Münster, Germania

Olimpiada de informatică a EUROPEI CENTRALE

În perioada 5-12 iulie 2003 a avut loc cea de-a X-a ediție a Olimpiadei de Informatică a Europei Centrale. Acest concurs a fost inițiat de țara noastră în anul 1994. Anul acesta, concursul a fost găzduit de orașul german Münster.

Cea de-a X-a ediție a concursului internațional de programare *CEOI* (*Central European Olympiad in Informatics*) a avut loc, anul acesta, în orașul *Münster*, *Germania*.

Țările care au participat la acest concurs sunt: *Cehia, Croația, Germania, Polonia, România, Slovacia, Slovenia și Ungaria.* De asemenea, au fost invitate *Statele Unite* și *Iran*. Echipa *României* a fost reprezentată de:

Victor-Marius Costan, București
Radu Berinde, București
Ștefan Ciobâcă, Suceava
Mircea Digulescu, București

Echipa s-a întors în țară cu trei medalii: una de aur (*Victor-Marius Costan*), una de argint (*Radu Berinde*) și una de bronz (*Ștefan Ciobâcă*).

În continuare vă prezentăm enunțurile celor șase probleme propuse spre rezolvare.

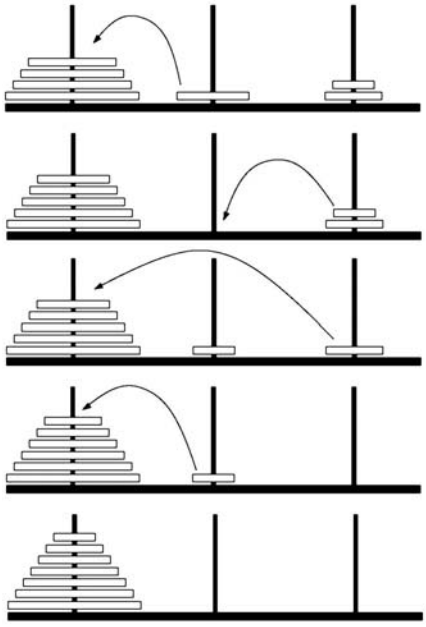
P060307: Hanoi

Toată lumea cunoaște problema *Turnurilor din Hanoi*: se consideră trei tije pe care sunt așezate discuri de mărimi diferite. Inițial toate se află pe aceeași tijă, aranjate în ordine descrescătoare de la bază spre vârf. Discurile trebuie mutate de pe tija sursă pe tija destinație, utilizând o tijă de manevră, respectând regula conform căreia pe un disc se poate așeza un altul doar dacă are diametrul mai mic.

Legenda povestește despre călugării dintr-o mănăstire tibetană care încearcă de mii de ani să rezolve problema, folosind 47 de discuri. Deoarece sunt necesare $2^{47} - 1$ mutări, nu au reușit să rezolve problema nici până în prezent. Acum discurile sunt împrăștiate pe trei tije, pe fiecare aflându-se un număr oarecare de discuri. Călugării doresc să rearanjeze discurile pe o singură tijă cu număr minim de mutări, respectând regulile privind mutările. Ei doresc să cunoască pe care tijă trebuie să mute discurile precum și numărul minim de mutări necesare.

Programul vostru trebuie să fie capabil să mute N discuri. Numerele reprezentând rezultatele parțiale, pot fi foarte mari. Din această cauză călugării sunt interesați de numărul mutărilor modulo 1.000.000.

În figura următoare este prezentată o configurație care poate fi rezolvată din patru mutări.



Date de intrare

Pe prima linie a fișierului de intrare `hanoi.in` este scris numărul N, reprezentând numărul de discuri.

Cea de-a doua linie conține trei numere întregi s_1, s_2, s_3, unde $0 \le s_1, s_2, s_3 \le N$ și $s_1 + s_2 + s_3 = N$, reprezentând numărul discurilor de pe fiecare dintre cele trei tije.

Liniile 3-5 conțin fiecare dimensiunile discurilor aflate pe fiecare dintre cele trei tije. Mai precis: cea de a $(i + 2)$-a linie din fișierul de intrare conține numerele întregi $m_{i,1}$, ..., $m_{i,si}$, unde $1 \le m_{i,j} \le N$ reprezintă mărimile discurilor de pe tija i. Diametrele discurilor sunt precizate de la bază spre vârf, adică $m_{i,1} > m_{i,2} > ... > m_{i,si}$. Rețineți că o tijă fără discuri este reprezentată printr-o linie vidă. Dimensiunile diametrelor celor N discuri formează o mulțime de numere distincte. Toate numerele sunt separate printr-un singur spațiu.

Date de ieșire
Prima linie a fișierului de ieșire **hanoi.out** va conține numărul de ordine al tijei ($d \in \{1, 2, 3\}$) pe care se vor așeza discurile printr-un număr minim de mutări. Pe cea de a doua linie se va scrie numărul M, reprezentând numărul de mutări cerut modulo 1 000 000.

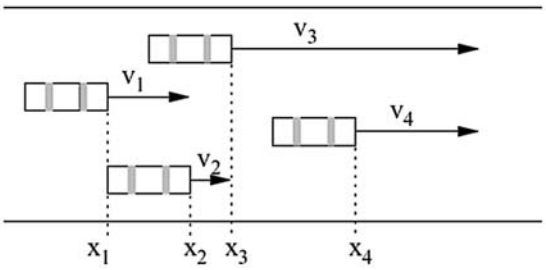
Restricție
- $1 < N \le 100.000$.

Exemplu
hanoi.in
```
7
2 1 4
2 1
3
7 6 5 4
```

hanoi.out
```
3
4
```

Timp de execuție: 1 secundă/test

P060308: Cursa
În timpul *Competiției Anuale pentru Navete Spațiale* se întrec N vehicule spațiale. Fiecare navă spațială i este lansată astfel încât poate să accelereze instantaneu până la viteza maximă V_i, rămânând fixată pe această viteză. Fiecare navă pornește de la o poziție X_i, specificându-se câți kilometri o despart de linia de start.



Cursa are lungime infinită. Datorită faptului că vitezele navelor sunt foarte mari, traiectoriile navelor sunt liniare. Navele se pot depăși fără ca traiectoriile lor să se intersecteze (acestea sunt paralele).

Sarcina voastră este să determinați câte depășiri vor avea loc și să precizați primele 10.000 de depășiri în ordine cronologică.

Rețineți că navele pornesc din poziții diferite și că la un anumit moment nu vor exista mai mult de două nave în aceeași poziție în timpul cursei.

Date de intrare
Prima linie a fișierului **therace.in** conține numărul N al navelor participante la cursă. Pe fiecare dintre următoarele N linii sunt descrise proprietățile uneia dintre nave.

Pe cea de-a $(i + 1)$-a linie a fișierului se descrie a i-a navă prin două numere întregi X_i și V_i, reprezentând poziția de start și viteza navei respective. Navele sunt aliniate la start după poziție adică $X_1 < X_2 < ... < X_N$. Poziția de start a navei este reprezentată prin numărul de kilometri ce o despart de linia de start, iar viteza este măsurată în kilometri pe secundă.

Date de ieșire
Prima linie a fișierului de intrare **therace.out** va conține numărul de depășiri pe perioada cursei modulo 1.000.000.

Pe liniile următoare se vor descrie depășirile în ordine cronologică. În cazul în care sunt mai mult de 10.000 de depășiri, doar primele 10.000 se vor afișa. Dacă sunt mai puțin de 10.000, se vor afișa toate.

Fiecare linie va conține două numere întregi i și j, reprezentând faptul că nava i o depășește pe nava j.

Dacă există mai multe depășiri în același timp, acestea vor fi sortate după pozițiile lor din cursă, deci depășirile care au loc mai aproape de punctul de start se vor scrie mai repede.

Restricții și precizări
- $0 < N \le 250\ 000$;
- $0 \le X_i \le 1.000.000$, $i = 1, 2, ... N$;
- $0 < V_i < 100$, $i = 1, 2, ... N$.

Exemplu
therace.in
```
4
0 2
2 1
3 8
6 3
```

therace.out
```
2
3 4
1 2
```

Observație
Dacă găsiți numărul corect de depășiri veți primi 40% din scorul pe test.

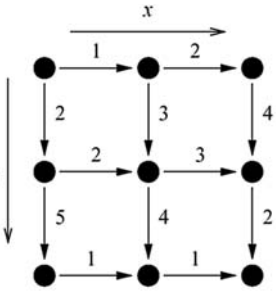
Găsirea corectă a primelor 10.000 de depășiri va fi punctată cu 60% din scor. În cazul în care programul se termi-

nă normal în cadrul timpului alocat, cele două părți din fișierul de ieșire se vor puncta separat.

Pentru această problemă sunt 20 de teste și se cunoaște conținutul fișierelor de test. Pentru fiecare test în parte valorile lui N sunt: 5, 10, 15, 20, 50, 100, 150, 200, 1000, 2000, 3000, 4000, 30000, 40000, 50000, 60000, 70000, 80000, 90000, respectiv 100000.

Timp de execuție: 1 secundă/test

P060309: Pătrat



Se consideră un graf ale cărui noduri sunt punctele de intersecție ale unui caroiaj de dimensiune $N \times N$. Fiecare nod are muchii orientate spre nodurile vecine din dreapta și de jos, dacă un astfel de nod există. Muchiile au costuri având valori întregi w. Orice drum care pornește din nodul din stânga-sus (situat în poziția $v_{1,1}$) către nodul $v_{x,y}$ are același cost. Costul unui drum este egal cu suma costurilor muchiilor de-a lungul drumului respectiv.

În figura alăturată este prezentat graful corespunzător valorii $N = 3$. Fiecare cost de la nodul $v_{1,1}$ la $v_{2,2}$ este egal cu 4. Singurul drum de la $v_{1,1}$ la $v_{1,2}$ are costul 2.

Cunoscând valoarea numărului întreg L, determinați vârful $v_{x,y}$ pentru care costul drumului de la $v_{1,1}$ la $v_{x,y}$ este exact L.

Costurile nu sunt date direct programului, ci acesta le va cere prin intermediul unei biblioteci. Programul vostru va pune cel mult 6667 de întrebări referitoare la costuri.

Bibliotecile Pascal și C/C++

Pentru a putea folosi biblioteca va trebui să includeți în programul vostru, pentru limbajul *Pascal*, instrucțiunea `uses square_lib.h;`.

Dacă utilizați limbajul *C/C++*,va trebui să includeți în programul vostru, secvența: `#include "square_lib.h"`.

În continuare sunt prezentate sintaxele funcțiilor și procedurilor incluse în biblioteci:

```
function getN: Longint;
int getN()
```

- returnează valoarea lui N.

```
function getL: Longint;
int getL()
```

- returnează valoarea lui L.

```
function getWeight(x, y, directie: Longint):
                                        Longint;
int getWeight(int x, int y, int directie)
```

- întrerupe execuția programului dacă este apelată de mai mult de 6667 de ori, caz în care veți primi 0 puncte pentru testul respectiv;
- returnează costul drumului pornind din $v_{x,y}$ spre dreapta (`directie = 0`) sau în jos (`directie = 1`).

```
procedure solution(x, y: Longint);
void solution(int x, int y)
```

- se apelează în momentul în care ați găsit nodul $v_{x,y}$ pentru care costul drumului de la $v_{1,1}$ până la el este L;
- calculează punctajul și întrerupe execuția programului.

Restricții și precizări

- $0 \le N \le 2003$;
- $0 \le w \le 500.000$;
- $0 \le L \le 2.000.000$;
- în cazul în care nu ați găsit nici o soluție, apelați funcția `solution` cu parametrii `-1` și `-1`(`solution(-1,-1)`).

Exemplu de utilizare a bibliotecii

Subprogram apelat	*Valoare returnată*
`GetN()`	3
`getL()`	4
`getWeight(1,1,0)`	1
`getWeight(2,1,1)`	3
`solution(2,2)`	-

Timp de execuție: 0,1 secunde/test

P060310: Colierul de perle

Un grup de pitici roșii și verzi au găsit un șirag format din mărgele de sticlă negre și albe, la sfârșitul căruia se află un valoros diamant.

Fiecare clan dorește să intre în posesia diamantului. Piticii decid să rezolve problema amiabil, jucând următorul joc.

Cei N pitici sunt numerotați de la 1 la N, număr numit în continuare *ID*. Fiecărui pitic i se comunică două liste de *ID*-uri: o listă albă și una neagră. (Listele pot fi diferite de la un pitic la altul.) Fiecare dintre aceste liste conține *ID*-uri ale unor pitici roșii și verzi.

În timpul jocului șiragul se transmite de la un pitic la altul folosind următoarele reguli: atunci când șiragul ajunge la un pitic, acesta desprinde din el prima mărgea.

Dacă mărgeaua desprinsă este albă, piticul va alege un *ID* de pe lista sa albă și va transmite restul șiragului piticului respectiv (acesta poate fi și el însuși).

Dacă mărgeaua este neagră, piticul va alege un *ID* de pe lista sa neagră și va transmite restul șiragului piticului respectiv (acesta poate fi și el însuși).

Primul pitic, care începe jocul, este determinat aleator.

Este posibil ca șiragul să conțină doar singurul diamant. Jocul este câștigat de acel pitic care ajunge în posesia diamantului.

Sarcina voastră este să scrieți un program care să îi ajute pe piticii verzi să obțină diamantul.

Pentru a rezolva problema aveți la dispoziție o bibliotecă și un fișier de intrare și puteți presupune că piticii roșii joacă optim.

Bibliotecile Pascal și C/C++

Pentru a putea folosi biblioteca va trebui să includeți în programul vostru, pentru limbajul *Pascal*, instrucțiunea `uses pearls_lib;` și, pentru limbajul *C/C++*, `#include "pearls_lib.h"`.

În continuare sunt prezentate sintaxele funcțiilor și procedurilor incluse în biblioteci:

```
function getNext: Integer;
int getNext()
```

- trebuie apelată atunci când este rândul unui pitic roșu;
- returnează *ID*-ul piticului căruia piticul roșu îi va da restul șiragului;
- întrerupe execuția programului dacă este apelată când este rândul piticului verde, caz în care veți primi 0 puncte pentru testul respectiv.

```
procedure setNext(d: Integer);
void setNext(int d)
```

- trebuie apelată atunci când este rândul unui pitic verde;
- parametrul *d* specifică *ID*-ul piticului căruia piticul verde îi va da restul șiragului;
- întrerupe execuția programului dacă este apelată când este rândul piticului roșu sau dacă valoarea lui *d* nu este validă, caz în care veți primi 0 puncte pentru testul respectiv.

```
procedure finish;
void finish(void)
```

- se apelează la sfârșitul jocului când programul vostru se termină;
- calculează punctajul obținut și întrerupe execuția programului (în cazul în care piticul roșu este în posesia diamantului sau dacă șiragul conține mărgele pe lângă diamant veți primi 0 puncte pentru testul respectiv).

Date de intrare

Prima linie a fișierului de intrare **`pearls.in`** conține trei numere naturale: lungimea inițială L a șiragului, numărul N al piticilor și F ($1 \le F \le N$) care reprezintă *ID*-ul piticului care începe jocul. Rețineți că $1 \le i \le N$, pentru orice i reprezentând un *ID*.

A doua linie conține L caractere care descriu șiragul. Primele L - 1 caractere sunt litere *B* sau *W*. *B* reprezintă mărgeaua neagră, iar *W* reprezintă mărgeaua albă. Ultimul caracter este litera *D*, reprezentând diamantul.

Următoarele N linii descriu piticii împreună cu listele lor. A i-a linie descrie piticul având *ID*-ul i. Primul număr de pe o astfel de linie reprezintă culoarea piticului: 0 pentru un pitic verde, 1 pentru un pitic roșu. Urmează numărul L_B, reprezentând numărul piticilor din lista neagră, apoi cele L_B *ID*-uri din lista neagră a piticului respectiv ($1 \le L_B \le 20$). Urmează numărul L_W, reprezentând numărul piticilor din lista albă, apoi cele L_W *ID*-uri din lista albă a piticului respectiv ($1 \le L_W \le 20$).

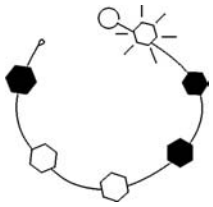
Restricții și precizări

- $1 \le L, N \le 1.000$;
- testele sunt construite astfel încât este posibil întotdeauna ca piticul verde să intre în posesia diamantului.

Exemplu de utilizare a bibliotecii

`pearls.in`

```
6 4 2
BWWBBD
0 1 2 1 4
0 2 1 3 1 1
1 1 4 1 4
1 2 2 3 1 1
```



Subprogram apelat	*Valoare returnată*
`setNext(1)`	-
`setNext(4)`	-
`getNext()`	1
`setNext(2)`	-
`setNext(1)`	-
`finish()`	-

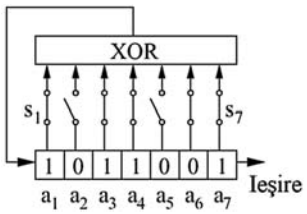
Timp de execuție: 2 secunde/test

P060311: Registru de deplasare

Un registru al unui calculator înmagazinează N biți. Biții memorați într-un registru de shiftare se translatează cu o poziție.

Folosind un registru de deplasare spre dreapta se pot genera numere binare aleatoare în felul următor: un registru de deplasare de dimensiune N se umple inițial cu biți având valorile $a_1, a_2, ..., a_N$.

La fiecare tact al ceasului sistemului de calcul, registrul furnizează la ieșire valoarea celui mai din dreapta bit a_N. Ceilalți biți sunt translatați cu o poziție spre dreapta. Prima poziție este asociată unei noi valori a'_1 astfel:



Fiecare bit din registru este conectat la o poartă *XOR* printr-un comutator (figura anterioară); adică pentru fiecare bit i există un comutator de valoare s_i (care este 1 sau 0) și care determină dacă valoarea bitului a_i este trimisă spre poarta *XOR* sau nu.

Fie $k_i = s_i \cdot a_i$. Noua valoare a'_1 este atribuită valorii de ieșire a porții *XOR*, $XOR(k_1, k_2, ..., k_N)$. (Notă: dacă numărul elementelor egale cu 1 din secvența $k_1, k_2, ..., k_N$ este impar, atunci valoarea lui $XOR(k_1, k_2, ..., k_N)$ este 1, altfel este 0).

Considerăm următoarele definiții formale:
- $a'_1 = XOR(k_1, k_2, ..., k_N)$;
- $a'_i = a_{i-1}$, pentru $2 \le i \le N$;
- *valoarea furnizată la ieșire* = a_N.

În exemplul din figura următoare valoarea a_1 la tactul 1 se calculează astfel:
$XOR(1 \cdot 1, 0 \cdot 0, 1 \cdot 1, 1 \cdot 1, 0 \cdot 0, 1 \cdot 0, 1 \cdot 1) = 0$.

tact	a_1	a_2	a_3	a_4	a_5	a_6	a_7	ieşire
0	1	0	1	1	0	0	1	-
1	0	1	0	1	1	0	0	1
2	1	0	1	0	1	1	0	0
3	1	1	0	1	0	1	1	0
4	0	1	1	0	1	0	1	1
5	0	0	1	1	0	1	0	1
6	1	0	0	1	1	0	1	0
7	1	1	0	0	1	1	0	1
8	0	1	1	0	0	1	1	0
9	1	0	1	1	0	0	1	1
10	0	1	0	1	1	0	0	1
11	1	0	1	0	1	1	0	0
12	1	1	0	1	0	1	1	0
13	0	1	1	0	1	0	1	1
14	0	0	1	1	0	1	0	1

Se consideră primele $2 \cdot N$ valori furnizate la ieșire ale unui astfel de registru de deplasare. Pe baza acestor valori trebuie să determinați valorile s_i ale comutatorilor.

Date de intrare
Prima linie a fișierului de intrare **`register.in`** conține dimensiunea N a registrului de deplasare. A doua linie conține $2 \cdot N$ numere de 0 sau 1, reprezentând primele $2 \cdot N$ valori ale biților furnizați la ieșire de către registrul de deplasare.

Date de ieșire
Fișierul de ieșire **`register.out`** va conține o singură linie pe care veți scrie valorile s_i ale comutatorilor dintr-o configurație, începând cu s_1, care furnizează secvența de biți dată în fișierul de intrare, în cazul în care o astfel de configurație există. Dacă nu există nici o configurație corespunzătoare a comutatorilor, atunci în fișierul de ieșire veți scrie `-1`.

Restricție
- $1 \le N \le 750$.

Exemple
`register.in`
```
7
1 0 0 1 1 0 1 0 1 1 0 0 1 1
```

`register.out`
```
1 0 1 1 0 1 1
```

`register.in`
```
3
0 0 0 1 1 1
```

`register.out`
```
-1
```

Timp de execuție: 1,5 secunde/test

P060312: Călătoria
Alice și *Bob* vor să meargă în vacanță. Cei doi și-au planificat câte un traseu pe care doresc să-l urmeze și au scris o listă de orașe pe care vor să le viziteze într-o anumită ordine. O listă poate să conțină un anumit oraș de mai multe ori.

Deoarece cei doi vor să călătorească împreună, trebuie să cadă de acord privind un traseu comun. Nici unul nu vrea să schimbe ordinea în care au planificat vizitarea orașelor conform listei proprii și nu vor să adauge orașe noi la ea. În concluzie, ei nu au altă posibilitate decât să șteargă anumite orașe din lista proprie. Bineînțeles, traseul comun trebuie să conțină cel mai mare număr de orașe posibile de vizitat.

În regiune există exact 26 de orașe, care au fost codificate cu litere mici de la `'a'` la `'z'`.

Date de intrare
Fișierul de intrare **`trip.in`** conține două linii; prima linie conține lista lui *Alice*, cea de a doua, lista lui *Bob*. Fiecare listă conține cel puțin 1 caracter literă mică și cel mult 80 de caractere litere mici, care nu sunt separate prin nici un spațiu.

Date de ieșire
Fișierul de ieșire **`trip.out`** va trebui să conțină toate traseele care respectă condițiile descrise mai sus și fiecare traseu se va scrie în fișier o singură dată. Pe o linie se va scrie un singur traseu.

Restricții și precizări
- se garantează existența a cel puțin unui traseu nevid și cel mult 1000 de trasee diferite;
- traseele pot fi scrise în fișierul de ieșire în orice ordine.

Exemplu
`trip.in`
```
abcabcaa
acbacba
```

`trip.out`
```
ababa
abaca
abcba
acbca
acaba
acaca
acbaa
```

Timp de execuție: 7 secunde/test