

Capitolo 8

La programmazione dinamica

La programmazione dinamica è un paradigma di problem-solving per problemi di ottimizzazione alternativo al divide et impera ricorsivo. Esso è applicabile a patto che il problema presenti una sotto-struttura ottima, cioè che il problema sia tale per cui la sua soluzione ottima comporti che siano ottime le soluzioni ai suoi sotto-problemi. Lo sviluppo di un algoritmo di programmazione dinamica può essere diviso in quattro fasi:

1. caratterizzazione della struttura di una soluzione ottima
2. definizione ricorsiva del valore di una soluzione ottima
3. calcolo del valore di una soluzione ottima con una strategia bottom-up
4. costruzione di una soluzione ottima a partire dalle informazioni calcolate.

La programmazione dinamica risolve un problema computazionale suddividendolo in sotto-problemi. Tali sotto-problemi non sono considerati indipendenti come avviene nel divide et impera ricorsivo. Essi vengono risolti una volta sola e le loro soluzioni riutilizzate, così da evitare di calcolarle nuovamente ogni volta che deve essere risolto lo stesso sotto-problema. Per la programmazione dinamica in generale la complessità è $T(n) = \Theta(n \cdot c)$ dove n è il numero di sotto-problemi e c è il costo di ricombinazione delle loro soluzioni ottime.

La programmazione dinamica è *conveniente* quando il numero di sotto-problemi indipendenti è limitato.

8.1 Prodotto in catena di matrici

8.1.1 Richiami di teoria

Data una sequenza di n matrici A_i con $1 \leq i \leq n$ si vuole calcolare il prodotto matriciale $A_1 \cdot A_2 \dots A_n$ utilizzando l'algoritmo del prodotto di due matrici, dopo una opportuna parentesizzazione che definisca l'ordine dei prodotti tra le matrici della sequenza. La struttura della parentesizzazione influenza sul costo del calcolo del prodotto matriciale.

Due matrici A e B possono essere moltiplicate solo se sono compatibili, cioè se il numero di colonne di A è uguale al numero delle righe di B . Se A è una matrice

$(p \times q)$ e B una matrice $(q \times r)$, il risultato del prodotto $A \cdot B$ è una matrice C di dimensioni $(p \times r)$ e il numero di operazioni necessario è $p \cdot q \cdot r$. La complessità per matrici quadrate è $T(n) = \Theta(n^3)$.

```

1 void matrMult(int **A, int **B, int **C, int p, int q, int r) {
2     int i, j, k;
3
4     for (i=0; i<p; i++)
5         for (j=0; j<r; j++) {
6             C[i][j] = 0;
7             for (k=0; k<q; k++)
8                 C[i][j] = C[i][j] + A[i][k]*B[k][j];
9         }
10    }

```

Formulazione del problema

Data una sequenza di n matrici compatibili a due a due A_1, A_2, \dots, A_n , dove la generica matrice A_i ha dimensioni $(p_{i-1} \times p_i)$ con $i = 1, 2, \dots, n$, si determini una parentesizzazione del prodotto $A_{1 \dots n} = A_1 \cdot A_2 \dots A_n$ che minimizzi il numero complessivo di moltiplicazioni scalari.

Per il prodotto di una coppia di matrici $A_1 \cdot A_2$ di dimensioni $(p_0 \times p_1)$ e $(p_1 \times p_2)$ rispettivamente, il costo, cioè il numero di moltiplicazioni scalari, è $p_0 \cdot p_1 \cdot p_2$.

Il numero di parentesizzazioni $P(n)$ per una catena di n matrici è $P(n) = C(n-1)$, dove $C(n)$ è detto numero di Catalan e vale

$$C(n) = \frac{1}{n+1} \binom{2n}{n}$$

Esempio 8.1 Date 3 matrici $A_1(10 \times 100)$, $A_2(100 \times 5)$ e $A_3(5 \times 50)$, vi sono 2 parentesizzazioni:

- ▷ $(A_1 \cdot A_2) \cdot A_3$
- ▷ $A_1 \cdot (A_2 \cdot A_3)$

Per la prima soluzione, il costo di $A_1 \cdot A_2$ è $10 \cdot 100 \cdot 5 = 5000$ e il risultato A_{12} ha dimensioni 10×5 . Il prodotto $A_{12} \cdot A_3$ ha costo $10 \cdot 5 \cdot 50 = 2500$. Il costo complessivo è quindi pari a 7500. Per la seconda soluzione, il costo di $A_2 \cdot A_3$ è $100 \cdot 5 \cdot 50 = 25000$ e il risultato A_{23} ha dimensioni 100×50 . Il prodotto $A_1 \cdot A_{23}$ ha costo $10 \cdot 100 \cdot 50 = 50000$. Il costo complessivo è quindi pari a 75000.

I passi della soluzione con la programmazione dinamica

Una soluzione ottima del problema della parentesizzazione del prodotto tra n matrici divide il problema nel prodotto di due matrici $A_{1 \dots k}$ e $A_{k+1 \dots n}$ ottenute una come prodotto delle prime k matrici e l'altra come il prodotto delle restanti $n - k$ matrici. Il costo di tale parentesizzazione è dato dalla somma del costo del calcolo di $A_{1 \dots k}$, del costo del calcolo di $A_{k+1 \dots n}$ e del costo del prodotto delle due matrici. Si dà per dimostrato il fatto che l'ottimalità della soluzione globale comporti che siano ottime le soluzioni dei due sotto-problemi (passo 1: caratterizzazione della struttura di una soluzione ottima).

Il secondo passo consiste nell'identificazione del valore della soluzione ottima in maniera ricorsiva. Si utilizzano 2 tabelle: $m[1 \dots n, 1 \dots n]$ per memorizzare il costo

$m[i,j]$ del prodotto $A_{i \dots j}$ e identificare il costo minimo e $s[1 \dots n, 1 \dots n]$ per tenere traccia del valore ottimo di k per ricostruire la soluzione. La condizione di terminazione si ha quando $i = j$ e non si effettua alcun prodotto. Di conseguenza $m[i,j] = 0$. Altrimenti per $i < j$ si ha che:

$$m[i,j] = m[i,k] + m[k+1,j] + p_{i-1} \cdot p_k \cdot p_j$$

dove $m[i,k]$ e $m[k+1,j]$ sono i costi minimi dei due sotto-problemi. Il valore di k è scelto in modo tale da rendere minimo $m[i,j]$ e tale valore viene memorizzato in $s[i,j]$. Il codice seguente implementa l'approccio ricorsivo.

```

1 int matrix_chainR(int *p, int n) {
2     return minCostR(p, 1, n, INT_MAX);
3 }
4 int minCostR(int *p, int i, int j, int minCost) {
5     int k, cost;
6
7     if (i == j)
8         return 0;
9     for (k=i; k<j; k++) {
10         cost = minCostR(p, i, k, minCost) + minCostR(p, k+1, j, minCost) +
11             p[i-1]*p[k]*p[j];
12         if (cost < minCost)
13             minCost = cost;
14     }
15 }
16
17 return minCost;
18 }
```

Nel terzo passo, ispirandosi all'algoritmo ricorsivo, si procede al calcolo bottom-up del valore della soluzione ottima secondo il codice seguente.

```

1 int matrix_chainDP(int *p, int n) {
2     int i, l, j, k, q, **m, **s;
3
4     m = calloc((n+1), sizeof(int *));
5     s = calloc((n+1), sizeof(int *));
6
7     for (i = 0; i <= n; i++) {
8         m[i] = calloc((n+1), sizeof(int));
9         s[i] = calloc((n+1), sizeof(int));
10    }
11
12    for (l = 2; l <= n; l++) {
13        for (i = 1; i <= n-l+1; i++) {
14            j = i+l-1;
15            m[i][j] = INT_MAX;
16            for (k = i; k <= j-1; k++) {
17                q = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j];
18                if (q < m[i][j]) {
19                    m[i][j] = q;
20                    s[i][j] = k;
21                }
22            }
23        }
24        displaySol(s, 1, n);
25        printf("\n");
26    }
27    return m[1][n];
28 }
29
30 void displaySol(int **s, int i, int j) {
31     if (j <= i) {
32         printf("A%d", i);
33     }
34 }
```

```

34     printf("(");
35     displaySol(s, i, s[i][j]);
36     printf(" x ");
37     displaySol(s, s[i][j]+1, j);
38     printf(")");
39     return;
40 }

```

Nel quarto passo, infine, si ricostruisce ricorsivamente la soluzione ottima secondo il codice seguente.

```

1 void displaySol(int **s, int i, int j) {
2     if (j <= i) {
3         printf("A%d", i);
4         return;
5     }
6     printf("(");
7     displaySol(s, i, s[i][j]);
8     printf(" x ");
9     displaySol(s, s[i][j]+1, j);
10    printf(")");
11
12    return;
13 }

```

8.1.2 Esercizi svolti

Esercizio

Data la catena (A_1, A_2, A_3, A_4) di matrici di dimensione (4×4) , (4×6) , (6×15) , (15×10) , rispettivamente, si determini, mediante un algoritmo di programmazione dinamica, la parentesizzazione ottima del prodotto di matrici che minimizza il numero di operazioni.

Soluzione

Il vettore p contiene $(4, 4, 6, 15, 10)$. In una prima iterazione si calcolano $m[1,1]$, $m[2,2]$, $m[3,3]$ e $m[4,4]$ tutti pari a 0. Nella seconda iterazione si calcolano, sfruttando i risultati del passo precedente:

$$\begin{aligned}
 m[1,2] &= m[1,1] + m[2,2] + p_0 \cdot p_1 \cdot p_2 = 0 + 0 + 96 \\
 m[2,3] &= m[2,2] + m[3,3] + p_1 \cdot p_2 \cdot p_3 = 0 + 0 + 360 \\
 m[3,4] &= m[3,3] + m[4,4] + p_2 \cdot p_3 \cdot p_4 = 0 + 0 + 900
 \end{aligned}$$

e i valori di k che, non essendo possibili scelte, sono rispettivamente 1, 2 e 3 e popolano la tabella s .

Nella terza iterazione si calcolano $m[1,3]$ e $m[2,4]$ sfruttando i risultati dei passi precedenti:

$$\begin{aligned}
 m[1,3] &= m[1,1] + m[2,3] + p_0 \cdot p_1 \cdot p_3 = 0 + 360 + 240 = 600 \quad \text{con } k=1 \\
 m[1,3] &= m[1,2] + m[3,3] + p_0 \cdot p_2 \cdot p_3 = 96 + 0 + 360 = 456 \quad \text{con } k=2
 \end{aligned}$$

Essendo il costo minore, si sceglie la soluzione con $k = 2$.

$$\begin{aligned}
 m[2,4] &= m[2,2] + m[3,4] + p_1 \cdot p_2 \cdot p_4 = 0 + 900 + 240 = 1140 \quad \text{con } k=1 \\
 m[2,4] &= m[2,3] + m[4,4] + p_1 \cdot p_3 \cdot p_4 = 360 + 0 + 600 = 960 \quad \text{con } k=3
 \end{aligned}$$

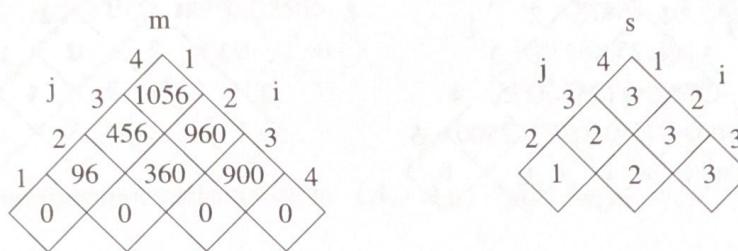
Essendo il costo minore, si sceglie la soluzione con $k = 3$.

Nella quarta iterazione si calcola infine $m[1,4]$, sempre sfruttando i risultati dei passi precedenti:

$$\begin{aligned} m[1,4] &= m[1,1] + m[2,4] + p_0 \cdot p_1 \cdot p_4 = 0 + 960 + 160 = 1120 \text{ con } k = 1 \\ m[1,4] &= m[1,2] + m[3,4] + p_0 \cdot p_2 \cdot p_4 = 96 + 900 + 240 = 1236 \text{ con } k = 2 \\ m[1,4] &= m[1,3] + m[4,4] + p_0 \cdot p_3 \cdot p_4 = 456 + 0 + 600 = 1056 \text{ con } k = 3 \end{aligned}$$

Essendo il costo minore, si sceglie la soluzione con $k = 3$.

La figura successiva riporta le configurazioni finali di m ed s . Da s si ricava ricorsivamente la parentesizzazione ottima $((A_1 \cdot A_2) \cdot A_3) \cdot A_4$.



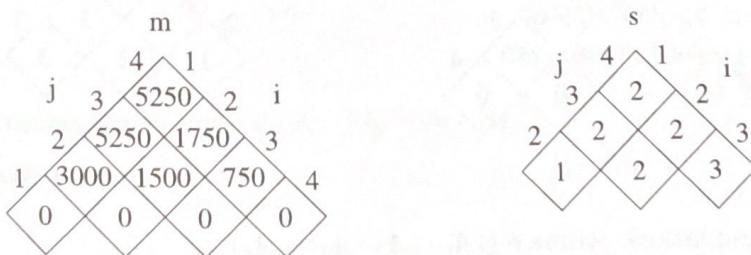
8.1.3 Esercizi risolti

Esercizio

Data la catena (A_1, A_2, A_3, A_4) di matrici di dimensione (30×20) , (20×5) , (5×15) , (15×10) rispettivamente, si determini, mediante un algoritmo di programmazione dinamica, la parentesizzazione ottima del prodotto di matrici che minimizza il numero di operazioni.

Soluzione

Nella figura successiva sono rappresentate le configurazioni finali delle matrici m e s .



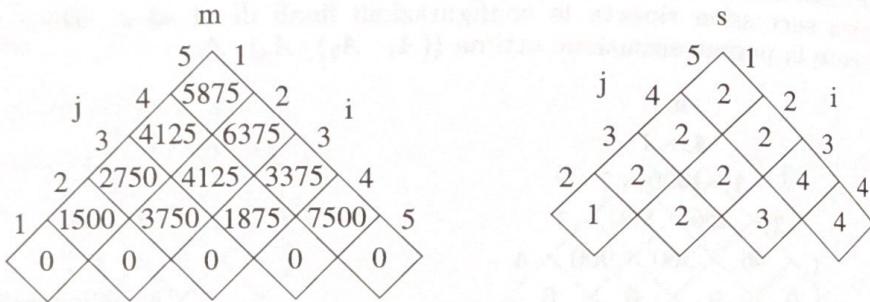
La parentesizzazione ottima è $((A_1 \cdot A_2) \cdot (A_3 \cdot A_4))$.

Esercizio

Data la catena $(A_1, A_2, A_3, A_4, A_5)$ di matrici di dimensione (10×30) , (30×5) , (5×25) , (25×15) , (15×20) rispettivamente, si determini, mediante un algoritmo di programmazione dinamica, la parentesizzazione ottima del prodotto di matrici che minimizza il numero di operazioni.

Soluzione

Nella figura successiva sono rappresentate le configurazioni finali delle matrici m e s .



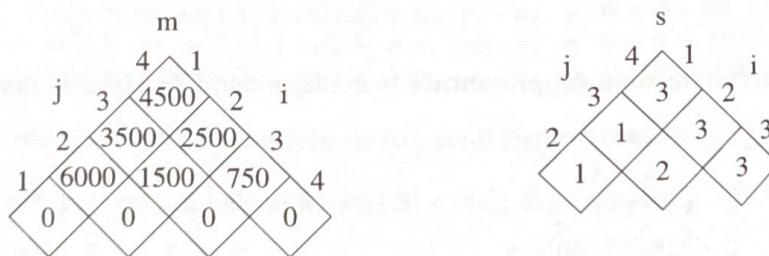
La parentesizzazione ottima è $((A_1 \cdot A_2) \cdot ((A_3 \cdot A_4) \cdot A_5))$.

Esercizio

Data la catena di matrici (A_1, A_2, A_3, A_4) di dimensione (20×20) , (20×15) , (15×5) , (5×10) rispettivamente, si determini, mediante un algoritmo di programmazione dinamica, la parentesizzazione ottima del prodotto di matrici che minimizza il numero di operazioni.

Soluzione

Nella figura successiva sono rappresentate le configurazioni finali delle matrici m e s .



La parentesizzazione ottima è $((A_1 \cdot (A_2 \cdot A_3)) \cdot A_4)$.

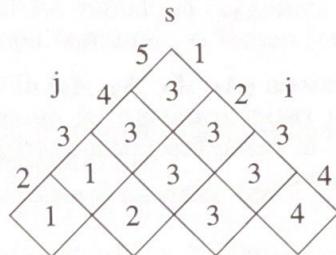
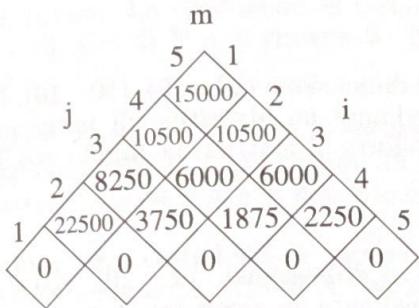
Esercizio

Data la catena $(A_1, A_2, A_3, A_4, A_5)$ di matrici di dimensione (30×30) , (30×25) , (25×5) , (5×15) , (15×30) rispettivamente, si determini, mediante un algoritmo

di programmazione dinamica, la parentesizzazione ottima del prodotto di matrici che minimizza il numero di operazioni.

Soluzione

Nella figura successiva sono rappresentate le configurazioni finali delle matrici m e s .



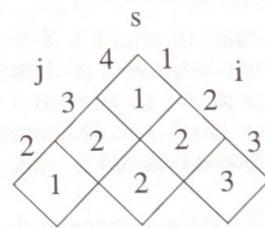
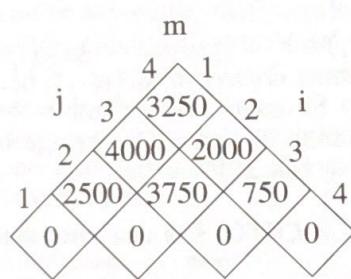
La parentesizzazione ottima è $((A_1 \cdot (A_2 \cdot A_3)) \cdot (A_4 \cdot A_5))$.

Esercizio

Data la catena (A_1, A_2, A_3, A_4) di matrici di dimensione (10×25) , (25×10) , (10×15) , (15×5) rispettivamente, si determini, mediante un algoritmo di programmazione dinamica, la parentesizzazione ottima del prodotto di matrici che minimizza il numero di operazioni.

Soluzione

Nella figura successiva sono rappresentate le configurazioni finali delle matrici m e s .



La parentesizzazione ottima è $(A_1 \cdot (A_2 \cdot (A_3 \cdot A_4)))$.

8.1.4 Esercizi proposti

Esercizio

Data la catena $(A_1, A_2, A_3, A_4, A_5)$ di matrici di dimensione (15×30) , (30×25) , (25×20) , (20×25) , (25×30) rispettivamente, si determini, mediante un algoritmo di programmazione dinamica, la parentesizzazione ottima del prodotto di matrici che minimizza il numero di operazioni.

Esercizio

Data la catena (A_1, A_2, A_3, A_4) di matrici di dimensione (10×20) , (20×10) , (10×30) , (30×2) rispettivamente, si determini, mediante un algoritmo di programmazione dinamica, la parentesizzazione ottima del prodotto di matrici che minimizza il numero di operazioni.

Esercizio

Data la catena (A_1, A_2, A_3, A_4) di matrici di dimensione (20×10) , (10×10) , (10×20) , (20×15) rispettivamente, si determini, mediante un algoritmo di programmazione dinamica, la parentesizzazione ottima del prodotto di matrici che minimizza il numero di operazioni.

Esercizio

Data la catena (A_1, A_2, A_3, A_4) di matrici di dimensione (10×20) , (20×5) , (5×15) , (15×5) rispettivamente, si determini, mediante un algoritmo di programmazione dinamica, la parentesizzazione ottima del prodotto di matrici che minimizza il numero di operazioni.

Esercizio

Data la catena (A_1, A_2, A_3, A_4) di matrici di dimensione (5×20) , (20×25) , (25×15) , (15×5) rispettivamente, si determini, mediante un algoritmo di programmazione dinamica, la parentesizzazione ottima del prodotto di matrici che minimizza il numero di operazioni.

8.2 Longest Common Subsequence

8.2.1 Richiami di teoria

Date 2 sequenze di simboli $X = (x_0, x_1, \dots, x_{m-1})$ e $Y = (y_0, y_1, \dots, y_{n-1})$ con $n \leq m$. Y si dice *sotto-sequenza* di X se esiste una sequenza crescente i_0, i_1, \dots, i_{n-1} di indici di X tali che $x_{i_j} = y_j$ per tutti i j tra 0 e $n-1$. Si osservi che in una sotto-sequenza gli indici non sono necessariamente contigui, mentre in una sotto-stringa lo sono. Si definisce *prefisso* i -esimo di una sequenza X , $X_i = (x_0, x_1, \dots, x_i)$.

Esempio 8.2 Date le stringhe $S_1 = \text{ACGCTAC}$ e $S_2 = \text{CGTC}$, S_2 è una sotto-sequenza di S_1 con indici 1, 2, 4 e 6, ma non è una sotto-stringa.

Formulazione del problema

Date 2 sequenze X e Y , determinare la sequenza Z di lunghezza massima comune ad entrambe (LCS - Longest Common Subsequence). Ad esempio, date le stringhe $S_1 = \text{ACGCTAC}$ e $S_2 = \text{CTGACA}$, le stringhe CGCA e CTAC sono sotto-sequenze comuni di lunghezza massima, mentre CGA è una sotto-sequenza comune, ma non di lunghezza massima.

I passi della soluzione con la programmazione dinamica

Si può dimostrare che, visto che una LCS di 2 sequenze X e Y contiene al suo interno una LCS dei prefissi delle 2 sequenze, il problema presenta una sotto-struttura

ottima, quindi la programmazione dinamica è applicabile (passo 1: caratterizzazione della struttura di una soluzione ottima). Essendo il numero di sotto-problemi indipendenti $\Theta(n \cdot m)$ dove n ed m sono lunghezze delle sequenze X e Y , la programmazione dinamica è conveniente.

Il secondo passo consiste nell'identificazione del valore della soluzione ottima in maniera ricorsiva. La condizione di terminazione si ha quando si raggiunge il primo elemento o di X o di Y e si ritorna 0. Nel caso non terminale si hanno le seguenti possibilità:

- ▷ l'elemento in coda a X e Y è lo stesso: si ritorna il valore ottenuto sommando a 1 la lunghezza della LCS del prefisso di X e Y accorciato di 1
- ▷ il carattere in coda a X e Y non è lo stesso: si ritorna il massimo tra:
 - ◊ la lunghezza della LCS del prefisso di X accorciato di 1 e Y (invariata)
 - ◊ la lunghezza della LCS di X (invariata) e del prefisso di Y accorciato di 1.

Il codice seguente realizza l'approccio ricorsivo.

```

1 int LCSlength(char *X, char *Y) {
2     return lengthR(X, Y, strlen(X), strlen(Y));
3 }
4
5 int lengthR(char *X, char *Y, int i, int j) {
6     if ((i == 0) || (j == 0))
7         return 0;
8     if (X[i] == Y[j])
9         return 1 + lengthR(X, Y, i-1, j-1);
10    else
11        return max(lengthR(X, Y, i-1, j), lengthR(X, Y, i, j-1));
12 }
```

Nel terzo passo, ispirandosi all'algoritmo ricorsivo, si procede al calcolo bottom-up secondo il codice seguente, utilizzando una tabella $c[0 \dots m, 0 \dots n]$ per memorizzare i costi $c[i, j]$ e identificare il costo minimo e una tabella $b[0 \dots m, 0 \dots n]$ per la costruzione dalla LCS. Si osservi che si potrebbe evitare la matrice b , in quanto il valore di $c[i, j]$ dipende solo da $c[i-1, j-1]$, $c[i-1, j]$ e $c[i, j-1]$. Negli esempi seguenti la matrice b viene utilizzata per maggiore chiarezza con i simboli \nwarrow , \uparrow , \leftarrow . Tali simboli corrispondono rispettivamente alle costanti UP_LEFT (riga 18), UP (riga 23) e LEFT (riga 27), rispettivamente.

```

1 int LCSlengthDP(char *X, char *Y) {
2     int i, j, m, n, **c, **b;
3     m = strlen(X);
4     n = strlen(Y);
5     c = calloc((m+1), sizeof(int *));
6     b = calloc((m+1), sizeof(int *));
7     for (i = 0; i <= m; i++) {
8         c[i] = calloc((n+1), sizeof(int));
9         b[i] = calloc((n+1), sizeof(int));
10    }
11    for (i=1; i<=m; i++)
12        for (j=1; j<=n; j++)
13            if (X[i-1]==Y[j-1]) {
14                c[i][j] = c[i-1][j-1]+1;
15                b[i][j] = UP_LEFT;
16            } else {
17                c[i][j] = max(c[i-1][j], c[i][j-1]);
18                b[i][j] = UP;
19            }
20 }
```

```

20     if (c[i-1][j] >= c[i][j-1]) {
21         c[i][j] = c[i-1][j];
22         b[i][j] = UP;
23     } else {
24         c[i][j] = c[i][j-1];
25         b[i][j] = LEFT;
26     }
27 }
28 printf("LCS is: ");
29 displaySol(X, Y, b);
30 printf("\n");
31
32 return c[m][n];
33 }

```

Nel quarto passo, infine, si ricostruisce ricorsivamente la soluzione ottima secondo il codice seguente.

```

1 void displaySol(char *X, char *Y, int **b) {
2     int m, n;
3     m = strlen(X);
4     n = strlen(Y);
5     displaySolR(X, Y, b, m, n);
6 }
7
8 void displaySolR(char *X, char *Y, int **b, int i, int j) {
9     if ((i==0) || (j==0)) {
10         return;
11     }
12     if (b[i][j]==UP_LEFT) {
13         displaySolR(X, Y, b, i-1, j-1);
14         printf("%c", X[i-1]);
15         return;
16     }
17     if (b[i][j]==UP)
18         displaySolR(X, Y, b, i-1, j);
19     else
20         displaySolR(X, Y, b, i, j-1);
21 }

```

8.2.2 Esercizi svolti

Esercizio

Si determini mediante un algoritmo di programmazione dinamica una Longest Common Subsequence (LCS) di: $X = ABCBDAB$ e $Y = BDCABA$.

Soluzione

Le configurazioni successive delle matrici c e b illustrano i passi intermedi necessari per costruire la prima riga significativa. La casella con il riquadro più marcato è quella corrente, quella selezionata ha sfondo grigio chiaro e quella non selezionata ha sfondo grigio scuro:

- ▷ se $X_i = Y_j$, come costo $c[i, j]$ si assegna $1 +$ il costo della casella in alto a sinistra e come $b[i, j]$ la freccia ↗
- ▷ altrimenti:
 - ◊ se il costo della casella immediatamente sopra è maggiore o uguale del costo della casella immediatamente a sinistra, si assegna il costo della casella immediatamente sopra e come $b[i, j]$ la freccia ↑

- se il costo della casella immediatamente a sinistra è maggiore o uguale del costo della casella immediatamente sopra, si assegna il costo della casella immediatamente a sinistra e come $b[i,j]$ la freccia \leftarrow .

$i = 1, j = 1$

	B	D	C	A	B	A
B	0	0	0	0	0	0
A	0	0	0	0	0	0
B	0	0	0	0	0	0
C	0	0	0	0	0	0
B	0	0	0	0	0	0
D	0	0	0	0	0	0
A	0	0	0	0	0	0
B	0	0	0	0	0	0

	B	D	C	A	B	A
A	0	0	0	0	0	0
B	0	0	0	0	0	0
C	0	0	0	0	0	0
B	0	0	0	0	0	0
D	0	0	0	0	0	0
A	0	0	0	0	0	0
B	0	0	0	0	0	0

$i = 1, j = 2$

	B	D	C	A	B	A
B	0	0	0	0	0	0
A	0	0	0	0	0	0
B	0	0	0	0	0	0
C	0	0	0	0	0	0
B	0	0	0	0	0	0
D	0	0	0	0	0	0
A	0	0	0	0	0	0
B	0	0	0	0	0	0

	B	D	C	A	B	A
A	0	0	0	0	0	0
B	0	0	0	0	0	0
C	0	0	0	0	0	0
B	0	0	0	0	0	0
D	0	0	0	0	0	0
A	0	0	0	0	0	0
B	0	0	0	0	0	0

$i = 1, j = 3$

	B	D	C	A	B	A
B	0	0	0	0	0	0
A	0	0	0	0	0	0
B	0	0	0	0	0	0
C	0	0	0	0	0	0
B	0	0	0	0	0	0
D	0	0	0	0	0	0
A	0	0	0	0	0	0
B	0	0	0	0	0	0

	B	D	C	A	B	A
A	0	0	0	0	0	0
B	0	0	0	0	0	0
C	0	0	0	0	0	0
B	0	0	0	0	0	0
D	0	0	0	0	0	0
A	0	0	0	0	0	0
B	0	0	0	0	0	0

$i = 1, j = 4$

	B	D	C	A	B	A
	0	0	0	0	0	0
A	0	0	0	0	1	0
B	0	0	0	0	0	0
C	0	0	0	0	0	0
B	0	0	0	0	0	0
D	0	0	0	0	0	0
A	0	0	0	0	0	0
B	0	0	0	0	0	0

	B	D	C	A	B	A
	0	0	0	0	0	0
A	0	↑	↑	↑	↖	0
B	0	0	0	0	0	0
C	0	0	0	0	0	0
B	0	0	0	0	0	0
D	0	0	0	0	0	0
A	0	0	0	0	0	0
B	0	0	0	0	0	0

 $i = 1, j = 5$

	B	D	C	A	B	A
	0	0	0	0	0	0
A	0	0	0	0	1	1
B	0	0	0	0	0	0
C	0	0	0	0	0	0
B	0	0	0	0	0	0
D	0	0	0	0	0	0
A	0	0	0	0	0	0
B	0	0	0	0	0	0

	B	D	C	A	B	A
	0	↑	↑	↑	↖	0
A	0	0	0	0	0	0
B	0	0	0	0	0	0
C	0	0	0	0	0	0
B	0	0	0	0	0	0
D	0	0	0	0	0	0
A	0	0	0	0	0	0
B	0	0	0	0	0	0

 $i = 1, j = 6$

	B	D	C	A	B	A
	0	0	0	0	0	0
A	0	0	0	0	1	1
B	0	0	0	0	0	0
C	0	0	0	0	0	0
B	0	0	0	0	0	0
D	0	0	0	0	0	0
A	0	0	0	0	0	0
B	0	0	0	0	0	0

	B	D	C	A	B	A
	0	↑	↑	↑	↖	←
A	0	0	0	0	0	0
B	0	0	0	0	0	0
C	0	0	0	0	0	0
B	0	0	0	0	0	0
D	0	0	0	0	0	0
A	0	0	0	0	0	0
B	0	0	0	0	0	0

La matrice successiva riporta invece la configurazione finale di c e b .

	B	D	C	A	B	A	c
A	0	0	0	0	0	0	0
B	0	0	0	1	1	1	
C	0	1	1	1	2	2	
B	0	1	1	2	2	3	
D	0	1	2	2	2	3	
A	0	1	2	2	3	3	4
B	0	1	2	2	3	4	4

	B	D	C	A	B	A	b
A	0	0	0	0	0	0	0
B	0	↑	↑	↑	↖	↖	↖
C	0	↖	↖	↑	↑	↑	↑
B	0	↖	↑	↑	↑	↑	↑
D	0	↑	↖	↑	↑	↑	↑
A	0	↑	↑	↑	↖	↖	↖
B	0	↖	↑	↑	↑	↑	↑

La matrice seguente illustra infine come viene costruita ricorsivamente la LCS a partire dalla matrice b .

$$X = ABCBDAB, Y = BDCABA, Z = BCBA$$

	B	D	C	A	B	A	c
A	0	0	0	0	0	0	0
B	0	0	0	0	1	1	1
C	0	1	1	1	1	2	2
B	0	1	1	2	2	3	3
D	0	1	2	2	2	3	3
A	0	1	2	2	3	3	4
B	0	1	2	2	3	4	4

	B	D	C	A	B	A	b
A	0	0	0	0	0	0	0
B	0	↑	↑	↑	↖	↖	↖
C	0	↖	↖	↑	↑	↑	↑
B	0	↖	↑	↑	↑	↑	↑
D	0	↑	↖	↑	↑	↑	↑
A	0	↑	↑	↑	↖	↖	↖
B	0	↖	↑	↑	↑	↑	↑

8.2.3 Esercizi risolti

Esercizio

Si determini mediante un algoritmo di programmazione dinamica una Longest Common Subsequence (LCS) di $X = ABADDACAC$ e $Y = DADCA$.

Soluzione

$$X = ABADDACAC, Y = DADCA, Z = ADCA$$

	D	A	D	C	A
A	0	0	0	0	0
B	0	0	1	1	1
A	0	0	1	1	1
D	0	1	1	2	2
D	0	1	1	2	2
A	0	1	2	2	2
C	0	1	2	2	3
A	0	1	2	2	3
C	0	1	2	2	3

	D	A	D	C	A	b
A	0	0	0	0	0	0
B	0	↑	↖	↑	↑	↖
A	0	↖	↖	↑	↑	↖
D	0	↑	↖	↑	↑	↖
D	0	↑	↖	↑	↑	↖
A	0	↑	↖	↑	↑	↖
C	0	↑	↖	↑	↑	↖
A	0	↑	↖	↑	↑	↖
C	0	↑	↖	↑	↑	↖

Esercizio

Si determini mediante un algoritmo di programmazione dinamica una Longest Common Subsequence (LCS) di $X = ACBACDB$ e $Y = ABDCA$.

Soluzione

$$X = ACBACDB, Y = ABDCA, Z = ACA$$

		c				
		A	B	D	C	A
		0	0	0	0	0
A	0	1	1	1	1	1
C	0	1	1	1	2	2
B	0	1	2	2	2	2
A	0	1	2	2	2	3
C	0	1	2	2	3	3
D	0	1	2	3	3	3
B	0	1	2	3	3	3

		b				
		A	B	D	C	A
		0	0	0	0	0
A	0	↖	↖	↖	↖	↖
C	0	↑	↑	↑	↑	↖
B	0	↑	↖	↑	↖	↖
A	0	↖	↑	↖	↑	↑
C	0	↑	↑	↑	↑	↖
D	0	↑	↑	↑	↖	↑
B	0	↑	↖	↑	↑	↑

8.2.4 Esercizi proposti**Esercizio**

Si determini mediante un algoritmo di programmazione dinamica una Longest Common Subsequence (LCS) di $X = ABACDEA$ e $Y = BCADE$.

Esercizio

Si determini mediante un algoritmo di programmazione dinamica una Longest Common Subsequence (LCS) di $X = ABACDEA$ e $Y = BCADAE$.