

CEOI 2026



1. nap
(Magyar)

Day 1
(Hungarian)

Madárlesők

Időlimit: 10 s Memórialimit: 512 MiB

A San Serriffe-i Madármegfigyelő Társaságának meglehetősen túlburjázott és folyamatosan változó belső szervezete van. A Társaság n helyi szervezetből áll, és a Társaság minden tagja pontosan egy helyi szervezetbe tartozik. A helyi szervezeteket 1-től n -ig számozzák, és az i -edik helyi szervezetnek m_i tagja van. Így a Társaság összesen $M = m_1 + m_2 + \dots + m_n$ taggal rendelkezik.

Minden helyi szervezetnek van egy vezetője, akit ebben a helyi szervezetben *tisztségviselőnek* nevezünk. A tisztségviselőket ugyanúgy számozzák, mint a helyi szervezeteket, így az i -edik tisztségviselő (minden $i = 1, \dots, n$ esetén) az i -edik helyi szervezet vezetője.

Továbbá a tisztségviselők hierarchikus rendszerbe szerveződnek: egy kivételével minden tisztségviselőnek van egy *mentora*, aki valamely másik helyi szervezet tisztségviselője. Az egyetlen tisztségviselő, akinek nincs mentora, a Társaság *Elnöke*. Ha az a tisztségviselő a b tisztségviselő mentora, akkor azt is mondjuk, hogy b tisztségviselő az a *tanítványa*.

Egyetlen tisztségviselő sem lehet saját maga mentora, sem közvetlenül, sem közvetve; ezért ha sorban haladunk egy tisztségviselőtől az ő mentorához, majd annak mentorához és így tovább, akkor végül mindig eljutunk az Elnökhöz.

Egy tisztségviselő *befolyása* a helyi szervezetében lévő tagok számának és valamennyi tanítványa befolyásának (ha vannak ilyenek) az **összege**. Könnyen belátható, hogy a legnagyobb befolyással rendelkező tisztségviselő az Elnök, akinek a befolyása mindig M . Egy tisztségviselőt *vezető tisztségviselőnek* nevezünk, ha a befolyása $\geq M/2$.

A Társaság alapszabálya kimondja, hogy a legkisebb befolyással rendelkező **vezető tisztségviselő** tölti be a Társaság *Pénztárosának* szerepét.

Időről időre egy tisztségviselő (az Elnök kivételével) *átpártolhat*, ekkor egy másik, az eddigi mentorától különböző mentor tanítványa lesz (úgy, hogy az új mentora nem lehet valamelyik tanítványa, tanítványának tanítványa stb.). Emiatt előfordulhat, hogy egyes tisztségviselők befolyása megváltozik, és a Pénztáros szerepét más tisztségviselő veszi át.

Írj programot, amely beolvassa a Társaság kezdeti állapotának leírását és az átpártolások sorozatát. A programnak ki kell írnia, hogy ki a Pénztáros a Társaság kezdeti állapotában, valamint minden egyes átpártolás után.

Bemenet

Az első sor két egész számot tartalmaz, n -et és q -t: n a helyi szervezetek száma, q pedig az átpártolások száma.

A következő n sor írja le a Társaság kezdeti állapotát.

Az i -edik sor két egész számot tartalmaz, s_i -t és m_i -t: s_i az i -edik tisztségviselő mentora (azaz az i -edik helyi szervezet vezetőjének mentora), míg m_i az i -edik helyi szervezet tagjainak száma. Az $s_i = 0$ érték azt jelenti, hogy az i -edik tisztségviselő a Társaság Elnöke, ezért nincs mentora.

A fennmaradó q sor írja le az átpártolásokat.

Ezek közül a j -edik sor két egész számot tartalmaz, \hat{x}_j -t és \hat{z}_j -t. Ezek jelentése a következő. Jelölje t_j ($j = 0, \dots, q$ esetén) a Pénztárost az első j átpártolás után (t_0 a kezdeti Pénztáros az első átpártolás előtt). Ekkor a j -edik átpártolás során z_j tisztségviselő lesz x_j tisztségviselő új mentora, ahol

$$x_j = 1 + ((t_{j-1} + \hat{x}_j) \bmod n) \quad \text{és} \quad z_j = 1 + ((t_{j-1} + \hat{z}_j) \bmod n).$$

Az x_j és z_j értékek ilyen reprezentációjának célja, hogy kikényszerítse az átpártolások bemenetben megadott sorrendben történő feldolgozását.

A bemenetben az átpártolások mindig érvényesek lesznek, azaz z_j nem lesz egyenlő x_j -vel, és z_j nem lesz x_j tanítványa, sem x_j tanítványának tanítványa stb. Előfordulhat azonban, hogy z_j már közvetlenül a j -edik átpártolás előtt is x_j mentora volt, így valójában semmi sem változik.

Ha a programod valamikor hibásan számolja ki t_j értékét, akkor a következő bemeneteket, \hat{x}_{j+1} -ot, \hat{z}_{j+1} -ot stb. is hibásan fogja dekódolni, ezért akár RTE (futási hiba) eredményt is kaphat WA (hibás válasz) helyett, mivel a hibásan dekódolt bemenet érvénytelen lehet (például hibásan olyan z_{j+1} adódhat, amely x_{j+1} tanítványa).

Korlátok

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

Részfeladatok

1. részfeladat (15 pont): $n \leq 100$
2. részfeladat (10 pont): $n \leq 1000$
3. részfeladat (50 pont): $n \leq 300\,000$
4. részfeladat (25 pont): Nincsenek további korlátozások.

Kimenet

A kimenetnek $q + 1$ sort kell tartalmaznia, a j -edik sorban a t_j értéket kell kiírni, ahol t_j a Pénztáros sorszáma a j -edik átpártolás után.

Példák

Bemenet	Kimenet
7 2	2
0 1	2
1 3	3
1 3	
2 3	
2 1	
5 2	
5 1	
3 7	
2 7	

Magyarázat

Kezdetben a 2-es sorszámú tisztségviselő a Pénztáros, így $t_0 = 2$. Az első átpártolásnál $\hat{x}_1 = 3$ és $\hat{z}_1 = 7$, így $x_1 = 1 + ((2+3) \bmod 7) = 6$ és $z_1 = 1 + ((2+7) \bmod 7) = 6$; vagyis a 3-as sorszámú tisztségviselő lesz a 6-os sorszámú tisztségviselő új mentora. A változtatás után a 2-es sorszámú tisztségviselő lesz a Pénztáros, így $t_1 = 2$. A második átpártolásnál $\hat{x}_2 = 2$ és $\hat{z}_2 = 7$, így $x_2 = 1 + ((2+2) \bmod 7)$ és $z_2 = 1 + ((2+7) \bmod 7) = 3$; vagyis a 3-as sorszámú tisztségviselő lesz az 5-ös sorszámú tisztségviselő új mentora. A változtatás után a 3-as sorszámú tisztségviselő lesz a Pénztáros, így $t_2 = 3$.

DFS

Időlimit: 1 s Memórialimit: 256 MiB

Lehet, hogy hallottál már a híres DFS (depth-first search, mélységi bejárás) gráfbejáró algoritmusról. Ebben a feladatban összefüggő, irányítatlan egyszerű gráfokra fogjuk a módszert alkalmazni. Egy n csúcsú gráf csúcsait a $0, 1, \dots, n - 1$ számokkal azonosítjuk, és a DFS algoritmus a következő módon járja be és **írja ki** a mélységeket és a csúcsokat:

DFS(d, v):

írd ki: d/v

jelöld meg a v csúcsot látogatottként

$W :=$ a v csúcs szomszédainak listája, azonosítók szerinti növekvő sorrendben minden w elemre W -ben:

ha a w csúcs még nincs meglátogatva:

DFS($d + 1, w$)

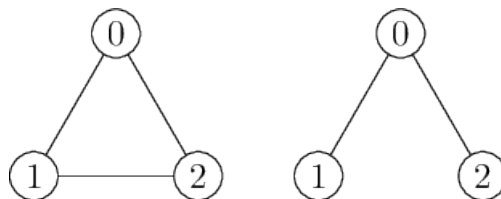
Kezdetben egyik csúcs sincs még meglátogatva.

Bemenetként megkapod egy ismeretlen gráfra a DFS($0, n - 1$) hívás során kiírt szövegeket. Írj programot, amely kiszámítja azoknak a **különböző** gráfoknak számát, amelyekre a DFS($0, n - 1$) hívás pontosan a megadott kiírásokat adja (a megadott sorrendben).

Például tegyük fel, hogy a DFS($0, 2$) hívás a következőket írja ki:

0/2
1/0
2/1

Ebben az esetben az ismeretlen gráf az alábbi két irányítatlan, egyszerű, $n = 3$ csúcsú gráf lehetett:



Bemenet

A bemenet a DFS($0, n - 1$) hívás kimenete egy ismeretlen, n csúcsú összefüggő irányítatlan egyszerű gráfon. Ez azt jelenti, hogy a bemenet n sorból áll, soronként a következő formátumban: d/v , ahol az első sor minden esetben $0/n-1$.

Korlátok

- $1 \leq n \leq 200\,000$

Részfeladatok

1. részfeladat (10 pont): $n \leq 6$.

2. részfeladat (20 pont): $n \leq 500$.
3. részfeladat (20 pont): $n \leq 10\,000$.
4. részfeladat (10 pont): Minden $i = 2, \dots, n$ -re az i -edik sor: $i-1/i-2$.
5. részfeladat (20 pont): Minden $i = 2, \dots, n$ -re az i -edik sor: $i-1/v$, ahol $0 \leq v \leq n-2$.
6. részfeladat (20 pont): Nincsenek további korlátozások.

Kimenet

A kimenetre egyetlen sort kell írnod, ami a követelményeknek megfelelő különböző gráfok darabszámát tartalmazza. Mivel ez a szám nagyon nagy is lehet, ezért az eredmény $1\,000\,000\,007$ -el vett osztási maradékát kell csak megadnod.

Példák

Bemenet

0/2

1/0

2/1

Kimenet

2

Kincsvadászat

Időlimit: 8 s Memórialimit: 256 MiB

Rég elveszett kincsek után kutatsz egy hatalmas, $N \times N$ cellából álló rácson a varázssiránytűd segítségével. Összesen $K \leq 3$ kincsesláda van elrejtve a mező K különböző cellájában. A célod egyszerű: találd meg mindegyiket!

Minden lépésben egy általad választott cellára helyezheted az iránytűd. Amikor az iránytűt egy cellára helyezed, az megkeresi az adott cellából induló **összes** létező legrövidebb utat, ami kincsesládához vezet, feltéve, hogy csak balra, felfelé, jobbra és lefelé lehet mozogni. Ezután megmutatja az **összes** lehetséges első lépés irányát. Vedd figyelembe, hogy ez jelentheti azt, hogy egyetlen legközelebbi kincsesláda van (amihez egy vagy több lehetséges első lépés vezethet el), de akár jelentheti azt is, hogy több különböző kincsesláda van azonos legkisebb távolságra (és ekkor mindegyik ilyen kincsesládához vezető összes legrövidebb út első lépéseit megkapod). Ha a cella maga tartalmaz kincset, akkor az iránytű ezt a tényt jelzi az irányok helyett.

Valahányszor megtalálsz egy kincsesládát, megszerzed a tartalmát, de magát a ládát nem tudod eltávolítani, mert túl nehéz. Az iránytű nem tudja, hogy egy láda tele van-e vagy üres: mindig a legközelebbi láda felé mutat, függetlenül attól, hogy üres vagy tele van.

Keresd meg az összes kincsesládát az iránytű lehető legkevesebb használatával.

Könyvtár

Ez egy interaktív feladat. Minden tesztesetben (vagyis a programod minden futtatásakor) a programodnak egy vagy több kincsvadászatot kell megoldania. Az értékelő programmal függvényhívások segítségével tudsz kommunikálni. A rendelkezésedre álló könyvtár a következő deklarációkat tartalmazza:

- **void** *NextHunt*(**int** &*N*, **int** &*K*) — hívd meg ezt a függvényt a soron következő kincsvadászat elindításához. Ez beállítja a rács méretét az általad átadott *N* változóban, valamint a keresett kincsek számát a *K* változóban. Ha az aktuális futás során már nincs több megoldandó kincsvadászat, a függvény *N*-et és *K*-t -1 -re állítja; ebben az esetben a programodnak 0 kilépési kóddal be kell fejeznie a futását. **Fontos**, hogy ezt a függvényt azelőtt is meghívhatod, hogy az aktuális vadászat összes kincsét megtaláltad volna, például ha csak részpontszámot szeretnél elérni.
- **enum** { *TREASURE* = 0 , *DIR_RIGHT* = 1 , *DIR_UP* = 2 , *DIR_LEFT* = 4 , *DIR_DOWN* = 8 }; — ezek azok a konstansok, amelyeket a *Query* függvény viszatérési értékei használnak (lásd lentebb).
- **int** *Query*(**int** *x*, **int** *y*) — ha az (x, y) koordinátájú cella kincset tartalmaz, akkor a függvény *TREASURE*-t ad vissza. Ellenkező esetben visszaadja a *DIR_RIGHT*, *DIR_UP*, *DIR_LEFT* és *DIR_DOWN* konstansok közül azoknak az **összegét**, amely irányú mozgásokat az iránytű megad, amikor az (x, y) cellára helyezed. Az *x* és *y* koordinátáknak 0 és $N - 1$ közötti egész számoknak kell lenniük. **Az *y* koordináták fentről lefelé növekednek.**

Miután a *NextHunt* beállította N és K értékét -1 -re, a programod többé nem hívhatja meg sem a *NextHunt*, sem a *Query* függvényt. A *Query* függvényt az első *NextHunt* hívás előtt sem szabad meghívni. Ha a programod nem tartja be ezeket a feltételeket, vagy egyetlen kincsvadászaton belül több mint 1000 lekérdezést hajt végre, vagy a *Query* hívásakor az x és/vagy y koordináták a megengedett tartományon kívül esnek, akkor a könyvtár leállítja a programot, és futási hiba (RTE) eredményt ad az aktuális tesztesetre.

Akkor tekintjük úgy, hogy egy kincset megtaláltál, ha legalább egyszer lekérdezted azt a cellát, amelyben található. Ha a programod meghívja a *NextHunt* függvényt, mielőtt az aktuális kincsvadászat összes kincsét megtalálta volna, az nem számít hibának, de befolyásolja a pontszámot (erről lentebb részletesebben).

A könyvtár használatához a programodnak tartalmaznia kell a

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

importáló sort. Az importálandó fájl innen tölthető le: `treasurehuntlib.h`.

A megoldás elkészítésének segítésére a könyvtár egy lokálisan használható implementációja is rendelkezésre áll itt: `treasurehuntlib-public.cpp`. Ahhoz, hogy ezt a programoddal együtt lefordítsd, add hozzá a nevét a fordító paramétereinek, például így:

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

ahol a `foo.cpp` a megoldásodat tartalmazó fájl neve.

A lokális teszteléshez letölthető `treasurehuntlib-public.cpp` a korábban felsoroltakon túl deklarálja a `void InitFromFile(const char *fileName)` függvényt is. Ez egy vagy több kincsvadászat leírását olvassa be a paraméterként megadott nevű fájlból. Ezt a programod elején kell meghívnod és lehetővé teszi, hogy a *NextHunt* hívások során a fájlban található kincsvadászatokat játszd le, a véletlenszerűen generáltak helyett. További leírást a használatáról (például a bemeneti fájl formátumáról) a `treasurehuntlib-public.cpp` fájlban találsz.

A kiértékelő szerveren a könyvtár egy másik implementációját fogják használni, ezért nem szabad semmilyen feltételezést tenned annak pontos működéséről. Feltételezheted azonban, hogy a globális névtér nem tartalmaz az itt felsoroltakon (*NextHunt*, *Query* és az öt konstans) kívül további deklarációkat.

A programod nem olvashat a standard bemenetről és nem írhat a standard kimenetre, mivel ezeket a kiértékelő szerveren futó könyvtárimplementációnk az értékelőrendszer többi részével való kommunikációra használja.

Bemenet

Korlátok

- $2 \leq N \leq 1\,000\,000$
- $1 \leq K \leq 3$
- Egyetlen programfuttatás során legfeljebb 100 000 kincsvadászat fordul elő.
- Egyetlen kincsvadászaton belül legfeljebb 1000 lekérdezés hajtható végre.
- Az értékelő **nem adaptív**, ami azt jelenti, hogy a rácsok mérete és a kincsek pozíciói a programfuttatás előtt rögzítve lettek.

Részfeladatok

1. részfeladat (10 pont): $K = 1$.
2. részfeladat (30 pont): $K = 2$.
3. részfeladat (60 pont): $K = 3$.

Pontozás

Egy részfeladat több tesztesetből (a program több futtatásából) állhat, és minden teszteset több kincsvadászatot tartalmazhat. A pontozás szempontjából egy adott részfeladat összes kincsvadászatát együtt értékeli, függetlenül attól, hogy eredetileg hogyan oszlottak meg a tesztesetek között. Az i -edik kincsvadászat esetén jelölje a rács méretét N_i , a kincsek számát K_i , a programod által végrehajtott lekérdezések számát Q_i , a megtalált kincsek számát pedig F_i . Továbbá jelölje S az adott részfeladatra szereshető összes pontot. Ekkor a programod által elért pontszám a részfeladatban:

- Ha a programod minden kincsvadászatban megtalálta az összes kincset (azaz $F_i = K_i$ minden i -re), akkor a pontszáma a $t_i = \frac{Q_i}{\lceil \log_2 N_i \rceil}$ értéktől függ:

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

- Ha a programod nem minden esetben találta meg az összes kincset (azaz $F_i < K_i$ legalább egy i -re), akkor a pontszáma:

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

Más szóval, a pontok felét az összes kincs megtalálásáért kapod, a másik felét pedig azért, hogy ezt minél kevesebb lekérdezéssel teljesítsd.

- A maximális pontszámhoz a megoldásodnak legfeljebb $11 \lceil \log_2 N_i \rceil$ lekérdezéssel kell megtalálnia az összes kincset.
- $11 \lceil \log_2 N_i \rceil$ és $20 \lceil \log_2 N_i \rceil$ lekérdezés között a pontszám lineárisan csökken;
- ha pedig a megoldásod több, mint $20 \lceil \log_2 N_i \rceil$ lekérdezést használ, akkor csak az összes kincs megtalálásáért járó pontokat kapja meg. (A $\lceil \cdot \rceil$ jelölés azt jelenti, hogy a $\log_2 N_i$ értékét a legközelebbi nagyobb, vagy egyenlő egész számra kerekítjük.)

<!--If the formulas above result in a non-integer number of points for the subtask, it will be rounded to the nearest integer.

If your program has a runtime error or doesn't adhere to the aforementioned protocol in using the library, it will get 0 points for the whole subtask. To receive partial points for finding a subset of the treasures, your program therefore needs to finish the search gracefully by calling *NextHunt*.-->

Ha a fenti képletek nem egész pontszámot eredményeznek egy részfeladatra, akkor azt a legközelebbi egész számra kerekítik.

Ha a programod futási hibát okoz, vagy nem tartja be a könyvtár használatának fent ismertetett protokollját, akkor 0 pontot kap az egész részfeladatra. Ahhoz, hogy részpontszámot kapj a kincsek csak egy részhalmazának megtalálásáért, a keresést szabályosan kell befejezned a *NextHunt* meghívásával.

Példa

Függvényhívás	Visszkapott értékek
$\text{NextHunt}(N, K)$	$N = 4, K = 1$
$\text{Query}(2, 0)$	$9 = \text{DIR_DOWN} + \text{DIR_RIGHT}$
$\text{Query}(3, 1)$	$8 = \text{DIR_DOWN}$
$\text{Query}(3, 2)$	$0 = \text{TREASURE}$
$\text{NextHunt}(N, K)$	$N = -1, K = -1$