

CEOI 2026



1. dan
(Hrvatski)

Day 1
(Croatian)

Promatrači ptica

Ograničenje vremena: 10 s Ograničenje memorije: 512 MiB

Društvo promatrača ptica iz San Serriffa ima neobično glomaznu i stalno promjenjivu unutarnju organizaciju. Društvo se sastoji od n *podružnica*, a svaki član Društva pripada točno jednoj podružnici. Podružnice su označene brojevima od 1 do n , a i -ta podružnica ima m_i članova. Stoga Društvo ukupno ima $M = m_1 + m_2 + \dots + m_n$ članova.

Svaku podružnicu vodi jedan od njezinih članova, koji se u toj ulozi naziva *službenik* podružnice. Službenici su označeni istim brojevima kao i podružnice, tako da je (za svaki $i = 1, \dots, n$) službenik i zadužen za podružnicu i .

Osim toga, službenici su hijerarhijski organizirani kroz sustav mentorstva: svaki službenik osim jednoga ima *mentora*, koji je službenik neke druge podružnice. Jedini službenik bez mentora jest *predsjednik* Društva. Ako je službenik a mentor službeniku b , tada kažemo i da je službenik b *učenik* službenika a . Nijedan službenik nije, izravno ili neizravno, mentor samome sebi; stoga, slijedeći niz od službenika do njegova mentora, zatim mentorova mentora i tako dalje, uvijek na kraju dolazimo do predsjednika.

Utjecaj službenika definiran je kao zbroj broja članova njegove podružnice i utjecaja svih njegovih učenika (ako ih ima). Lako je vidjeti da službenik s najvećim utjecajem uvijek jest predsjednik, čiji je utjecaj jednak M . Službenik se naziva *višim službenikom* ako je njegov utjecaj $\geq M/2$.

Statut Društva propisuje da će viši službenik s najmanjim utjecajem (među svim višim službenicima) obnašati dužnost *rizničara* Društva.

S vremena na vrijeme neki službenik (osim predsjednika) može *promijeniti odanost*, tako da od tada postaje učenik drugog mentora nego prije (pod uvjetom da njegov novi mentor nije jedan od njegovih učenika, učenika njegovih učenika itd.). Zbog toga se može promijeniti utjecaj nekih službenika, pa dužnost rizničara može prijeći na drugog službenika.

Zadatak

Napišite program koji učitava opis početnog stanja Društva i niz promjena odanosti. Vaš program mora ispisati tko je rizničar u početnom stanju Društva, kao i nakon svake promjene odanosti.

Ulaz

Prvi redak sadrži dva cijela broja, n i q , odvojena razmakom; n je broj podružnica, a q broj promjena odanosti.

Sljedećih n redaka opisuje početno stanje Društva.

i -ti od tih redaka sadrži dva cijela broja, s_i i m_i , odvojena razmakom; s_i je mentor službenika i (tj. službenika zaduženog za podružnicu i), dok je m_i broj članova podružnice i . Vrijednost $s_i = 0$ označava da je službenik i predsjednik Društva te stoga nema mentora.

Preostalih q redaka opisuje promjene odanosti.

j -ti od tih redaka sadrži dva cijela broja, \hat{x}_j i \hat{z}_j , odvojena razmakom. Značenje tih brojeva je sljedeće.

Neka je t_j (za $j = 0, \dots, q$) rizničar nakon prvih j promjena odanosti (dakle, t_0 je

početni rizničar prije prve promjene odanosti). Tada se j -ta promjena odanosti sastoji od toga da službenik z_j postaje novi mentor službeniku x_j , gdje je

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

i

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

Svrha ovakvog prikaza vrijednosti x_j i z_j jest prisiliti vaš program da promjene odanosti obrađuje upravo redoslijedom kojim se pojavljuju u ulazu.

Promjene odanosti u ulaznim podacima uvijek će biti valjane, tj. z_j neće biti jednak x_j , niti će z_j biti učenik službenika x_j , učenik njegova učenika itd. Ipak, moguće je da je z_j već bio mentor službeniku x_j neposredno prije j -te promjene, pa se tada zapravo ništa ne mijenja.

Napominjemo da će, ako vaš program u nekom trenutku izračuna pogrešnu vrijednost t_j , i kasniji ulazni podaci \hat{x}_{j+1} , \hat{z}_{j+1} itd. biti pogrešno dekodirani. Zbog toga program može završiti s presudom **RTE** (*runtime error*) umjesto **WA** (*wrong answer*), jer tako pogrešno dekodirani podaci mogu biti nevaljani (primjerice, može se dobiti da je z_{j+1} učenik službenika x_{j+1}).

Ograničenja

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

Podzadaci

- Podzadatak 1 (15 bodova): $n \leq 100$
- Podzadatak 2 (10 bodova): $n \leq 1000$
- Podzadatak 3 (50 bodova): $n \leq 300\,000$
- Podzadatak 4 (25 bodova): Nema dodatnih ograničenja.

Izlaz

Ispišite brojeve t_0, t_1, \dots, t_q , svaki u zasebnom retku, gdje je t_j rizničar nakon prvih j promjena odanosti.

Naravno, svaki t_j mora biti cijeli broj iz raspona $1 \leq t_j \leq n$.

Primjer

Ulaz	Izlaz
7 2	2
0 1	2
1 3	3
1 3	
2 3	
2 1	
5 2	
5 1	
3 7	
2 7	

Objašnjenje

U početnom stanju službenik 2 je rizničar (dakle, $t_0 = 2$).

Pri prvoj promjeni odanosti učitavamo $\hat{x}_1 = 3$ i $\hat{z}_1 = 7$ te izračunamo

$$x_1 = 1 + ((2 + 3) \bmod 7) = 6$$

i

$$z_1 = 1 + ((2 + 7) \bmod 7) = 3.$$

Dakle, službenik 3 postaje novi mentor službeniku 6; službenik 2 ostaje rizničar (stoga je $t_1 = 2$).

Pri drugoj promjeni odanosti učitavamo $\hat{x}_2 = 2$ i $\hat{z}_2 = 7$ te izračunamo

$$x_2 = 1 + ((2 + 2) \bmod 7) = 5$$

i

$$z_2 = 1 + ((2 + 7) \bmod 7) = 3.$$

Dakle, službenik 3 postaje novi mentor službeniku 5 te ujedno postaje novi rizničar (stoga je $t_2 = 3$).

DFS

Ograničenje vremena: 1 s Ograničenje memorije: 256 MiB

Možda ste već upoznati s poznatim algoritmom DFS (pretraživanje u dubinu, engl. *depth-first search*) za obilazak grafa. U ovom ćemo zadatku razmatrati samo povezane neusmjerene jednostavne grafove (bez petlji i višestrukih bridova) čiji su vrhovi označeni brojevima $0, 1, \dots, n - 1$, a algoritam DFS ispisuje dubine i vrhove na sljedeći način:

```
DFS(d, v):
    ispiši d/v
    označi vrh v kao posjećen
    W = popis susjeda vrha v poredan uzlazno po brojevima
    za svaki w u W:
        ako vrh w još nije posjećen:
            DFS(d + 1, w)
```

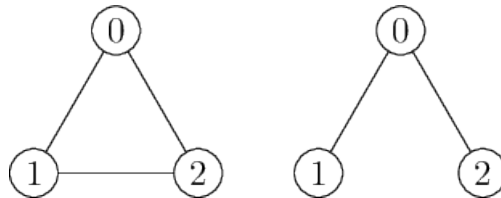
Napišite program koji ispisuje broj različitih grafova za koje poziv $\text{DFS}(0, n - 1)$ daje isti ispis kao onaj zadan na ulazu. Na primjer, ispis

0/2

1/0

2/1

dobiva se pozivom $\text{DFS}(0, 2)$ nad bilo kojim od sljedeća dva povezana neusmjerena jednostavna grafa s 3 vrha:



Ulaz

Ulaz predstavlja ispis poziva $\text{DFS}(0, n - 1)$ nad nepoznatim povezanim neusmjerenim jednostavnim grafom s n vrhova. Ulaz se stoga sastoji od n redaka u formatu d/v , pri čemu je prvi redak $0/n-1$.

Ograničenja

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

Podzadaci

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

- Podzadatak 4 (10 bodova): Za svaki $i \in \{2, \dots, n\}$, i -ti redak ulaza je $i-1/i-2$.
- Podzadatak 5 (20 bodova): Za svaki $i \in \{2, \dots, n\}$, i -ti redak ulaza je $i-1/v$ za neki $v \in \{0, \dots, n-2\}$.
- Podzadatak 6 (20 bodova): Nema dodatnih ograničenja.

Izlaz

Ispišite broj različitih grafova koji zadovoljavaju traženo svojstvo. Budući da taj broj može biti vrlo velik, ispišite rezultat modulo 1 000 000 007.

Primjer

Ulaz

0/2

1/0

2/1

Izlaz

2

Potruga za blagom

Ograničenje vremena: 8 s Ograničenje memorije: 256 MiB

Tražite davno izgubljeno blago na golemom polju veličine $N \times N$ pomoću čarobnog kompasa. Na polju su skrivena ukupno $K \leq 3$ sanduka s blagom, pri čemu se svaki nalazi u različitom polju. Vaš je cilj jednostavan: pronaći ih sve!

Kada postavite kompas na neko polje, on pronalazi najkraći put do sanduka s blagom koristeći samo pomake ulijevo, gore, udesno i dolje (odnosno do najbližeg sanduka prema Manhattanovoj udaljenosti). Zatim pokazuje smjer prvog koraka na tom putu. Ako postoji više mogućih prvih koraka koji vode do najbližeg sanduka (ili više njih), kompas će vratiti sve takve smjerove. Ako se na tom polju nalazi blago, kompas će to naznačiti.

Kad pronađete sanduk s blagom, ispraznite njegov sadržaj, ali ga ne možete ukloniti jer je pretežak. Kompas ne razlikuje puni i prazni sanduk. Uvijek pokazuje prema najbližem sanduku, bez obzira na to je li on pun ili prazan.

Pokušajte pronaći položaj svih sanduka koristeći što manje upita kompasu.

Zadatak

Ovo je interaktivan zadatak. U svakom testnom primjeru (tj. pri svakom pokretanju vašeg programa) vaš program mora riješiti više potraga za blagom. S ocjenjivačem komunicirate pomoću biblioteke koju su pripremili organizatori. Ta biblioteka sadrži sljedeće deklaracije:

- **void** *NextHunt*(**int** &*N*, **int** &*K*) — pozovite ovu funkciju za početak sljedeće potrage za blagom. Ona će postaviti veličinu mreže u varijablu *N* te broj blaga u *K*. Ako više nema potraga koje treba riješiti tijekom trenutnog pokretanja programa, funkcija će postaviti *N* i *K* na -1 ; u tom slučaju trebate završiti program s izlaznim kodom 0. Napomena: ovu funkciju možete pozvati i prije nego što pronađete sva blaga u trenutnoj potrazi, primjerice ako želite ostvariti samo djelomičan broj bodova.
- **enum** { *TREASURE* = 0, *DIR_RIGHT* = 1, *DIR_UP* = 2, *DIR_LEFT* = 4, *DIR_DOWN* = 8 }; — to su konstante koje se koriste u povratnim vrijednostima funkcije *Query* (vidi dolje).
- **int** *Query*(**int** *x*, **int** *y*) — ako se u polju s koordinatama (*x*, *y*) nalazi blago, funkcija vraća *TREASURE*. U suprotnom vraća zbroj jedne ili više konstanti *DIR_RIGHT*, *DIR_UP*, *DIR_LEFT* i *DIR_DOWN*, označavajući koje smjerove kompas vraća kada se postavi na polje (*x*, *y*). Koordinate *x* i *y* moraju biti cijeli brojevi iz raspona od 0 do $N - 1$. (Napomena: u ovom zadatku koordinata *y* raste od vrha prema dnu.)

Nakon što *NextHunt* vrati $N = K = -1$, vaš program više ne smije pozivati ni *NextHunt* ni *Query*. Također, *Query* se ne smije pozivati prije prvog poziva funkcije *NextHunt*. Ako vaš program prekrši ta pravila, napravi više od 1000 upita tijekom jedne potrage ili prosljedi vrijednosti *x* i/ili *y* izvan dopuštenog raspona pri pozivu funkcije *Query*, biblioteka će prekinuti izvršavanje programa te će za taj testni primjer biti dodijeljena presuda **RTE** (Run-Time Error).

Smatra se da je blago pronađeno ako je vaš program barem jednom postavio upit za polje koje ga sadrži. Ako vaš program pozove *NextHunt* prije nego što pronađe sva blaga u trenutnoj potrazi, to se ne smatra pogreškom, ali će utjecati na ostvareni broj bodova (više o tome u odjeljku o bodovanju).

Za korištenje biblioteke vaš program treba uključiti zaglavlje `treasurehuntlib.h`:

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

To zaglavlje možete preuzeti ovdje: `treasurehuntlib.h`.

Kako biste lakše razvili svoje rješenje, dostupna je i jednostavna implementacija biblioteke: `treasurehuntlib-public.cpp`. Da biste je preveli zajedno sa svojim programom, jednostavno dodajte njezino ime kao parametar prevoditelju, na primjer:

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

ako je `foo.cpp` naziv datoteke koja sadrži vaše rješenje.

Implementacija u datoteci `treasurehuntlib-public.cpp` uz gore navedene deklaracije podržava i funkciju `void InitFromFile(const char *fileName)`, koja učitava popis potraga iz datoteke i omogućuje vam igranje na njima umjesto na nasumično generiranim potragama. Više pojedinosti možete pronaći unutar same datoteke `treasurehuntlib-public.cpp`.

Na sustavu za ocjenjivanje koristit će se druga implementacija biblioteke, stoga ne smijete pretpostavljati ništa o načinu na koji je implementirana. Ipak, možete pretpostaviti da ne dodaje u globalni prostor imena nikakve deklaracije osim gore navedenih (*NextHunt*, *Query* i pet konstanti).

Vaš program ne smije čitati sa standardnog ulaza niti pisati na standardni izlaz, jer će se oni na sustavu za ocjenjivanje koristiti za komunikaciju između implementacije biblioteke i ostatka okruženja za ocjenjivanje.

Ulaz

Ograničenja

- $2 \leq N \leq 10^6$
- $1 \leq K \leq 3$
- Tijekom jednog pokretanja programa bit će najviše 100 000 potraga za blagom.
- Tijekom jedne potrage smijete postaviti najviše 1000 upita.
- Sustav za ocjenjivanje nije adaptivan.

Podzadaci

- Podzadatak 1 (10 bodova): $K = 1$.
- Podzadatak 2 (30 bodova): $K = 2$.
- Podzadatak 3 (60 bodova): $K = 3$.

Bodovanje

Jedan podzadatak može sadržavati više testnih primjera (više pokretanja vašeg programa), a svaki testni primjer može sadržavati više potraga za blagom. Za potrebe bodovanja sve se potrage unutar istog podzadatka promatraju zajedno, bez obzira na to kako su raspoređene po testnim primjerima.

Za i -tu potragu neka je veličina mreže $N_i \times N_i$, broj blaga K_i , broj upita koje je vaš program postavio Q_i , a broj pronađenih blaga F_i . Nadalje, neka je S ukupan broj bodova za taj podzadatak. Tada je broj bodova koji vaš program dobiva za taj podzadatak:

- Ako je vaš program uvijek pronašao sva blaga (tj. ako za svaki i vrijedi $F_i = K_i$), broj bodova ovisi o vrijednosti $t_i = \frac{Q_i}{\lceil \log_2 N_i \rceil}$:

$$\frac{S}{2} + \frac{S}{2} \cdot \min_i f(t_i) \text{ bodova, gdje je } f(t_i) = \begin{cases} 1, & t_i \leq 11 \\ 1 - (t_i - 11)/9, & 11 \leq t_i \leq 20 \\ 0, & t_i \geq 20. \end{cases}$$

- Ako vaš program nije uvijek pronašao sva blaga (tj. postoji i za koji vrijedi $F_i < K_i$), dobiva

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

Drugim riječima, polovicu bodova dobivate za pronalazak svih blaga, a drugu polovicu za to da ih pronađete koristeći što manje upita. Za maksimalan broj bodova vaše rješenje mora pronaći sva blaga u najviše $11 \lceil \log_2 N_i \rceil$ upita. Između $11 \lceil \log_2 N_i \rceil$ i $20 \lceil \log_2 N_i \rceil$ upita broj bodova linearno se smanjuje, a ako vaše rješenje napravi više od $20 \lceil \log_2 N_i \rceil$ upita, dobit će samo prvu polovicu bodova za pronalazak svih blaga. (Simboli $\lceil \cdot \rceil$ označavaju zaokruživanje vrijednosti $\log_2 N_i$ na najbliži veći cijeli broj.)

Ako navedene formule daju necijeli broj bodova za podzadatak, rezultat se zaokružuje na najbliži cijeli broj.

Ako vaš program završi s greškom tijekom izvođenja ili se ne pridržava prethodno opisanog protokola korištenja biblioteke, dobit će 0 bodova za cijeli podzadatak. Kako biste ostvarili djelomične bodove za pronalazak samo dijela blaga, vaš program mora uredno završiti trenutnu potragu pozivom funkcije *NextHunt*.

Primjer

Poziv	Povratna vrijednost
NextHunt(N, K)	$N = 4, K = 1$
Query(2, 0)	$DIR_DOWN + DIR_RIGHT = 9$
Query(3, 1)	DIR_DOWN
Query(3, 2)	$TREASURE$
NextHunt(N, K)	$N = -1, K = -1$