Un lungo cammino

Gli organizzatori delle IOI 2023 sono in un gran casino: si sono dimenticati di organizzare la gita a Ópusztaszer di domani! Ma forse non è ancora troppo tardi...

Ci sono N monumenti a Ópusztaszer (numerati da 0 a $N-1$), alcuni dei quali collegati da sentieri *bidirezionali*. Tuttavia, gli organizzatori non sanno quali coppie di monumenti sono collegate da un sentiero!

Sanno però che la **densità** della rete di sentieri è *almeno* D, con $D \in \{1, 2, 3\}$. Cioè, presi comunque 3 monumenti distinti u, v e w, esistono *almeno* D sentieri che li collegano, tra i tre teoricamente possibili: (u, v), (v, w), (w, u).

Gli organizzatori possono inoltre fare richiesta di informazioni sulla rete di sentieri di Ópusztaszer all'ufficio competente. In ogni modulo di richiesta, devono indicare due vettori non vuoti $[A[0], \ldots, A[P-1]]$ e $[B[0], \ldots, B[R-1]]$, di monumenti *tutti e* $P + R$ *distinti tra loro*. L'ufficio risponde alla richiesta con `true` se esiste almeno un sentiero tra un monumento $A[i]$ in A e uno $B[j]$ in B; altrimenti risponde con `false`.

Un **cammino** di lunghezza l è una sequenza di monumenti *distinti* $t[0], t[1] \ldots, t[l-1]$ per cui esiste un sentiero tra $t[i]$ e $t[i+1]$. Aiuta gli organizzatori a trovare un qualsiasi cammino di lunghezza massima, facendo opportune richieste all'ufficio competente.

Note di implementazione

Devi implementare la seguente funzione:

```
int[] longest_trip(int N, int D)
```

- N: il numero di monumenti.
- D: la densità minima garantita nella rete di senti L'ufficio risponde alla richiesta con `true` se esiste almeno un sentiero tra un monumento $A[i]$ in A e uno $B[j]$ in B; altrimenti risponde con `false`.
- Questa funzione deve restituire un array $t = [t[0], t[1], \ldots, t[l-1]]$ che rappresenta uno dei cammini di lunghezza massima.
- Questa funzione può essere chiamata **più volte** in uno stesso testcase.

Il tuo codice può chiamare la seguente funzione definita nel grader:

```
bool are_connected(int[] A, int[] B)
```

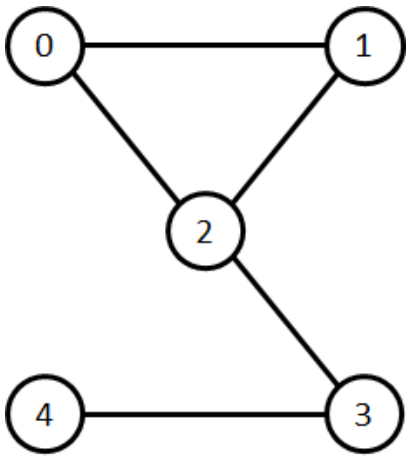
- A: un array non vuoto di monumenti distinti.
- B: un array non vuoto di monumenti distinti.
- A e B non devono avere elementi in comune.
- Questa funzione restituisce `true` se esistono un monumento in A e uno in B collegati da un sentiero; altrimenti restituisce `false`.
- Questa funzione ha complessità pari al prodotto delle lunghezze di A e B.
- Questa funzione può essere chiamata al massimo $32\,640$ volte per ogni chiamata a `longest_trip`, e al massimo $150\,000$ volte in totale in uno stesso testcase.
- La somma delle lunghezze totali degli array A e B passati alla funzione tra tutte le chiamate non può superare $1\,500\,000$, cioè $\sum [\text{len}(A) + \text{len}(B)] \leq 1\,500\,000$.

Il grader **non è adattivo**: i valori di N e D e l'insieme di monumenti collegati da sentieri è fissato prima di ciascuna chiamata a `longest_trip`.

Esempi

Esempio 1

Considera un primo scenario in cui $N = 5$, $D = 1$, e i sentieri sono quelli in figura:



La funzione `longest_trip` viene quindi chiamata così:

```
longest_trip(5, 1)
```

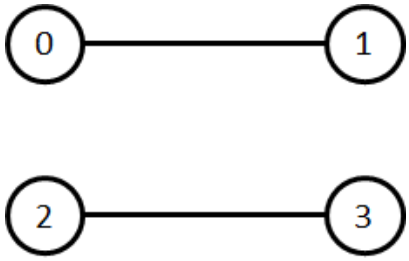
La funzione potrebbe per esempio fare chiamate ad `are_connected` come segue.

Chiamata	Coppie collegate da sentieri	Valore restituito
`are_connected([0], [1, 2, 4, 3])`	$(0, 1)$ e $(0, 2)$	`true`
`are_connected([2], [0])`	$(2, 0)$	`true`
`are_connected([2], [3])`	$(2, 3)$	`true`
`are_connected([1, 0], [4, 3])`	nessuna	`false`

Nella quarta chiamata, *nessuna* delle coppie $(1, 4)$, $(0, 4)$, $(1, 3)$ e $(0, 3)$ sono collegate da sentieri. Dato che la densità della rete è almeno $D = 1$, considerando la tripletta di monumenti $(0, 3, 4)$, si deduce che i monumenti $(3, 4)$ devono essere collegati da un sentiero. Analogamente, anche i monumenti 0 e 1 devono essere collegati.

A questo punto, si può dedurre che $t = [1, 0, 2, 3, 4]$ è un cammino valido di lunghezza 5, e non può esistere un cammino più lungo di 5. Quindi la funzione `longest_trip` può restituire $[1, 0, 2, 3, 4]$.

Considera inoltre un secondo scenario in cui $N = 4$, $D = 1$, e la rete di sentieri è come segue:



La funzione `longest_trip` è quindi chiamata come segue:

```
longest_trip(4, 1)
```

In questo scenario, la massima lunghezza di un cammino è 2. Quindi, dopo alcune chiamate ad `are_connected`, la funzione `longest_trip` deve restituire uno tra $[0, 1]$, $[1, 0]$, $[2, 3]$ o $[3, 2]$.

Esempio 2

Il subtask 0 contiene anche un altro testcase che puoi scaricare tra gli allegati, con un unico scenario con $N = 256$ monumenti.

Assunzioni

- $3 \leq N \leq 256$.
- La somma dei valori di N tra tutte le chiamate a `longest_trip` non supera $1\,024$.
- $1 \leq D \leq 3$.

Subtask

1. (5 punti) $D = 3$.
2. (10 punti) $D = 2$.
3. (25 punti) $D = 1$. Sia $l^\star$ la lunghezza massima di un cammino. La funzione `longest_trip` può restituire un qualunque cammino di lunghezza almeno $\left\lceil \frac{l^\star}{2} \right\rceil$.
4. (60 punti) $D = 1$.

Se, in uno qualunque dei testcase, una chiamata ad `are_connected` non rispetta i limiti descritti nel testo, oppure l'array restituito da `longest_trip` non è valido, il punteggio della tua soluzione per quel subtask sarà 0.

Nel subtask 4 il tuo punteggio sarà calcolato sulla base del numero di chiamate alla funzione `are_connected` per una singola chiamata a `longest_trip`. Sia q il massimo numero di chiamate ad `are_connected` tra tutte le chiamate a `longest_trip` di ogni testcase del subtask 4. Il tuo punteggio per questo subtask sarà calcolato come:

Condizione	Punteggio
$2\,750 < q \leq 32\,640$	20
$550 < q \leq 2\,750$	30
$400 < q \leq 550$	45
$q \leq 400$	60

Grader di esempio

Sia C il numero di scenari (chiamate a `longest_trip`) in un testcase. Il grader di esempio legge l'input nel seguente formato:

- riga 1: C

Deve seguire la descrizione di C scenari, ciascuno nel seguente formato:

- riga 1: N D
- riga $1 + i$ ($1 \leq i < N$): $U_i[0]$ $U_i[1]$ $\ldots$ $U_i[i-1]$

Dove U_i ($1 \leq i < N$) è un array di dimensione i che descrive le coppie di monumenti collegati da un sentiero: per ogni $j < i < N$, i monumenti j e i sono collegati se e solo se $U_i[j]$ vale 1.

In ogni scenario, prima di chiamare `longest_trip`, il grader di esempio controlla che la densità della rete sia effettivamente almeno D, e se non è così stampa il messaggio `Insufficient Density` e termina immediatamente.

Se il grader di esempio riscontra una violazione del protocollo delle chiamate ad `are_connected`, stampa `Protocol Violation: <MSG>`, dove `<MSG>` è uno dei seguenti messaggi di errore:

- `invalid array`: in una chiamata ad `are_connected`, uno degli array A o B è vuoto, oppure contiene monumenti ripetuti.
- `non-disjoint arrays`: in una chiamata ad `are_connected`, gli array A e B non sono disgiunti.
- `too many calls`: il numero di chiamate ad `are_connected` supera $32\,640$ durante una singola chiamata a `longest trip`, o supera $150\,000$ in totale.
- `too many elements`: il numero totale di monumenti passati ad `are_connected` tra tutte le sue chiamate supera $1\,500\,000$.

Se il grader non riscontra violazioni, sia $t[0], t[1], \ldots, t[l-1]$ il cammino restituito da `longest_trip` per uno scenario. Il grader stampa quindi il risultato di quello scenario nel seguente formato:

- riga 1: l
- riga 2: $t[0]\ t[1]\ \ldots\ t[l-1]$
- riga 3: il numero di chiamate ad `are_connected` in questo scenario

Infine, il grader stampa:

- riga $1 + 3 \cdot C$: il massimo numero di chiamate ad `are_connected` tra tutte le chiamate a `longest_trip`