Laboratorio di Algoritmi e Strutture Dati

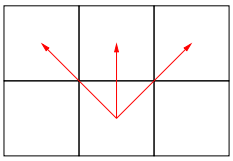
Docente: Camillo Fiorentini

8 gennaio 2008

Il problema è simile all'esercizio 15.6 del libro di testo di algoritmi (*Introduzione agli algoritmi e strutture dati*, T. Cormen et al., McgGraw-Hill, seconda edizione, 2005). Si consiglia di risovere anche qualche altro esercizio del capitolo 15 (programmazione dinamica).

Il problema

Si suppone di avere una scacchiera in cui a ogni casella è associato un valore. Da una casella della scacchiera è possibile spostarsi in una casella adiacente nella riga sopra a quella in cui ci si trova.



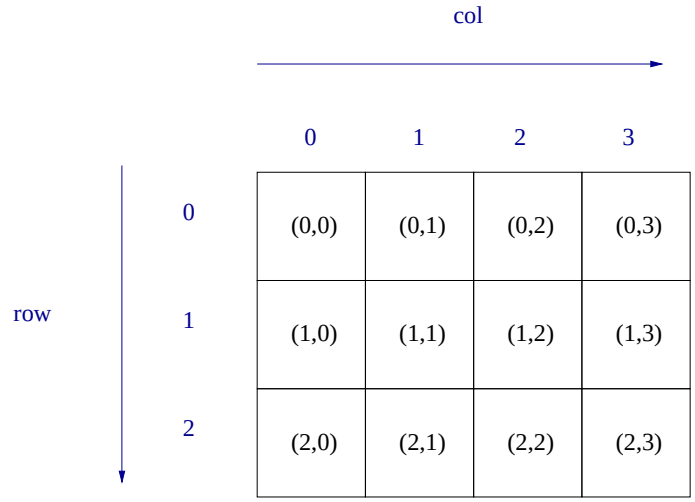
Un percorso si ottiene compiendo degli spostamenti fino ad arrivare alla prima riga della scacchiera, il valore del percorso è dato dalla somma dei valori delle caselle attraversate.

L'obiettivo è quello di determinare un percorso ottimale a partire da una casella c, ossia un percorso il cui valore è maggiore o uguale a quello di qualunque altro percorso che parte da c.

Formalizzazione del problema

La scacchiera è definita da una *matrice* di row righe e col colonne. Ogni *cella* della matrice è identificata da una coppia di interi positivi (r, c), dove r rappresenta la riga e c la colonna. Usando le convenzioni del C, gli indici di riga e colonna partono da 0, quindi $0 \leq r < row$ e $0 \leq c < col$.

Ad esempio, le celle di una matrice di dimensione 3×4 sono:



Un *percorso* dalla cella (r_0, c_0) è determinato da una successione di $r+1$ celle

$$(r_0, c_0) \longrightarrow (r_0 - 1, c_1) \longrightarrow \cdots \longrightarrow (0, c_{n+1})$$

tali che:

- per ogni $0 \leq i \leq n$, $c_{i+1} = c_i + k$, con $k \in \{-1, 0, 1\}$.

Questo significa che dalla cella (r_i, c_i) ci si può spostare in un passo in una delle celle $(r_i - 1, c_i - 1)$ (se $c_i > 0$) oppure $(r_i - 1, c_i)$ oppure $(r_i - 1, c_i + 1)$ (se $c_i < col - 1$). Il percorso termina quando si è raggiunta una cella nella riga 0.

Ad ogni cella (r, c) è associato un *valore* $Val(r, c)$, che è un numero intero. Il valore $Val(P)$ di un percorso P è dato dalla somma dei valori delle celle del percorso.

Un percorso P dalla cella (r, c) è *ottimale* se, per ogni altro percorso P' che parte dalla cella (r, c), si ha che $Val(P') \leq Val(P)$.

Esempio

Si supponga di avere una matrice 3×4 in cui i valori delle celle sono

	0	1	2	3
0	200	2	−20	4
1	−10	−10	70	5
2	0	1	2	3

Allora:

- Esiste un unico percorso ottimale dalla cella $(2, 3)$

  $$(2, 3) \longrightarrow (1, 2) \longrightarrow (0, 3)$$

  di valore 77.

- Esiste un unico percorso ottimale dalla cella $(2, 2)$ (indicato nella figura)

  $$(2, 2) \longrightarrow (1, 1) \longrightarrow (0, 0)$$

  di valore 192. Si noti che la seconda casella non è quella di valore massimo fra le scelte possibili, quindi non si può risolvere il problema usando un algoritmo greedy.

- Esistono due percorsi ottimali dalla cella $(2, 1)$

  $$(2, 1) \longrightarrow (1, 0) \longrightarrow (0, 0) \qquad\qquad (2, 1) \longrightarrow (1, 1) \longrightarrow (0, 0)$$

  di valore 191.

- Esistono due percorsi ottimali dalla cella $(2, 0)$

$$(2,0) \longrightarrow (1,0) \longrightarrow (0,0) \qquad\qquad (2,0) \longrightarrow (1,1) \longrightarrow (0,0)$$

di valore 190.

Input e output

Il programma deve leggere da standard input una sequenza di istruzioni secondo il formato nella tabella, dove row, col, r e c sono interi e $fileName$ è il nome di un file (si può assumere che $fileName$ contenga al massimo 50 caratteri). I vari elementi sulla riga sono separati da uno o più spazi. Quando una riga è letta, viene eseguita l'operazione associata; le operazioni di stampa sono effettuate sullo standard output, e ogni operazione deve iniziare su una nuova riga.

Istruzione in input	**Operazione**
`c` row col $fileName$	Costruisce una nuova matrice di dimensione $row \times col$. Se $fileName$ non esiste, i valori della matrice sono posti a 0. Altrimenti, $fileName$ deve contenere $row \cdot col$ interi separati da spazi che vengono usati per determinare i valori delle celle della matrice (nell'ordine $(0,0)$, $(0,1)$, $(0,2), \ldots, (row-1, col-1)$)
`p`	Stampa i valori delle matrici usate nel programma
`v` r c	Se la cella (r, c) è nella matrice, stampa il valore del percorso ottimale da (r, c) usando la soluzione ricorsiva
`V` r c	Se la cella (r, c) è nella matrice, stampa il valore del percorso ottimale da (r, c) usando la programmazione dinamica
`f`	Termina l'esecuzione

Si assume che l'input sia inserito correttamente. Conviene scrivere le istruzioni di input in un file `in.txt` ed eseguire il programma redirigendo lo standard input.

I valori delle celle letti in input devono essere memorizzati in una matrice di dimensione $row \times col$ da definire in `main()`. Per i dettagli sul codice, fare riferimento al file `path.c` (da completare).

Soluzione 1 (ricorsione)

Definire una funzione ricorsiva

```
int maxPathRic(int r, int c, int **mVal, int row, int col)
```

che, data la matrice dei valori `mVal` di dimensione $row \times col$, determina il valore del percorso ottimale dalla cella (r, c). Si *assume* che (r, c) sia una cella della matrice. Si noti che, nelle ipotesi fatte, il valore di una cella (r, c) è dato da $mVal[r][c]$.

Occorre ragionare per *induzione* sulla riga r della cella.

- Se $r = 0$ (la cella è nella prima riga), il valore del percorso è dato dal valore della cella.
- Se $r > 0$ si calcola ricorsivamente il valore del percorso ottimale dalle celle $(r-1, c-1)$ (se la cella è definita), $(r-1, c)$ e $(r-1, c+1)$ (se la cella è definita). Sia $maxVal$ il massimo valore ottenuto. Allora, il valore del percorso ottimale da (r, c) è dato dal valore della cella (r, c) più $maxVal$.

Soluzione 2 (programmazione dinamica)

La soluzione ricorsiva non è efficiente (provare ad esempio con una tabella 100×100). Il tempo $T(r)$ impiegato per determinare il valore del percorso ottimale da una cella sulla riga r è dato approsimativamente da:

$$T(r) \quad = \quad \begin{cases} c_1 & \text{se } r = 0 \\ 3 \cdot T(r-1) \ + \ c_2 & \text{se } r > 0 \end{cases}$$

dove c_1 e c_2 sono opportune costanti. Quindi $T(r) = \Theta(3^r)$, che significa che la funzione richiede tempo *esponenziale* rispetto a r.

Il problema deriva del fatto una stessa chiamata può essere eseguita più di una volta.

Esempio

Denotiamo con $f(r, c)$ la chiamata alla funzione ricorsiva `maxPathRic()` in cui i primi due parametri sono r e c e supponiamo di voler calcolare $f(10, 10)$.

- Il calcolo di $f(10, 10)$ richiede il calcolo di $f(9, 9)$, $f(9, 10)$ e $f(9, 11)$.
- A sua volta, per calcolare $f(9, 9)$, $f(9, 10)$ e $f(9, 11)$ occorre calcolare $f(8, 10)$. Questo significa che la chiamata a $f(8, 10)$ viene effetuata 3 volte.

Rappresentando il grafo delle chiamate, si vede che ci sono molte chiamate che vengono inutilmente ripetute. In questi casi occorre usare le metodologie della *programmazione dinamica* che consiste nel memorizzare i risultati delle chiamate ricorsive, in modo che il risultato di una chiamata $f(r, c)$ venga calcolato solo la prima volta e memorizzato in una tabella. Le successive chiamate a $f(r, c)$ non richiedono alcun calcolo, ma solo la lettura del risultato dalla tabella.

Nel nostro caso si può definire in `main()` una matrice $matrPathVal$ (matrice dei valori dei percorsi ottimali) di dimensione $row \times col$ tale che, per ogni cella (r, c) nella matrice, vale:

- $matrPathVal[r][c]$ = valore del percorso ottimale dalla cella (r, c).

In altri termini, $matrPathVal[r][c]$ è il risultato della chiamata $f(r, c)$. Seguendo il ragionamento del punto precedente, la matrice va compilata partendo dalla riga $r = 0$ e man mano aumentando r di uno, fino all'ultima riga ($r = row - 1$).

Esempio

Supponiamo che la matrice di valori sia

	0	1	2	3
0	200	2	−20	4
1	−10	−10	70	5
2	0	1	2	3

La matrice dei percorsi ottimali è

	0	1	2	3
0	200	2	−20	4
1	190	190	74	9
2	190	191	192	77

Per determinare i valori della matrice, occorre scrivere il codice della funzione `setPathVal()` di `path.c`

Nella funzione `main()` si può fare in modo che la tabella dei valori ottimali sia calcolata subito dopo la lettura della tabella dei valori. Il calcolo della tabella richiede tempo $O(row \cdot col)$ e le successive operazioni di calcolo del percorso ottimale hanno tempo costante (si deve semplicemente leggere un valore in tale tabella).

Fare in modo che operazione `p` stampi il contenuto della tabella.

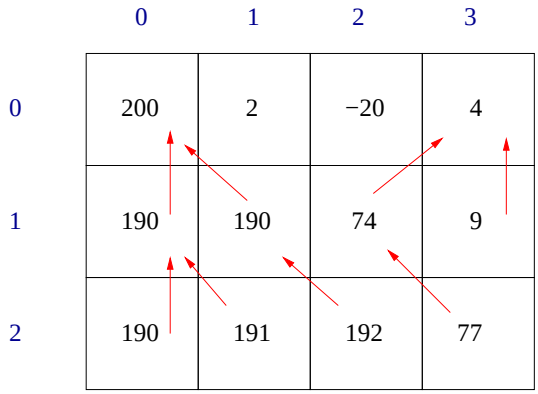
Verificare i miglioramenti che si ottengono con una matrice 100×100.

Determinzione del percorso

Usando la programmazione dinamica, si può a basso costo determinare come sono fatti i percorsi ottimali.

Esempio

Nella tabella precedente, supponiamo di aggiungere in ogni cella una freccia che indica come è stato ottenuto il valore ottimale



Seguendo le frecce, è possibile determinare il percorso ottimale a partire da una qualunque cella.

Conviene definire in `main()` una nuova matrice $matrPathDir$ (matrice della direzione dei percorsi ottimali) di dimensione $row \times col$ tale che $mPathDir[r][c]$ indica qual è la prossima cella di un percorso ottimale che parte da (r, c). Più precisamente

- $matrPathDir[r][c] = -1$ se la cella successiva a (r, c) è la cella $(r-1, c-1)$.
- $matrPathDir[r][c] = 0$ se la cella successiva a (r, c) è la cella $(r-1, c)$.
- $matrPathDir[r][c] = 1$ se la cella successiva a (r, c) è la cella $(r-1, c+1)$.

Nell'esempio precedente la matrice delle direzioni è:

	0	1	2	3
0	0	0	0	0
1	0	−1	1	0
2	0	−1	−1	−1

Per inizializzare la tabella, occorre scrivere il codice della funzione `setPathVal1()`. Il tempo richiesto è $O(row \cdot col)$.