

CEOI 2026



Ziua 1
(Română)

Day 1
(Romanian)

Observatorul de păsări

Limită de timp: 10 s Limită de memorie: 512 MiB

Societatea Observatorilor de Păsări din San Serriffe are o structură internă aflată în continuă schimbare. Societatea este formată din n filiale, iar fiecare membru al Societății aparține exact unei filiale. Filialele sunt numerotate de la 1 la n , iar filiala a i -a are m_i membri. Astfel, Societatea are în total $M = m_1 + m_2 + \dots + m_n$ membri.

Fiecare filială este condusă de unul dintre membrii săi, care în acest rol este numit *ofițerul* filialei. Ofițerii sunt numerotați la fel ca filialele, astfel încât (pentru fiecare $i = 1, \dots, n$) ofițerul i este cel care conduce filiala i .

Mai mult, ofițerii sunt organizați ierarhic printr-un sistem de mentorat: fiecare ofițer, cu excepția unuia, are un *mentor*, care este ofițerul unei alte filiale. Singurul ofițer fără mentor este *Președintele* Societății. Dacă ofițerul a este mentorul ofițerului b , spunem și că ofițerul b este un *discipol* al ofițerului a .

Niciun ofițer nu este, direct sau indirect, mentorul său propriu; astfel, urmărind șirul de la un ofițer la mentorul său, apoi la mentorul mentorului său și așa mai departe, ajungem întotdeauna în final la Președinte.

Definim *influența* unui ofițer ca fiind suma numărului de membri ai filialei sale și a influențelor tuturor discipolilor săi (dacă are). Se poate observa ușor că ofițerul cu cea mai mare influență este Președintele, a cărui influență este întotdeauna egală cu M . Un ofițer este numit *ofițer senior* dacă influența sa este $\geq M/2$.

Statutul Societății specifică faptul că ofițerul senior cu cea mai mică influență (dintre toți ofițerii seniori) va îndeplini funcția de *Trezorier* al societății.

Din când în când, un ofițer (altul decât Președintele) poate să își schimbe afilierea, astfel încât de atunci înainte să devină discipolul unui alt mentor decât înainte (cu condiția ca noul său mentor să nu fie unul dintre discipolii săi, sau un discipol al discipolilor săi etc.). Din această cauză, se poate întâmpla ca influența unor ofițeri să se schimbe, iar rolul de Trezorier să revină unui alt ofițer decât înainte.

Cerință

Scrieți un program care citește descrierea stării inițiale a Societății și o succesiune de schimbări de afiliere. Programul trebuie să afișeze cine este Trezorierul în starea inițială a Societății, precum și după fiecare schimbare de afiliere.

Intrare

Prima linie conține două numere întregi, n și q , separate printr-un spațiu; n este numărul de filiale, iar q este numărul de schimbări de afiliere.

Următoarele n linii descriu starea inițială a Societății.

Linia i dintre acestea conține două numere întregi, s_i și m_i , separate printr-un spațiu; s_i este mentorul ofițerului i (adică al ofițerului care conduce filiala i), iar m_i este numărul de membri ai filialei i . Valoarea $s_i = 0$ indică faptul că ofițerul i este Președintele Societății și, prin urmare, nu are mentor.

Celelalte q linii descriu schimbările de afiliere.

Linia j dintre acestea conține două numere întregi, \hat{x}_j și \hat{z}_j , separate printr-un spațiu. Semnificația acestor numere este următoarea.

Notăm cu t_j (pentru $j = 0, \dots, q$) Trezorierul după primele j schimbări de afiliere (astfel, t_0 este Trezorierul inițial înainte de prima schimbare de afiliere). Atunci schimbarea de afiliere cu numărul j constă în faptul că ofițerul z_j devine noul mentor al ofițerului x_j , unde

$$x_j = 1 + ((t_{j-1} + \hat{x}_j) \bmod n)$$

și

$$z_j = 1 + ((t_{j-1} + \hat{z}_j) \bmod n).$$

Scopul acestei reprezentări a valorilor x_j și z_j este de a forța programul să proceseze schimbările de afiliere în ordinea în care apar.

Schimbările de afiliere din datele de intrare vor fi întotdeauna valide, adică z_j nu va fi egal cu x_j , și nici nu va fi un discipol al lui x_j , un discipol al unui discipol al lui x_j etc. Totuși, este posibil ca z_j să fi fost deja mentorul lui x_j imediat înainte de schimbarea cu numărul j (astfel încât în acel moment să nu se schimbe efectiv nimic).

Rețineți că dacă programul dumneavoastră calculează greșit rezultatul t_j la un moment dat, va decodifica în mod greșit și intrările ulterioare $\hat{x}_{j+1}, \hat{z}_{j+1}$ etc. și ar putea primi un verdict RTE (eroare în timpul execuției) în loc de WA (răspuns greșit), deoarece datele de intrare decodificate incorect pot fi invalide (de exemplu, poate obține în mod eronat un z_{j+1} care este discipol al lui x_{j+1}).

Restricții

- $1 \leq n \leq 1\,000\,000$
- $1 \leq q \leq 30\,000$
- $1 \leq m_i$ pentru fiecare $i = 1, \dots, n$
- $m_1 + m_2 + \dots + m_n \leq 10^9$
- $1 \leq \hat{x}_j \leq n$ și $1 \leq \hat{z}_j \leq n$ pentru fiecare $j = 1, \dots, q$.

Subtaskuri

- Subtaskul 1 (15 puncte): $n \leq 100$
- Subtaskul 2 (10 puncte): $n \leq 1000$
- Subtaskul 3 (45 puncte): $n \leq 300\,000$
- Subtaskul 4 (30 puncte): Fără restricții suplimentare.

Ieșire

Afișați numerele t_0, t_1, \dots, t_q , fiecare pe propria linie, unde t_j este Trezorierul după primele j schimbări de afiliere.

Desigur, fiecare t_j trebuie să fie un număr întreg din intervalul $1 \leq t_j \leq n$.

Exemplu

Intrare	Ieșire
7 2	2
0 1	2
1 3	3
1 3	
2 3	
2 1	
5 2	
5 1	
3 7	
2 7	

Comentariu

Inițial, ofițerul 2 este Trezorierul (deci $t_0 = 2$). La prima schimbare de afiliere, citim $\hat{x}_1 = 3$ și $\hat{z}_1 = 7$ și calculăm $x_1 = 1 + ((2 + 3) \bmod 7) = 6$ și $z_1 = 1 + ((2 + 7) \bmod 7) = 3$; astfel, ofițerul 3 devine noul mentor al ofițerului 6; ofițerul 2 rămâne Trezorier (deci $t_1 = 2$). La a doua schimbare de afiliere, citim $\hat{x}_2 = 2$ și $\hat{z}_2 = 7$ și calculăm $x_2 = 1 + ((2 + 2) \bmod 7) = 5$ și $z_2 = 1 + ((2 + 7) \bmod 7) = 3$; astfel, ofițerul 3 devine noul mentor al ofițerului 5 și devine, de asemenea, noul Trezorier (deci $t_2 = 3$).

DFS

Limită de timp: 1 s Limită de memorie: 256 MiB

Probabil sunteți deja familiari cu celebrul algoritm DFS (depth-first search, parcurgere în adâncime) pentru parcurgerea unui graf. În această problemă, vom considera doar grafuri simple neorientate conexe (fără bucle și muchii duble) cu vârfurile numerotate cu $0, 1, \dots, n - 1$, iar algoritmul DFS va afișa adâncimile și vârfurile după cum urmează:

DFS(d, v):

 afișează d/v

 marchează vârful v ca vizitat

W = lista vecinilor lui v ordonată crescător după numerele lor
 pentru fiecare w din W :

 dacă vârful w nu a fost încă vizitat:

 DFS($d + 1, w$)

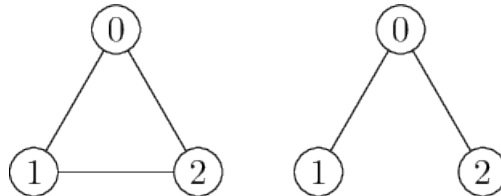
Scrieți un program care afișează numărul de grafuri diferite pentru care apelul DFS($0, n - 1$) produce exact aceeași afișare precum cea dată la intrare. De exemplu, ieșirea

0/2

1/0

2/1

este produsă de apelul DFS($0, 2$) pe oricare dintre următoarele două grafuri simple neorientate conexe cu 3 vârfuri:



Intrare

Intrarea este ieșirea apelului DFS($0, n - 1$) pe un graf simplu neorientat conex necunoscut cu n vârfuri. Astfel, intrarea conține n linii în formatul d/v , prima linie fiind $0/n-1$.

Restricții

- $1 \leq n \leq 2 \cdot 10^5$

Subtaskuri

- Subtaskul 1 (10 puncte): $n \leq 6$.
- Subtaskul 2 (20 puncte): $n \leq 500$.
- Subtaskul 3 (20 puncte): $n \leq 10^4$.

- Subtaskul 4 (10 puncte): Pentru fiecare $i \in \{2, \dots, n\}$, a i -a linie de intrare este $i-1/i-2$.
- Subtaskul 5 (20 puncte): Pentru fiecare $i \in \{2, \dots, n\}$, a i -a linie de intrare este $i-1/v$ pentru un anumit $v \in \{0, \dots, n-2\}$.
- Subtaskul 6 (20 puncte): Fără restricții suplimentare.

Ieșire

Afișați numărul de grafuri diferite, având proprietatea descrisă în enunț. Deoarece acest număr poate fi foarte mare, afișați rezultatul modulo 1 000 000 007.

Exemplu

Intrare

0/2

1/0

2/1

Ieșire

2

Vânătoare de comori

Limită de timp: 8 s Limită de memorie: 256 MiB

Cauți o comoară pierdută de mult timp pe un câmp uriaș de celule de dimensiune $N \times N$, folosind o busolă magică. Există în total K cufere cu comori ascunse în diferite celule ale câmpului. Scopul tău este simplu: să le găsești pe toate!

Când așezi busola pe una dintre celule, aceasta va găsi cel mai scurt drum către un cufăr cu comori folosind numai mișcări la stânga, sus, dreapta și jos (adică spre cel mai apropiat cufăr conform distanței Manhattan). Apoi va afișa direcția primei mișcări. Dacă există mai multe prime mișcări valide care duc către cel mai apropiat cufăr (sau către mai multe dintre ele), busola le va returna pe toate. Dacă această celulă conține o comoară, busola va indica acest lucru în schimb.

Ori de câte ori găsești un cufăr cu comori, îi poți goli conținutul, dar nu poți elimina cufărul — este prea greu. Busola ta nu știe dacă un cufăr este plin sau gol. Ea va indica drumul către cel mai apropiat cufăr, indiferent dacă este gol sau plin.

Încearcă să găsești locația tuturor cuferelelor cu comori folosind cât mai puține interogări ale busolei.

Cerință

Aceasta este o problemă interactivă. În fiecare test (adică la fiecare execuție a programului tău), programul va trebui să rezolve mai multe vânătoari de comori. Trebuie să interacționezi cu evaluatorul folosind o bibliotecă furnizată de organizatori. Această bibliotecă conține următoarele declarații:

- **void** *NextHunt*(**int** & N , **int** & K) — apelează această funcție pentru a începe următoarea vânătoare de comori. Ea va returna dimensiunea grilei în parametrul N și numărul de comori în K . Dacă nu mai există alte vânătoari de comori de rezolvat în execuția curentă, funcția va seta N și K la -1 ; în acest caz, trebuie să închei programul cu codul de ieșire 0. Reține că poți apela această funcție înainte de a găsi toate comorile din vânătoarea curentă, de exemplu dacă încerci să obții doar punctaj parțial.
- **enum** { $TREASURE = 0$, $DIR_RIGHT = 1$, $DIR_UP = 2$, $DIR_LEFT = 4$, $DIR_DOWN = 8$ }; — acestea sunt constantele folosite în valorile returnate de funcția *Query* (vezi mai jos).
- **int** *Query*(**int** x , **int** y) — dacă celula cu coordonatele (x, y) conține o comoară, această funcție returnează $TREASURE$. În caz contrar, returnează suma uneia sau mai multor constante dintre DIR_RIGHT , DIR_UP , DIR_LEFT și DIR_DOWN , indicând ce mișcări returnează busola atunci când este plasată pe celula (x, y) . Coordonatele x și y trebuie să fie numere naturale între 0 și $N - 1$. (Notă: în această problemă, coordonatele y cresc de sus în jos.)

După ce *NextHunt* setează $N = K = -1$, programul tău nu mai trebuie să apeleze nici *NextHunt*, nici *Query*. De asemenea, nu trebuie să apeleze *Query* înainte de primul apel al lui *NextHunt*. Dacă programul nu respectă aceste restricții, sau dacă efectuează mai

mult de 1000 de interogări într-o singură vânătoare de comori, sau dacă furnizează valori pentru x și/sau y în afara intervalului permis atunci când apelează *Query*, biblioteca va termina execuția programului și va returna verdictul run-time-error (RTE) pentru cazul de test curent.

O comoară este considerată găsită dacă ai interogat celula care o conține cel puțin o dată. Dacă programul tău apelează *NextHunt* înainte de a găsi toate comorile din vânătoarea curentă, acest lucru nu este considerat o eroare, dar îți va afecta scorul (mai multe detalii despre punctaj mai jos).

Pentru a accesa biblioteca, programul tău trebuie să includă fișierul header `treasurehuntlib.h`:

```
#include "treasurehuntlib.h"
```

Poți descărca acest fișier header de aici: `treasurehuntlib.h`.

Pentru a te ajuta în dezvoltarea soluției, este disponibilă aici o implementare simplă a bibliotecii: `treasurehuntlib-public.cpp`. Pentru a o compila împreună cu programul tău, este suficient să adaugi numele fișierul care conține implementarea funcțiilor din header ca parametru al comenzii de compilare, de exemplu:

```
g++ foo.cpp treasurehuntlib-public.cpp
```

dacă `foo.cpp` este numele fișierului care conține soluția ta.

Implementarea din `treasurehuntlib-public.cpp` conține, pe lângă funcțiile menționate anterior, o altă funcție cu antetul `void InitFromFile(const char *fileName)` care citește o listă de vânători de comori și vă permite să simulați un joc cu acestea în locul unora random. Veți găsi mai multe detalii în fișierul `treasurehuntlib-public.cpp`.

Pe serverul de evaluare va fi folosită o implementare diferită a bibliotecii, așa că nu trebuie să faci presupuneri despre modul exact în care funcționează implementarea. Poți însă presupune că nu poluează spațiul global de nume cu alte declarații decât cele enumerate mai sus (*NextHunt*, *Query* și cele cinci constante).

Codul tău nu trebuie să citească de la intrarea standard și nici să scrie la ieșirea standard, deoarece acestea vor fi folosite de implementarea noastră a bibliotecii de pe serverul de evaluare pentru a comunica cu restul mediului de evaluare.

Intrare

Restricții

- $1 \leq N \leq 10^6$
- $1 \leq K \leq 3$
- În cadrul unei singure execuții a programului tău vor exista cel mult 100 000 vânători de comori.
- În cadrul unei singure vânători de comori poți efectua cel mult 1000 de interogări.
- Sistemul de evaluare nu este adaptiv.

Subtaskuri

- Subtask 1 (10 puncte) $K = 1$
- Subtask 2 (30 puncte) $K = 2$
- Subtask 3 (60 puncte) $K = 3$.

Punctaj

Un subtask poate consta din mai multe cazuri de test (mai multe execuții ale programului tău), iar fiecare caz de test poate conține mai multe vânători de comori. În scopul punctării, toate vânătorile de comori ale unui anumit subtask sunt evaluate împreună, indiferent de modul în care au fost distribuite inițial între cazurile de test. Pentru a i -a vânătoare de comori, notăm dimensiunea grilei cu $N_i \times N_i$, numărul de comori cu K_i , numărul de interogări efectuate de programul tău cu Q_i și numărul de comori găsite cu F_i . De asemenea, notăm cu S numărul total de puncte disponibile pentru acest subtask. Atunci, numărul de puncte acordat programului tău pentru acest subtask este:

- Dacă programul tău a găsit întotdeauna toate comorile (adică dacă $F_i = K_i$ pentru toate valorile lui i), acesta primește

$$\frac{S}{2} + \frac{S}{2} \cdot \min_i f\left(\frac{Q_i}{\max\{1, \lceil \log_2 N_i \rceil\}}\right) \text{ puncte, unde } f(t) = \begin{cases} 1, & t \leq 11 \\ 1 - (t - 11)/9, & 11 \leq t \leq 20 \\ 0, & t \geq 20. \end{cases}$$

- Dacă programul tău nu a găsit întotdeauna toate comorile (adică există un i pentru care $F_i < K_i$), acesta primește

$$\frac{S}{2} \cdot \min_i \frac{F_i}{K_i} \text{ puncte.}$$

Cu alte cuvinte, primești jumătate din puncte pentru găsirea tuturor comorilor și cealaltă jumătate pentru găsirea lor folosind cât mai puține interogări. Pentru un scor perfect, soluția ta trebuie să găsească toate comorile în cel mult $11 \lceil \log_2 N_i \rceil$ interogări. Între $11 \lceil \log_2 N_i \rceil$ și $20 \lceil \log_2 N_i \rceil$ interogări, scorul scade liniar; iar dacă soluția ta efectuează mai mult de $20 \lceil \log_2 N_i \rceil$ interogări, va primi doar prima jumătate a punctelor pentru găsirea tuturor comorilor. (Simbolurile $\lceil \cdot \rceil$ înseamnă că valoarea lui $\log_2 N_i$ este rotunjită în sus la cel mai apropiat număr întreg.)

Dacă formulele de mai sus produc un număr real de puncte pentru subtask, acesta va fi rotunjit la cel mai apropiat număr întreg.

Dacă programul tău produce o eroare de execuție sau nu respectă protocolul descris mai sus pentru utilizarea bibliotecii, va primi 0 puncte pentru întregul subtask. Pentru a obține puncte parțiale pentru găsirea doar a unui subset al comorilor, programul tău trebuie, prin urmare, să încheie căutarea în mod corespunzător prin apelarea funcției *NextHunt*.

Exemplu

Apel	Valoare returnată
NextHunt(N, K)	$N = 4, K = 1$
Query(2, 0)	<i>DIR_DOWN</i> <i>DIR_RIGHT</i>
Query(3, 1)	<i>DIR_DOWN</i>
Query(3, 2)	<i>TREASURE</i>
NextHunt(N, K)	$N = -1, K = -1$