Skyline II (`skyline2`)

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

All'incirca un anno fa la città di Yendys aveva indetto un concorso, con l'obiettivo di rinnovare il proprio skyline. A causa degli innumerevoli progetti presentati (nonchè del complesso criterio di valutazione), la giuria si è pronunciata soltanto oggi, rivelando il progetto vincitore. La direzione dei lavori di rinnovamento della città è stata affidata all'illustre architetto Ilazimi.

Lo *skyline* di Yendys consisteva di N altissimi grattacieli, i quali sono stati appena demoliti e devono essere ricostruiti secondo il progetto vincitore del concorso; tale progetto contiene $H[0], H[1], \ldots, H[N-1]$, ossia il numero di piani che ciascun grattacielo dovrà avere. All'inizio di ogni settimana giunge in città un carico di materiali da costruzione, grazie ai quali è possibile edificare M nuovi piani, **appartenenti tutti a grattacieli diversi**. A causa delle difficili condizioni economiche che la città è costretta ad affrontare, a Ilazimi è stato imposto di non sprecare il materiale da costruzione: per ogni carico che arriva, è necessario edificare tutti gli M piani, ciascuno su un diverso grattacielo.

Per Ilazimi, dunque, un progetto è *accettabile* soltanto se è realizzabile utilizzando il materiale da costruzione proveniente da un numero intero di carichi, senza sprecarne nemmeno un po'. D'altro canto, si sa, i grandi artisti cambiano idea assai frequentemente; a Ilazimi, per esempio, capita spesso di non essere soddisfatto del proprio progetto, e di modificarlo alterando l'altezza di un grattacielo. Dopodiché, si chiede se il progetto modificato risulta ancora accettabile; aiutalo a scoprirlo!

Implementazione

Dovrai sottoporre esattamente un file con estensione `.c` o `.cpp`.

> ☞ Tra gli allegati a questo task troverai un template (`skyline2.c`, `skyline2.cpp`) con un esempio di implementazione.

Il tuo programma dovrà implementare le seguenti funzioni.

```
void Inizializza(int N, int M, int* H);
```

Questa funzione verrà chiamata una volta all'inizio di ogni testcase.

- N è il numero di grattacieli.
- M è il numero di piani edificabili con un carico di materiali.
- $H[i]$ è l'altezza dello i-esimo grattacielo (con $0 \leq i < N$).

```
int Cambia(int P, int V);
```

Questa funzione verrà chiamata ogni volta che Ilazimi cambia idea.

- P indica l'indice del grattacielo che deve essere modificato.
- V indica il nuovo numero di piani di quel grattacielo.

Questa funzione deve restituire 1 se il piano di costruzione è ancora accettabile, 0 altrimenti.

> ☞ Ogni volta che Ilazimi cambia idea, la modifica al progetto rimane valida anche per le successive chiamate a `Cambia`. In altre parole, le modifiche effettuate sono permanenti.

Il numero di chiamate fatte a `Cambia` sarà esattamente uguale a Q.

Assunzioni

- $0 < N \leq 50\,000$.
- $0 < M \leq N$.
- $0 \leq H[0], \ldots, H[N-1] \leq 1\,000\,000\,000$.
- $0 \leq P < N$ e $0 \leq V \leq 1\,000\,000\,000$ per ogni chiamata a `Cambia`.
- $0 < Q \leq 200\,000$.
- Il progetto iniziale (quello descritto da $H[0], \ldots H[N-1]$) è accettabile.

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 relativi ad esso.

- **Subtask 1 [0 punti]:** Caso d'esempio.
- **Subtask 2 [11 punti]:** $H[0], \ldots, H[N-1] \leq 1$; $V \leq 1$.
- **Subtask 3 [19 punti]:** $N, Q \leq 10$; $H[0], \ldots, H[N-1] \leq 100$; $V \leq 100$.
- **Subtask 4 [28 punti]:** $N, Q \leq 2000$.
- **Subtask 5 [42 punti]:** Nessuna limitazione specifica.

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 di input dal file `input.txt`, a quel punto chiama la funzione `Inizializza` una sola volta, poi la funzione `Cambia` Q volte, e infine scrive i risultati restituiti dalle chiamate a `Cambia` sul file `output.txt`.

Nel caso vogliate generare un input, il file `input.txt` deve avere questo formato:

- Riga 1: contiene N, M, Q separati da uno spazio.
- Riga 2: contiene i numeri $H[0], \ldots, H[N-1]$ separati da uno spazio.
- Riga $3+i$: contiene P_i, V_i, i parametri con cui viene chiamata la funzione `Cambia`, separati da uno spazio (con $0 \leq i < Q$).

Esempi di input/output

stdin	stdout
5 2 3 2 1 2 0 1 1 0 0 5 3 2	0 0 1

Spiegazione

In questo caso d'esempio $N = 5, M = 2$ e Ilazimi cambia idea 3 volte. Nel progetto iniziale, i grattacieli hanno altezze $[2, 1, 2, 0, 1]$. Dopo la prima chiamata a `Cambia` le altezze divengono $[2, 0, 2, 0, 1]$, dopo la seconda divengono $[5, 0, 2, 0, 1]$, dopo la terza divengono $[5, 0, 2, 2, 1]$.

Note

In C/C++, il tipo `int` può contenere interi fino a $2^{31} - 1$, mentre il tipo `long long int` può contenere interi fino a $2^{63} - 1$.

Soluzione

Consideriamo un progetto in cui le altezze dei grattacieli sono $H_0, H_1 \dots H_{N-1}$. Indichiamo con S la somma $H_0 + H_1 + \cdots + H_{N-1}$. Supponendo che il progetto sia accettabile, possiamo dedurre che:

(1) S è divisibile per M.

Infatti, supponendo che i lavori vengano portati a termine in k settimane, e dovendo costruire esattamente M piani ogni settimana, si deve avere $S = k \cdot M$. Da questa osservazione deduciamo anche che i lavori si protraggono per esattamente $\frac{S}{M}$ settimane.

(2) Ogni grattacielo previsto dal progetto ha al massimo $\frac{S}{M}$ piani.

Infatti, supponiamo per assurdo che ci sia un grattacielo con un numero maggiore di piani. Gli operai possono costruire al massimo un piano di tale grattacielo ogni settimana. Pertanto, al termine delle $\frac{S}{M}$ settimane di lavoro, potranno essere stati costruiti al massimo $\frac{S}{M}$ piani del grattacielo, che non sono sufficienti per portarlo a termine.

Abbiamo quindi mostrato che un progetto accettabile deve soddisfare le condizioni (1) e (2).

Ora, l'osservazione cruciale per risolvere il problema è che, supponendo che un progetto soddisfi le condizioni (1) e (2), possiamo concludere che esso è accettabile.

La dimostrazione di questa affermazione può essere fatta per induzione su $\frac{S}{M}$, il quale è necessariamente intero, visto che il progetto rispetta la condizione (1).

Caso base $\frac{S}{M} = 0$

Ogni grattacielo ha 0 piani, e i lavori terminano in 0 settimane. Non c'è nessun carico che viene usato solo parzialmente, e non accade mai di costruire due piani dello stesso grattacielo nella stessa settimana; dunque il progetto è accettabile.

Passo induttivo Come ipotesi induttiva sappiamo che, se un progetto per cui $\frac{S}{M} = k$ rispetta le condizioni (1) e (2), allora esso è accettabile.

Dobbiamo dimostrare, sotto questa ipotesi, che dato un progetto P tale che $\frac{S}{M} = k+1$ e che rispetta le condizioni (1) e (2), allora esso è accettabile.

Consideriamo gli M grattacieli che, secondo il progetto P, dovranno essere i più alti. Questi grattacieli esistono in quanto $M \leq N$. Ognuno di essi è progettato con almeno 1 piano, perché in caso contrario si avrebbero meno di M grattacieli che hanno effettivamente dei piani; ognuno di essi ha al massimo $\frac{S}{M}$ piani per la condizione (2), e quindi si avrebbero in totale meno di S piani, che è assurdo in quanto S è per definizione il numero totale di piani.

Allora, nella prima settimana, facciamo costruire un piano per ognuno degli M grattacieli appena selezionati. A questo punto la situazione è equivalente a non aver ancora iniziato i lavori, e di avere un progetto P' leggermente diverso da P; in particolare, P' si ottiene decrementando di 1 le altezze degli M grattacieli più alti presenti in P. Se P' fosse accettabile, avremmo dimostrato che anche P è accettabile, che è proprio quello che vogliamo.

Analizziamo più nel dettaglio il progetto P'. Denotiamo con S' la somma delle altezze dei grattacieli previste da esso. Innanzitutto notiamo che $\frac{S'}{M} = \frac{S-M}{M} = \frac{S}{M} - 1 = (k+1) - 1 = k$. In particolare P' rispetta la condizione (1). Vorremmo ora dimostrare che esso soddisfa anche la (2).

Supponiamo per assurdo che ci sia un grattacielo in P' con almeno $\frac{S'}{M} + 1 = \frac{S}{M}$ piani. Esso non può essere uno degli M grattacieli che sono stati selezionati prima, perché in tal caso esso aveva in

P al massimo $\frac{S}{M}$ piani, e dopo la modifica al massimo $\frac{S}{M} - 1$ piani. Dunque deve trattarsi di uno dei grattacieli non modificati. Ma i grattacieli modificati erano i più alti, e allora nel progetto P ci devono essere almeno $M + 1$ grattacieli con un numero di piani non inferiore a $\frac{S}{M}$. Il che è assurdo, perché avremmo un numero di piani totale strettamente maggiore di S.

Allora P' soddisfa sia la (1) che la (2), e per ipotesi induttiva concludiamo che è accettabile, il che conclude la dimostrazione.

A questo punto possiamo dire che un progetto è accettabile se e solo se rispetta le condizioni (1) e (2).

Controllare se il progetto corrente rispetta la condizione (1) è semplice: basta mantenere una variabile che contenga la somma delle altezze di tutti i grattacieli. All'interno della funzione Cambia tale variabile può essere facilmente aggiornata, e poi è sufficiente controllare se tale variabile è divisibile per M.

Collezione di pietre (`pietre`)

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

Hai $2N$ scatole, ognuna contiene una delicatissima pietra ed è contrassegnata da un'etichetta che riporta un numero da 1 a $2N$. Le scatole numerate da 1 a N sono rosse, le altre sono blu. Date due scatole dello stesso colore, quella contrassegnata dal numero più basso è più leggera.

Hai a disposizione una bilancia a due piatti: puoi posizionare una scatola su ogni piatto per sapere quale delle due scatole contiene la pietra più pesante. Un'operazione di questo tipo si chiama *confronto*.

Qual è il numero identificativo della G-esima pietra in ordine crescente di peso?

Implementazione

Dovrai sottoporre esattamente un file con estensione `.c` o `.cpp`.

> ☞ Tra gli allegati a questo task troverai un template (`pietre.c`, `pietre.cpp`) con un esempio di implementazione.

Dovrai implementare la funzione `Trova` utilizzando il seguente prototipo:

```
int Trova(int N, int G);
```

La funzione deve restituire il numero identificativo della G-esima pietra in ordine crescente di peso.

Per confrontare la scatola numero a con la scatola numero b puoi chiamare la seguente funzione:

```
int Confronta(int a, int b);
```

Non devi implementare questa funzione: essa è già implementata nel grader. Essa restituisce 0 se la scatola a contiene la pietra più leggera, altrimenti restituisce 1. Ad ogni chiamata, a deve essere diverso da b, altrimenti la soluzione sarà considerata errata. Inoltre a e b devono essere compresi tra 1 e $2N$. Questa funzione può essere chiamata al più D volte, dove D è un parametro che varia a seconda del subtask. Se la funzione `Confronta` viene chiamata più di D volte, l'esecuzione del programma viene immediatamente bloccata e la soluzione considerata errata.

Assunzioni

- $1 \leq N \leq 1000000$.
- Non ci sono due scatole con lo stesso peso.
- $1 \leq G \leq 2N$

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 relativi ad esso.

- **Subtask 1 [0 punti]:** Caso d'esempio.
- **Subtask 2 [9 punti]:** $N \leq 1000; D = 10\,000\,000$.

- **Subtask 3 [13 punti]:** $N \leq 10\,000; D = 300\,000$.
- **Subtask 4 [20 punti]:** $N \leq 100\,000; D = 200\,000$.
- **Subtask 5 [18 punti]:** $N \leq 1\,000\,000; D = 1000$.
- **Subtask 6 [40 punti]:** $N \leq 1\,000\,000; D = 1000$.

Per ottenere il punteggio relativo ad uno dei subtask 1, 2, 3, 4, 5 è necessario risolvere correttamente tutti i test relativi ad esso.

Il subtask 6 è invece valutato diversamente. Ad ogni test di questo subtask è associato un valore Q; Q rappresenta il numero minimo di chiamate a `Confronta` che la giuria ha ritenuto sufficienti per risolvere il test in questione. A seconda della risposta data dal programma e da X, il numero di chiamate a `Confronta` effettuate dal programma, viene assegnato un punteggio relativo.

- Se la risposta non è corretta o il programma risulta interrotto viene assegnato punteggio 0.
- Se $X > 4Q$ viene assegnato punteggio 0.
- Se la risposta è corretta e $X < Q$ viene assegnato punteggio 1.
- Se la risposta è corretta e $Q \leq X \leq 4Q$ viene assegnato il punteggio p calcolato secondo la seguente formula
$$p = \frac{4Q - X}{3Q}$$
In altre parole, p vale 1 se $X = Q$, e decresce linearmente fino a valere 0 se $X = 4Q$.

Infine, se m è il minimo punteggio relativo ottenuto nei test del subtask 6, il punteggio totale per questo subtask sarà $40 \cdot m$.

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 di input dal file `input.txt`, a quel punto chiama la funzione `Trova` che dovete implementare, e scrive il risultato restituito dalla vostra funzione sul file `output.txt`.

Nel caso vogliate generare un input, il file `input.txt` deve avere questo formato:

- Riga 1: contiene N e G separati da uno spazio.
- Riga 2: contiene D.
- Riga 3: contiene una sequenza di N interi separati da spazi, i pesi delle scatole da 1 a N.
- Riga 4: contiene una sequenza di N interi separati da spazi, i pesi delle scatole da $N + 1$ a $2N$.

Nel file `output.txt` verranno stampati il numero di confronti e la risposta data dalla tua funzione.

Esempi di input/output

stdin	stdout
3 4 10 2 4 6 1 3 5	Domande effettuate = 3 Risposta = 2

Soluzione

La soluzione ottima si basa sull'idea della ricerca binaria, solo che in questo problema la lista (le pietre) non sono già completamente ordinate. Per ovviare a questa problematica serve notare l'*osservazione chiave*, che è l'esistenza di confronti *particolari* che permettono di dimezzare il numero di scatole in cui si può trovare la G-esima pietra.

Il numero di chiamate effettuate dalla soluzione ottima è $\log_2(N)$ a meno di costanti (dovrebbe essere $\log_2(N) + 2$).

■ Osservazione chiave

Scegliamo un intero $\max(1, G-N) \leq k \leq \min(N, G)$ e consideriamo la domanda Confronta$(k, N+G-k)$. Assumiamo che la risposta sia 0, cioè che la k-esima scatola rossa contiene una pietra più leggera di quella nella $(G-k)$-esima scatola blu.

Allora la k-esima scatola rossa contiene una pietra che ne ha *al più* $k-1+G-k-1 = G-2$ più leggere di lei ($k-1$ nelle scatole rosse e al più $G-k-1$ nelle scatole blu). E analogamente la $(G-k)$-esima scatola blu (che ha indice $N+G-k$) contiene una pietra che ha *almeno* $G-k-1+k = G-1$ più leggere di lei ($G-k-1$ nelle scatole blu e almeno k nelle scatole rosse). Riassumendo queste due osservazioni, arriviamo alla certezza che la G-esima pietra (quella che cerchiamo) si trova o in una scatola rossa con indice *maggiore* di k oppure in una scatola blu con indice *minore o uguale* a $N+G-k$.

Nel caso in cui Confronta$(k, N+G-k)$ risponda 1, si arriva a conclusioni opposte: la G-esima pietra (quella che cerchiamo) si trova o in una scatola rossa con indice *minore o uguale* a k oppure in una scatola blu con indice *maggiore* di $N+G-k$.

■ Pseudo-binaria

Inizializziamo due indici

$$l = \max(1, G-N)$$
$$r = \min(N, G)$$

che rappresentano il fatto che siamo sicuri che la G-esima pietra si trova in una scatola rossa tra la l e la r oppure in una scatola blu tra la $N+G-r$ e la $N+G-l$. Ed è proprio questo il punto chiave della soluzione, la scelta di questo "tipo di intervalli". Se fosse una binaria classica avremmo scelto un solo intervallo tra l ed r e inizializzato $l = 1$ e $r = 2N$, mentre nel nostro caso scegliamo due intervalli in qualche senso "simmetrici" rispetto alla soluzione.

A questo punto poniamo la domanda Confronta$(m, N+G-m)$ dove $m = \frac{l+r}{2}$ è il punto medio tra l e r (vogliamo dimezzare la larghezza dell'intervallo in cui si può trovare la G-esima pietra). Allora, in virtù dell'*osservazione chiave*, in base alla risposta a tale domanda possiamo trasformare (l, r) in $(m+1, r)$ oppure in (l, m) (rispettivamente nel caso in cui la risposta sia 0 e nel caso in cui la risposta sia 1).

Ripetendo tale dimezzamento (stando attenti a trattare adeguatamente eventuali momenti in cui risulta falso $\max(1, G-N) \leq m \leq \min(N, G)$) fino ad ottenere $l = r$, si giunge a sapere che la G-esima pietra si trova o nella l-esima scatola rossa o nella $(G-l)$-esima scatola blu. Infine per decidere quale delle due scatole sia quella corretta sarà sufficiente porre la domanda Confronta$(l, N+G-l)$.

Incendia lo trabucco (`trabucco`)

Limite di tempo:	1.75 secondi
Limite di memoria:	256 MiB

Al calar del sole, i migliori strateghi saracini si presentano al cospetto del Saladino per pianificare una sortita notturna. Bersaglio della sortita è un trabucco, enorme macchina d'assedio che conferisce ai crociati un notevole vantaggio. Le truppe saracine dovranno uscire silenziosamente dalle porte di Gerusalemme, raggiungere il trabucco e incendiarlo, di modo che la mattina seguente Goffredo, il condottiero cristiano, invece di un carismatico «Alla pugna!», si ritrovi costretto a gridare un disperato «Ove est lo trabucco?».

Gli strateghi srotolano una mappa del campo di battaglia: questa consiste in una griglia con N colonne, numerate per semplicità da 0 a $N-1$, e M righe, numerate da 0 a $M-1$. La casella (x, y) si trova all'incrocio fra la colonna x e la riga y. I movimenti, tanto delle truppe saracine quanto di quelle cristiane, si traducono sulla mappa in spostamenti da una casella a una adiacente (cioè con un lato in comune).

Le porte di Gerusalemme si trovano in $(0, 0)$, mentre il trabucco da incendiare è collocato in $(N-1, M-1)$. Tuttavia, sulla mappa sono riportate anche le posizioni di K postazioni di guardie cristiane, che i saracini preferirebbero evitare.

La scelta del percorso da seguire è fondamentale per il buon esito della missione. Innanzitutto il percorso dev'essere più breve possibile, cioè attraversare il minimo numero di caselle: ciò riduce le probabilità di essere scoperti. Ma nel caso i soldati saracini fossero individuati dalle poco sveglie e spesso ubriache sentinelle cristiane, essi preferirebbero poter rientrare in Gerusalemme illesi.

Gli strateghi definiscono *sicurezza* di una casella c il minimo numero di spostamenti necessari alla guardia più vicina per raggiungere c. La *sicurezza di un percorso* è uguale alla minore fra le sicurezze di tutte le caselle che esso attraversa (in particolare, la sicurezza di un percorso può essere 0).

Gli strateghi e il Saladino, dunque, decidono di considerare solo i percorsi della minima lunghezza possibile, e fra questi scartano tutti quelli che non hanno la massima sicurezza.

A questo punto, il sovrano vorrebbe inviare nella sortita un numero praticamente infinito di truppe, ma i suoi strateghi lo avvertono che, per migliorare le probabilità di successo, è preferibile far sì che ogni saracino segua un percorso diverso (due percorsi sono diversi se differiscono per almeno una casella).

Quanti saracini possono partecipare al più alla missione?

Implementazione

Dovrai sottoporre esattamente un file con estensione `.c` o `.cpp`.

> ☞ Tra gli allegati a questo task troverai un template (`trabucco.c`, `trabucco.cpp`) con un esempio di implementazione.

Dovrai implementare la funzione `ContaPercorsi` utilizzando il seguente prototipo:

```
int ContaPercorsi(int N, int M, int K, int* X, int* Y);
```

- N è il numero di colonne della mappa, e M è il numero di righe.
- K è il numero di postazioni di guardia cristiane segnate sulla mappa; queste si trovano sulle caselle $(X[i], Y[i])$, per $0 \leq i < K$.

La funzione deve restituire il numero massimo di saracini che possono prendere parte alla sortita, ovvero il numero di percorsi distinti che:

- partono da $(0, 0)$ e terminano in $(N - 1, M - 1)$;
- hanno la minima lunghezza;
- fra quelli che rispettano le precedenti condizioni, hanno la massima sicurezza.

☞ Attenzione: siccome questo numero potrebbe essere enorme, si richiede di restituirlo modulo $1\,000\,000\,007$ $(10^9 + 7)$.

Assunzioni

- $2 \leq N, M \leq 2000$.
- $0 \leq K \leq 200\,000$.
- $0 \leq X[i] < N$ per ogni $0 \leq i < K$.
- $0 \leq Y[i] < M$ per ogni $0 \leq i < K$.
- Le caselle $(0, 0)$ e $(N - 1, M - 1)$ sono prive di sentinelle cristiane.
- Le sentinelle si trovano tutte in posizioni diverse tra loro.

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 relativi ad esso.

- **Subtask 1 [0 punti]:** Casi d'esempio.
- **Subtask 2 [8 punti]:** $K = 0$ (non ci sono postazioni di guardia).
- **Subtask 3 [9 punti]:** $M = 2$.
- **Subtask 4 [19 punti]:** $N, M \leq 75$.
- **Subtask 5 [20 punti]:** $N, M \leq 400$.
- **Subtask 6 [22 punti]:** $N, M \leq 1200$.
- **Subtask 7 [22 punti]:** Nessuna limitazione specifica.

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 di input dal file `input.txt`, a quel punto chiama la funzione `ContaPercorsi` che dovete implementare, e scrive il risultato restituito dalla vostra funzione sul file `output.txt`.

Nel caso vogliate generare un input, il file `input.txt` deve avere questo formato:

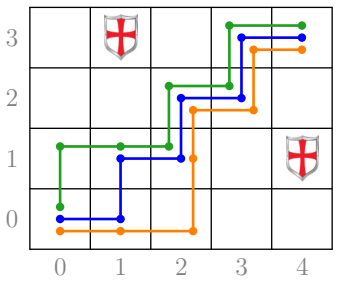
- Riga 1: contiene i numeri N, M e K.
- Riga $2 + i$: contiene $X[i], Y[i]$ separati da uno spazio (con $0 \leq i < K$).

Esempi di input/output

stdin	stdout
5 4 2 4 1 1 3	3
6 6 3 0 2 4 3 5 3	4

Spiegazione

Nel primo caso d'esempio la mappa contiene 5 colonne e 4 righe. Ci sono 2 sentinelle, una nella casella $(4, 1)$ e l'altra nella casella $(1, 3)$. Si vede facilmente che la minima lunghezza di un percorso che vada da $(0, 0)$ a $(4, 3)$ è 7, e fra i percorsi di lunghezza 7 quelli con massima sicurezza hanno sicurezza 2. Come si può notare dalla figura a destra, i percorsi cercati sono esattamente 3.



Nel secondo caso d'esempio la mappa contiene 6 righe e 6 colonne. Le 3 sentinelle si trovano nelle caselle $(0, 2)$,$(4, 3)$,$(5, 3)$. La massima sicurezza fra tutti i percorsi di minima lunghezza è 2, e i percorsi con tale sicurezza sono 4.

Note

In C/C++, l'operazione di modulo può essere effettuata per mezzo dell'operatore %.

Soluzione

Per prima cosa notiamo che i percorsi che stiamo cercando hanno lunghezza esattamente $N+M-2$. Da un lato, infatti, non esistono percorsi di lunghezza inferiore a $N+M-2$ che partono da $(0,0)$ e terminano in $(N-1, M-1)$, poichè un percorso siffatto deve comprendere almeno $N-1$ spostamenti orizzontali e almeno $M-1$ verticali, per un totale di $N+M-2$ spostamenti. D'altro canto, tutti i percorsi che partono da $(0,0)$, comprendono esattamente $N-1$ spostamenti a destra ed esattamente $M-1$ in alto terminano in $(N-1, M-1)$ e sono percorsi validi. D'ora in poi il termine *percorso* sarà riferito a un percorso di questo tipo.

■ Calcolare la sicurezza delle caselle

Vogliamo innanzitutto trovare efficientemente la sicurezza di ogni casella.

Per fare ciò possiamo considerare il grafo implicito i cui nodi sono le caselle, e gli archi collegano due caselle adiacenti. A questo punto, la sicurezza di una casella c è semplicemente la minima distanza da una casella che ospita una sentinella a c. Per calcolarla possiamo avvalerci di una versione lievemente modificata della BFS: inizialmente inseriamo nel container FIFO che stiamo usando (ad esempio una `queue` del linguaggio C++) non un solo nodo sorgente, ma tutte le K caselle che contengono una sentinella. Queste K caselle vengono marcate con sicurezza (cioè distanza) uguale a 0; da qui in poi, l'algoritmo è una comune BFS.

Concettualmente, questo algoritmo è equivalente a creare un nodo fittizio χ collegato alle K caselle contenenti una sentinella, assegnare a χ sicurezza uguale a -1 e poi lanciare una BFS con χ come sorgente.

Il grafo implicito che abbiamo utilizzato ha NM nodi e $O(NM)$ archi (da ogni nodo escono generalmente 4 archi), quindi la complessità di questo algoritmo è $O(NM)$.

■ Trovare la sicurezza massima

A questo punto siamo interessati a capire quale sia la massima sicurezza che un percorso può avere. Supponiamo di aver già calcolato la sicurezza di ogni casella (come nella sezione precedente). Chiamiamo $s(x,y)$ la sicurezza della casella (x,y).

Chiamiamo $\sigma(x,y)$ la massima sicurezza che può avere un percorso di lunghezza minima che parte da $(0,0)$ e arriva in (x,y). Cerchiamo ora di calcolare $\sigma(x,y)$ supponendo di sapere già i valori di $\sigma(x-1,y)$ e di $\sigma(x,y-1)$. Ogni percorso che parte da $(0,0)$ e termina in (x,y) passa necessariamente per (x,y) e per una fra $(x-1,y)$,$(x,y-1)$.

Fra tutti i percorsi che passano per $(x-1,y)$ per arrivare a (x,y), la sicurezza massima sarà il minimo fra la massima sicurezza ottenibile per arrivare in $(x-1,y)$ e la sicurezza di (x,y), ovvero $\min\{\sigma(x-1,y), s(x,y)\}$. Analogamente, fra tutti i percorsi che passano per $(x,y-1)$ per arrivare a (x,y), la sicurezza massima sarà $\min\{\sigma(x,y-1), s(x,y)\}$.

Quindi, complessivamente, vale

$$\sigma(x,y) = \min\{s(x,y), \max\{\sigma(x-1,y), \sigma(x,y-1)\}\}$$

Se definiamo $\sigma(x,y) = -\infty$ quando uno fra x, y è negativo (per evitare i casi estremali), otteniamo una definizione completa e ricorsiva di σ:

$$\sigma(x,y) = \begin{cases} -\infty & \text{se } x<0 \vee y<0 \\ s(0,0) & \text{se } x=y=0 \\ \min\{s(x,y), \max\{\sigma(x-1,y), \sigma(x,y-1)\}\} & \text{altrimenti} \end{cases}$$

In questo modo possiamo calcolare tutti i valori di σ utilizzando la tecnica della programmazione dinamica; la risposta alla domanda iniziale, ovvero quale sia la massima sicurezza che un percorso può avere, è esattamente $\sigma(N-1, M-1)$.

Vi sono $N \cdot M$ possibili stati (x, y) per la programmazione dinamica, dunque la complessità di questo algoritmo è $O(NM)$.

■ Contare i percorsi

Infine, vogliamo contare quanti sono i percorsi di sicurezza massima. Abbiamo già individuato al punto precedente qual è la massima sicurezza che un percorso può avere: chiamiamola Σ (ricordiamo che $\Sigma = \sigma(N-1, M-1)$).

Chiamiamo $f(x, y)$ il numero di percorsi di lunghezza minima che partono da $(0, 0)$ e arrivano a (x, y) passando solo per le caselle con sicurezza maggiore o uguale a Σ; la risposta al problema sarà dunque $f(N-1, M-1)$.

Se (x, y) ha una sicurezza sufficientemente alta (cioè $s(x, y) \geq \Sigma$), allora

$$f(x, y) = f(x-1, y) + f(x, y-1)$$

in quanto i modi di arrivare in (x, y) sono tanti quanti quelli di arrivare in $(x-1, y)$ sommati a quelli di arrivare in $(x, y-1)$. Se invece $s(x, y) < \Sigma$ allora $f(x, y) = 0$. Pertanto, se definiamo $f(x, y) = 0$ quando uno fra x, y è negativo (per evitare i casi estremali), otteniamo una definizione completa e ricorsiva di f:

$$f(x, y) = \begin{cases} 0 & \text{se } x < 0 \vee y < 0 \\ 1 & \text{se } x = y = 0 \\ 0 & \text{se } s(x, y) < \Sigma \\ f(x-1, y) + f(x, y-1) & \text{altrimenti} \end{cases}$$

Ancora una volta, possiamo calcolare $f(N-1, M-1)$ con la tecnica della programmazione dinamica, ottenendo la risposta al problema.

Vi sono $N \cdot M$ possibili stati (x, y) per la programmazione dinamica, dunque la complessità di questo algoritmo è $O(NM)$.