

CEOI 2026



Ден 1
(български)

Day 1
(Bulgarian)

Наблюдатели на птици

Ограничение по време: 10 s Ограничение по памет: 512 MiB

Обществото на наблюдателите на птици на Сан Сериф има любопитно раздута и постоянно променяща се вътрешна структура. Обществото се състои от n клона, като всеки член принадлежи на точно един клон. Клоновете са номерирани с числата от 1 до n , като i -тият клон има m_i членове. Така обществото се състои от общо $M = m_1 + m_2 + \dots + m_n$ членове.

Всеки клон се ръководи от точно един от членовете си, който се нарича *секретар* на клона. Секретарите са номерирани точно както и съответните клонове, така че (за всяко $i = 1, \dots, n$) секретар i ръководи клон i .

Освен това секретарите са организирани в йерархична структура чрез система от менторства: всеки секретар без един има *ментор*, който е секретар на някой клон. Единственият секретар без ментор е *президентът* на обществото. Ако секретар a е ментор на секретар b , считаме, че секретар b е *последовател* на секретар a . Няма секретар, който е пряко или косвено ментор на себе си. Това означава, че като следваме поредицата от секретар към техния ментор, менторът на този ментор и т.н., накрая винаги достигахме президента.

Дефинираме *влиянието* на секретар като сума от броя членове на неговия клон и влиянието на неговите последователи (ако има такива). Тогава секретарят с най-голямо влияние е президентът, чието влияние е винаги равно на M . Секретар се нарича *старши секретар*, ако влиянието му е $\geq M/2$.

Уставът на обществото постановява, че старшият секретар с най-ниско влияние (сред всички старши секретари) изпълнява ролята на *ковчежник* на обществото.

От време на време секретар (освен президента) може да *смени своя ментор* и вече да стане последовател на различен ментор спрямо преди (при условие, че новият му ментор не е бил негов последовател или последовател на негов последовател и т.н.). Заради това може да се промени влиянието на някои секретари и ролята на ковчежника може да се поеме от друг секретар спрямо преди.

Задача

Напишете програма, която чете описанието на началното състояние на обществото и поредица от смени на ментори. Програмата трябва да отпечата кой е ковчежникът в началното състояние на обществото, както и след всяка смяна на ментор.

Вход

Първият ред съдържа две цели числа n и q , отделени с един интервал; n е броят на клоновете, q е броят на смените на ментори.

Следващите n реда описват началното състояние на обществото. i -тият от тези редове съдържа две цели числа, s_i и m_i , отделени с един интервал; s_i е менторът на секретар i (т.е. на секретарят, който ръководи клон i), докато m_i е броят на членовете на клон i . Стойността $s_i = 0$ обозначава, че секретар i е президент на обществото и затова няма ментор.

Оставащите q реда описват смените на ментори. j -тият от тези редове съдържа две цели числа, \hat{x}_j и \hat{z}_j , отделени с един интервал. Значението на тези числа е

следното. Нека означим с t_j (за $j = 0, \dots, q$) ковчезникът след първите j смени на ментори (така t_0 е ковчезникът в началото преди първата смяна на ментор). Тогава j -тата смяна на ментор е, че секретар z_j става новият ментор на секретар x_j , където $x_j = 1 + ((t_{j-1} + \hat{x}_j) \bmod n)$ и $z_j = 1 + ((t_{j-1} + \hat{z}_j) \bmod n)$. Целта на това представяне на стойностите на x_j и z_j е, за да е задължително Вашата програма да обработва смените на ментори в реда, в който се появяват.

Смените на ментори във входните данни ще бъдат винаги валидни, т.е. z_j няма да е равно на x_j , както и z_j няма да е последовател на x_j , няма да е последовател на последовател на x_j и т.н. Възможно е z_j вече да е бил ментор на x_j точно преди j -тата промяна (така че да не се случва реална промяна).

Обърнете внимание, че ако Вашата програма намери грешно t_j по някое време, то ще декодира грешно поредицата от входове $\hat{x}_{j+1}, \hat{z}_{j+1}$ и т.н., и може да се получи резултат RTE (грешка по време на изпълнение) вместо WA (грешен отговор), защото е възможно грешно декодираните входове да са невалидни (т.е. грешно да се получи z_{j+1} , който е последовател на x_{j+1}).

Ограничения

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

Подзадачи

- Подзадача 1 (15 точки): $n \leq 100$
- Подзадача 2 (10 точки): $n \leq 1000$
- Подзадача 3 (50 точки): $n \leq 300\,000$
- Подзадача 4 (25 точки): Няма допълнителни ограничения.

Изход

Отпечатайте числата t_0, t_1, \dots, t_q , всяко на отделен ред, където t_j е ковчезникът след първите j смени на ментор. Разбира се, всяко t_j трябва да е цяло число в интервала $1 \leq t_j \leq n$.

Пример

Вход	Изход
7 2	2
0 1	2
1 3	3
1 3	
2 3	
2 1	
5 2	
5 1	
3 7	
2 7	

Коментар

В началото, секретар 2 е ковчезник (така че $t_0 = 2$). При първата смяна на ментор, прочитаем $\hat{x}_1 = 3$ и $\hat{z}_1 = 7$ и пресмятаме $x_1 = 1 + ((2 + 3) \bmod 7) = 6$ и $z_1 = 1 + ((2 + 7) \bmod 7) = 3$; така че секретар 3 става новият ментор на секретар 6; секретар 2 остава ковчезник (така че $t_1 = 2$). При втората смяна на ментор, прочитаем $\hat{x}_2 = 2$ и $\hat{z}_2 = 7$ и пресмятаме $x_2 = 1 + ((2 + 2) \bmod 7) = 5$ и $z_2 = 1 + ((2 + 7) \bmod 7) = 3$; така че секретар 3 става новият ментор на секретар 5 и освен това става новият ковчезник (така че $t_2 = 3$).

DFS

Ограничение по време: 1 s Ограничение по памет: 256 MiB

Може би вече сте запознати с известния алгоритъм DFS (обхождане в дълбочина) за обхождане на граф. В тази задача разглеждаме единствено свързани неориентирани прости графи (без примки и мултиребра), чиито върхове са номерирани с числата $0, 1, \dots, n - 1$. DFS алгоритъмът ще отпечата по интересен начин дълбочините и върховете, както следва:

DFS(d, v):

отпечатай d/v

маркирай връх v за посетен

W = списък със съседите на v , подреден в нарастващ ред на номерата за всяко w в W :

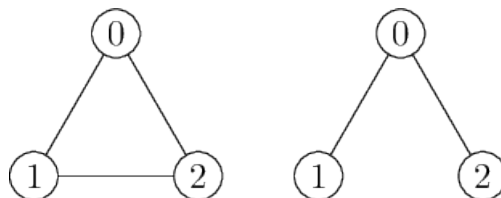
ако връх w не е бил посетен:

DFS($d + 1, w$)

Напишете програма, която намира броя на различните графи, за които извикването DFS($0, n - 1$) отпечатва текст, идентичен на този, който получавате като вход. Например, текстът

```
0/2
1/0
2/1
```

се получава от извикването DFS($0, 2$) за всеки от следните два свързани неориентирани прости графа с 3 върха:



Вход

Входът е текстът, получен от извикването DFS($0, n - 1$) за някакъв неизвестен свързан неориентиран прост граф с n върха. Затова входът се състои от n реда във формата d/v , като първият ред задължително е $0/n-1$.

Ограничения

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

Подзадачи

- Подзадача 1 (10 точки): $n \leq 6$.
- Подзадача 2 (20 точки): $n \leq 500$.

- Подзадача 3 (20 точки): $n \leq 10^4$.
- Подзадача 4 (10 точки): За всяко $i \in \{2, \dots, n\}$, i -тият ред е $i-1/i-2$.
- Подзадача 5 (20 точки): За всяко $i \in \{2, \dots, n\}$, i -тият ред е $i-1/v$ за някое $v \in \{0, \dots, n-2\}$.
- Подзадача 6 (20 точки): Няма допълнителни ограничения.

Изход

Отпечатайте броя на различните графи с търсеното свойство. Понеже това число може да е много голямо, то намерете само остатка му по модул 1 000 000 007.

Пример

Вход	Изход
0/2	2
1/0	
2/1	

Лов за съкровища

Ограничение по време: 8 s Ограничение по памет: 256 MiB

Търсите отдавна изгубени съкровища в поле с размер $N \times N$, използвайки магически компас. Общо $K \leq 3$ съкровищни сандъка са скрити в различни клетки на полето. Вашата цел е проста: да откриете всички!

Когато поставите компаса в дадена клетка, той ще намери най-кратките пътища до съкровищен сандък, използвайки само движения наляво, нагоре, надясно и надолу (т.е. най-близък по Манхатъново разстояние сандък). След това ще покаже посоката на първата стъпка от пътя. Ако има няколко валидни първи движения, водещи до най-близък сандък (или до няколко от тях), компасът ще върне всички от тях. Ако клетката съдържа съкровищен сандък, компасът ще индикира това.

Всеки път, когато намерите съкровищен сандък, изпразвате съдържанието му, но не премахвате самия сандък – той е твърде тежък. Вашият компас не знае дали даден сандък е пълен или празен. Той ще сочи към най-близък сандък, независимо дали празен или пълен.

Опитайте се да намерите всички съкровищни сандъци с възможно най-малко заявки към компаса.

Задача

Това е интерактивна задача. При всеки тест (т.е. при всяко изпълнение на програмата Ви) тя трябва да реши няколко отделни лова за съкровища. Ще комуникирате с проверяващата система чрез библиотека, предоставена от организаторите. Тази библиотека съдържа следните декларации:

- **void** *NextHunt*(**int** &*N*, **int** &*K*) — извикайте тази функция, за да започнете следващия лов за съкровища. Тя ще запише размера на полето в променливата *N* и броя на съкровищата в *K*. Ако няма повече ловове за решаване в текущото изпълнение, функцията ще запише -1 в *N* и *K*; в такъв случай програмата Ви трябва да терминира с изходен код 0. Обърнете внимание, че можете да извикате тази функция, преди да сте намерили всички съкровища в текущия лов, например ако се стремите към частични точки.
- **enum** { *TREASURE* = 0, *DIR_RIGHT* = 1, *DIR_UP* = 2, *DIR_LEFT* = 4, *DIR_DOWN* = 8 }; — това са константи, използвани в стойностите, които функцията *Query* връща (вижте по-долу).
- **int** *Query*(**int** *x*, **int** *y*) — ако клетката с координати (x, y) съдържа съкровищен сандък, функцията връща *TREASURE*. В противен случай, връща сумата на една или повече от константите *DIR_RIGHT*, *DIR_UP*, *DIR_LEFT* и *DIR_DOWN*, за да обозначи кои движения връща компасът, когато е поставен в клетката (x, y) . Координатите *x* и *y* трябва да бъдат цели числа от 0 до $N-1$. (Забележка: в тази задача координатата *y* нараства отгоре надолу.)

След като *NextHunt* запише $N = K = -1$, програмата Ви не трябва повече да извиква нито *NextHunt*, нито *Query*. Тя също така не трябва да извиква *Query* преди

първото извикване на *NextHunt*. Ако програмата Ви не спази тези ограничения, или ако направи повече от 1000 заявки в рамките на един лов на съкровища, или ако подаде стойности за x и/или y извън допустимите граници при извикване на *Query*, библиотеката ще прекрати програмата и ще даде резултат `run-time-error` (RTE) за съответния тест.

Съкровище се счита за намерено, ако сте направили поне една заявка към клетката, която го съдържа. Ако програмата Ви извика *NextHunt* преди да е намерила всички съкровища от текущия лов, това не се счита за грешка, но ще повлияе на резултата Ви (вижте оценяването по-долу).

За да използва библиотеката, програмата Ви трябва да включи хедър файла `treasurehuntlib.h`:

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

Можете да изтеглите този хедър файл от тук: `treasurehuntlib.h`.

За Ваша помощ, е предоставена проста реализация на библиотеката тук: `treasurehuntlib-public.cpp`. За да я компилирате заедно с програмата си, просто добавете името ѝ като параметър на компилатора, например:

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

ако `foo.cpp` е името на файла, съдържащ Вашето решение.

На оценяващата система ще бъде използвана различна реализация на библиотеката, така че не бива да правите никакви предположения за начина, по който тя работи. Все пак може да приемете, че тя не замърсява глобалното пространство от имена с други декларации освен изброените по-горе (*NextHunt*, *Query* и петте константи).

Вашата програма не трябва да чете от стандартния вход и не трябва да пише в стандартния изход, защото те ще бъдат използвани от нашата реализация на библиотеката на сървъра за оценяване за комуникация с останалата част от средата за оценяване.

Вход

Ограничения

- $1 \leq N \leq 10^6$
- $1 \leq K \leq 3$
- В рамките на едно изпълнение на програмата ще има най-много 100 000 лова на съкровища.
- В рамките на един лов на съкровища можете да направите най-много 1000 заявки.
- Оценяващата система не е адаптивна.

Подзадачи

- Подзадача 1 (10 точки) $K = 1$
- Подзадача 2 (30 точки) $K = 2$
- Подзадача 3 (60 точки) $K = 3$

Оценяване

Една подзадача може да се състои от няколко теста (няколко изпълнения на програмата), а всеки тест може да съдържа няколко лова на съкровища. За целите на оценяването всички ловове на съкровища от дадена подзадача се разглеждат заедно, независимо как са били разпределени между тестовете. За i -тия лов на съкровища ще означим размера на полето с $N_i \times N_i$, броя на съкровищата с K_i , броя на заявките, направени от програмата Ви, с Q_i , а броя на намерените съкровища с F_i . Освен това нека S е общият брой точки, предназначени за тази подзадача. Тогава броят точки, които програмата Ви получава за тази подзадача, е:

- Ако програмата Ви винаги е намерила всички съкровища (т.е. ако $F_i = K_i$ за всички i), точките ѝ зависят от $t_i = \frac{Q_i}{\lceil \log_2 N_i \rceil}$:

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

- Ако програмата Ви не винаги е намерила всички съкровища (т.е. ако съществува i , такова че $F_i < K_i$), тя получава:

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

С други думи, получавате половината от точките за намирането на всички съкровища и другата половина — за това да ги намирате с възможно най-малко заявки. За максимален резултат решението Ви трябва да намира всички съкровища с не повече от $11 \lceil \log_2 N_i \rceil$ заявки. Между $11 \lceil \log_2 N_i \rceil$ и $20 \lceil \log_2 N_i \rceil$ заявки резултатът намалява линейно; а ако програмата прави повече от $20 \lceil \log_2 N_i \rceil$ заявки, ще получи само първата половина от точките за намирането на всички съкровища.

Ако горните формули доведат до нецелочислен брой точки за подзадачата, резултатът се закръгля до най-близкото цяло число.

Ако програмата Ви получи грешка по време на изпълнение или не спазва описания по-горе протокол за използване на библиотеката, тя ще получи 0 точки за цялата подзадача. За да получите частични точки за намиране само на част от съкровищата, все пак трябва вярно да приключите търсенето чрез извикване на *NextHunt*.

Пример

Извикване	Върната стойност
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)	$N = -1, K = -1$