Problemi risolvibili con la programmazione dinamica

Abbiamo usato la programmazione dinamica per risolvere due problemi.

Cerchiamo ora di capire quali problemi si possono risolvere con questa tecnica.

Sono dei ***problemi di ottimizzazione*** in cui da un insieme (generalmente molto grande) di soluzioni possibili vogliamo estrarre una soluzione ottima rispetto ad una determinata misura.

Per poter applicare vantaggiosamente la programmazione dinamica bisogna che:

a) una soluzione ottima si possa costruire a partire da soluzioni ottime di sottoproblemi:

Proprietà di ***sottostruttura ottima*** **.**

b) che il numero di sottoproblemi distinti sia molto minore del numero di soluzioni possibili tra cui cercare quella ottima.

Altrimenti una enumerazione di tutte le soluzioni può risultare più conveniente.

Se ciò è vero significa che uno stesso problema deve comparire molte volte come sottoproblema di altri sottoproblemi.

Proprietà della ***ripetizione dei sottoproblemi*** .

Supposto che le condizioni a) e b) siano verificate, occorre scegliere l'ordine in cui calcolare le soluzioni dei sottoproblemi.

Tale ordine ci deve assicurare che nel momento in cui si risolve un sottoproblema le soluzioni dei sottoproblemi da cui esso dipende siano già state calcolate.

Ordine ***bottom-up***.

Alternativamente si può usare una procedura ricorsiva ***top-down*** che esprima direttamente la soluzione di un sottoproblema in termini delle soluzioni dei sottoproblemi da cui essa dipende.

In questo caso occorre però memorizzare le soluzioni trovate in modo che esse non vengano ricalcolate più volte.

Confronto tra algoritmo iterativo ***bottom-up*** ed algoritmo ricorsivo ***top-down*** con memoria:

Se per il calcolo della soluzione globale servono le soluzioni di tutti i sottoproblemi l'algoritmo ***bottom-up*** è migliore di quello ***top-down***.

Entrambi gli algoritmi calcolano una sola volta le soluzioni dei sottoproblemi ma il secondo è ricorsivo ed inoltre effettua un controllo in più.

Se per il calcolo della soluzione globale servono soltanto alcune delle soluzioni dei sottoproblemi l'algoritmo ***bottom-up*** le calcola comunque tutte mentre quello ***top-down*** calcola soltanto quelle che servono effettivamente.

In questo caso l'algoritmo ***top-down*** può risultare migliore di quello ***bottom-up***.

Il prossimo problema è un esempio di questa situazione.

Massima sottosequenza comune

In questo problema sono date due sequenze

$$X = x_1 x_2 \ldots x_m \quad \text{e} \quad Y = y_1 y_2 \ldots y_n$$

e si chiede di trovare la più lunga sequenza

$$Z = z_1 z_2 \ldots z_k$$

che è sottosequenza sia di X che di Y

Ricordiamo che una sottosequenza di una sequenza X è una qualsiasi sequenza ottenuta da X cancellando alcuni elementi.

Il problema della massima sottosequenza ha molte applicazioni.

Per citarne solo alcune:

- individuare le parti comuni di due versioni dello stesso file (sequenze di caratteri ASCII).

- valutare la similitudine tra due segmenti di DNA (sequenze di simboli A,C,G,T).

Passo 1: *Struttura di una massima sottosequenza comune* (*LCS*)

Sia $\boldsymbol{Z} = z_1 \ldots z_k$ una ***LCS*** di

$$\boldsymbol{X} = x_1 \ldots x_m \quad \text{e} \quad \boldsymbol{Y} = y_1 \ldots y_n$$

La sottostruttura ottima di $\boldsymbol{Z}$ discende dalle seguenti ***proprietà***:

1. se $x_m = y_n$ allora $z_k = x_m = y_n$ e $\boldsymbol{Z}_{k-1}$ è una ***LCS*** di $\boldsymbol{X}_{m-1}$ e $\boldsymbol{Y}_{n-1}$
2. altrimenti se $z_k \neq x_m$ allora $\boldsymbol{Z}$ è ***LCS*** di $\boldsymbol{X}_{m-1}$ e $\boldsymbol{Y}$
3. altrimenti $z_k \neq y_n$ e $\boldsymbol{Z}$ è una ***LCS*** di $\boldsymbol{X}$ e $\boldsymbol{Y}_{n-1}$

***<u>Dimostrazione</u>*:**

1. Supponiamo $x_m = y_n$
   Se $z_k \neq x_m = y_n$ potremmo aggiungere il simbolo $x_m = y_n$ in coda a Z ottenendo una sottosequenza comune più lunga contro l'ipotesi che Z sia una ***LCS***.
   Quindi $z_k = x_m = y_n$ e Z_{k-1} è sottosequenza comune di X_{m-1} e Y_{n-1} .
2. Se $z_k \neq x_m$ allora Z è sottosequenza di X_{m-1} e Y
   Essendo Z una ***LCS*** di X e Y essa è anche una ***LCS*** di X_{m-1} e Y.
3. il caso $z_k \neq y_n$ è simmetrico.

Data una sequenza

$$X = x_1 x_2 \ldots x_m$$

indicheremo con

$$X_i = x_1 x_2 \ldots x_i$$

il prefisso di X di lunghezza i.

L'insieme dei sottoproblemi è costituito quindi dalla ricerca delle ***LCS*** di tutte le coppie di prefissi (X_i, Y_j), per $i = 0,\ldots,m$ e $j = 0,\ldots,n$.

Totale $(m+1)(n+1) = \Theta(mn)$ sottoproblemi.

Passo 2: *soluzione ricorsiva*

Siano $X = x_1 \ldots x_m$ e $Y = y_1 \ldots y_n$ le due sequenze di cui vogliamo calcolare una ***LCS*** e per $i = 0,1,\ldots,m$ e $j = 0,1,\ldots,n$ sia $c_{i,j}$ la lunghezza di una ***LCS*** dei due prefissi X_i e Y_j .

Usando le proprietà che abbiamo appena dimostrato possiamo scrivere:

$$c_{i,j} = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ c_{i-1,j-1} + 1 & \text{se } i, j > 0 \text{ e } x_i = y_j \\ \max(c_{i,j-1}, c_{i-1,j}) & \text{se } i, j > 0 \text{ e } x_i \neq y_j \end{cases}$$

Passo 3

Esempio

X=ABCBDAB
Y=BDCABA

X \ Y		0	B 1	D 2	C 3	A 4	B 5	A 6
		0	1	2	3	4	5	6
	0	*c* 0 *s*	0	0	0	0	0	0
A	1	*c* 0 *s*	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖
B	2	*c* 0 *s*	1 ↖	1 ←	1 ←	1 ↑	2 ↖	2 ←
C	3	*c* 0 *s*	1 ↑	1 ↑	2 ↖	2 ←	2 ↑	2 ↑
B	4	*c* 0 *s*	1 ↖	1 ↑	2 ↑	2 ↑	3 ↖	3 ←
D	5	*c* 0 *s*	1 ↑	2 ↖	2 ↑	2 ↑	3 ↑	3 ↑
A	6	*c* 0 *s*	1 ↑	2 ↑	2 ↑	3 ↖	3 ↑	4 ↖
B	7	*c* 0 *s*	1 ↖	2 ↑	2 ↑	3 ↑	4 ↖	4 ↑

i ↓ *j* →

Terzo passo: *lunghezza di una LCS*

```
LCS-Length(X, Y, m, n)
    for i = 0 to m
        c[i, 0] = 0
    for j = 1 to n
        c[0, j] = 0
    for j = 1 to n
        for i = 1 to m
            if x_i == y_j
                c[i, j] = c[i-1, j-1]+1, s[i, j] = "↖"
            elseif c[i-1, j] ≥ c[i, j-1]
                c[i, j] = c[i-1, j], s[i, j] = "↑"
            else c[i, j] = c[i, j-1], s[i, j] = "←"
    return c,s
```

Quarto passo

Esempio

X=ABCBDAB
Y=BDCABA

LCS=BCBA

X \ Y		0	B 1	D 2	C 3	A 4	B 5	A 6
		0	0	0	0	0	0	0
	0	c 0 s						
A	1	c 0 s	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖
B	2	c 0 s	1 ↖	1 ←	1 ←	1 ↑	2 ↖	2 ←
C	3	c 0 s	1 ↑	1 ↑	2 ↖	2 ←	2 ↑	2 ↑
B	4	c 0 s	1 ↖	1 ↑	2 ↑	2 ↑	3 ↖	3 ←
D	5	c 0 s	1 ↑	2 ↖	2 ↑	2 ↑	3 ↑	3 ↑
A	6	c 0 s	1 ↑	2 ↑	2 ↑	3 ↖	3 ↑	4 ↖
B	7	c 0 s	1 ↖	2 ↑	2 ↑	3 ↑	4 ↖	4 ↑

j →

i ↓

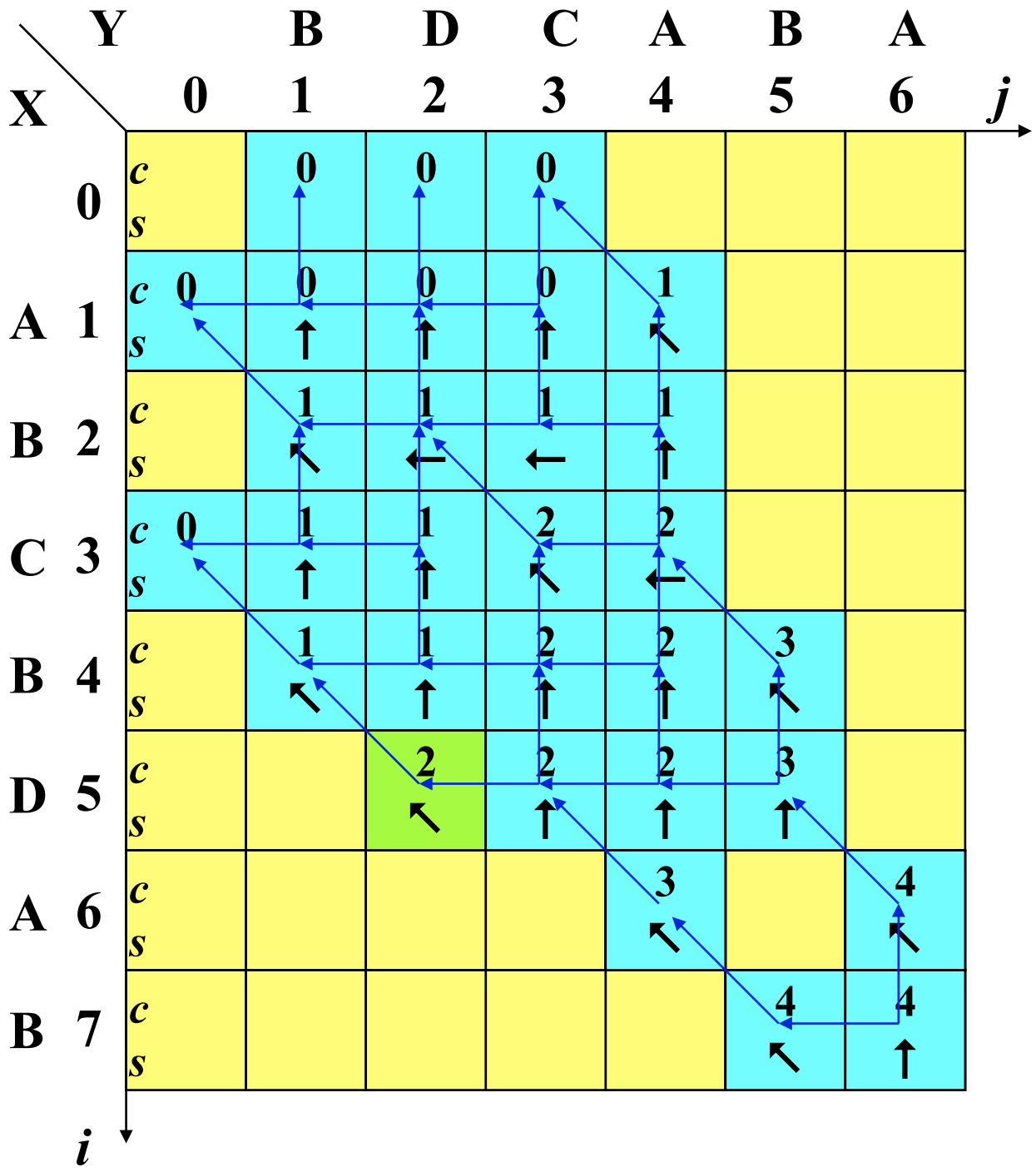
Quarto passo: *Stampa della LCS*

```
Print-LCS(X, s, i, j)
    if i > 0 and j > 0
        if s[i, j] == "↖"
            Print-LCS(X, s, i-1, j-1)
            print X[i]
        elseif s[i, j] == "↑"
            Print-LCS(X, s, i-1, j)
        else Print-LCS(X, s, i, j-1)
```

Metodo top-down

Esempio

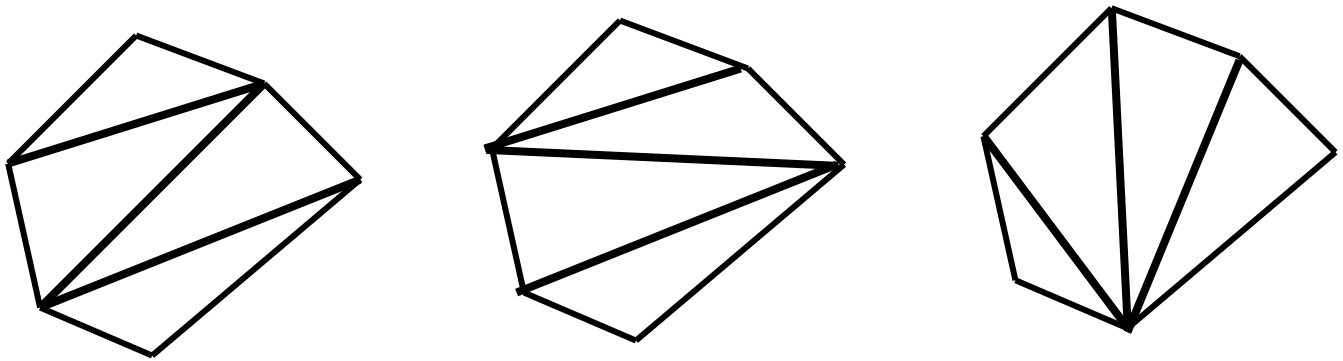
X=ABCBDAB
Y=BDCABA



Triangolazione ottima

Una triangolazione di un poligono convesso è una suddivisione del poligono in triangoli ottenuta tracciando delle diagonali che non si intersecano .

Vi sono più triangolazioni possibili dello stesso poligono



In questo problema sono dati i vertici $q_1, q_2, \ldots, q_n$ di un poligono convesso P presi in ordine antiorario.

Ad ogni triangolo T è attribuito un costo $c(T)$.

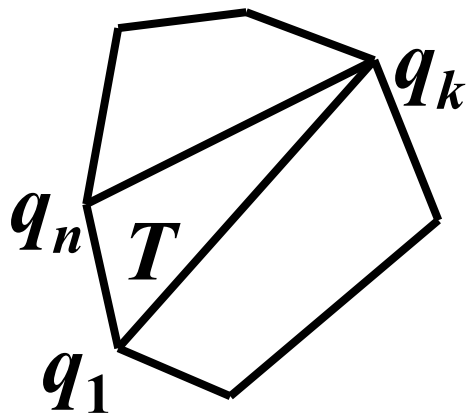
Ad esempio $c(T)$ potrebbe essere la lunghezza del perimetro, la somma delle altezze, il prodotto delle lunghezze dei lati, (l'area ?), ecc.

Si vuole trovare una triangolazione del poligono P tale che la somma dei costi dei triangoli sia minima.

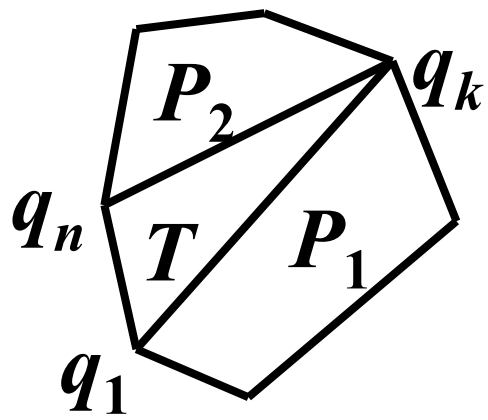
*In quanti modi possiamo suddividere in triangoli un poligono convesso di **n** vertici?*

Ogni lato del poligono ***P*** appartiene ad un solo triangolo della triangolazione .

Siano $q_1 q_k q_n$ i vertici del triangolo ***T*** a cui appartiene il lato $q_1 q_n$



Il triangolo T suddivide il poligono P nel triangolo T stesso e nei due poligoni P_1 e P_2 di vertici q_1 ,…,q_k e q_k,…,q_n



Il vertice q_k può essere scelto in n-2 modi diversi e i due poligoni hanno rispettivamente $n_1 = k$ ed $n_2 = n-k+1$ vertici.

P_1 quando $k = 2$ e P_2 quando $k = n-1$ sono poligoni degeneri, ossia sono un segmento.

Il numero $\boldsymbol{T(n)}$ di triangolazioni possibili di un poligono di $\boldsymbol{n}$ vertici si esprime ricorsivamente come segue

$$\boldsymbol{T(n)} = \begin{cases} 1 & \text{se } \boldsymbol{n} \leq 2 \\ \displaystyle\sum_{k=2}^{n-1} \boldsymbol{T(k)T(n-k+1)} & \text{se } \boldsymbol{n} > 2 \end{cases}$$

E' facile verificare che $\boldsymbol{T(n) = P(n\text{-}1)}$ dove $\boldsymbol{P(n)}$ sono le parentesizzazioni del prodotto di n matrici.

Quindi $\boldsymbol{T(n)}$ cresce esponenzialmente.

Primo passo: *struttura di una triangolazione ottima*.

Supponiamo che una triangolazione ottima suddivida il poligono convesso $\boldsymbol{P}$ di vertici $\boldsymbol{q_1 q_2 \ldots q_n}$ nel triangolo $\boldsymbol{T}$ di vertici $\boldsymbol{q_1 q_k q_n}$ e nei due poligoni $\boldsymbol{P_1}$ e $\boldsymbol{P_2}$ di vertici $\boldsymbol{q_1 \ldots q_k}$ e $\boldsymbol{q_k \ldots q_n}$ rispettivamente.

Le triangolazioni subordinate di $\boldsymbol{P_1}$ e di $\boldsymbol{P_2}$ sono triangolazioni ottime. Perché?

Secondo passo: *soluzione ricorsiva*

I sottoproblemi sono le triangolazioni dei poligoni $P_{i..j}$ di vertici $q_i \ldots q_j$. Sia $c_{i,j}$ la somma dei costi dei triangoli di una triangolazione ottima di $P_{i..j}$.

Se $j = i+1$ allora $P_{i..j}$ è degenere e $c_{i,j} = 0$.

Se $j > i+1$ allora $P_{i..j}$ si può scomporre in un triangolo T di vertici $q_i q_k q_j$ e nei due poligoni P_1 e P_2 di vertici $q_i \ldots q_k$ e $q_k \ldots q_j$ con $i < k < j$

$$c_{i,j} = \begin{cases} 0 & \text{se } j \leq i+1 \\ \min\limits_{i<k<j} \left(c_{i,k} + c_{k,j} + c(q_i q_k q_j) \right) & \text{se } j > i+1 \end{cases}$$

Terzo passo: *calcolo costo minimo*

```
Triangulation-Cost(q, n)
    for i = 1 to n-1
        c[i, i+1] = 0
    for j = 3 to n
        for i = j-2 downto 1
            c[i, j] = ∞
            for k = i+1 to j-1
                q = c[i, k]+c[k, j]+c(q_i q_k q_j)
                if q < c[i, j]
                    c[i, j] = q
                    s[i, j] = k
    return c,s
```

Complessità: $O(n^3)$

Quarto passo: *Stampa triangolazione*

```
Print-Triangulation(s, i, j)
    if j > i+1
        k = s[i, j]
        Print-Triangulation(s, i, k)
        print "triangolo:", i, j, k
        Print-Triangulation(s, k, j)
```

Complessità: $O(n)$