

# Algoritmi e Strutture Dati

## Elementi di Programmazione Dinamica

Maria Rita Di Berardini, Emanuela Merelli<sup>1</sup>

<sup>1</sup>Dipartimento di Matematica e Informatica  
Università di Camerino

19 novembre 2008

## Parte I

# Elementi di programmazione dinamica

# Elementi di programmazione dinamica

Abbiamo visto come è possibile applicare la tecnica di PD per risolvere il problema della moltiplicazione di una sequenza di matrici (algoritmo **MatrixChainOrder**)

Ma, da un punto di vista ingegneristico, quando dovremmo cercare di risolvere un problema con la PD?

Cerchiamo di rispondere a questa domanda esaminando i due ingredienti chiave che deve avere un problema di ottimizzazione affinché possa essere applicata la PD: la **sottostruttura ottima** e i **sottoproblemi ripetitivi**

# Elementi di programmazione dinamica: sottostruttura ottima

- un problema presenta una **sottostruttura ottima** se una soluzione ottima del problema contiene al suo interno soluzioni ottime di sottoproblemi
- se un problema presenta una sottostruttura ottima, allora è possibile costruire una soluzione ottima del problema combinando soluzioni ottime di sottoproblemi
- ciò potrebbe essere un buon indizio dell'applicabilità della PD

# Elementi di programmazione dinamica: sottoproblemi ripetuti

- lo spazio dei sottoproblemi da risolvere deve essere “**piccolo**” e la soluzione deve essere costruita risolvendo ripetutamente gli stessi sottoproblemi (senza generarne sempre di nuovi)
- Questo, più l'uso di una tabella per memorizzare soluzioni parziali, consente di ridurre il tempo di esecuzione
- **Obiettivo**: scrivere un algoritmo che possa essere eseguito in un **tempo polinomiale**

# Sottostruttura Ottima

Per capire se un dato problema ha una sottostruttura ottima si segue il seguente schema:

- 1 si suppone di conoscere la scelta che porta ad una soluzione ottima. **NB:** in questa fase non interessa sapere come tale scelta sia stata determinata; ci limitiamo a supporre di conoscerla  
assumiamo che la parentesizzazione ottima di  $A_i \cdots A_j$  suddivida il prodotto fra  $A_k$  e  $A_{k+1}$
- 2 determiniamo quali sottoproblemi considerare e come caratterizzare lo spazio risultante dei sottoproblemi.  
se  $k$  (con  $i \leq k < j$ ) caratterizza la soluzione ottima, risolviamo i due sottoproblemi  $A_i \cdots A_k$  e  $A_{k+1} \cdots A_j$
- 3 dimostrare, utilizzando la tecnica "taglia e incolla", che le soluzioni dei sottoproblemi utilizzati all'interno della soluzione ottima sono a loro volta ottime

# Sottostruttura Ottima

La sottostruttura ottima varia a seconda del problema in due modi

- 1 per il numero di sottoproblemi utilizzati all'interno di una soluzione ottima del problema originale  
la suddivisione ottima del prodotto  $A_i \dots j$  contiene due sottoproblemi
- 2 per il numero di possibili scelte per determinare quale sottoproblema (o quali sottoproblemi) utilizzare in una soluzione ottima.  
l'indice  $k$  che determina la suddivisione ottima, varia nell'intervallo  $i \leq k < j$  e può essere scelto fra  $j - i$  candidati

# Calcolo del valore della soluzione ottima

Una volta verificata la sottostruttura ottima del problema:

1. si identifica il numero di sottoproblemi da risolvere

risolvere il problema del prodotto di una sequenza di matrici, significa risolvere un numero di sottoproblemi dell'ordine di  $\Theta(n^2)$ . Perché?

Possibili sottoproblemi: tutti quelli della forma  $A_{i..j}$  con  $i = 1, \dots, n$  e  $j = i, \dots, n$ . Fissato  $i$  dobbiamo risolvere  $n - i + 1$  sottoproblemi. In totale

$$\sum_{i=1}^n n - i + 1 = \sum_{j=1}^n j = \frac{n(n+1)}{2} = \Theta(n^2)$$

2. se tale numero è "ragionevole", forniamo una soluzione ricorsiva che calcola il valore della soluzione ottima

definiamo ricorsivamente il costo del prodotto  $A_{i..j}$  tramite la funzione  $m[i, j]$



## Calcolo del valore della soluzione ottima

- la soluzione ricorsiva viene successivamente implementata mediante uno schema bottom-up (dal basso verso) l'alto: si costruisce la soluzione del problema originario a partire dalle soluzioni dei sottoproblemi

Il costo della soluzione del problema è pari al costo della soluzione dei sottoproblemi più il costo imputabile alla scelta effettuata

se  $i < j$ :

$$m[i, j] = \min_{i \leq k < j} \left\{ m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j \right\}$$

Tipicamente, in questa fase vengono raccolte tutte le informazioni utili per costruire la soluzione ottima

**Matrix-Chain-Order** memorizza tali informazioni in  $m[i, j]$

# Tempo di esecuzione

Informalmente, il tempo di esecuzione di un algoritmo di programmazione dinamica dipende dal prodotto di due fattori:

- 1 il numero dei sottoproblemi
- 2 il numero di scelte da considerare per ogni sottoproblema

Nel caso della moltiplicazione di una sequenza di  $n$  matrici;

- 1 il numero complessivo dei sottoproblemi è  $\Theta(n^2)$
- 2 e per ognuno di essi avevamo al massimo  $n - 1$  scelte

La complessità dell'algoritmo è  $\Theta(n^3)$

# Ruolo della sottostruttura ottima

Bisogna fare molta attenzione a non assumere la sottostruttura ottima quando non è possibile farlo

Sia  $G = (V, A)$  un grafo orientato ed  $u, v \in V$  due nodi in  $G$

**Cammino minimo in un grafo non pesato:** trovare un cammino semplice (senza cicli) da  $u$  a  $v$  con minimo numero di archi

**Cammino massimo in un grafo non pesato:** trovare un cammino semplice (senza cicli) da  $u$  a  $v$  con massimo numero di archi

Questi due problemi sembrano molto simili, ma **solo** il primo presenta una sottostruttura ottima

# Cammino minimo e sottostruttura ottima

Sia  $p$  un cammino  $u \rightsquigarrow^p v$  dal vertice  $u$  al vertice  $v$  con minimo numero di archi

Se  $u \neq v$  (cioè se il problema non è banale), allora deve esistere un nodo intermedio  $w$  ( $w$  può anche essere  $u$  o  $v$ ) tale che il cammino  $u \rightsquigarrow^p v$  può essere decomposto nei cammini  $u \rightsquigarrow^{p_1} w$  e  $w \rightsquigarrow^{p_2} v$

Chiaramente il  $\#_{\text{archi}}(p) = \#_{\text{archi}}(p_1) + \#_{\text{archi}}(p_2)$

**Sottostruttura ottima:** se  $p$  è un cammino minimo da  $u$  a  $v$ , allora  $p_1$  e  $p_2$  sono cammini minimi da  $u$  a  $w$  e da  $w$  a  $v$  rispettivamente

Di nuovo, dimostriamo la sottostruttura ottima usando la tecnica taglia e incolla

# Cammino minimo e sottostruttura ottima

Assumiamo, per assurdo, che il cammino  $p_1$  non sia minimo

Deve esistere un cammino alternativo  $u \rightsquigarrow w$  t.c.  $\#_{\text{archi}}(p'_1) < \#_{\text{archi}}(p_1)$

Sostituendo in  $p$   $p_1$  con  $p'_1$  otteniamo un cammino da  $u$  a  $v$  con un numero di archi pari a

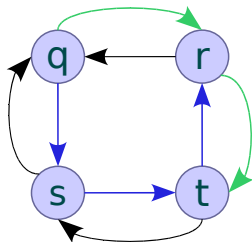
$$\#_{\text{archi}}(p'_1) + \#_{\text{archi}}(p_2) < \#_{\text{archi}}(p_1) + \#_{\text{archi}}(p_2) = \#_{\text{archi}}(p)$$

con  $p$  cammino minimo – contraddizione

In maniera analoga possiamo dimostrare che anche il cammino  $p_2$  ha un numero di archi minimo

Saremmo tentati di supporre che anche il problema del cammino semplice massimo presenti una sottostruttura ottima

# Cammino massimo e sottostruttura ottima

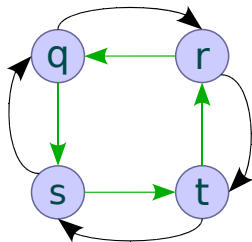


Un cammino semplice massimo tra  $q$  ed  $t$  è  $q \rightarrow r \rightarrow t$ , ma:

- $q \rightarrow r$  non è un cammino semplice massimo tra  $q$  e  $r$  (esiste un cammino semplice tra i due nodi con numero maggiore di archi,  $q \rightarrow s \rightarrow t \rightarrow r$ )
- in maniera analoga  $r \rightarrow t$  non è un cammino semplice massimo tra  $r$  e  $t$  ( $r \rightarrow q \rightarrow s \rightarrow t$ )

# Cammino massimo e sottostruttura ottima

Inoltre ...

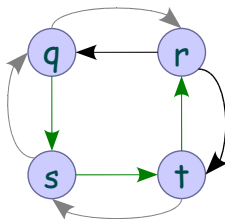


assemblando soluzioni ottime di sotto problemi, ad es.  $q \rightarrow s \rightarrow t \rightarrow r$  e  $r \rightarrow q \rightarrow s \rightarrow t$ , non otteniamo una soluzione valida del problema

La combinazione delle due soluzioni da luogo ad un ciclo ed il cammino non è un cammino semplice

# Cammino massimo e sottostruttura ottima

I sottoproblemi non sono **indipendenti**: la soluzione di un sottoproblema può influire sulla soluzione di un altro sottoproblema



L'aver usato i vertici  $s$  e  $t$  per costruire il cammino massimo tra  $q$  ed  $r$ , ci impedisce di usare questi vertici per costruire il cammino massimo tra  $r$  e  $t$  (la combinazione delle due soluzioni produrrebbe un cammino non semplice) e quindi di risolvere il problema



# Cammino massimo e sottostruttura ottima

Non è stato ancora trovato un algoritmo di programmazione dinamica in grado di risolvere il problema del cammino massimo in un grafo non pesato

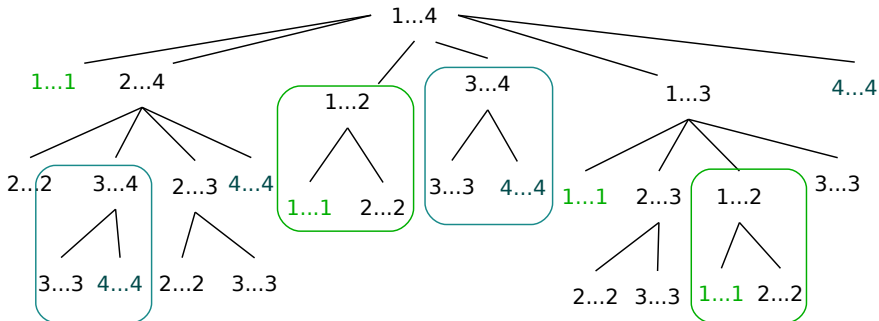
In effetti, questo problema è NP-completo ossia è improbabile che possa essere risolto in un tempo polinomiale

# Sottoproblemi ripetuti ed efficienza

## Recursive-Matrix-Chain-Order( $p, i, j$ )

1. **for**  $i = j$
2.     **then return** 0
3.  $m[i, j] \leftarrow \infty$
4. **for**  $k \leftarrow i$  **to**  $j - 1$
5.     **do**  $q \leftarrow$  **Recursive-Matrix-Chain-Order**( $p, i, k$ )  
       + **Recursive-Matrix-Chain-Order**( $p, k + 1, j$ )  
       +  $p_{i-1}p_kp_j$
6.     **if**  $q < m[i, j]$
7.         **then**  $m[i, j] \leftarrow q$
8. **return**  $m[i, j]$

## Sottoproblemi ripetuti ed efficienza



# Sottoproblemi ripetuti ed efficienza

Poichè eseguire le righe 1-2 e 6-7 richiede almeno un'unità di tempo, abbiamo

$$T(1) \geq 1$$

$$T(n) \geq 1 + \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1) \quad \text{per } n \geq 1$$

Inoltre:

$$\begin{aligned}
 & 1 + \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1) \\
 & 1 + \sum_{k=1}^{n-1} T(k) + \underbrace{\sum_{j=1}^{n-1} T(j)}_{\sum_{k=1}^{n-1} T(n-k)} + \sum_{k=1}^{n-1} 1 = \\
 & 1 + 2 \sum_{k=1}^{n-1} T(k) + (n-1) = 2 \sum_{k=1}^{n-1} T(k) + n
 \end{aligned}$$

# Sottoproblemi ripetuti ed efficienza

Possiamo riscrivere la ricorrenza come  $T(n) \geq 2 \sum_{k=1}^{n-1} T(k) + n$ . Dimostriamo per induzione che  $T(n) \geq 2^{n-1}$  per ogni  $n \geq 1$  e, quindi, che  $T(n) = \Omega(2^n)$

**Caso base:**  $T(1) \geq 1 = 2^0$

**Passo induttivo:** Assumiamo  $n \geq 2$

$$\begin{aligned}
 T(n) &\geq 2 \sum_{k=1}^{n-1} T(k) + n && \text{per ip. induttiva} \\
 &\geq 2 \sum_{k=1}^{n-1} 2^{k-1} + n && \text{ponendo } i = k - 1 \\
 &= 2 \sum_{i=0}^{n-2} 2^i + n = 2 \frac{(2^{n-1} - 1)}{2 - 1} + n \\
 &= 2^n - 2 + n && \text{poichè } n \geq 2 \\
 &\geq 2^n \\
 &\geq 2^{n-1}
 \end{aligned}$$

## Parte II

# La più lunga sottosequenza comune

## Definizione del problema

Nelle applicazioni biologiche spesso si confronta il DNA di due, o più, organismi differenti

La struttura del DNA è formata da una stringa di molecole dette **basi**: adenina, citosina, guanina, e timina

La struttura del DNA è rappresentata da una stringa sull'insieme finito  $\{A, C, G, T\}$

Il DNA di due organismi

$$S_1 = ACCGGTCGCGCGGAAGCCGGCCGAA$$

$$S_2 = GTCGTTCTCGGAATGCCGTTGCTCTGTAAA$$

## Definizione del problema

Uno degli scopi del confronto tra due molecole di DNA è quello di determinare il grado di somiglianza delle due molecole

Potremmo dire che due molecole si somigliano se, date le stringhe  $S_1$  e  $S_2$  che rappresentano le molecole:

- una è sottostringa di un'altra
- il numero delle modifiche richieste per trasformare l'una nell'altra è piccolo
- trovare una terza stringa  $S_3$  le cui basi si trovano in ciascuna delle stringhe  $S_1$  ed  $S_2$ : le basi devono presentarsi nello stesso ordine, senza essere necessariamente consecutive



## Definizione del problema

Ad esempio, date le stringhe:

$$S_1 = \text{ACCG} \color{red}{\text{GTCG}} \color{black}{\text{AGTG}} \color{blue}{\text{CGCGG}} \color{red}{\text{AAGC}} \color{blue}{\text{CGGC}} \color{black}{\text{CGAA}}$$
$$S_2 = \color{red}{\text{GTCG}} \color{black}{\text{TTCGGAAT}} \color{blue}{\text{GCGTTGCTCTG}} \color{black}{\text{TAAA}}$$

la stringa  $S_3$  è:

$$S_3 = \color{red}{\text{GTCG}} \color{blue}{\text{TCCGGAAGC}} \color{black}{\text{CGGC}} \color{red}{\text{CGAA}}$$

Formalizziamo questo concetto

## Definizione del problema

Siano  $X = \langle x_1, x_2, \dots, x_m \rangle$  e  $Z = \langle z_1, z_2, \dots, z_k \rangle$  due sequenze; diciamo che  $Z$  è una **sottosequenza** di  $X$  se esiste una sequenza crescente  $i_1, i_2, \dots, i_k$  di indici di  $X$  tali che  $x_{i_j} = z_j$  per ogni  $j = 1, 2, \dots, k$

Ad esempio, se  $X = \langle A, B, C, B, D, A, B \rangle$  e  $Z = \langle B, C, D, B \rangle$  la sequenza di indici è:  $\langle 2, 3, 5, 7 \rangle$

$$x_{i_1} = x_2 = B, x_{i_2} = x_3 = C, x_{i_3} = x_5 = D \text{ e } x_{i_4} = x_7 = B$$

## Definizione del problema

Date due sequenze  $X$  e  $Y$  diciamo che  $Z$  è una **sottosequenza comune** di  $X$  e  $Y$  se è una sottosequenza sia di  $X$  che di  $Y$

Ad esempio, se  $X = \langle A, B, C, B, D, A, B \rangle$  e  $y = \langle B, D, C, A, B, A \rangle$  la sequenza  $\langle B, C, A \rangle$  è una sottosequenza di  $X$  e  $Y$ , ma non è la più lunga sottosequenza comune (Longest Common Subsequence, LCS)

Sottosequenze comuni più lunghe:  $\langle B, C, B, A \rangle$  e  $\langle B, D, A, B \rangle$

**Problema:** Siano  $X = \langle x_1, x_2, \dots, x_m \rangle$  ed  $Y = \langle y_1, y_2, \dots, y_n \rangle$  due sequenze; trovare una sottosequenza di lunghezza massima che è comune sia a  $X$  che a  $Y$

## Fase 1: struttura della soluzione ottima

In questo caso, applicare la “forza bruta” significa considerare tutte le possibili sottosequenze di  $X$  e confrontarle con tutte le sottosequenze di  $Y$

Quante sono le possibili sottosequenze di  $X = \langle x_1, x_2, \dots, x_m \rangle$ ?

Ogni sottosequenza di  $X$  corrisponde ad un sottoinsieme degli indici  $\{1, 2, \dots, m\}$  e tutti i possibili sottoinsiemi di  $\{1, 2, \dots, m\}$  sono  $2^m$

Il **solo** enumerare tutte le possibili sottosequenze di  $X$  richiede un tempo esponenziale

Di nuovo, la tecnica della forza bruta non è adeguata

## Fase 1: struttura della soluzione ottima

Nel seguito definiremo classi di sottoproblemi corrispondenti a coppie di prefissi delle sequenze in input

Sia  $X = \langle x_1, x_2, \dots, x_m \rangle$  e  $i = 1, \dots, m$ ; l' **$i$ -esimo prefisso** di  $X$  è definito come  $X_i = \langle x_1, x_2, \dots, x_i \rangle$  (con  $X_m = X$  e  $X_0 = \langle \rangle$ , i.e. la sequenza vuota)

Siano  $X = \langle x_1, x_2, \dots, x_m \rangle$  e  $Y = \langle y_1, y_2, \dots, y_n \rangle$  le due sequenze input e  $Z = \langle z_1, z_2, \dots, z_k \rangle$  una LCS di  $X$  e  $Y$ . Quale è la struttura di  $Z$ ??

**Caso banale:** almeno una delle due sequenze è vuota (ossia,  $X = \langle \rangle$  or  $Y = \langle \rangle$ ). In questo caso,  $X$  e  $Y$  possono avere una sola sottosequenza comune, la sottosequenza vuota e quindi  $Z = \langle \rangle$

# Fase 1: struttura della soluzione ottima

**Caso non banale:**  $X$  e  $Y$  sono entrambe non vuote; in questo caso usiamo il seguente teorema

**Teorema 1** (*sottostruttura ottima di LCS*): siano  $X = \langle x_1, x_2, \dots, x_m \rangle$ ,  $Y = \langle y_1, y_2, \dots, y_n \rangle$  e  $Z = \langle z_1, z_2, \dots, z_k \rangle$  una qualsiasi LCS di  $X$  e  $Y$ . Allora:

- 1 se  $x_m = y_n$ , allora  $z_k = x_m = y_n$  e  $Z_{k-1}$  è una LCS di  $X_{m-1}$  e  $Y_{n-1}$
- 2 se  $x_m \neq y_n$  e  $z_k \neq x_m$  allora  $Z$  è una LCS di  $X_{m-1}$  e  $Y$
- 3 se  $x_m \neq y_n$  e  $z_k \neq y_n$  allora  $Z$  è una LCS di  $X$  e  $Y_{n-1}$

## L'idea è la seguente

Per trovare una LCS di  $X = \langle x_1, x_2, \dots, x_m \rangle$  e  $Y = \langle y_1, y_2, \dots, y_n \rangle$  distinguiamo due casi:

- 1  $x_m = y_n$ : troviamo una LCS di  $X_{m-1}$  e  $Y_{n-1}$  e accodiamo  $x_m$  come ultimo simbolo
- 2  $x_m \neq y_n$ :
  - troviamo una LCS di  $X_{m-1}$  e  $Y$
  - troviamo una LCS di  $X$  e  $Y_{n-1}$

La più lunga di queste sequenze è una LCS di  $X$  e  $Y$

la proprietà della sottostruttura ottima è verificata perchè una soluzione ottima contiene al suo interno soluzioni ottime di sottoproblemi

## L'idea è la seguente

La sottostruttura ottima ci consente di risolvere il problema originario, identificare una LCS di  $X = \langle x_1, x_2, \dots, x_m \rangle$  e  $Y = \langle y_1, y_2, \dots, y_n \rangle$ , cercando LCS di coppie di prefissi di  $X$  e  $Y$

Quanti sottoproblemi dobbiamo risolvere? Tutti quelli della forma:

*trovare una LCS di  $X_i$  e  $Y_j$  con  $i = 1 \dots m$  e  $j = 1 \dots n$*

In totale sono  $nm$  sottoproblemi distinti



## Fase 2: una soluzione ricorsiva

Definiamo  $c[i, j]$  come **la lunghezza di una qualsiasi LCS** tra  $X_i$  e  $Y_j$ :

- se  $i = 0$  o  $j = 0$  (almeno una delle due è vuota)  $c[i, j] = 0$
- se  $i, j > 0$  e  $x_i = y_j$ ,  $c[i, j] = c[i - 1, j - 1] + 1$
- se  $i, j > 0$  e  $x_i \neq y_j$ ,  $c[i, j] = \max(c[i - 1, j], c[i, j - 1])$

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

## Fase 3: calcolare la lunghezza di una LCS

**Esempio:** calcolare la lunghezza di una LCS di  $X = \langle A, B, C, B, D, A, B \rangle$   
e  $Y = \langle B, D, C, A, B, A \rangle$

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0						
B	2	0						
C	3	0						
B	4	0						
D	5	0						
A	6	0						
B	7	0						

**Casi Base:**  $c[i, 0] = c[0, j] = 0$  per ogni  $i = 1, \dots, m$  e  $j = 1, \dots, n$

## Fase 3: calcolare la lunghezza di una LCS

Se  $i, j > 0$  di quali posizioni abbiamo bisogno per calcolare  $c[i, j]$

- se  $x_i = y_j$ , abbiamo bisogno di  $c[i - 1, j - 1]$  (posizione in alto e a sinistra)
- se  $x_i \neq y_j$ , abbiamo bisogno di  $c[i - 1, j]$  (in alto) e  $c[i, j - 1]$  (a sinistra)

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
A	1	0						
B	2	0						
C	3	0		↖	↑			
B	4	0		←	$i, j$			
D	5	0						
A	6	0						
B	7	0						

Ci basta riempire la matrice riga per riga

# Fase 3: calcolare la lunghezza di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0						

# Fase 3: calcolare la lunghezza di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0					

# Fase 3: calcolare la lunghezza di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0	0				

## Fase 3: calcolare la lunghezza di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0	0	0			

## Fase 3: calcolare la lunghezza di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0	0	0	1		



## Fase 3: calcolare la lunghezza di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0	0	0	1	1	

# Fase 3: calcolare la lunghezza di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0	0	0	1	1	1

## Fase 3: calcolare la lunghezza di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0						

## Fase 3: calcolare la lunghezza di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1					

## Fase 3: calcolare la lunghezza di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1				

## Fase 3: calcolare la lunghezza di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1			

## Fase 3: calcolare la lunghezza di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1		

## Fase 3: calcolare la lunghezza di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	



## Fase 3: calcolare la lunghezza di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2

## Fase 3: calcolare la lunghezza di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0						

## Fase 3: calcolare la lunghezza di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1					

## Fase 3: calcolare la lunghezza di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1				

## Fase 3: calcolare la lunghezza di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2			

## Fase 3: calcolare la lunghezza di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2		

## Fase 3: calcolare la lunghezza di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	

## Fase 3: calcolare la lunghezza di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2



## Fase 3: calcolare la lunghezza di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0	1	2	2	3	3	4
B	7	0	1	2	2	3	4	4

La lunghezza di una LCS tra  $X$  e  $Y$  è data da  $c[n, m]$

# L' algoritmo per il calcolo della lunghezza di una LCS

**LcsLength**( $X, Y$ )

```

1.  $m \leftarrow \text{length}[X]$ 
2.  $n \leftarrow \text{length}[Y]$ 
3. for  $i \leftarrow 1$  to  $m$  do  $c[i, 0] \leftarrow 0$ 
4. for  $j \leftarrow 1$  to  $n$  do  $c[0, j] \leftarrow 0$ 
5. for  $i \leftarrow 1$  to  $m$ 
6.     do for  $j \leftarrow 1$  to  $n$ 
7.         do if  $x_i = y_j$ 
8.             then  $c[i, j] \leftarrow c[i - 1, j - 1] + 1$ 
9.             else if  $c[i - 1, j] \geq c[i, j - 1]$ 
10.                 then  $c[i, j] \leftarrow c[i - 1, j]$ 
11.                 else  $c[i, j] \leftarrow c[i, j - 1]$ 
12. return  $c$ 

```





Il calcolo di ogni posizione della tabella richiede un tempo costante; allora il costo di esecuzione della procedura è  $O(nm)$

## Fase 4: costruzione di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0	1	2	2	3	3	4
B	7	0	1	2	2	3	4	4

Possiamo riesaminare le scelte fatte per calcolare la lunghezza di una LCS al contrario, ossia dall'ultima alla prima, per costruire una soluzione ottima

## Fase 4: costruzione di una LCS

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0		0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	 1	← 1	1	1	2	2
C	3	0	1	1	 2	← 2	2	2
B	4	0	1	1	2	2	 3	3
D	5	0	1	2	2	2	↑ 3	3
A	6	0	1	2	2	3	3	 4
B	7	0	1	2	2	3	4	↑ 4

# L' algoritmo per il calcolo della lunghezza di una LCS

**LcsLength**( $X, Y$ )

1.  $m \leftarrow \text{length}[X]$
2.  $n \leftarrow \text{length}[Y]$
3. **for**  $i \leftarrow 1$  **to**  $m$  **do**  $c[i, 0] \leftarrow 0$
4. **for**  $j \leftarrow 1$  **to**  $n$  **do**  $c[0, j] \leftarrow 0$
5. **for**  $i \leftarrow 1$  **to**  $m$
6.     **do for**  $j \leftarrow 1$  **to**  $n$
7.         **do if**  $x_i = y_j$
8.             **then**  $c[i, j] \leftarrow c[i - 1, j - 1] + 1$
9.                  $b[i, j] \leftarrow "\swarrow"$
10.             **else if**  $c[i - 1, j] \geq c[i, j - 1]$
11.                 **then**  $c[i, j] \leftarrow c[i - 1, j]$
12.                      $b[i, j] \leftarrow "\uparrow"$
13.             **else**  $c[i, j] \leftarrow c[i, j - 1]$
14.                  $b[i, j] \leftarrow "\leftarrow"$
15. **return**  $b$  e  $c$

# L'algoritmo per la costruzione di una LCS

**PrintLCS**( $b, X, i, j$ )

1. **if**  $i = 0$  **or**  $j = 0$  **then return**
2. **if**  $b[i, j] = "\diagdown"$
3.     **then PrintLCS**( $b, X, i - 1, j - 1$ )
4.         stampa  $x_i$
5.     **else if**  $b[i, j] = "\uparrow"$
6.         **then PrintLCS**( $b, X, i - 1, j$ )
7.         **else PrintLCS**( $b, X, i, j - 1$ )

La chiamata iniziale è **PrintLCS**( $b, X, \text{length}[X], \text{length}[Y]$ )

La procedura impiega  $O(n + m)$ , poichè almeno una delle due dimensioni coinvolte diminuisce ad ogni passo della ricorsione