

CEOI 2026



День 1
(Українська)

Day 1
(Ukrainian)

Слонопаси

Ліміт часу: 10 s Ліміт пам'яті: 512 MiB

Товариство слонопасів має на диво роздуту та непостійну внутрішню структуру. Товариство складається з n відділень, і кожен член Товариства належить рівно до одного відділення. Відділення пронумеровані від 1 до n , і i -те відділення має m_i членів. Таким чином, Товариство має загалом $M = m_1 + m_2 + \dots + m_n$ членів.

Кожне відділення очолює один із його членів, який у цій ролі називається *керівником* відділення. Керівники нумеруються так само, як і відділення, тому (для кожного $i = 1, \dots, n$) керівник i очолює відділення i .

Крім того, керівники організовані ієрархічно через систему наставництва: кожен керівник, крім одного, має *наставника*, яким є керівник іншого відділення. Єдиний керівник без наставника — це *Президент* Товариства. Якщо керівник a є наставником керівника b , то ми також кажемо, що керівник b є *учнем* керівника a .

Жоден керівник не є, прямо чи опосередковано, наставником самого себе; отже, якщо рухатися від керівника до його наставника, потім до наставника його наставника і так далі, то зрештою ми завжди дійдемо до Президента.

Ми визначаємо *вплив* керівника як суму кількості членів у його відділенні та впливів усіх його учнів (якщо в нього вони є). Легко побачити, що керівником із найбільшим впливом є Президент, вплив якого завжди дорівнює M . Керівник називається *старшим керівником*, якщо його вплив $\geq M/2$.

Кодекс Товариства визначає, що старший керівник із найменшим впливом (серед усіх старших керівників) виконує роль *Казначея* Товариства.

Час від часу керівник (окрім Президента) може *змінити підпорядкування*, після чого він стає учнем іншого наставника, ніж раніше (за умови, що його новий наставник не є одним із його учнів, учнем одного з його учнів тощо). Через це може статися, що вплив деяких посадовців зміниться і роль Казначея перейде до іншого керівника.

Завдання

Напишіть програму, яка читає опис початкового стану Товариства та послідовність змін підпорядкування. Ваша програма повинна вивести, хто є Казначеем у початковому стані Товариства, а також після кожної зміни підпорядкування.

Вхідні дані

Перший рядок містить два цілі числа n і q , розділені пробілом; n — це кількість керівників, а q — кількість змін підпорядкування.

Наступні n рядків описують початковий стан Товариства.

i -й із цих рядків містить два цілі числа s_i і m_i , розділені пробілом; s_i — це наставник керівника i (тобто керівника, який очолює відділення i), а m_i — кількість членів відділення i . Значення $s_i = 0$ означає, що посадовець i є Президентом Товариства і тому не має наставника.

Решта q рядків описують зміни підпорядкування.

j -й із цих рядків містить два цілі числа \hat{x}_j і \hat{z}_j , розділені пробілом. Значення цих чисел таке. Позначимо через t_j (для $j = 0, \dots, q$) Казначея після перших j змін підпорядкування (отже, t_0 — це початковий Казначей перед першою зміною

підпорядкування). Тоді j -та зміна підпорядкування полягає в тому, що керівник z_j стає новим наставником керівника x_j , де $x_j = 1 + ((t_{j-1} + \hat{x}_j) \bmod n)$ та $z_j = 1 + ((t_{j-1} + \hat{z}_j) \bmod n)$. Мета такого подання значень x_j і z_j полягає в тому, щоб змусити вашу програму обробляти зміни підпорядкування в тому порядку, у якому вони задані.

Зміни підпорядкування у вхідних даних завжди будуть коректними, тобто z_j не дорівнюватиме x_j , а також z_j не буде учнем x_j , учнем учня x_j тощо. Однак можливо, що z_j уже був наставником x_j безпосередньо перед j -ю зміною підпорядкування (так що фактично в цей момент нічого не зміниться).

Зауважте, що якщо ваша програма в якийсь момент обчислить неправильний результат t_j , то вона також неправильно декодує наступні вхідні значення \hat{x}_{j+1} , \hat{z}_{j+1} тощо і може отримати вердикт RTE (помилка виконання) замість WA (неправильна відповідь), оскільки неправильно декодовані вхідні дані можуть виявитися некоректними (наприклад, вона може помилково отримати z_{j+1} , який є учнем x_{j+1}).

Обмеження

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

Підзадачі

- Підзадача 1 (15 балів): $n \leq 100$
- Підзадача 2 (10 балів): $n \leq 1000$
- Підзадача 3 (50 балів): $n \leq 300\,000$
- Підзадача 4 (25 балів): Без додаткових обмежень.

Вихідні дані

Виведіть числа t_0, t_1, \dots, t_q , кожне в окремому рядку, де t_j — це Казначей після перших j змін підпорядкування. Звісно, кожне t_j повинно бути цілим числом із діапазону $1 \leq t_j \leq n$.

Приклад

Вхід	Вихід
7 2	2
0 1	2
1 3	3
1 3	
2 3	
2 1	
5 2	
5 1	
3 7	
2 7	

Коментар

Спочатку керівник 2 є Казначеем (звідси $t_0 = 2$). Під час першої зміни підпорядкування ми зчитуємо $\hat{x}_1 = 3$ та $\hat{z}_1 = 7$ й обчислюємо $x_1 = 1 + ((2 + 3) \bmod 7) = 6$ і $z_1 = 1 + ((2 + 7) \bmod 7) = 3$; отже, керівник 3 стає новим наставником керівника 6; керівник 2 залишається Казначеем (звідси $t_1 = 2$). Під час другої зміни підпорядкування ми зчитуємо $\hat{x}_2 = 2$ та $\hat{z}_2 = 7$ й обчислюємо $x_2 = 1 + ((2 + 2) \bmod 7) = 5$ і $z_2 = 1 + ((2 + 7) \bmod 7) = 3$; отже, керівник 3 стає новим наставником керівника 5 і також стає новим Казначеем (звідси $t_2 = 3$).

DFS

Ліміт часу: 1 s Ліміт пам'яті: 256 MiB

Можливо, ви вже знайомі з відомим алгоритмом DFS (depth-first search, пошук у глибину) для обходу графа. У цій задачі розглядатимуться лише зв'язні неорієнтовані прості графи (без петель і кратних ребер) з вершинами, пронумерованими $0, 1, \dots, n-1$, а сам алгоритм DFS виводитиме глибини та номери вершин у такому порядку:

DFS(d, v):

вивести d/v

позначити вершину v як відвідану

W = список сусідів вершини v , впорядкований за зростанням номерів для кожної $w \in W$:

якщо вершину w ще не було відвідано:

DFS($d + 1, w$)

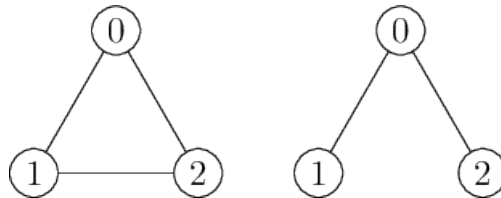
Напишіть програму, яка виведе кількість різних графів, для яких виклик DFS($0, n - 1$) генерує такий самий вивід, що й заданий у вхідних даних. Наприклад, вивід

0/2

1/0

2/1

є результатом виклику DFS($0, 2$) на будь-якому з двох наведених нижче зв'язних неорієнтованих простих графів із 3 вершинами:



Вхідні дані

Вхідні дані є результатом виклику DFS($0, n - 1$) для деякого невідомого зв'язного неорієнтованого простого графа з n вершинами. Отже, вхідні дані складаються з n рядків у форматі d/v , причому перший рядок має вигляд $0/n-1$.

Обмеження

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

Підзадачі

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

- Підзадача 4 (10 балів): Для кожного $i \in \{2, \dots, n\}$, i -й рядок вхідних даних має вигляд $i-1/i-2$.
- Підзадача 5 (20 балів): Для кожного $i \in \{2, \dots, n\}$, i -й рядок вхідних даних має вигляд $i-1/v$ для деякого $v \in \{0, \dots, n-2\}$.
- Підзадача 6 (20 балів): Без додаткових обмежень.

Вихідні дані

Виведіть кількість різних графів, що задовольняють задану властивість. Оскільки це число може бути дуже великим, виведіть результат за модулем 1 000 000 007

Приклад

Вхід	Вихід
0/2	2
1/0	
2/1	

Пошук горіхів

Ліміт часу: 8 s Ліміт пам'яті: 256 MiB

Ви шукаєте давно загублений горіх-скарб на величезному полі клітин розміром $N \times N$ за допомогою чарівної білки, на ім'я Ілля. Усього на полі заховано $K \leq 3$ скринь із горіхами в різних клітинах. Ваша мета проста: знайти їх усі!

Коли ви ставите білку на одну з клітин, вона знаходить найкоротші шляхи до якоїсь скрині, використовуючи лише переміщення ліворуч, угору, праворуч і вниз (тобто до найближчої скрині за Манхеттенською відстанню). Потім вона показує напрямок першого кроку. Якщо існує кілька можливих перших кроків, що ведуть до якоїсь найближчої скрині (або до кількох таких скринь), білка поверне всі ці напрямки. Якщо клітина містить скриню, білка вкаже на це замість напрямку.

Щоразу, коли ви знаходите скриню з горіхом, ви спорожнюєте її вміст, але не можете прибрати саму скриню — вона надто важка. Ваша білка не знає, порожня скриня чи повна. Білка показуватиме шлях до найближчої скрині незалежно від того, порожня вона чи повна.

Спробуйте знайти розташування всіх скринь із горіхом, використовуючи якомога менше запитів до білки.

Задача

Це інтерактивна задача. У кожному тестовому випадку (тобто під час кожного запуску) програма повинна буде розв'язати кілька пошуків горіхів. Для взаємодії з перевіряючою системою слід використовувати бібліотеку, надану організаторами. Ця бібліотека містить такі оголошення:

- **void** *NextHunt*(**int** &*N*, **int** &*K*) — викликайте цю функцію, щоб розпочати наступний пошук горіхів. Вона задасть розмір сітки у змінній *N* та кількість горіхів у *K*. Якщо в поточному запуску більше немає пошуків горіхів, які потрібно розв'язати, функція встановить *N* і *K* в -1 ; у цьому випадку слід завершити програму з кодом виходу 0 . Зверніть увагу, що ви можете викликати цю функцію ще до того, як знайдете усі горіхи поточного пошуку, наприклад, якщо намагаєтеся отримати лише часткові бали.
- **enum** {*TREASURE* = 0, *DIR_RIGHT* = 1, *DIR_UP* = 2, *DIR_LEFT* = 4, *DIR_DOWN* = 8 }; — це константи, які використовуються у значеннях, що повертає функція *Query* (див. нижче).
- **int** *Query*(**int** *x*, **int** *y*) — якщо клітина з координатами (*x*, *y*) містить скриню, ця функція повертає *TREASURE*. Інакше вона повертає суму однієї або кількох констант *DIR_RIGHT*, *DIR_UP*, *DIR_LEFT* і *DIR_DOWN*, вказуючи, які переміщення повертає білка, коли її розміщено в клітині (*x*, *y*). Координати *x* і *y* повинні бути цілими числами від 0 до $N - 1$. (Примітка: у цій задачі координати *y* зростають зверху вниз.)

Після того як *NextHunt* присвоє $N = K = -1$, ваша програма більше не повинна викликати ні *NextHunt*, ні *Query*. Вона також не повинна викликати *Query* до

першого виклику *NextHunt*. Якщо програма не дотримається цих вимог, або якщо вона виконає понад 1000 запитів під час одного пошуку горіхів, або якщо передасть у *Query* значення x та/або y поза дозволеним діапазоном, бібліотека завершить роботу програми та поверне вердикт помилка виконання (RTE) для поточного тестового випадку.

Горіх вважається знайденим, якщо ви хоча б один раз виконали запит до клітини, яка його містить. Якщо програма викликає *NextHunt* до того, як знайде всі горіхи поточного пошуку, це не вважається помилкою, але вплине на результат (детальніше про оцінювання нижче).

Для доступу до бібліотеки програма повинна включити заголовний файл `treasurehuntlib.h`:

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

Цей файл можна завантажити тут: `treasurehuntlib.h`.

Для допомоги в розробці рішення доступна проста реалізація бібліотеки: `treasurehuntlib-public.cpp`. Аби скомпілювати її разом зі своєю програмою, просто додайте її ім'я як аргумент компілятора, наприклад:

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

де `foo.cpp` — це ім'я файлу, що містить ваше рішення.

Реалізація у `treasurehuntlib-public.cpp`, окрім наведених вище оголошень, також підтримує ще одну функцію, `void InitFromFile(const char *fileName)`, яка зчитує список конфігурацій пошуку горіхів з файлу та дозволяє грати в гру саме з ними замість випадково згенерованих конфігурацій, які бібліотека створює самостійно. Докладнішу інформацію можна знайти безпосередньо у файлі `treasurehuntlib-public.cpp`.

На сервері перевірки буде використовуватися інша реалізація бібліотеки, тому не слід робити жодних припущень щодо того, як саме вона працює. Водночас можна вважати, що вона не засмічує глобальний простір імен жодними оголошеннями, окрім наведених вище (*NextHunt*, *Query* та п'яти констант).

Ваша програма не повинна читати зі стандартного вводу чи записувати у стандартний вивід, оскільки вони будуть використовуватися реалізацією бібліотеки на сервері перевірки для зв'язку з рештою середовища оцінювання.

Вхідні дані

Обмеження

- $1 \leq N \leq 10^6$
- $1 \leq K \leq 3$
- Під час одного запуску програми буде не більше ніж 100 000 пошуків горіхів.
- Під час одного пошуку горіхів можна виконати не більше ніж 1000 запитів.
- Система оцінювання не адаптивна.

Підзадачі

- Підзадача 1 (10 балів) $K = 1$
- Підзадача 2 (30 балів) $K = 2$
- Підзадача 3 (60 балів) $K = 3$.

Оцінювання

Підзадача може складатися з кількох тестових випадків (кількох запусків програми), а кожен тестовий випадок може містити кілька пошуків горіхів. Для цілей оцінювання всі пошуки в межах однієї підзадачі оцінюються разом, незалежно від того, як вони були розподілені між тестовими випадками. Для i -го пошуку горіхів позначимо розмір поля як $N_i \times N_i$, кількість горіхів як K_i , кількість запитів, виконаних програмою, як Q_i , а кількість знайдених горіхів як F_i . Також нехай S — загальна кількість балів, доступних за цю підзадачу. Тоді кількість балів, отриманих програмою за цю підзадачу:

- Якщо програма завжди знаходила всі горіхи (тобто якщо $F_i = K_i$ для всіх i), її бали залежать від $t_i = \frac{Q_i}{\lceil \log_2 N_i \rceil}$

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

- Якщо програма не завжди знаходила всі горіхи (тобто існує i , для якого $F_i < K_i$), вона отримує

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

Іншими словами, половину балів ви отримуєте за те, що знаходите усі горіхи, а іншу половину — за те, що знаходите їх, використовуючи якомога менше запитів. Для максимального результату ваше рішення повинно знаходити всі горіхи не більше ніж за $11 \lceil \log_2 N_i \rceil$ запитів. Між $11 \lceil \log_2 N_i \rceil$ і $20 \lceil \log_2 N_i \rceil$ запитами оцінка лінійно зменшується; якщо ж рішення виконає більше ніж $20 \lceil \log_2 N_i \rceil$ запитів, воно отримає лише першу половину балів за знаходження всіх горіхів. (Символи $\lceil \cdot \rceil$ означають, що значення $\log_2 N_i$ округлюється вгору до найближчого цілого числа.)

Якщо наведені вище формули дають нецілу кількість балів за підзадачу, результат округлюється до найближчого цілого числа.

Якщо програма завершиться з помилкою виконання або не дотримуватиметься описаного вище протоколу використання бібліотеки, вона отримає 0 балів за всю підзадачу. Щоб отримати часткові бали за знаходження лише підмножини горіхів, програма повинна коректно завершити пошук, викликавши *NextHunt*.

Приклад

Виклик	Повернене значення
$\text{NextHunt}(N, K)$	$N = 4, K = 1$
$\text{Query}(2, 0)$	$\text{DIR_DOWN} + \text{DIR_RIGHT} = 9$
$\text{Query}(3, 1)$	DIR_DOWN
$\text{Query}(3, 2)$	TREASURE
$\text{NextHunt}(N, K)$	$N = -1, K = -1$