

CEOI 2026



Giorno 1
(italiano)

Day 1
(Italian)

Tesoriere ANPC

Limite di tempo: 10 s Limite di memoria: 512 MiB

ANPC ha una struttura interna curiosamente arzigogolata e in continua evoluzione. L'Associazione è composta da n sezioni, e ogni membro dell'Associazione appartiene a esattamente una sezione. Le sezioni sono numerate da 1 a n e la i -esima sezione ha m_i membri. Pertanto, l'Associazione ha un totale di $M = m_1 + m_2 + \dots + m_n$ membri.

Ogni sezione è guidata da uno dei suoi membri, che in questo ruolo viene chiamato il *funzionario* della sezione. I funzionari sono numerati allo stesso modo delle sezioni, cosicché per ogni $i = 1, \dots, n$ il funzionario i sia colui che è a capo della sezione i .

Inoltre, i funzionari sono organizzati gerarchicamente attraverso un sistema di mentoraggio: ogni funzionario, tranne uno, ha un *mentore*, che è il funzionario di qualche altra sezione. L'unico funzionario senza un mentore è il *Presidente* Dario. Se il funzionario a è il mentore del funzionario b , diciamo anche che il funzionario b è un *discepolo* del funzionario a . Nessun funzionario è, direttamente o indirettamente, mentore di se stesso; pertanto, seguendo la sequenza da un funzionario al suo mentore, al mentore del suo mentore e così via, alla fine si raggiunge sempre il Presidente.

Definiamo l'*influenza* di un funzionario come la somma del numero di membri nella sua sezione e delle influenze di tutti i suoi discepoli (se ne ha). Si può facilmente notare che il funzionario con l'influenza maggiore è il Presidente, la cui influenza è sempre pari a M . Un funzionario è chiamato *funzionario anziano* se la sua influenza è almeno $M/2$.

Lo Statuto dell'Associazione specifica che il funzionario anziano con l'influenza minore (tra tutti i funzionari anziani) fungerà da *Tesoriere* dell'Associazione.

Di tanto in tanto, un funzionario (diverso dal Presidente) può *cambiare la propria affiliazione*, diventando da quel momento in poi il discepolo di un mentore diverso rispetto a prima (a condizione che il suo nuovo mentore non sia uno dei suoi discepoli, o discepolo dei suoi discepoli, ecc.). Per questo motivo, può accadere che l'influenza di alcuni funzionari cambi e che il ruolo di Tesoriere passi a un funzionario diverso rispetto a prima.

Richiesta

Scrivi un programma che legga la descrizione dello stato iniziale dell'Associazione e una sequenza di cambiamenti di affiliazione. Il tuo programma deve stampare chi è il Tesoriere nello stato iniziale dell'Associazione e dopo ogni cambiamento di affiliazione.

Dati di ingresso

La prima riga contiene due interi, n e q , separati da uno spazio; n è il numero di sezioni, q è il numero di cambiamenti di affiliazione.

Le successive n righe descrivono lo stato iniziale dell'Associazione. La i -esima di queste righe contiene due interi, s_i e m_i , separati da uno spazio; s_i è il mentore del funzionario i (ovvero del funzionario a capo della sezione i), mentre m_i è il numero di membri della sezione i . Il valore $s_i = 0$ indica che il funzionario i è il Presidente dell'Associazione e quindi non ha alcun mentore.

Le restanti q righe descrivono i cambiamenti di affiliazione. La j -esima di queste righe contiene due interi, \hat{x}_j e \hat{z}_j , separati da uno spazio. Il significato di questi interi è il seguente. Indichiamo con t_j (per $j = 0, \dots, q$) il Tesoriere dopo i primi j cambiamenti di

affiliazione (quindi t_0 è il Tesoriere iniziale prima del primo cambiamento di affiliazione). Allora il j -esimo cambiamento di affiliazione consiste nel fatto che il funzionario z_j diventa il nuovo mentore del funzionario x_j , dove

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

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

Lo scopo di questa rappresentazione dei valori x_j e z_j è costringere il tuo programma a elaborare i cambiamenti di affiliazione nell'ordine in cui appaiono.

I cambiamenti di affiliazione nei dati di input saranno sempre validi, cioè z_j non sarà uguale a x_j , né z_j sarà un discepolo di x_j , un discepolo di un discepolo di x_j , ecc. È tuttavia possibile che z_j fosse già il mentore di x_j subito prima del j -esimo cambiamento (quindi in quel momento non cambia nulla all'atto pratico).

Nota che se a un certo punto il tuo programma calcola un risultato errato per t_j , decodificherà in modo errato anche i successivi input \hat{x}_{j+1} , \hat{z}_{j+1} , ecc., e potrebbe terminare con un verdetto di runtime-error (RTE) invece di wrong-answer (WA), siccome gli input decodificati erroneamente potrebbero non essere validi.

Dati di uscita

Stampa i numeri t_0, t_1, \dots, t_q , ciascuno su una riga a sé stante, dove t_j è il Tesoriere dopo i primi j cambiamenti di affiliazione. Naturalmente, ciascun t_j deve essere un intero compreso nell'intervallo $1 \leq t_j \leq n$.

Vincoli

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

Sottoproblemi

- Sottoproblema 1 (15 punti): $n \leq 100$
- Sottoproblema 2 (10 punti): $n \leq 1000$
- Sottoproblema 3 (50 punti): $n \leq 300\,000$
- Sottoproblema 4 (25 punti): Nessun vincolo aggiuntivo.

Esempio

Input	Output
7 2	2
0 1	2
1 3	3
1 3	
2 3	
2 1	
5 2	
5 1	
3 7	
2 7	

Commento

Inizialmente, il funzionario 2 è il Tesoriere (quindi $t_0 = 2$). Nel primo cambiamento di affiliazione, leggiamo $\hat{x}_1 = 3$ e $\hat{z}_1 = 7$ e calcoliamo $x_1 = 1 + ((2 + 3) \bmod 7) = 6$ e $z_1 = 1 + ((2 + 7) \bmod 7) = 3$; quindi, il funzionario 3 diventa il nuovo mentore del funzionario 6; il funzionario 2 rimane Tesoriere (quindi $t_1 = 2$). Nel secondo cambiamento di affiliazione, leggiamo $\hat{x}_2 = 2$ e $\hat{z}_2 = 7$ e calcoliamo $x_2 = 1 + ((2 + 2) \bmod 7) = 5$ e $z_2 = 1 + ((2 + 7) \bmod 7) = 3$; quindi, il funzionario 3 diventa il nuovo mentore del funzionario 5 e diventa anche il nuovo Tesoriere (quindi $t_2 = 3$).

DFS

Limite di tempo: 1 s Limite di memoria: 256 MiB

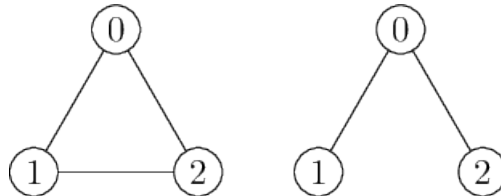
Probabilmente conosci già la DFS (depth-first search, o ricerca in profondità). In questo problema, considereremo solo grafi non orientati, semplici (cioè senza cappi e senza archi doppi) e connessi con n vertici numerati da 0 a $n-1$. L'algoritmo DFS produrrà in output profondità e vertici nel seguente modo:

```
DFS(d, v):
  stampa d/v
  marca il vertice v come visitato
  W = la lista dei vicini di v ordinati in ordine crescente
  per ogni w in W:
    se il vertice w non è ancora stato visitato:
      DFS(d + 1, w)
```

Scrivi un programma che stampi il numero di grafi distinti per cui la chiamata $\text{DFS}(0, n-1)$ produce lo stesso output fornito nel file di input. Ad esempio, l'output

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

è prodotto chiamando $\text{DFS}(0, 2)$ su uno qualsiasi dei seguenti due grafi semplici connessi non orientati a 3 vertici:



Input

L'input è l'output della chiamata $\text{DFS}(0, n-1)$ su un grafo semplice connesso non orientato sconosciuto di n vertici. L'input consiste quindi di n righe nel formato d/v , in cui la prima riga è $0/n-1$.

Output

Stampa il numero di grafi distinti che possiedono la proprietà richiesta. Dato che questo numero può essere molto grande, restituisci il risultato modulo 1 000 000 007.

Assunzioni

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

Subtask

- Subtask 1 (10 punti): $n \leq 6$.
- Subtask 2 (20 punti): $n \leq 500$.
- Subtask 3 (20 punti): $n \leq 10^4$.
- Subtask 4 (10 punti): Per ogni $i \in \{2, \dots, n\}$, la i -esima riga di input è $i-1/i-2$.
- Subtask 5 (20 punti): Per ogni $i \in \{2, \dots, n\}$, la i -esima riga di input è $i-1/v$ per qualche $v \in \{0, \dots, n-2\}$.
- Subtask 6 (20 punti): Nessun vincolo aggiuntivo.

Esempio**Input**

0/2

1/0

2/1

Output

2

Caccia al tesoro

Limite di tempo: 8 s Limite di memoria: 256 MiB

Stai cercando un tesoro perduto da tempo su un enorme griglia di $N \times N$ quadrati. Ci sono in totale $K \leq 3$ scrigni del tesoro nascosti in varie celle distinte del campo. Il tuo obiettivo è semplice: trovarne il più possibile con l'aiuto di una bussola magica.

Quando posizioni la bussola su una delle celle, essa troverà magicamente tutti i cammini di lunghezza minima verso uno scrigno del tesoro a distanza minima che utilizzano solo spostamenti a sinistra, a destra, in alto e in basso. Mostrerà quindi le direzioni di **tutti** gli spostamenti che siano il primo spostamento di un percorso verso uno scrigno di lunghezza minima. Se la cella contiene un tesoro, la bussola indicherà invece "TREASURE".

Quando trovi uno scrigno, ne saccheggia il tesoro ma non rimuovi lo scrigno perché è troppo pesante. Per questo la bussola continuerà a considerare anche i tesori già trovati quando trova i percorsi minimi.

Trova la posizione di tutti gli scrigni, utilizzando la bussola quanto meno possibile.

Task

Questo è un task interattivo. In ciascun testcase (cioè ciascuna esecuzione), il tuo programma dovrà risolvere diverse cacce al tesoro. Dovrai interagire con il grader utilizzando una libreria fornita dagli organizzatori, che contiene le seguenti dichiarazioni:

- **void** *NextHunt*(**int** &*N*, **int** &*K*): chiama questa funzione per iniziare una caccia al tesoro (all'inizio dell'esecuzione o dopo aver terminato quella precedente). La funzione scriverà la dimensione della griglia in *N* e il numero di scrigni in *K*. Se non ci sono più cacce al tesoro da risolvere nell'esecuzione corrente, la funzione imposterà invece $N = K = -1$. In tal caso, termina l'esecuzione del tuo programma. **Nota bene:** Puoi chiamare questa funzione prima di aver trovato tutti i tesori della caccia corrente, se il tuo obiettivo è solo ottenere un punteggio parziale.
- **enum** { *TREASURE* = 0, *DIR_RIGHT* = 1, *DIR_UP* = 2, *DIR_LEFT* = 4, *DIR_DOWN* = 8 }; costanti utilizzate in quanto segue
- **int** *Query*(**int** *x*, **int** *y*): Se la cella alle coordinate (*x*, *y*) contiene un tesoro, questa funzione restituisce *TREASURE*. Altrimenti, restituisce la somma di una o più costanti tra *DIR_RIGHT*, *DIR_UP*, *DIR_LEFT* e *DIR_DOWN*, corrispondenti alle mosse restituite dalla bussola quando viene posizionata sulla cella (*x*, *y*). Le coordinate *x* e *y* devono essere interi compresi tra 0 e $N - 1$. (Nota: in questo task, le coordinate *y* aumentano dall'alto verso il basso.)

Dopo che *NextHunt* ha impostato $N = K = -1$, il tuo programma non deve più chiamare né *NextHunt* né *Query*. Non deve inoltre chiamare *Query* prima della prima chiamata a *NextHunt*.

Se il tuo programma:

- non rispetta questi vincoli, o

- effettua più di 1000 query all'interno di una singola caccia al tesoro, o
- fornisce valori fuori dai limiti per x e/o y quando chiama `Query`

la libreria terminerà il tuo programma e otterrai il verdetto runtime-error (RTE) per il caso di prova corrente.

Un tesoro si considera trovato se hai effettuato una query sulla cella che lo contiene almeno una volta.

Per accedere alla libreria, il tuo programma deve includere l'header file `treasurehuntlib.h`:

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

Puoi scaricare questo header file qui: `treasurehuntlib.h`.

Per aiutarti nello sviluppo della tua soluzione, è disponibile una semplice implementazione della libreria qui: `treasurehuntlib-public.cpp`. Per compilarla insieme al tuo programma, aggiungi semplicemente il suo nome come parametro al compilatore, ad esempio:

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

se `sol.cpp` è il nome del file contenente la tua soluzione.

L'implementazione in `treasurehuntlib-public.cpp` supporta, in aggiunta alle dichiarazioni sopra menzionate, un'ulteriore funzione:

- `void InitFromFile(const char *fileName)`: questa funzione legge un elenco di cacce al tesoro da un file ed esegue la tua soluzione con esse invece di generarle casualmente. Troverai maggiori dettagli all'interno dello stesso `treasurehuntlib-public.cpp`.

L'effettiva valutazione verrà effettuata con un'implementazione diversa da quella d'esempio. Puoi comunque assumere che non inquina il namespace globale con dichiarazioni diverse da quelle sopra elencate.

Il tuo codice **non può** leggere dallo standard input né scrivere sullo standard output.

Input

Assunzioni

- $2 \leq N \leq 10^6$
- $1 \leq K \leq 3$
- All'interno di una singola esecuzione del tuo programma, ci saranno al massimo 100 000 cacce al tesoro.
- All'interno di una singola caccia al tesoro, puoi effettuare al massimo 1000 query.
- Il sistema di valutazione non è adattivo.

Subtask

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

Punteggio

Un subtask può consistere di diversi casi di prova (diverse esecuzioni del tuo programma), e ciascun caso di prova può consistere di diverse cacce al tesoro. Ai fini del punteggio, tutte le cacce al tesoro di un determinato subtask vengono valutate insieme, indipendentemente da come erano state originariamente suddivise tra i casi di prova. Per la i -esima caccia al tesoro, indichiamo la dimensione della griglia con $N_i \times N_i$, il numero di tesori con K_i , il numero di query effettuate dal tuo programma con Q_i e il numero di tesori trovati dal tuo programma con F_i . Indichiamo inoltre con S il numero totale di punti disponibili per questo subtask. Allora il numero di punti assegnati al tuo programma per questo subtask è:

- Se il tuo programma ha sempre trovato tutti i tesori (cioè se $F_i = K_i$ per ogni i), il suo punteggio dipende da $t_i = \frac{Q_i}{\lceil \log_2 N_i \rceil}$:

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

- Se il tuo programma non ha sempre trovato tutti i tesori (cioè se esiste un i tale per cui $F_i < K_i$), riceve:

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

In altre parole, si ottiene metà dei punti per aver trovato tutti i tesori, e l'altra metà per averli trovati nel minor numero possibile di query. Per un punteggio pieno, la tua soluzione dovrebbe trovare tutti i tesori in al massimo $11 \lceil \log_2 N_i \rceil$ query. Tra $11 \lceil \log_2 N_i \rceil$ e $20 \lceil \log_2 N_i \rceil$ query, il punteggio decresce linearmente; e se si effettuano più di $20 \lceil \log_2 N_i \rceil$ query, la tua soluzione otterrà solo la prima metà dei punti per aver trovato tutti i tesori.

Se le formule sopra indicate portano a un numero non intero di punti per il subtask, questo verrà arrotondato all'intero più vicino.

Esempio

Chiamata	Valore di ritorno
<code>NextHunt(N, K)</code>	$N = 4, K = 1$
<code>Query(2, 0)</code>	<code>DIR_DOWN</code> + <code>DIR_RIGHT</code> = 9
<code>Query(3, 1)</code>	<code>DIR_DOWN</code>
<code>Query(3, 2)</code>	<code>TREASURE</code>

`NextHunt(N, K)` $N = -1, K = -1$
