Olimpiada Societății pentru Excelență și Performanță în Informatică

ETAPA JUDEȚEANĂ, 2021

SUBIECTE

Enunțuri și rezolvări

2021 - Editura L&S Soft | Infobits Academy
https://ebooks.infobits.ro

ISBN: 978-606-95233-0-8

M A T E R I A L U L E S T E D I S T R I B U I T G R A T U I T .
S P O R L A S T U D I U !

Editura L&S SOFT

Adresa: Aleea Aviației nr. 10, Voluntari, Ilfov
RO14864546, J23/1969/2019
Mobil: 0727.731.947
E-mail: hello@infobits.ro

www.manuale-de-informatica.ro

Infobits Academy
www.infobits.ro

Cărți în format electronic cu specific informatic
https://ebooks.infobits.ro

Autori

Clasa a V-a

- ☺ prof. Florentina Ungureanu - Colegiul Național de Informatică Piatra-Neamț (coordonator)
- ☺ prof. Georgeta Iulia Balacea - Colegiul Național "Vasile Alecsandri"
- ☺ prof. Dan Octavian Dumitrașcu - Colegiul Național Dinicu Golescu Câmpulung-Muscel
- ☺ prof. Violeta - Marilena Grecea - Colegiul Național de Informatică "Matei Basarab" Râmnicu-Vâlcea
- ☺ prof. Marius Nicoli - Colegiul National "Frații Buzești" Craiova
- ☺ prof. Adriana Simulescu - Liceul Teoretic "Grigore Moisil" Timișoara
- ☺ olimpic internațional (2003 - 2006) Dan-Constantin Spătărel

Clasa a VI-a

- ☺ prof. Daniela Lica, Centrul Județean de Excelență Prahova, Ploiești (coordonator)
- ☺ prof. Cristina Anton, Colegiul Național "Gheorghe Munteanu Murgoci", Brăila
- ☺ prof. Ana-Maria Arișanu, Colegiul Național "Mircea cel Bătrân", Rm. Vâlcea
- ☺ stud. Stelian Chichirim, Universitatea București
- ☺ prof. Cristina Iordaiche, Liceul Teoretic "Grigore Moisil", Timișoara
- ☺ stud. masterand Radu Muntean, Master CS - Zürich
- ☺ prof. Marinel Șerban, Colegiul Național "Emil Racoviță", Iași
- ☺ prof. Roxana Tîmplaru, Liceul Tehnologic "Ștefan Odobleja"/ ISJ Dolj

Clasa a VII-a

- ☺ prof. Emanuela Cerchez - Colegiul Naţional "Emil Racoviţă", Iaşi (coordonator)
- ☺ prof. Nistor Moţ - Şcoala "Dr. Luca", Brăila
- ☺ prof. Marius Nicoli - Colegiul Național "Frații Buzești", Craiova
- ☺ prof. Adrian Pintea - Colegiul Național "Andrei Mureşanu", Dej
- ☺ prof. Ionel-Vasile Piţ-Rada - Colegiul Naţional "Traian", Drobeta-Turnu Severin
- ☺ stud. Theodor-Gabriel Tulbă-Lecu - Universitatea Politehnica București

Clasa a VIII-a

- ☺ prof. Carmen Mincă - Colegiul Național de Informatică “Tudor Vianu”, București (coordonator)
- ☺ ș.l. Stelian Ciurea - Universitatea ”Lucian Blaga” din Sibiu
- ☺ prof. Flavius Dumitru Boian - Colegiul Național “Spiru Haret” , Târgu-Jiu
- ☺ prof. Veronica Raluca Costineanu - Colegiul Național “Ștefan cel Mare”, Suceava
- ☺ drd. Diana Ghinea -Eidgenössische Technische Hochschule (ETH) Zürich
- ☺ stud.masterand Bogdan Ioan Iordache - Universitatea din București
- ☺ prof. Mirela Mlisan - Colegiul Național “Mircea cel Bătrân”, Râmnicu Vâlcea
- ☺ prof. Mircea Dorin Rotar - Colegiul Național “Samuil Vulcan”, Beiuș
- ☺ stud. Ioan Cristian Pop - Universitatea Politehnică București

Clasa a IX-a

- ☺ prof. Zoltan Szabo, Liceul Tehnologic ”Petru Maior” Reghin / ISJ Mureș - Tg.Mureș (coordonator)
- ☺ stud. Andrei Arhire, Universitatea ”Alexandru Ioan Cuza” din Iași
- ☺ prof. Alin Burța, Colegiul Național ”B.P. Hasdeu”, Buzău
- ☺ stud. Mihaela Cișmaru, Universitatea Politehnica București
- ☺ software engineer Vlad Gavrilă, Google, Zürich
- ☺ stud. Ștefan Manolache, University of Oxford, Lady Margaret Hall College
- ☺ prof. Adrian Panaete, Colegiul Național ”A.T.Laurian”, Botoșani
- ☺ stud. Alexandru Petrescu, University of Oxford, Keble College
- ☺ stud. Bogdan-Ioan Popa, Universitatea din București
- ☺ stud. Cezar Trișcă-Vicol, University of Oxford, Christ Church College.
- ☺ stud. Theodor-Gabriel Tulbă-Lecu, Universitatea Politehnica București

Clasa a X-a

- ☺ prof. Marius Nicoli, Colegiul Național ”Frații Buzești” Craiova (coordonator)
- ☺ conf.univ.dr. Doru Anastasiu Popescu, Universitatea din Pitești
- ☺ drd. Lucian Bicsi, Universitatea din București
- ☺ prof. Mihai Bunget, Colegiul Național ”Tudor Vladimirescu” Târgu Jiu,
- ☺ stud. Ștefan-Cosmin Dăscălescu, Universitatea din București
- ☺ prof. Gheorghe Manolache, Colegiul Național de Informatică Piatra Neamț

☺ stud. masterand Tamio-Vesa Nakajima, Universitatea Oxford

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

☺ stud. Costin-Andrei Oncescu, Universitatea Oxford

☺ stud. Bogdan-Petru Pop, Universitatea "Babeș-Bolyai" Cluj Napoca

☺ stud. Mihai-Cristian Popescu, Universitatea Babeș-Bolyai Cluj Napoca

Clasa a XI-XII

☺ prof. Adrian Panaete, Colegiul "A. T. Laurian", Botoșani (coordonator)

☺ prof. Zoltan Szabo, Liceul Tehnologic "Petru Maior" Reghin / ISJ Mures, Tg. Mureș,

☺ stud. masterand George Chichirim, University of Oxford, Keble College

☺ stud. Stelian Chichirim, Universitatea din București

☺ Bogdan Ciobanu, software engineer, Hudson River Trading

☺ stud. masterand Andrei Constantinescu, University of Oxford, Balliol College

☺ stud.Adrian Emanuel Dicu, Universitatea Politehnică din București

☺ stud. Theodor Pierre Moroianu, Universitatea din București

☺ stud. Costin-Andrei Oncescu, University of Oxford, St. John's College

☺ stud. Bogdan Sitaru, University of Oxford, Hertford College

☺ stud. Alexa Maria Tudose, University of Oxford, University College

☺ stud. Alexandra Maria Udristoiu, Universitatea din București

Cuprins

Capitolul 1

Clasa a V-a

1.1 Problema Ghiocel

Propunător: prof. Georgeta Iulia Balacea
Colegiul Național "Vasile Alecsandri" Galați

Sesiunea de antrenament

Enunț

Ghiocel, nepotul cel mai mic al Babei Dochia, este elev în clasa a V-a și dorește să ofere câte un buchet de ghiocei fiecăreia dintre cele n doamne profesoare. Primul buchet va avea b_1 ghiocei, cel de-al doilea buchet va avea b_2 ghiocei, cel de-al treilea buchet va avea b_3 ghiocei, și așa mai departe, iar ultimul buchet, cel de al n-lea, va avea b_n ghiocei.

Copilul dorește să afle câte dintre numerele b_1, b_2, b_3, ... , b_n au aspect de *urcuș*, *coborâș*, *deal* sau *vale*.

Spunem că un număr natural nenul are aspect de *urcuș* dacă cifrele sale apar în ordine strict crescătoare, de la stânga la dreapta. De exemplu, numărul 2346 are aspect de urcuș.

Spunem că un număr natural nenul are aspect de *coborâș* dacă cifrele sale apar în ordine strict descrescătoare, de la stânga la dreapta. De exemplu, numărul 520 are aspect de *coborâș*.

Spunem că un număr natural nenul are aspect de *deal* dacă cifrele sale se succed întâi în ordine strict crescătoare, apoi în ordine strict descrescătoare. De exemplu, numărul 135421 are aspect de *deal*.

Spunem că un număr natural nenul are aspect de *vale* dacă cifrele sale se succed întâi în ordine strict descrescătoare, apoi în ordine strict crescătoare. De exemplu, numărul 210345 are aspect de *vale*.

Cerință

Cunoscând numărul n de buchete, iar pentru fiecare buchet i ($1 \leq i \leq n$), numărul b_i de ghiocei, determinați câte dintre numerele b_i au aspect de *urcuș*, câte au aspect de *coborâș*, câte au aspect de *deal* și câte au aspect de *vale*.

Date de intrare

Prima linie a fișierului *ghiocel.in* conține un număr natural n, care reprezintă numărul de buchete de ghiocei. Fiecare următoarele n linii ale fișierului conține câte un număr natural nenul b_i, reprezentând numărul de flori din buchetul b_i.

Date de ieșire

Prima linie a fișierului *ghiocel.out* va conține o valoare reprezentând numărul de numere b_i cu aspect de *urcuș*, cea de a doua linie va conține o valoare reprezentând numărul de numere b_i cu aspect de *coborâș*, cea de a treia linie va conține o valoare reprezentând numărul de numere b_i cu aspect de *deal*, iar cea de a patra linie va conține o valoare reprezentând numărul de numere b_i cu aspect de *vale*.

Restricții și precizări

- $1 \leq n \leq 2\ 000$.
- Fiecare număr b_i, $(1 \leq i \leq n)$ are cel puțin două cifre și cel mult 18 cifre.
- Pentru 30 de puncte numerele b_i, $(1 \leq i \leq n)$ au cel mult 4 cifre fiecare.
- Pentru 30 de puncte numărul de n este mai mic sau egal decât 1 000.

Exemplu

ghiocel.in	***ghiocel.out***	***Explicații***
8 121 243 1134 21323 2 12 323	1 1 2 1	$n = 8$ (Sunt 8 buchete de ghiocei). Numărul 12 are aspect de *urcuș*. Numărul 21 are aspect de *coborâș*. Numerele 2343 și 121 au aspect de *deal*. Numărul 323 are aspect de *vale*.

Descrierea soluției

Pentru rezolvarea problemei se utilizează date simple, întregi cu cel mult 18 cifre. În rezolvarea problemei se utilizează elemente de matematică din programa școlară a clasei a V-a. (Operații cu numere naturale, scrierea în baza 10 a unui număr natural) și structuri elementare de control (structura alternativă și structura repetitivă).

Pentru a stabili aspectul numărului b_i se determină relația dintre ultimele două cifre ale sale. Cât timp relația se păstrează eliminăm câte o cifră din număr.

Dacă s-au eliminat toate cifrele se incrementează corespunzător, în funcție de inegalitate, contorul de numere de tip *coborâș*, respectiv, contorul de numere de tip *urcuș*.

Dacă mai există cifre în număr, se continuă eliminarea câte unei cifre, cât timp este respectată noua relație dintre cifre. Dacă s-au epuizat toate cifrele numărului se incrementează corespunzător contorul de numere de tip *deal*, respectiv, contorul de numere de tip *vale*.

Dacă după cele două etape mai există cifre în număr, atunci acesta nu are niciunul dintre cele 4 aspecte.

1.2 Problema Concurs

Propunător: prof. Violeta-Marilena Grecea
Colegiul Național de Informatică "Matei Basarab" Râmnicu-Vâlcea

Enunț

În orașul X va avea loc o nouă ediție a concursului Y, la care participă 3 echipe având numerele de concurs 1, 2 și 3. Echipele pot avea număr diferit de concurenți.

Ordinea în care participanții intră în concurs este una oarecare. Fiecare concurent are de susținut 9 probe. La fiecare probă, un concurent obține un punctaj exprimat printr-un număr natural, cuprins între 0 și 10, inclusiv.

La scurt timp după ce un concurent a susținut toate cele 9 probe, se afișează performanța concurentului sub forma a două numere naturale, astfel:

- primul număr poate fi 1, 2 sau 3 și reprezintă echipa din care face parte concurentul;
- al doilea număr este obținut prin concatenarea (alipirea) numerelor ce reprezintă punctajele **nenule** obținute de concurent la cele 9 probe. Dacă un concurent are punctaj 0 la toate probele atunci al doilea număr este 0.

Punctajul total al unui concurent se obține adunând punctajele obținute de acesta la cele 9 probe. Punctajul unei echipe se obține adunând punctajele totale obținute de membrii acesteia.

De exemplu, afișajul 2 14102172, semnifică faptul că acest concurent face parte din echipa 2 și are punctajele nenule 1, 4, 10, 2, 1, 7 și 2, la 7 dintre cele 9 probe susținute. La celelalte două probe a avut punctajul 0. Punctajul său total este 27, contribuția sa la punctajul echipei 2 fiind de 27 de puncte.

Este declarată campioană echipa cu punctajul cel mai mare. Dacă mai multe echipe au obținut cel mai mare punctaj, atunci toate aceste echipe sunt declarate campioane. Totuși, dacă toate echipele au totalizat 0 puncte, atunci nicio echipă nu este declarată campioană.

Cerințe

Cunoscând numărul N de concurenți, echipele din care fac parte precum și punctajele obținute de fiecare dintre ei, să se determine:

1. punctajul maxim obținut de un concurent și numărul de concurenți care au obținut acest punctaj;
2. numărul sau numerele de concurs ale echipelor declarate campioane, în ordine crescătoare, și punctajul obținut de acestea. Dacă toate echipele au punctajul final 0, se va afișa textul **FARA CAMPION**

Date de intrare

Fișierul de intrare *concurs.in* conține pe prima linie un număr C (care poate fi 1 sau 2), indicând cerința de rezolvat. Pe a doua linie se găsește un număr natural N reprezentând numărul de concurenți, iar pe fiecare dintre următoarele N linii se găsesc câte două numere naturale, separate printr-un spațiu, reprezentând echipa și punctajele fiecăruia dintre cei N concurenți, în ordinea intrării în concurs.

Date de ieșire

1. Dacă $C = 1$, fișierul de ieșire *concurs.out* va conține pe o singură linie, două numere naturale, separate printr-un spațiu, reprezentând punctajul maxim obținut de un concurent și numărul de concurenți care au obținut acest punctaj.

2. Dacă $C = 2$, fișierul de ieșire va conține pe o singură linie textul **FARA CAMPION** dacă toate echipele au la final punctajul 0. În caz contrar linia va conține două, trei sau patru numere naturale separate prin câte un spațiu, reprezentând numărul sau numerele de concurs ale echipelor declarate campioane, în ordine crescătoare, și apoi punctajul obținut de acestea.

Restricții și precizări

- Se garantează că datele din fişier respectă formatul precizat.
- $1 \leq N \leq 100\ 000$.
- Pentru teste în valoare de 35 de puncte avem $C = 1$.
- Pentru teste în valoare de 65 de puncte avem $C = 2$.

Exemplu

	concurs.in	*concurs.out*	*Explicații*
1	1 7 1 1111973 2 3101971 1 1999 2 1010101 3 1010101 3 0 31371910	31 4	În primul exemplu punctajele obținute de concurenți sunt: 23, 31, 28, 31, 31, 0, 31 deci punctajul maxim este 31 și sunt 4 concurenți cu acest punctaj.
2	2 5 1 1111973 2 3101971 1 1999 3 1010101 3 1371910	3 62	În al doilea exemplu sunt 5 concurenți, primul concurent este din echipa 1 și are punctajul 23, cel de-al doilea concurent este din echipa 2 și are punctajul 31, cel de-al treilea este din echipa 1 și are punctajul 28, al patrulea este din echipa 3 și are 31 de puncte, iar al cincilea este din echipa 3 și are 31 de puncte. Punctajul total al echipei 1 este 51, punctajul total al echipei 2 este 31, punctajul total al echipei 3 este 62. Deci, va câștiga echipa 3 cu 62 de puncte.
3	2 3 2 1111973 3 31019 1 1010111	1 2 3 23	În al treilea exemplu sunt 3 concurenți, primul concurent este din echipa 2 și are punctajul 23, cel de-al doilea concurent este din echipa 3 și are punctajul 23, cel de-al treilea este din echipa 1 și are punctajul 23. Toate cele trei echipe au punctaj maxim 23.

Descrierea soluției

Pentru rezolvarea problemei se utilizează date simple, întregi cu cel mult 18 cifre, algoritmi cu cifrele unui număr, calculul maximului. Atât pentru rezolvarea cerinței 1 cât și pentru rezolvarea cerinței 2 trebuie calculat punctajul total al unui concurent, ținând cont de modalitatea de afișare a punctajelor acestuia la cele 9 probe susținute: pentru fiecare probă punctajul este cuprins între 0 și 10 inclusiv dar, se afișează doar punctaje nenule (Se afișează 0 doar dacă toate cele 9 punctaje au fost nule.) De aceea, dacă valoarea afișată după susținerea celor 9 probe este nenulă, stabilirea punctajul total al concurentului se reduce la a calcula suma cifrelor numărului ce reprezintă performanța sa, cu observat, ia că, dacă cifra prelucrată la un moment dat este 0, se va asocia cu valoarea 1 din fața sa, și la punctajul total se adună 10 puncte.

Cerința 1 presupune calculul maximului dintr-un șir și de câte ori apare acesta.

Cerința 2 presupune calculul maximului dintre trei valori, valori asociate punctajelor totale ale celor 3 echipe și afișarea etichetei echipelor care au punctajul egal cu cel maxim.

1.3 Problema Șir

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

Enunț

Se dă un șir format din n numere naturale nenule. Elementele șirului sunt numerotate de la stânga la dreapta începând cu poziția 1.

Cerință

Scrieți un program care să determine răspunsul pentru întrebări de următoarele tipuri:

1. Care este cea mai din stânga poziție care conține o valoare strict mai mare decât toate cele din dreapta sa? - întrebare de tipul 1
2. Care sunt pozițiile care conțin valori strict mai mari decât toate cele din stânga lor? - întrebare de tipul 2
3. Dacă fiecărui element aflat între prima și ultima apariție a maximului i-am mări valoarea pentru a ajunge egal cu maximul, care este suma totală a valorilor adăugate? - întrebare de tipul 3

Date de intrare

Fișierul de intrare *sir.in* conține pe prima linie un număr C (care poate fi 1, 2 sau 3), indicând tipul întrebării. Pe linia a doua se află un număr natural N, reprezentând numărul de elemente din șir. Pe a treia linie a fișierului de intrare se află N numere naturale, reprezentând elementele șirului, date de la stânga la dreapta (cel mai din stânga are poziția 1 și cel mai din dreapta are poziția N). Numerele de pe această linie sunt separate prin câte un spațiu.

Date de ieșire

Dacă $C = 1$, fișierul de ieșire *sir.out* trebuie să conțină un număr natural ce reprezintă răspunsul la o întrebare de tipul 1.

Dacă $C = 2$, fișierul de ieșire trebuie să conțină, separați prin câte un spațiu și în ordine crescătoare, indicii determinați ca răspuns la o întrebare de tipul 2.

Dacă $C = 3$, fișierul de ieșire trebuie să conțină un număr ce reprezintă răspunsul la o întrebare de tipul 3.

Restricții și precizări

- $1 \leq C \leq 3$.
- $1 \leq N \leq 100\ 000$.
- Numerele din șirul dat sunt cuprinse între 1 și 10 000, inclusiv.

- Pentru teste în valoare de 24 de puncte avem $C = 1$.
- Pentru teste în valoare de 32 de puncte avem $C = 2$.
- Pentru teste în valoare de 44 de puncte avem $C = 3$.

Exemple

	sir.in	***sir.out***	***Explicații***
1)	1 7 3 2 2 5 3 5 4	6	Cea mai din stânga poziție a unei valori care este mai mare decât toate cele din dreapta sa este 6 (acolo unde se află valoarea 5)
2)	2 7 3 2 2 5 3 5 4	1 4	Pentru exemplul 2, 1 și 4 sunt pozițiile unde se află valori mai mari decât toate cele din stânga lor.
3)	3 8 3 2 2 5 3 1 5 4	6	Pentru exemplul 3, maximul fiind 5, conform explicației de la întrebarea de tipul 3, trebuie mărite două elemente pentru a ajunge egale cu 5. Acestea sunt cel aflat pe poziția 5 (de mărit cu 2) precum și cel de pe poziția 6 (de mărit cu 4). Suma valorilor cu care avem de mărit este 2 + 4 = 6.
4)	3 5 3 2 7 5 3	0	Pentru exemplul 4, maximul este 7 și apare o singură dată, deci nu se mai mărește nicio valoare.

Descrierea soluției

Pentru rezolvarea cerinței 1 este necesară identificarea celei mai din dreapta poziții pe care se află valoarea maximă. Acest lucru se realizează la citirea datelor, păstrând o variabilă pe care o actualizăm cu indicele curent, în momentul schimbării maximului sau în momentul găsirii unei valori egale cu maximul.

Pentru rezolvarea cerinței 2 trebuie identificate momentele în care maximul se schimbă. Deci, la citirea datelor, atunci când valoarea curentă este strict mai mare decât maximul deja întâlnit, actualizăm maximul și afișăm poziția curentă.

Pentru rezolvarea cerinței 3 păstrăm o variabilă S pe care o gestionăm astfel:

- Când se schimbă maximul reinițializăm S cu 0.
- Dacă valoarea curentă x nu schimbă maximul, adunăm la S diferența $maxim - x$.
- De fiecare dată când întâlnim o valoare egală cu maximul, salvăm valoarea curentă a lui S (înainte să o reinițializăm) într-o variabilă *sol*, pe care la final o afișăm.

Mai sus am descris o strategie de a procesa informațiile dintr-o secvență cuprinsă între primul și ultimul element cu valoarea maximă, realizând calculele în timpul citirii datelor.

Capitolul 2

Clasa a VI-a

2.1 Problema Ștergeri

Propunător: prof. Dana Lica
Colegiul Național "I.L.Caragiale", Ploiești

Sesiunea de antrenament

Enunț

Se dă un șir V cu N elemente, numere naturale nenule. La primul pas secvența de elemente cuprinse între prima și ultima valoare cu suma cifrelor maximă (inclusiv acestea) se elimină. Ne imaginăm că toate elementele din dreapta secvenței eliminate sunt deplasate la stânga, ajungând în continuarea celor din stânga secvenței eliminate. Reluăm pe noul șir operația descrisă anterior. Procedeul se repetă până se ajunge la un șir fără elemente.

Cerințe
Determinați numărul de pași necesari pentru a ajunge la șirul vid. Determinați și numărul cel mai mare de elemente care s-au eliminat la același pas.

Date de intrare
Prima linie a fișierului de intrare conține numărul N. A doua linie conține cele N elemente ale șirului, separate prin câte un spațiu.

Date de ieșire
Se vor afișa două valori separate prin spațiu: numărul de pași necesari până când șirul devine vid respectiv numărul maxim de elemente eliminate la același pas.

Restricții

- $1 \leq N \leq 100\ 000$
- $1 \leq v[i] \leq 1.000.000.000$

Exemplu

	stergeri.in	stergeri.out
	6 6 111 19 27 4 55 1	3 4

Explicații

La primul pas se elimină secvența 19 27 4 55 iar șirul rămas: 111 1. La al doilea pas se elimină 111 (este un singur element cu suma cifrelor maximă), șirul rămas fiind format doar din 1. La al treilea pas se elimină acest element și ajungem la șirul vid. Cele mai multe elemente s-au eliminat la primul pas, respectiv 4 elemente.

Descrierea soluției

Problema ne cere să aflăm numărul de secvențe care vor fi șterse în condițiile specificate în enunț, respectiv secvențele de elemente delimitate de valori cu suma maximă a cifrelor.

Soluție în O(N * Smax). Problema se rezolvă simulând pașii prezentați în enunț. La prima vedere numărul mare de elemente din șir ne face să credem că nu ne vom încadra în timp cu un algoritm care la fiecare pas calculează prima și ultima poziție pe care se află maximul în șir (încă de la citire putem afla suma cifrelor fiecărei valori și să memorăm doar aceste sume). Întrucât valoarea maximă a unei sume de cifre este 81 (obținută pentru valoarea 999.999.999). Notăm Smax = 81. Observăm că de fapt sunt necesare maxim Smax de parcurgeri ale șirului de numere, așadar algoritmul care simulează operațiile descrise se va încadra în timp.

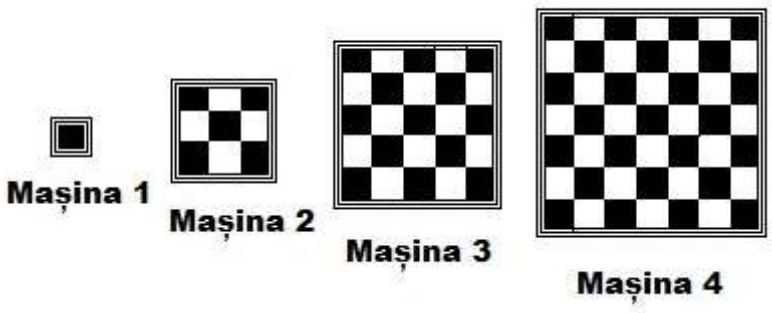
2.2 Problema Formula1

Propunător: prof. Ana-Maria Arișanu

Colegiul Național "Mircea cel Bătrân", Rm. Vâlcea

Enunț

La o cursă de *Formula 1*, fiecare echipă participantă își construiește propria mașină cu care va concura. Numerotarea mașinilor în concurs este realizată de organizatori, cu ajutorul unor steguleț pătrate ce conțin alternativ, pe fiecare rând (pe orizontală și verticală), pătrățele albe și negre de dimensiuni identice. În figura următoare sunt prezentate, în ordine, steguleț primelor 4 mașini din concurs. Observăm că fiecare steguleț are cu două rânduri (pe orizontală și verticală) mai mult decât steguleț precedent, iar în toate cele patru colțuri ale oricărui steguleț se află un pătrățel negru.



Cerințe

Scrieți un program care citește două numere naturale K și N și determină:

1. Câte pătrățele albe și negre în total sunt pe steguleț mașinii cu numărul K;
2. Notând cu A numărul total de pătrățele albe de pe steguleț primelor N mașini din concurs câte pătrățele albe și negre în total sunt pe cel mai mare steguleț care conține cel mult A pătrățele albe.

Date de intrare

Fişierul de intrare **formula1.in** conţine pe prima linie un număr natural C. Pentru toate testele de intrare, numărul C poate avea doar valoarea 1 sau valoarea 2 și reprezintă numărul cerinței care trebuie rezolvată. Pe a doua linie a fișierului **formula1.in** se găsesc, în ordine, numerele naturale K și N.

Date de ieșire

Dacă $C = 1$, se va rezolva cerința 1. În acest caz, fişierul de ieşire **formula1.out** va conține pe prima linie un număr natural reprezentând numărul total de pătrățele existente pe steguleţul mașinii cu numărul K. Dacă $C = 2$, se va rezolva cerința 2. În acest caz, fişierul de ieşire **formula1.out** va conține pe prima linie un număr natural reprezentând numărul total de pătrățele existente pe cel mai mare steguleț ce conține cel mult A pătrățele albe.

Restricții și precizări

- $1 \leq K \leq 100\ 000$
- $1 \leq N \leq 500\ 000$
- Pentru rezolvarea corectă a primei cerințe se obțin 20 de puncte, iar pentru rezolvarea corectă a celei de a doua cerințe se obțin 80 de puncte.

Exemple

1)	`formula1.in`	`formula1.out`	**Explicații**
	`1` `3 4`	`25`	Se rezolvă cerința **1**. Se va folosi doar valoarea K. Steguleţul celei de a treia mașini are 25 de pătrățele albe și negre în total.
2)	`formula1.in`	`formula1.out`	**Explicații**
	`2` `3 4`	`81`	Se rezolvă cerința **2**. Se va folosi doar valoarea N. Pe steguleţele primelor 4 mașini apar, în total, $0 + 4 + 12 + 24 = 40$ pătrățele albe. Cel mai mare steguleț care conține cel mult 40 de pătrățele albe aparține mașinii cu numărul 5 care are în total 81 de pătrățele.

Descrierea soluției

Soluția se poate obține parcurgând șirul numerelor de mere din coșuri și reținând cele mai mici 2 valori (**min1**, **min2**), respectiv cele mai mari 2 valori din șir (**max1**, **max2**).

1. Steguleţul mașinii cu numărul k are $(2k-1)^2$ pătrățele
2. Numărul de pătrățele albe de pe steguleţul mașinii cu numărul k este:

$$\frac{(2k-1)^2 - 1}{2} = k(2k-2) = 2k^2 - 2k \tag{2.1}$$

Prin urmare:

$$A = (2 * 1^2 - 2 * 1) + (2 * 2^2 - 2 * 2) + (2 * 3^2 - 2 * 3) + ... + (2 * N^2 - 2 * N) \quad (2.2)$$

$$A = 2 * (1^2 + 2^2 + 3^2 + ... + N^2) - 2 * (1 + 2 + 3 + ... + N) \quad (2.3)$$

Valoarea lui A se poate determina în 2 moduri:

- iterativ (un for de la 1 la N)
- cu formula $A = 2 * \frac{N(N+1)*(2N+1)}{6} - 2 * \frac{N(N+1)}{2} = \frac{N(N+1)*(2N+1)}{3} - N(N+1)$

Ambele modalități de calcul iau 100 de puncte.

Presupunem că cel mai mare steguleț care are cel mult A pătrățele albe aparține mașinii cu numărul x. Astfel, acest steguleț va avea $(2x-1)^2$ pătrățele în total, dintre care $2x^2 - 2x$ albe.

Prin urmare, x se determină ca fiind cel mai mare număr impar cu proprietatea că $2x^2 - 2x \leq A$

Determinarea lui x se poate face în următoarele moduri:

- Pornim iterativ cu x de la 1 până la cel mai mare x care încă respectă $2x^2 - 2x \leq A$. ($\rightarrow$ scor obținut: 70-80 de puncte)
- Ne folosim de căutarea binară pentru a găsi în timp logaritmic cea mai mare valoare impară x care respectă $2x^2 - 2x \leq A$. ($\rightarrow$ scor obținut: 100 de puncte)
- Putem calcula și formulă directă: $x = \frac{(int)\sqrt{2*A+1}+1}{2}$. ($\rightarrow$ scor obținut: 100 de puncte)

2.3 Problema Seism

Propunător: prof. Cristina Iordaiche

Liceul Teoretic "Grigore Moisil", Timișoara

Enunț

Cercetătorii de la NASA au instalat pe Marte un seismograf cu ajutorul căruia s-au înregistrat mișcările la nivelul solului planetei.

Seismograful a trimis în fiecare din cele N secunde ce definesc perioada de timp analizată, câte un semnal pe Pământ ce a fost codificat de cercetători cu valoarea 1, dacă seismograful a detectat mișcare și 0, în cazul în care nu s-a înregistrat mișcare la nivelul solului planetei.

Astfel, un seism de pe Marte a fost definit de cercetători ca fiind o perioadă continuă de timp în care seismograful a trimiș din secundă în secundă, câte un semnal codificat cu 1 și care începe după cel puțin două semnale codificate cu 0, iar la sfârșitul ei sunt înregistrate cel puțin două semnale codificate cu 0.

Cerințe

Cunoscând șirul celor N valori transmise în ordine de seismograf, scrieți un program care să determine:

1. Care a fost durata maximă, exprimată în secunde a unui seism;
2. Câte seisme au avut loc în perioada de timp analizată;
3. Din cauza unei erori tehnice, o perioadă continuă de timp seismograful a transmis eronat. Astfel, în șirul inițial format din cele N semnale, trebuie să înlocuim valoarea 0 cu valoarea 1, într-o singură secvență, de lungime nevidă, de elemente nule alăturate. Analizând toate posibilitățile de a face această modificare, determinați durata maximă a unui seism care se obține după modificarea șirului inițial de semnale.

Date de intrare

Fişierul de intrare **seism.in** conţine pe prima linie, un număr natural C care poate avea valorile 1, 2 sau 3 şi reprezintă numărul cerinţei. Pe cea de-a doua linie, un număr natural N având semnificația din enunț. Pe următoarea linie, N numere naturale despărțite prin câte un spațiu, reprezentând codificarea semnalului transmis de seismograf, din secundă în secundă, începând cu secunda 1 și până la secunda N.

Date de ieșire

Fişierul de ieşire **seism.out** va conţine pe prima linie un singur număr natural, reprezentând rezultatul determinat conform cerinței.

Restricții și precizări

- $5 \leq N \leq 100\ 000$
- 1 secundă $\leq$ durata unui seism $\leq N - 4$ secunde
- Pentru cerințele 1 și 2 se garantează că seismograful a detectat cel puțin un seism.
- La cerința 3 se garantează că există cel puțin o secvența nevidă de elemente egale cu 0 ce pot fi schimbate în 1 pentru a avea cel puțin un seism în tot șirul.
- Pentru rezolvarea corectă a primei cerințe se obțin 40 de puncte, pentru rezolvarea corectă a celei de a doua cerințe se obțin 40 de puncte, iar pentru rezolvarea corectă a celei de a treia cerințe se obțin 20 de puncte.

Exemple

1)	`seism.in`	`seism.out`	**Explicații**

	`1`
	`21`
	`0 0 1 1 1 1 0 0 0 0 0 1 0 1 0 0 1 1 0`
	`0 1`
		`4`	Se rezolvă cerința **1**.
			Durata maximă a unui seism este de 4 secunde.

2)	`seism.in`	`seism.out`	**Explicații**

	`2`
	`21`
	`0 0 1 1 1 1 0 0 0 0 0 1 0 1 0 0 1 1 0`
	`0 1`
		`3`	Se rezolvă cerința **2**.
			Seismograful a înregistrat 3 seisme. Primul seism are durata de 4 secunde, al doilea are durata de 1 secundă și ultimul are durata de 2 secunde.

3)	`seism.in`	`seism.out`	**Explicații**

	`3`
	`8`
	`0 0 1 1 0 1 0 0`
		`4`	Se rezolvă cerința **3**.
			Elementul din șir de pe poziția 5 se schimbă în 1 și se obține un seism de durată 4 secunde.

4)	`seism.in`	`seism.out`	**Explicații**

	`4`
	`14`
	`0 1 1 0 0 0 0 0 0 0 0 0 1 0`
		`4`	Se rezolvă cerința **4**.
			Se schimbă în 1 semnalele asociate secundelor 6, 7, 8, 9 și 10 și se obține un seism de durată 5 secunde.

Descrierea soluției

Cerințele 1 și 2

Notăm vectorul de valori transmise cu v.

O soluție ar fi să parcurgem șirul celor N semnale și să ne oprim la fiecare poziție i unde $v[i] = 1$, $v[i-1] = 0$ și $v[i-2] = 0$. În continuare trebuie analizat dacă acest i poate fi capătul din stânga al unui seims. Putem găsi exact porțiunea formată din valori alăturate de 1 prin iterarea la dreapta a unui j ce pornește din i și avansează atâta timp cât $v[j] = 1$. Apoi, dacă $v[j+1] = 0$ și $v[j+2] = 0$, atunci secvență $[i, j]$ este un seims valid.

Cu această abordare putem trece prin absolut toate seismele și calcula care este cel de durată maximă (cerința 1). În cazul cerinței 2 trebuie doar să contorizăm numărul de seisme găsite.

Soluția descrisă are complexitate $O(n)$ și obține punctajul maxim pe primele 2 cerințe. Motivația complexității liniare este că j-ul începe mișcarea la dreapta doar după ce am fixat capătul stânga al unei secvențe de 1, iar secvențele de 1 nu se pot suprapune.

Cerința 3

Pentru corectarea erorilor transmise de seismograf, analizăm fiecare secvență din șir ce este formată doar din zerouri. Notăm capetele secvenței analizate: i_1 și i_2. Identificăm 4 cazuri de modalități de corectare:

- **Cazul 1** înlocuim toate zerourile din secvența găsită cu valori de 1. Figura 2.1 prezintă cum ne putem deplasa la stânga până la j_1 și la dreapta până la j_2 atâta timp cât trecem numai prin valori de 1. Verificăm apoi dacă secvența cu capetele în j_1 și j_2 are în stânga și în dreapta sa cel puțin două elemente de 0. În caz afirmativ am identificat un seism și îi calculăm durata.

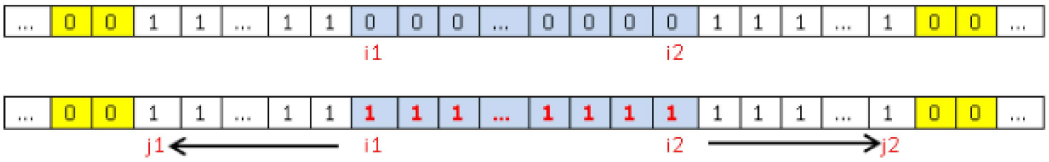


Figure 2.1: Cazul 1, transformare a tuturor elmentelor din $[i_1, i_2]$ în 1

- **Cazul 2** înlocuim numai zerourile din intervalul $[i_1 + 2, i_2 - 2]$ ca în Figura 2.2. Această construcție are sens doar dacă $i_2 - i_1 \geq 4$. Acest seims are durata de $i_2 - i_1 - 3$ secunde.

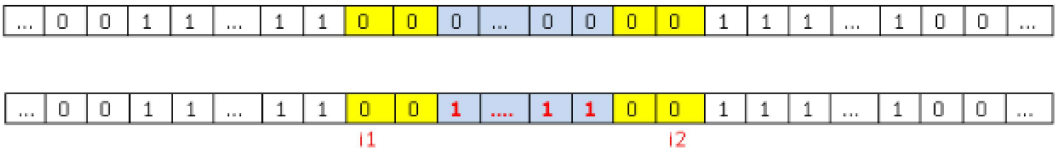


Figure 2.2: Cazul 2, transformare a zerourilor din $[i_1 + 2, i_2 - 2]$ în 1

- **Cazul 3** înlocuim zerourile din intervalul $[i_1, i_2 - 2]$ ca în Figura 2.3. Această construcție are sens doar dacă $i_2 - i_1 \geq 2$. Apoi ne extindem la stânga până la poziția j_1, atâta timp cât trecem numai prin valori de 1. Dacă $v[j_1 - 1] = 0$ și $v[j_1 - 2] = 0$, atunci seismul $[j_1, i_2 - 2]$ este valid și trebuie să îi verificăm lungimea

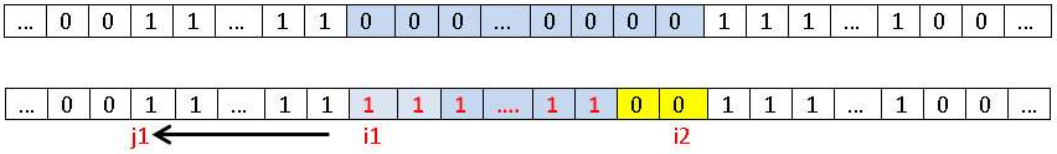


Figure 2.3: Cazul 3, transformare a zerourilor din $[i_1, i_2 - 2]$ în 1

- **Cazul 4** foarte simular cu precedentul caz: înlocuim zerourile din intervalul $[i_1 + 2, i_2]$. Apoi trebuie să ne extindem la dreapta până la poziția j_2 și verificăm dacă $v[j_2 + 1] = 0$ și $v[j_2 + 2] = 0$; Figura 2.4

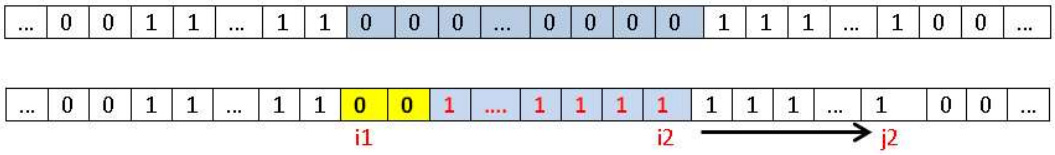


Figure 2.4: Cazul 4, transformare a zerourilor din $[i_1 + 2, i_2]$ în 1

Complexitatea finală de evaluare a tuturor celor 4 cazuri este $O(n)$

Capitolul 3

Clasa a VII-a

3.1 Problema Pătrate

Propunător: prof. Nistor Moţ
Şcoala "Dr. Luca", Brăila

Sesiunea de antrenament

Enunț

Gigel are o masă dreptunghiulară împărţită în m×n pătrate egale. În fiecare pătrat se găseşte cel puţin o bomboană. El are dreptul să ia bomboane din mai multe pătrate care să formeze un "L" de lăţime 1. El poate alege "colţul" L-ului în orice pătrat, singura restricţie fiind ca ramura orizontală şi cea verticală să aibă lungimea cel puţin 2, deci să fie cel puţin încă un pătrat alăturat colţului pe orizontală şi cel puţin unul alăturat pe verticală.

Cerinţă

Găsiţi poziţia L-ului astfel ca numărul bomboanelor de pe pătratele ce compun L-ul să fie maxim.

Date de intrare

Fişierul de intrare `patrate.in` conţine pe prima linie numerele naturale m şi n reprezentând dimensiunea mesei. Pe următoarele m linii sunt câte n numere întregi separate de câte un spaţiu, matricea ce dă numărul de bomboane din fiecare pătrat.

Date de ieșire

Fişierul de ieşire `patrate.out` va conţine două linii. Pe prima linie va fi scris numărul maxim de bomboane pe care le poate lua Gigel, iar pe următoarea linie vor fi scrise 4 numere întregi

separate de câte un spaţiu, precizând poziţia L-ului ce conţine acest număr maxim de bomboane. Poziţia e dată astfel: coordonatele colţului (linia şi coloana sa, numerotarea făcându-se de la 1), apoi lungimea ramurii orizontale, apoi lungimea ramurii verticale. Lungimile ramurilor înseamnă numărul de pătrate (inclusiv colţul) care alcătuiesc ramura, precedat de semnul minus dacă ramura orizontală este în stânga colţului, respectiv ramura verticală este în josul colţului. Dacă sunt mai multe soluţii posibile se alege cea care respectă, în ordine, criteriile următoare: are linia colţului minimă, are coloana colţului minimă, are ramura orizontală în stânga, are ramura verticală în jos.

Restricții

- $2 \leq m, n \leq 200$
- numărul de bomboane dintr-un pătrat este cel mult 1000

Exemplu

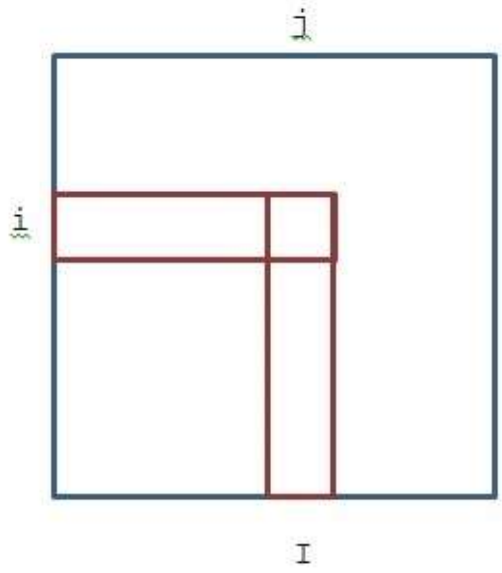
1)	patrate.in	patrate.out
	4 4 3 1 1 4 2 1 9 2 6 1 1 8 4 3 2 1	24 2 1 4 -3

Descrierea soluției

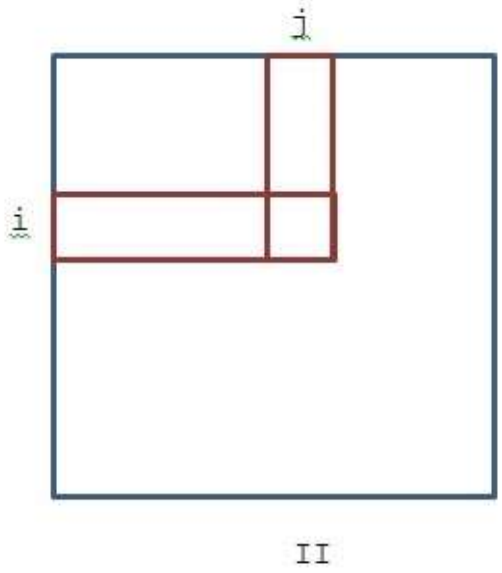
Vom construi două matrice `SL` şi `SC`, în care:

- SL[i,j]=suma primelor `j` numere din linia `i`
- SC[i,j]=suma primelor `i` numere din coloana `j`.

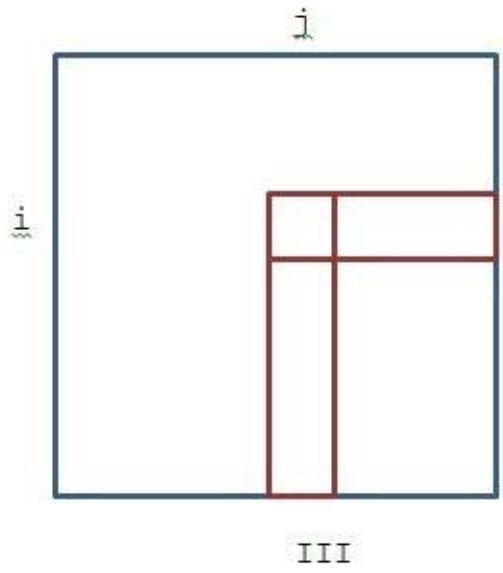
Apoi pentru fiecare poziţie `(i,j)` a colţului vom verifica cele 4 posibile L-uri (evident, în fiecare direcţie mergem până la margine).
Pentru simplificarea codului vom presupune existenţa unei borduri (câte o linie, respectiv coloană) iniţializată cu 0.



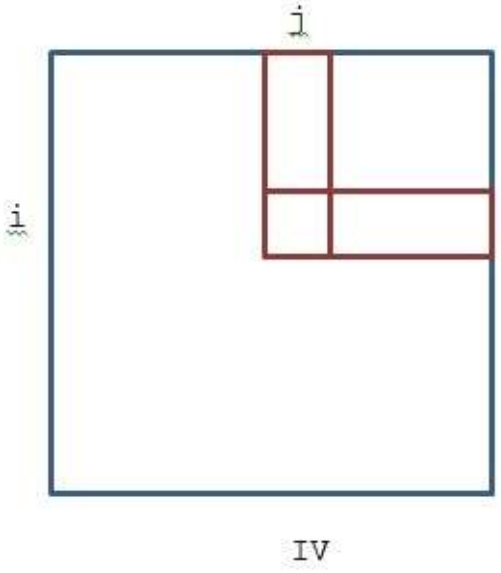
I



II



III



IV

3.2 Problema Campionat

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

Enunț

Ne aflăm la un anumit moment al desfășurării campionatului naţional de fotbal. O parte dintre meciuri s-au jucat, altele urmează să fie disputate. Se cunoaște numărul de puncte acumulate deja de fiecare echipă înaintea desfășurării meciurilor restante.

Se cunoaște, de asemenea, că un meci se poate termina egal, caz în care fiecare dintre echipe primește câte un punct, sau cu victoria uneia dintre echipe, iar în acest caz acea echipă primește trei puncte, iar cealaltă zero puncte.

Cerinţă

Avem de răspuns la întrebări de două tipuri:

1. Care echipe ar fi pe locul I dacă toate meciurile restante s-ar termina la egalitate? O echipa este pe locul I dacă are număr maxim de puncte.
2. Care echipe depind strict de propriile rezultate pentru a deveni campioane? O echipă devine campioană (câștigă campionatul) dacă termină cu număr de puncte strict mai mare decât oricare dintre celelalte echipe. Spunem că o echipă depinde strict de propriile rezultate pentru a deveni campioană dacă ea devine campioană câştigând toate meciurile pe care trebuie să le mai joace, indiferent de rezultatele celorlalte meciuri.

Date de intrare

Fişierul de intrare *campionat.in* conţine pe prima linie un număr T, reprezentând tipul de întrebare (1 sau 2).

Pe linia a doua se află un număr N reprezentând numărul de echipe din campionat (considerăm că echipele sunt etichetate cu numere distincte de la 1 la N).

Pe linia a treia se află N numere naturale separate prin câte un spațiu, al i-lea număr reprezentând, punctajul celei de-a i-a echipe.

Pe linia a patra se află un număr D, reprezentând numărul de meciuri restante.

Pe fiecare dintre următoarele D linii se află câte două numere distincte i, j, cuprinse între 1 și N, cu semnificaţia că echipele i și j au de disputat un meci restant.

Date de ieşire

Fişierul de ieşire *campionat.out* va conţine o singură linie.

- Dacă $T = 1$, linia va conţine etichetele echipelor care termină pe locul I, în cazul în care toate meciurile restante se termină la egalitate.
- Dacă $T = 2$, linia va conţine etichetele echipelor care depind strict de propriile rezultate pentru a deveni campioane. Dacă nicio echipă nu poate deveni campioană depinzând doar de rezultatele sale, în fișierul de ieșire se va scrie doar numărul 0.

Atât pentru $T = 1$, cât şi pentru $T = 2$ etichetele echipelor vor fi scrise în ordine crescătoare, separate prin câte un spaţiu.

Restricţii şi precizări

- $2 \leq N \leq 1000$
- $1 \leq D \leq 500000$
- Punctajele inițiale ale echipelor sunt numere naturale cel mult egale cu 1000.
- Regulile de desfășurare a campionatului sunt mai ciudate, nu trebuie să vă puneți problema dacă este posibil ca echipele să aibă șirul dat al punctajelor în urma meciurilor disputate deja (considerăm că până la momentul de față federația a acordat diverse bonusuri și depunctări).
- Dacă între meciurile rămase de jucat este vreunul care apare de mai multe ori (fie sub forma (i, j) fie sub forma (j, i)), el se va disputa o singură dată.
- Programarea meciurilor s-a făcut în mod indisciplinat, deci este posibil ca unele echipe să mai aibă de jucat mai multe meciuri decât altele, iar unele chiar să nu mai aibă de jucat niciun meci.
- Pentru teste valorând 22 de puncte $T = 1$.
- Pentru alte teste valorând 9 puncte $T = 2$ și fiecare echipă are de disputat exact 2 meciuri cu alte echipe.
- Pentru alte teste valorând 8 puncte $T = 2$ și fiecare echipă are de disputat câte un meci cu fiecare altă echipă.
- Pentru alte teste valorând 10 puncte $T = 2$ și există o singură echipă care joacă câte un meci cu fiecare altă echipă, celelalte echipe neavând alte meciuri restante de jucat.

Exemple

1)	`campionat.in`	`campionat.out`	**Explicații**

	`1`
	`4`
	`2 3 2 1`
	`3`
	`1 3`
	`1 2`
	`3 1`
		`1 2`	Echipa 2 are de disputat un singur meci cu echipa 1, iar al doilea meci se va disputa între echipele 1 și 3 (chiar dacă în listă apare de două ori perechea 1 3, este vorba despre un singur meci). Cele două jocuri se vor termina la egalitate şi astfel punctajele echipelor la final vor fi: 4 4 3 1. Așadar echipele 1 si 2 ies pe primul loc.

2)	`campionat.in`	`campionat.out`	**Explicații**

	`2`
	`4`
	`1 3 2 1`
	`3`
	`1 3`
	`1 2`
	`3 1`
		`1 2`	Echipa 1 devine campioană dacă învinge în ambele meciuri, nicio altă echipă neputând să o egaleze la puncte sau să o depășească. De asemenea, echipa 2 devine campioană dacă învinge echipa 1 în meciul restant, indiferent de rezultatul celuilalt meci. Echipa 3, chiar dacă învinge în meciul cu echipa 1, nu depinde doar de rezultatul său întrucât echipa 2 ar depăși-o la puncte dacă și ea ar câștiga meciul pe care îl mai are de disputat.

Descrierea soluției

Pentru fiecare echipă i $(1 \leq i \leq N)$ vom face următoarele notații:

- cu $scor[i]$ vom nota scorul inițial al echipei i, pe care îl vom citi din input.
- cu $meciuri[i]$ vom nota numărul de meciuri restante pe care le mai are de jucat echipa i.
- Deoarece meciurile restante pot apărea în mod repetat în fişierul de intrare, vom construi un tablou A cu N linii şi N coloane cu elemente din mulţimea $\{0, 1\}$ astfel încat $A[i][j] = A[j][i] = 1$ dacă echipele i şi j mai au de jucat un meci restant, respectiv 0 în caz contrar.

Pentru a determina numărul de meciuri restante pe care trebuie să le mai joace echipa i este suficient să adunăm valorile de pe linia i a tabloului A:

$$meciuri[i] = A[i][1] + A[i][2] + \cdots + A[i][N-1] + A[i][N]$$

Cerinţa 1.

Pentru cerinţa 1 este suficient să determinăm punctajul final pe care fiecare echipă îl va obţine atunci când toate jocurile se termină la egalitate.

Acest scor este egal cu suma dintre punctajul iniţial al echipei şi numărul de jocuri restante pe care echipa respectivă le mai are de jucat. Aflăm maximul dintre aceste punctaje finale $scor[i]+meciuri[i]$, $\forall\, i \in \{1, 2, \ldots, n\}$, iar pe locul I se vor situa toate echipele care au punctajul final egal cu punctajul final maxim.

Cerinţa 2.

O echipă i poate deveni campioană strict pe baza propriilor rezultate dacă după ce va câştiga toate meciurile pe care le mai are de jucat, indiferent de rezultatele celorlalte meciuri, va avea un punctaj final strict mai mare decât al tuturor celorlalte echipe.

Vom parcurge toate echipele succesiv şi vom verifica pentru fiecare dacă poate indeplini acest criteriu.

Să analizăm dacă echipa i poate deveni campioană strict pe baza propriilor rezultate. Punctajul final al echipei, dacă va câştiga toate meciurile, va fi egal cu $scor[i] + 3 \cdot meciuri[i]$. Să notăm acest punctaj cu $scorMax[i]$.

Vom determina pentru fiecare echipă $j, (j \neq i)$ care ar fi punctajul maxim pe care aceasta l-ar putea obţine (considerăm că j va câştiga toate meciurile restante în care este implicată, exceptând meciul pe care eventual l-ar juca cu echipa i). astfel scorul maxim pe care il poate obtine echipa j este $scorMax[j] - 3 \cdot A[i][j]$.

Dacă $scorMax[i]$ este strict mai mare decât punctajul maxim al oricărei alte echipe j, atunci i poate deveni campioană.

3.3 Problema Exclusiv

Propunător: prof. Nistor Moț
Școala "Dr. Luca", Brăila

Enunț

Se consideră doi vectori care conţin numere naturale: s cu M elemente şi v cu N elemente. Numim secvenţă $i-exclusivă$ o secvenţă a vectorului s care nu conţine niciuna dintre valorile $v[1], v[2], \cdots, v[i]$.

Cerinţă

Scrieţi un program care să determine, pentru orice $1 \leq i \leq N$, lungimea maximă a unei secvenţe $i-exclusive$.

Date de intrare

Fişierul de intrare *exclusiv.in* conţine pe prima linie numerele naturale M şi N. Pe linia a doua se află M numere naturale reprezentând elementele vectorului s, iar pe linia a treia N numere naturale reprezentând elementele vectorului v. Valorile scrise pe aceeaşi linie sunt separate prin câte un spaţiu.

Date de ieşire

Fişierul de ieşire *exclusiv.out* va conţine N linii. Pe linia i $(1 \leq i \leq N)$ va fi scris un număr natural care reprezintă lungimea maximă a unei secvenţe $i-exclusive$.

Restricţii şi precizări

- $1 \leq N \leq 2000$
- $3 \leq M \leq 10^5$
- Vectorii s şi v conţin numere naturale $\leq 2 \cdot 10^9$, memorate începând cu poziţia 1
- Valorile din fiecare vector nu sunt obligatoriu distincte două câte două.
- O subsecvenţă **nevidă** în s este formată din elemente situate pe poziţii consecutive $(s[i], s[i+1], \cdots, s[j]),\ i \leq j$. O subsecvenţă $i-exclusivă$ poate fi și vidă, lungimea ei fiind 0.
- Pentru teste valorând 10 puncte $N = 1$.
- Pentru alte teste valorând 30 de puncte $1 < N \leq 50$ şi $M \leq 1000$.
- Pentru alte teste valorând 40 de puncte $50 < N \leq 2000$ şi $1000 < M \leq 2000$.
- Pentru alte valorând 20 de puncte $N = 2000$ şi $10^4 < M \leq 10^5$.

Exemplu

exclusiv.in	exclusiv.out
20 6 11 5 11 7 2 10 11 9 2 77 88 88 88 2 7 2 2 77 2 11 11 5 7 9 5 2	12 12 7 6 6 4

Explicație

Cea mai lungă secvenţă 1−*exclusivă* (care nu conţine valoarea 11) este 9 2 77 88 88 88 2 7 2 2 77 2 şi are lungimea 12.

Cea mai lungă secvenţă 2−*exclusivă* (care nu conţine valorile 11 şi 5) este 9 2 77 88 88 88 2 7 2 2 77 2 şi are lungimea 12.

Cea mai lungă secvenţă 3−*exclusivă* (care nu conţine valorile 11, 5 şi 7) este 9 2 77 88 88 88 2 şi are lungimea 7.

Cea mai lungă secvenţă 4−*exclusivă* (care nu conţine valorile 11, 5, 7 şi 9) este 2 77 88 88 88 2 şi are lungimea 6.

Cea mai lungă secvenţă 5−*exclusivă* (care nu conţine valorile 11, 5, 7, 9 şi 5) este 2 77 88 88 88 2 şi are lungimea 6.

Cea mai lungă secvenţă 6−*exclusivă* (care nu conţine valorile 11, 5, 7, 9, 5 şi 2) este 77 88 88 88 şi are lungimea 4.

Descrierea soluției

Soluție de 100 puncte – complexitate $\mathcal{O}(N^2 + M \cdot log(N))$

Vom răspunde la cele N întrebări de tipul *"care este lungimea maximă a unei secvenţe i − exclusive"* pentru i de la N la 1. Motivația acestui lucru este faptul că dacă parcurgem întrebările în sens invers, vom adăuga elemente noi la vectorul s, care initial va fi gol, în loc să stergem elemente.

Pentru a rezolva problema într-o complexitate optimă, vor trebui efectuate urmatoarele transformari:

- După ce se citesc datele de intrare se ordonează perechile $(v[i], i)$ crescător în funcție de valoare, iar în caz de egalitate crescător după poziție.
- Pentru fiecare element $s[j]$ se caută, utilizând o căutare binară, cea mai mică poziție i astfel incat $s[j] = v[i]$ și se înlocuiește $s[j]$ cu i. În cazul in care $s[j]$ nu apare în v vom înlocui $s[j]$ cu valoarea $N+1$. Astfel am obținut un vector echivalent cu cel citit din punct de vedere al răspunsului, dar care are valori din mulțimea $\{1, 2, \ldots, N, N+1\}$.

În continuare, vom realiza următoarele construcții pentru a actualiza cât mai rapid actualizarea de la o întrebare la alta.

- Vom construi un vector urm definit prin $urm[j]$ = următoarea poziție din s unde apare $s[j]$, respectiv $urm[j] = M + 1$, dacă $s[j]$ nu mai apare în dreapta poziției j. Cu ajutorul acestui vector urm vom parcurge, pentru fiecare valoare $v[i]$ doar pozițiile din s unde apare valoarea i. Astfel parcurgerea tuturor valorilor din v în s se poate trecând o singură dată peste fiecare element din s.

- Vom utiliza și vectorii $stanga$ și $dreapta$, cu semnificația, $stanga[j]$ = începutul intervalului maximal din care face parte elementul de pe poziția j, respectiv $dreapta[j]$ = sfârșitul intervalului maximal din care face parte elementul de pe poziția j. Acești vectori ne vor ajuta să identificăm secvențele maximale din s care nu conțin valori interzise la acea etapă. Vectorii vor fi inițializați cu $stanga[j] = dreapta[j] = 0,\ \forall j \in \{1, 2, \ldots, M\}$.

Vom parcurge cu $i = N + 1, N, \ldots, 3, 2$. La etapa i vom determina secvențele maximale din s care nu au voie să conțină valori din mulțimea $\{1, 2, \ldots, i - 1\}$. Pe parcursul determinării acestor secvențe maximale vom actualiza valorile $lMax[i]$, care vor conține la final rezultatele căutate.

O observație importantă este aceea că la etapa i valoarea i nu poate apărea în s în interiorul niciunei secvențe maximale obținute la etapele anterioare, deoarece ea a fost interzisă la etapele anterioare. Astfel valoarea i poate forma secvențe maximale formate doar din valoarea i (de lungime 1) sau poate extinde sau unifica secvențele maximale formate anterior. Astfel vom realiza următorii pași pentru fiecare etapă:

1. De fiecare dată când găsim $s[j] = i$ actualizăm cu $stanga[j] = j$ și $dreapta[j] = j$, ca pe o secvență independentă;

2. Analizăm la stânga și apoi la dreapta poziției j. Avem următoarele cazuri:

   - Dacă $stanga[j - 1] > 0$, vom extinde secvența spre stânga astfel:

     $$stanga[j] = stanga[j - 1]$$

   - Dacă $dreapta[j + 1] > 0$, vom extinde secvența spre dreapta astfel:

     $$dreapta[j] = dreapta[j + 1]$$

3. Finalizarea se face cu:

   $$dreapta[stanga[j]] = dreapta[j]$$
   $$stanga[dreapta[j]] = stanga[j]$$

Lungimea secvenței maximale curente este actualizată la fiecare pas:

$$lMax[i] = max(lMax[1], dreapta[j] - stanga[j] + 1)$$

La final se afișează valorile $lMax[1], lMax[2], \ldots, lMax[N]$.

Trebuie avut grijă la implementare deoarece apar diverse cazuri particulare. De exemplu: valorile din v nu apar în s, valorile din v nu sunt distincte, toate valorile din s apar și în v.

Capitolul 4

Clasa a VIII-a

4.1 Problema Coșuri

Propunător: stud. Ioan-Cristian Pop

Universitatea din București

Sesiunea de antrenament

Enunț

Se consideră $2 \cdot N$ coșuri numerotate cu numerele distincte de la 1 la $2 \cdot N$. Coșul 1 conține C_1 mere, coșul 2 conține C_2 mere, ..., coșul $2 \cdot N$ conține $C_{2 \cdot N}$ mere.

Cele $2 \cdot N$ coșuri vor fi grupate două câte două, rezultând N perechi de coșuri. Fiecare coș poate face parte dintr-o singură pereche. Numărul de mere dintr-o pereche de coșuri este egal cu suma numerelor de mere din cele două coșuri. Pentru fiecare pereche, valoarea absolută a diferenței numerelor de mere din cele două coșuri ale perechii reprezintă *excedentul* perechii.

Ionuț, având o fire curioasă, vrea să afle mai multe despre aceste perechi.

Mai întâi vrea să afle numărul minim posibil, respectiv maxim posibil, de mere dintr-o pereche, indiferent de gruparea coșurilor în perechi de câte două.

Apoi, Ionuț vă întreabă dacă este posibil să grupeze cele $2 \cdot N$ coșuri în perechi de câte două, astfel încât cele N perechi rezultate să aibă același număr de mere. Dacă este posibil, atunci Ionuț vrea să afle și care este valoarea minimă a excedentelor perechilor din noua grupare.

Voi va trebui să îl ajutați pe Ionuț să afle răspunsurile la întrebările lui.

Cerințe

Scrieți un program care să citească numărul N și cele $2 \cdot N$ numere naturale C_1, C_2, ..., $C_{2 \cdot N}$, iar apoi să determine:

1) Numărul minim de mere și numărul maxim de mere pe care le pot avea perechile de coșuri.

2) Răspunsul **DA** sau **NU** la întrebarea lui Ionuț; dacă răspunsul este **DA**, se va determina și care este valoarea minimă a excedentelor perechilor din noua grupare.

Date de intrare

Fișierul `cosuri.in` conține pe prima linie două numere naturale T și N separate printr-un spațiu. A doua linie a fișierului conține cele $2 \cdot N$ numere naturale C_1, C_2, ..., $C_{2 \cdot N}$, separate prin câte un spațiu.

Date de ieșire

Fișierul `cosuri.out` va conține:

1) Dacă $T = 1$, pe prima linie, două numere naturale separate printr-un singur spațiu reprezentând numărul minim, respectiv maxim de mere pe care le pot avea perechile (răspunsul la cerinţa 1);

2) Dacă $T = 2$, pe prima linie, un șir de două caractere care poate fi **DA** sau **NU**, reprezentând răspunsul la întrebarea lui Ionuț; dacă răspunsul este **DA**, atunci a doua linie a fișierului va conține un număr natural reprezentând valoarea minimă a excedentelor perechilor din noua grupare (răspunsul la cerința 2).

Restricții

- $1 \leq N \leq 500\ 000$
- $1 \leq C_i \leq 1\ 000\ 000$, pentru oricare $1 \leq i \leq 2 \cdot N$
- Pentru teste în valoare de 30 de puncte, $T = 1$
- Pentru restul de 70 de puncte, $T = 2$
- Pentru teste în valoare de 20 de puncte, $N \leq 5$
- Pentru alte teste în valoare de 40 de puncte, $N \leq 70000$

Observație: Valoarea absolută a numărului pozitiv X este egală cu X; valoarea absolută a numărului negativ Y este $-Y$

Exemple

1)	`cosuri.in`	`cosuri.out`	**Explicații**
	`1 3` `1 7 4 10 2 9`	`3 19`	Se rezolvă cerința **1**. Perechea formată din coșurile 1 și 5 are 3 mere, iar perechea formată din coșurile 4 și 6 are 19 mere. Se observă că aceste valori sunt minimul, respectiv maximul posibil.
2)	**`cosuri.in`**	**`cosuri.out`**	**Explicații**
	`2 3` `1 7 4 10 2 9`	`DA` `3`	Se rezolvă cerința **2**. Grupând coșurile 1 cu 4, 2 cu 3 și 5 cu 6, obținem 3 perechi cu câte 11 mere fiecare ($1 + 10$, $7 + 4$, $2 + 9$) și cu excedentele: $9(= \lvert 1 - 10 \rvert)$, $3(= \lvert 7 - 4 \rvert)$, $7(= \lvert 2 - 9 \rvert)$. Valoarea minimă a excedentelor perechilor din noua grupare este 3.
3)	**`cosuri.in`**	**`cosuri.out`**	**Explicații**
	`2 3` `1 7 5 12 2 9`	`NU`	Se rezolvă cerința **2**. Este imposibil să formăm perechile de coșuri, deci răspunsul este NU.

Descrierea soluției

Cerința 1 (30 de puncte)

Soluția se poate obține parcurgând șirul numerelor de mere din coșuri și reținând cele mai mici 2 valori (**min1**, **min2**), respectiv cele mai mari 2 valori din șir (**max1**, **max2**).

Numerele cerute sunt **min1** + **min2** și **max2** + **max2**. Complexitate: **O**(**N**).

Cerința 2 (70 de puncte)

- O soluție cu punctaj parțial se poate obține generând toate perechile de câte două coșuri (de exemplu, utilizând două for-uri, în complexitate $O(N^2)$ (setând suma perechii, cautam pentru fiecare coș perechea lui) sau tehnica Backtracking, în complexitate exponențială) (punctaj parțial: 20 de puncte).
- O altă soluție se poate obține prin sortarea șirului numerelor de mere din coșuri. Observăm că, prin sortarea valorilor din coșuri, perechile ideale ar fi: primul coș cu ultimul, al doilea coș cu penultimul etc. Pentru fiecare pereche, se va calcula excedentul, și se va memora valoarea minimă a excedentelor.

  Sortând șirul din exemplu 1 9 4 2 10 7, obținem șirul 1 2 4 7 9 10. Se vor forma 3 perechi de sumă 11: $1 + 10$, $2 + 9$, $4 + 7$, iar excedentul minim este 3.

  Punctajul și complexitatea unui astfel de algoritm sunt date de complexitatea sortării (punctaj maxim pentru $O(N \log N)$).
- O soluție de punctaj maxim folosește un vector de frecvență pentru a optimiza soluția anterioară, valoriile find maxim 1 000 000. Complexitatea este $O(N)$.

4.2 Problema Cartofi

Propunător: prof. Flavius Boian
Colegiul Național "Spiru Haret", Târgu Jiu

Enunț

Fermierul Feder cultivă cartofi pe un teren dreptunghiular de lățime **N** metri și lungime **M** metri, compartimentat în $\mathbf{N} \cdot \mathbf{M}$ zone pătratice identice de lungime **1** metru, dispuse alăturat, câte **N** pe lățime (pe **N** linii, numerotate de la **1** la **N**) și câte **M** pe lungime (pe **M** coloane, numerotate de la **1** la **M**).

În fiecare zonă pătratică se află câte o plantă de cartofi. Parcurgând terenul de la prima linie către ultima, fiecare linie cu număr impar parcurgând-o de la coloana **1** către coloana **M**, iar fiecare linie cu număr par parcurgând-o de la coloana **M** către coloana **1**, fermierul (pasionat de matematică) a scris numerele cartofilor produși de fiecare plantă, în ordinea parcurgerii, și a constatat că a obținut șirul cifrelor unităților primilor $\mathbf{N} \cdot \mathbf{M}$ termeni ai șirului Fibonacci (vezi Figura **1** în care $\mathbf{N} = \mathbf{3}$ și $\mathbf{M} = \mathbf{6}$).

	Coloana 1	Coloana 2	Coloana 3	Coloana 4	Coloana 5	Coloana 6
Linia 1	1	1	2	3	5	8
Linia 2	4	9	5	4	1	3
Linia 3	3	7	0	7	7	4

Figura 1

1	1	2	3	5	8
4	9	5	4	1	3
3	7	0	7	7	4

Figura 2

1	1	2	3	5	8
4	9	5	4	1	3
3	7	0	7	7	4
8	7	1	6	5	1
5	3	8	1	9	0

Figura 3

Cerințe

Scrieţi un program care citește numerele **N** și **M** (cu semnificația din enunț), iar apoi determină:

1. numărul plantelor din teren care nu au produs niciun cartof;
2. numărul maxim de cartofi care pot fi produși de plantele dintr-o suprafață pătratică din terenul fermierului;
3. pentru fiecare dintre cele **Q** perechi de numere $(\mathbf{A}, \mathbf{B})$ citite, numărul cartofilor produși de plantele aflate în zonele pătratice situate între coloanele cu numerele **A** și **B**, inclusiv acestea.

Date de intrare

Fișierul `cartofi.in` conține pe prima linie un număr natural **C** reprezentând cerința din problemă care trebuie rezolvată (**1**, **2** sau **3**). A doua linie a fișierului conține cele două numere naturale **N** și **M**, separate printr-un spațiu, cu semnificația din enunț. Dacă $\mathbf{C} = \mathbf{3}$, atunci

fișierul va mai conține pe a treia linie numărul natural **Q**, iar pe fiecare linie dintre următoarele **Q**, câte două numere naturale separate printr-un spațiu. reprezentând câte o pereche de numere $(\mathbf{A}, \mathbf{B})$ dintre cele **Q**.

Date de ieșire

Fișierul `cartofi.out` va conține:

- Dacă $\mathbf{C} = \mathbf{1}$, pe prima linie un număr natural reprezentând răspunsul la cerința **1**.
- Dacă $\mathbf{C} = \mathbf{2}$, pe prima linie un număr natural reprezentând răspunsul la cerința **2**.
- Dacă $\mathbf{C} = \mathbf{3}$, **Q** linii, câte o linie pentru fiecare pereche $(\mathbf{A}, \mathbf{B})$ dintre cele **Q**. Linia corespunzătoare fiecărei perechi $(\mathbf{A}, \mathbf{B})$ va conține un număr natural reprezentând numărul cartofilor produși de plantele aflate în zonele pătratice situate între coloanele cu numerele **A** și **B**, inclusiv aceste valori, reprezentând răspunsul la cerința **3**.

Restricții

- $2 \leq \mathbf{N} \leq 5 \cdot 10^8$
- $3 \leq \mathbf{M} \leq 10^9$
- $\mathbf{N} \leq \mathbf{M}$
- $\mathbf{Q} \leq 10^5$
- $1 \leq \mathbf{A} \leq \mathbf{B} \leq \mathbf{M}$
- Pentru cerința **1** se acordă **20p**, iar pentru cerințele **2** și **3** se acordă câte **40p**.
- Pentru teste în valoare de 17 puncte: $\mathbf{N} \leq 50$, $\mathbf{M} \leq 100$ și $\mathbf{Q} \leq 20$
- Pentru alte teste în valoare de 24 puncte: $\mathbf{N} \leq 100$, $\mathbf{M} \leq 1000$ și $\mathbf{Q} \leq 10000$
- Pentru alte teste în valoare de 31 puncte: $\mathbf{N} \leq 1000$, $\mathbf{M} \leq 10000$ și $\mathbf{Q} \leq 100000$

Observații:

- Șirul Fibonacci este definit astfel: $f_1 = 1$, $f_2 = 1$ și $f_n = f_{n-1} + f_{n-2}$, dacă $n \geq 3$, (n este un număr natural nenul).
- O suprafață pătratică din teren este formată $\mathbf{K} \cdot \mathbf{K}$ zone pătratice alăturate dispuse câte **K** pe linie și câte **K** pe coloană, oricare ar fi $1 \leq \mathbf{K} \leq \min\{\mathbf{N}, \mathbf{M}\}$.

Exemple

1)	`cartofi.in`	`cartofi.out`	**Explicații**
	`1` `3 6`	`1`	Se rezolvă cerința **1**. $\mathbf{N} = 3$, $\mathbf{M} = 6$. Primii $\mathbf{N} \cdot \mathbf{M} = 18$ termeni ai șirului Fibonacci sunt: $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584$. Astfel, numerele cartofilor produși de fiecare plantă din teren sunt cele din Figura **1**. În teren există o singură plantă care nu a produs niciun cartof (cea din linia 3, coloana 3)
2)	`cartofi.in`	`cartofi.out`	**Explicații**
	`2` `3 6`	`42`	Se rezolvă cerința **2**. $\mathbf{N} = 3, \mathbf{M} = 6$. Numerele cartofilor produși de fiecare plantă din teren sunt cele din tabelul din Figura **1**. Plantele aflate în suprafața pătratică galbenă din tabelul din Figura **2** au produs cel mai mare număr de cartofi.
3)	`cartofi.in`	`cartofi.out`	**Explicații**

	`3`
	`5 6`
	`3`
	`1 2`
	`4 6`
	`2 3`
		`48`
		`64`
		`43`
			Se rezolvă cerința **3**.
			$\mathbf{N} = 5, \mathbf{M} = 6, \mathbf{Q} = 3$.
			Tabelul din Figura **3** conține numerele cartofilor produși de fiecare plantă din teren sunt cele din Figura **3**.
			48 =suma elementelor situate între coloanele $(1, 2)$, inclusiv 1 și 2.
			64 =suma elementelor situate între coloanele $(4, 6)$, inclusiv 4 și 6.
			43 =suma elementelor situate între coloanele $(2, 3)$, inclusiv 2 și 3.

Descrierea soluției

Soluție cu punctaj parțial

Cerința 1 (20 de puncte)

- O soluție cu punctaj parțial se poate obține generând șirul cifrelor unităților primilor $\mathbf{N} \cdot \mathbf{M}$ termeni ai șirului Fibonacci, contorizându-se apariția fiecărei cifre 0. Complexitate: $\mathcal{O}(\mathbf{N} \cdot \mathbf{M})$.

Cerința 2 (40 de puncte)

- O soluție cu punctaj parțial se poate obține utilizând matricea $\mathbf{N} \cdot \mathbf{M}$ a numerelor cartofilor produși de plante. Se observă că suprafața pătratică de sumă maximă trebuie căutată printre submatricele $\mathbf{N} \cdot \mathbf{N}$ ale matricei. Astfel, suma numerelor din prima submatrice (formată din coloanele $(\mathbf{1}, \mathbf{N})$) este egală cu suma valorilor din primele $\mathbf{N}$ coloane. Submatricea formată din coloanele $(\mathbf{2}, \mathbf{N} + \mathbf{1})$ va avea suma egală cu suma valorilor din coloanele $(\mathbf{2}, \mathbf{N} + \mathbf{1})$. Această sumă se poate obține și direct, din suma anterioară scăzând suma primei coloane și adăugând suma coloanei $(\mathbf{N} + \mathbf{1})$. Procedăm asemănător pentru celelalte submatrice, ultima fiind formată din ultimele $\mathbf{N}$ coloane. Cea mai mare sumă dintre acestea reprezintă răspunsul la cerința 2.

Pentru a evita calculul repetat al sumelor valorilor din fiecare coloană, putem calcula mai întâi aceste sume într-un vector de sume ($SC[K]$ = suma valorilor din coloana K, $K = 1, 2, ..., \mathbf{M}$). Se observă inutilitatea formării matricei. Calculul sumei fiecărei coloane în vector se poate face simulând construirea matricei, adunând ultima cifră a termenului Fibonacci la suma ce îi corespunde. Complexitate $\mathcal{O}(\mathbf{M})$.

Cerința 3 (40 de puncte)

- O soluție cu punctaj parțial se poate utilizând matricea, calculând pentru fiecare pereche (A, B) suma elementelor situate între aceste coloane.

- O soluție cu un punctaj parțial mai mare ca anteriorul se obține utilizând un alt mod de calculare a sumelor elementelor cuprinse între coloanele (A, B) și constă în calculul sumelor parțiale ale vectorului SC intr-un alt vector S cu $\mathbf{M}$ componente. Astfel, suma elementelor dintre coloanele (A, B) va fi egală cu $S[B] - S[A-1]$, iar $S[0] = 0$. Complexitate $\mathcal{O}(\mathbf{Q} \cdot \mathbf{M})$.

Soluție cu punctaj maxim

Propusă de stud. Ioan-Cristian Pop (UPB) și stud. Ioan-Bogdan Iordache (UB)

General

- Știm că șirul cifrelor unităților termenilor din șirul lui Fibonacii este periodic cu perioada **60**. Fie $\mathbf{R} = \mathbf{60}$. Pe baza acestei proprietăți, putem construi strict primele **60** de elemente din acest șir, iar pe baza acestora putem calcula răspunsul la fiecare cerință.
  De exemplu, al **500**-lea termen din șir va fi egal cu al **20**-lea termen.

Cerința 1 (20 de puncte)

- Plecând de la observația de mai sus, construind vectorul alcătuit din primele **60** de numere din șir. De aici aflăm că cifra **0** apare de **4** ori în acest set. Știind că există $(\mathbf{N} \cdot \mathbf{M})/\mathbf{R}$ seturi complete, rămâne să determinăm numărul de apariții ale cifrei **0** în secvența formată din ultimii $(\mathbf{N} \cdot \mathbf{M})\%\mathbf{R}$ termeni ai vectorului.
  De exemplu, pentru $\mathbf{N} = \mathbf{50}$ și $\mathbf{M} = \mathbf{100}$, obținem $\mathbf{N} \cdot \mathbf{M} = \mathbf{5000}$.
  Știm că vor fi $[\mathbf{5000}/\mathbf{60}] = \mathbf{83}$ seturi complete, iar în ultimii **20** de termeni, **0** se află o singură data. Deci, răspunsul va fi $\mathbf{83} \cdot \mathbf{4} + \mathbf{1} = \mathbf{333}$.
- Complexitate: $\mathcal{O}(\mathbf{R})$.

Cerința 2 (40 de puncte)

- Este suficient să calculăm submatricea din colțul stânga sus (o numim submatricea $\mathbf{A}$, de dimensiuni $\mathbf{min}(\mathbf{N}, \mathbf{60}) \cdot \mathbf{min}(\mathbf{M}, \mathbf{60})$) și să calculăm primele $\mathbf{min}(\mathbf{M} - \mathbf{N} + \mathbf{1}, \mathbf{60})$ matrici de dimensiuni $\mathbf{N} \cdot \mathbf{N}$, începând de la stânga. Matricea are colțul stânga-sus în coordonatele $(\mathbf{1}, \mathbf{1})$.

- Ca să construim matricea, calculăm pentru fiecare poziție (i, j) din matrice al câtelea termen din șir este cu ajutorul formulei:

  - $(\mathbf{i} - \mathbf{1}) \cdot \mathbf{M} + \mathbf{j}$, pentru $\mathbf{i}$ impar;
  - $(\mathbf{i} - \mathbf{1}) \cdot \mathbf{M} + (\mathbf{M} - \mathbf{j} + \mathbf{1})$, pentru $\mathbf{i}$ par.

De exemplu, pentru $\mathbf{N = 50}$, $\mathbf{M = 100}$, termenul de pe poziția $\mathbf{i = 20}$, $\mathbf{j = 30}$ este al $\mathbf{i \cdot (N - 1) + (M - j + 1) = 1021}$-lea termen din șir, care la rândul lui este egal cu primul termen din șir (șirul fiind periodic cu perioada **60**).

- Pentru a construi eficient sumele parțiale pe coloane, determinăm frecvența de apariții a fiecărui termen din matricea **A** pe linii.

  De exemplu, dacă **N** ar fi egal cu **460**, termenul de pe prima linie se va repeta de **7** ori $((\mathbf{460 - 1})/\mathbf{7})$, în timp ce termenul de pe linia **60** se va repeta de **6** ori $((\mathbf{460 - 60})/\mathbf{7})$.

- Pentru a rezolva optim, calculăm inițial suma primei matrice, cu colțul stânga-sus în $(\mathbf{1}, \mathbf{1})$, pe baza sumelor parțiale pe coloane. O variantă ar fi să determinăm frecvența apariției fiecărei coloane în această matrice.

  De exemplu, pentru $\mathbf{N = 79}$, frecvența primei coloane este **2**, în timp ce frecvența celei de-a **30**-a coloane este **1**.

  Apoi, pentru a calcula suma matricei următoare, scădem suma de pe prima coloană și adăugam suma de pe coloana urmatoare (*sliding window*).

  Complexitate $\mathcal{O}(\mathbf{R} \cdot \mathbf{R})$.

Cerința 3 (40 de puncte)

- O soluție cu punctaj mediu se poate obține în maniera descrisă la cerința 2, plecând de la sumele parțiale de pe coloane. Pentru fiecare pereche de coloane $(\mathbf{X}, \mathbf{Y})$ determinăm frecvența de apariție a fiecărei coloane dintre cele **60** și calculăm suma.

  Complexitate: $\mathcal{O}(\mathbf{Q} \cdot \mathbf{R})$.

- O soluție cu punctaj maxim, eficientă, presupune construirea unor sume parțiale pe baza sumelor parțiale de pe coloane (ideea poate fi aplicată și la cerința 2).

- Fie $\mathbf{sp[j]}$ = suma elementelor din primele j coloane. Apoi, pentru fiecare pereche de coloane $(\mathbf{X}, \mathbf{Y})$, aflăm de câte ori se repetă setul de **60** de coloane, și adaugăm restul de coloane neutilizate.

  De exemplu, pentru $\mathbf{X = 45}$ și $\mathbf{Y = 130}$ observăm că este un set complet de coloane care se repetă. Rămâne să adăugăm restul de coloane neutilizate: $\mathbf{45}, \ldots, \mathbf{60}$ și $\mathbf{121}, \ldots, \mathbf{130}$. Cu ajutorul sumelor parțiale, putem calcula imediat aceste sume: $\mathbf{sp[60] - sp[44]}$, respectiv $\mathbf{sp[10] - sp[0]}$.

- Astfel, raspundem in $\mathcal{O}(\mathbf{1})$ la fiecare întrebare. Complexitate finală: $\mathcal{O}(\mathbf{R \cdot R + Q})$ ($\mathbf{R \cdot R}$) provine de la generarea matricei de $\mathbf{60 \cdot 60}$, apoi construirea sumelor parțiale).

4.3 Problema Tunel

AUTOR:
PROF. CRISTINA SICHIM

Problema Tunel este ultima problemă propusă de regretata noastră colegă prof. Cristina Sichim. Pro- blema a fost selectată pentru etapa națională a Olimpiadei de Informatică Gimnaziu - 2020, clasa a VII-a (ONIGim 2020), etapă anulată din cauza pandemiei. Selectarea și pregătirea acestei pro- bleme pentru OSEPI 2021 reprezintă un omagiu pe care îl aducem celei care timp de două decenii ne-a fost o foarte apreciată și respectată colegă în comisiile de organizare a competițiilor naționale și internaționale de informatică din țara noastră.

Enunț

Tommy este un motan alintat care adoră să se plimbe prin orice tunel. De aceea, stăpânii lui i-au construit o nouă jucărie, formată din **N** tuneluri interconectate (etichetate cu numerele distincte de la **1** la **N**). Toate tunelurile au aceeași lungime, sunt formate din **M** elemente unitare identice (numerotate cu numerele distincte de la 1 la **M**) și au ieșiri la ambele capete. Conectarea dintre două tuneluri alăturate se face printr-un element unitar numit pasaj. În exemplul din **Figura 1**, jucăria este formată din **4** tuneluri, fiecare tunel fiind format din **9** elemente unitare. Pentru a fi mai provocator, stăpânii motanului plasează în ultimul element unitar al ultimului tunel o recompensă.

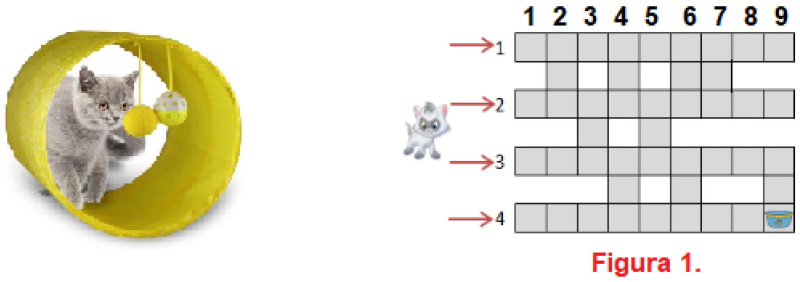


Figura 1.

Motan isteț, Tommy a învățat deja toate regulile jocului:

- poate intra prin capătul din stânga al oricărui tunel (prin elementul unitar 1);
- nu trece de mai multe ori prin același pasaj;
- dacă nu se află lângă un pasaj, continuă să meargă prin tunel către dreapta;
- dacă ajunge la un pasaj, atunci trece prin acesta în tunelul alăturat;
- dacă ajunge în ultimul element unitar al tunelului etichetat cu **N**, atunci Tommy iese din acest tunel cu recompensă, chiar dacă ar exista un pasaj ce conectează acest ultim element la ultimul element din tunelul $\mathbf{N-1}$ (vezi **Figura 2.b**);
- dacă ajunge în ultimul element unitar al tunelului etichetat cu $\mathbf{N-1}$ și există un pasaj care conectează acest element cu ultimul element unitar al tunelului etichetat cu **N**, atunci Tommy trece prin acest pasaj în ultimul element din ultimul tunel, ia recompensa și iese din tunel (vezi **Figura 2.a**). În cazul în care acest pasaj nu există, Tommy iese din tunelul $\mathbf{N-1}$ fără recompensă;

- dacă ajunge în ultimul element unitar al unui tunel cu eticheta strict mai mică decât **N − 1**, atunci Tommy iese din tunel fără recompensă.

Ajutați-l pe Tommy să ajungă cât mai repede la recompensă respectând regulile jocului!

Cerințe

Scrieţi un program care citește numerele naturale **N**, **M** și **X**, iar apoi determină:

1. eticheta tunelului prin care iese Tommy dacă intră prin tunelul cu eticheta **X**, respectând regulile jocului;
2. numărul minim **L** de elemente unitare (ale tunelurilor și ale pasajelor) prin care Tommy ar trebui să treacă, respectând regulile jocului, pentru a ajunge la recompensă.

Date de intrare

Fișierul `tunel.in` conține pe prima linie un număr natural **C** reprezentând cerința din problemă care trebuie rezolvată (**1** sau **2**). A doua linie a fișierului conține cele trei numere naturale **N**, **M** și **X**, separate prin câte un spațiu, cu semnificația din enunț. Următoarele **N − 1** linii descriu pasajele dintre tuneluri. Prima linie dintre cele **N − 1** indică pasajele dintre tunelurile etichetate cu **1** și **2**, următoarea linie indică pasajele dintre tunelurile etichetate cu **2** și **3**, . . . , ultima dintre cele **N − 1** linii indică pasajele dintre tunelurile etichetate cu **N − 1** și **N**. Primul număr din fiecare astfel de linie reprezintă numărul **P** de pasaje, iar următoarele **P** numere distincte, **scrise în ordine crescătoare**, reprezintă pozițiile elementelor unitare (dintre cele două tuneluri) conectate prin cele **P** pasaje.

Date de ieșire

- Dacă **C = 1**, fișierul `tunel.out` va conține pe prima linie un număr natural reprezentând răspunsul la cerința **1**.
- Dacă **C = 2**, fișierul `tunel.out` va conține pe prima linie numărul natural **L**, reprezentând răspunsul la cerința **2**.

Restricții și precizări

- **3 ≤ N ≤ 1 000**
- **4 ≤ M ≤ 20 000**
- **1 ≤ P ≤ M − 2**
- Pot exista cel mult **150 000** pasaje care interconectează tunelurile.
- Pot exista pasaje învecinate care să conecteze elementele unitare din două tuneluri alăturate (vezi **Figura 1** în care tunelurile 1 și 2 sunt interconectate prin pasajele învecinate dintre elementele 6, respectiv 7).
- Primul element unitar din fiecare tunel nu este conectat la niciun pasaj.
- Ultimul element unitar din tunelurile etichetate cu $1, 2, ..., \mathbf{N}-2$ nu este conectat la niciun pasaj.
- Oricare element unitar poate fi conectat la cel mult un pasaj.
- Oricare două tuneluri etichetate cu numere consecutive sunt interconectate prin cel puțin un pasaj.
- Pentru fiecare intrare într-un tunel există traseu către ieșire.
- Pentru fiecare test există cel puțin o intrare într-un tunel prin care Tommy poate ajunge la ieșirea cu recompensă din tunelul **N** .
- Pentru cerința **1** se acordă **40p**. iar pentru cerința **2** se acordă **60p**.

Exemple

1)	tunel.in	tunel.out	**Explicații**
	1 4 9 4 3 2 4 6 2 3 5 3 4 6 9	1	Se rezolvă cerința **1**. $\mathbf{N} = 4$, $\mathbf{M} = 9$, $\mathbf{X} = 4$. Tommy, intra prin tunelul etichetat cu $\mathbf{X} = 4$ și iese prin tunelul etichetat cu 1 (vezi Figura 2.d).
2)	**tunel.in**	**tunel.out**	**Explicații**
	2 4 9 4 3 2 4 6 2 3 5 3 4 6 9	17	Se rezolvă cerința **2**. $\mathbf{N} = 4$, $\mathbf{M} = 9$, $\mathbf{X} = 4$. Figurile 2.a, 2.b, 2.c, 2.d conțin trecerea lui Tommy prin tunele pentru fiecare din cele patru intrări. Doar 2 dintre aceste treceri îl duc pe Tommy la recompensă (vezi Figura 2.a și Figura 2.b). Dacă Tommy intră prin tunelul 2 atunci el ajunge la recompensă trecând prin numărul minim $\mathbf{L} = 17$ $(= \min(17, 19))$ elemente unitare.

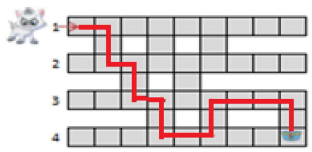


Figura 2.a

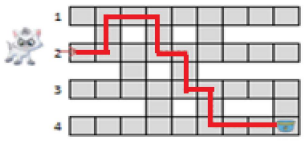


Figura 2.b

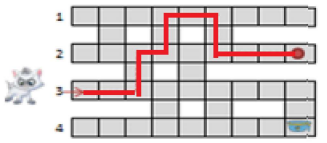


Figura 2.c

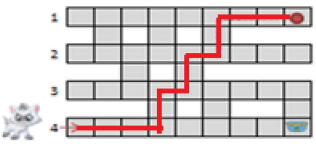


Figura 2.d

Descrierea soluției

Soluție cu punctaje variate

Soluția propusă utilizează o matrice binară A de dimensiune $(2\mathbf{N} - 1) \cdot \mathbf{M}$, în care valorile de 1 din liniile impare reprezintă elementele unitare, iar valorile 1 din liniile pare reprezintă pasajele.

Observații: a) umplerea inutilă cu 1 a liniilor din matrice corespunzătoare tunelurilor va mări timpul de executare, diminuând punctajul; b) simularea deplasării în matrice se poate face din element în element sau din pasaj în pasaj.

Cerința 1 (40 de puncte)

- O soluție cu punctaj maxim (complexitate $\mathcal{O}(\mathbf{M})$) se obține pornind din elementul $A[2\mathbf{X} - 1, 1]$ corespunzător intrării în tunelul $\mathbf{X}$, simulând deplasarea după regulile date.
- Pentru fiecare intrare, traseul lui Tommy spre ieșire este unic.
- Astfel, din $\mathbf{A}[\mathbf{i}, \mathbf{j}]$, Tommy se deplasează în: a) $\mathbf{A}[\mathbf{i} + \mathbf{2}, \mathbf{j} + \mathbf{1}]$ dacă $\mathbf{A}[\mathbf{i} + \mathbf{1}, \mathbf{j}] = \mathbf{1}$ (există pasaj în jos) sau b) $\mathbf{A}[\mathbf{i} - \mathbf{2}, \mathbf{j} + \mathbf{1}]$ dacă $\mathbf{A}[\mathbf{i} - \mathbf{1}, \mathbf{j}] = \mathbf{1}$ (există pasaj în sus) sau c) $\mathbf{A}[\mathbf{i}, \mathbf{j} + \mathbf{1}]$. Doar una din cele 3 situații a), b) și c) poate să apară.
- Exceptând tunelul cu eticheta $\mathbf{N} - \mathbf{1}$ (căruia îi corespunde linia $\mathbf{2N} - \mathbf{3}$ din matrice), deplasarea se finalizează în momentul în care se ajunge în coloana $j = \mathbf{M}$, valoarea indicelui i reprezentând numărul tunelului prin care Tommy iese.

- În cazul în care Tommy ajunge în ultimul element al tunelului cu eticheta $\mathbf{N}$, indiferent de valoarea $A[2\mathbf{N}-2, \mathbf{M}]$ (adică indiferent de existența unui pasaj către tunelul $\mathbf{N}-1$) Tommy ia recompensa si iese prin tunelul $\mathbf{N}$ cu recompensă.
- În cazul în care Tommy ajunge în ultimul element din tunelul $N-1$, dacă $A[2\mathbf{N}-2, \mathbf{M}] = 1$ (adică există pasaj către ultimul tunel) atunci Tommy trece prin pasaj, ia recompensa si iese prin tunelul $\mathbf{N}$. Altfel, va ieși prin tunelul $\mathbf{N}-1$.

Cerința 2 (60 de puncte)

- O soluție cu punctaj partial și complexitate $\mathcal{O}(\mathbf{N} \cdot \mathbf{M})$ se obține folosind algoritmul descris la cerința 1 pentru fiecare dintre cele $\mathbf{N}$ tuneluri. În plus, vom contoriza fiecare element unitar și fiecare pasaj prin care vom trece. Valoarea acestui contor va reprezenta lungimea traseului parcurs. Răspunsul la cerința 2 va fi valoarea minimă a lungimilor traseelor cu ieșire prin tunelul $\mathbf{N}$.
- O soluție cu punctaj maxim se obține simulând deplasarea de la ieșire către intrare pentru cele maxim două trasee posibile. Plecând din $\mathbf{A}[\mathbf{2N}-\mathbf{1}, \mathbf{M}]$, se trece în $\mathbf{A}[\mathbf{2N}-\mathbf{1}, \mathbf{M}-\mathbf{1}]$ sau în $\mathbf{A}[\mathbf{2N}-\mathbf{3}, \mathbf{M}]$ (doar dacă $\mathbf{A}[\mathbf{2N}-\mathbf{2}, \mathbf{M}] = \mathbf{1}$ adică există pasaj între ultimele elemente ale tunelurilor $\mathbf{N}$ și $\mathbf{N}-\mathbf{1}$).
  Pentru fiecare dintre aceste aceste trasee vom proceda astfel: din $\mathbf{A}[\mathbf{i}, \mathbf{j}]$ trecem în a) $\mathbf{A}[\mathbf{i}-\mathbf{2}, \mathbf{j}-\mathbf{1}]$ dacă $\mathbf{A}[\mathbf{i}-\mathbf{1}, \mathbf{j}] = \mathbf{1}$ sau b) $\mathbf{A}[\mathbf{i}+\mathbf{2}, \mathbf{j}-\mathbf{1}]$ dacă $\mathbf{A}[\mathbf{i}+\mathbf{1}, \mathbf{j}] = \mathbf{1}$ sau c) $\mathbf{A}[\mathbf{i}, \mathbf{j}-\mathbf{1}]$. Doar una dintre situațiile a), b) sau c) poate să apară. Astfel, sunt parcurse cel mult două trasee din cele N posibile (complexitate $\mathcal{O}(\mathbf{M})$).

Soluție cu punctaj maxim

Propusă de drd. Diana Ghinea (ETH Zürich) și ș.l. Stelian Ciurea (ULB Sibiu)

Complexitatea soluției, atât pentru cerința 1 cât și pentru cerința 2 este $\mathcal{O}(\mathbf{M})$ cu memorie suplimentară $\mathcal{O}(\mathbf{N} \cdot \mathbf{M})$, sau $\mathcal{O}(\mathbf{M} \log \mathbf{N})$ cu memorie suplimentară $\mathcal{O}(\sum_{\mathbf{i}=\mathbf{1}}^{\mathbf{N}} \mathbf{P_i})$, unde P_i reprezintă numărul de pasaje din tunelul i.

Cerința 1 (40 de puncte)

Soluția se poate obține simulând traseul lui Tommy element cu element de la intrarea dată până la ieșire. Notăm cu (T, E) elementul unitar cu numărul E din tunelul cu eticheta T. Orice traseu conține $\mathcal{O}(\mathbf{M})$ elemente întrucât fiecare element unitar al oricărui tunel este conectat la cel mult un pasaj. Pentru fiecare element $(i, j) \neq (\mathbf{N}, \mathbf{M})$, verificăm dacă există un pasaj spre tunelul $i-1$ sau spre tunelul $i+1$ din elementul (i, j). În caz afirmativ, Tommy va merge prin pasajul corespunzător către elementul $(i-1, j)$, respectiv $(i+1, j)$. Altfel, Tommy va continua să meargă către dreapta: fie în elementul $(i, j+1)$, fie iese din pasaj.

Putem verifica dacă există un pasaj din elementul $(i, j) \neq (\mathbf{N}, \mathbf{M})$ în mai multe moduri:

- Reținem o matrice `bool` P cu $\mathbf{N}$ linii și $\mathbf{M}$ coloane, unde $P[i][j] =$ `true` dacă și numai dacă există un pasaj din celula (i, j) către celula $(i+1, j)$. Astfel, putem verifica dacă există un pasaj în complexitate $\mathcal{O}(\mathbf{1})$, însă cu memorie suplimentară $\mathcal{O}(\mathbf{N} \cdot \mathbf{M})$.
- Utilizăm structura `vector` din C++: pentru fiecare coloană j, `vector`-ul $P[j]$ reține pasajele de pe coloana j (cel mult $\mathcal{O}(\mathbf{N})$ pasaje). Putem verifica dacă există un pasaj folosind căutare binară, în complexitate $\mathcal{O}(\log \mathbf{N})$, cu memorie suplimentară $\mathcal{O}(\sum_{\mathbf{i}=\mathbf{1}}^{\mathbf{N}} \mathbf{P_i})$, unde P_i reprezintă numărul de pasaje din tunelul i.
- Utilizăm structura `set` din C++: pentru fiecare coloană j, `set`-ul $P[j]$ reține pasajele de pe coloana j (cel mult $\mathcal{O}(\mathbf{N})$ pasaje). Putem verifica dacă există un pasaj (pentru

elementul (i, j), dacă i aparține `set`-ului $P[j]$) în complexitate $\mathcal{O}(\log \mathbf{N})$, cu memorie suplimentară $\mathcal{O}(\sum_{\mathbf{i=1}}^{\mathbf{N}} \mathbf{P_i})$, unde P_i reprezintă numărul de pasaje din tunelul i.

Astfel, utilizând prima modalitate, obținem complexitatea $\mathcal{O}(\mathbf{M})$, iar utilizând una dintre celelalte două modalități, obținem complexitatea $\mathcal{O}(\mathbf{M} \log \mathbf{N})$.

Cerința 2 (60 de puncte)

O soluție naivă ar simula traseul lui Tommy pornind din intrarea fiecărui tunel. Fără a lua în calcul timpul pentru verificarea pasajelor, o astfel de soluție ar avea complexitate $\mathcal{O}(\mathbf{N} \cdot \mathbf{M})$.

Pentru a obține o soluție în complexitatea de timp dorită, avem nevoie de următoarea **observație**:

Există cel mult două tuneluri din care Tommy poate ajunge la recomensă.

Orice traseu care ajunge la recompensă (îl vom denumi *traseu cu succes*) fie trece prin elementul $(\mathbf{N}-1, \mathbf{M})$ (dacă există un pasaj către $(\mathbf{N}, \mathbf{M})$), fie trece prin elementul $(\mathbf{N}, \mathbf{M}-1)$.

Presupunem că există cel puțin trei trasee distincte cu succes. Atunci, două dintre aceste trasee trec prin același element $(\mathbf{N}-1, \mathbf{M})$ sau $(\mathbf{N}, \mathbf{M}-1)$. Vom presupune că cele două trasee trec prin elementul $(\mathbf{N}-1, \mathbf{M})$ (celălalt caz este analog).

Vom nota aceste trasee cu:

$A = (a_l = (x, 1), a_{l-1}, \ldots, a_2 = (\mathbf{N}-1, \mathbf{M}), a_1 = (\mathbf{N}, \mathbf{M}))$

$B = (b_{l'} = (x', 1), b_{l'-1}, \ldots, b_2 = (\mathbf{N}-1, \mathbf{M}), b_1 = (\mathbf{N}, \mathbf{M}))$

(aceste trasee nu includ pasajele).

Din moment ce A și B sunt doua trasee distincte, dar ultimele două elementul ale lui A și B sunt comune, există un element $C = a_k = b_k = (i_C, j_C)$ astfel încât $a_i = b_i$ pentru $1 \leq i \leq k$ și $a_{k-1} \neq b_{k-1}$. Acest lucru este posibil doar dacă unul dintre cele două trasee a trecut aici printr-un pasaj. Întrucât un element poate fi conectat la maxim un pasaj, doar unul dintre traseele A și B poate trece aici printr-un pasaj: presupunem ca acest traseu este A și că $a_{k+1} = (i_C - 1, j_C)$ (analog pentru celelalte cazuri). Atunci, $b_{k+1} = (i_C, j_C - 1)$ și, din moment ce aici este un pasaj, $b_{k-1} = (i_C - 1, j_C) \neq (i_C, j_C + 1) = a_{k-1}$, ceea ce contrazice modul în care am ales elementul C și demonstrează observația noastră.

Astfel, nu este necesar să simulăm toate traseele posibile pentru a obține răspunsul la această cerință: trebuie să simulăm cel mult două trasee.

Adăugăm încă o observație:

- *Există un traseu cu succes în care Tommy trece prin elementul* $(\mathbf{N}, \mathbf{M}-1)$.
- *Dacă există un pasaj între* $(\mathbf{N}-1, \mathbf{M})$ *și* $(\mathbf{N}, \mathbf{M})$*, atunci există un traseu cu succes în care Tommy trece prin elementul* $(\mathbf{N}, \mathbf{M}-1)$.

În concluzie, o soluție pentru această cerință va simula *în sens invers* deplasarea lui Tommy în cel mult două trasee (de la recompensă către intrare) și va calcula lungimea acestora:

- unicul traseu cu succes ce trece prin celula $(\mathbf{N}, \mathbf{M}-1)$, dacă acest traseu există (în cazul în care există un pasaj între celulele $(\mathbf{N}, \mathbf{M})$ și $(\mathbf{N}, \mathbf{M}-1)$);
- unicul traseu cu succes ce trece prin celula $(\mathbf{N}, \mathbf{M}-1)$.

Lungimea unui traseu poate fi calculată ca $\mathbf{M} + 2p$, unde p este numărul de pasaje prin care Tommy trece în acel traseu.

Capitolul 5

Clasa a IX-a

5.1 Problema Ascdesc

Propunător: stud. Bogdan-Ioan Popa

Universitatea din București

Sesiunea de antrenament

Enunț

Nu demult RANDy și-a cumpărat un lanț muntos, iar acum acesta dorește să-l modifice astfel încât să-i ofere atât o reședință, cât și un loc pentru distracție. Acest lanț muntos este alcătuit din N vârfuri aranjate într-o linie, al i-ulea vârf având înălțimea $H[i]$. Pentru un vârf cu numărul de ordine k pe care îl va alege ca și loc unde își va construi reședința, acesta este interesat să-și creeze și o tiroliană ce leagă vârful k de vârfurile $k-1, k-2, ..., 2, 1$, și încă o tiroliană ce leagă vârful k de vârfurile $k+1, k+2, ..., N-1, N$. Pentru ca acest lucru să fie posibil, această tiroliană trebuie ori să-și păstreze altitudinea, ori să coboare, atât la stânga palatului cât și la dreapta sa (formal el este interesat ca $H[i] \leq H[i+1]$ oricare ar fi $1 \leq i < k$ și $H[i] \geq H[i+1]$ oricare ar fi $k \leq i < N$). Ca să-și atingă scopul, RANDy va cere ajutorul gigantului Athos. Pentru fiecare lemn ars ca ofrandă de către RANDy, gigantul Athos va crește înălțimea oricărui vârf indicat de către RANDy cu 1. Fiind un activist în salvarea pădurilor planetei, RANDy dorește să ardă cât mai puțin lemn pentru a-și îndeplini scopul.

Ajută-l pe RANDy să afle care este cantitatea minimă de lemn pe care o va arde pentru fiecare poziție k de la 1 la N.

Date de intrare

Pe prima linie din fișierul de intrare se va găsi valoarea N. Pe următoarea linie se vor afla N numere naturale, separate prin câte un spațiu, al i-ulea număr reprezentând înălțimea celui de-al i-ulea vârf.

Date de ieșire

În fișierul de ieșire, pe prima linie, se vor afișa N numere separate prin câte un spațiu, al k-lea număr reprezentând cantitatea minimă de lemn ars necesară construirii reședinței pe vârful k.

Restricții și precizări

- Pentru cazul în care $k = 1$ sau $k = N$, RANDy va face o singură tiroliană, una către dreapta respectiv una către stânga
- $1 \leq N \leq 100.000$
- $1 \leq H[i] \leq 1.000.000.000$
- Subtask 1 (60 puncte)
  - $N \leq 1000$
- Subtask 2 (40 puncte)
  - Fară restricții suplimentare.

Exemplu

	ascdesc.in	ascdesc.out	Explicații
	6 2 3 1 5 4 2	9 6 4 2 3 6	Pentru cazul în care $k = 4$, după modificări lanțul muntos va arăta în felul următor: 2 3 3 5 4 2. Astfel RANDy va ardedoar 2 lemne, crescând înălțimea vârfului cu numărul de ordine 3 cu 2. De aceea, a 4-a valoare din *ascdesc.out* este 2. Pentru cazul în care $k = 1$, după modificări lanțul muntos va arăta în felul următor: 5 5 5 5 4 2. Astfel RANDy va arde 9 lemne pentru a modifica înălțimile vârfurilor $1, 2, 3$. De aceea, prima valoare din fișierul de ieșire este 9.

Descrierea soluției

Soluția 1, complexitate $O(N^2)$:

Pentru un vârf k fixat pe care RANDy își va construi reședința, putem parcurge în ordine numerele de la 2 la k folosind un i și să aplicăm următoarele modificări:

- Dacă $H[i] \geq H[i-1]$, atunci nu vom crește înălțimea vârfului $H[i]$
- Dacă $H[i] < H[i-1]$, atunci vom crește înălțimea vârfului i cu $H[i-1] - H[i]$ unități, astfel încat acesta ajunge la fel de înalt ca și vârful $i-1$.

Asemănător se vor trata și modificările făcute pentru i de la $N-1$ la k.

Soluția 2, complexitate $O(N)$:

Se observă că modificările făcute nu depind de k, decât prin locul unde se opresc. Acest lucru ne conduce la ideea să precalcuăm:

- $cle[i]$ = numărul minim de lemne ce trebuiesc arse pentru ca vârfurile de la 1 la i să abiă valori ascendente.
- $cri[i]$ = numărul minim de lemne ce trebuiesc arse pentru ca vârfurile de la i la N să aibă valori descendente.

Fie $maxH$ = valoarea maximă al unui H[i]. Atunci răspunsul pentru un vârf k este egal cu $cle[k-1] + cri[k+1] + maxH - H[k]$

5.2 Problema Aproape

Propunător: stud. Alexandru Petrescu

University of Oxford, Keble College

Enunț

Se dă un număr N în baza 10. Un număr M se numește aproape de N dacă îndeplinește următoarele trei condiții:

1. Are același număr de cifre cu N.
2. Reprezentarea în baza 10 diferă față de cea a lui N în exact o poziție. Altfel spus, nu mai mult, nici mai puțin, o singură cifră diferă.
3. Această cifră este fie cu 1 mai mică, fie cu 1 mai mare decât cifra corespunzătoare din N.

Exemplu:

Să presupunem că $N = 1903$. Un exemplu de număr M care este aproape de N este 1913, pentru că diferă (doar) cifra zecilor, iar diferența între 0 (cifra zecilor în N) și 1 (cifra zecilor în M) este 1. Numerele 1903, 903 și 1893 nu sunt aproape de N.

Cerință

Cunoscând numărul N, să se scrie un program care determină:

1. Numărul de cifre ale acestui număr.
2. Numărul de numere aproape de N.
3. Numărul de numere aproape de cel puțin un număr aproape de N.

Date de intrare

Fişierul de intrare *aproape.in* conţine pe prima linie un număr V a cărui valoare poate fi doar 1, 2 sau 3, iar pe a două linie numărul natural N.

Date de ieșire

Dacă valoarea lui V este 1, atunci fişierul *aproape.out* va conţine pe prima linie un singur număr ce reprezintă numărul de cifre ale lui N.

Dacă valoarea lui V este 2, atunci fişierul *aproape.out* va conţine pe prima linie un singur număr natural ce reprezintă numărul de numere aproape de N.

Dacă valoarea lui V este 3, atunci fişierul *aproape.out* va conţine pe prima linie un singur număr natural ce reprezintă numărul de numere aproape de un număr aproape de N.

Restricții și precizări

- $0 \leq N < 1.000.000.000$
- Pentru teste în valoare de 20 de puncte avem $V = 1$.
- Pentru teste în valoare de 30 de puncte avem $V = 2$.
- Pentru teste în valoare de 50 de puncte avem $V = 3$.
- Aveți grijă să nu numărați vreun număr de mai multe ori!

Exemple

1)	aproape.in	aproape.out	**Explicații**
	1 1903	4	Cele 15 numere sunt: 1703, 1802, 1804, 1813, 1901, 1903, 1905, 1912, 1914, 1923, 2803, 2902, 2904, 2913 și 3903.
2)	aproape.in	aproape.out	
	2 1903	5	.
2)	aproape.in	aproape.out	
	3 1903	15	

Descrierea soluției

Soluție parțială – 70 de puncte

Este suficient să iterăm prin toate numerele care au același număr de cifre ca numărul N și să verificăm dacă respectă proprietățile unui număr aproape de N, respectiv unui număr aproape de cel puțin un număr aproape de N.

Soluție oficială – 100 de puncte

Vom începe prin a defini urmatoarele proprietăți ale numarului N:

- cnt_1 - numărul de cifre care pot fi fie doar incrementate cu 1, fie doar decrementate cu 1.
- cnt_2 - numărul de cifre care pot fi și incrementate și decrementate cu 1.
- $aproape_2$ - numărul de moduri în care putem modifica numărul N prin incrementarea sau decrementarea unei singure cifre cu 2.

Folosind aceste 3 rezultate, se pot rezolva toate cele 3 cerințe, confrom următoarelor formule.

Cerința 1

Deoarece o cifră este fie numarata de cnt_1 fie de cnt_2, numărul de cifre ale numărului N este:

$$cif_1 + cif_2$$

Cerința 2

Deoarece o cifră numarata de cnt_1 poate fi modificată într-un singur mod, iar o cifră numarată de cnt_2 poate fi modificată în 2 moduri, numărul de numere aproape de N este:

$$cif_1 + 2 \cdot cif_2$$

Cerința 3

Pentru a calcula cât mai ușor răspunsul pentru cerința 3 vom împărți numerele de numere aproape de numărul N în 3 categori:

- Numerele care nu diferă prin nicio cifră față de numarul N. Singurul număr din această categorie este chiar N.
- Numerele care diferă prin exact o cifră față de numărul N. Există $aproape_2$ numere de acest fel, deoarece dacă modificăm o cifră de 2 ori și nu ne întroarcem tot la numărul N atunci am decrementat cu 1 aceeași cifră a numărului de 2 ori.
- Numerele care diferă prin exact două cifre față de numărul N. În această categorie există 3 cazuri de perechi de cifre:
  - Ambele cifre sunt numărate de cnt_1. În acest caz există o singură modalitate de a modifica numărul N și există $\frac{cnt_1 \cdot (cnt_1 - 1)}{2}$ astfel de perechi.
  - O cifră este numarată de cnt_1, iar cealaltă este numarată de cnt_2. În acest caz există 2 modalități de a modifica numărul N și există $cnt_1 \cdot cnt_2$ astfel de perechi.
  - Ambele cifre sunt numărate de cnt_1. În acest caz există 4 modalități de a modifica numărul N și există $\frac{cnt_2 \cdot (cnt_2 - 1)}{2}$ astfel de perechi.

Adunand numărul de numere din fiecare categorie, numărul de numere aproape de cel puțin un număr aproape de N este:

$$1 + aproape_2 + \frac{cnt_1 \cdot (cnt_1 - 1)}{2} + 2 \cdot cnt_1 \cdot cnt_2 + 4 \cdot \frac{cnt_2 \cdot (cnt_2 - 1)}{2}$$

5.3 Problema Cochilie

Propunător: Szabó Zoltan

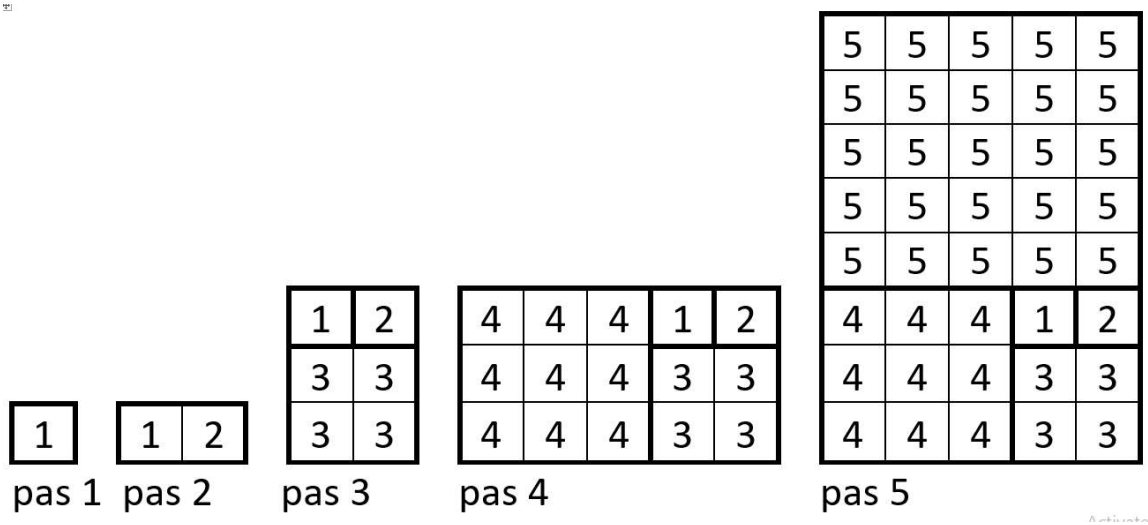
Liceul Tehnologic "Petru Maior" Reghin/ ISJ Mureș

Enunț

O matrice se numește *cochilie de ordin* N, sau mai simplu *cochilie*, dacă a fost construită în funcție de un număr natural N nenul după următoarea regulă:

- Cochilia este formată inițial dintr-un pătrat de latură 1 cu valoarea 1.
- Pentru fiecare pas I cu valorile $2, 3, ..., N$ la cochilia deja existentă, se va alătura pe rând la DREAPTA, JOS, STÂNGA, SUS, în mod repetat în această ordine, câte un pătrat în care toate elementele au valoarea I, iar lungimea laturii pătratului nou corespunde cu latura cochiliei la care se lipește.

O cochilie de ordin 5 se formează în 5 pași astfel:



Liniile și coloanele sunt numerotate de sus în jos și de la stânga la dreapta începând cu valoarea 1.

Cerință

Cunoscând valorile numerelor naturale N și P, va trebui să răspundeți la următoarele întrebări:

- Ce dimensiuni are cochilia de ordin N?
- Ce elemente se află pe linia P a cochiliei de ordin N?

Date de intrare

Pe prima linie din fișierul de intrare *cochilie.in* se va găsi valoarea C, care poate să aibă una dintre valorile 1 sau 2.

Dacă valoarea lui C este 1, atunci pe linia următoare se va găsi valoarea lui N.

Dacă valoarea lui C este 2, atunci pe linia următoare se vor găsi valorile lui N și P separate printr-un spațiu.

Date de ieșire

Datele de ieșire se vor afișa pe prima linie a fișierului de ieșire *cochilie.out* în funcție de valoarea lui C astfel:

Dacă valoarea lui C este 1, atunci se vor afișa $NRLIN$ și $NRCOL$ separate printr-un spațiu, reprezentând numărul de linii, respectiv numărul de coloane ale cochiliei de ordin N.

Dacă valoarea lui C este 2, atunci se vor afișa elementele de pe linia P ale cochiliei de ordin N, separate prin câte un spațiu.

Restricții și precizări

- $1 < N < 30$
- Linia P întotdeauna se referă la o linie validă a cochiliei.
- Pentru teste în valoare de **8 puncte** avem $C = 1$.
- Pentru alte teste în valoare de **36 de puncte** avem $C = 2$ și $N \leq 17$.
- Pentru alte teste în valoare de **20 de puncte** avem $C = 2$ și P se referă la prima sau ultima linie a cochiliei.
- Pentru alte teste în valoare de **36 de puncte** avem $C = 2$ fără alte restricții.

Exemple

1)	`cochilie.in`	`cochilie.out`	**Explicații**
	`1` `5`	`8 5`	Cochilia de ordin 5 are 8 linii și 5 coloane.

2)	`cochilie.in`	`cochilie.out`	**Explicații**
	`2` `5 6`	`4 4 4 1 2`	Linia a 6-a a cochiliei de ordin 5 este formată din numerele 4 4 4 1 2.

Descrierea soluției

Observăm că dimensiunile cochiliei reprezintă numere Fibonacci, valoarea lor fiind în functie de paritatea lui N.

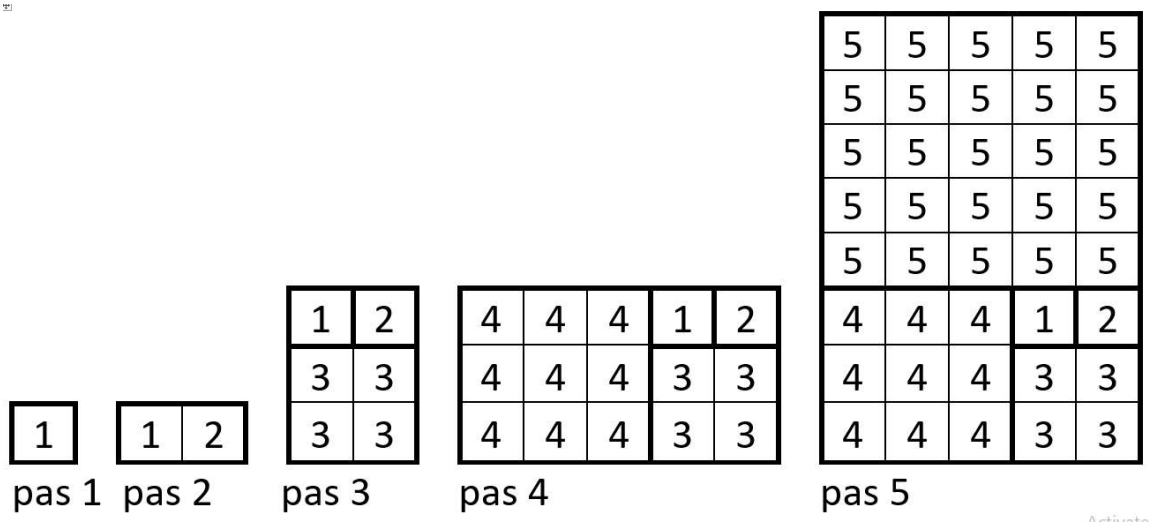
Pentru cerința 1:

Dimensiunea cochiliei în funcție de N:

1. dacă N este impar, avem $FIB(N+1)$ linii și $FIB(N)$ coloane
2. dacă N este par, avem $FIB(N)$ linii și $FIB(N+1)$ coloane

Pentru cerința 2:

Observăm, că multe linii ale cochiliei vor fi identice. Întrucât pătratele adăugate, în funcție de N, pot avea pasul final SUS, JOS, STANGA sau DREAPTA, de aceea, trebuie să studiem 4 cazuri distincte în funcție de N modulo 4.



Observăm, că linia cu construcția cea mai simplă va conține una sau două valori:

1. Dacă $N = 4k + 1$,
   prima linie (primele $FIB(N)$ linii) va/vor conține o singură valoare N de $FIB(N)$ ori;
2. Dacă $N = 4k + 2$,
   prima linie (primele $FIB(N-1)$ linii) va/vor conține două valori, și anume $N-1$ de $FIB(N-1)$ ori, urmate de N de $FIB(N)$ ori;
3. Dacă $N = 4k + 3$,
   ultima linie (ultimele $FIB(N)$ linii) va/vor conține o singură valoare N de $FIB(N)$ ori;
4. Dacă $N = 4k$,
   ultima linie (ultimele $FIB(N-1)$ linii) va/vor conține două valori, și anume N de $FIB(N)$ ori, urmate de $N-1$ de $FIB(N-1)$ ori;

Avem următoarele configurații distincte de linii:

- Pentru N par avem $N/2$ configurații distincte
- Pentru N impar avem $(N+1)/2$ configurații distincte

De exemplu, pentru $N = 5$ avem $(5+1)/2 = 3$ configurații distincte de linii, și anume:

5 5 5 5 5 (de 5 ori) (de $FIB(5)$ ori)

4 4 4 1 2 (o dată) (de $FIB(1)$ ori)

4 4 4 3 3 (de 2 ori) (de $FIB(3)$ ori)

Observăm, că pentru diferitele valori ale lui N modulo 4, avem aceeași regulă, dar cu valori diferite de pornire. În funcție de cele 4 cazuri menționate mai sus, putem porni de prima sau ultima linie, cu 1 sau două valori numere naturale.

Vom construi alternativ câte o linie de sus respectiv de jos, până ce realizăm configuratia tuturor liniilor.

nr	nr aparitii linie	valori
1	$FIB(9)$	9
2	$FIB(5)$	8,5,6
3	$FIB(1)$	8,4,1,2,6
4	$FIB(3)$	8,4,3,6
5	$FIB(7)$	8, 7

Pentru cazul $N = 4k + 1$, vom avea L=(N+1)/2 linii. Modul de construire este următorul:

- Construim un tablou pentru numărul de apariții, cu valoarea de pornire NRAP(1)=N, și un tablou bidimensional V pentru valori, cu valoarea de pornire V(1)=[N].

- Construim, în această ordine, liniile:

- ultima linie: NRAP(L)=N-2, V(L)=[N-1,N-2].
- linia 2: NRAP(2)=N-4, V(2)=[N-1,N-4,N-3].
- penultima linie: NRAP(L-1)=N-6, V(L)=[N-1,N-5,N-6,N-3].
- linia 4: NRAP(2)=N-8, V(3)=[N-1,N-5,N-8,N-7,N-3].
  ș.a.m.d...

Algoritmul continuă, până când toate liniile se vor construi.

Regula de construcție a elementelor este următoarea: Pentru numărul de apariții Descrește valoarea anterioară cu 2 de la un pas la altul. Pentru a construi elementele șirului de valori V[p] în functie de linia precedenta, ne folosim de formula recursivă a șirului lui Fibonacci, F(N)=F(N-1)+F(N-1). Având valorile șirului curent, depistăm cea mai mică valoare K din șir și o vom înlocui cu valorile K-1, K-2, scriindu-le în ordine crescătoare, dacă ne aflăm pe primele L/2 linii, și in ordine descrescătoare dacă ne aflăm pe ultimele L/2 linii.

Fiecare element x luat de FIB(x) ori, va reprezenta o linie a cochiliei.

Pentru a calcula continutul liniei cu nr de ordine K, vom scadea pe rand numarul de linii FIB(N), FIB(n-4), ..., până când se va ajunge la linia dorita.

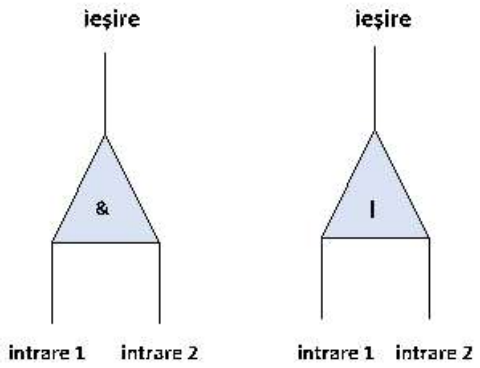
Se vor prelucra cu atentie toate cele 4 cazuri al lui N ($4k$, $4k + 1$, $4k + 2$, $4k + 3$)

5.4 Problema Logic

Propunător: stud. Alin Burța
Colegiul Național "B.P. Hasdeu", Buzău

Enunț

Costel este pasionat de circuitele logice. El are la dispoziție două tipuri de circuite logice simple: circuit ȘI, respectiv circuit SAU. Circuitele logice simple au două intrări și o ieșire.

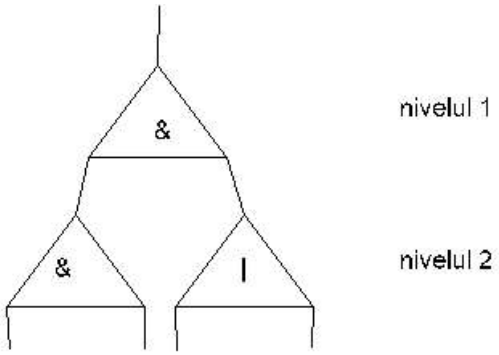


La fiecare intrare în circuit se poate introduce un bit 0 sau un bit 1, iar circuitul este capabil să calculeze operația logică respectivă (ȘI ori SAU) și să trimită rezultatul obținut la ieșire. Costel a învățat că poate combina mai multe circuite simple pentru a obține circuite complexe astfel: leagă ieșirea unui circuit de orice tip la una din intrările altui circuit, deci rezultatul obținut la ieșirea dintr-un circuit se transmite la intrarea celuilalt. In acest fel se pot construi circuite complexe, care au mai multe intrări și o singură ieșire.

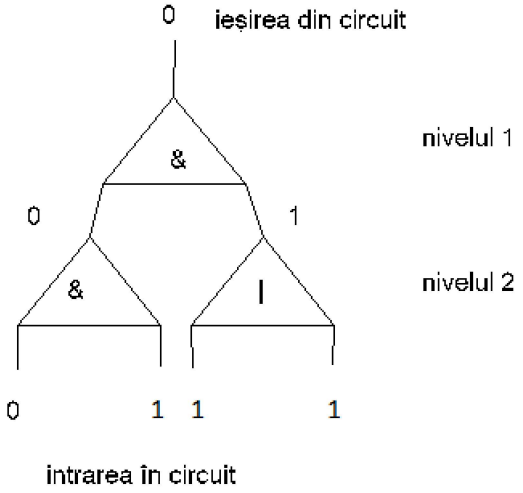
Ultima descoperire a lui Costel este circuitul logic piramidal (prescurtat CLP), care are structura următoare:

- Circuitul cu un singur nivel este cel mai simplu tip de circuit și este compus dintr-un circuit ȘI ori dintr-un circuit SAU;
- Pentru un circuit cu mai multe nivele avem:
  - pe nivelul 1 se găsește un singur circuit (ȘI ori SAU);
  - pe nivelul 2 se găsesc două circuite simple de oricare tip; ieșirea primului circuit este conectată la intrarea 1 a circuitului de pe nivelul 1, iar ieșirea celui de-al doilea circuit este conectată la intrarea 2 a circuitului de pe nivelul 1;
  - pe nivelul N sunt 2^{N-1} circuite simple; ieșirile primelor două circuite de pe linia N sunt conectate la intrările primului circuit de pe nivelul $N-1$, ieșirile următoarelor două sunt conectate la intrările celui de-al doilea circuit de pe linia $N-1$ etc;

Exemplu de CLP cu 2 nivele:



Intr-un CLP cu N nivele avem 2^N intrări, corespunzătoare circuitelor de pe nivelul N. La fiecare intrare se poate introduce un bit 0 sau un bit 1, deci un șir de 2^N biți.



Pentru circuitul din figura de mai sus presupunem că la cele patru intrări ale circuitelor de pe nivelul 2 avem, în ordine, biții 0111. La ieșirea din circuit (ieșirea circuitului simplu de pe primul nivel) se obține valoarea 0, deoarece acest circuit este echivalent cu expresia logică ((0 și 1) și (1 sau 1)).

Cerința 1 (30 de puncte)

Pentru un CLP dat, cu N nivele și pentru K șiruri de biți date la intrarea circuitului, să se determine, pentru fiecare șir, valoarea calculată la ieșirea din circuit.

Cerința 2 (70 de puncte)

Pentru un CLP dat, cu N nivele și cunoscând valoarea obținută la ieșire (0 sau 1), să se determine numărul șirurilor de biți distincte ce pot fi date la intrare pentru a se obține valoarea specificată la ieșire. Rezultatul poate fi un număr foarte mare, de aceea el se va afișa **modulo 666013**.

Date de intrare

Pe prima linie a fișierului *logic.in* se găsește un număr natural C ($C = 1$ pentru cerința 1, respectiv $C = 2$ pentru cerința 2)

Pe a doua linie se găsește numărul natural N, reprezentând numărul de nivele ale circuitului;

Pe următoarele N linii (linii de la 3 la $N + 2$) se găsește descrierea circuitului, **fără spații între caractere**, astfel:

- pe linia 3 un caracter '&' sau $'|'$, unde prin caracterul '&' se codifică un circuit **ȘI**, iar prin caracterul $'|'$ se codifică un circuit **SAU** ;
- pe linia 4 două caractere din mulțimea $\{'\&','|'\}$;
- pe linia 5 patru caractere din mulțimea $\{'\&','|'\}$
- ...
- pe linia $N + 2$ avem 2^{N-1} caractere din mulțimea $\{'\&','|'\}$

Pentru cerința 1:

- Pe linia $N + 3$ avem un număr natural K, reprezentând numărul șirurilor de biți date la intrarea în circuit;
- Pe fiecare dintre următoarele K linii avem câte un șir compus din 2^N biți (caractere 0 sau 1), reprezentând șirul de biți dat la intrare;

Pentru cerința 2:

- Pe linia $N+3$ avem un număr natural din mulțimea $\{0, 1\}$, reprezentând valoarea pe care circuitul trebuie să o scoată la ieșire.

Date de ieșire

- Pentru cerința 1 se vor afișa, în fișierul *logic.out*, pe linii separate, K numere naturale din mulțimea $\{0, 1\}$, cu semnificația din enunț;
- Pentru cerința 2 se va afișa, în fișierul *logic.out*, un număr natural cu semnificația din enunț.

Restricții și precizări

- $1 \leq N \leq 8$
- $1 \leq K \leq 10$
- Tabelele operațiilor logice sunt:

x	y	x SAU y
0	0	0
0	1	1
1	0	1
1	1	1

x	y	x ȘI y
0	0	0
0	1	0
1	0	0
1	1	1

Exemple

1)	logic.in	logic.out	**Explicații**
	1 **2** & &| 3 1101 0100 1000	1 0 0	$C = 1$, deci rezovăm cerința 1. Pentru șirul de biți 1101 valoarea obținută la ieșirea din circuit este rezultatul evaluării expresiei: ((1 și 1) și (1 sau 0)) = (1 și 1) = 1 Pentru șirul de biți 0100 valoarea obținută la ieșirea din circuit este rezultatul evaluării expresiei: ((0 și 1) și (0 sau 0)) = (0 și 0) = 0 Pentru șirul de biți 1000 valoarea obținută la ieșirea din circuit este rezultatul evaluării expresiei: ((1 și 0) și (0 sau 0) = (0 și 0) = 0
2)	logic.in	logic.out	**Explicații**
	2 **2** & &| 1	3	$C = 2$, deci rezovăm cerința 2. Sunt 3 șiruri de 4 biți pentru care se obține valoarea 1 la ieșirea din circuit: $1101, 1110, 1111$.

Descrierea soluției

Cerința 1, complexitate $O(2^N)$:

Simulăm comportamentul circuitului în functie de șirul de biți aflat la intrare (avem un șir cu 2^N biți).

Pornim de pe linia N a circuitului:

- primii doi biți reprezintă cele două intrări ale primului circuit de pe linia N, următorii doi biți reprezintă cele două intrări ale celui de-al doilea circuit de pe linia N etc;
- pentru fiecare circuit de pe linia N calculăm valoarea obținută prin aplicarea operației SI ori SAU asupra biților de la intrare, iar în final vom avea un șir de 2^{N-1} biți. Sirul de biți obținut reprezintă intrarea în circuitele de pe linia $N - 1$.

Aplicăm procedeul descris până ce ajungem pe linia 1, unde se găsește un singur circuit, iar șirul de biți de la intrarea în acesta are lungimea 2.

Cerința 2 - soluția 1, complexitate $O(2^N)$:

Fie: $T[i][j][k]$ = numărul de intrări pentru care la ieșirea din circuitul de pe linia i și coloana j se obține rezultatul k, unde:

$(k = 0, 1)$

$i = 1..N$

$j = 1..2^{N-1}$

- Dacă $i = N$ (ultimul nivel)
  - Dacă circuitul (N,j) este SI atunci:

  $T[N][j][1] = 1$ (e o singura combinație: 11)

$T[N][j][0] = 3$ (sunt 3 combinații: $00, 01, 11$)

- Daca circuitul (N, j) este SAU atunci:

$T[N][j][1] = 3$ (sunt 3 combinații: $01, 10, 11$)

$T[N][j][0] = 1$ (e o singura combinație: 00)

- Daca $i \leq N-1$ știm că, pentru oricare circuit aflat pe linia i și coloana j, intrările acestuia sunt ieșirile circuitelor de pe linia $i+1$ și coloana $2j-1$, respectiv linia $i+1$ și coloana $2j$.

  - Cazul 1 : (i, j) este circuit SI:
    $T[i][j][1] = T[i+1][2j-1][1] * T[i+1][2j][1]$
    (obțin 1 la ieșire doar dacă la intrare am 1 si 1)
    $T[i][j][0] = T[i+1][2j-1][0] * T[i+1][2j][0] + T[i+1][2j-1][0] * T[i+1][2j][1] + T[i+1][2j-1][1] * T[i+1][2j][0]$ adică:
    = în câte moduri obțin 0 în circuitul $(i+1, 2j-1)*$
    în câte moduri obțin 0 în circuitul $(i+1, 2j)$
    + în câte moduri obțin 0 în circuitul $(i+1, 2j-1)*$
    în câte moduri obțin 1 în circuitul $(i+1, 2j)$
    + în câte moduri obțin 1 în circuitul $(i+1, 2j-1)*$
    în câte moduri obțin 0 în circuitul $(i+1, 2j)$
  - Cazul 2 : (i, j) este circuit SAU:
    T[i][j][0] = T[i+1][2j-1][0] * T[i+1][2j][0]
    (obtin 0 la ieșire doar daca la intrare am 0 si 0)
    $T[i][j][1] = T[i+1][2j-1][1] * T[i+1][2j][1] + T[i+1][2j-1][0] * T[i+1][2j][1] + T[i+1][2j-1][1] * T[i+1][2j][0]$ adică:
    = în câte moduri obțin 1 în circuitul $(i+1, 2j-1)*$
    în câte moduri obțin 1 în circuitul $(i+1, 2j)$
    + în câte moduri obțin 0 în circuitul $(i+1, 2j-1)*$
    în câte moduri obțin 1 în circuitul $(i+1, 2j)$
    + în câte moduri obțin 1 în circuitul $(i+1, 2j-1)*$
    în câte moduri obțin 0 în circuitul $(i+1, 2j)$

  Rezultatul se obține în $T[1][1][k], k = 0$ sau 1, după caz.

Cerința 2 - soluția 2, complexitate $O(2^{2^N})$:

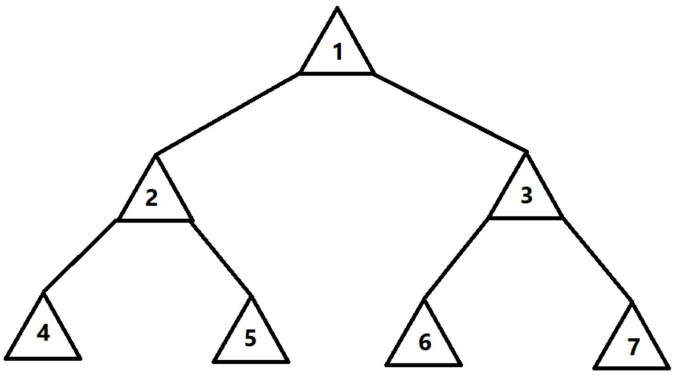
Soluția aceasta obține doar 11 puncte din cele 70.

- Generăm printr-un procedeu oarecare, pe rând, toate șirurile de 2^N biți, care ar putea fi introduse la intrarea în circuit.
- Pentru fiecare șir de biți calculăm ce valoare se va obține la ieșire, folosind algoritmul descris la cerința 1 și contorizăm ca soluție dacă valoarea obținută este valoarea cerută.

Metoda va funcționa doar pe circuite cu nivele puține, $N \leq 4$.

Soluție alternativă propusă de stud. Bogdan-Ioan Popa, Universitatea din București

Vom numerota circuitele logice simple cu numerele $1, 2, 3, ...$ de la stânga la dreapta, de sus în jos, așa cum se vede în exemplul de mai jos:



Se observă că pentru un circuit logic simplu numerotat cu i, el este legat cu circuitele $2 * i$ și $2 * i + 1$, situate pe nivelul următor. Se definește subcircuitul i ca fiind circuitul format din circuitul logic simplu numerotat cu i reunit cu subcircuitele $2 * i$ și $2 * i + 1$. Fie $op[i]$ = operația asociată circuitului logic simplu numerotat cu i. Fie L = numărul total de circuite logice simple din CLP.

Cerința 1, complexitate $O(2^N)$:

Fie S un șir binar de lungime 2^N pentru care vom dori să calculăm valoarea la ieșirea din circuit. Fie $E[i]$ = valoarea evaluată la ieșirea din subcircuitul i. Vom face următoarea inițializare $E[L + i] = S[i]$ pentru fiecare i de la 1 la 2^N. Mai departe, pentru fiecare i de la L la 1 vom calcula $E[i]$:

- Dacă $op[i] = \&$, atunci $E[i] = E[2 * i] \& E[2 * i + 1]$
- Dacă $op[i] = |$, atunci $E[i] = E[2 * i] | E[2 * i + 1]$

Răspunsul se găsește în $E[1]$.

Cerința 2, complexitate $O(2^N)$:

Fie $cnt[i][res]$ = numărul de intrări de lungime corespunzătoare subcircuitului i, care evaluate de către subcircuitul i se obține rezultatul res (res poate fi doar 0 sau 1). Pentru fiecare i de la 1 la 2^N se inițializează $cnt[L + i][0] = cnt[L + i][1] = 1$ (Putem să ne imaginăm ca fiecare subcircuit fictiv $L + i$ este de fapt un singur bit). Mai departe, pentru fiecare i de la L la 1 vom calcula $cnt[i][res]$. Pentru fiecare le între 0 și 1 și fiecare ri între 0 și 1 vom face prelucrările de mai jos:

- Dacă $op[i] = \&$, atunci $cnt[i][le \& ri]$ se adună cu $cnt[2 * i][le] * cnt[2 * i + 1][ri]$.
- Dacă $op[i] = |$, atunci $cnt[i][le | ri]$ se adună cu $cnt[2 * i][le] * cnt[2 * i + 1][ri]$.

Expresia $cnt[2 * i][le] * cnt[2 * i + 1][ri]$ reiese din următorul fapt: pentru fiecare intrare oferită subcircuitului $2 * i$ ce întoarce rezultatul le ($cnt[2 * i][le]$ astfel de intrări) avem $cnt[2 * i + 1][ri]$ intrări oferite subcircuitului $2 * i + 1$ pentru care acesta întoarce ri.

Nu uităm ca la fiecare pas să facem operațiile modulo 666013. Răspunsul se găsește în $cnt[1][res]$, unde res este 0 sau 1, după caz.

Capitolul 6

Clasa a X-a

6.1 Problema Matrice

Propunător: prof. Mihai Bunget
Colegiul Naţional "Tudor Vladimirescu" Târgu Jiu

Sesiunea de antrenament

Enunț

Se dă o matrice cu N linii şi N coloane, ale cărei elemente sunt numere naturale. Numim submatrice pătratică diagonală o submatrice a matricei date, cu număr egal de linii şi coloane, astfel încât cel puţin un element dintr-un colţ al submatricei să coincidă cu un element dintr-un colţ al matricei date. Un element din matricea dată are proprietatea P dacă există cel puţin o submatrice pătratică diagonală astfel încât suma elementelor acestei submatrice să fie egală cu valoarea elementului respectiv.

Cerință

Cunoscând dimensiunea N a matricei date şi elementele matricei, aflaţi câte elemente din matrice au proprietatea P.

Date de intrare

Fişierul de intrare `matrice.in` conţine pe prima linie numărul N, iar pe următoarele N linii câte N numere naturale, reprezentând elementele matricei, separate prin spaţiu.

Date de ieșire

Fişierul de ieşire `matrice.out` va conține pe prima linie numărul elementelor din matricea dată, care au proprietatea P.

Restricții

- $1 \leq N \leq 400$.
- Elementele matricei sunt numere naturale mai mici decât 10^9.

- Pentru teste în valoare de 40 de puncte, $1 \leq N \leq 20$ si elementele matricei sunt numere naturale mai mici decât 10^3.
- Pentru alte teste în valoare de 30 de puncte, $21 \leq N \leq 100$ si elementele matricei sunt numere naturale mai mici decât 10^6.
- Pentru alte teste în valoare de 15 de puncte, $101 \leq N \leq 400$ si elementele matricei sunt numere naturale mai mici decât 10^6.

Exemple

1)	matrice.in	matrice.out	**Explicații**
	3 1 4 8 0 2 5 19 7 6	5	Submatricele pătratice diagonale sunt:(1) (6) (8) (19) $\begin{pmatrix}1&4\\0&2\end{pmatrix}$ $\begin{pmatrix}4&8\\2&5\end{pmatrix}$ $\begin{pmatrix}0&2\\19&7\end{pmatrix}$ $\begin{pmatrix}2&5\\7&6\end{pmatrix}$ $\begin{pmatrix}1&4&8\\0&2&5\\19&7&6\end{pmatrix}$ Suma elementelor acestor submatrice este: $1, 6, 8, 19, 7, 19, 28, 20$, respectiv 52. Elementele matricei date care sunt egale cu cel puţin una dintre aceste sume sunt: $1, 8, 19, 7, 6$. Deci numărul acestor elemente este 5.
2)	matrice.in	matrice.out	
	4 1 1 1 1 1 4 1 1 1 4 5 1 1 1 1 1	15	

Descrierea soluției

Cunoştinţe necesare

- sume parţiale în matrice
- căutare binară
- vector de apariţii

Prezentare soluţie

O soluţie ar fi să calculăm pentru fiecare colţ al matricei date, suma elementelor fiecărei submatrice pătratică diagonală ce are un element în acel colţ, şi să reţinem aceste sume într-un vector. Apoi, pentru fiecare element al matricei, să verificăm prin căutare secvenţială dacă se află în acest vector de sume.(complexitate $O(N^3)$)

Dacă precalculăm sumele parţiale din matricea dată în complexitate $O(N^2)$, putem calcula ulterior sumele tuturor submatricelor pătratice diagonale în complexitate $O(N)$. Sortând şirul acestor sume, pentru fiecare element al matricei date putem afla prin căutare binară dacă este egal cu una dintre sume în complexitate $O(N^2 logN)$.

Dacă elementele matricei date sunt mai mici decât 10^6, atunci căutarea binară poate fi înlocuită cu un vector de apariţii pentru sumele submatricelor pătratice diagonale, complexitatea reducându-se la $O(N^2)$.

6.2 Problema Labirint

prof. Gheorghe Manolache, stud. Ștefan-Cosmin Dăscălescu
Colegiul Național de Informatică Piatra Neamț, Universitatea din București

Enunț

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

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

Cerință

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

Date de intrare

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

Date de ieșire

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

Restricții

- $1 \leq N \leq 1000$.
- $1 \leq M \leq 1000$.
- Pe pozițiile $(1, 1)$ și (N, M) se vor afla valori 0.
- Se garantează că există un drum în matricea inițială între pozițiile $(1, 1)$ și (N, M).
- Pentru teste în valoare de 10 puncte, $1 \leq N, M \leq 50$ si $d_0 = N + M - 1$.
- Pentru alte teste în valoare de 30 de puncte, $1 \leq N, M \leq 50$.

Exemplu

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

Descrierea soluției

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

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

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

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

Complexitatea algoritmului pentru soluția ce obține 100 de puncte devine astfel $O(NM)$.

6.3 Problema SDistanțe

STUD. BOGDAN-PETRU POP
UNIVERSITATEA "BABEȘ-BOLYAI" CLUJ NAPOCA

Enunț

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

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

Definim o *subsecvență* a unui șir s de caractere ca fiind un șir format din caractere de pe poziții consecutive din s. Considerăm două subsecvențe ca fiind distincte dacă încep sau se termină la poziții diferite. Vom nota cu $s(i, j)$ subsecvența formată din caracterele indexate de la i la j ale șirului s. Șirurile se indexează de la 0. Exemplu: pentru șirul $s =$"`abc`" subsecvențele sunt $s(0, 0) =$ "`a`", $s(1, 1) =$ "`b`", $s(2, 2) =$ "`c`", $s(0, 1) =$ "`ab`", $s(1, 2) =$"`bc`", $s(0, 2) =$"`abc`", iar pentru șirul $s =$ "`aa`" acestea sunt $s(0, 0) =$"`a`", $s(1, 1) =$"`a`", $s(0, 1) =$ "`aa`".

Se dă un șir de caractere s, care poate conține doar litere mici și mari ale alfabetului englez (de la '`a`' la '`z`' și de la '`A`' la '`Z`'). Pentru toate perechile neordonate de subsecvențe distincte ale șirului s care au lungimi egale, vrem să calculăm distanța dintre ele și să afișăm suma acestora modulo $10^9 + 7$. Formal, se cere suma valorilor $dist(s(a, b), s(c, d))$, pentru toți indicii a, b, c, d cu $0 \leq a, b, c, d < |s|$, $a < c$, $a \leq b$, $c \leq d$, $b - a = d - c$, **modulo $10^9 + 7$**. $|s|$ reprezintă lungimea șirului s, care este indexat de la 0.

Date de intrare

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

Date de ieșire

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

Restricții

- $|s| \leq 4.000.000$, unde $|s|$ este lungimea șirului s
- Pentru teste în valoare de 11 puncte, $|s| \leq 20$ și s conține doar litere mici.
- Pentru alte teste în valoare de 5 puncte, $|s| \leq 50$ și s conține doar caracterele '`a`' și '`b`'
- Pentru alte teste în valoare de 15 puncte, $|s| \leq 350$ și s conține doar litere mici.
- Pentru alte teste în valoare de 6 puncte, $|s| \leq 1000$ și s conține doar caracterele '`a`' și '`b`'.
- Pentru alte teste în valoare de 30 de puncte, $|s| \leq 5.000$ și s conține doar litere mici.

- Pentru alte teste în valoare de 5 puncte, $|s| \leq 100.000$ și s conține doar caracterele 'a' și 'b'.
- Pentru alte teste în valoare de 4 puncte, $|s| \leq 100.000$ și s conține doar litere mici.
- Pentru alte teste în valoare de 6 puncte, $|s| \leq 1.000.000$ și s conține doar caracterele 'a' și 'b'.

Exemplu

1)	sdistante.in	sdistante.out	**Explicații**
	abc	5	$dist(s(0,0), s(1,1)) = dist(\text{“a”}, \text{“b”}) = 1$ $dist(s(0,0), s(2,2)) = dist(\text{“a”}, \text{“c”}) = 1$ $dist(s(1,1), s(2,2)) = dist(\text{“b”}, \text{“c”}) = 1$ $dist(s(0,1), s(1,2)) = dist(\text{“ab”}, \text{“bc”}) = 2$
2)	sdistante.in	sdistante.out	**Explicații**
	aab	3	$dist(s(0,0), s(1,1)) = dist(\text{“a”}, \text{“a”}) = 0$ $dist(s(0,0), s(2,2)) = dist(\text{“a”}, \text{“b”}) = 1$ $dist(s(1,1), s(2,2)) = dist(\text{“a”}, \text{“b”}) = 1$ $dist(s(0,1), s(1,2)) = dist(\text{“aa”}, \text{“ab”}) = 1$
3)	sdistante.in	sdistante.out	**Explicații**
	ABa	5	$dist(s(0,0), s(1,1)) = dist(\text{“A”}, \text{“B”}) = 1$ $dist(s(0,0), s(2,2)) = dist(\text{“A”}, \text{“a”}) = 1$ $dist(s(1,1), s(2,2)) = dist(\text{“B”}, \text{“a”}) = 1$ $dist(s(0,1), s(1,2)) = dist(\text{“AB”}, \text{“Ba”}) = 2$
4)	sdistante.in	sdistante.out	
	aaaaabbbaaaa	480	
5)	sdistante.in	sdistante.out	
	abcdefghizabcdefghiz	7095	

Descrierea soluției

Vom nota cu N lungimea șirului s.

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

Algorithm 1 Soluție brute-force

```
N ← lungime(s)
raspuns ← 0
for i ← 0...N − 1 do
    for j ← i + 1..N − 1 do
        for lungime ← 1..N − j do
            for indice ← 0..lungime do
                if s(i + indice) ≠ s(j + indice) then
                    raspuns ← raspuns + 1 mod 1.000.000.007
                end
            end
        end
    end
end
```

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

Solutie in $O(N^2)$ - 67 de puncte Asemănător cu pasul anterior, vom încerca să fixăm o anumită pereche de indici cu caractere diferite și să observăm cu cât contribuie la rezultat. Pentru asta, vom rescrie în funcție de pozițiile pe care le comparăm condițiile ca restul variabilelor să fie valide. Notăm cu a și b, $a < b$, pozițiile pe care le comparăm la fiecare pas și scriem condițiile pentru ca $i, j, lungime, indice$ sa fie valide.

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

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

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

Observați ca i și j sunt unic determinați dacă știm toate valorile $a, b, lungime, indice$, deci nu este necesar să păstram primele două ecuații pentru a verifica validitatea unei soluții. Astfel,

dacă avem a și b fixat, condițiile pe care trebuie să le îndeplinească *lungime* și *indice* sunt pentru a fi valide sunt:

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

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

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

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

Astfel, problema se reduce la calculul eficient al sumei $(a_1 + 1) + (a_2 + 1) + \ldots + (a_k + 1)$. Pentru a obține suma acestor indici, ne vom folosi de proprietatea lor: conțin o altă valoare decât $s(b)$. Putem ține un vector de sume $sum(c)$ care conține suma de $i+1$ pentru indicii i mai mici decât b-ul curent pentru care $s(i) = c$. Acest vector poate fi actualizat în $O(1)$ la fiecare pas, adăugând $b + 1$ la $sum(s(b))$. Atunci când vrem să obținem suma a-urilor din expresia de mai sus, vom însuma pur și simplu toate valorile $sum(c), c \neq s(b)$, ceea ce poate fi realizat în $O(sigma)$.

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

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

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

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

6.4 Problema Tort

prof. Marius Nicoli
Colegiul Național "Frații Buzești" Craiova

Enunț

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

Cerință

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

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

Date de intrare

Prima linie a fișierului de intrare `tort.in` conține numărul N. A doua linie conține valorile $a_1, \ldots, a_N$, separate prin spații.

Date de ieșire

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

Restricții

- $1 \leq N \leq 200.000$
- $1 \leq a_1, \ldots, a_N \leq 400.000$
- $a_1 + \ldots + a_N \leq 400.000$
- Pentru teste în valoare de 12 puncte, $1 \leq N \leq 20$
- Pentru alte teste în valoare de 12 puncte, $1 \leq N \leq 100$, $a_1 = \ldots = a_N = 1$.
- Pentru alte teste în valoare de 20 de puncte, $1 \leq N \leq 100$.
- Pentru alte teste în valoare de 28 de puncte, $1 \leq N \leq 1000$, $a_1 + \ldots + a_N \leq 2000$.

Exemplu

1)	tort.in	tort.out	**Explicații**
	5 1 1 2 1 1	6	Împărțirile valide sunt: $[1], [1, 2, 1, 1]$ $[1, 1], [2, 1, 1]$ $[1, 1], [2], [1, 1]$ $[1, 1, 2], [1, 1]$ $[1, 1, 2], [1], [1]$ $[1, 1, 2, 1], [1]$

Descrierea soluției

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

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

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

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

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

Capitolul 7

Clasele XI-XII

7.1 Problema Polihroniade

Propunător: stud. masterand Andrei Constantinescu

University of Oxford

7.2 Problema Bob

Propunător: stud. Costin-Andrei Oncescu
University of Oxford

7.3 Problema Dreptunghi

Propunător: Szabó Zoltan
Liceul Tehnologic "Petru Maior" Reghin/ ISJ Mureș

7.4 Descrierea soluțiilor

7.1 Problema Polihroniade

Figura 1: Elisabeta Polihroniade, 1979 în Rio de Janeiro

O matrice pătratică de dimensiuni $N \times N$ cu N par și elemente din mulțimea $\{0, 1\}$ se numește **tablă de șah** dacă oricare două celule vecine pe o linie sau pe o coloană au valori diferite (cu alte cuvinte, dacă nu există două valori egale alăturate).

De ziua ei, Victor i-a cumpărat Elisabetei o astfel de matrice A, care nu este *neapărat* tablă de șah. Aflând despre pasiunea ei, acesta vrea acum să transforme matricea A într-o tablă de șah. Timpul fiind limitat, el poate efectua doar următoarele tipuri de operații asupra matricei:

1. Interschimbă liniile i și j din A (celelalte linii rămân neschimbate, iar valorile din interiorul liniilor i și j rămân neschimbate și își păstrează ordinea). Operația are sens pentru $1 \leq i, j \leq N$.
2. Interschimbă coloanele i și j din A (celelalte coloane rămân neschimbate, iar valorile din interiorul coloanelor i și j rămân neschimbate și își păstrează ordinea). Operația are sens pentru $1 \leq i, j \leq N$.

Dorind să o impresioneze pe Elisabeta, Victor apelează la voi, programatori renumiți, să îl ajutați în a transforma matricea A într-o tablă de șah. Pentru aceasta, Victor are nevoie de următoarele informații:

1. Poate fi transformată matricea A în tablă de șah?
2. Care este numărul minim de operații necesare pentru a duce la îndeplinire acest scop?
3. Care ar fi o succesiune de operații care transformă matricea A într-o tablă de șah?

În cazul ultimei cerințe, pentru a intra în grațiile lui Victor va trebui ca numărul de operații efectuate să fie minim. Totuși, chiar și un număr neminim de operații va fi răsplătit, însă nu într-atât de mult.

Vi se dau două numere P, T și T matrice A, reprezentând mai multe instanțe ale problemei. Pentru fiecare matrice A dintre cele T va trebui să rezolvați cerința cu numărul $P \in \{1, 2, 3\}$ dintre cele listate mai sus.

Date de intrare

Pe prima linie se găsesc două numere întregi pozitive P și T, reprezentând numarul cerinței de rezolvat și, respectiv, numărul de scenarii pentru care va trebui să rezolvați problema.

Urmează cele T instanțe ale problemei, fiecare fiind compusă din $N + 1$ linii: pe prima linie se va afla numarul N, iar pe următoarele N linii câte N cifre binare **neseparate** prin spații, reprezentând câte o linie a matricei A.

Date de ieșire

Pentru fiecare dintre cele T instanțe se va afișa răspunsul, începând de la o linie nouă, după cum urmează:

1. Dacă $P = 1$, atunci se va afișa pe o singură linie 1 dacă matricea A poate fi transformată în tablă de șah, și 0 altfel.
2. Dacă $P = 2$, atunci se va afișa pe o singură linie un întreg reprezentând numărul minim de interschimbări de linii și/sau coloane necesare pentru a transforma matricea A în tablă de șah.
3. Dacă $P = 3$, atunci se va afișa pe o linie un număr X. Apoi, pe fiecare dintre următoarele X linii se va afișa câte o interschimbare de linii sau coloane, după următorul format: `L i j` are semnificația că liniile i și j se interschimbă, iar `C i j` are semnificația că coloanele i și j se interschimbă. Matricea obținută după aplicarea celor X operații asupra lui A în ordinea dată trebuie să fie o tablă de șah.

Restricții și precizări

- $1 \leq T \leq 350$
- $1 \leq N \leq 1\ 000$
- N este par.
- Pentru cerințele de tip $P = 2$ și $P = 3$ se garantează că matricea A poate fi transformată în tablă de șah folosind interschimbări de linii și/sau coloane.
- Suma valorilor N pentru cele T scenarii nu depășește 2 000.

Pentru 40 de puncte

- $P = 1$

Pentru alte 34 de puncte

- $P = 2$

Pentru alte 26 de puncte

- $P = 3$
- Dacă există mai multe soluții, oricare este considerată corectă.
- Dacă numărul X de operații folosite nu este minim, atunci se acordă 50% din punctajul pe test.

Exemple

stdin	stdout
1 3 2 11 11 4 1100 1100 0011 0011 2 10 01	0 1 1
2 2 4 1100 1100 0011 0011 2 10 01	2 0
3 2 4 1100 1100 0011 0011 2 10 01	3 L 2 4 C 2 3 L 1 2 0

Explicații

Pentru primul exemplu, avem $P = 1$, deci pentru cele $T = 3$ instanțe se cere numai dacă se pot transforma în tablă de șah prin interschimbări de linii și/sau coloane. Pentru prima instanță acest lucru **nu** este posibil, deoarece nu avem niciun 0 în matrice. Pentru cea de-a doua instanță, putem efectua următoarele operații:

$$\begin{matrix}1100\\1100\\0011\\0011\end{matrix} \xrightarrow{\texttt{L 2 3}} \begin{matrix}1100\\\mathbf{0011}\\\mathbf{1100}\\0011\end{matrix} \xrightarrow{\texttt{C 2 3}} \begin{matrix}1\mathbf{01}0\\0\mathbf{10}1\\1\mathbf{01}0\\0\mathbf{10}1\end{matrix}$$

Pentru ultima instanță, matricea A este deja tablă de șah.

Pentru al doilea exemplu, avem $P = 2$, deci pentru cele $T = 2$ instanțe se cere numărul minim de operații pentru a le transforma în tablă de șah. Pentru prima instanță, o soluție cu 2 operații este prezentată mai sus, și nu există soluții mai scurte. Pentru cea de-a doua instanță, matricea este deja tablă de șah, deci răspunsul este 0.

Al treilea exemplu este exact ca anteriorul, numai că avem $P = 3$, deci trebuie tipărite exact care sunt operațiile ce aduc matricea la forma de tablă de șah. Pentru punctaj maxim pe acest exemplu, ar fi obligatoriu să afișăm o soluție cu 2 operații (adică număr minim), cum ar fi cea de mai sus. Cu toate acestea, cu scop demonstrativ, noi venim cu o soluție compusa din 3 operatii, **prin care obinem doar** 50% **din punctaj**:

$$\begin{matrix}1100\\1100\\0011\\0011\end{matrix} \xrightarrow{\texttt{L 2 4}} \begin{matrix}1100\\\mathbf{0011}\\0011\\\mathbf{1100}\end{matrix} \xrightarrow{\texttt{C 2 3}} \begin{matrix}1\mathbf{01}0\\0\mathbf{10}1\\0\mathbf{10}1\\1\mathbf{01}0\end{matrix} \xrightarrow{\texttt{L 1 2}} \begin{matrix}\mathbf{0101}\\\mathbf{1010}\\0101\\1010\end{matrix}$$

7.2 Problema Bob

Dezamăgiți de lipsa fotbalului din ultima perioadă, Ștefan și Georgian și-au deschis (în secret) o afacere cu boabe de cafea, comercializând K tipuri diferite de cafea. Astfel, timp de N zile ei produc cafea, urmând să formeze din boabele obținute în zile **consecutive** pachete ce conțin **toate** tipurile de cafea.

Concret, cei doi știu pentru fiecare zi ce tipuri de cafea produc în acea zi (posibil niciun tip, caz în care afacerea ia o pauză), după care ei împart zilele în secvențe continue astfel încât, pentru fiecare tip de cafea, fiecare secvență de zile să conțină cel puțin o zi în care să fie produs acel tip de cafea.

Înainte de a se apuca de împachetat boabele, Ștefan și Georgian își pun două întrebări:

1. Care este numărul maxim de pachete ce pot fi formate?
2. Care este numărul de moduri de a împărți zilele astfel încât să se formeze număr maxim de pachete valide (ce conțin toate tipurile de cafea)?

Date de intrare

Pe prima linie se găsește un număr întreg P, reprezentând numărul cerinței de rezolvat.

Pe cea de-a doua linie se găsește un număr întreg T, reprezentând numărul de scenarii pentru care va trebui să rezolvați problema.

Urmează cele T instanțe ale problemei, fiecare fiind compusă din $N + 1$ linii: pe prima linie se vor afla două numere întregi N și K, reprezentând numărul de zile, respectiv numărul de tipuri diferite de cafea; pe următoarele N linii câte K cifre binare, cea de-a j-a cifră de pe linia i fiind 0 dacă în ziua i tipul j de cafea **nu** este produs, sau fiind 1 dacă în ziua i tipul j de cafea este produs.

Date de ieșire

Pentru fiecare dintre cele T instanțe se va afișa răspunsul, începând de la o linie noua, după cum urmeaza:

1. Daca $P = 1$, atunci se va afisa pe o singura linie numărul maxim de pachete valide ce pot fi formate.
2. Daca $P = 2$, atunci se va afisa pe o singura linie numărul de moduri de a împărți zilele în secvențe continue astfel încât să se formeze număr maxim de pachete. Răspunsul va fi afișat **modulo 1 000 000 007**.

Restricții și precizări

- $1 \leq P \leq 2$
- $1 \leq T \leq 3$
- $1 \leq N \leq 200\ 000$
- $1 \leq K \leq 20$
- Se garantează că fiecare tip de cafea apare în cel puțin una dintre cele N zile.

Punctare

- Pentru 6 puncte: $P = 1, N \leq 15$
- Pentru alte 6 puncte: $P = 1, N \leq 100$
- Pentru alte 9 puncte: $P = 1, N \leq 2\ 000$
- Pentru alte 10 puncte: $P = 1, N \leq 200\ 000$
- Pentru alte 10 puncte: $P = 2, K = 1, N \leq 200\ 000$
- Pentru alte 4 puncte: $P = 2, N \leq 15$
- Pentru alte 4 puncte: $P = 2, N \leq 20$
- Pentru alte 9 puncte: $P = 2, N \leq 100$

- Pentru alte 8 puncte: $P = 2, N \leq 700$
- Pentru alte 8 puncte: $P = 2, N \leq 2\ 000$
- Pentru alte 8 puncte: $P = 2, N \leq 10\ 000$
- Pentru alte 9 puncte: $P = 2, N \leq 70\ 000$
- Pentru alte 9 puncte: $P = 2, N \leq 200\ 000$

Exemple

stdin	stdout
1 3 3 3 010 101 111 6 2 10 01 00 10 11 01 5 4 0100 1010 0000 1110 0001	2 2 1
2 3 3 3 010 101 111 6 2 10 01 00 10 11 01 5 4 0100 1010 0000 1110 0001	1 3 1

Explicații

În primul exemplu, tipurile de cafea produse în fiecare zi sunt:

- În prima zi se va produce doar tipul 2 de cafea
- În cea de-a doua zi se vor produce tipurile 1 și 3 de cafea
- În ultima zi se vor produce toate cele 3 tipuri de cafea

Numărul maxim de pachete este 2, și este obținut în mod unic, grupând zilele în următorul fel: $[1, 2], [3]$.

În cel de-al doilea exemplu, numărul maxim de pachete este 2, fiind obținut în urmatoarele 3 moduri:

- $[1, 2], [3, 4, 5, 6]$
- $[1, 2, 3], [4, 5, 6]$
- $[1, 2, 3, 4], [5, 6]$

În cel de-al treilea exemplu, numărul maxim de pachete este 1, fiind obținut prin gruparea tuturor zilelor într-o singură secvență ($[1, 2, 3, 4, 5]$).

7.3 Problema dreptunghi

Avem la dispoziție un dreptunghi de dimensiuni $N \times M$. Ne este util ca dreptunghiul nostru să se asemene cu o matrice, de aceea vom considera că are N linii și M coloane. Vom segmenta si numerota dreptunghiul nostru după un anumit cod C. Prin segmentare se înțelege trasarea unei linii orizontale sau verticale la o anumită poziție k, ce va despărți dreptunghiul nostru în alte două dreptunghiuri mai mici:

- de dimensiuni $k \times M$ (cel de sus) și $(N-k) \times M$ (cel de jos) – în cazul unei linii (H)orizontale, operație codificată prin Hk
- de dimensiuni $N \times k$ (cel din stânga) și $N \times (M-k)$ (cel din dreapta) – în cazul unei linii Verticale, operație codificată prin Vk

Numerotarea dreptunghiului se realizează cu numerele naturale $1, 2, 3, ...$, în această ordine.

Codul C pentru segmentarea și numerotarea unui dreptunghi se definește recursiv. Dacă C_1 si C_2 sunt coduri de segmentare și numerotare, atunci:

- $*$ – în fiecare căsuță a dreptunghiului se va scrie valoarea curentă a numerotării. După aceea, această valoare este incrementată pentru a fi folosită de o ulterioară operație de tipul *;
- HkC_1C_2 – se trasează linia **orizontală** la poziția k, se segmentează și numerotează dreptunghiul de sus conform codului C_1, apoi se continuă cu segmentarea și numerotarea dreptunghiului de jos conform codului C_2;
- VkC_1C_2 – se trasează linia **verticală** la poziția k, se segmentează și numerotează dreptunghiul din stânga conform codului C_1, apoi se continuă cu segmentarea și numerotarea dreptunghiului din dreapta conform codului C_2.

De exemplu, dreptunghiul de dimensiuni 8×6 (8 linii, 6 coloane) segmentat și numerotat conform codului $C = H5H3V2**V3**V5V2***$, va arăta ca în Figura 1.

Un cod de segmentare și numerotare C este **valid** pentru un dreptunghi de dimensiuni $N \times M$ dacă și numai dacă pentru fiecare operație de tipul HkC_1C_2 și de tipul VkC_1C_2 din cadrul lui C, poziția k la care se trage linia orizontală, sau verticală respectiv, se află **strict** în interiorul dreptunghiului curent (adică pe **ambele** părți ale liniei trasate există cel puțin o linie si cel puțin o coloană rămase care vor fi ulterior numerotate conform definiției recursive a codului C).

Un cod de segmentare și numerotare C valid pentru un dreptunghi de dimensiuni $N \times M$ generează mai multe **subdiviziuni** (dreptunghiuri mai mici) delimitate de liniile orizontale și verticale trasate în cadrul lui C. De exemplu, pentru dreptunghiul din Figura 1, codul C din exemplul de mai sus generează **7** subdiviziuni.

1	1	2	2	2	2
1	1	2	2	2	2
1	1	2	2	2	2
3	3	3	4	4	4
3	3	3	4	4	4
5	5	6	6	6	7
5	5	6	6	6	7
5	5	6	6	6	7

Figure 1: Dreptunghi de dimensiuni 8×6.

Codul C nu este unic determinat. Pentru dreptunghiul segmentat și numerotat din Figura 1 există 4 coduri echivalente, pe care le scriem în ordine **lexicografică** în cele ce urmează:

1. $H3V2**H2V3**V2*V3**$
2. $H3V2**H2V3**V5V2***$
3. $H5H3V2**V3**V2*V3**$
4. $H5H3V2**V3**V5V2***$

Pentru stabilirea ordinii lexicografice a două codificări, fiecare informație **compacta** ce face parte din secvență se va considera entitate **separată**: adică simbolurile H, V, $*$ de tip caracter, respectiv numerele k de tip întreg, indiferent de numărul de cifre din care sunt formate.

La nivel de caractere ordinea lexicografică este $H < V < *$. Numerele se vor compara în funcție de valoarea lor, de exemplu $1 < 7 < 12$. Vom considera că un caracter este mai mic lexicografic decât un număr întreg.

De exemplu, următoarele două coduri echivalente sunt scrise în ordine lexicografică:

1. $V7 * V6 * *$
2. $V13V7 * **$

și corespund dreptunghiului de mai jos:

1	1	1	1	1	1	1	2	2	2	2	2	2	3	3
1	1	1	1	1	1	1	2	2	2	2	2	2	3	3
1	1	1	1	1	1	1	2	2	2	2	2	2	3	3
1	1	1	1	1	1	1	2	2	2	2	2	2	3	3
1	1	1	1	1	1	1	2	2	2	2	2	2	3	3

Cerință

Se dă un cod de segmentare și numerotare și se cere să se afle:

1. numărul de subdiviziuni pe care acesta le generează;
2. dimensiunile unui dreptunghi de arie minimă pentru care acest cod este valid;
3. numărul de codificări distincte **modulo 1 000 000 007**, echivalente cu codul citit (în acest număr va fi inclus și codul inițial);
4. primul cod în ordine lexicografică echivalent cu cel dat.

Date de intrare

De la intrarea standard se vor citi:

- de pe prima linie valoarea lui P;
- de pe linia următoare un șir de caractere reprezentând codul de segmentare și numerotare C.

Date de ieșire

- **Dacă valoarea citită pentru** P **este 1**, atunci la ieșirea standard se va tipări numărul de subdiviziuni pe care codul C le generează;
- **Dacă valoarea citită pentru** P **este 2**, atunci la ieșirea standard se vor tipări două numere N și M separate printr-un spațiu, dimensiunile unui dreptunghi de arie minimă pentru care codul C citit este valid. În caz că există mai multe se acceptă oricare;
- **Dacă valoarea citită pentru** P **este 3**, atunci la ieșirea standard se va tipări numărul de codificări distincte **modulo 1 000 000 007** echivalente cu codul citit (in acest număr va fi inclus și codul C citit).
- **Dacă valoarea citită pentru** P **este 4**, atunci la ieșirea standard se va tipări primul cod în ordine lexicografică echivalent cu cel dat;

Restricții și precizări

- $0 <$ lungimea codului C (număr de caractere) < 350
- Pentru teste în valoare de 14 puncte avem $P = 1$.
- Pentru teste în valoare de 21 de puncte avem $P = 2$.
- Pentru teste în valoare de 29 de puncte avem $P = 3$.
- Pentru teste în valoare de 36 de puncte avem $P = 4$.

Exemplu 1

stdin	stdout
1 H3V2**H2V3**V2*V3**	7

Explicație: În urma segmentării se obțin 7 dreptunghiuri.

Exemplu 2

stdin	stdout
2 H3V2**H2V3**V2*V3**	6 6

Explicație: Cel mai mic dreptunghi pentru care codul este valid are 6 linii și 6 coloane.

Exemplu 3

stdin	stdout
3 H3V2**H2V3**V2*V3**	4

Explicație: Numărul codurilor echivalente cu cel citit este 4 (vezi exemplul din enunț).

Exemplu 4

stdin	stdout
4 H3V2**H2V3**V2*V3**	H3V2**H2V3**V2*V3**

Explicație: Primul cod în ordine lexicografică echivalent cu cel citit este $H3V2 * *H2V3 * *V2 * V3 * *$.

7.4 Descrierea Soluțiilor

1 Problema Polihroniade

Propunător: Andrei Constantinescu, University of Oxford

Problema Polihroniade a fost gandita initial ca problema de dificultate relativ mai scazuta a setului, deoarece aceasta nu necesita vreo tehnica cunoscuta in prealabil. Desi ea poate parea greu de abordat la prima vedere, observam in cele ce urmeaza cum cateva observatii, relativ simple, fac problema mai usor de atacat.

1.1 Operatiile Comuta

Fie i, j, k, l numere intre 1 si N, atunci operatiile O_1 = `L i j` si O_2 = `C k l` comuta. Cu alte cuvinte, ne este indiferent daca asupra matricei A efectuam intai O_1 si apoi O_2 sau invers (adica O_2 si apoi O_1). In mod inductiv, deducem ca daca problema admite o solutie, atunci ea admite o solutie in care intai permutam liniile lui A, si apoi coloanele lui A. Mai mult, in mod informal putem spune ca liniile si coloanele sunt independente, fiecare fiind o problema de sine statatoare. Asadar, este suficient sa explicam cum efectuam operatiile pe linii, cele pe coloane fiind rezolvate analog. Totusi, rezolvarea este inca neclara - avem nevoie de alte observatii.

1.2 Metoda Mersului Invers

Sa consideram problema inversa: plecam de la o tabla de sah si vrem ca prin interschimbari succesive de linii si/sau coloane sa ajungem la matricea A. Cum operatiile din problema sunt reversibile, problema inversa nu este cu nimic mai grea sau mai usoara decat cea pusa in discutie. Intr-o tabla de sah exista fix 2 tipuri de linii: cele de forma `0101...01` si, respectiv, cele de forma `1010...10`. Mai mult, avem fix $\frac{N}{2}$ linii din fiecare tip. Ce este acum vital sa observam este ca operatiile pe linii si coloane nu schimba foarte mult aceasta configuratie. Mai exact, indiferent ce interschimbari am efectua, matricea rezultata va avea $\frac{N}{2}$ linii identice si alte $\frac{N}{2}$ linii identice care sunt fix complementul primelor $\frac{N}{2}$ (adica au `0` unde era `1` si vice-versa). Bineinteles, in ambele tipuri trebuie sa avem exact $\frac{N}{2}$ biti de `0` si $\frac{N}{2}$ biti de `1`, cum interschimbarile de orice fel nu schimba continutul liniilor, ci doar ordinea elementelor din ele. In mod clar, aceleasi observatii sunt valabile si pentru coloane.

Date fiind cele spuse, avem acum o conditie necesara pentru ca problema sa aiba solutie pentru o matrice A data: matricea A trebuie sa aiba exact 2 tipuri de linii: $\frac{N}{2}$ de un tip, si $\frac{N}{2}$ de celalalt tip, unde tipurile sunt complementare si contin exact $\frac{N}{2}$ biti de `1` (automat, si $\frac{N}{2}$ biti de `0`) fiecare. De asemenea, aceeasi conditie trebuie sa fie indeplinita si pentru coloane. Ce este posibil suprinzator acum este ca aceste conditii sunt si suficiente, adica, cu alte cuvinte, matricea A poate fi transformata intr-o tabla de sah daca **si numai daca** aceste conditii sunt adevarate pentru instanta noastra. Verificarea acestor conditii se poate face in timp $O(N^2)$ si aduce cu sine 40 de puncte pe problema (credem noi, aceasta este cea mai dificila parte a problemei, si este, prin urmare, rasplatita generos ca punctaj).

1.3 Transformarea Matricei A in Tabla de Sah cu Numar Minim de Operatii

Pentru restul probemei, facem presupunerea ca matricea A se poate transforma in tabla de sah, asa cum este specificat, de altfel, si in enuntul problemei. Cheia rezolvarii sta in urmatoarea observatie: atata timp cat aranjam ca prima linie a matricei A sa alterneze (`0101...01` sau `1010...10`) si ca prima coloana a matricei A sa alterneze, restul matricei va fi, in mod garantat, o tabla de sah (acest lucru rezulta din observatia de mai sus legata de tipul liniilor: avem $\frac{N}{2}$ linii de fiecare tip, si tipurile sunt complementare; similar pentru coloane). Mai sus am stabilit ca liniile si coloanele sunt independente, deci cele doua probleme se pot trata separat. De asemenea, ele se pot reformula mai simplu: dat fiind un sir binar de lungime N format din $\frac{N}{2}$ caractere `0` si $\frac{N}{2}$ caractere `1`, transformati-l intr-un sir alternant folosind un numar minim de operatii de interschimbare a doua caractere.

Exista doar doua siruri binare alternante de acest tip: `0101...01` si `1010...10`. Fara a restrange generalitatea, vom incerca sa il obtinem pe primul, dar pentru o solutie completa trebuiesc incercate ambele si intoarsa solutia care duce la un numar minim de operatii. In sirul `0101...01` observam ca toate caracterele `0` sunt pe pozitii pare si toate caracterele `1` sunt pe pozitii impare. In sirul nostru pe care dorim sa il aducem la aceasta forma exista trei tipuri de caractere:

1. Caractere care sunt la locul lor - adica `0` pe pozitii pare si `1` pe pozitii impare.
2. Caractere `0` care sunt pe o pozitie impara (deci trebuie obligatoriu sa faca parte din cel putin o interschimbare). Sa notam multimea pozitiilor de acest tip din sirul nostru cu S_0.
3. Caractere `1` care sunt pe o pozitie para (deci trebuie obligatoriu sa faca parte din cel putin o interschimbare). Sa notam multimea pozitiilor de acest tip din sirul nostru cu S_1.

Reiterand, pozitiile din $S_0 \cup S_1$ trebuie sa faca parte dintr-o interschimbare pentru a le aduce pe o pozitie cu paritatea corecta. Observam ca sunt exact atatea caractere `0` care nu sunt la locul lor cate caractere `1` nu sunt la locul lor (notam pentru aceasta ca numarul de caractere `0` si de caractere `1` este egal), deci $|S_0| = |S_1|$. Prin urmare, cele de mai sus ne spun ca avem nevoie de cel putin $|S_0|$ interschimbari pentru a rezolva problema. Mai mult, o solutie cu fix $|S_0|$ interschimbari se poate construi: interschimbam pe rand cate o pozitie din S_0 cu una din S_1 pana cand epuizam ambele multimi (fiecare interschimbare practic pune la locul lor doua caractere). Asadar, aceasta solutie este si optima din punctul de vedere al numarului de interschimbari. Bineinteles, trebuie sa aplicam acest algoritm si pentru coloane, dar acest lucru nu schimba conceptual rezolvarea. In total, solutia se construieste in timp $O(N)$ dupa citire. Cum este posibil acest lucru daca complexitatea de citire a matricei este $O(N^2)$? Ei bine, orice matrice care se poate transforma in tabla de sah este unic determinata de prima sa linie si prima sa coloana, deci s-ar putea spune ca insasi formatul de intrare este inhibitor pentru obtinerea unei complexitati timp mai bune (daca ni s-ar fi dat doar prima linie si prima coloana problema se putea rezolva in $O(N)$, lucru mentionat aici mai mult

ca o curiozitate matematica decat ca o observatie necesara pentru rezolvarea problemei). Folosind aceasta solutie obtinem restul de 60 de puncte pe problema.

2 Problema Bob

Propunător: Costin-Andrei Oncescu, University of Oxford

Problema Bob a fost pusa in concurs ca o problema de tehnica, care nu este foarte dificila pentru cunoscatorii de programare dinamica.

2.1 Calculul Numarului Maxim de Pachete

Prima cerinta, valorand 31 de puncte, se preteaza la un algoritm de tip Greedy relativ simplu: Se itereaza prin zile in ordine crescatoare, la fiecare pas retinand pentru fiecare tip de cafea daca l-am intalnit deja in pachetul curent sau nu. In momentul in care avem in pachetul curent toate tipurile de cafea, putem inchide pachetul si sa deschidem unul nou, reluand algoritmul de unde am ramas. Daca dupa terminarea celor N zile ramanem cu un pachet incomplet de cafea, atunci il vom alipi ultimului pachet inchis. Complexitatea totala este dominata de citire, si este, deci, $O(NK)$.

2.2 Calculul Numarului de Impartiri Maxime in Pachete

Aceasta cerinta este mai dificila, necesitand generalizarea ideii de mai sus. In primul rand, am dori sa calculam

e_i = cel mai mic j astfel incat intervalul de zile $i \dots j$ sa contina fiecare tip de cafea cel putin o data

Observati cum, de la un punct incolo, valoarea e_i este nedefinita, deoarce nici macar intervalul $i \dots N$ nu contine toate tipurile de cafea - in acest caz luam $e_i = N + 1$ prin conventie, lucru pe care il simulam, pentru simplitate, presupunand existenta unei zile fictive $N + 1$ in care se produc toate tipurile de cafea.

Vom calcula e_i in ordine descrescatoare a zilelor. Mai exact, pe masura ce iteram i, vom mentine inca un indice j (initial egal cu $N + 1$), care va scadea pe masura ce i scade ("tehnica celor 2 pointeri"). In particular, vom mentine tot timpul invariantul ca intervalul $i \dots j$ contine toate tipurile de cafea ce putin o data. De asemenea, vom retine un vector de frecventa f_k = in cate zile din intervalul $i \dots j$ se produce tipul de cafea k, pentru $1 \leq k \leq K$. Acum, cand ajungem la o noua valoare a lui i, actualizam f cu tipurile de cafea din ziua i, si apoi incercam sa scadem j cat mai mult; adica scadem j cat timp $i \dots j - 1$ contine toate tipurile de cafea (lucru pe care il verificam inspectand tipurile de cafea din ziua j si vazand daca eliminarea lor din f ar pastra toate valorile din f pozitive). Prin constructie, algoritmul prezentat ne asigura ca la finalul fiecarui pas avem $j = e_i$. Pe parcursul executiei atat i cat si j scad de $O(N)$ ori, deci complexitatea acestui past din rezolvare este $O(NK)$ si calculeaza corect vectorul e.

Mai apoi, ne propunem sa calculam

m_i = numarul maxim de pachete de cafea care se pot obtine considerand doar zilele $i \dots N + 1$

In mod intuitiv, $m_{N+1} = 1$, deoarce ziua $N+1$ constituie un pachet de sine statator. In spiritul algoritmului Greedy din prima parte a problemei, se remarca relativ usor ca $m_i = 1 + m_{e_i+1}$, oricare

ar fi $1 \leq i \leq N$. Putem, asadar, calcula vectorul m in timp $O(N)$, iterand i in ordine descrescatoare. Bineinteles, $m_1 - 1$ reprezinta chiar raspunsul la prima cerinta, calculat pe o alta cale (scazatorul se datoreaza pachetului fictiv $N + 1$), dar acum avem informatii suplimentare ce ne vor fi de folos.

In cele din urma, vrem sa calculam

$$d_i = \text{numarul de impartiri intr-un numar maxim de pachete de cafea (adica } m_i \text{ pachete) a zilelor } i \dots N + 1$$

deoarece raspunsul la a doua cerinta a problemei este chiar d_1. Pentru a continua rezolvarea, avem nevoie de observatia ca m_i **este monoton descrescator in** i.

Din definitie, avem ca $d_{N+1} = 1$. Pentru restul valorilor i, calculul lui d_i se va face in ordine descrescatoare de la N la 1. Fixam acum o valoare i si vedem in cele ce urmeaza cum putem calcula d_i in functie de $d_{i+1}, d_{i+2}, \dots$. Orice impartire in cat mai multe pachete valide a zilelor $i \dots N$ se compune dintr-un interval de forma $i \dots j$, unde obligatoriu $j \geq e_i$, si inca $m_i - 1$ alte intervale. Pentru a ne asigura ca impartirea obtinuta este optima, j trebuie ales de asa natura incat $m_{j+1} = m_i - 1$ si $j \leq N$ (intervalele de forma $i \dots N + 1$ sunt artificiale si nu trebuie luate in calcul). Deoarece m este monoton descrescator, valorile $e_i \leq j \leq N$ pentru care $m_{j+1} = m_i - 1$ formeaza un interval $[j_0, j_1]$, ale carui capete pot fi cautate binar in timp $O(\log N)$.[1] Date fiind acestea, putem calcula d_i prin expresia $d_i = \sum_{j=j_0}^{j_1} d_{j+1}$. O implementare naiva a acestei idei duce la o complexitate de $O(N^2)$, si este insuficienta.

Din fericire, ultimul pas este simplu: suma mentionata poate fi calculata in $O(1)$ daca in timp ce calculam valorile d_i mentinem si vectorul de sume partiale pe sufixe ale lui d (suma respectiva devine apoi o diferenta a doua sume partiale). Complexitatea calculului vectorului d este astfel $O(N \log N)$ datorita cautarii binare (si poate fi adusa la $O(N)$ folosind ideea mai rafinata din nota de subsol), deci complexitatea pentru toata problema este $O(NK + N \log N)$, suficienta pentru 100 de puncte.

[1] Pentru cei mai bravi dintre voi, ele pot fi calculate si in timp total $O(N)$, folosind paradigma celor 2 pointeri, dar acest lucru nu este necesar pentru punctaj maxim.

3 Problema Dreptunghi

Propunător: Szabó Zoltan, Liceul Tehnologic Petru Maior Reghin / ISJ Mureș Tg. Mureș

Problema **dreptunghi** are 4 cerințe cu diferite grade de dificultate. Însă rezolvarea principală se bazează pe faptul că observăm că definiția recursivității permite salvarea informațiilor într-o structură de arbore binar.

Arborele binar are ca rădăcină informația Hk sau Vk, apoi se definește subarborele stâng, și apoi subarborele drept.

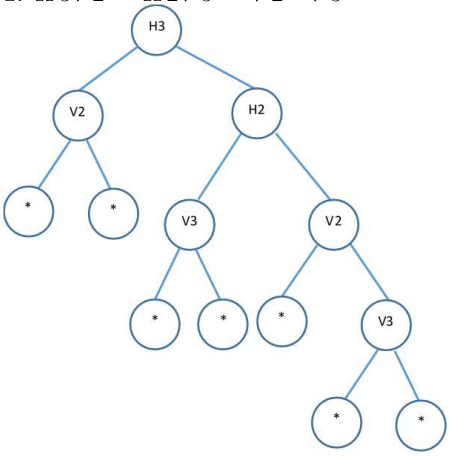
Informațiile din șirul de caractere se găsesc astfel în preodine, și putem să construim arborele.

1	1	2	2	2	2
1	1	2	2	2	2
1	1	2	2	2	2
3	3	3	4	4	4
3	3	3	4	4	4
5	5	6	6	6	7
5	5	6	6	6	7
5	5	6	6	6	7

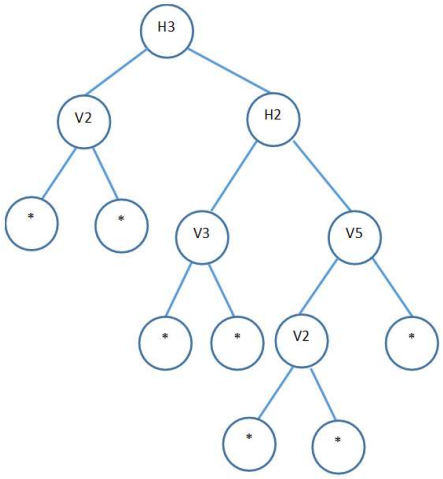
Pentru dreptunghiul de mai sus există 4 coduri, așa cum s-a prezentat în enunțul problemei. Fiecărui cod îi corespunde un arbore binar. În continuare vom desena fiecare arbore, iar apoi vom studia prorpietățile necesare pentru a rezolva cerințele problemei.

În continuare prezentăm arborii corespunzători celor 4 coduri de numerotare, luate în ordine lexicografică:

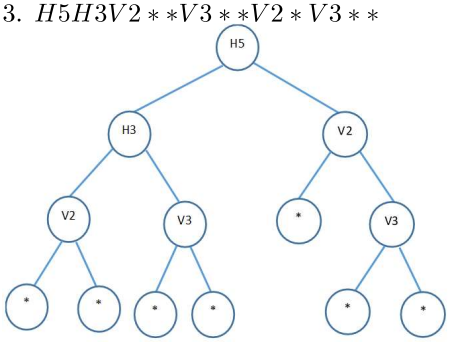
1. $H3V2 * *H2V3 * *V2 * V3 * *$



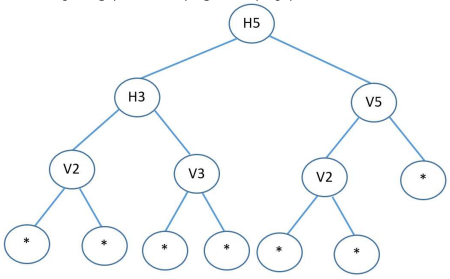
2. $H3V2**H2V3**V5V2***$



3. $H5H3V2**V3**V2*V3**$



4. $H5H3V2**V3**V5V2***$



3.1 Prima cerință (14 puncte)

$P = 1$ **(Numărul de subdiviziuni pe care codul C le generează)**

Răspunsul la această întrebare este numărul caracterelor '*' pe care le conține codul C.

Este cerința cea mai simplă a problemei și se poate rezolva prin simpla parcurgere a tuturor elementelor șirului de caractere citit și calcularea numărului de apariții al caracterului '*' în șir.

De asemenea din arborii desenați mai sus, observăm că numărarea unui dreptunghi se întâmplă când suntem la o frunză a arborelui, deci rezultatul cerut va fi numărul frunzelor arborelui binar pe care îl construim la citire. Acest rezultat se poate obține printr-o parcurgere în adâncime a arborelui.

3.2 A doua cerință (21 de puncte)

$P = 2$ **(Dimensiunile unui dreptunghi de arie minimă pentru care acest cod este valid)**

Studiind informațiile din arborii desenați mai sus, vom observa că:
Dacă nodul din rădăcina (sub)arborelui conține:

(a) o literă H și numărul k - asta înseamnă că

- dimensiunea orizontală a dreptunghiului trebuie să permită o linie la coordonata k, deci dreptunghiul va avea o dimensiune orizontală de cel puțin valoare k.
- toate liniile orizontale în subarborele stâng vor fi trasate în subdreptunghiul de sus de coordonate mai mici, deci **nu vor influența** dimensiunea orizontală a dreptunghiului actual.
- dimensiunea orizontală va fi afectată **cumulativ** de dimensiunea orizontală depistată în subarborele drept.
- dimensiunea verticală va fi maximul dintre dimensiunile verticale indicate de subarborele stang respectiv subarborele drept.

(b) o literă V și numărul k - asta înseamnă că

- dimensiunea verticală a dreptunghiului trebuie să permită o linie la coordonata k, deci dreptunghiul va avea o dimensiune verticală de cel puțin valoare k.
- toate liniile verticale în subarborele stâng vor fi trasate în subdreptunghiul din stânga de coordonate mai mici, deci **nu vor influența** dimensiunea verticală a dreptunghiului actual.
- dimensiunea verticală va fi afectată **cumulativ** de dimensiunea verticală depistată în subarborele drept.
- dimensiunea orizontală va fi maximul dintre dimensiunile orizontale indicate de subarborele stâng respectiv subarborele drept.

(c) un caracter '*' - asta înseamnă că

- este vorba despre cazul banal de dreptunghi minim 1×1.

3.3 A treia cerință (29 de puncte)

$P = 3$ **(Numărul de codificări distincte)**

Se știe că numărul arborilor binari cu N noduri este egal cu numărul lui Catalan de ordin N.

O parte conexă maximală a arborelui în care toate nodurile sunt etichetate identic cu litera H respectiv V, au proprietatea că permit rescrierea arborelui pentru o codificare echivalentă, prin schimbarea raportului Tată-Fiu al rădăcinii și a descendentului din stânga sau dreapta. Dacă partea conexă conține K noduri, atunci numărul arborilor pe care ii putem construi va fi $C(K)$, fiind numărul lui Catalan de ordin K. Rezultatul pentru problema noastră va fi produsul numerelor Catalan al numărului de noduri din *fiecare componentă conexă din arbore*, etichetate cu aceeași literă H sau V. Întrucât lungimea șirului de caractere ce reprezintă intrarea pentru problema noastră nu depășește 350 de carctere, rezultă cu nu suntem nevoiți să calculăm optim acest număr, chiar si o solutie calculată în $O(N^2)$ va intra satisfăcător în timpul de execuție.

Formula propusă este $C(n) = C(0) \cdot C(n-1) + C(1) \cdot C(n-2) + \ldots + C(n-1) \cdot C(0)$ pentru orice $n > 0$, și $C(0) = 1$.

Pentru exemplul nostru, observăm că oricare din subarbori are 6 noduri care formează 4 zone conexe:

- o zonă formată din 2 noduri etichetate cu H.
- o zonă formată din 2 noduri etichetate cu V.
- o zonă formată dintr-un nod etichetat cu V.
- o zonă formată dintr-un nod etichetat cu V.

Deci, rezultatul calculat va fi $C(2) \cdot C(2) \cdot C(1) \cdot C(1) = 2 \cdot 2 \cdot 1 \cdot 1 = 4$.

3.4 A patra cerință (36 de puncte)

$P = 4$ **(Primul cod în ordine lexicografică echivalent cu cel dat)**

Studiind arborii pe care i-am prezentat mai sus, observăm:

Arborele corespunzător primului cod în ordine lexicografică are proprietatea că toate nodurile arborelui care sunt etichetate cu V sau H au un nod fiu din stânga, iar eticheta acestui nod diferă de valoarea respectivă.

Sarcina noastră este ca să modificăm structura nodurilor astfel ca pentru niciun nod să nu existe etichetă identică de H sau V pentru nodul fiului din stânga.

Cu o parcurgere în adâncime vom depista toate aceste noduri, și vom rearanja nodurile din arbore, astfel încat descendentul din stânga să devină tată la vechiul tată, iar nodul tată se va transforma în fiu din dreapta.

Algoritmul va realiza în mod repetat modificarea arborelui, până când nu vor mai exista noduri cu această proprietate, apoi cu o parcurgere în preordine se va putea tipări răspunsul la cerința problemei, și anume primul cod în ordine lexicografică.

Capitolul 8

Anexe - Exemple implementări soluții în limbajul C/C++

8.1 Soluții - Clasa a V-a

1. Ghiocel

```
/*
1.1. Autor Gina Balacea
*/
#include<fstream>
using namespace std;
long long n,p,n1,n2,nrap;
int C,c,N,i;
ifstream f("ghiocel.in");
ofstream g("ghiocel.out");
int main()
{
  f>>N;
  int urc=0,cob=0,deal=0,vale=0,k,d;
  for(i=1;i<=N;i++)
    {   f>>n;
        k=0;d=0;
        int aux=n;
        while(n>9 && (n%10==(n/10)%10)) n=n/10;
        if(n>9)
            if(n%10<(n/10)%10) d=-1;
        else d=1;
        n=n/10;
        while(n>9)
        {

         if(((d<0)&&(n%10>(n/10)%10)) || ((d>0)&&(n%10<(n/10)%10)))
        {
            k++;d=-d;
        }
        n=n/10;
        }
        if(k==0)
           {if(d==0) ;
            else
                if(d>0) {urc++; }
                else {cob++;}
           }
        else
            if(k==1)
              if(d<0) {  vale++;}
              else { deal++; }
            else
                if(k>1)  ;
    }
```

```
   g<<urc<<'\n'<<cob<<'\n'<<deal<<'\n'<<vale<<'\n';
   f.close();
   g.close();
   return 0;
}
```

```
/*
 1.2. Autor: Florentina Ungureanu
*/
#include <fstream>
using namespace std;
ifstream in("ghiocel.in");
ofstream out("ghiocel.out");
unsigned long long n, x,i,nu,nc,nv,nd,c;
int main()
{
    in>>n;
    for(i=1;i<=n;i++)
        {
            in>>x;
            c=x%10;x/=10;
            if(x%10<c)
            {
                while(x&&x%10<c) c=x%10,x/=10;
                if(!x) nc++;
                else
                {
                    while(x&&x%10>c) c=x%10,x/=10;
                    if(!x) nv++;
                }
            }
            else
            {
                while(x&&x%10>c) c=x%10,x/=10;
                if(!x) nu++;
                else
                {
                    while(x&&x%10<c) c=x%10,x/=10;
                    if(!x) nd++;
                }
            }
        }
    out<<nc<<endl;
    out<<nu<<endl;
    out<<nd<<endl;
    out<<nv<<endl;
    return 0;
}
```

2. Concurs

```
/*
2.1. Autor: Georgeta Balacea
*/
#include<fstream>
using namespace std;
int x,n,nca,ncb,ncc,pmax,pa,pb,pc,pct,nrc;
int C;
long long P;
```

```
ifstream f("concurs.in");
ofstream g("concurs.out");
int main()
{   f>>x>>n;
    if(x==2)
    {
        for(int i=1; i<=n; i++)
        {   f>>C>>P;
            pct=0;
            while(P)
            {
                if(P%10==0)
                {
                    pct+=10;
                    P/=100;
                }
                else
                {
                    pct+=P%10;
                    P/=10;
                }
            }
            if(C==1) pa+=pct;
            else if(C==2) pb+=pct;
                   else pc+=pct;
        }
        pmax=max(pa,max(pb,pc));
        if(pmax)
        {   if(pa==pmax) g<<1<<' ';
            if(pb==pmax) g<<2<<' ';
            if(pc==pmax) g<<3<<' ';
            g<<pmax<<endl;
        }
        else
            g<<"FARA CAMPION"<<endl;
    }
    else
    {   for(int i=1; i<=n; i++)
        {   f>>C>>P;
            pct=0;long long aux=P;
            while(P)
            {   if(P%10==0)
                {
                    pct+=10; P/=100;
                }
                else
                {
                    pct+=P%10; P/=10;
                }
            }
            if(pct>pmax)
            {
                nrc=1; pmax=pct;
            }
            else
                if(pct==pmax) nrc++;
        }
         g<<pmax<<" "<<nrc<<endl;
        }
    f.close(); g.close();
    return 0;
}
```

```
/*
2.2. Autor Dumitrascu Dan Octavian
*/
#include <fstream>
using namespace std;

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

long long smax, nr, p,x,n,i,c, max1, c1, s, aux, s1, s2, s3;

int main()
{
    f>>c;
    if (c==1)
    {
        f>>n;
        max1=0;
        for (i=1;i<=n;i++)
        {
            f>>p;
            f>>x;

                aux=x;
                s=0;
                while (aux>0)
                {
                    if (aux%10==0)
                    {
                        s=s+10;
                        aux=aux/100;
                    }
                    else
                    {
                        s=s+aux%10;
                        aux=aux/10;
                    }
                }
                if (s>max1)
                {
                    max1=s;
                    nr=1;
                }
                else
                    if (s==max1) nr++;


        }
        g<<max1<<" "<<nr<<"\n";
    }
    else
    {
        f>>n;
        max1=0;
        s1=0;
        s2=0;
        s3=0;
        for (i=1;i<=n;i++)
        {
            f>>p;
            f>>x;
            if (x==0) s=0;
```

```
            else
            {
                aux=x;
                s=0;
                while (aux>0)
                {
                    if (aux%10==0)
                    {
                        s=s+10;
                        aux=aux/100;
                    }
                    else
                    {
                        s=s+aux%10;
                        aux=aux/10;
                    }
                }
                if (p==1) s1=s1+s;
                if (p==2) s2=s2+s;
                if (p==3) s3=s3+s;
            }
        }
        if (s1==0&&s2==0&&s3==0)
            g<<"FARA CAMPION"<<"\n";
        else
        {
            max1=0;
            if (s1>max1) max1=s1;
            if (s2>max1) max1=s2;
            if (s3>max1) max1=s3;
            if (s1==max1) g<<"1 "<<max1<<"\n";
            if (s2==max1) g<<"2 "<<max1<<"\n";
            if (s3==max1) g<<"3 "<<max1<<"\n";
        }
    }

    return 0;
}
```

```
/*
2.3. Autor: Florentina Ungureanu
*/
#include <fstream>
using namespace std;

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

int E,c, n;
long long x, p1, p2, p3, p, pmax, k;

int main()
{
    in>>c>>n;
    p1=p2=p3=p=pmax=k=0;
    for(int i=1; i<=n; i++)
    {
        in>>E>>x;
        p=0;
        while(x)
        {
```

```
            if(x%10) p=p+x%10;
            else
            {
                p+=10;
                x/= 10;
            }
            x/= 10;
        }
        if(c==1)
            if(p>pmax)
            {
                pmax=p;
                k=1;
            }
            else if(p==pmax) k++;
            else;
        else E==1?p1+=p:(E==2?p2+=p:p3+=p);
    }
    if(c==1) out<<pmax<<' '<<k<<'\n';
    else if(p1+p2+p3==0) out<<"FARA CAMPION\n";
    else
    {
        pmax=max(p1,max(p2,p3));
        if(p1==pmax) out<<"1 ";
        if(p2==pmax) out<<"2 ";
        if(p3==pmax) out<<"3 ";
        out<<pmax<<'\n';
    }
    return 0;
}
```

3. Șir

```
/*
3.1.Autor Georgeta Balacea
*/
#include<fstream>
using namespace std;
int C,N,x,maxi,pmax,i,j,s,S;
ifstream f("sir.in");
ofstream g("sir.out");
int main()
{
    f>>C>>N;
    if(C==1)
    {   for(i=1; i<=N; i++)
        {
            f>>x;
            if(x>=maxi)
            {
                maxi=x;
                pmax=i;
            }
        }
        g<<pmax<<endl;
    }
    else if(C==2)
    {
        for(i=1; i<=N; i++)
```

```
        {
            f>>x;
            if(x>maxi)
            {
                maxi=x;
                g<<i<<" ";
            }
        }
        g<<endl;
    }
    else
    {
        for(i=1; i<=N; i++)
        {
            f>>x;
            if(x>maxi)
            {
                maxi=x;
               s=0;
            }
            else
            {
                s+=maxi-x;
                if(x==maxi) S=s;
            }
        }
        g<<S<<endl;
    }
    f.close();
    g.close();
    return 0;
}
```

```
/*
3.2. Autor Dumitrascu dan Octavian
*/
#include <fstream>
using namespace std;
ifstream f("sir.in");
ofstream g("sir.out");
int c,n,i,x,max1,p1,p2,sf,p,s;

int main()
{   f>>c;
    if (c==1)
    {   f>>n;
        max1=0;
        for (i=1;i<=n;i++)
        {
            f>>x;
            if (x>=max1)
            {
                max1=x; p=i;
            }
        }
        g<<p<<"\n";
    }
    else
    if (c==2)
    {   f>>n;
        max1=0;
```

```
        for (i=1;i<=n;i++)
        {   f>>x;
            if (x>max1)
            {
                max1=x; g<<i<<" ";
            }
        }
        g<<"\n";
    }
    else
    {
        f>>n;
        max1=0;
        p1=0;
        p2=0;
        s=0;
        sf=0;
        for (i=1;i<=n;i++)
        {
            f>>x;
            s=s+x;
            if (x>max1)
            {
                max1=x;
                p1=i;
                p2=p1;
                s=0;
            }
            else
            if (x==max1)
            {
                p2=i;
                sf=s;
            }
        }
        g<<(p2-p1-1)*max1-(sf-max1);
    }
    return 0;
}
```

```
/*
3.3. Autor: Florentina Ungureanu
*/
#include <fstream>
using namespace std;

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

int n, c, x, maxi, i, poz, g, rez;
int main ()
{
    in>>c>>n;
    switch (c)
    {
    case 1:
    {
        maxi=poz= 0;
        for (i=1; i<=n; i++)
        {
            in>>x;
```

```
        if (x >= maxi)
            {
                maxi = x;
                poz = i;
            }
        }
        out<<poz<<'\n';
        return 0;
    }
    case 2:
    {
        maxi=0;
        for (i=1; i<=n; i++)
        {
            in>>x;
            if (x > maxi)
            {
                maxi = x;
                out<<i<<' ';
            }
        }
        out<<'\n';
        return 0;
    }
    case 3:
    {
        for (i=1; i<=n; i++)
        {
            in>>x;
            if (x > maxi)
            {
                maxi = x;
                g = 0;
            }
            else g += maxi-x;
            if (x == maxi)rez = g;
        }
        out<<rez<<'\n';
    }
    }
    return 0;
}
```

8.2. Soluții - Clasa a VI-a

1. Stergeri

```
/*

1.1.Autor Lica Daniela

*/

#include <fstream>
#define DIM 100010
using namespace std;
int w[DIM], n, pas, i, j, x, maxim, p, u, m;

int main () {

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

    fin>>n;
    for (i=1;i<=n;i++) {
        fin>>x;
        while (x) {
            w[i]+=x%10;
            x /= 10;
        }
    }
    m = 0;
    while (n)
    {
        pas++;
        maxim = 0;
        for (i=1;i<=n;i++)
            if (w[i] > maxim)
            {
                maxim = w[i];
                p = i;
                u = i;
            }
            else
                if (w[i] == maxim)
                    u = i;
        for (i=p,j=u+1;j<=n;i++,j++)
            w[i] = w[j];

        if (u-p+1 > m)
            m = u-p+1;
        n -= (u-p+1);
    }

    fout<<pas<<" "<<m<<"\n";
    return 0;
}
```

```
/*
1.2.Autor Șerban Marinel
*/
#include <fstream>

using namespace std;

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

short int V[100005];

int n, i, p, u, smax, pasi, cate_max;;

int sum(int x)
{
   int s = 0;
   while (x)       //calculez suma cifrelor
   {
      s += x % 10;
      x /= 10;
   }
   return s;
}

void cauta_max(int pana_unde, int &p, int &u)
{
   int i, Vmax;
   Vmax = V[1]; p = u = 1;          //un element maxim
   for (i = 2; i <= pana_unde; i++)//caut alt maxim
      if (V[i] > Vmax)
      {
         Vmax = V[i];               //am gasit un alt maxim
         p = u = i;                 //este singur
      }
      else
         if (V[i] == Vmax)          //am mai gasit unul
            u = i;                  //este ultimul gasit
}
void sterge(int p, int u)
{
   int i, de_unde = p, pana_unde = u;
   pasi++;
   if (u - p + 1 > cate_max)      //determin numarul maxim
      cate_max = u - p + 1;       //de elemente eliminate
   for (i = u + 1; i <= n; i++)   //elimin elementele
   {
      V[p] = V[i];
      p++;
   }
   n -= (pana_unde - de_unde + 1);//elemente ramase
}
void eliminare()
{
   int p, u;
   while (n)              //cat timp mai am elemente
   {
      cauta_max(n, p, u);//cauta p si u pentru maxim
      sterge(p, u);      //elimina secventa
   }
}
```

```
int main()
{
   int x, i;
   fin >> n;
   for (i = 1; i <= n; i++)
   {
      fin >> x;       //citesc elementul
      V[i] = sum(x); //suma cifrelor
   }
   eliminare();       //simulez eliminarea
   fout << pasi << ' ' << cate_max << '\n';
   return 0;
}
```

2. Formula 1

```
/*
2.1. Autor Prof. Miana Arisanu
     algoritm liniar, 70-80 puncte
*/

#include <fstream>
using namespace std;

ifstream f("formula1.in");
ofstream g("formula1.out");
int main()
{
    unsigned long long N, P, k, A,x;
    f>>P>>k>>N;
    if (P==1)
        g<<(2*k-1)*(2*k-1);
    else
    {
        A=N*(N+1)*(2*N+1)/3- N*(N+1);
        // x= numarul masinii care are A stegulete albe
        x=1;
        while (2*x*x-2*x<=A)x++;
        x--;
       // pentru eficienta x se determina direct cu formula sau se cauta binar
        g<<(2*x-1)*(2*x-1)<<'\n';

    }
}
```

```
/*
 2.2. Autor Roxana Timplaru
*/

#include <fstream>
#include <cmath>
using namespace std;
ifstream fin("formula1.in");
ofstream fout("formula1.out");
int k,n;
void cerinta_1()
```

```
{
    long long s=1;
    s=s+2*(k-1);
    fout<<s*s;
}
void cerinta_2()
{
    long long s,a,i,x;
    s=0;a=1;
    for(i=1;i<=n;i++)
    {
        x=a*a/2;
        s=s+x;
        a=a+2;
    }
   a=1;x=0;
   while (a*a/2<=s)
   {
       a=a+2;
   }
   a=a-2;
   fout<<a*a;
}

void cerinta2_formula()
{
    long long s,a,i,x;
    s=0;
    a=1;
    for(i=1;i<=n;i++)
    {
        x=a*a/2;
        s=s+x;
        a=a+2;
    }
    x=sqrt(2*s+1);
    if(x%2==0) x--;
    fout<<x*x;

}

int main()
{
    int c;
    fin>>c>>k>>n;
    if(c==1)
        cerinta_1();
        else
            {
                //cerinta_2();
                cerinta2_formula();
            }
    return 0;
}
```

```
/**
 *  2.3 Autor sursa: Radu Muntean
 *  Complexitate: O(1)
 *  Scor: 100 puncte
 *
 */


#include <fstream>
#include <assert.h>
#include <cmath>
using namespace std;

int main() {

      ifstream in ("formula1.in");
      ofstream out ("formula1.out");
      int mode;
      long long n, k;

      in >> mode >> k >> n;
      assert(mode > 0);
      assert(mode <= 2);

      if (mode == 1) {
      out << (2*k - 1) * (2*k - 1) << "\n";
      return 0;
      }

      // A =(1^2-1)/2 + (3^2-1)/2 + ... ((2n-1)^2-1)/2

      // A =(1^2-1 + 3^2-1 + ... (2n-1)^2-1)/2

      // A = (1^2 + 3^2 + ... (2n-1)^2
      //    -n)/2

      // A = (1^2 + 2^2 + 3^2 + ... + (2n-1)^2 -      adunam nr pare si impare
      //    2^2 + 4^2 + ... + (2n - 2)^2        scadem numerele pare
      //    -n) / 2

      // 1^2 + 2^2 + ... + k^2 = k * (k+1) * (2k + 1) / 6


     long long A =  (1LL * (2*n - 1) * (2*n) * (4*n - 1) / 6  -
                        // suma pentru toate patratele de la 1 la (2n-1)^2

                4 * (n - 1) * n * (2*n - 1) / 6  - n ) / 2;
                        // suma pentru toate patratele de nr pare
                        // cumul de n ori "-1"

      long long x = sqrt(A * 2 + 1);
                         // A reprezinta numarul maxim de patratele albe.

      if (x % 2 == 0) {
      x --;
      }

      out << x*x << "\n";

      return 0;
}
```

```
/*
2.4. Autor Prof. Daniela Lica
     Solutie de 100 de puncte folosind cautarea binară la cerința 2
*/

#include <fstream>
using namespace std;

ifstream f("formula1.in");

ofstream g("formula1.out");

int P, N, K;

void Read (){

    f >> P >> K >> N;
    return;
}
long long cnt (int k){
    return (1LL * (1LL * (k << 1LL) - 1LL) * 1LL * ((k << 1LL) - 1LL));
}
void Task_1 (){
    g << cnt(K) << '\n';
    return;
}
///cautare binară a rezultatului
void Task_2 (){

    long long total = 0;

    for(int k = 1; k <= N; ++k)

        total += 1LL * (cnt(k) >> 1LL);

    long long Keep = 0;

    int Left = 1, Right = 1e8;

    while(Left <= Right)

    {

        int Mid = ((Left + Right) >> 1);
        if((cnt(Mid) >> 1LL) <= total)
            Keep = cnt(Mid), Left = Mid + 1;
        else
            Right = Mid - 1;
    }
    g << Keep << '\n';
    return;
}
void Solve (){
    if(P == 1)
        Task_1();
    else
        Task_2();
    return;
}

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

3. Seism

```
/*
3.1 Autor Iordaiche Cristina
*/
#include <fstream>
using namespace std;
ifstream fin("seism.in");
ofstream fout("seism.out");

bool v[100001];
int N,C,s;
int durata_seism,cate_seisme,durata_max;

int main()
{
    int i,incep_seism,sf_seism,j;
    fin>>C>>N;
    for(i=1; i<=N; i++)
        fin>>v[i];

    if (C==1 || C==2)
    {
        i=1;
        while(v[i]) //sar peste elementele nenule de la inceput
            i++;  //deoarece nu au 0 in stanga
        while(i<=N-2)
        {
            while(v[i]==0&&i<=N-2) //sar peste 0
                i++;
            incep_seism=i;
            while(v[i]&&i<=N-2) //secventa cu elemente nenule
                i++;
            sf_seism=i-1;
            if(incep_seism-2>=1 && v[incep_seism-1]==0 && v[incep_seism-2]==0)
                if(v[sf_seism+1]==0 && v[sf_seism+2]==0 && sf_seism+2<=N)
                // E seism incadrat stanga dreapta de cate 2 zerouri

                {
                    cate_seisme++;
                    durata_seism=sf_seism-incep_seism+1;
                    if(durata_seism>durata_max)
                        durata_max=durata_seism;
                }
            i++;
        }

        if (C==1)
            fout<<durata_max<<'\n';
        else if(C==2)
            fout<<cate_seisme<<'\n';
    }

    else //if(C==3)
    {
        int incep_secv0,sf_secv0,k1,k2,durata_seism=0,durata_max=0;
        i=1;
        while(v[i]) i++;//SAR peste elementele egale cu 1 de la început
```

```
        while(i<=N-2)
        {
            if(i!=1)
                incep_secv0=i;
            else incep_secv0=3;

            while(v[i]==0 && i<=N-2) i++;//secventa de 0
            sf_secv0=i-1;
            for(int j=incep_secv0; j<=sf_secv0; j++)
                v[j]=1;//schimb  fiecare secventa de 0 in 1
            if( sf_secv0-incep_secv0>=5)
            // tratez cazul in care fac seism in interiorul unei secvente de 0
            {
                //if(i==N)
                durata_seism=(sf_secv0-incep_secv0+1)-4;
                if (durata_seism>durata_max)
                    durata_max=durata_seism;
            }
            // recalculez lungimea seismului din acest moment adica ma duc
            // stanga-dreapta
            durata_seism=sf_secv0-incep_secv0+1;
            k1=incep_secv0-1;
            while(v[k1] &&k1>=3) k1--;
            k1++;

            k2=sf_secv0+1;
            while(v[k2]&& k2<=N-2) k2++;
            k2--;

            if(k1-2>=1 && k2+2<=N)
                if(v[k1-1]==0 && v[k1-2]==0 && v[k2+1]==0 && v[k2+2]==0)
                //verific daca are in stanga si dreapta cate 2 de 0
                {
                    durata_seism=k2-k1+1;
                    if( durata_seism>durata_max)
                        durata_max=durata_seism;
                }
            // refac secventa de 0
            for(int j=incep_secv0; j<=sf_secv0; j++)
                v[j]=0;
            while(v[i] &&i<=N-2) i++;
        }
        fout<<durata_max<<'\n';
    }
    return 0;
}
```

```
/*
3.2 Autor Daniela Lica
*/
#include <fstream>
#include <cassert>
using namespace std;
ifstream f("seism.in");
ofstream g("seism.out");

const int NMAX = 1e5 + 1;

int P, N;
bool A[NMAX];
int M, V[NMAX];
int ans_1, ans_2, ans_3;
```

```
void Read ()
{
     f >> P >> N;

    for(int i = 1; i <= N; ++i)
        f >> A[i];

    return;
}
int my_abs (int x)
{
    if(x < 0)
        return -x;
    return x;
}
///secvențele de 0 sunt memorate în vectorul V cu valori negative,
///în valoare absolută egale cu lungimea lor
///secvențele de 1 sunt memorate în vectorul V cu valori pozitive,
///egale cu lungimea lor

void Load ()
{
    M = 0;
    for(int i = 1; i <= N; ++i)
        if(A[i] == 0)
        {
            V[++M] = 0;

            while(i <= N && A[i] == 0)
                ++V[M], ++i;
            V[M] = -V[M];
            --i;
        }
        else
        {
            V[++M] = 0;
            while(i <= N && A[i] == 1)
                ++V[M], ++i;

            --i;
        }

    return;
}
int my_max (int a, int b)
{
    return ((a > b) ? a : b);
}
void Task_1 ()
{
    ans_1 = 0;

    for(int i = 2; i < M; ++i)
        if(V[i] > 0)
            if((-V[i - 1]) > 1 && (-V[i + 1]) > 1)
                ans_1 = my_max(ans_1, V[i]);

    g << ans_1 << '\n';
    return;
}

void Task_2 ()
```

```
{
    ans_2 = 0;

    for(int i = 2; i < M; ++i)
        if(V[i] > 0)
            if((-V[i - 1]) > 1 && (-V[i + 1]) > 1)
                ++ans_2;

    g << ans_2 << '\n';
    return;
}
void Task_3 ()
{
    ans_3 = 0;
   ///creez un seism intr-o secventa de 0 de lungime > 4
    for(int i = 1; i <= M; ++i)
        if(V[i] < 0 && my_abs(V[i]) > 4)
            ans_3 = my_max(ans_3, my_abs(V[i]) - 4);

    /// unesc 2 secvente de 1
    for(int i = 3; i <= (M - 2); ++i)
        if(V[i] < 0)
        {
            if(my_abs(V[i - 2]) > 1 && my_abs(V[i + 2]) > 1)
                ans_3 = my_max(ans_3, V[i - 1] + my_abs(V[i]) + V[i + 1]);
        }
   /// extindem durata seismului spre stanga
    for(int i = 3; i <= M; ++i)
        if(V[i] < 0)
        {
            if(my_abs(V[i - 2]) > 1 && my_abs(V[i]) > 2)
                ans_3 = my_max(ans_3, V[i - 1] + my_abs(V[i]) - 2);
        }
  /// extindem durata seismului spre dreapta
    for(int i = 1; i <= (M - 2); ++i)
        if(V[i] < 0)
        {
            if(my_abs(V[i + 2]) > 1 && my_abs(V[i]) > 2)
                ans_3 = my_max(ans_3, V[i + 1] + my_abs(V[i]) - 2);
        }

    g << ans_3 << '\n';
    return;
}
void Solve ()
{
    Load();

    if(P == 1)
        Task_1();
    else if(P == 2)
        Task_2();
    else
        Task_3();
    return;
}
int main ()
{
    Read();
    Solve();
    return 0;
}
```

```
/*
3.3 Autor Roxana Timplaru
*/
#include <fstream>
using namespace std;
ifstream fin("seism.in");
ofstream fout("seism.out");
int c,x,nrz1,nrunu,nrz2,k,i,maxi,maxi1,n,nru, a1,a2,b1,b2,a3,a4,b3,b4;
void cerinte()
{
    fin >> n ;
    i = 0;
    while(i < n )
    {
        fin >> x;
        i++;
        if (x == 0)
           nrz1++;
           else break;
    }
    if (nrz1>4)
        maxi1=nrz1-4;
    while(i < n)
    {
        nrunu=1;
        while(i < n)
        {
            fin >> x;
            i++;
            if (x == 1)
                nrunu++;
            else  break;
        }
        nru++;
        nrz2=1;
        while(i < n)
        {
            fin >> x;
            i++;
            if (x == 0)  nrz2++;
            else  break;
        }
        if (nru % 2 == 1)
        {
            a1=nru;    a2=nrunu;
            a3=nrz1;   a4=nrz2;
        }
        else
        {
            b1=nru;
            b2=nrunu;
            b3=nrz1;
            b4=nrz2;
        }

        if (nrz1 >= 2 && nrz2 >= 2)
               {
                   maxi=max(maxi,nrunu);
                   maxi1=max(maxi1,nrunu);
                   k++;
                   if(nrz1>nrz2)
                       nrunu += nrz1-2;
```

```
                    else
                        nrunu += nrz2-2;
                    maxi1 = max(maxi1,nrunu);
                }

        if (b1-a1 == 1 && a3>=2 && b4>=2)
            maxi1=max(maxi1, b2+a2+b3);
           else
                if (a1-b1 == 1 && b3>=2 && a4>=2)
                maxi1=max(maxi1,a2+b2+a3);
         if (nrz2>4)
          maxi1 = max(maxi1, nrz2-4);
        nrz1 = nrz2;
        }
        if (c == 1)
        fout << maxi;
        else
        if (c == 2)  fout << k;
        else
            fout << maxi1;
    }

int main()
{
    fin >> c;
    cerinte();
    return 0;
}
```

```
/*
3.4. Autor Stelian Chichirim
*/
#include <bits/stdc++.h>
using namespace std;
const int Nmax = 1e5;
int v[Nmax + 10];

int main()
{
    freopen("seism.in", "r", stdin);
    freopen("seism.out", "w", stdout);

    int c, n, ans1 = 0, ans2 = 0, ans3 = 0;
    scanf("%d", &c);
    scanf("%d", &n);
    assert(1 <= n && n <= Nmax);
    assert(1 <= c && c <= 3);

    for (int i = 1; i <= n; ++i) {
        scanf("%d", &v[i]);
        assert(0 <= v[i] && v[i] <= 1);
    }
    int last1 = 1, pos1 = 0, last2 = 1, pos2 = 0, nrZero = 0;
    v[n + 1] = 1;
    v[0] = 1;
    for (int i = 1; i <= n; ++i) {
        if (v[i]) {
            nrZero = 0;
            if (!v[i-1] or i == 1) {
                pos2 = pos1;
                last2 = last1;
                pos1 = i;
```

```
                last1 = 1;
            }
            else last1++;
            if (!v[pos1 - 1] && !v[pos1 - 2] && !v[i + 1] && !v[i + 2]) {
                ans1 = max(ans1, last1);
                ans2++;
            }
        }
        else nrZero++;
        if (nrZero > 4) ans3 = max(ans3, nrZero - 4);
        if (!v[i] && !v[i - 1] && pos1 > 0) {
            if (!v[pos1 - 1] && !v[pos1 - 2]) {
                ans3 = max(ans3, i - pos1 - 1);
                ans3 = max(ans3, last1 + pos1 - pos2 - last2 - 2);
            }
            if (pos2 > 0 && !v[pos2 - 1] && !v[pos2 - 2])
             ans3 = max(ans3, pos1 - pos2 + last1);
        }
    }
    if (c == 1) printf("%d", ans1);
    else if (c == 2) printf("%d", ans2);
    else printf("%d", ans3);
    return 0;
}
```

8.3 Soluții - Clasa a VII-a

1. Pătrate

```
/*
1.1. Autor prof. Emanuela Cerchez
*/
#include <fstream>
#define NMAX 202
using namespace std;
ifstream fin("patrate.in");
ofstream fout("patrate.out");
int n, m, S, SMAX, lmax, cmax, lgo, lgv;
int a[NMAX][NMAX];
int SL[NMAX][NMAX];
int SC[NMAX][NMAX];

int main()
{int i, j;
  fin>>n>>m;
  for (i=1; i<=n; i++)
       for (j=1; j<=m; j++)
           {
            fin>>a[i][j];
            SL[i][j]=a[i][j]+SL[i][j-1];
            SC[i][j]=a[i][j]+SC[i-1][j];
           }
   for (i=1; i<=n; i++)
       for (j=1; j<=m; j++)
       {
           ///I
           S=SL[i][j]+SC[n][j]-SC[i][j];
           if (S>SMAX)
             {SMAX=S; lmax=i; cmax=j; lgo=-j; lgv=-(n-i+1);}
           ///II
           S=SL[i][j]+SC[i-1][j];
```

```
            if (S>SMAX) {SMAX=S; lmax=i; cmax=j; lgo=-j; lgv=i;}
            ///III
            S=SL[i][m]-SL[i][j-1]+SC[n][j]-SC[i][j];
            if (S>SMAX)
               {SMAX=S; lmax=i; cmax=j; lgo=m-j+1; lgv=-(n-i+1);}
            ///IV
            S=SL[i][m]-SL[i][j-1]+SC[i-1][j];
            if (S>SMAX) {SMAX=S; lmax=i; cmax=j; lgo=m-j+1; lgv=i;}
        }
   fout<<SMAX<<'\n';
   fout<<lmax<<' '<<cmax<<' '<<lgo<<' '<<lgv<<'\n';
   return 0;
}
```

2. Campionat

```
/*
2.1. Autor prof. Marius Nicoli
*/
#include <fstream>
#include <iostream>
using namespace std;
ifstream fin ("campionat.in");
ofstream fout("campionat.out");
int v[1001];
int a[1001][1001];
int n, i, j, sol[1001], maxim, ok, p, t, d, nr, campioane;
int main () {
    fin>>t;
    fin>>n;
    for (i=1;i<=n;i++)
        fin>>v[i];
    if (t == 1) {
        fin>>d;
        for (;d--;) {
            fin>>i>>j;
            if (a[i][j] == 0) {
                v[i]++;
                v[j]++;
            }
            a[i][j] = 1;
            a[j][i] = 1;
        }
        maxim = v[1];
        for (i=1;i<=n;i++)
            if (v[i] > maxim) {
                maxim = v[i];
                nr = 1;
            }
```

```
            else
                if (v[i] == maxim)
                    nr++;
        for (i=1;i<=n;i++)
            if (v[i] == maxim) {
                fout<<i;
                nr--;
                if (nr!=0)
                    fout<<" ";
                else
                    fout<<"\n";
            }
    }
    else
    {
        /// t = 2
        fin>>d;
        for (;d--;) {
            fin>>i>>j;
            if (a[i][j] == 0) {
                v[i]+=3;
                v[j]+=3;
            }
            a[i][j] = 1;
            a[j][i] = 1;
        }
        for (i=1;i<=n;i++)
        {
            ok = 1;
            for (j=1;j<=n;j++)
                if (i!=j) {
                    p = v[j];
                    if (a[i][j])
                        p-=3;
                    if (p >= v[i]) {
                        ok = 0;
                        break;
                    }
                }
            if (ok) {
                sol[++campioane] = i;
            }
        }
        for (i=1; i<=campioane; i++)
        {
            fout<<sol[i];
            if (i!=campioane)
                   fout<<" ";
            else   fout<<"\n";
        }
        if (campioane == 0)
               fout<<"0\n";
    }
    return 0;
}
```

3. Exclusiv

```
/*
  3.1. Autor prof. Ionel-Vasile Pit-Rada
       Complexitate O(n*n+m*log(n)+m)
*/
#include <fstream>
#define mmax 100005
#define nmax 2005

using namespace std;
ifstream  fin("exclusiv.in");
ofstream fout("exclusiv.out");

int n, v[nmax], s[mmax], poz[nmax], urm[mmax], lmax[nmax],
    stanga[mmax], dreapta[mmax], notfirst[nmax], m, k,
    l, i, j, r1, r2, r, mij, ok, y;

struct secventa
    { int p1,p2; } w[nmax],x;

int main(){
    fin>>m>>n;
    for(i=1; i<=m; i++){
        fin>>s[i];
    }
    for (i=1; i<=n; i++){
        fin>>v[i];
        w[i].p1=v[i];
        w[i].p2=i;
    }
    for (i=1; i<=n-1; i++){
        for (j=i+1; j<=n; j++){
            if ((w[i].p1>w[j].p1)||(w[i].p1==w[j].p1 && w[i].p2>w[j].p2))
                {
                x=w[i];
                w[i]=w[j];
                w[j]=x;
            }
        }
    }
    for (i=2;i<=n;i++){
        if (w[i].p1==w[i-1].p1){
            notfirst[w[i].p2]=1;
        }
    }
    for (i=1; i<=n+1; i++){ poz[i]=m+1;}
    for (j=m; j>=1; j--) { r1=1; r2=n; ok=0;
        while (r1<=r2) {  mij=(r1+r2)/2;
            if (s[j]==w[mij].p1){
                ok=w[mij].p2;
                r2=mij-1;
            }
            else{
                if (s[j]<w[mij].p1) {r2=mij-1; }
                else{r1=mij+1;}
            }
        }
```

```
        if (ok>0) {
            r=ok;
        }
        else {
            r=n+1;
         }
        urm[j]=poz[r];  poz[r]=j;
    }
    for (i=n+1; i>=1; i--){
        lmax[i]=lmax[i+1];
        if (notfirst[i]==1){ continue; }
        for (j=poz[i]; j<=m; j=urm[j]){

            stanga[j]=j; dreapta[j]=j;
            if (stanga[j-1]>0){
                stanga[j]=stanga[j-1];
            }
            if (dreapta[j+1]>0){
                dreapta[j]=dreapta[j+1];
            }
            dreapta[stanga[j]]=dreapta[j];
            stanga[dreapta[j]]=stanga[j];

            y=dreapta[j]-stanga[j]+1;
            if (y>lmax[i]){
                lmax[i]=y;
            }
        }
    }
    for (i=2; i<=n+1; i++){ fout<<lmax[i]<<"\n";}
    fin.close(); fout.close();
    return 0;
}
```

8.4 Soluții - Clasa a VIII-a

1. Coşuri

```
/*
1.1. Autor Ioan - Cristian Pop
*/
#include <cstdio>
#include <algorithm>
using namespace std;
const int NMAX = 1000000;
int f[NMAX + 1];

int main()
{   freopen("cosuri.in", "r", stdin);
    freopen("cosuri.out", "w", stdout);
    int n, s, x, y, t, mn, mx, mn2, mx2, diff;
    bool ok = 1;
    scanf("%d%d", &t, &n);
    if (t == 1) {
        scanf("%d %d", &x, &y);
        mn = mx2 = min(x, y);
        mx = mn2 = max(x, y);
        for (int i = 3; i <= 2 * n; ++i) {
            scanf("%d", &x);
            if (x < mn) {
                mn2 = mn; mn = x;
            } else if (x < mn2)
                mn2 = x;
            if (x > mx) {
                mx2 = mx; mx = x;
            } else if (x > mx2)
                mx2 = x;
        }
        printf("%d %d\n", mn + mn2, mx + mx2);
        return 0;
    }
    scanf("%d", &x);
    mn = mx = x;
    ++f[x];
    for (int i = 2; i <= 2 * n; ++i) {
        scanf("%d", &x);
        ++f[x];
        if (mn > x) mn = x;
        if (mx < x) mx = x;
    }
    s = mn + mx;
    for (int i = 1; i <= (s + 1) / 2; ++i) {
        if (f[i] != f[s - i]) {
            ok = 0;
            break;
        }
        if (f[i] > 0)
            diff = (s - i) - i;
    }
    if (ok == 0)
        printf("NU\n");
```

```
    else
        printf("DA\n%d\n", diff);
    return 0;
}
```

```
/*
1.2. Autor: Bogdan Iordache
*/
#include <bits/stdc++.h>
using namespace std;

const int VMAX = 1000005;

void solve1(const vector<int>& v, ostream& out) {
    int m1 = VMAX, m2 = VMAX, M1 = 0, M2 = 0;
    for (int x : v) {
        if (x < m1) {
            m2 = m1;  m1 = x;
        } else if (x < m2) {
            m2 = x;
        }
        if (x > M1) {
            M2 = M1; M1 = x;
        } else if (x > M2) {
            M2 = x;
        }
    }
    out << m1 + m2 << ' ' << M1 + M2 << '\n';
}

void solve2(vector<int>& v, int N, ostream& out) {
    sort(v.begin(), v.end());
    int sum = v.front() + v.back();
    int diff_min = VMAX;
    for (int i = 0, j = 2 * N - 1; i < j; ++i, --j) {
        if (v[i] + v[j] != sum) {
            out << "NU\n";
            return;
        }
        diff_min = min(diff_min, max(v[i], v[j]) - min(v[i], v[j]));
    }
    out << "DA\n" << diff_min << '\n';
}

int main() {
    ifstream cin("cosuri.in");
    ofstream cout("cosuri.out");
    int T, N;
    cin >> T >> N;
    vector<int> v(2 * N);
    for (auto& it : v) cin >> it;
    if (T == 1)  solve1(v, cout);
    else
          solve2(v, N, cout);
    return 0;
}
```

```
/*
1.3. Autor: Raluca Costineanu O(n)
*/
#include <bits/stdc++.h>
using namespace std;
ifstream f("cosuri.in");
ofstream g("cosuri.out");

int n, C;
int ap[1000001];

int main()
{
    int i, mn1, mn2, mx1, mx2, x, diff=(1<<30);
    long long sum=0;
    mn1=mn2=1000001;
    mx1=mx2=0;
    f>>C>>n;
    int N=2*n;
    for(i=1; i<=N;i++)
    {
        f>>x;ap[x]++;sum=sum+x;
        if(x<mn1)mn2=mn1, mn1=x;
        else if(x<mn2)mn2=x;
        if(x>mx1)mx2=mx1, mx1=x;
        else if(x>mx2)mx2=x;
    }
    if(C==1)g<<mn1+mn2<<' '<<mx1+mx2<<'\n';
    else
    {   if(sum%n){g<<"NU"<<'\n';return 0;}
        sum/=n;
        for(i=1;i<=mx1;i++)
            while(ap[i])
            {
                if(sum-2*i<diff)
                diff=sum-2*i;
                ap[i]--;
                if(ap[sum-i]==0){g<<"NU"<<'\n';return 0;}
                else ap[sum-i]--;
            }
        g<<"DA"<<'\n'<<diff<<'\n';
    }
    return 0;
}
```

2. Cartofi

```
/*
2.1. Autor: Bogdan Iordache
COMPLEXITATE: O(1)
*/
#include <bits/stdc++.h>
using namespace std;
int vals[65], mat[65][65];
long long sum[65], cnt0[65];
int main() {
    ifstream cin("cartofi.in");
    ofstream cout("cartofi.out");
    int task;
    assert(cin >> task);
    assert(1 <= task && task <= 3);
    int N, M;
    assert(cin >> N >> M);
    assert(2 <= N && N <= 500000000);
    assert(3 <= M && M <= 1000000000);
    assert(N <= M);
    vals[0] = vals[1] = 1;
    for (int i = 2; i < 60; ++i) {
        vals[i] = vals[i - 2] + vals[i - 1];
        if (vals[i] >= 10) vals[i] -= 10;
    }
    for (int i = 1; i <= 60 && i <= N; ++i) {
        for (int j = 1; j <= 60 && j <= M; ++j) {
            if (i % 2) {
                int pos = (1LL * (i - 1) * M + j - 1) % 60;
                mat[i][j] = vals[pos];
            } else {
                int pos = (1LL * (i - 1) * M + M - j) % 60;
                mat[i][j] = vals[pos];
            }
        }
    }

    for (int i = 1; i <= 60 && i <= M; ++i) {
        for (int j = 1; j <= 60 && j <= N; ++j) {
            sum[i] += mat[j][i] * (N / 60 + (j <= N % 60));
            cnt0[i] += (mat[j][i] == 0) * (N / 60 + (j <= N % 60));
        }
    }

    if (task == 1) {
        long long sol = 0;
        for (int i = 1; i <= 60 && i <= M; ++i)
            sol += cnt0[i] * (M / 60 + (i <= M % 60));
```

```
        cout << sol << '\n';
        char c;
        assert(!(cin >> c));
        return 0;
    }

    if (task == 2) {
        long long best = 0;
        for (int lef = 1, rig = N; lef <= 60 && rig <= M; ++lef, ++rig) {
            long long curr = 0;
            for (int i = 1; i <= 60 && i <= M; ++i) {
              int x = i > lef ? 0 : (lef - i) / 60 + ((lef - i) % 60 != 0);
              int y = i > rig ? -1 : (rig - i) / 60;
              int cnt = y - x + 1;
              curr += cnt * sum[i];
            }
            best = max(best, curr);
        }
        cout << best << '\n';
    }
    else
    {
        int Q;
        assert(cin >> Q);
        assert(1 <= Q && Q <= 100000);
        while (Q--) {
            int lef, rig;
            assert(cin >> lef >> rig);
            assert(1 <= lef && lef <= rig && rig <= M);
            long long curr = 0;
            for (int i = 1; i <= 60 && i <= M; ++i) {
              int x = i > lef ? 0 : (lef - i) / 60 + ((lef - i) % 60 != 0);
              int y = i > rig ? -1 : (rig - i) / 60;
              int cnt = y - x + 1;
              curr += cnt * sum[i];
            }
            cout << curr << '\n';
        }
    }
    char c;
    assert(!(cin >> c));
    return 0;
}
```

```
/*
2.3. Autor: Ioan Cristian Pop
*/
#include <cstdio>
#include <string>
using namespace std;

const int R = 60;

int sp_0[R + 1], f[R + 1];
int a[R + 1][R + 1], sp_c[R + 1][R + 1];
long long s_c[R + 1], sp[R + 1];

int main()
{
    int t, N, M;

    freopen("cartofi.in", "r", stdin);
    freopen("cartofi.out", "w", stdout);

    scanf("%d %d %d", &t, &N, &M);
    f[1] = f[2] = 1;
    for (int i = 3; i <= R; ++i) {
        f[i] = (f[i - 1] + f[i - 2]) % 10;

        sp_0[i] = sp_0[i - 1];
        if (f[i] == 0) {
            ++sp_0[i];
        }
    }

    if (t == 1) {
        long long nr, c, r;
        nr = 1LL * N * M;
        c = nr / R;
        r = nr % R;
        printf("%lld\n", 1LL * c * sp_0[R] + sp_0[r]);
        return 0;
    }
    for (int i = 1; i <= R && i <= N; ++i) {
        for (int j = 1; j <= R && j <= M; ++j) {
            long long poz;
            if (i % 2 == 1) {
                poz = (1LL) * (i - 1) * M + j;
            } else {
                poz = (1ll) * (i - 1) * M + (M - j + 1);
            }
            poz %= R;
            if (poz == 0) {
                poz = R;
            }
            a[i][j] = f[poz];
            sp_c[i][j] = sp_c[i - 1][j] + a[i][j];
        }
```

```
    }

    for (int j = 1; j <= R && j <= M; ++j) {
        s_c[j] = 1LL * sp_c[R][j] * (N / R) + sp_c[N % R][j];
        sp[j] = sp[j - 1] + s_c[j];
    }


    if (t == 2) {
        long long s = sp[R] * (N / R) + sp[N % R];
        long long ans = s;
        for (int j = 1; j <= R && j <= M - N; ++j) {
            s -= s_c[j];
            int k = (j + N) % R;
            if (k == 0) {
                k = R;
            }
            s += s_c[k];
            if (s > ans) {
                ans = s;
            }
        }
        printf("%lld\n", ans);
    } else {
        int Q;
        for (scanf("%d", &Q); Q > 0; --Q){
            int x, y, dif, st, dr;
            scanf("%d %d", &x, &y);
            dif = (y - x) + 1;
            st = x % R;
            if (st == 0) {
                st = R;
            }
            dr = y % R;
            if (dr == 0) {
                dr = R;
            }

            long long rez = 1LL * (dif / R) * sp[R];
            if (st <= dr && (st != 1 || dr != 60)) {
                rez += sp[dr] - sp[st - 1];
            } else {
                int p = dr + 1;
                if (p > 60) {
                    p -= 60;
                }
                if (p != st) {
                    rez += sp[R] - sp[st - 1] + sp[dr];
                }
            }
            printf("%lld\n", rez);
        }
    }
    return 0;
}
```

```
/*
2.3. Autor: Flavius Boian
*/
#include <bits/stdc++.h>
using namespace std;
ifstream f("cartofi.in");
ofstream g("cartofi.out");
pair<int, int> fibo(long long n)
{
    if (n == 0)
        return {0, 1};
    auto p = fibo(n >> 1ll);
    int ax = ((p.second << 1) - p.first + 10) % 10;
    int c = (p.first * ax) % 10;
    int d = (p.first * p.first + p.second * p.second) % 10;
    if (n & 1)
        return {d, (c + d) % 10};
    return {c, d}; }
int n, m;
long long cols[61];
long long query(int);
void build();
void solve1();
void solve2();
void solve3();
int32_t main()
{
    int task;
    f >> task >> n >> m;
    if (task == 1)
        solve1();
    else if (task == 2)
        solve2();
    else
        solve3();
    return 0;
}
void solve1()
{
    long long ct = 0, ax = (1ll * n * m) / 60ll, ap = (1ll * n * m) % 60ll;
    long long rez = 0;
    for (int i = 0, a = 0, b = 1, c; i < 60; i++)
    {
        ct += !b;
        if (i < ap)
            rez += !b;
        c = (a + b) % 10;
        a = b;
        b = c;
    }
    rez += 1ll * ax * ct;
    g << rez;
}
```

```
void solve2()
{
    build();
    long long rez = 0, ax;
    for (int i = n; i <= min(n + 60, m); i++)
    {
        ax = query(i) - query(i - n);
        rez = max(rez, ax);
    }
    g << rez;
}

void solve3()
{
    build();
    int q, st, dr;
    for (f >> q; q; q--)
    {
        f >> st >> dr;
        g << (long long)query(dr) - query(st - 1) << '\n';
    }
}

void build()
{
    long long aux[61], prv = 0, md = n % 60, ct = n / 60, ind;
    memset(aux, 0, sizeof aux);
    for (int i = 0, nr; i < min(n, 60); i++, prv += m)
        for (int j = 1; j <= min(m, 60); j++)
        {
            ind = prv;
            if (i & 1)
                ind += m - j + 1;
            else
                ind += j;
            nr = fibo(ind).first;
            cols[j] += nr;
            if (i < md)
                aux[j] += nr;
        }
    for (int i = 1; i <= min(m, 60); i++)
        cols[i] = 1ll * cols[i] * ct + aux[i], cols[i] += cols[i - 1];
}

long long query(int poz)
{
    long long rez = 1ll * (poz / 60) * cols[60];
    rez += cols[poz % 60];
    return rez;
}
```

3. Tunel

```
/*
3.1.Autor Carmen Mincă
*/
#include <fstream>
using namespace std;
ifstream f("tunel.in");
ofstream g("tunel.out");
int C, N,NN,M,X,Y,  lmin=-1, t;
char a[3000][20020];

void date()
{
    int i,j,p,x;
    f>>C>>N>>M>>X;
    NN=2*N-1;
    for(j=2;j<=NN;j=j+2)
    {
        f>>p;
        for(i=1;i<=p;++i)
           { f>>x; a[j][x]=1; }
    }
    NN=2*N-1;
}
void drum1(int j, int i, int k, int t)
{
    Y=i;
    if(j==M)
    {
        if(i==NN) {
                  if(lmin==-1)lmin=k;
                  else lmin=min(lmin,k);
                 }
        else
        if(i<NN-2);
        else
            if(i==NN-2)
                if(a[NN-1][M]==1) Y=NN;
                else Y=(i+1)/2;
    }
    else
    {
        if(a[i+1][j]==1)drum1(j+1,i+2,k+3,t);
        else
            if(a[i-1][j]==1)drum1(j+1,i-2,k+3,t);
            else
                drum1(j+1,i,k+1,t);
    }
}
void drum2(int j, int i, int k)
{
    if(j==1)
    {
        if(lmin==-1)lmin=k;
        else lmin=min(lmin,k);
    }
```

```
    else
    {
        if(a[i-1][j]==1)drum2(j-1,i-2,k+3);
        else
            if(a[i+1][j]==1)drum2(j-1,i+2,k+3);
            else
                drum2(j-1,i,k+1);
    }
}

int main()
{ int i,j,ok,k;
  date();
  if(C==1)
    {
        drum1(1,2*X-1,1,X);
        g<<(Y+1)/2;
    }
   else
    {
      drum2(M-1,NN,2);
      if(a[NN-1][M]==1) drum2(M-1,NN-2,4);
      g<<lmin;
    }
    g<<'\n';
    return 0;
}
```

```
/*
3.2. Autor: Diana Ghinea
 * iterativ + căutare binară
 * O(M log N) timp
 * O(max pasaje) memorie suplimentara
*/


#include <iostream>
#include <fstream>
#include <vector>

using namespace std;

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

const int NMAX = 1000;
const int MMAX = 200000;

const int SUS = 0;
const int JOS = 1;

const int STANGA = 2;
const int DREAPTA = 3;

int c, n, m, x;

vector <int> pasaj_in_jos[MMAX + 2];

void citeste() {

    f >> c;
    f >> n >> m >> x;
```

```
    for (int i = 1; i < n; i++) {
        int p;
        f >> p;
        for (int j = 1; j <= p; j++) {
            int poz;
            f >> poz;
            pasaj_in_jos[poz].push_back(i);
        }
    }
}
bool am_pasaj_in_jos(int i, int j) {
    int st = 0, dr = pasaj_in_jos[j].size() - 1;

    while (st <= dr) {
        int m = (st + dr) / 2;
        if (pasaj_in_jos[j][m] == i) return true;
        if (pasaj_in_jos[j][m] < i) st = m + 1;
        else dr = m - 1;
    }
    return false;
}
bool am_pasaj_in_sus(int i, int j) {
    return am_pasaj_in_jos(i - 1, j);
}
void c1() {
    int i = x, j = 1;
    int vin_din = STANGA;
    while (j <= m && !(i == n && j == m)) {
        if (vin_din == STANGA) {
            if (am_pasaj_in_sus(i, j)) {
                i--;
                vin_din = JOS;
                continue;
            }
            if (am_pasaj_in_jos(i, j)) {
                i++;
                vin_din = SUS;
                continue;
            }
        }
        j++;
        vin_din = STANGA;
    }
    g << i << '\n';
}
int traseu_c2(int i, int j, int vin_din, int pasaje) {
    while (j > 0) {
        if (vin_din == DREAPTA) {
            if (am_pasaj_in_sus(i, j)) {
                i--;
                vin_din = JOS;
                pasaje++;
                continue;
            }
            if (am_pasaj_in_jos(i, j)) {
                i++;
                vin_din = SUS;
                pasaje++;
                continue;
            }
        }
        j--;
```

```
        vin_din = DREAPTA;
    }

    return m + 2 * pasaje;
}

void c2() {
    int lungime_minima = traseu_c2(n, m - 1, DREAPTA, 0);
    if (am_pasaj_in_sus(n, m))
        lungime_minima = min(lungime_minima, traseu_c2(n - 1, m, JOS, 1));
    g << lungime_minima << '\n';
}

int main() {

    citeste();
    if (c == 1) c1();
    else c2();
    return 0;
}
```

```
/**
3.3. Autor: Diana Ghinea
 * iterativ + matrice N * M
 * O(M) timp
 * O(N * M) memorie suplimentara
**/

#include <iostream>
#include <fstream>
#include <vector>

using namespace std;
ifstream f("tunel.in");
ofstream g("tunel.out");

const int NMAX = 1000;
const int MMAX = 200000;
const int SUS = 0;
const int JOS = 1;
const int STANGA = 2;
const int DREAPTA = 3;

int c, n, m, x;
bool pasaj_in_jos[NMAX + 2][MMAX + 2];

void citeste() {

    f >> c;
    f >> n >> m >> x;
    for (int i = 1; i < n; i++) {
        int p;
        f >> p;
        for (int j = 1; j <= p; j++) {
            int poz;
            f >> poz;
            pasaj_in_jos[i][poz] = true;
        }
    }
}
```

```
inline bool am_pasaj_in_jos(int i, int j) {
    return pasaj_in_jos[i][j];
}
inline bool am_pasaj_in_sus(int i, int j) {
    return pasaj_in_jos[i - 1][j];
}
void c1() {
    int i = x, j = 1;
    int vin_din = STANGA;
    while (j <= m && !(i == n && j == m)) {
        if (vin_din == STANGA) {
            if (am_pasaj_in_sus(i, j)) {
                i--;
                vin_din = JOS;
                continue;
            }
            if (am_pasaj_in_jos(i, j)) {
                i++;
                vin_din = SUS;
                continue;
            }
        }
        j++;
        vin_din = STANGA;
    }
    g << i << '\n';
}
int traseu_c2(int i, int j, int vin_din, int pasaje) {
    while (j > 0) {
        if (vin_din == DREAPTA) {
            if (am_pasaj_in_sus(i, j)) {
                i--;
                vin_din = JOS;
                pasaje++;
                continue;
            }
            if (am_pasaj_in_jos(i, j)) {
                i++;
                vin_din = SUS;
                pasaje++;
                continue;
            }
        }
        j--;
        vin_din = DREAPTA;
    }
    return m + 2 * pasaje;
}
void c2() {
    int lungime_minima = traseu_c2(n, m - 1, DREAPTA, 0);
    if (am_pasaj_in_sus(n, m))
        lungime_minima = min(lungime_minima, traseu_c2(n - 1, m, JOS, 1));
    g << lungime_minima << '\n';
}
int main() {
    citeste();
    if (c == 1) c1();
    else c2();
    return 0;
}
```

```
/**
3.4. Autor: Diana Ghinea
 * recursivitate + matrice N * M
 * O(M) timp
 * O(N * M) memorie suplimentara
*/

#include <iostream>
#include <fstream>

using namespace std;

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

const int NMAX = 1000;
const int MMAX = 200000;
const int SUS = 0;
const int JOS = 1;
const int STANGA = 2;
const int DREAPTA = 3;

int c, n, m, x;
bool pasaj_in_jos[NMAX + 2][MMAX + 2];

void citeste() {

    f >> c;
    f >> n >> m >> x;
    for (int i = 1; i < n; i++) {
        int p;
        f >> p;
        for (int j = 1; j <= p; j++) {
            int poz;
            f >> poz;
            pasaj_in_jos[i][poz] = true;
        }
    }
}

inline bool am_pasaj_in_jos(int i, int j) {
    return pasaj_in_jos[i][j];
}

inline bool am_pasaj_in_sus(int i, int j) {
    return pasaj_in_jos[i - 1][j];
}

int DFS_c1(int i, int j, int vin_din) {
    if (j == m + 1 || (i == n && j == m)) return i;
    if (vin_din == STANGA) {
        if (am_pasaj_in_sus(i, j))
            return DFS_c1(i - 1, j, JOS);
        if (am_pasaj_in_jos(i, j))
            return DFS_c1(i + 1, j, JOS);
    }
    return DFS_c1(i, j + 1, STANGA);
}
void c1() {
    g << DFS_c1(x, 1, STANGA) << '\n';
}
```

```
int DFS(int i, int j, int vin_din, int l) {

    if (j == 0) return l;
    if (vin_din == DREAPTA) {
        if (am_pasaj_in_sus(i, j))
            return DFS(i - 1, j, JOS, l + 2);
        if (am_pasaj_in_jos(i, j))
            return DFS(i + 1, j, SUS, l + 2);
    }
    return DFS(i, j - 1, DREAPTA, l + 1);
}

void c2() {

    int lungime_minima = DFS(n, m - 1, DREAPTA, 1);
    if (am_pasaj_in_sus(n, m))
        lungime_minima = min(lungime_minima, DFS(n - 1, m, JOS, 2));
    g << lungime_minima << '\n';
}

int main() {

    citeste();
    if (c == 1) c1();
    else c2();
    return 0;
}
```

```
/**
 3.5 Autor: Diana Ghinea
 * recursivitate + set
 * O(M log N) timp
 * O(max pasaje) memorie suplimentara

*/

#include <iostream>
#include <fstream>
#include <set>

using namespace std;

ifstream f("tunel2.in");
ofstream g("tunel2.out");

const int NMAX = 1000;
const int MMAX = 20000;

const int SUS = 0;
const int JOS = 1;
const int STANGA = 2;
const int DREAPTA = 3;

int c, n, m, x;
set <int> pasaj_in_jos[MMAX + 2];

void citeste() {
      f >> c;
      f >> n >> m >> x;
```

```
	for (int i = 1; i < n; i++) {
		int p;
		f >> p;
		for (int j = 1; j <= p; j++) {
			int poz;
			f >> poz;
			pasaj_in_jos[poz].insert(i);
		}
	}
}
bool am_pasaj_in_jos(int i, int j) {

	return pasaj_in_jos[j].find(i) != pasaj_in_jos[j].end();
}
bool am_pasaj_in_sus(int i, int j) {

	return am_pasaj_in_jos(i - 1, j);
}
int DFS_c1(int i, int j, int vin_din) {

	if (j == m + 1 || (i == n && j == m)) return i;

	if (vin_din == STANGA) {
		if (am_pasaj_in_sus(i, j))
			return DFS_c1(i - 1, j, JOS);
		if (am_pasaj_in_jos(i, j))
			return DFS_c1(i + 1, j, JOS);
	}

	return DFS_c1(i, j + 1, STANGA);
}
void c1() {

	g << DFS_c1(x, 1, STANGA) << '\n';
}
int DFS(int i, int j, int vin_din, int l) {
	if (j == 0) return l;

	if (vin_din == DREAPTA) {
		if (am_pasaj_in_sus(i, j))
			return DFS(i - 1, j, JOS, l + 2);
		if (am_pasaj_in_jos(i, j))
			return DFS(i + 1, j, SUS, l + 2);
	}

	return DFS(i, j - 1, DREAPTA, l + 1);
}
void c2() {

	int lungime_minima = DFS(n, m - 1, DREAPTA, 1);
	if (am_pasaj_in_sus(n, m))
		lungime_minima = min(lungime_minima, DFS(n - 1, m, JOS, 2));
	g << lungime_minima << '\n';
}
int main() {

	citeste();
	if (c == 1) c1();
	else c2();
	return 0;
}
```

```
/*
3.5. AUTOR: Bogdan Iordache
 * COMPLEXITATE: O(M + sum(P_i))
*/
#include <bits/stdc++.h>
using namespace std;

const int NMAX = 1005;

vector<int> pasages[NMAX];
int idx[NMAX];

int main() {
    ifstream cin("tunel.in");
    ofstream cout("tunel.out");

    int task;
    assert(cin >> task);
    assert(task == 1 || task == 2);

    int N, M, X;
    assert(cin >> N >> M >> X);
    assert(3 <= N && N <= 1000);
    assert(4 <= M && M <= 20000);
    assert(1 <= X && X <= N);

    int sum_P = 0;
    for (int i = 1; i < N; ++i) {
        int P;
        assert(cin >> P);
        assert(1 <= P && P <= M - 2);
        sum_P += P;
        pasages[i].resize(P);
        for (int j = 0; j < P; ++j) {
            assert(cin >> pasages[i][j]);
            assert(1 < pasages[i][j] && pasages[i][j] <= M);
            if (i < N - 1) assert(pasages[i][j] < M);
            if (j > 0) assert(pasages[i][j - 1] < pasages[i][j]);
        }
    }
    assert(sum_P <= 150000);
    for (int i = 1; i < N - 1; ++i) {
        size_t p1 = 0, p2 = 0;
        while (p1 < pasages[i].size() && p2 < pasages[i + 1].size()) {
            assert(pasages[i][p1] != pasages[i + 1][p2]);
            if (pasages[i][p1] < pasages[i + 1][p2])
                p1++;
            else
                p2++;
        }
    }
```

```
    if (task == 1) {
        int curr_i = X, curr_j = 1;
        while (curr_j < M) {
            curr_j++;
            if (curr_j == M && curr_i == N) break;
            if (curr_i > 1) {
                while (idx[curr_i - 1] < (int)pasages[curr_i - 1].size() &&
                       pasages[curr_i - 1][idx[curr_i - 1]] < curr_j)
                    idx[curr_i - 1]++;
                if (idx[curr_i - 1] < (int)pasages[curr_i - 1].size() &&
                    pasages[curr_i - 1][idx[curr_i - 1]] == curr_j) {
                    curr_i--;
                    continue;
                }
            }
            if (curr_i < N) {
                while (idx[curr_i] < (int)pasages[curr_i].size() &&
                       pasages[curr_i][idx[curr_i]] < curr_j)
                    idx[curr_i]++;
                if (idx[curr_i] < (int)pasages[curr_i].size() &&
                    pasages[curr_i][idx[curr_i]] == curr_j)
                    curr_i++;
            }
        }
        cout << curr_i << '\n';
        char ch;
        assert(!(cin >> ch));
        return 0;
    }
    vector<pair<int, int>> lines;
    lines.emplace_back(N, 1);
    if (!pasages[N - 1].empty() && pasages[N - 1].back() == M)
        lines.emplace_back(N - 1, 3);
    for (int col = M - 1; col; --col) {
        vector<pair<int, int>> new_lines;
        for (auto&& p : lines) {
            int line = p.first;
            int cost = p.second;
            while (line > 1 && !pasages[line - 1].empty() &&
                   pasages[line - 1].back() > col)
                pasages[line - 1].pop_back();
            if (line > 1 && !pasages[line - 1].empty() &&
                pasages[line - 1].back() == col) {
                new_lines.emplace_back(line - 1, cost + 3);
                continue;
            }
            while (line < N && !pasages[line].empty() &&
                   pasages[line].back() > col)
                pasages[line].pop_back();
            if (line < N && !pasages[line].empty() &&
                pasages[line].back() == col) {
                new_lines.emplace_back(line + 1, cost + 3);
                continue;
            }
```

```
            new_lines.emplace_back(line, cost + 1);
        }
        lines = new_lines;
    }
    int best = 0x3f3f3f3f;
    for (auto&& p : lines) {
        // cerr << p.first << ' ' << p.second << '\n';
        best = min(best, p.second);
    }
    cout << best << '\n';
    char ch;
    assert(!(cin >> ch));
    return 0;
}
```

```
/*
3.6. Autor: Stelian Ciurea
*/
//parcurge in sensul iesire->intrare cele doua rute posibile,
//structura de date care retine o matrice cu 2n linii
#include <iostream>
#include <fstream>
#include <algorithm>
#define lin  first
#define lungime  second
#define nmax 1001
#define mmax 20002
using namespace std;
ifstream in("tunel.in");
ofstream out("tunel.out");
int cer, n, m, x, nrp;
bool directie, oriz = 0, vert = 1;
bool mat[nmax + 2][mmax + 2];
void citeste()
{
      int coloana;
      in >> cer >> n >> m >> x;
      for (int i = 1; i <= n - 1; i++)
      {
            in >> nrp;
            for (int j = 1; j <= nrp; j++)
            {
                  in >> coloana;
                  mat[i][coloana] = 1;
            }
      }
}
pair<int, int> parcurgeredir(int intrare) //pt cerinta 1
{
      int nrpasi = 1;
      int lincrt = intrare, colcrt = 1;
      while (colcrt < m)
      {//nu merge decat pe liniile impare!!
            if (directie == vert)  //a venit pe verticala
```

```
		{
			colcrt++;
			directie = oriz;
			nrpasi++;
		}
		else //a venit pe orizontala
		{
			int linup = lincrt / 2;
			int lindown = linup + 1;
			if (mat[linup][colcrt] == 1) //pasaj in sus
			{
				lincrt = lincrt - 2;
				nrpasi += 2;
				directie = vert;
			}
			else
				if (mat[lindown][colcrt] == 1) //pasaj in jos
				{
					lincrt = lincrt + 2;
					nrpasi += 2;
					directie = vert;
				}
				else  //fara pasaj
				{
					colcrt++;
					nrpasi++;
				}
		}
	}
	return { lincrt,nrpasi };
}
pair<int, int> parcurgereinv(int intrare) //pentru cerinta 2
{
	int lincrt = intrare, colcrt = m;
	int nrpasi = 1;
	while (colcrt > 1)
	{
		if (directie == vert)  //a venit pe verticala
		{
			colcrt--;
			directie = oriz;
			nrpasi++;
		}
		else //a venit pe orizontala
		{
			int linup = lincrt / 2;
			int lindown = linup + 1;
			if (mat[linup][colcrt] == 1) //pasaj in sus
			{
				lincrt = lincrt - 2;
				nrpasi += 2;
				directie = vert;
			}
			else
```

```
                        if (mat[lindown][colcrt] == 1) //pasaj in jos
                        {
                              lincrt = lincrt + 2;
                              nrpasi += 2;
                              directie = vert;
                        }
                        else  //fara pasaj
                        {
                              colcrt--;
                              nrpasi++;
                        }
            }
      }
      return { lincrt,nrpasi };
}

int main()
{
      citeste();
      if (cer == 1)
      {
            pair<int, int> iesire = parcurgeredir(2 * x - 1);
                        if (iesire.lin == 2 * n - 3 and mat[n - 1][m] == 1)
                        {
                              cout << n << endl;
                              out << n << endl;
                        }
                        else
                        {
                              cout << iesire.lin / 2 + 1 << endl;
                              out << iesire.lin / 2 + 1 << endl;
                        }
      }

      else
      {
            int lungmin;
            if (mat[n - 1][m] == 1)
            {
                  mat[n - 1][m] = 0;
                  pair <int, int> iesire1 = parcurgereinv(2 * n - 1);
                  lungmin = iesire1.lungime;
                  pair <int, int> iesire2 = parcurgereinv(2 * n - 3);
                  lungmin = min(iesire1.lungime, iesire2.lungime + 2);
            }

            else
            {
                  pair <int, int> iesire1 = parcurgereinv(2 * n - 1);
                  lungmin = iesire1.lungime;
            }
            out << lungmin << endl;
      }
}
```

```
/*
3.7. Autor: Stelian Ciurea
*/
//determina drumurile din toate cele n tuneluri, sensul intrare->iesire
//structura de date utilizata doar pentru a retine amplasarea portalurilor
//cautare binara a urmatorului portal din tunelul curent
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#define lin  first
#define nrpas  second
#define nmax 100001
using namespace std;

ifstream in("tunel.in");
ofstream out("tunel.out");
int cer, n, m, x, nrp;
vector<int> pasaje[nmax + 2];
void citeste()
{
	in >> cer >> n >> m >> x;
	int coloana;
	for (int i = 1; i <= n - 1; i++)
	{
		in >> nrp;
		for (int j = 1; j <= nrp; j++)
		{
			in >> coloana;
			pasaje[i].push_back(coloana);
		}
	}
}
pair<int, int> parcurgere(int intrare)
{
	int nrpasaje = 0;
	int lincrt = intrare, colcrt = 1;
	while (colcrt < m)
	{
			int linup =lincrt-1;
			int lindown =lincrt;
			int colsus;
			if (linup == 0) colsus = m + 1;
			else
			{
				auto p = upper_bound(pasaje[linup].begin(),
				pasaje[linup].end(), colcrt);
				if (p == pasaje[linup].end())
					colsus = m + 1;
				else
					colsus = *p;
			}
			int coljos;
			if (lindown == n) coljos = m + 1;
```

```
                  else
                  {
                        auto p = upper_bound(pasaje[lindown].begin(),
                        pasaje[lindown].end(), colcrt);
                        if (p == pasaje[lindown].end())
                              coljos = m + 1;
                        else
                              coljos = *p;
                  }
                  if (colsus >= m and coljos >= m)
                        return { lincrt,nrpasaje };
                  if (colsus < coljos)
                  {
                        lincrt = lincrt - 1;
                        nrpasaje++;
                        colcrt = colsus;
                  }
                  else
                  {
                        lincrt = lincrt + 1;
                        nrpasaje++;
                        colcrt = coljos;
                  }
      }
}
int main()
{
      citeste();
      if (cer == 1)
      {
            pair<int, int> iesire = parcurgere(x);
            if (iesire.lin == n-1 and *pasaje[n-1].rbegin() == m)
                  out << n << endl;
            else
                  out << iesire.lin  << endl;
      }
      else
      {
            int lungmin = 1000000000;
            for (int i = 1; i <= n; i++)
            {
                  pair <int, int> iesire = parcurgere(i);

                  if (iesire.lin == n)
                        lungmin = iesire.nrpas * 2 + m;
                  if (iesire.lin == n - 1 and *pasaje[n - 1].rbegin() == m)
                  {
                        lungmin = min(lungmin, m + 2 * iesire.nrpas + 2);
                  }
            }
            out << lungmin << endl;
      }
}
```

8.5 Soluții - Clasa a IX-a

1. Ascdesc

```
#include <fstream>
using namespace std;
ifstream cin ("ascdesc.in");
ofstream cout ("ascdesc.out");
const int NR = 1e5 + 5;
typedef long long ll;
ll v[NR], dif[NR], creste[NR];
int main()
{
    ll a,b,c,i,n,j,suma=0,Max=0;
    cin>>n;
    for(i = 0; i < n; i++)
        cin>>v[i];
    for(i = 0; i < n; i++)
    {
        if(Max < v[i])
        {
            Max = v[i];
        }
        creste[i] = Max - v[i];
        suma += Max - v[i];
        dif[i] += suma;
    }
    suma = 0;
    Max = 0;
    for(i = n - 1; i >= 0; i--)
    {
        if(Max < v[i])
        {
            Max = v[i];
        }
        creste[i] = min(creste[i],Max-v[i]);
        suma += Max - v[i];
        dif[i] += suma;
    }
    for(i = 0; i < n; i++)
    {
        cout<<dif[i] - creste[i]<<" ";
    }
    return 0;
}
```

2. Aproape

```
/* Autor: Tulba-Lecu Theodor-Gabriel
 * Complexitate: O(log(N))
 * Implementare: Formula
 */
#include <bits/stdc++.h>
using namespace std;
int main() {
    bool first = true;
    int N, V, mod1 = 0, mod2 = 0, mod3 = 0;
    FILE *in = fopen("aproape.in", "r");
    FILE *out = fopen("aproape.out", "w");
    fscanf(in, "%d %d", &V, &N);
    do {
        if (N < 10) {
            if (N <= 1 || N == 9) {
                if (N == 1 && first) {
                    mod2++;
                } else {
                    mod1++;
                }
            } else
                mod2++;

            if (N <= 2 || N >= 8) {
                if (N == 2 && first) {
                    mod3 += 2;
                } else {
                    mod3++;
                }
            } else
                mod3 += 2;
        } else {
            if (N % 10 == 0 || N % 10 == 9)
                mod1++;
            else
                mod2++;
            if (N % 10 <= 1 || N % 10 >= 8)
                mod3++;
            else
                mod3 += 2;
        }
        N /= 10;
        first = false;
    } while (N > 0);
    if (V == 1) {
        fprintf(out, "%d\n", mod1 + mod2);
    } else if (V == 2)
        fprintf(out, "%d\n", mod1 + 2 * mod2);
    else {
        // {numarul n} + numerele in care schimbam o singrua cifra cu 2 +
        // numarul de numere in care schimbam 2 cifre (perechile ordonate
        // cu o singura optiune + 2 * perechile ordonate prima cu o optiune, a
        // doua cu 2 optiuni + 4 * perechile ordonate de cifre cu 2 optiuni)
        fprintf(out, "%d\n",
                1 + mod3 + mod1 * (mod1 - 1) / 2 + 2 * mod2 * (mod2 - 1) +
                    2 * mod1 * mod2);
    }
    return 0;
}
```

3. Cochilie

```
/**
 3.1.Autor: Alexandru Petrescu
 Sursa oficiala de 100 de puncte, fara functii si fara vectori!
**/
#include <stdio.h>
#define PRINT_MULTIPLES_OF_4 0
#define PRINT_LABEL 1
#define PRINT_OTHER_EVEN 2
#define PRINT_NEWLINE 3

int main() {
    FILE *fin = fopen("cochilie.in", "r");
    FILE *fout = fopen("cochilie.out", "w");
    int c; /// task index
    int n; /// shell order
    int p; /// row number
    fscanf(fin,"%d%d", &c, &n);
    if (c == 2) {
        fscanf(fin,"%d", &p);
    }
    int i, above = 0, previous = 0, current = 1;
    for (i = 2; i <= n; i++) {
        int aux = current;
        current += previous;
        previous = aux;
        if (i % 4 == 1) {
            above += current;
        }
    }
    if (c == 1) {
        if (n % 2 == 0) {
            fprintf(fout,"%d %d\n", current, current + previous);
        } else {
            fprintf(fout,"%d %d\n", current + previous, current);
        }
    } else {
        previous = 0;
        current = 1;
        int label = 1;
        while (label % 2 == 0 || above >= p || p > above + current + previous)
{
            label++;
            int aux = current;
            current += previous;
            previous = aux;
            if (label % 4 == 1) {
                above -= current;
            }
        }

        /// label is the only odd number in the output
        /// all even numbers greater than label are to be printed as well

        int phase = PRINT_MULTIPLES_OF_4;
        int k = n / 4 * 4; /// largest multiple of 4 <= n
```

```
        while (phase != PRINT_NEWLINE)
        {
            int toPrint = 0;
            switch (phase) {
            case PRINT_MULTIPLES_OF_4:
                if (k > label)
                {
                    toPrint = k;
                    k -= 4;
                } else {
                    phase = PRINT_LABEL;
                    toPrint = label;
                    phase = PRINT_OTHER_EVEN;
                    k = label + label % 4;
                    /// smallest multiple of 4 plus 2 > label
                }
                break;
            case PRINT_OTHER_EVEN:
                if (k <= n) {
                    toPrint = k;
                    k += 4;
                } else {
                    phase = PRINT_NEWLINE;
                }
            }

            if (toPrint != 0)
            {
                previous = 0;
                current = 1;
                for (i = 2; i <= toPrint; i++) {
                    int aux = current;
                    current += previous;
                    previous = aux;
                }
                for (i = 0; i < current; i++) {
                    fprintf(fout,"%d ", toPrint);
                }
            }
        }
    }
    return 0;
}
```

```
/*
3.2. Autor Adrian Panaete
*/

#include <bits/stdc++.h>

using namespace std;
ifstream f("cochilie.in");
ofstream g("cochilie.out");

int n,c,p,F[30],L,C;
vector<pair<int,int>> v;
int main()
{
    f>>c>>n;
    F[1]=F[2]=1;
    for(int i=3; i<=n+1; i++)F[i]=F[i-1]+F[i-2];
    if(c==1)
    {
```

```
        L=F[n];
        C=F[n+1];
        if(n%2)swap(L,C);
        g<<L<<' '<<C<<'\n';
         return 0;
    }
    f>>p;
    L=1+(n%4==3)*F[n-1];
    C=1+(n%4==2)*F[n-1];;
    for(int i=n; i>=1; i--)
    {
        if(L<=p&&L+F[i]>p)
            v.push_back(make_pair(C,i));
        L+=(i%4==0)*F[i-2]+(i%4==1)*F[i]-(i%4==3)*F[i-1];
        C+=(i%4==0)*F[i]-(i%4==2)*F[i-1]+(i%4==3)*F[i-2];
    }
    sort(v.begin(),v.end());
    for(auto it:v)
    {
        int i=it.second;
        for(int cnt=F[i]; cnt; cnt--)
            g<<i<<' ';
    }
    g<<'\n';
    return 0;
}
```

```
/*

3.3. Autor: Popa Bogdan Ioan

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

struct ValPos
{
    int val;
    int posi, posj, sz;
};

int N, P;
vector <ValPos> V;
int curr, last;
int L, C;

bool cmp(ValPos a, ValPos b)
{
    return a.posj < b.posj;
}
int main()
{   int k;
    fin>>k;
    fin >> N >> P;
    if (k==1)
    {   int a=1,b=1,c=1;
        for (int i=3;i<=N+1;i++)
        {
            c=a+b; a=b; b=c;
        }
```

```
        if (N%2)
            fout<<b<<" "<<a;
        else
            fout<<a<<" "<<b;
        return 0;
    }
    V.push_back({1, 1, 1, 1});
    last = 0;
    curr = 1;
    L = 1;
    C = 1;
    for(int i = 2; i <= N; i++) {
        int aux = last + curr;
        last = curr;
        curr = aux;

        if(i % 4 == 2) {
            V.push_back({i, 1, C + 1, curr});
            C += curr;
        }
        if(i % 4 == 3) {
            V.push_back({i, L + 1, 1, curr});
            L += curr;
        }
        if(i % 4 == 0) {
            for(auto &it : V) {
                it.posj += curr;
            }
            V.push_back({i, 1, 1, curr});
            C += curr;
        }
        if(i % 4 == 1) {
            for(auto &it : V) {
                it.posi += curr;
            }
            V.push_back({i, 1, 1, curr});
            L += curr;
        }
    }
    sort(V.begin(), V.end(), cmp);

    for(auto it : V) {
        if(it.posi <= P && P <= it.posi + it.sz - 1) {
            for(int i = 1; i <= it.sz; i++) {
                fout << it.val << " ";
            }
        }
    }
    fout << "\n";
    return 0;
}
```

```
#include <fstream>
using namespace std;
ifstream fin("cochilie.in");
ofstream fout("cochilie.out");
int main()
{   int f[100],c,n,k,a[100][100],linii,mini, b[100],p,jos,sus,
     nrlinii,nrcoloane;
    fin>>c;
    if (c==1)
    {   fin>>n;
```

```
        f[1]=f[2]=1;
        for (int i=3;i<=n+1;i++)
            f[i]=f[i-1]+f[i-2];
        if (n%4==0)
            nrlinii=f[n],
            nrcoloane=f[n+1];
        else
            if (n%4==1)
                nrlinii=f[n+1],
                nrcoloane=f[n];
            else
                if (n%4==2)
                    nrlinii=f[n],
                    nrcoloane=f[n+1];
                else
                    nrlinii=f[n+1],
                    nrcoloane=f[n];
        fout<< nrlinii<<" "<<nrcoloane<<"\n";
        return 0;
    }
    fin>>n>>k;
    f[1]=f[2]=1;
    for (int i=3;i<=n+1;i++)
        f[i]=f[i-1]+f[i-2];
    linii=(n+1)/2;
    if (n%4==1)
    {   sus=1;
        jos=linii;
        a[sus][0]=1;
           /// numarul de elemente de pe prima linie
        a[sus][1]=n;
           /// prima linie este formata dintr-un singur numar n, va aprea de
           ///fib(n) ori
        b[sus]=f[n];
        a[jos][0]=2;   /// numarul de elemente de pe ultima linie
        a[jos][1]=n-1;
        a[jos][2]=n-2;
           /// ultima linie va contine numarul n-1 de fib(n-1) ori, iar n-2 de
           ///fib(n-2) ori
        mini=n-2;
        b[jos]=f[mini];
           /// linia a[jos] va aparea in cochilie de b[jos] de ori
    }
    if(n%4==3)
    {   sus=0;
        jos=linii;
        a[jos][0]=1;  /// numarul de elemente de pe ultima linie
        a[jos][1]=n;  /// ultima linie contine doar numarul n de fib(n) de ori
        mini=n;
        b[jos]=f[mini];       /// numarul liniilor identtice
    }
    if(n%4==2)
    {
        sus=0;
        jos=linii+1;    /// aceasta linie nu aprtine matricei, dar pentru
                        /// pornirea algoritmului este ok
        a[jos][0]=1;    /// numarul de elemente de pe ultima linie
        a[jos][1]=n+1;  /// interpretam ca si cum cochilia ar fi mai mare cu
                        ///  1 ordin, solutia finala nu este influentata
        mini=n+1;
    }
    if(n%4==0)
```

```
    {
        sus=0;
        jos=linii;
        a[jos][0]=2;   /// numarul de elemente de pe ultima linie
        a[jos][1]=n;   /// interpretam ca si cum cochilia ar fi mai mare cu 1
                       /// ordin, solutia finala nu este influentata
        a[jos][2]=n-1; /// interpretam ca si cum ochilia ar fi mai mare cu 1
                       /// ordin, solutia finala nu este influentata
        mini=n-1;
        b[jos]=f[mini];             /// numarul liniilor identtice
    }

    while (jos-sus>1)   /// cat timp mai sunt linii necalculate
    {
        sus++;          /// calculam linia de sus in functie de ultima linie
                        /// de jos calculata
        a[sus][0]=a[jos][0]+1;
        int i=1;
        while (a[jos][i]>mini)     /// elementele neminime raman nemodificate
        {
            a[sus][i]=a[jos][i];
            i++;
        }
        a[sus][i]=a[jos][i]-2;     /// elementul minim se schimba cu cele doua
                    /// pozitii anterioare (se refera la pozitii fibonacci)
        mini=a[sus][i];
        a[sus][i+1]=a[jos][i]-1;
        i+=2;
        while (i<=a[sus][0])     /// celelalte elemente din sir raman identice
        {
            a[sus][i]=a[jos][i-1];
            i++;
        }
        b[sus]=f[mini];

        if (jos-sus>1)           /// daca mai avem o linie necalculata, o vom
                                 /// calcula si pe acesta
        {
            jos--;               /// calculam lnia de jos in functie de linia
                                 ///de sus calculata
            a[jos][0]=a[sus][0]+1;
            i=1;
            while (a[sus][i]>mini)   /// elementele neminimale vor rămâne
                                     /// nemodificate
            {
                a[jos][i]=a[sus][i];    /// numarul liniilor identtice
                i++;
            }
            a[jos][i]=a[sus][i]-1;
            a[jos][i+1]=a[sus][i]-2;
            mini=a[jos][i+1];
            i=i+2;


            while (i<=a[jos][0])
            {
                a[jos][i]=a[sus][i-1];
                i++;
            }
            b[jos]=f[mini];
        }
```

```
    }
    p=1;
    while (b[p]<k)
    {
        k-=b[p];
        p++;
    }

    for (int i=1;i<=a[p][0];i++)
        for (int j=1;j<=f[a[p][i]];j++)    /// numarul liniilor identtice
            fout<<a[p][i]<<" ";
    fout<<"\n";
    return 0;
}
```

4. Logic

```
/*
  4.1. Autor: Alin Burța
  O(2^N)
*/

#include <iostream>
#include <fstream>

#define FIN "logic.in"
#define FOU "logic.out"
#define nivMAX 10
#define linMAX 1025

using namespace std;
char C[nivMAX][linMAX+1];                  //memoreaza circuitul
long long T[nivMAX][linMAX][2];            //memoreaza nr. sol cerinta 2
short V[nivMAX][linMAX];                   //pentru sol cerintei 1
int N;                                     //numarul nivelelor
int K;                                     //numarul șirurilor de la cerința 1
int nFinal;                                // 0 sau 1 pentru cerința 2
unsigned long long doi[100];               //vector cu puterile lui 2

long long nrsol(int nfinal)
{
    int i, j;
    //initializez ultima linie a piramidei
    for(j = 1; j <= doi[N-1]; j++)
    {
        if( C[N][j-1] == '&' )
            T[N][j][0] = 3, T[N][j][1] = 1;
        else
            T[N][j][0] = 1, T[N][j][1] = 3;
        T[N][j][0] %= 666013;
        T[N][j][1] %= 666013;
    }
    //calculez pentru restul liniilor
    for(i = N-1; i >= 1; i-- )
    {
        for(j = 1; j <= doi[i-1]; j++)
        {
          if( C[i][j-1] == '&' )
```

```
            T[i][j][0] = T[i+1][2*j-1][0] * (T[i+1][2*j][0] + T[i+1][2*j][1])
                         + T[i+1][2*j-1][1] * T[i+1][2*j][0],
            T[i][j][1] = T[i+1][2*j-1][1] * T[i+1][2*j][1];
          else
            T[i][j][0] = T[i+1][2*j-1][0] * T[i+1][2*j][0],
            T[i][j][1] = T[i+1][2*j-1][0] * T[i+1][2*j][1] +  T[i+1][2*j-1][1]
                         * ( T[i+1][2*j][1] + T[i+1][2*j][0] );
            T[i][j][0] %= 666013;
            T[i][j][1] %= 666013;
        }

    }
    return T[1][1][nFinal];
}
int evalueaza(char s[linMAX+1])
{
    int i, j;
    for(j = 1; j <= doi[N-1]; j++)
    {
        if( C[N][j-1] == '&' )
            V[N][j] = (s[2*j - 2] - 48) && (s[2*j - 1] - 48);
        else
            V[N][j] = (s[2*j - 2] - 48) || (s[2*j - 1] - 48);
    }
    for(i = N-1; i >= 1; i-- )
    {
        for(j = 1; j <= doi[i-1]; j++)
            if( C[i][j-1] == '&' )
                V[i][j] = V[i+1][2 * j] && V[i+1][2 * j - 1];
            else
                V[i][j] = V[i+1][2 * j] || V[i+1][2 * j - 1];
    }
    return V[1][1];
}
int main()
{
    ifstream citeste(FIN);
    ofstream scrie(FOU);
    int i,j, Cerinta;
    char sir[linMAX+1];

    //intializez un vector cu puterile lui 2
      doi[0] = 1;
      for(i = 1; i <= 63; i++)
      {
            doi[i] = doi[i-1] * 2;
      }
    //citire date
    citeste >> Cerinta;
    citeste >> N;
    for(i = 1; i <= N; i++)
    {   for(j = 0; j < doi[i-1]; j++)
            citeste>>C[i][j];
  }
    if(Cerinta == 1)
    {
        citeste>>K;
        for(i = 1; i <= K; i++)
        {
            for(j = 0; j < doi[N]; j++)
                citeste>>sir[j];
```

```
            scrie << evalueaza(sir) << '\n';
        }
    }
    else
    {
        citeste>>nFinal;
        scrie<<nrsol(nFinal)<<endl;
    }

    citeste.close();
    scrie.close();
    return 0;
}
```

```
/*
  4.2. Autor: Bogdan-Ioan Popa
*/
#include <bits/stdc++.h>

#define MOD 666013
#define N_MAX 10
using namespace std;

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

int t;
int N, Q;
int L;
char op[1 << N_MAX];
int E[1 << N_MAX];
int cnt[1 << N_MAX][2];

int main()
{
    fin >> t;
    fin >> N;
    for(int i = 0; i < N; i++) {
        for(int j = 1; j <= (1 << i); j++) {
            fin >> op[++L];
        }
    }
    if(t == 1) {
        for(fin >> Q; Q; Q--) {
            for(int i = 1; i <= (1 << N); i++) {
                char c;
                fin >> c;
                E[L + i] = c - '0';
            }

            for(int i = L; i >= 1; i--) {
                if(op[i] == '|') {
                    E[i] = E[2 * i] | E[2 * i + 1];
                }
                else {
                    E[i] = E[2 * i] & E[2 * i + 1];
                }
            }

            fout << E[1] << "\n";
        }
    }
```

```
    else {
        int result;
        fin >> result;
        for(int i = 1; i <= (1 << N); i++) {
            cnt[L + i][0] = cnt[L + i][1] = 1;
        }

        for(int i = L; i >= 1; i--) {
            for(int le = 0; le <= 1; le++) {
                for(int ri = 0; ri <= 1; ri++) {
                    if(op[i] == '|') {
                    cnt[i][le | ri] = (cnt[i][le | ri] + 1ll * cnt[2 * i][le] *
                                       cnt[2 * i + 1][ri]) % MOD;
                    }
                    else {
                        cnt[i][le & ri] = (cnt[i][le & ri] + 1ll *
                                    cnt[2 * i][le] * cnt[2 * i + 1][ri]) % MOD;
                    }
                }
            }
        }

        fout << cnt[1][result] << "\n";
    }

    return 0;
}
```

8.6 Soluții - Clasa a X-a

1. Matrice

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

using namespace std;

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

int n, a[405][405], s[405][405], sume[1600], k;

void Citire()
{
    int i, j;
    fin >> n;
    for(i = 1; i <= n; i++)
        for(j = 1; j <= n; j++)
            fin >> a[i][j];
    fin.close();
}

void Sume()
{
    int i, j;
    for(i = 1; i <= n; i++)So
        for(j = 1; j <= n; j++)
            s[i][j] = a[i][j] + s[i - 1][j] + s[i][j - 1] - s[i - 1][j - 1];
}

bool CB(int x)
{
    int st = 1, dr = k, mij;
    while(st <= dr)
    {
        mij = (st + dr) / 2;
        if(sume[mij] == x) return true;
        else if(sume[mij] < x) st = mij + 1;
        else dr = mij - 1;
    }
    return false;
}

void Rezolva()
{
    int i, j, cnt = 0;
    for(i = 1; i <= n - 1; i++)
    {
        /// l1 / 1, c1 / 1, l2 / i, c2 / i
        sume[++k] = s[i][i];
        /// l1 / 1, c1 / n - i + 1, l2 / i, c2 / n
        sume[++k] = s[i][n] - s[i][n - i];
        /// l1 / n - i + 1, c1 / 1, l2 / n, c2 / i
        sume[++k] = s[n][i] - s[n - i][i];
```

```
        /// l1 / n - i + 1, c1 / n - i + 1, l2 / n, c2 / n
        sume[++k] = s[n][n] - s[n - i][n] - s[n][n - i] + s[n - i][n - i];
    }
    sume[++k] = s[n][n];
    sort(sume + 1, sume + k + 1);
    for(i = 1; i <= n; i++)
        for(j = 1; j <= n; j++)
            if(CB(a[i][j]) == true) cnt++;
    fout << cnt << "\n";
    fout.close();
}

int main()
{

    Citire();
    Sume();
    Rezolva();

    return 0;
}
```

2. Labirint

```
/*
  Autor Ștefan Dăscălescu
*/
#include<bits/stdc++.h>
using namespace std;

int n, m, q;

bool ok[1000002];
int mx[1000002];

char mat[1002][1002];
int dist[2][1002][1002];

bool viz[1002][1002];

int ox[] = {-1, 0, 1, 0};
int oy[] = {0, 1, 0, -1};

bool check(int x, int y)
{
    return ((x >= 1) && (x <= n) && (y >= 1) && (y <= m) && (viz[x][y] == 0));
}

void lee(int drum, int x, int y)
{
    memset(viz, 0, sizeof(viz));
    deque<pair<int, int> > d;
    d.push_back({x, y});
    viz[x][y] = 1;
    for(int i = 1; i <= n; ++i)
        for(int j = 1; j <= m; ++j)
            dist[drum][i][j] = -1;
    dist[drum][x][y] = 1;
    while(!d.empty())
    {
```

```
        pair<int, int> nod = d[0];
        d.pop_front();
        for(int i = 0; i <= 3; ++i)
        {
            int nxt_x = ox[i] + nod.first;
            int nxt_y = oy[i] + nod.second;
            if(check(nxt_x, nxt_y) && dist[drum][nxt_x][nxt_y] == -1)
            {
              dist[drum][nxt_x][nxt_y] = dist[drum][nod.first][nod.second] + 1;
              if(mat[nxt_x][nxt_y] == '0')
                {
                    viz[nxt_x][nxt_y] = 1;
                    d.push_back({nxt_x, nxt_y});
                }
            }
        }
    }
}

char ans[1002][1002];

int main()
{
    ifstream cin("labirint.in");
    ofstream cout("labirint.out");

    cin >> n >> m;
    for(int i = 1; i <= n; ++i)
    {
        cin >> (mat[i] + 1);
    }

    lee(0, 1, 1);
    lee(1, n, m);

    for(int i = 1; i <= n; ++i)
        for(int j = 1; j <= m; ++j)
            if(mat[i][j] == '1' && dist[0][i][j] != -1 && dist[1][i][j] != -1)
            {
                if(dist[0][i][j] + dist[1][i][j] - 1 < dist[1][1][1])
                    ans[i][j] = '1';
                else
                    ans[i][j] = '0';
            }
            else
                ans[i][j] = '0';

    for(int i = 1; i <= n; ++i)
        cout << (ans[i] + 1) << '\n';
    return 0;
}
```

3.SDistanțe

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

using namespace std;

int main() {
  ifstream cin("sdistante.in");
  ofstream cout("sdistante.out");

  string s; cin >> s;
  vector<long long> cnt(256, 0);
  long long ans = 0, tot = 0;
  for (int i = 0; i < (int)s.size(); ++i) {
    ans = (ans + (tot - cnt[s[i]]) * (s.size() - i)) % 1000000007;
    tot += i + 1; cnt[s[i]] += i + 1;
  }
  cout << ans << endl;

  return 0;
}
```

4.Tort

```
/*
  4.1. Autor Marius Nicoli
*/
#include <fstream>
#define DIM 400010
using namespace std;
int v[DIM];
int f[DIM];
int n, s, i, sol, j;
int main () {
    ifstream fin ("tort.in");
    ofstream fout("tort.out");
    fin>>n;
    for (i=1;i<=n;i++)
        fin>>v[i];
    int s = 0;
    for (i=n;i>=2;i--) {
        s += v[i];
        f[s] = i;
    }
    for (i=1;i<=s;i++) {
        for (j=i;j<=s;j+=i) {
            if (f[j] == 0) {
                break;
            } else {
                sol++;
            }
        }
    }
    fout<<sol<<"\n";
    return 0;
}
```

8.7 Soluții - Clasele XI-XII

1. Polihroniade

```
/*
  1.1.Autor Andrei Constantinescu - 100 puncte
*/
#include <bits/stdc++.h>

using namespace std;

using Test = vector<string>;

struct Group {
    int type;
    vector<Test> tests;
    friend istream& operator>>(istream& i, Group& g) {
        i >> g.type;
        assert(g.type == 1 || g.type == 2 || g.type == 3);
        int T;
        i >> T;
        assert(T >= 1);
        g.tests.resize(T);
        for (int ii = 0; ii < T; ++ii) {
            int N;
            i >> N;
            assert(N >= 2 && N % 2 == 0);
            g.tests[ii].resize(N);
            for (int j = 0; j < N; ++j) {
                i >> g.tests[ii][j];
                assert(g.tests[ii][j].size() == N);
            }
        }
        return i;
    }
    friend ostream& operator<<(ostream& o, const Group& g) {
        o << g.type << ' ' << g.tests.size() << endl;
        for (int ii = 0; ii < g.tests.size(); ++ii) {
            o << g.tests[ii].size() << endl;
            for (int j = 0; j < g.tests[ii].size(); ++j) {
                o << g.tests[ii][j] << endl;
            }
        }
        return o;
    }
    int sum_n() {
        int ans = 0;
        for (int i = 0; i < tests.size(); ++i) {
            ans += tests[i].size();
        }
        return ans;
    }
};
```

```
struct Swap {
    int i, j;

    enum class Type {
        Row,
        Col
    };
    Type type;

    Swap(int _i, int _j, Type _type) : i(_i), j(_j), type(_type) {}

    friend ostream& operator<<(ostream& o, const Swap& s) {
        switch (s.type) {
            case Type::Row:
                o << 'L';
                break;
            case Type::Col:
                o << 'C';
        }
        o << ' ' << s.i + 1 << ' ' << s.j + 1;
        return o;
    }
};

bool solve_task1(const Test& /* = vector<string>& */ t) {
    const int N = t.size();
    int lin_0 = 1, col_0 = 1;
    for (int i = 1; i < N; ++i) {
        int dif = 0;
        for (int j = 0; j < N; ++j) {
            dif += (t[0][j] != t[i][j]);
        }
        if (dif != 0 && dif != N) return false;
        if (dif == 0) ++lin_0;
        dif = 0;
        for (int j = 0; j < N; ++j) {
            dif += (t[j][0] != t[j][i]);
        }
        if (dif != 0 && dif != N) return false;
        if (dif == 0) ++col_0;
    }
    return lin_0 == N / 2 && col_0 == N / 2;
}

vector<Swap> get_opt_swaps(const string& bits, Swap::Type type) {
    const int N = bits.size();
    vector<int> freq[2][2];  // freq[i][j] = bit i on position j mod 2.
    for (int i = 0; i < N; ++i) {
        freq[bits[i] - '0'][i % 2].push_back(i);
    }
    assert(freq[0][1].size() == freq[1][0].size());
    assert(freq[0][0].size() == freq[1][1].size());

    vector<Swap> swp;
    if (freq[0][1].size() < freq[0][0].size()) {
        for (int i = 0; i < freq[0][1].size(); ++i) {
            swp.emplace_back(freq[0][1][i], freq[1][0][i], type);
        }
    } else {
        for (int i = 0; i < freq[0][0].size(); ++i) {
```

```
            swp.emplace_back(freq[0][0][i], freq[1][1][i], type);
        }
    }
    return swp;
}

template <typename T>
void append(std::vector<T>& a, const std::vector<T>& b) {
    a.reserve(a.size() + b.size());
    a.insert(a.end(), b.begin(), b.end());
}

vector<Swap> solve_task23(const Test& /* = vector<string>& */ t) {
    const int N = t.size();
    vector<Swap> ans = get_opt_swaps(t[0], Swap::Type::Col);
    string bits;
    for (int i = 0; i < N; ++i) {
        bits += t[i][0];
    }
    append(ans, get_opt_swaps(bits, Swap::Type::Row));
    return ans;
}

void print_solve_group(ostream& o, const Group& g) {
    assert(g.type == 1 || g.type == 2 || g.type == 3);
    if (g.type == 1) {
        for (int i = 0; i < g.tests.size(); ++i) {
            o << solve_task1(g.tests[i]) << endl;
        }
        return;
    }
    for (int i = 0; i < g.tests.size(); ++i) {
        const vector<Swap> swaps = solve_task23(g.tests[i]);
        o << swaps.size() << endl;
        if (g.type == 3) {
            for (int j = 0; j < swaps.size(); ++j) {
                o << swaps[j] << endl;
            }
        }
    }
}
int main() {
    Group g;
    cin >> g;
    print_solve_group(cout, g);
    return 0;
}
```

```
/*
  1.2.Autor Stelian Chichirim - 100 puncte
*/
#include <bits/stdc++.h>

using namespace std;

const int Nmax = 1000;

struct str {
    char c;
    int x, y;
};
```

```
vector<int> lines[2], columns[2];
vector<str> ans;
int vaz[Nmax + 10], n;
char A[Nmax + 10][Nmax + 10];

void calc(vector<int> &v) {
    for (int i = 1; i <= n; ++i) {
        if (vaz[i]) continue;
        if (v.empty()) {
            v.push_back(i);
            vaz[i] = 1;
            continue;
        }
        int eq = 1;
        for (int j = 1; j <= n; ++j)
            if (A[i][j] != A[v[0]][j]) {eq = 0; break;}
        if (eq) {
            v.push_back(i);
            vaz[i] = 1;
        }
    }
}

void add(char op, int x, int y) {
    ans.push_back({op, x, y});
}

void solve(char op, vector<int> &odd, vector<int> &even) {
    int cntOdd = 0;
    for (int x : odd) cntOdd += x % 2;
    if (cntOdd < odd.size() - cntOdd) swap(odd, even);
    for (int i = 0 ; i < odd.size(); ++i)
        if (odd[i] % 2 == 0)
            for (int j = 0; j < even.size(); ++j)
                if (even[j] % 2 == 1) {
                    add(op, odd[i], even[j]);
                    swap(odd[i], even[j]);
                    break;
                }
}

int main()
{
    int P, T, S = 0;
    scanf("%d%d", &P, &T);
    assert(1 <= P && P <= 3);
    assert(1 <= T && T <= 350);
    while (T--) {
        scanf("%d", &n);
        S += n;
        assert(1 <= n && n <= 1000);
        int ok = 1;
        ans.clear();
        for (int i = 0; i < 2; ++i) {
            lines[i].clear();
            columns[i].clear();
        }
        for (int i = 1; i <= n; ++i) {
            scanf("\n%s", A[i] + 1);
            vaz[i] = 0;
        }
```

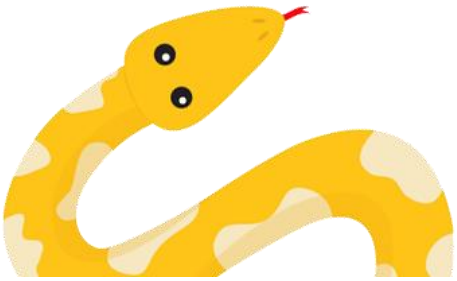
```
        calc(lines[0]);
        calc(lines[1]);
        if (lines[0].size() != n / 2 or lines[1].size() != n / 2) ok = 0;
        else {
            int eq = 1;
            for (int i = 1; i <= n; ++i)
                if (A[lines[0][0]][i] == A[lines[1][0]][i]) {eq = 0; break;}
            if (!eq) ok = 0;
        }
        for (int i = 1; i <= n; ++i)
            if (A[1][i] == '0') columns[0].push_back(i);
            else columns[1].push_back(i);
        if (columns[0].size() != n / 2) ok = 0;
        if (P == 1) {
            if (ok) printf("1\n");
            else printf("0\n");
            continue;
        }
        solve('L', lines[0], lines[1]);
        solve('C', columns[0], columns[1]);
        printf("%d\n", ans.size());
        if (P == 3)
            for (auto r : ans) printf("%c %d %d\n", r.c, r.x, r.y);
    }
    assert(S <= 2000);
    return 0;
}
```

```
/*
  1.3.Autor Adrian Panaete
*/
#include <bits/stdc++.h>

using namespace std;
void solve(),afisare(char,vector<int>&,vector<int>&);
int p,t,n,balans(string);
bool ok;
int main()
{
    cin>>p>>t;
    for(;t;t--)solve();
    return 0;
}
void solve()
{
    ok=true;
    cin>>n;
    string s,L,C,LN;
    cin>>L;
    LN=L;
    for(auto &it:LN)it='1'-it+'0';
    C.push_back(L[0]);
    for(int i=1;i<n;i++)
    {
        cin>>s;
        C.push_back(s[0]);
        if(s==L||s==LN)
            continue;
        ok=false;
    }
    if(balans(L)||balans(C))ok=false;
    if(p==1){cout<<ok<<'\n';return;}
```

```
    vector<int> a[2][2],b[2][2];
    for(int i=0;i<n;i++)
        a[i%2][int(L[i]-'0')].push_back(i+1);
    for(int i=0;i<n;i++)
        b[i%2][int(C[i]-'0')].push_back(i+1);
    cout<<min(a[0][0].size(),a[0][1].size())+min(b[0][0].size(),b[0]
[1].size())<<'\n';
    if(p==2)return;
    if(a[0][0].size()<a[0][1].size())
        afisare('C',a[0][0],a[1][1]);
    else
        afisare('C',a[0][1],a[1][0]);
    if(b[0][0].size()<b[0][1].size())
        afisare('L',b[0][0],b[1][1]);
    else
        afisare('L',b[0][1],b[1][0]);

}
int balans(string w)
{
    int ret=0;
    for(auto it:w)if(it=='0')ret--;else ret++;
    return ret;
}
void afisare(char ch,vector<int> &X,vector<int> &Y)
{
    int k=X.size();
    for(int i=0;i<k;i++)
        cout<<ch<<' '<<X[i]<<' '<<Y[i]<<'\n';
}
```

Editura L&S Soft | Infobits Academy
https://www.infobits.ro

Cărți în format electronic cu specific informatic
https://ebooks.infobits.ro



Curs online gratuit interactiv – limbajul Python 3
https://www.pythonisti.ro