Thousand Islands

Thousand Islands è un arcipelago indonesiano composto da N isole, numerate da 0 a $N-1$.

Tramite M canoe, numerate da 0 a $M-1$, è possibile navigare tra le isole. Per ogni $0 \leq i < M$, la canoa i può essere ancorata all'isola $U[i]$ oppure all'isola $V[i]$, e può essere usata per spostarsi da $U[i]$ a $V[i]$ e viceversa. In particolare, quando la canoa è ancorata:

- all'isola $U[i]$, può essere usata per andare a $V[i]$, rimanendo poi ancorata a $V[i]$;
- all'isola $V[i]$, può essere usata per andare a $U[i]$, rimanendo poi ancorata a $U[i]$.

All'inizio la canoa è ancorata a $U[i]$. È possibile che più di una canoa connetta la stessa coppia di isole, e che più di una canoa sia ancorata a una stessa isola.

Miss Dengolecca deve viaggiare nell'arcipelago senza farsi scoprire. Un viaggio è **valido** se:

- Inizia e finisce nell'isola 0, e visita almeno un'isola diversa da 0.
- Alla fine del viaggio, ogni canoa è ancorata alla stessa isola a cui era ancorata all'inizio (mai lasciare tracce!). Ovvero, la canoa i deve essere ancorata all'isola $U[i]$.
- Non usa la stessa canoa due volte di seguito: dopo aver usato una certa canoa i, bisogna usare un'altra canoa j prima di poter usare i di nuovo.

Aiuta Miss Dengolecca a trovare un viaggio valido, preferibilmente fatto di al più $2\,000\,000$ di spostamenti in canoa, o determina che nessun viaggio valido esiste. Si può dimostrare che, sotto le limitazioni di questo problema, se esiste un viaggio valido ne esiste anche uno che usa al più $2\,000\,000$ di spostamenti in canoa.

Dettagli di implementazione

Devi implementare la seguente funzione:

```
union(bool, int[]) find_journey(int N, int M, int[] U, int[] V)
```

- N: il numero di isole.
- M: il numero di canoe.
- U, V: array di lunghezza M che descrivono le canoe.
- La funzione deve restituire un booleano **oppure** un array di interi:
  - Se non esiste alcun viaggio valido, deve restituire `false` (in questo caso, la soluzione ottiene punteggio pieno);

    - Se esiste almeno un viaggio valido, deve restituire `true` o un array di interi. Per ottenere punteggio pieno, deve restituire un array di dimensione non superiore a $2\,000\,000$ che rappresenta un viaggio valido, ovvero che contiene i numeri delle canoe usate, nell'ordine in cui sono usate. In tutti gli altri casi (incluso quello in cui viene restituito un array di interi arbitrario), la soluzione ottiene un punteggio parziale.
- Questa funzione è chiamata esattamente una volta.

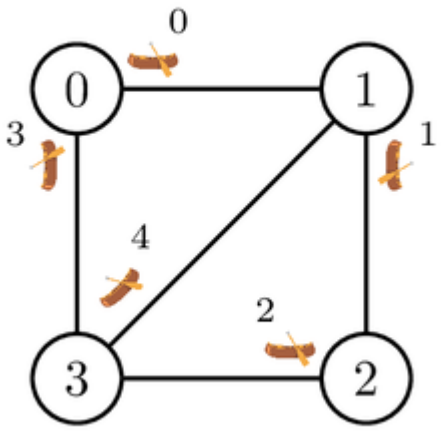
Casi di esempio

Esempio 1

Considera la seguente chiamata:

```
find_journey(4, 5, [0, 1, 2, 0, 3], [1, 2, 3, 3, 1])
```

Questa configurazione di isole e canoe è rappresentata come segue.



Se Miss Dengolecca usa le canoe 0, 1, 2 e 4, in quest'ordine, si ritrova nell'isola 1. Dopodiché, può usare di nuovo la canoa 0 (che adesso è ancorata in 1), poiché non è l'ultima che ha usato, arrivando all'isola 0. Può adesso usare le canoe 3, 2, 1, 4, e ancora 3. Miss Dengolecca è ora tornata in 0 e tutte le canoe sono nelle isole iniziali.

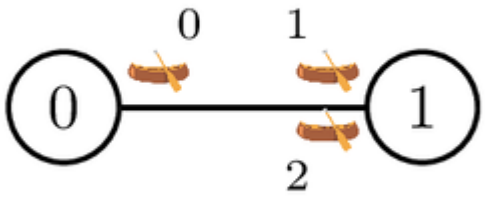
Pertanto, $[0, 1, 2, 4, 0, 3, 2, 1, 4, 3]$ è una soluzione che vale punteggio pieno.

Esempio 2

Considera la seguente chiamata:

```
find_journey(2, 3, [0, 1, 1], [1, 0, 0])
```

Questa configurazione di isole e canoe è rappresentata come segue.



Miss Dengolecca non può che iniziare con la canoa 0, e poi può usare la canoa 1 o la 2 (infatti, non può usare la canoa 0 due volte di seguito). In entrambi i casi, si ritrova nell'isola 0. Ma adesso non tutte le canoe sono ancorate come all'inizio, e Miss Dengolecca non può usare nessuna canoa (l'unica ancorata in 0 è quella che ha appena usato). Pertanto, la funzione deve restituire `false`.

Assunzioni

- $2 \leq N \leq 100\,000$.
- $1 \leq M \leq 200\,000$.
- $0 \leq U[i] < N$ e $0 \leq V[i] < N$ (per ogni $0 \leq i < M$).
- $U[i] \neq V[i]$ (per ogni $0 \leq i < M$).

Subtask

1. (5 punti) $N = 2$.
2. (5 punti) $N \leq 400$. Per ogni coppia di isole distinte x e y, ci sono esattamente due canoe che le connettono. Una di esse è ancorata a x, l'altra a y.
3. (21 punti) $N \leq 1000$, M è pari, e per ogni i **pari** ($0 \leq i < M$) le canoe i e $i+1$ connettono le isole $U[i]$ e $V[i]$. La canoa i è inizialmente ancorata in $U[i]$, la canoa $i+1$ è inizialmente ancorata in $V[i]$. Formalmente, $U[i] = V[i+1]$ e $V[i] = U[i+1]$.
4. (24 punti) $N \leq 1000$, M è pari, e per ogni i **pari** ($0 \leq i < M$) le canoe i e $i+1$ connettono le isole $U[i]$ e $V[i]$. Entrambe le canoe sono inizialmente ancorate in $U[i]$.
5. (45 punti) Nessuna limitazione aggiuntiva.

Per ogni caso di test in cui esiste un viaggio valido, la tua soluzione riceve:

- punteggio pieno se restituisce un viaggio valido con al massimo $2\,000\,000$ spostamenti,
- il 35% dei punti se restituisce `true`, un viaggio non valido o troppo lungo,
- 0 punti in tutti gli altri casi.

Per ogni caso di test in cui non esiste alcun viaggio valido, la tua soluzione:

- riceve punteggio pieno se restituisce `false`,
- riceve 0 punti altrimenti.

Nota che il punteggio finale di un subtask è il minimo dei punteggi dei casi di test di quel subtask.

Grader di esempio

Il grader di esempio legge l'input secondo il seguente formato:

- riga 1: N M
- riga $2 + i$ ($0 \leq i < M$): $U[i]$ $V[i]$

Il grader di esempio stampa l'output secondo il seguente formato:

- Se `find_journey` ritorna un `bool`:
  - riga 1: il numero 0
  - riga 2: 0 se `find_journey` ritorna `false`, 1 altrimenti
- Se `find_journey` ritorna un `int[]`, siano $c[0]$, $c[1]$, $\ldots$, $c[k-1]$ i suoi elementi. Il grader di esempio stampa:
  - riga 1: il numero 1
  - riga 2: k
  - riga 3: $c[0]$ $c[1]$ $\ldots$ $c[k-1]$