Mosse sulla scacchiera. *Supponete di avere una scacchiera di $n \times n$ caselle e una pedina che dovete muovere dall'estremità inferiore della scacchiera, rispettando le seguenti regole. In ogni mossa la pedina può andare in una delle seguenti caselle:*

(1) *la casella immediatamente in alto;*
(2) *la casella immediatamente in alto a sinistra (a meno che la pedina non si trovi già nell'ultima colonna a sinistra);*
(3) *la casella immediatamente in alto a destra (a meno che la pedina non si trovi già nell'ultima colonna a destra).*

Ogni volta che spostate la pedina dalla casella x alla casella y, riceverete $p(x, y)$ Euro. Riceverete tale importo per tutte le coppie (x, y) per le quali è valida la mossa da x a y. Non bisogna supporre che $p(x, y)$ sia positivo.

Descrivete un algoritmo che calcola l'insieme delle mosse che dovrà fare la pedina partendo da una casella dell'estremità inferiore della scacchiera fino a raggiungere una casella dell'estremità superiore, guadagnando il maggiore numero possibile di Euro. L'algortimo è libero di scegliere qualsiasi casella nell'estremità inferiore come punto di partenza e qualsiasi casella nell'estremità superiore come punto di arrivo. Qual è il tempo di esecuzione del vostro algortimo?

Soluzione.

Denotiamo con $M(r, c)$ il massimo pagamento ottenibile da un cammino che parte dalla casella (r, c) e che arriva ad una casella sull'estremità superiore della scacchiera. Ovviamente abbiamo che $M(n, c) = 0$, per ogni $1 \leq c \leq n$. Se invece $1 \leq r < n$ abbiamo

$$M(r,c) = \begin{cases} \max\{M(r+1,c) + p((r,c),(r+1,c)), \\ \qquad M(r+1,c+1) + p((r,c),(r+1,c+1))\}, & \text{se } c = 1; \\ \max\{M(r+1,c) + p((r,c),(r+1,c)), \\ \qquad M(r+1,c-1) + p((r,c),(r+1,c-1))\}, & \text{se } c = n; \\ \max\{M(r+1,c) + p((r,c),(r+1,c)), \\ \qquad M(r+1,c-1) + p((r,c),(r+1,c-1)), \\ \qquad M(r+1,c+1) + p((r,c),(r+1,c+1))\}, & \text{altrimenti.} \end{cases}$$

Progettiamo quindi un algoritmo che calcola la matrice $M(r, c)$ per valori decrescenti di r. Una volta calcolati i valore della prima riga, il valore richiesto è il massimo di $M(1, c)$ per $1 \leq c \leq n$. Per poter dare in output non solo il valore massimo ma anche il cammino che ottiene tale valore massimo, per ogni casella (r, c) conserviamo anche il primo passo del cammino che, tra quelli che partono da (r, c), ha il valore massimo. Più precisamente, per ogni casella (r, c) l'algoritmo calcola $S(r, c)$ che è uguale a $-1, 0$, oppure $+1$ a secondo se il cammino di valore massimo che parte dalla casella (r, c) si sposta nella colonna a sinistra (-1), resta nella stessa colonna (0) o si sposta nella colonna a destra $(+1)$ quando passa dalla riga r alla riga $r + 1$.

L'algoritmo discusso è descritto dal seguente pseudocodice.

Algoritmo PEDINA$(n, p(\cdot,\cdot))$

01. **for** $c = 1$ **to** n
02. $\quad M(n, c) = 0$;
03. **end**;
04. **for** $r = n - 1$ **downto** 1
05. $\quad$ **for** $c = 1$ **to** n
06. $\quad\quad M(r, c) = M(r, c + 1) + p((r, c), (r + 1, c))$;
07. $\quad\quad S(r, c) = 0$;
08. $\quad\quad$ **if** $(c \neq 1)$ **and** $M(r + 1, c - 1) + p((r, c), (r + 1, c - 1)) > M(r, c)$ **then**
09. $\quad\quad\quad M(r, c) = M(r + 1, c - 1) + p((r, c), (r + 1, c - 1))$;
10. $\quad\quad\quad S(r, c) = -1$;
11. $\quad\quad$ **if** $(c \neq n)$ **and** $M(r + 1, c + 1) + p((r, c), (r + 1, c + 1)) > M(r, c)$ **then**
12. $\quad\quad\quad M(r, c) = M(r + 1, c + 1) + p((r, c), (r + 1, c + 1))$;
13. $\quad\quad\quad S(r, c) = +1$;
14. $\quad$ **end**;
15. **end**;
16. $M = M(1, 1)$;
17. $S = 1$;
18. **for** $c = 2$ **to** n
19. $\quad$ **if** $(M < M(1, c))$ **then**
20. $\quad\quad M = M(1, c)$;
21. $\quad\quad S = c$;
22. **end**;
23. **Print** LA SEQUENZA DI MASSIMO PROFITTO VALE M;
24. **Print** E CONSISTE DELLE SEGUENTI MOSSE;
25. **for** $r = 1$ **to** $n - 1$
26. $\quad$ **Print**$((r, S), (r + 1, S + S(r, c))$;
27. $\quad S = S + S(r, c)$;

Esercizio 15.1-4

Le tabelle con i valori di $f_i[j]$ e $l_i[j]$ contengono, in totale, $4n-2$ elementi. Spiegate come ridurre lo spazio in memoria a un totale di $2n+2$ elementi, continuando a calcolare $f^\star$ e a essere in grado di elencare tutte le stazione del percorso più rapido che attraversa lo stabilimento.

Soluzione. Notiamo che ad ogni iterazione del ciclo *for* su j dell'algoritmo FASTEST-WAY, il calcolo di $f_1[j]$, $f_2[j]$, $l_1[j]$ e $l_2[j]$ dipende solo dai valori $f_1[j-1]$, $f_2[j-1]$. Quindi possiamo evitare di memorizzare l'intera tabella $f_i[j]$ (che occupa spazio $2n$) e invece memorizzare solo i due valori calcolati all'iterazione precedente e i due valori calcolati all'iterazione corrente (occupando quindi spazio 4). La memoria richiesta è quindi $2n-2$ (per la tabella $l_i[j]$) $+ 4$.

Esercizio 16.1-2

Anziché selezionare sempre l'attività che finisce per prima, supponete di selezionare quella che inizia per ultima tra quelle compatibili con tutte le attività precedentemente selezionate. Dimostrate che questo algoritmo fornisce una soluzione ottima.

Soluzione. Consideriamo le n attività $A = (a_1, \cdots, a_n)$ ordinate in ordine decrescente per tempo di inizio. Proviamo che esiste sempre una soluzione ottima che consiste dell'attività a_1 e dalla soluzione ottima per l'istanza A' che si ottiene da A eliminando a_1 e tutte le attività in conflitto con a_1.

Supponiamo che esista una soluzione S per A che non contiene a_1 e sia a_j l'attività che inizia per ultima tra quelle di S. Allora la soluzione $S^\star = S \setminus \{a_j\} \cup \{a_1\}$ è ammissibile, contiene a_1 ed è ottima.

Sia ora S una soluzione ottima che contiene a_1 e supponiamo che $S - \{a_1\}$ non sia ottima per A'. Allora, se S' è una soluzione ottima per A', abbiamo che $S' \cup \{a_1\}$ è una soluzione per A di cardinalità maggiore di S.

Una soluzione alternativa al problema si ottiene facendo la seguente osservazione. Denotiamo con N il tempo massimo di fine di un'attività. Per un'istanza A costruiamo l'istanza $\bar{A}$ che, per ogni $a_i \in A$, contiene l'attività $\bar{a}_i$ definita come segue: se $a_i = [s_i, f_i)$ allora $\bar{a}_i = [N - f_i, N - s_i)$. È facile osservare che se S è una soluzione ottima per A allora $\bar{S}$ è una soluzione ottima per $\bar{A}$. Infine osserviamo che scegliere le attività in A in ordine crescente di tempo di fine corrisponde a scegliere le attività in ordine decrescente di tempo di inizio in $\bar{A}$. Poiché sappiamo che il primo algoritmo (ordine crescente per tempo di fine) restituisce una soluzione ottima per A, allora il secondo algoritmo (ordine decrescente per tempo di inizio) fornisce una soluzione ottima per $\bar{A}$.