Pazza gioia (caduta)

Come da tradizione OII, il giorno di gara precede la lunga notte di festeggiamenti. Quest'anno verrà organizzato un party sfrenato in perfetto stile trentino, durante il quale i ragazzi e le ragazze si daranno alla pazza gioia. Ovviamente, per "pazza gioia" intendiamo cimentarsi in un'attività intellettualmente stimolante ed allo stesso tempo non troppo faticosa!

Quest'anno, la scelta dell'attività è ricaduta su un passatempo piuttosto sofisticato: costruire una lunghissima catena di domino.

La festa si protrae per tutta la notte e viene bruscamente interrotta quando Monica arriva a richiamare gli atleti per portarli al liceo Galilei, dove si terrà la cerimonia di premiazione.

Finora gli atleti sono riusciti a posizionare in tutto N tessere, ognuna ad esattamente un centimentro di distanza dalla precedente. Non avendo a disposizione abbastanza tessere tradizionali, hanno usato un po' di tutto: ciascuna "tessera" potrebbe perciò essere alta pochi centimetri ma anche svariati metri!

Figura 1: Atleti all'opera

Per non vanificare gli sforzi fatti, gli atleti vogliono ora concludere la festa in grande stile spingendo la prima tessera così da far cadere l'intera catena. La caduta di una tessera alta h fa cadere le successive $h-1$ tessere: quindi le tessere di altezza 1 non fanno cadere alcuna tessera.

Nella stanchezza della nottata, gli atleti temono di aver commesso qualche errore nel posizionare le tessere. Non c'è però tempo di rifare tutto: aiutali a capire se è possibile scambiare due tessere qualsiasi (ma solo due) in modo che l'intera catena venga abbattuta!

Chiarimenti

Le posizioni in cui si trovano le tessere sono numerate da 0 a $N-1$. Le posizione 0 è quella più a sinistra e le tessere cadono da sinistra verso destra.

Dopo aver effettuato l'eventuale scambio, la tessera in posizione 0 viene fatta cadere per prima. Se la tessera in posizione i cade, ed è alta h, allora cadranno anche tutte le tessere che si trovano nelle posizioni fino a $i+h-1$, inclusa.

Quindi, la tessera in posizione 0 cade sempre, e una tessera in posizione $i \geq 1$ cade se e solo se esiste una tessera in posizione $i' < i$, che cade, alta almeno $i - i' + 1$.

Nota che tutte le tessere cadono se e solo se cade la tessera in posizione $N-1$.

Implementazione

Dovrai sottoporre un unico file, con estensione `.cpp` o `.c`.

> ☞ Tra gli allegati a questo task troverai un template `catena.cpp` e `catena.c` con un esempio di implementazione.

Dovrai implementare la seguente funzione.

C/C++	`stato_t correggi(int N, int altezze[], coppia_t* scambio);`

- L'intero N rappresenta il numero di tessere.
- L'array `altezze`, indicizzato da 0 a $N-1$, contiene l'altezza di ciascuna tessera. La tessera inizialmente in posizione i, con $0 \leq i \leq N-1$, è alta `altezze`$[i]$.
- Il tipo di dato `stato_t` è una `enum` che può assumere i valori `OK`, `RISOLTO` o `IMPOSSIBILE`. La funzione deve restituire:
  - `OK`, se non è necessario alcuno scambio affinché cadano tutte le tessere; altrimenti,
  - `RISOLTO`, se è possibile scambiare due tessere, in modo che dopo lo scambio tutte le tessere cadano; altrimenti,
  - `IMPOSSIBILE`, quando un singolo scambio non è sufficiente.
- Il tipo di dato `coppia_t` è una `struct` che contiene i campi `domino1` e `domino2`. Qualora la funzione restituisca `RISOLTO`, i campi `domino1` e `domino2` del parametro di output `scambio` devono essere riempiti con le posizioni delle due tessere da scambiare.

La funzione `correggi` sarà chiamata una sola volta. Sarà registrato il valore di ritorno della funzione e, se questo valore è `RISOLTO`, anche le posizioni memorizzate nella struttura `scambio`.

Grader di prova

Nella directory relativa a questo problema è presente una versione semplificata del grader usato durante la correzione, che potete usare per testare le vostre soluzioni in locale. Il grader di esempio legge i dati da `stdin`, chiama le funzioni che dovete implementare e scrive su `stdout`, secondo il seguente formato.

Il file di input è composto da 2 righe, contenenti:

- Riga 1: il numero intero N;
- Riga 2: i numeri interi `altezze`$[i]$, per $i = 0 \dots N-1$, separati da spazi.

Il file di output è composto da una sola riga che contiene una tra le seguenti possibilità:

- la stringa `OK`, se la catena di domino viene completamente abbattuta senza effettuare alcuno scambio;
- la stringa `IMPOSSIBILE`, se non è possibile aggiustare la catena con un solo scambio;
- due numeri interi i e j, separati da spazio, se è possibile aggiustare la catena scambiando le tessere nelle posizioni i e j.

Assunzioni

- $1 \leq N \leq 5\,000\,000$.
- $1 \leq \texttt{altezze}[i] \leq 1000$ per ogni $i = 0, \ldots, N-1$.

Assegnazione del punteggio

Il tuo programma verrà testato su diversi test case raggruppati in subtask. Per ottenere il punteggio relativo ad un subtask, è necessario risolvere correttamente tutti i test che lo compongono.

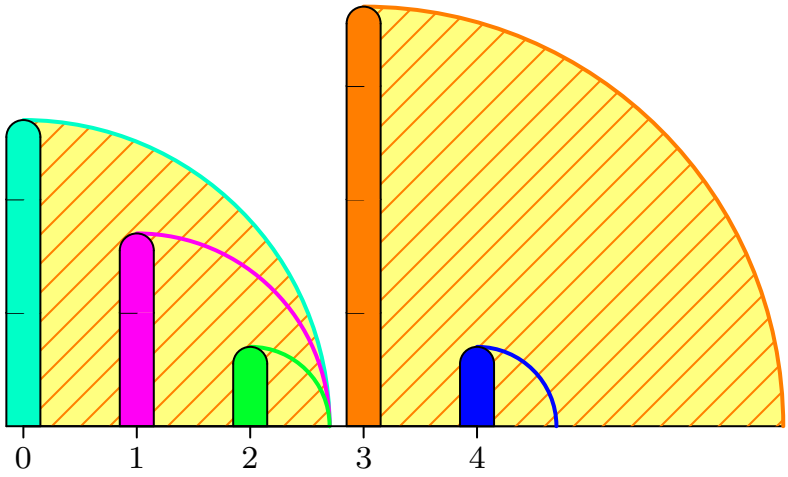
- **Subtask 1** [**0 punti**]: Casi d'esempio.
- **Subtask 2** [**4 punti**]: Le altezze sono tutte uguali.
- **Subtask 3** [**7 punti**]: $N \leq 5000$ e la risposta è `OK` o `IMPOSSIBILE`.
- **Subtask 4** [**9 punti**]: La risposta è `OK` o `IMPOSSIBILE`.
- **Subtask 5** [**13 punti**]: $N \leq 50$.
- **Subtask 6** [**19 punti**]: $N \leq 1000$.
- **Subtask 7** [**28 punti**]: $N \leq 3000$.
- **Subtask 8** [**11 punti**]: $N \leq 100\,000$.
- **Subtask 9** [**9 punti**]: Nessuna limitazione specifica.

Esempi di input/output

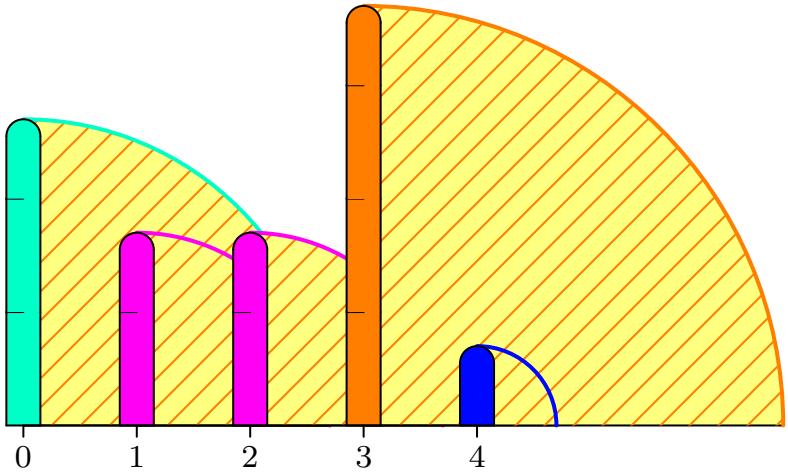
stdin	stdout
5 3 2 1 4 1	2 3
5 3 2 2 4 1	OK
5 1 1 1 1 1	IMPOSSIBILE

Spiegazione

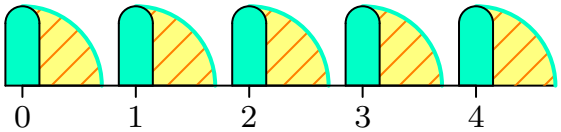
Nel **primo caso di esempio** è sufficiente scambiare le tessere nelle posizioni 2 e 3:



Nel **secondo caso di esempio** la tessera in posizione 2 è alta abbastanza: la catena va bene così com'è.



Nel **terzo caso di esempio** le tessere sono troppo basse: la catena non può cadere.



Ordine pubblico (corteo)

Il 15 settembre, in occasione delle Olimpiadi di Informatica, due cortei di programmatori sfileranno per le vie e le piazze di Trento. Il primo corteo, quello dei sostenitori degli *Spazi*, ed il secondo, quello dei sostenitori dei *Tab*.

La città ha N piazze, collegate da M strade bidirezionali. I due cortei partono rispettivamente dalle piazze P_1 e P_2 e devono giungere rispettivamente alle piazze D_1 e D_2. Ad ogni rintocco della campana di Torre Vanga, la torre cittadina di Trento, uno dei due cortei si sposta da una piazza ad una vicina, percorrendo una strada da un capo all'altro. Nel frattempo, l'altro corteo staziona nella sua piazza attuale. La manifestazione festosa ha termine quando entrambi i cortei sono giunti nelle rispettive piazze di destinazione.

Tutti sanno che non è una buona idea mescolare *Spazi* e *Tab*: per questo motivo gli organizzatori vogliono evitare che i due cortei si avvicinino troppo. Aiuta gli organizzatori nel loro compito! Scrivi un programma che calcoli quale massima distanza è possibile mantenere fra i due cortei durante tutta la manifestazione, consentendo a ciascun corteo di giungere alla propria destinazione.

Chiarimenti

In ogni spostamento, uno solo dei due cortei transita da una piazza a un'altra. Uno spostamento indica quale dei due cortei si muove e verso quale piazza, ed è valido solo se c'è una strada che collega *direttamente* la piazza dove attualmente si trova il corteo da muovere e la piazza in cui portarlo.

Gli spostamenti avvengono in modo regolare, uno ad ogni rintocco, ma **non** è richiesto che gli spostamenti dei due cortei si alternino: è possibile che lo stesso corteo si muova per due o più spostamenti consecutivi.

Un corteo *può* percorrere una stessa strada più di una volta, come anche rivisitare una piazza più di una volta, se lo desidera.

Dopo l'ultimo spostamento, ciascun corteo deve trovarsi a stazionare nella propria piazza di destinazione: la manifestazione può terminare solo in questo modo. Le piazze di partenza e destinazione di un corteo possono però coincidere fra loro.

La sequenza di spostamenti, effettuati uno dopo l'altro durante la manifestazione, è chiamata *pianificazione* e gli organizzatori possono scegliere la pianificazione che desiderano.

Due piazze sono a *distanza* d se è necessario attraversare almeno d strade per spostarsi da una all'altra. Il *margine* associato a una pianificazione è definito come la minima distanza d fra due piazze **simultaneamente** occupate dai due cortei. Ti viene chiesto di individuare qual è il massimo margine che gli organizzatori possono ottenere, scegliendo opportunamente fra tutte le pianificazioni possibili.

Se il tuo programma non calcola correttamente il margine massimo, potrai comunque ottenere un **punteggio parziale** individuando una pianificazione che ottenga un margine alto anche se non ottimo (vedi sezione di assegnazione del punteggio).

Implementazione

Dovrai sottoporre un unico file, con estensione `.cpp` o `.c`.

> ☞ Tra gli allegati a questo task troverai un template `corteo.cpp` e `corteo.c` con un esempio di implementazione.

Dovrai implementare la seguente funzione:

C/C++	`int pianifica(int N, int M, int P1, int D1, int P2, int D2, int A[], int B[]);`

La funzione `pianifica` viene chiamata una sola volta con i seguenti parametri.

- L'intero N rappresenta il numero di piazze della città (le piazze sono numerate da 0 a $N-1$),
- L'intero M rappresenta il numero di strade della città.
- L'intero P_1 rappresenta la piazza di partenza del corteo degli *Spazi* e D_1 la piazza di arrivo, l'intero P_2 rappresenta la piazza di partenza del corteo dei *Tab* e D_2 la piazza di arrivo (P_1, D_1, P_2 e D_2 sono numeri compresi tra 0 e $N-1$).
- Gli array `A` e `B`, indicizzati da 0 a $M-1$, rappresentano le strade nella città. La strada numero i, per $0 \leq i \leq M-1$, collega le piazze `A[`i`]` e `B[`i`]`. Ricordiamo che le strade sono bidirezionali: utilizzando la strada i, è possibile muoversi sia dalla piazza `A[`i`]` alla piazza `B[`i`]`, sia dalla piazza `B[`i`]` alla piazza `A[`i`]`.
- La funzione deve restituire un numero intero che rappresenta il margine massimo possibile in una pianificazione.

Se desideri esibire la tua pianificazione, con lo scopo di ottenere quantomeno un punteggio parziale nel caso in cui il margine calcolato non sia ottimo, puoi comunicare uno ad uno gli spostamenti nella sequenza tramite la seguente funzione:

C/C++	`void sposta(int chi, int dove);`

- L'intero `chi` può essere 1 o 2, e indica indica quale corteo si vuole spostare.
- L'intero `dove` indica il numero della piazza verso cui tale corteo deve muoversi.

La funzione `pianifica` sarà chiamata una sola volta all'inizio dell'esecuzione. All'interno della tua implementazione della funzione `pianifica` potrai chiamare la funzione `sposta` un qualunque numero di volte. Verranno registrate tutte le chiamate a `sposta` e il valore di ritorno di `pianifica`.

Grader di prova

Nella directory relativa a questo problema è presente una versione semplificata del grader usato durante la correzione, che potete usare per testare le vostre soluzioni in locale. Il grader di esempio legge i dati da `stdin`, chiama le funzioni che dovete implementare e scrive su `stdout`, secondo il seguente formato.

Il file di input è composto da $M+2$ righe, contenenti i seguenti numeri interi separati da spazi:

- Riga 1: gli interi N e M.
- Riga 2: gli interi P_1, D_1, P_2 e D_2.
- Riga $3+i$ per $i = 0, \ldots, M-1$: gli interi `A[`i`]` e `B[`i`]`.

Il file di output è composto da $K+1$ righe, dove K rappresenta il numero di volte in cui la funzione `sposta` è stata chiamata, contenenti i seguenti numeri interi separati da spazi:

- Righe $1, \ldots, K$: i parametri con cui è stata chiamata `sposta`.
- Riga $K+1$: il valore di ritorno di `pianifica`.

Assunzioni

- $2 \leq N \leq 1000$.
- $1 \leq M \leq 5000$.
- $0 \leq P_1, D_1, P_2, D_2 \leq N-1$.
- $0 \leq \texttt{A}[i], \texttt{B}[i] \leq N-1$ per ogni $i = 0, \ldots, M-1$.
- Non ci sono strade che collegano una piazza a se stessa e non ci sono strade che collegano la stessa coppia di piazze.
- Da ogni piazza è possibile raggiungere ogni altra piazza.

Assegnazione del punteggio

Il tuo programma verrà testato su diversi casi di test, raggruppati in subtask. Ad un caso di test viene assegnato:

- `1` se la funzione `pianifica` ritorna il valore ottimo di margine per quel caso, oppure se esibisce una pianificazione corretta che di fatto realizza il margine ottimo; altrimenti,
- `0` se non esibisce una pianificazione corretta; altrimenti,
- il punteggio dato dalla formula:

$$\frac{1}{2} \cdot \frac{\text{margine della pianificazione restituita}}{\text{margine delle pianificazione ottima}}$$

  e quindi un punteggio tra `0` e `0.5`.

Il punteggio assegnato a un subtask è il *peggiore* dei punteggi ottenuti sui suoi casi di test, moltiplicato per il valore del subtask. Il punteggio assegnato a questo problema è quindi dato dalla somma dei punteggi sui vari subtask.

- **Subtask 1** [**0 punti**]: Casi d'esempio.
- **Subtask 2** [**9 punti**]: I due cortei sono già a destinazione.
- **Subtask 3** [**10 punti**]: Il grafo è un ciclo.
- **Subtask 4** [**13 punti**]: $N, M \leq 10$.
- **Subtask 5** [**17 punti**]: Un corteo non necessita di spostarsi (in una pianificazione ottima).
- **Subtask 6** [**22 punti**]: $N, M \leq 100$.
- **Subtask 7** [**29 punti**]: Nessuna limitazione specifica.

Esempi di input/output

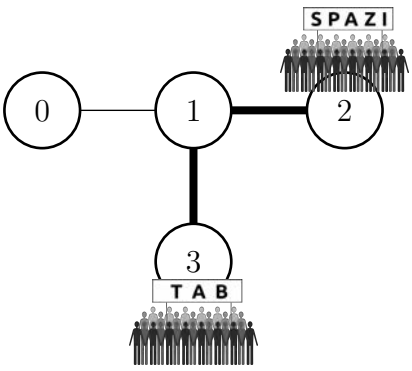
stdin	stdout
10 11 2 5 3 7 0 1 1 2 2 6 4 6 4 5 3 5 3 0 6 7 5 8 7 9 9 8	3
4 3 2 1 3 2 0 1 1 2 1 3	1 1 1 0 2 1 2 2 1 1 99999

Spiegazione

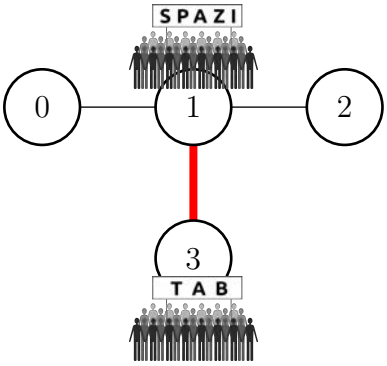
Nel **primo caso di esempio** la soluzione calcola il margine ottimo 3. Infatti, i due cortei riescono sempre a mantenersi a distanza almeno 3, ma non è possibile fare di meglio. Uno delle pianificazioni possibili è la seguente:

- all'inizio la distanza è 3;
- *Tab* si sposta da 3 a 5 (distanza 3), poi da 5 a 8 (distanza 4), poi da 8 a 9 (distanza 3);
- *Spazi* si sposta da 2 a 1 (distanza 4), poi da 1 a 0 (distanza 4), poi da 0 a 3 (distanza 3);
- *Tab* si sposta da 9 a 7 (distanza 4);
- *Spazi* si sposta da 3 a 5 (distanza 3).

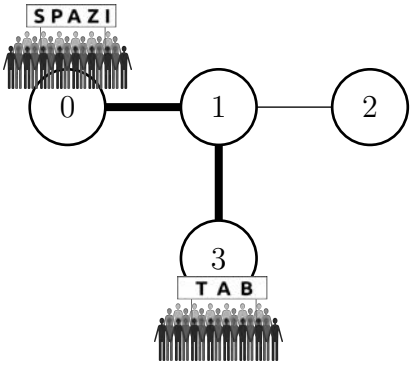
Nel **secondo caso di esempio** la soluzione calcola esplicitamente una pianificazione che realizza il margine ottimo 1. La pianificazione è mostrata nelle figure seguenti. Non è possibile ottenere un margine migliore di 1.



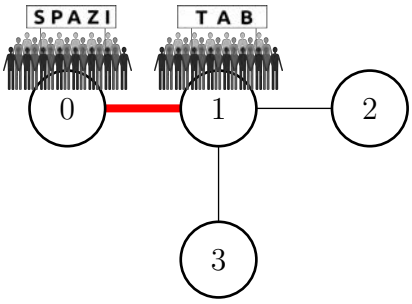
Situazione iniziale



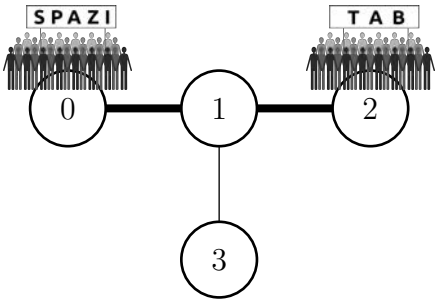
`sposta(1, 1)`



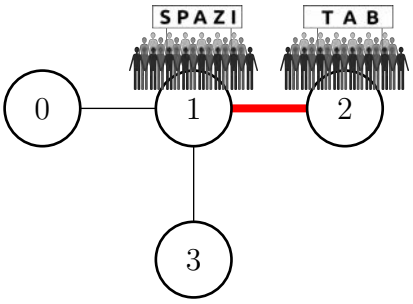
`sposta(1, 0)`



`sposta(2, 1)`



`sposta(2, 2)`



`sposta(1, 1)`

Nota che entrambe le soluzioni di esempio ottengono il punteggio massimo: la prima in quanto calcola il margine ottimo, anche se non esibisce una pianificazione, la seconda in quanto esibisce una pianificazione ottimale, anche se restituisce un valore errato come margine ottimo.

Stanza degli specchi (`specchi`)

Mojito intende prepararsi per la sua visita al MUSE, il museo delle scienze di Trento, dove lo attende una sfida con Monica nella *stanza degli specchi*.

Il pavimento della stanza è rettangolare, ed è ricoperto con piastrelle quadrate disposte su una griglia di R righe e C colonne. Ogni piastrella presenta due scanalature diagonali a "X" che consentono l'eventuale inserimento di uno specchio posizionato in piedi lungo una delle due diagonali della piastrella. Gli specchi sono dei pannelli rettangolari senza spessore, con entrambe le facce perfettamente riflettenti.

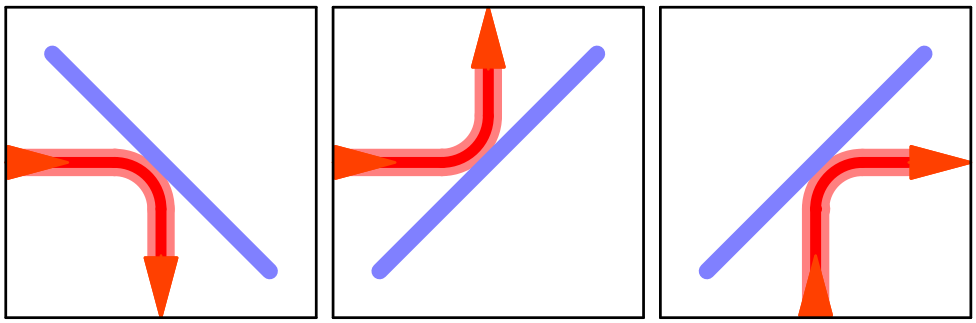


Figura 1: Alcuni dei possibili modi in cui la luce si può riflettere

Su ciascuna delle quattro pareti della stanza, in corrispondenza di ogni riga o colonna, è presente un foro attraverso il quale Monica può immettere un raggio laser. Una volta entrato nella stanza, il raggio *rimbalza* sugli specchi che incontra lungo il proprio percorso (possibilmente nessuno), e infine fuoriesce da uno degli altri fori. È lì che Mojito vorrebbe farsi trovare pronto ad accogliere ed acchiappare il raggio uscente.

All'inizio la stanza è vuota, ossia tutte le piastrelle sono prive di specchi. Nel corso del gioco, Monica può decidere di aggiungere uno specchio su una delle piastrelle ancora libere, posizionandolo su una delle due diagonali a sua scelta, oppure può immettere il raggio laser attraverso un foro. Ogni volta che Mojito vede Monica avvicinarsi ad uno dei fori, cerca di anticipare da quale foro fuoriuscirà il laser.

Aiuta Mojito scrivendo un programma che calcoli il foro di uscita del raggio, tenendo conto degli specchi man mano aggiunti da Monica!

Specifiche

Le griglia di piastrelle è formata da R righe, numerate da sinistra verso destra con i numeri da 0 a $R-1$, e C colonne numerate dall'alto verso il basso con i numeri da 0 a $C-1$. Le quattro pareti della stanza sono chiamate `SOPRA`, `DESTRA`, `SOTTO`, `SINISTRA`. Vi sono R fori sulle pareti `DESTRA` e `SINISTRA`, e C fori sulle pareti `SOPRA` e `SOTTO`, per un totale di $2R+2C$ fori. I fori su ciascuna parete sono identificati dal numero della riga o colonna su cui si affacciano.

Non possono *mai* trovarsi due specchi sulla stessa piastrella, nemmeno su diagonali diverse. Inoltre, gli specchi sono posizionati in modo molto preciso lungo le diagonali, perciò il raggio laser rimane parallelo alle pareti della stanza, e procede sempre esattamente lungo una delle righe o delle colonne. Lo spessore degli specchi è trascurabile, anche dopo tutti i rimbalzi la linea del raggio non fuoriesce dall'area dei fori.

Implementazione

Dovrai sottoporre un unico file, con estensione `.cpp` o `.c`.

> ☞ Tra gli allegati a questo task troverai un template `specchi.cpp` e `specchi.c` con un esempio di implementazione.

Dovrai implementare le seguenti funzioni:

C/C++	`void inizia(int R, int C);`

La funzione `inizia` viene invocata all'inizio con i seguenti parametri:

- l'intero R che rappresenta il numero di righe della stanza,
- l'intero C che rappresenta il numero di colonne della stanza.

C/C++	`void aggiungi(int r, int c, char diagonale);`

La function `aggiungi` viene chiamata ogni volta che Monica aggiunge uno specchio, con i seguenti parametri:

- un numero intero `r`, compreso fra 0 a $R-1$, che indica il numero della riga in cui viene aggiunto lo specchio,
- un numero intero `c`, compreso fra 0 a $C-1$, che indica il numero della colonna in cui viene aggiunto lo specchio,
- un carattere `diagonale` che indica l'orientamento dello specchio e che può assumere solo i valori '\' o '/'.

C/C++	`foro_t calcola(foro_t ingresso);`

La funzione `calcola` viene chiamata ogni volta che Monica introduce il raggio laser in un foro.

- Il parametro `ingresso`, di tipo `foro_t`, indica il foro di ingresso del raggio.
- La funzione deve restituire il foro di uscita del raggio, in una `struct` di tipo `foro_t`.

Il tipo di dato `foro_t` è una `struct` che specifica un foro su una parete della stanza, e contiene i seguenti campi:

- `parete`: un `enum` che può assumere i valori `SOPRA`, `DESTRA`, `SOTTO` o `SINISTRA` e che individua la parete su cui si trova il foro,
- `posizione`: un intero compreso tra 0 e $R-1$ (se `parete` è `DESTRA` o `SINISTRA`) o tra 0 e $C-1$ (se `parete` è `SOPRA` o `SOTTO`), che indica la riga o colonna su cui si affaccia il foro.

La funzione `inizia` sarà chiamata per prima, una sola volta. Successivamente verranno chiamate le funzioni `aggiungi` o `calcola`, in nessun ordine specifico. Le funzioni `aggiungi` o `calcola` vengono chiamate per un totale di Q volte. Il numero Q non è specificato in anticipo al programma, ma rispetta le assunzioni indicate nelle sezioni successive.

Grader di prova

Nella directory relativa a questo problema è presente una versione semplificata del grader usato durante la correzione, che potete usare per testare le vostre soluzioni in locale. Il grader di esempio legge i dati da `stdin`, chiama le funzioni che dovete implementare e scrive su `stdout`, secondo il seguente formato.

Il file di input è composto $Q + 1$ righe, contenenti:

- Riga 1: gli interi R, C e Q.
- Righe $2, \ldots, Q + 1$: un carattere c seguito da due numeri interi a e b.
  - Se il carattere c è `?` viene chiamata la funzione `calcola`, dove il primo numero a indica la parete (da 0 a 3 in senso orario: `SOPRA`, `DESTRA`, `SOTTO`, `SINISTRA`) e il secondo numero b indica la posizione del foro.
  - Se il carattere c è `'\'` o `'/'`, viene chiamata la funzione `aggiungi` dove il numero a indica la riga e il numero b la colonna.

Il file di output contiene una riga per ogni chiamata alla funzione `calcola`, contenente il valore di ritorno di tale chiamata.

Assunzioni

- $1 \leq R, C \leq 100\,000$.
- $1 \leq Q \leq 250\,000$
- Non viene mai inserito più di uno specchio nella stessa posizione.

Assegnazione del punteggio

Il tuo programma verrà testato su diversi test case raggruppati in subtask. Per ottenere il punteggio relativo ad un subtask, è necessario risolvere correttamente tutti i test che lo compongono.

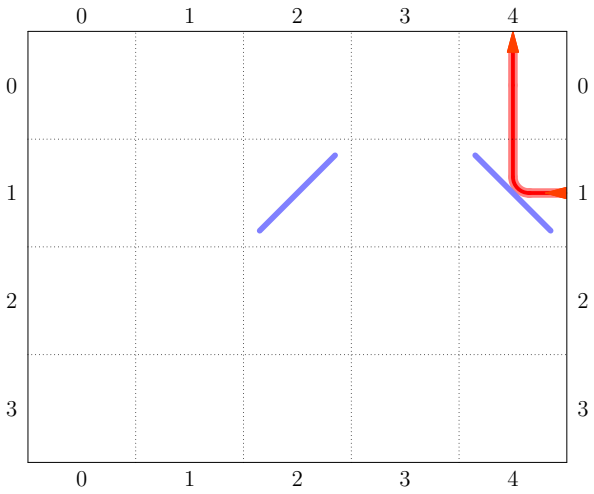
- **Subtask 1 [0 punti]**: Casi d'esempio.
- **Subtask 2 [8 punti]**: $R, C \leq 10$ e $Q \leq 100$.
- **Subtask 3 [21 punti]**: $R, C \leq 1000$ e $Q \leq 5000$.
- **Subtask 4 [14 punti]**: $R, C \leq 1000$, $Q \leq 100\,000$ e tutte le chiamate a `aggiungi` precedono tutte le chiamate a `calcola`.
- **Subtask 5 [24 punti]**: $R, C \leq 100\,000$ e $Q \leq 5000$.
- **Subtask 6 [13 punti]**: $R, C \leq 100\,000$, $Q \leq 250\,000$ e tutte le chiamate a `aggiungi` precedono tutte le chiamate a `calcola`.
- **Subtask 7 [12 punti]**: $R, C \leq 100\,000$ e $Q \leq 100\,000$.
- **Subtask 8 [8 punti]**: Nessuna limitazione specifica.

Esempi di input/output

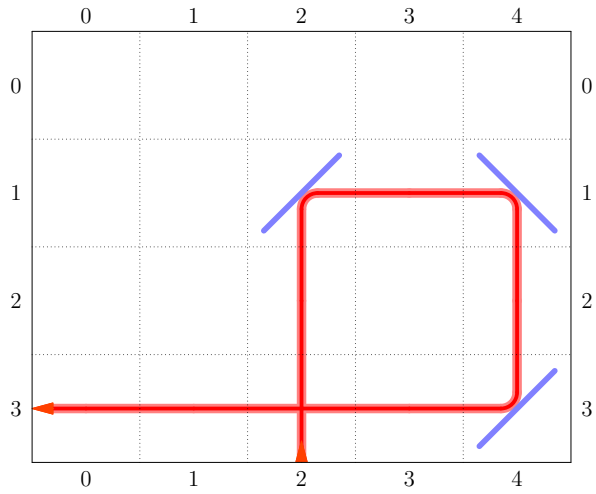
stdin	stdout
4 5 5 / 1 2 \ 1 4 ? 1 1 / 3 4 ? 2 2	0 4 3 3
6 2 7 ? 1 1 / 1 1 \ 4 0 / 4 1 / 1 0 ? 0 1 ? 1 1	3 1 1 1 0 1

Spiegazione

Le richieste del **primo caso d'esempio** possono essere così rappresentate:

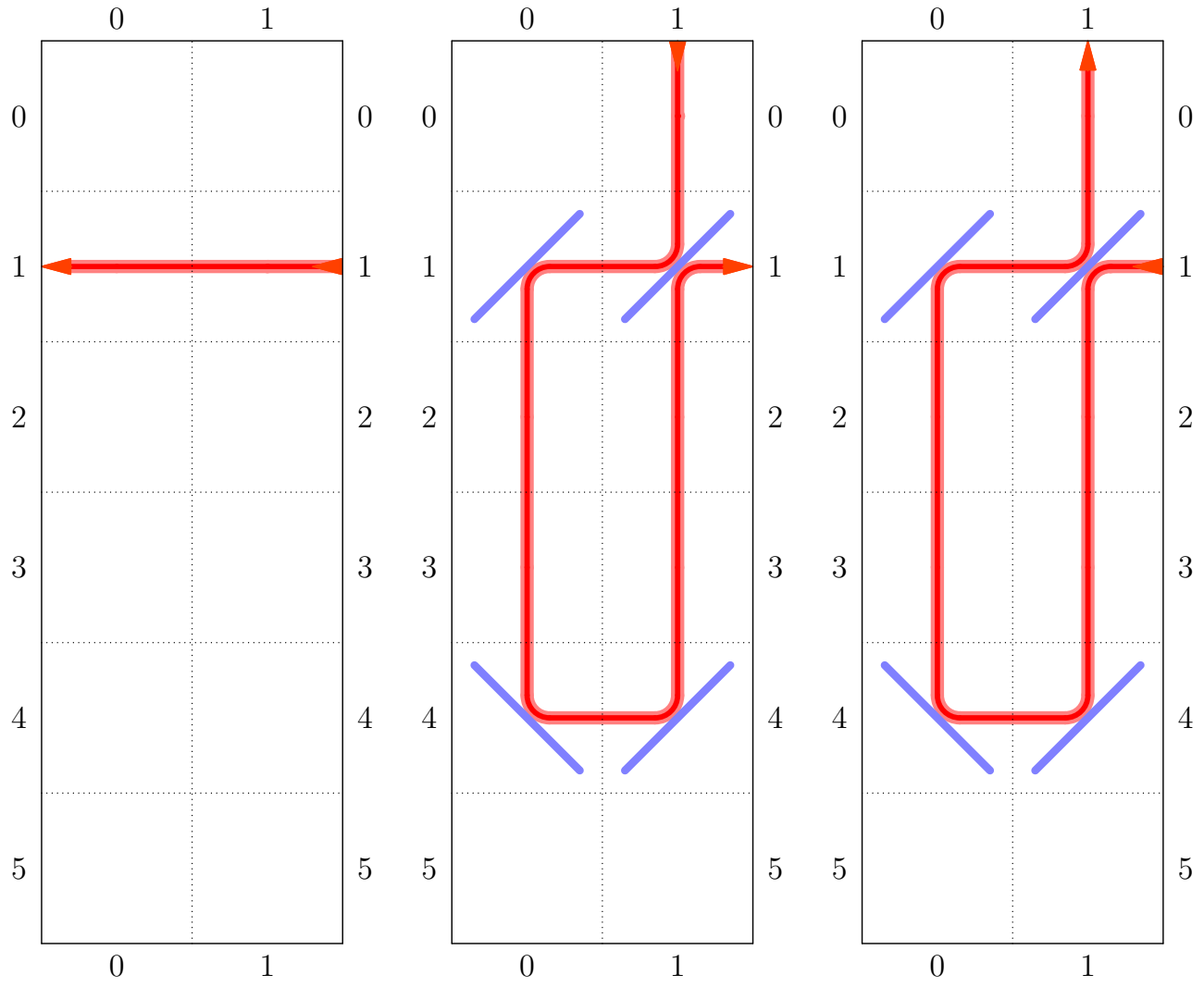


`calcola({ DESTRA, 1 }) ⇒ { SOPRA, 4 }`



`calcola({ SOTTO, 2 }) ⇒ { SINISTRA, 3 }`

Il **secondo caso d'esempio** è il seguente:



`calcola({ DESTRA, 1 })`
`{ SINISTRA, 1 }`

`calcola({ SOPRA, 1 })`
`{ DESTRA, 1 }`

`calcola({ DESTRA, 1 })`
`{ SOPRA, 1 }`