ESERCIZI PD: soluzioni

1) Supponete di avere una scacchiera con n × n caselle e una pedina. Quando posizionata sulla generica casella (i,j) per la pedina sono possibili al più due mosse:

- spostarsi verso il basso nella casella (i+1, j), se i < n-1;
- spostarsi verso destra nella casella (i, j+1), se j < n-1.

Progettare un algoritmo di programmazione dinamica che calcola il numero di percorsi possibili per spostare la pedina dalla casella (0, 0) in alto a sinistra alla casella (n-1, n-1) in basso a destra (ad esempio, in una scacchiera 3 × 3 i percorsi possibili sono 6). Valutare la complessità dell'algoritmo proposto.

SOLUZIONE

ContaPercorsi(n)

```
L = nuova matrice n x n;
// soluzione problemi elementari e memorizzazione nella matrice
for (i = 0; i < n; i++) L[i,0] = 1;
for (j = 0; j < n; j++) L[0,j] = 1;
// riempimento della matrice
for (i = 1; i < n; i++) {
  for (j = 1; j < n; j++) {
        L[i,j] = L[i-1,j] + L[i,j-1];
  }
}
return L[n-1,n-1];
```

Complessità: $T(n) = \Theta(n^2)$

2) Una pedina è posizionata sulla casella (0, 0) in alto a sinistra di una scacchiera n x n e deve raggiungere la casella (n-1, n-1) in basso a destra. Quando posizionata sulla generica casella (i, j) per la pedina sono possibili al più due mosse:

- spostarsi verso il basso nella casella (i+1, j), posto che i < n-1
- spostarsi verso destra nella casella (i, j+1), posto che j < n-1.

La pedina raggiungerà dunque la casella (n-1, n-1) con un cammino che la porterà a toccare (2n-2) caselle. Ad ogni casella della scacchiera è inoltre associato un valore c[i,j], ed il valore del cammino è dato dalla somma dei valori delle caselle toccate dalla pedina.
Progettare un algoritmo che trova il cammino di valore massimo in $O(n^2)$.

SOLUZIONE

1) Sottoproblemi

Trovare il cammino di valore massimo dalla casella (0,0) alla casella (i,j).
Il valore del cammino si memorizza in una matrice T.

2) Sottoproblemi elementari

$$T[0,0] = c[0,0]$$

Per raggiungere una casella sulla prima riga ci si può muovere solo in orizzontale:

$$T[0,j] = T[0,j-1] + c[0,j] \qquad j > 0$$

Per raggiungere una casella sulla prima colonna ci si può muovere solo in verticale:

$$T[i,0] = T[i-1,0] + c[i,0] \qquad i > 0$$

3) Passo

Una casella si può raggiungere solo spostandosi verso destra oppure verso il basso:

$$T[i,j] = c[i,j] + \max \{T[i-1,j], T[i,j-1]\}$$

4) Soluzione

$$T[n-1,n-1]$$

```
Scacchiera(c) //c: matrice dei valori delle caselle

T = nuova matrice n x n;
T[0,0] = c[0,0];
for (i = 1; i < n; i++) T[i,0] = T[i-1,0] + c[i,0];
for (j = 1; j < n; j++) T[0,j] = T[0,j-1] + c[0,j];
for (i = 1; i < n; i++) {
  for (j = 1; j < n; j++) {
        if (T[i-1,j] ≥ T[i,j-1])  T[i,j] = c[i,j] + T[i-1,j];
        else T[i,j] = c[i,j] + T[i,j-1];
  }
}
// ricostruzione del cammino
cammino = nuovo array di dimensione 2n–2
i = n-1;
j = n-1;
for (k = 2n-3; k ≥ 0; k--) {
  if (T[i,j] == c[i,j] + T[i-1,j]) { cammino[k] = ↓; i--; }
  else { cammino[k] = →; j--; }

print(cammino);
return T[n-1,n-1];
```

3) Supponete di avere una scacchiera con n x n caselle e una pedina che dovete muovere dall'estremità inferiore a quella superiore. Una pedina si può muovere una cella in alto, oppure una cella in diagonale destra, oppure una cella in diagonale sinistra. Le celle sono denotate da una coppia di coordinate (x,y). Quando una cella (x,y) viene visitata, guadagnate p(x,y) ≥ 0.
Calcolare un percorso da una qualunque casella dell'estremità inferiore ad una qualunque casella dell'estremità superiore, massimizzando il profitto. Definire anche una procedura che stampi il percorso calcolato.

```
RicercaCammino(p) //p: matrice dei profitti

g = nuova matrice n x n; // tabella di PD
m = nuova matrice n x n; // matrice per la ricostruzione del percorso
// l'ultima riga di g si inizializza con i profitti
for (j = 0; j < n; j++) g[n-1,j] = p[n-1,j];
// riempimento della tabella dal basso verso l'alto
for (i = n-2; i ≥ 0; i--) {
     for (j = 0; j < n; j++) {
           // ricerca del massimo tra g[i+1,j-1], g[i+1,j], g[i+1,j+1]
           g[i,j] = g[i+1,j];
           m[i,j] = 0; // movimento in alto: j invariato
           if (j > 0 && g[i+1,j-1] > g[i,j]) {
                 g[i,j] = g[i+1,j-1];
                 m[i,j] = -1; // movimento diagonale sx: j decrementato
           }
           if (j < n-1 && g[i+1,j+1] > g[i,j]) {
                 g[i,j] = g[i+1,j+1];
                 m[i,j] = 1; // movimento diagonale dx: j incrementato
           }
       g[i,j] = g[i,j] + p[i,j];  // profitto della cella (i,j)
     }
}
// ricerca della casella iniziale con massimo profitto
max = g[0,0];
start = 0;
for (j = 1; j < n; j++) {
     if (g[0,j] > max) {
           max = g[0,j];
           start = j;
     }
}
//stampa del percorso
j = start;
for (i = 0; i < n-1; i++) {
     print  (i,j) → (i+1, j+m[i,j]);
     j = j+m[i,j];
}
```