- $Q = x_1^2 + y_1^2 - x_3^2 - y_3^2$,
- $c = 2 \cdot (x_1 - x_3) \neq 0$,
- $d = 2 \cdot (y_1 - y_3) \neq 0$.

Expresiile T, a și b pot fi găsite scăzând primele două *ecuații ale cercului* una din cealaltă, iar expresiile Q, c și d pot fi găsite scăzând prima și cea de a treia *ecuație* una din cealaltă.

Pentru a rezolva sistemul de mai sus, putem folosi, de exemplu, *metoda substituției*: se află y_C din prima ecuație (în funcție de x_C) și apoi se înlocuiește această expresie în cea de a doua ecuație. Astfel, se află valoarea lui x_C. Apoi, se află și valoarea lui y_C.

De asemenea, pentru a determina $w(i)$ se poate folosi fie calcul de determinanți (ca și în cazul calculării, de exemplu, a ariei unui triunghi), fie se compară valorile pantelor celor două drepte suport ale segmentelor determinate de punctele (x_1, y_1) și (x_2, y_2), respectiv de punctele (x_2, y_2) și (x_3, y_3).

Meet-in-the-middle

Pentru a afla răspunsul la problemă, vom utiliza tehnica *Meet-in-the-middle*, despre care puteți afla mai multe pe USACO Guide.

Conform restricțiilor din enunț, știm că $A \cup B = \{1, 2, \ldots, N\}$ și $A \cap B = \emptyset$. Astfel, fiecare modalitate (A, B) prin care SAM și JOHN își pot împărți între ei cele N traiectorii, respectând restricția bunicilor cu privire la numărul L, poate fi unic identificată folosind doar mulțimea A.

Să introducem: $M = \lfloor \frac{N}{2} \rfloor \geq 1$, unde $\lfloor \frac{N}{2} \rfloor$ reprezintă valoarea părții întregi a împărțirii numărului N la 2. Considerăm cele două *jumătăți* nevide ale șirului V: $l = [V_1, V_2, \ldots, V_M]$ (prima *jumătate*) și $r = [V_{M+1}, V_{M+2}, \ldots, V_N]$ (a doua *jumătate*). Astfel, $1 \leq |l| \leq |r| = \lfloor \frac{N+1}{2} \rfloor$.

Evident, mulțimea A are cel puțin un element, iar elementele ce constituie A pot face parte fie doar din mulțimea $\{1, 2, \ldots, M\}$, fie doar din mulțimea $\{M+1, M+2, \ldots, N\}$, fie din ambele mulțimi. Cu acest gând, atât pentru l, cât și pentru r, vom calcula în complexitatea de timp: $O(2^{\lfloor \frac{N+1}{2} \rfloor} \cdot \lfloor \frac{N+1}{2} \rfloor)$, utilizând, de exemplu, *reprezentarea în baza 2 a numerelor naturale*, toate perechile de forma $(sum(\sigma), |\sigma|)$, unde: σ este un subșir (format nu neapărat din elemente consecutive) al șirului l (sau al șirului r, pentru calculele ce implică elementele din a doua *jumătate* a șirului V),

iar $sum(\sigma)$ = [suma elementelor din σ]. Trebuie să fim atenți la faptul că σ poate fi și vid, caz în care considerăm că $sum(\sigma) = |\sigma| = 0$.

Pentru fiecare subșir σ al șirului r, vom *insera* pe linia $|\sigma|$ a unei *colecții* (mai formal, o listă) bidimensionale valoarea $sum(\sigma)$. După această etapă, elementele fiecărei linii i (cu $0 \leq i \leq |r| = \lfloor \frac{N+1}{2} \rfloor$) din *colecție* vor trebui să ajungă în ordine non-descrescătoare. Linia i conține $\binom{|r|}{i} = \binom{\lfloor \frac{N+1}{2} \rfloor}{i}$ elemente. Astfel, [numărul maxim de elemente din cadrul unei linii a *colecției*] $= \binom{|r|}{\lfloor \frac{|r|}{2} \rfloor}$ $= max_n$ (notație), din Triunghiul lui Pascal. Cele $2^{\lfloor \frac{N+1}{2} \rfloor}$ elemente ale colecției pot fi sortate în complexitatea de timp: $O(2^{\lfloor \frac{N+1}{2} \rfloor} \cdot \log(2^{\lfloor \frac{N+1}{2} \rfloor})) = O(2^{\lfloor \frac{N+1}{2} \rfloor} \cdot \lfloor \frac{N+1}{2} \rfloor)$, înainte de a *insera* pe linia corespunzătoare fiecare element.

În continuare, să presupunem că am fixat cu ajutorul unui subșir σ (notații: $sum(\sigma) = sum_a$, $|\sigma| = cnt_a$) al șirului l valorile traiectoriilor din mulțimea $\{1, 2, \ldots, M\}$ ce fac parte dintr-o posibilă mulțime *soluție* A, pe care încercăm să o numărăm. Considerând că S este suma: $S = V_1 + V_2 + \ldots + V_N$, atunci dorim să numărăm câte subșiruri σ' (notații: $sum(\sigma') = sum_b$, $|\sigma'| = cnt_b$) ale șirului r respectă proprietatea:

$$|(sum_a + sum_b) \cdot (cnt_a + cnt_b) - (S - (sum_a + sum_b)) \cdot (N - (cnt_a + cnt_b))| \leq L.$$

Prelucrând inegalitatea de mai sus, obținem că:

$$-L - S \cdot (v - N) - sum_a \cdot N \leq sum_b \cdot N \leq L - S \cdot (v - N) - sum_a \cdot N$$

unde: $v = (cnt_a + cnt_b)$ și v poate fi fixat: $cnt_a \leq v < N$ și $1 \leq v$ (mulțimea A este nevidă).

Astfel, pentru a număra câte subșiruri σ' satisfac inegalitatea de mai sus, se va *căuta binar* pe linia $(v - cnt_a)$ în *colecție*, în complexitatea de timp: $O(\log max_n)$, pentru fiecare pereche (σ, v).

Complexitatea totală de timp a algoritmului este: $O(2^{\lfloor \frac{N+1}{2} \rfloor} \cdot \lfloor \frac{N+1}{2} \rfloor + 2^{\lfloor \frac{N+1}{2} \rfloor} \cdot N \cdot \log max_n)$, adică: $O(2^{\lfloor \frac{N+1}{2} \rfloor} \cdot N \cdot \log max_n)$. În cazul în care $N = 34$, înseamnă că $max_n = \binom{17}{8} = 24\,310$.

Soluție alternativă - viitor stud. Cristian Verde

Problema presupune aflarea razei cercului circumscris celor 3 puncte, pe care o putem calcula găsind coordonatele centrului acestuia. Centrul este localizat la intersecția mediatoarelor laturilor triunghiului format de punctele (x_1, y_1), (x_2, y_2) și (x_3, y_3). Știm că mediatoarea unei laturi este perpendiculară pe aceasta, de unde aflăm panta (dacă $d1$ și $d2$ sunt drepte perpendiculare, atunci fie sunt una paralelă cu Ox și cealaltă cu Oy, fie respectă relația $a_{d1} \cdot a_{d2} = 1$, unde a_{d1} și a_{d2} sunt pantele celor 2 drepte), și faptul că trece prin mijlocul laturii, lucru ce ne permite să calculăm b-ul ecuației $y = a \cdot x + b$ pentru mediatoare, iar acum problema se reduce la o intersecție de drepte și distanța dintre centru și unul dintre puncte. Calculez $w(i)$ tot cu determinanți, mai exact luând semnul expresiei $(x_3 - x_2) \cdot (y_1 - y_2) - (x_1 - x_2) \cdot (y_3 - y_2)$, ce ne oferă semiplanul față de dreapta reprezentată de punctele (x_1, y_1) și (x_3, y_3) în care se află punctul (x_2, y_2).

Partea a doua

Având valorile vectorului V, ne mai rămâne de calculat numărul de moduri de a partiționa elementele sale în două submulțimi ce respectă relația din enunț. Rescriind, obținem:

$$|(V(a_1) + V(a_2) + \ldots + V(a_k) + V(b_1) + V(b_2) + \ldots + V(b_{N-k})) \cdot k - (V(b_1) + V(b_2) + \ldots + V(b_{N-k})) \cdot N| <= L$$

sau

$$|sum_V \cdot k - (V(b_1) + V(b_2) + \ldots + V(b_{N-k})) \cdot N| <= L$$

Cu alte cuvinte, vrem:

$$sum_V \cdot k + L >= sum_B \cdot N >= sum_V \cdot k - L, \tag{13.4}$$

unde B este o submulțime de $N-k$ elemente a lui V.

Soluție de complexitate $O(2^N)$ - 53 de puncte

Fixăm, utilizând o mască pe biți, toate submulțimile posibile (scrierea fiecărui număr în baza 2 poate reprezenta o submulțime - dacă cifra de pe poziția i este 1 atunci elementul $V(i+1)$ se regăsește în ea). Așadar, printr-un for de la 1 la $2^N - 2$, avem

$$sum_i = sum_{i\ xor\ 2^{first\_bit(i)}} + V(first\_bit(i) + 1)$$

unde $first\_bit(i)$ este `31 - __builtin_clz(i)`, iar numărul de cifre de 1 din scrierea binară al lui i (pentru aflarea lui $N-k$) este `__builtin_popcount(i)`. Mai rămâne doar să contorizăm câte dintre acestea satisfac condiția de mai sus pentru o mulțime B.

Soluție de complexitate $O(2^{\frac{N}{2}} \cdot N)$ - 100 de puncte

Întocmai ca Andrei, folosesc tehnica „Meet-in-the-middle". Îmi împart elementele șirului în primele $[\frac{N}{2}]$ și ultimele $[\frac{N+1}{2}]$, pe care folosesc abordarea din soluția de complexitate $O(2^N)$, introduc valorile din *sum* în vectori STL în funcție de numărul de elemente ale submulțimii și îi sortez. Pentru fiecare i de la 1 la $N-1$, trebuie să calculăm în câte feluri putem combina o submulțime cu x elemente din primele $[\frac{N}{2}]$ și $i-x$ din ultimele $[\frac{N+1}{2}]$, astfel încât să respecte condiția (13.4) ($i = N-k$, iar $\frac{N}{2} >= x >= 0$ și $\frac{N+1}{2} >= i-x >= 0$). Având sumele deja sortate, putem aplica metoda Two Pointers, complexitatea acestei bucăți a algoritmului, la fel ca cea a sortării, fiind $O(2^{\frac{N}{2}} \cdot N)$.

13.12 Cod-sursă pentru problema Circles

```
#include <bits/stdc++.h>
using namespace std;
typedef int64_t Int;
vector<Int> va[50],vb[50];
Int sol,T,R,L;
int n;
Int det(Int a11,Int a12,Int a21,Int a22)
{
    return a11*a22-a12*a21;
}
Int semn(Int X)
{
    if(X>=0)return 1LL;
    return -1LL;
}
void sist(Int a11,Int a12,Int b1,Int a21,Int a22,Int b2,Int &X,Int &Y)
{
```

```
    Int d=det(a11,a12,a21,a22),dx=det(b1,a12,b2,a22),dy=det(a11,b1,a21,b2);
    X=dx/d;
    Y=dy/d;
}
Int elem()
{
    Int xa,ya,xb,yb,xc,yc;
    cin>>xa>>ya>>xb>>yb>>xc>>yc;xa-=xc;ya-=yc;xb-=xc;yb-=yc;
    sist(2LL*xa,2LL*ya,xa*xa+ya*ya,2LL*xb,2LL*yb,xb*xb+yb*yb,xc,yc);
    return semn(det(xa,ya,xb,yb))*(xc*xc+yc*yc);
}
void updSol(Int Lo,Int Hi,vector<Int> &A,vector<Int> &B)
{
    int a=A.size(),b=B.size(),c=0,s=0,d=0;
    for(c=0;c<a&&s<b;c++){for(;s<b&&A[c]+B[s]<Lo;s++);for(;d<b&&A[c]+B[d]<=Hi;d++);sol+=d-s;}
}
int main()
{
    ios_base ::sync_with_stdio(false);
    cin.tie(NULL);
    cin>>n;
    va[0].push_back(0);
    for(int i=1; i<=n/2;i++){
      R=elem();T+=R;for(int j=i;j>=1;j--)for(auto it:va[j-1])va[j].push_back(it+R*n);
    }
    vb[0].push_back(0);
    for(int i=1; i<=n-n/2;i++){
      R=elem();T+=R;for(int j=i;j>=1;j--)for(auto it:vb[j-1])vb[j].push_back(it+R*n);
    }
    for(int i=1;i<=n/2;i++)sort(va[i].rbegin(),va[i].rend());
    for(int i=1;i<=n-n/2;i++)sort(vb[i].begin(),vb[i].end());
    cin>>L;
    for(int k=1; k<n; k++)for(int i=0,j=k; j>=0; i++,j--)
                            if(i<=n/2&&j<=n-n/2)updSol(-L+T*(n-k),L+T*(n-k),va[i],vb[j]);
    if(sol==0)sol=-1;
    cout<<sol<<'\n';
    return 0;
}
```