Prova scritta del 4 febbraio 2019 – Nota sulla soluzione dell'esercizio 3

Questa NON è la soluzione completa dell'esercizio, ma una spiegazione su come possa essere risolto.

(a) Il robot può raggiungere la generica casella (i, j) della griglia dalle caselle $(i-1, j)$ e $(i, j-1)$, se presenti. Calcolando il valore massimo tra $c_{i-1,j}$ e $c_{i,j-1}$ si ottiene il massimo numero di monete con cui il robot *può entrare* nella casella. Se $m_{i,j} > 0$ il robot aggiungerà a tale numero quello delle monete presenti nella casella, cioè proprio $m_{i,j}$; se $m_{i,j} < 0$ il robot pagherà un pedaggio (uguale a $-m_{i,j}$) se ha un numero sufficiente di monete, altrimenti resta bloccato. Il valore di $c_{i,j}$ è pertanto uguale a $\max(c_{i-1,j}, c_{i,j-1}) + m_{i,j}$ se questo valore non è negativo, a $-\infty$ in caso contrario.

È necessario considerare alcuni casi particolari, corrispondenti alle caselle della prima riga o della prima colonna:

- Se $i = j = 1$ il robot entra nella casella $(1, 1)$ senza monete. Se $m_{1,1} > 0$ il robot acquisisce le monete presenti, pertanto $c_{1,1} = m_{1,1}$ (questo anche nel caso $m_{1,1} = 0$). Se $m_{1,1} < 0$ *il robot resta subito bloccato* perché non è in grado di pagare il pedaggio. Pertanto $c_{1,1} = -\infty$.
- Se $i = 1$ e $j > 1$ (altre caselle della prima riga) oppure $i > 1$ e $j = 1$ (altre caselle della prima colonna), occorre tenere conto di un solo elemento precedente, sulla stessa riga o sulla stessa colonna. Come per le altre caselle, in caso di pedaggio le monete in possesso del robot potrebbero non essere sufficienti.

Considerando i casi precedenti, per $i, j = 1, \ldots, n$, abbiamo la seguente formula:

$$c_{i,j} = \begin{cases} m_{1,1} & \text{se } i = 1,\ j = 1 \text{ e } m_{1,1} \geq 0, \\ c_{1,j-1} + m_{1,j} & \text{se } i = 1,\ j > 1 \text{ e } c_{1,j-1} + m_{1,j} \geq 0, \\ c_{i-1,1} + m_{i,1} & \text{se } i > 1,\ j = 1 \text{ e } c_{i-1,1} + m_{i,1} \geq 0, \\ \max(c_{i-1,j}, c_{i,j-1}) + m_{i,j} & \text{se } \max(c_{i-1,j}, c_{i,j-1}) + m_{i,j} \geq 0, \\ -\infty & \text{altrimenti.} \end{cases}$$

Per evitare di trattare separatamente i casi di riga 1 e di colonna 1, al fine di ottenere una formula più compatta si può definire $c_{i,j}$ con indici da 0, ponendo uguale a $-\infty$ ciascun elemento con uno dei due indici a 0. In tal caso la formula <u>per $i, j = 0, \ldots, n$</u> è:

$$c_{i,j} = \begin{cases} m_{1,1} & \text{se } i = 1,\ j = 1 \text{ e } m_{1,1} \geq 0, \\ \max(c_{i-1,j}, c_{i,j-1}) + m_{i,j} & \text{se } \max(c_{i-1,j}, c_{i,j-1}) + m_{i,j} \geq 0, \\ -\infty & \text{altrimenti.} \end{cases}$$

Per evitare di trattare separatamente il caso $(1, 1)$, in una delle posizioni $(1, 0)$ e $(0, 1)$ di C (corrispondenti a "caselle immaginarie" da cui si può raggiungere la casella di partenza $(1, 1)$) si può inserire 0. Per $i, j = 0, \ldots, n$, la formula diventa:

$$c_{i,j} = \begin{cases} 0 & \text{se } i = 1 \text{ e } j = 0, \\ \max(c_{i-1,j}, c_{i,j-1}) + m_{i,j} & \text{se } \max(c_{i-1,j}, c_{i,j-1}) + m_{i,j} \geq 0, \\ -\infty & \text{altrimenti.} \end{cases}$$

È invece *sbagliato* aggiungere alla matrice C una riga 0 e una colonna 0 contenenti 0 in tutte le posizioni, per trattare gli elementi della prima riga e della prima colonna come quelli di indici > 1. Ad esempio, per $M = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$, si otterrebbe $c_{1,3} = 1$, mentre la matrice C corretta è $\begin{bmatrix} 1 & -\infty & -\infty \\ -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty \end{bmatrix}$.

(b) Si può scrivere un algoritmo che riempie la matrice C, calcolando i valori sulla base di una delle formule precedenti. Una volta riempita la matrice, il valore $c_{n,n}$ permette di fornire la risposta al problema: se è $-\infty$ il robot non riesce a raggiungere la casella (n, n) o la raggiunge ma è bloccato perché non può pagare il pedaggio, altrimenti $c_{n,n}$ indica il numero massimo di monete che il robot riesce a collezionare.

La parte di algoritmo che riempie la matrice C può essere scritta in vari modi. Utilizziamo qui la prima delle formule presentate sopra. Osserviamo che nei primi 4 casi della formula, il valore x assegnato a $c_{i,j}$ è utilizzato anche per esprimere la condizione, nella quale si richiede $x \geq 0$. In caso contrario si deve assegnare $-\infty$. Per scrivere il codice in maniera più compatta, nei vari casi assegneremo il valore calcolato x a $c_{i,j}$ e, successivamente, una volta sola per tutti i casi (incluso $i = j = 1$), se $c_{i,j} < 0$ assegnamo a $c_{i,j}$ $-\infty$.

La parte di pseudocodice che riempie la matrice C è la seguente (dove $\max(x, y)$ indica una funzione che restituisce il massimo tra i valore degli argomenti):

```
FOR i ← 1 TO n DO
    FOR j ← 1 TO n DO
        IF i = 1 AND j = 1 THEN c_{1,1} ← m_{1,1}
        ELSE IF i = 1 THEN c_{1,j} ← c_{1,j−1} + m_{1,j}
        ELSE IF j = 1 THEN c_{i,1} ← c_{i−1,1} + m_{i,1}
        ELSE c_{i,j} ← max(c_{i−1,j}, c_{i,j−1}) + m_{i,j}
        IF c_{i,j} < 0 THEN c_{i,j} ← −∞
```

Le parti di pseudocodice non indicate sono la creazione iniziale della matrice C e la restituizione finale del risultato.

(c) La stima del tempo deve essere ricavata e *giustificata* sulla base dello pseudocodice scritto. In questo caso la parte più costosa è quella riportata sopra, con due cicli innestati, ciascuno dei quali effettua n iterazioni. Pertanto le istruzioni nelle 5 righe interne vengono ripetute n^2 volte. Ciascuna di tali istruzioni può eseguire un numero costante di confronti e assegnamenti, ciascuno dei quali può essere effettuato in tempo costante. Pertanto il tempo totale risulta dell'ordine di n^2.

(d) In generale, una casella (i, j) può essere stata raggiunta da una delle due caselle $(i-1, j)$ o $(i, j-1)$ (o da una sola di queste, se $i = 1$ o $j = 1$). In base alle scelte fatte dall'algoritmo, la casella che precede (i, j) sul cammino migliore che arriva in (i, j) è quella corrispondente a $\max(c_{i-1,j}, c_{i,j-1})$. Si può utilizzare questo criterio per trovare il cammino *partendo dalla casella* (n, n): si individua la casella precedente tra $(n-1, n)$ e $(n, n-1)$ calcolando $\max(c_{n-1,n}, c_{n,n-1})$. Dalla casella trovata si procede a ritroso ripetendo nello stesso modo sino ad arrivare ad $(1, 1)$.

(e) Con la strategia indicata al punto (d), dalla casella (n, n) si raggiunge in un certo numero di passi la casella $(1, 1)$. All'inizio la somma dei due indici è $2n$; ad ogni passo un indice risulta decrementato, mentre l'altro non cambia; alla fine la somma degli indici è 2. Pertanto il numero di passi è $2n - 2$. Dunque questa parte di algoritmo effettua $\Theta(n)$ iterazioni tra caselle. Anche in questo caso ogni iterazione effettua un numero costante di operazioni elementari per determinare la casella precedente. Pertanto il tempo utilizzato da questa parte è dell'ordine di n.

Per la parte (d), si osservi che la strategia *greedy* che cerca di ricostruire il cammino "in avanti" a partire dalla casella $(1, 1)$, muovendosi da una casella (i, j) alla casella $(i, j + 1)$ o $(i + 1, j)$ con valore massimo nella matrice C, non trova in generale la soluzione corretta (si considerino gli esempi forniti nel testo dell'esercizio).