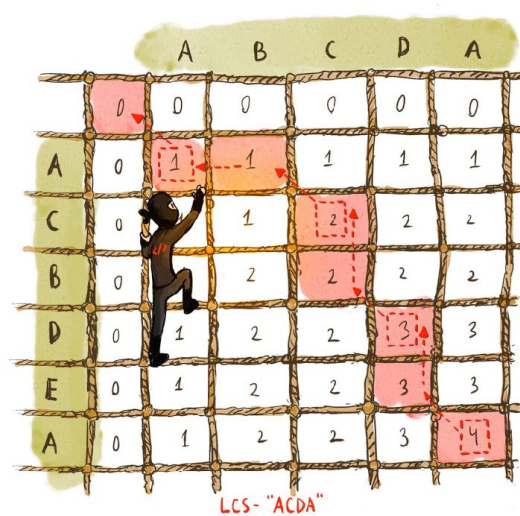


Programmazione dinamica

Giuseppe Prencipe
giuseppe.prencipe@unipi.it

Paradigma della programmazione dinamica

- Altro **paradigma fondamentale**, come il divide et impera
- Calcolo efficiente di una funzione ricorsiva mediante la **memorizzazione** dei suoi **valori intermedi** in una **tabella** (detta di *programmazione dinamica*)



Programmazione dinamica: Fibonacci

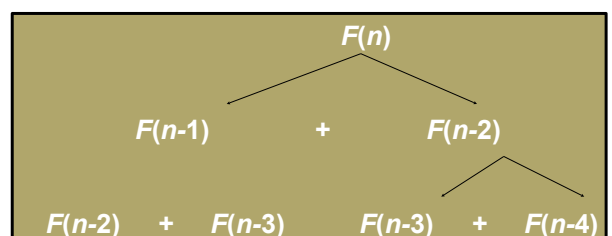
Esempio: numeri di Fibonacci (0, 1, 1, 2, 3, 5, 8, 13,...)

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ F_n = F_{n-1} + F_{n-2} \quad \text{per } n \geq 2 \end{cases}$$

@2020
giuseppe.precupce@unipi.it

Soluzione ricorsiva

```
Fib( n ) {           // n >= 0
    if (n <= 1)
        RETURN n;
    else
        RETURN Fib(n-1) + Fib(n-2);
}
```



- Approccio **top-down**

- $\text{Fib}(n)$ richiede $O(F_n) = O(\phi^n)$ passi,

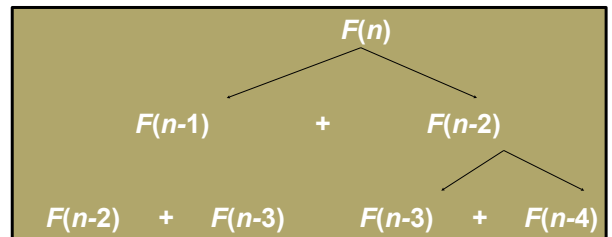
dove $\phi = (1 + \sqrt{5})/2 = 1,6180339\dots$ è il **rapporto aureo**

@2020
giuseppe.precupce@unipi.it

Soluzione ricorsiva

```
Fib( n ) {           // n >= 0
    if (n <= 1)
        RETURN n;
    else
        RETURN Fib(n-1) + Fib(n-2);
}
```

Versione più efficiente?



• Approccio **top-down**

- $Fib(n)$ richiede $O(F_n) = O(\phi^n)$ passi,

dove $\phi = (1 + \sqrt{5})/2 = 1,6180339\dots$ è il **rapporto aureo**

@2020
giuseppe.precipec@unipi.it

Fibonacci con programmazione dinamica

• Approccio **bottom-up**

- $F(0) = 0$
- $F(1) = 1$
- $F(2) = 1 + 0 = 1$
-
- $F(n-2) =$
- $F(n-1) =$
- $F(n) = F(n-1) + F(n-2)$

0	1	1	...	$F(n-2)$	$F(n-1)$	$F(n)$
---	---	---	-----	----------	----------	--------

@2020
giuseppe.precipec@unipi.it

Fibonacci con programmazione dinamica

```
Fib( n ) {           // n >= 0
    F[0] = 0;
    F[1] = 1;
    for (k = 2; k <= n; k = k + 1)
        F[k] = F[k-1] + F[k-2];
    RETURN F[n];
}
```

- L'array F è la **tabella di programmazione dinamica**
- **Idea di base:** F[k] si calcola utilizzando valori di F già calcolati

@2020
giuseppe.prencipe@unipi.it

Fibonacci con programmazione dinamica

```
Fib( n ) {           // n >= 0
    F[0] = 0;
    F[1] = 1;
    for (k = 2; k <= n; k = k + 1)
        F[k] = F[k-1] + F[k-2];
    RETURN F[n];
}
```

- Costo:
- Tempo – ????
- Spazio – ????

- L'array F è la **tabella di programmazione dinamica**
- **Idea di base:** F[k] si calcola utilizzando valori di F già calcolati

@2020
giuseppe.prencipe@unipi.it

Fibonacci con programmazione dinamica

```
Fib( n ) {           // n >= 0
    F[0] = 0;
    F[1] = 1;
    for (k = 2; k <= n; k = k + 1)
        F[k] = F[k-1] + F[k-2];
    RETURN F[n];
}
```

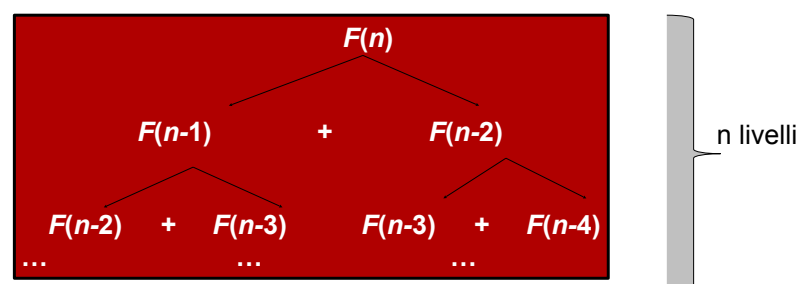
- Costo:
- Tempo – $O(n)$
- Spazio – $O(n)$

- L'array F è la **tabella di programmazione dinamica**
- **Idea di base:** F[k] si calcola utilizzando valori di F già calcolati

@2020
giuseppe.prencipe@unipi.it

Fibonacci: programmazione dinamica vs ricorsione

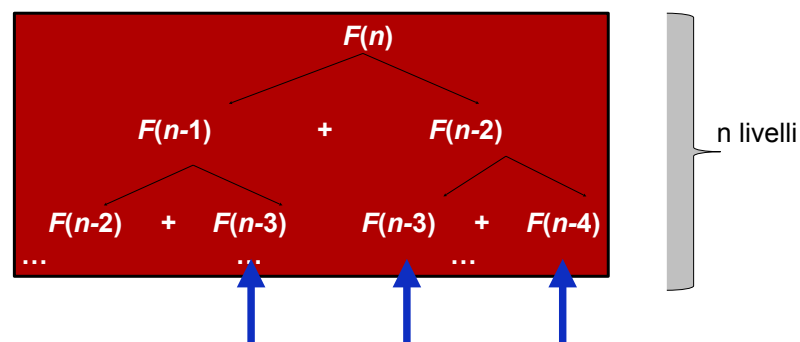
- L'approccio bottom-up è $\Theta(n)$ → **efficiente**
- Perché invece quello top-down è così **inefficiente**?



@2020
giuseppe.prencipe@unipi.it

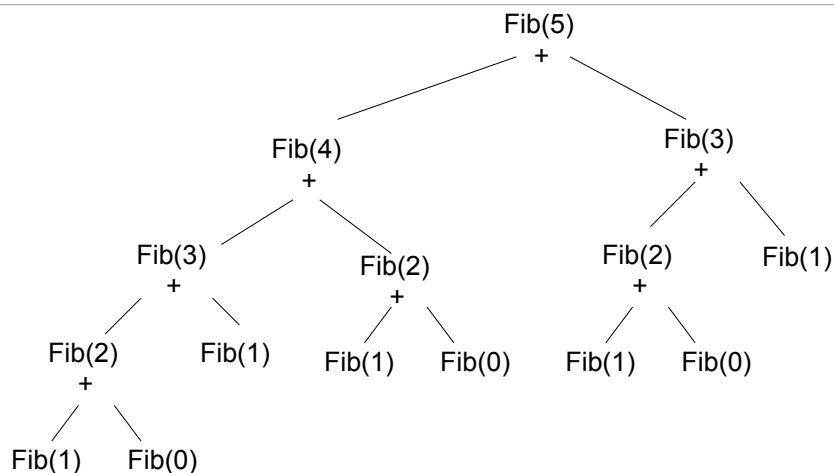
Fibonacci: programmazione dinamica **vs** ricorsione

- L'approccio bottom-up è $\Theta(n)$ → **efficiente**
- Perché invece quello top-down è così **inefficiente**?
 - Calcola **più volte** soluzioni dei sotto-problemi



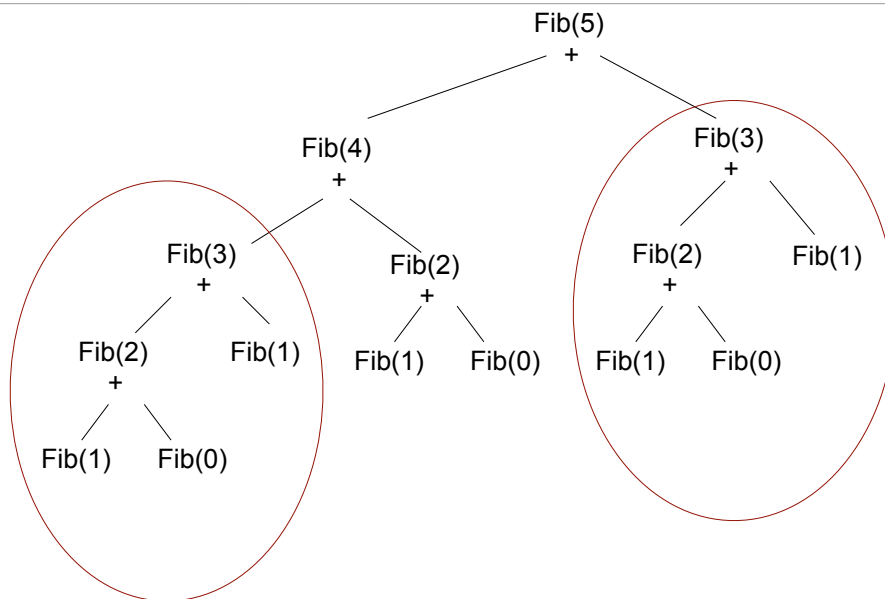
@2020
giuseppe.precipec@unipi.it

Fibonacci: programmazione dinamica **vs** ricorsione



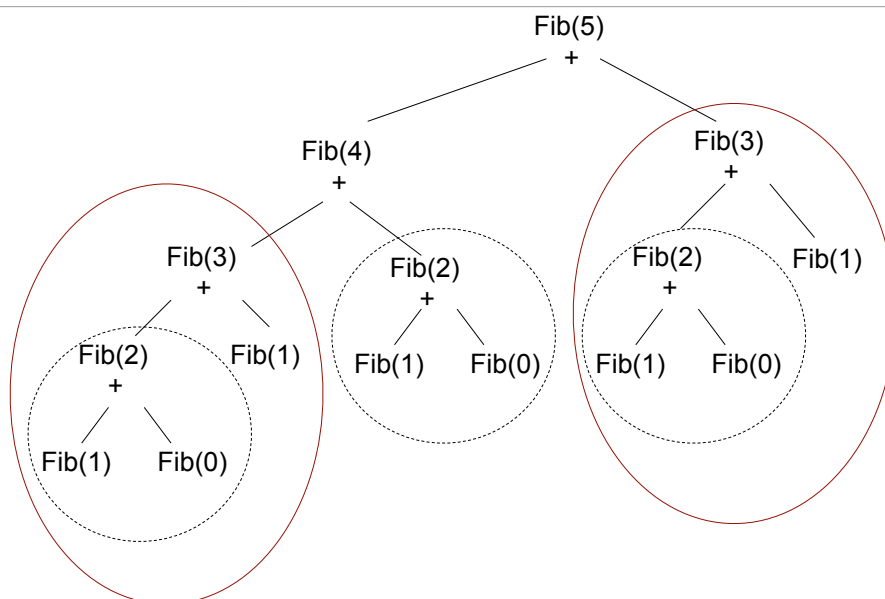
@2020
giuseppe.precipec@unipi.it

Fibonacci: programmazione dinamica **vs** ricorsione



@2020
giuseppe.prenepice@unipi.it

Fibonacci: programmazione dinamica **vs** ricorsione



@2020
giuseppe.prenepice@unipi.it

Programmazione Dinamica

- La **Programmazione Dinamica** è una tecnica di progettazione di algoritmi utilizzata con **problemi di ottimizzazione**
- Come il *divide et impera*, risolve problemi **combinando soluzioni di sotto-problemi**
- A differenza del *divide et impera*, i **sotto-problemi non sono indipendenti**
 - Sotto-problemi (tipicamente) condividono sotto-problemi

@2020
giuseppe.prencipe@unipi.it

Programmazione Dinamica vs Divide et impera

- | | |
|--|---|
| <ul style="list-style-type: none">• Divide et impera<ul style="list-style-type: none">– Partiziona il problema in sotto-problemi indipendenti– Risolvi i sotto-problemi ricorsivamente– Combina le soluzioni per risolvere il problema originale | <ul style="list-style-type: none">• Programmazione Dinamica<ul style="list-style-type: none">– Partiziona il problema in sotto-problemi (non indipendenti)– Risolvi i sotto-problemi ricorsivamente– Combina le soluzioni per risolvere il problema originale |
|--|---|

@2020
giuseppe.prencipe@unipi.it

Programmazione Dinamica

- Il termine **Dynamic Programming** viene dalla *Teoria dei Controlli*, non dall'Informatica
 - Termine introdotto negli anni '50 da Richard Bellman, che sviluppò metodi di **Programmazione** logistica per l'aeronautica.
- **Dinamica** si riferisce al fatto che la tabella è riempita progressivamente
- **Programming** si riferisce all'utilizzo di tabelle (array) per costruire una soluzione

Programmazione Dinamica

Programmazione Dinamica

- Nella programmazione dinamica tipicamente si **riduce il tempo** **aumentando** l'utilizzo di **spazio**



@2020
giuseppe.prencepi@unipi.it

Programmazione Dinamica

- Nella programmazione dinamica tipicamente si **riduce il tempo** **aumentando** l'utilizzo di **spazio**
- Risolviamo il problema risolvendo sotto-problemi di dimensione crescente e **memorizzando** ognuna delle sotto-soluzioni (ottimali) **in una tabella** (tipicamente)



@2020
giuseppe.prencepi@unipi.it

Programmazione Dinamica

- Nella programmazione dinamica tipicamente si **riduce il tempo** **aumentando** l'utilizzo di **spazio**
- Risolviamo il problema risolvendo sotto-problemi di dimensione crescente e **memorizzando** ognuna delle sotto-soluzioni (ottimali) **in una tabella** (tipicamente)
- La tabella è poi usata per trovare la soluzione ottima ai problemi più grandi



@2020
giuseppe.prenepi@unipi.it

Programmazione Dinamica

- Nella programmazione dinamica tipicamente si **riduce il tempo** **aumentando** l'utilizzo di **spazio**
- Risolviamo il problema risolvendo sotto-problemi di dimensione crescente e **memorizzando** ognuna delle sotto-soluzioni (ottimali) **in una tabella** (tipicamente)
- La tabella è poi usata per trovare la soluzione ottima ai problemi più grandi
- Si riduce il costo in tempo **risolvendo** ognuno dei sotto-problemi **solo una volta**



@2020
giuseppe.prenepi@unipi.it

Programmazione Dinamica

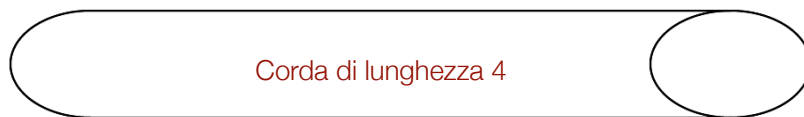
- Il modo migliore per comprendere questa tecnica è attraverso esempi — ragionamenti **simili** (ma leggermente **diversi**) per la **stessa conclusione**
 - Taglio della corda
 - Calcolo coefficienti binomiali
 - Matrix Chaining optimization
 - Longest Common Subsequence
 - Problema dello Zaino 0-1
 - Chiusura transitiva di un grafo orientato

@2020
giuseppe.prencepi@unipi.it

Taglio della corda (rod-cutting)

Taglio della corda (rod-cutting)

- Ogni pezzo di corda ha un **prezzo**
- **Massimizzare** il guadagno!

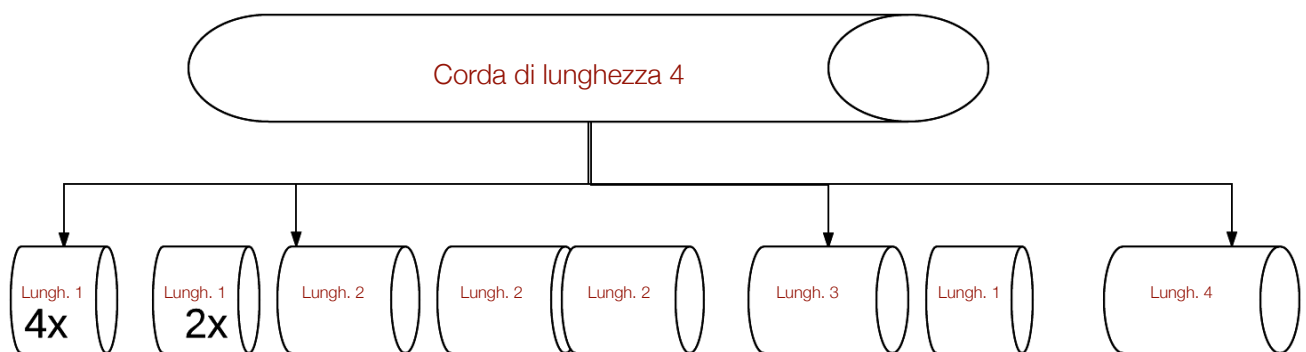


Possibili tagli?

giuseppe.prencipe@unipi.it

Taglio della corda (rod-cutting)

- Ogni pezzo di corda ha un **prezzo**
- **Massimizzare** il guadagno!



Possibili tagli?

giuseppe.prencipe@unipi.it

Taglio della corda (rod-cutting)

Definizione del problema

- Data una corda di lunghezza n centimetri e una tabella di prezzi p_i , $i=1,2,\dots,n$, trovare il massimo guadagno r_n ottenibile tagliando la corda e vendendo i singoli pezzi
 - Lunghezze della corda sono interi
 - Per $i=1,2,\dots,n$ conosciamo il prezzo p_i di un pezzo di corda lungo i cm

@2020
giuseppe.prencipe@unipi.it

Taglio della corda: esempio

lunghezza l	1	2	3	4	5	6	7	8	9	10
prezzo p_i	1	5	8	9	10	17	17	20	24	30

- Per una corda di **lunghezza 4**: quale taglio è ottimale?

@2020
giuseppe.prencipe@unipi.it

Taglio della corda: esempio

lunghezza l	1	2	3	4	5	6	7	8	9	10
prezzo p_i	1	5	8	9	10	17	17	20	24	30

- Per una corda di lunghezza 4: $2+2$ è ottimale ($p_2+p_2=10$)
- In generale, ci sono 2^{n-1} modi per tagliare una corda di lunghezza n !

Taglio della corda

Taglio della corda

- Se la soluzione ottima taglia la corda in k pezzi, allora
 - Decomposizione ottima: $n=i_1+i_2+\dots+i_k$
 - Guadagno: $r_n=p_{i_1}+p_{i_2}+\dots+p_{i_k}$

Taglio della corda

- Se la soluzione ottima taglia la corda in k pezzi, allora
 - Decomposizione ottima: $n=i_1+i_2+\dots+i_k$
 - Guadagno: $r_n=p_{i_1}+p_{i_2}+\dots+p_{i_k}$
- Come potremmo procedere?

Taglio della corda

- Se la soluzione ottima taglia la corda in k pezzi, allora
 - Decomposizione ottima: $n=i_1+i_2+\dots+i_k$
 - Guadagno: $r_n=p_{i_1}+p_{i_2}+\dots+p_{i_k}$

Taglio della corda

- Se la soluzione ottima taglia la corda in k pezzi, allora
 - Decomposizione ottima: $n=i_1+i_2+\dots+i_k$
 - Guadagno: $r_n=p_{i_1}+p_{i_2}+\dots+p_{i_k}$
- Taglio iniziale della corda: due pezzi lunghi i e $n-i$

Taglio della corda

- Se la soluzione ottima taglia la corda in k pezzi, allora
 - Decomposizione ottima: $n=i_1+i_2+\dots+i_k$
 - Guadagno: $r_n=p_{i_1}+p_{i_2}+\dots+p_{i_k}$
- Taglio iniziale della corda: due pezzi lunghi i e $n-i$
 - Guadagno r_i e r_{n-i} derivati da questi due pezzi

Taglio della corda

- Se la soluzione ottima taglia la corda in k pezzi, allora
 - Decomposizione ottima: $n=i_1+i_2+\dots+i_k$
 - Guadagno: $r_n=p_{i_1}+p_{i_2}+\dots+p_{i_k}$
- Taglio iniziale della corda: due pezzi lunghi i e $n-i$
 - Guadagno r_i e r_{n-i} derivati da questi due pezzi
 - Bisogna considerare tutti i possibili valori di i

Taglio della corda

- Se la soluzione ottima taglia la corda in k pezzi, allora
 - Decomposizione ottima: $n=i_1+i_2+\dots+i_k$
 - Guadagno: $r_n=p_{i_1}+p_{i_2}+\dots+p_{i_k}$
- Taglio iniziale della corda: due pezzi lunghi i e $n-i$
 - Guadagno r_i e r_{n-i} derivati da questi due pezzi
 - Bisogna considerare tutti i possibili valori di i
 - Ovviamente va considerato anche il caso in cui la corda è venduta senza tagli

Taglio della corda

- Se la soluzione ottima taglia la corda in k pezzi, allora
 - Decomposizione ottima: $n=i_1+i_2+\dots+i_k$
 - Guadagno: $r_n=p_{i_1}+p_{i_2}+\dots+p_{i_k}$
- Taglio iniziale della corda: due pezzi lunghi i e $n-i$
 - Guadagno r_i e r_{n-i} derivati da questi due pezzi
 - Bisogna considerare tutti i possibili valori di i
 - Ovviamente va considerato anche il caso in cui la corda è venduta senza tagli
- In altre parole: $r_n=\max\{p_n, \text{????}, \text{????}, \dots\}$

Taglio della corda

- Se la soluzione ottima taglia la corda in k pezzi, allora
 - Decomposizione ottima: $n=i_1+i_2+\dots+i_k$
 - Guadagno: $r_n=p_{i_1}+p_{i_2}+\dots+p_{i_k}$
- Taglio iniziale della corda: due pezzi lunghi i e $n-i$
 - Guadagno r_i e r_{n-i} derivati da questi due pezzi
 - Bisogna considerare tutti i possibili valori di i
 - Ovviamente va considerato anche il caso in cui la corda è venduta senza tagli
- In altre parole: $r_n=\max\{p_n, r_1+r_{n-1}, r_2+r_{n-2}, \dots, r_{n-1}+r_1\}$

@2020
giuseppe.precipe@unipi.it

Una differente visione del problema

- Decomponiamo in
 - Un primo pezzo, a sx, di lunghezza i
 - Il secondo pezzo, che resta a dx, di lunghezza $n-i$
 - Si prova a dividere ulteriormente solo il pezzo a dx
 - Questo per ogni i
 - Quindi
 - $r_n=\max\{p_i + \text{????} \mid i \leq n\}$
 - Cerchiamo la soluzione solo a un sotto-problema

@2020
giuseppe.precipe@unipi.it

Una differente visione del problema

- Decomponiamo in
 - Un primo pezzo, a sx, di lunghezza i
 - Il secondo pezzo, che resta a dx, di lunghezza $n-i$
 - Si prova a dividere ulteriormente solo il pezzo a dx
 - Questo per ogni i
- Quindi
 - $r_n = \max\{p_i + r_{n-i}, 1 \leq i \leq n\}$
- Cerchiamo la soluzione solo a un sotto-problema

@2020
giuseppe.prencipe@unipi.it

Taglio della corda: esempio

Corda da 4 **Taglia(4)**

$p[] = [1, 5, 8, 9]$

$i = 1$ **Taglia(4)** Costo????

@2020
giuseppe.prencipe@unipi.it

Taglio della corda: esempio

Corda da 4 **Taglia(4)**

$p[] = [1, 5, 8, 9]$

$i = 1$ **Taglia(4)** Costo: $p_1 + \text{Taglia}(3)$

Taglio della corda: esempio

Corda da 4 **Taglia(4)**

$p[] = [1, 5, 8, 9]$

$i = 1$ **Taglia(4)** Costo: $p_1 + \text{Taglia}(3)$

----- Ricorsivamente, ripartiamo da $i=1$

Taglio della corda: esempio

Corda da 4 **Taglia(4)** $p[] = [1, 5, 8, 9]$

$i = 1$ **Taglia(4)** Costo: $p_1 + \text{Taglia}(3)$ Ricorsivamente, ripartiamo da $i=1$

$i = 1$ **Taglia(3)**

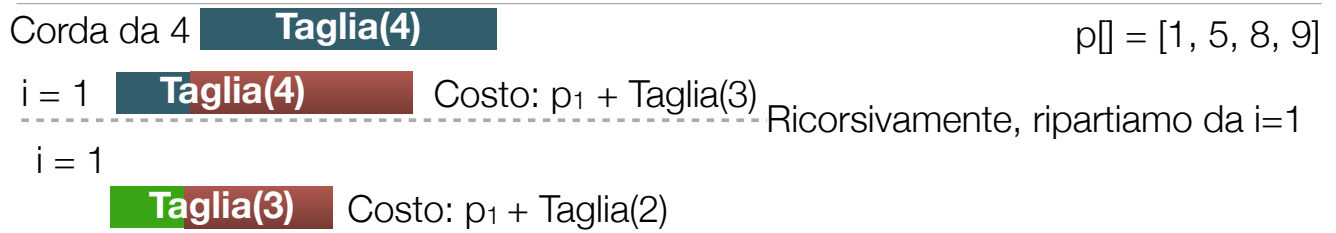
Taglio della corda: esempio

Corda da 4 **Taglia(4)** $p[] = [1, 5, 8, 9]$

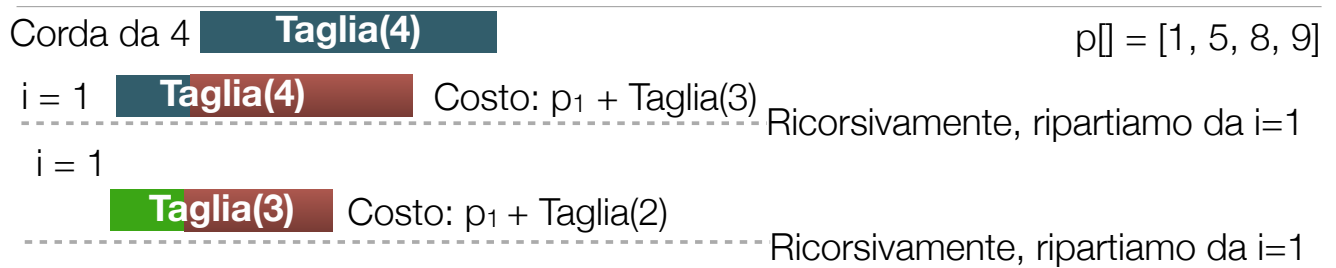
$i = 1$ **Taglia(4)** Costo: $p_1 + \text{Taglia}(3)$ Ricorsivamente, ripartiamo da $i=1$

$i = 1$ **Taglia(3)** Costo????

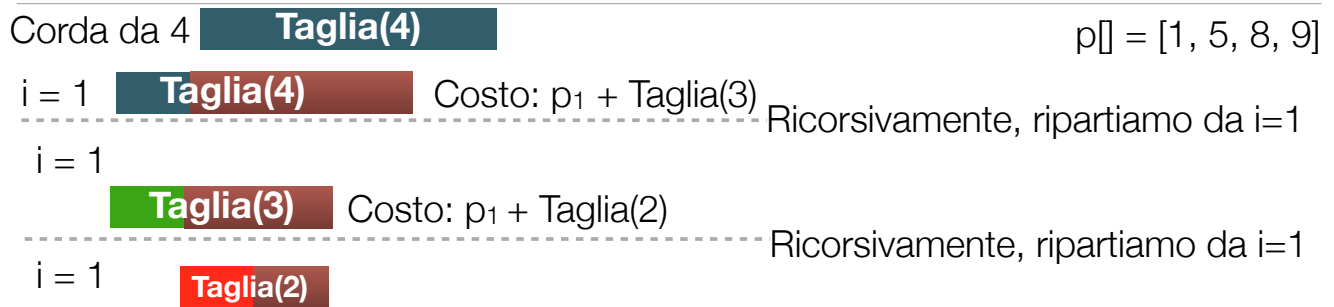
Taglio della corda: esempio



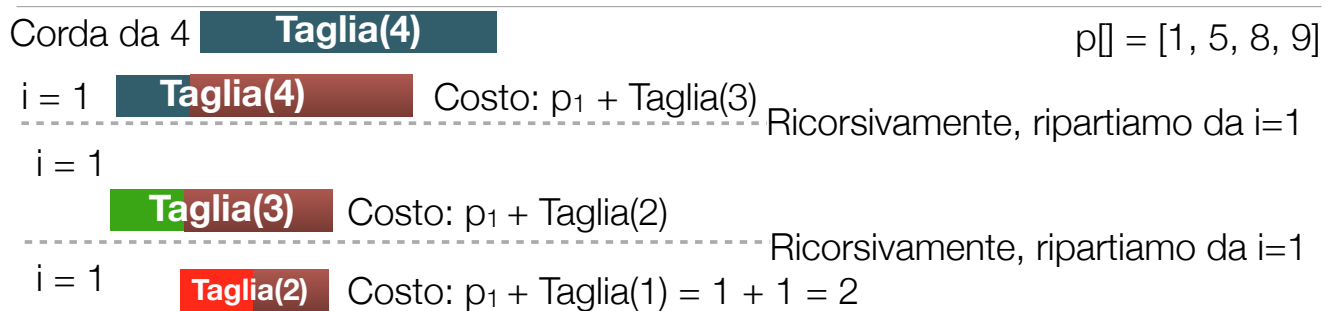
Taglio della corda: esempio



Taglio della corda: esempio



Taglio della corda: esempio



Taglio della corda: esempio

Corda da 4 **Taglia(4)** $p[] = [1, 5, 8, 9]$

$i = 1$ **Taglia(4)** Costo: $p_1 + \text{Taglia}(3)$ Ricorsivamente, ripartiamo da $i=1$

$i = 1$ **Taglia(3)** Costo: $p_1 + \text{Taglia}(2)$ Ricorsivamente, ripartiamo da $i=1$

$i = 1$ **Taglia(2)** Costo: $p_1 + \text{Taglia}(1) = 1 + 1 = 2$

ORA????

Taglio della corda: esempio

Corda da 4 **Taglia(4)** $p[] = [1, 5, 8, 9]$

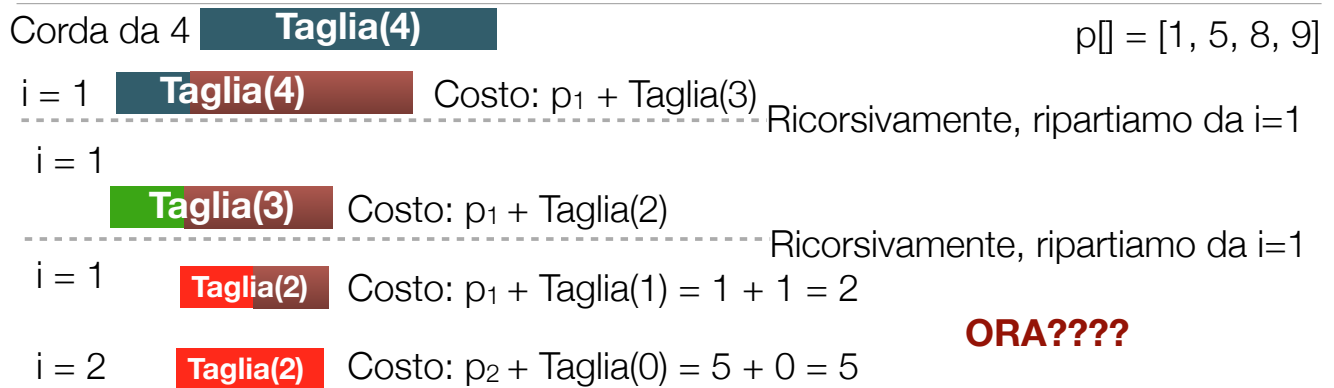
$i = 1$ **Taglia(4)** Costo: $p_1 + \text{Taglia}(3)$ Ricorsivamente, ripartiamo da $i=1$

$i = 1$ **Taglia(3)** Costo: $p_1 + \text{Taglia}(2)$ Ricorsivamente, ripartiamo da $i=1$

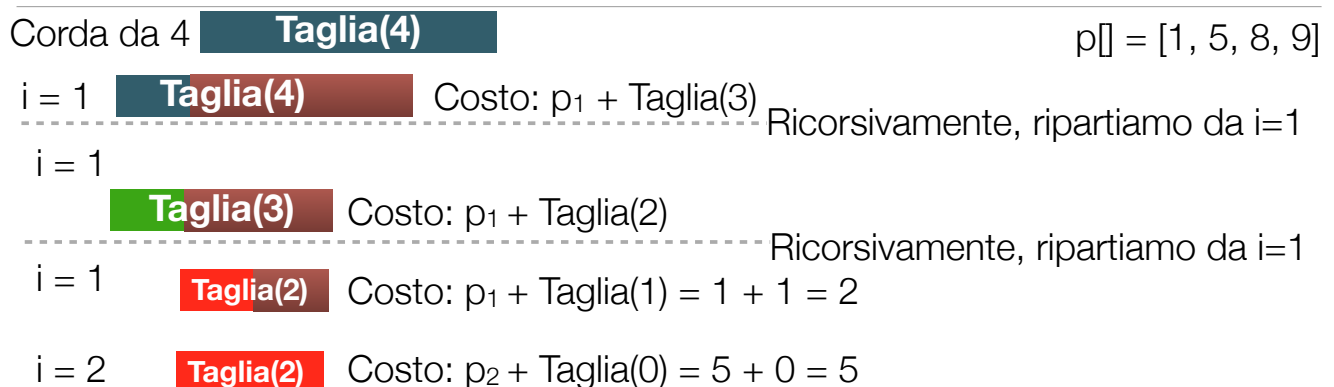
$i = 1$ **Taglia(2)** Costo: $p_1 + \text{Taglia}(1) = 1 + 1 = 2$

ORA????

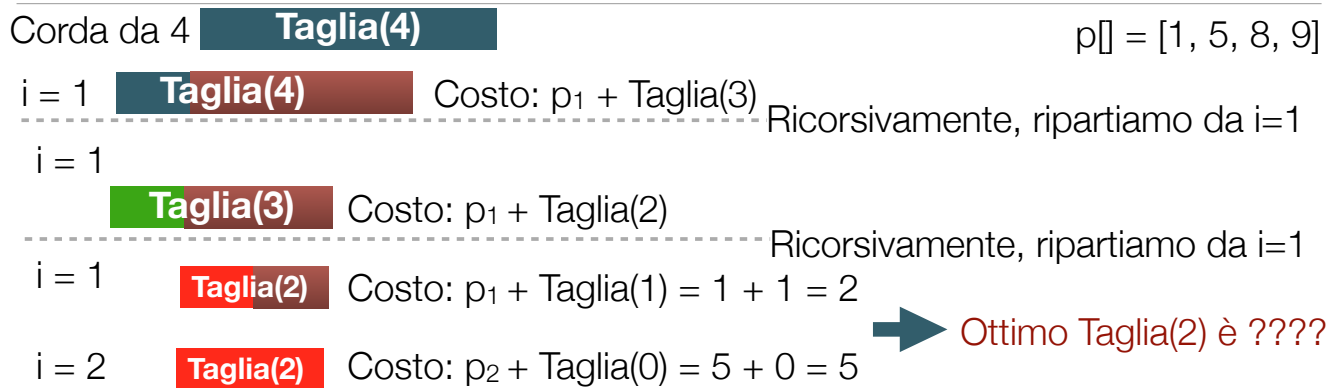
Taglio della corda: esempio



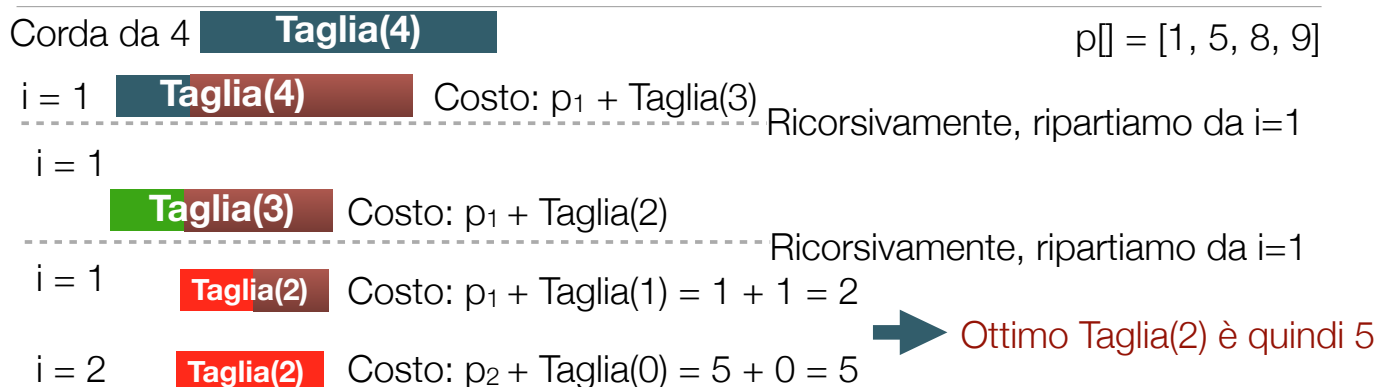
Taglio della corda: esempio



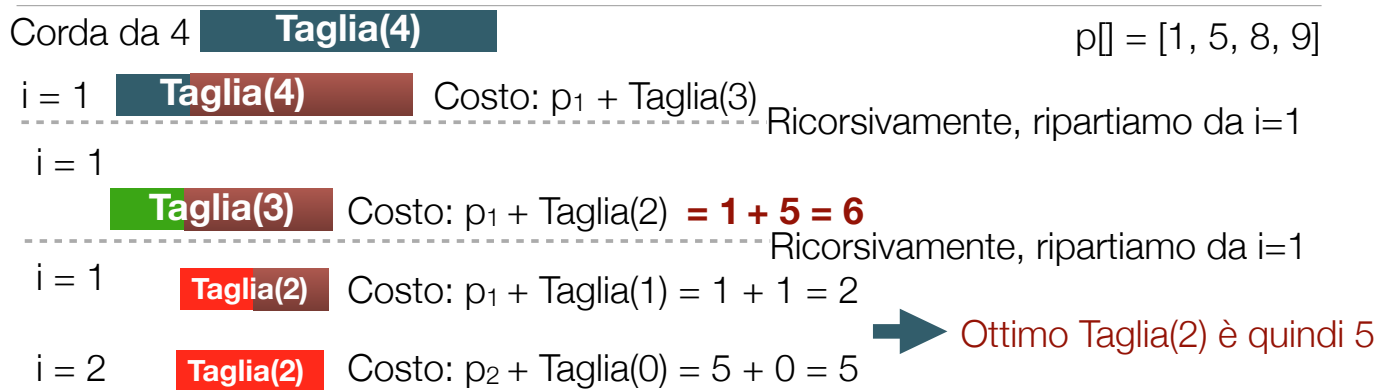
Taglio della corda: esempio



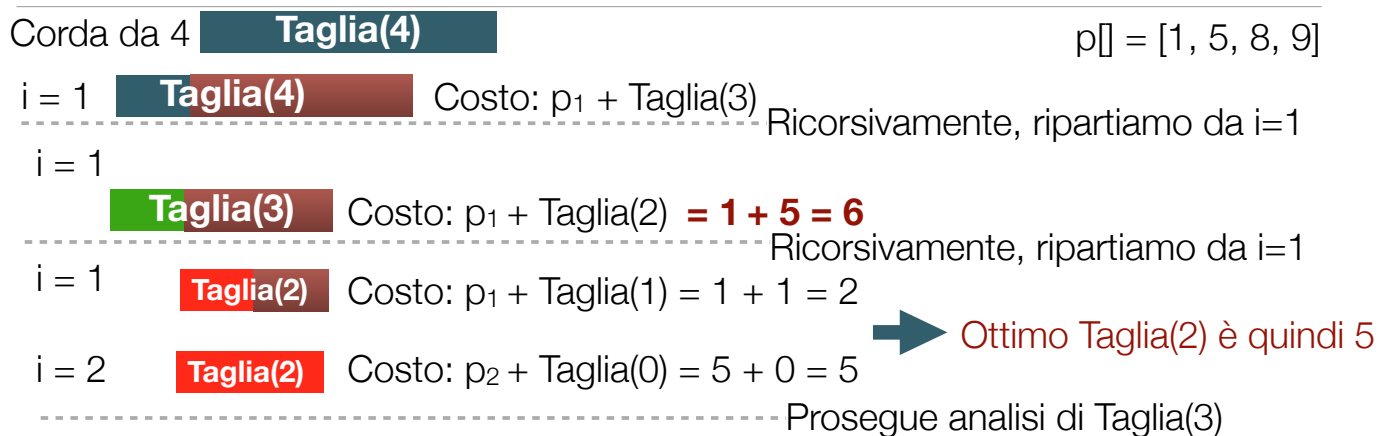
Taglio della corda: esempio



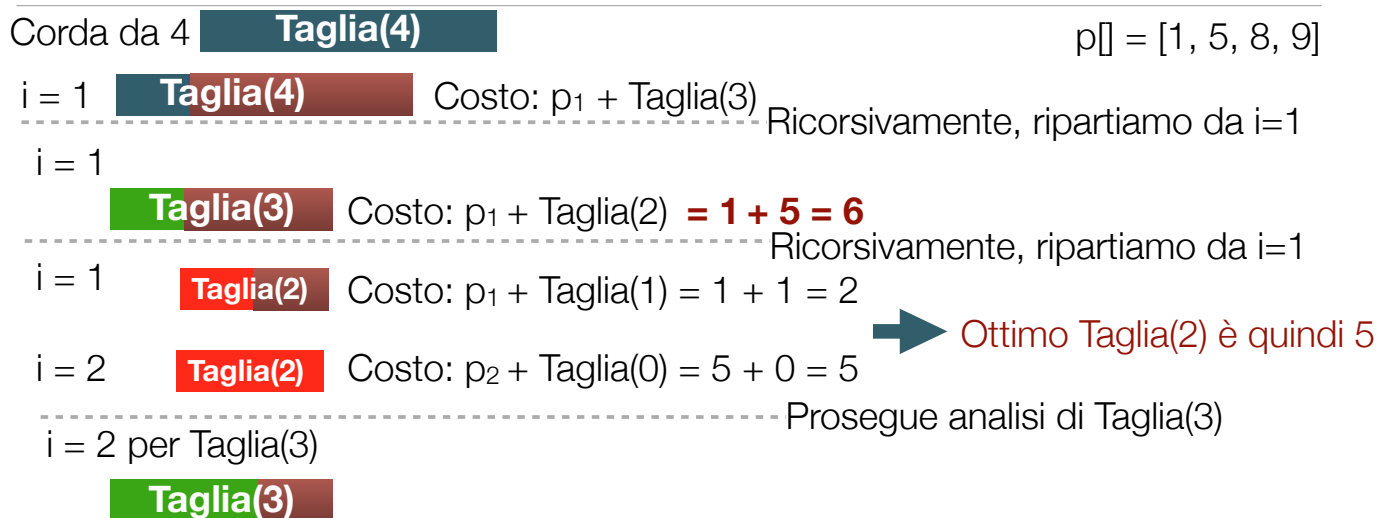
Taglio della corda: esempio



Taglio della corda: esempio

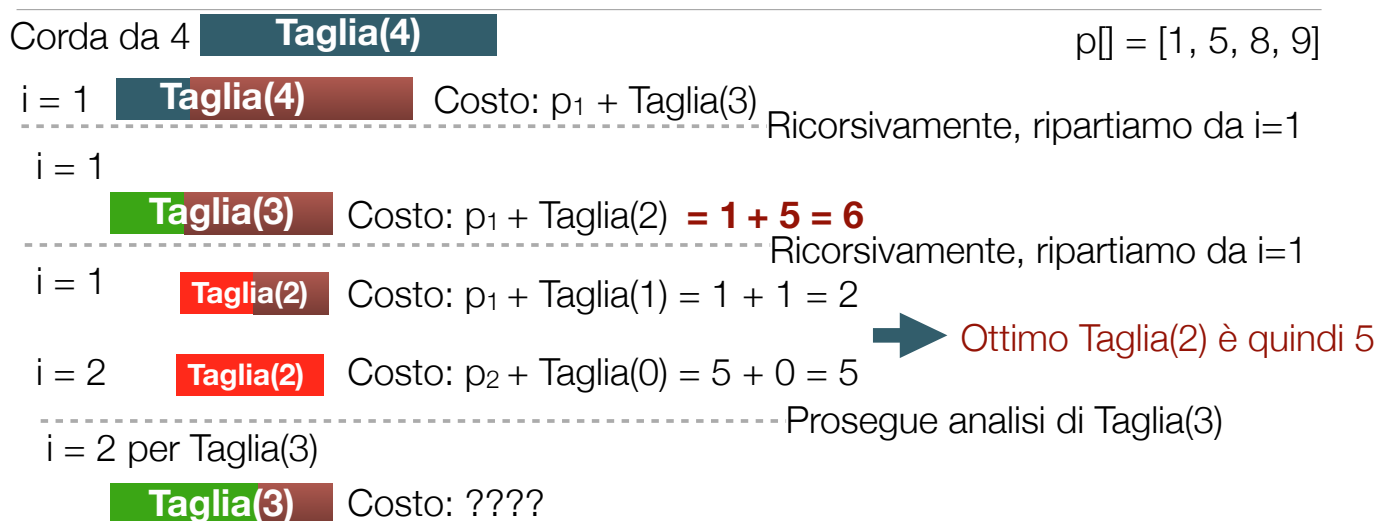


Taglio della corda: esempio



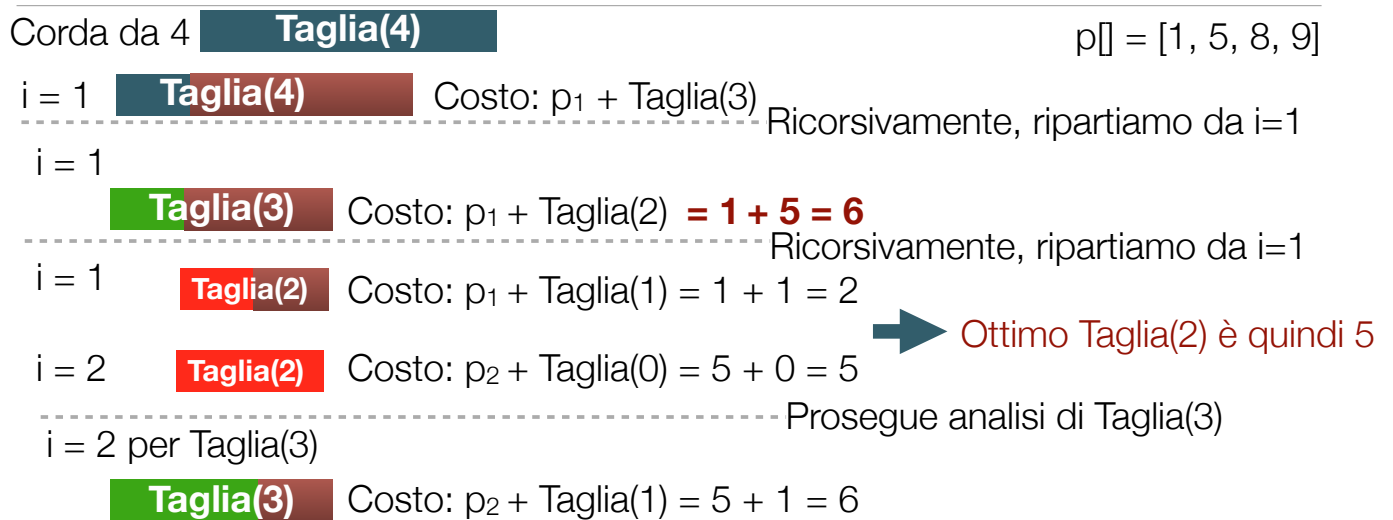
@2020
giuseppe.prencipe@unipi.it

Taglio della corda: esempio



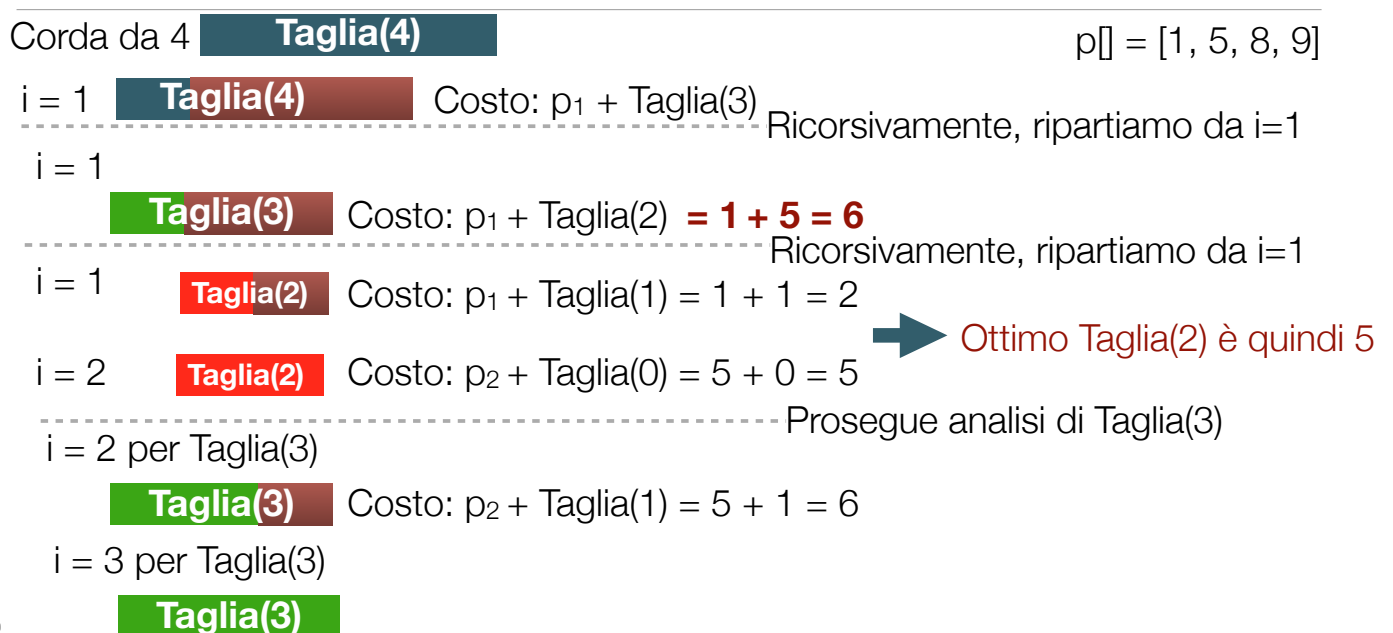
@2020
giuseppe.prencipe@unipi.it

Taglio della corda: esempio



@2020
giuseppe.prencipe@unipi.it

Taglio della corda: esempio



@2020
giuseppe.prencipe@unipi.it

Taglio della corda: esempio

Corda da 4 **Taglia(4)** $p[] = [1, 5, 8, 9]$

$i = 1$ **Taglia(4)** Costo: $p_1 + \text{Taglia}(3)$ Ricorsivamente, ripartiamo da $i=1$

$i = 1$ **Taglia(3)** Costo: $p_1 + \text{Taglia}(2) = 1 + 5 = 6$ Ricorsivamente, ripartiamo da $i=1$

$i = 1$ **Taglia(2)** Costo: $p_1 + \text{Taglia}(1) = 1 + 1 = 2$

$i = 2$ **Taglia(2)** Costo: $p_2 + \text{Taglia}(0) = 5 + 0 = 5$ ➔ **Ottimo Taglia(2) è quindi 5**

$i = 2$ per Taglia(3) Prosegue analisi di Taglia(3)

Taglia(3) Costo: $p_2 + \text{Taglia}(1) = 5 + 1 = 6$

$i = 3$ per Taglia(3)

Taglia(3) Costo: ????

@2020
giuseppe.precipe@unipi.it

Taglio della corda: esempio

Corda da 4 **Taglia(4)** $p[] = [1, 5, 8, 9]$

$i = 1$ **Taglia(4)** Costo: $p_1 + \text{Taglia}(3)$ Ricorsivamente, ripartiamo da $i=1$

$i = 1$ **Taglia(3)** Costo: $p_1 + \text{Taglia}(2) = 1 + 5 = 6$ Ricorsivamente, ripartiamo da $i=1$

$i = 1$ **Taglia(2)** Costo: $p_1 + \text{Taglia}(1) = 1 + 1 = 2$

$i = 2$ **Taglia(2)** Costo: $p_2 + \text{Taglia}(0) = 5 + 0 = 5$ ➔ **Ottimo Taglia(2) è quindi 5**

$i = 2$ per Taglia(3) Prosegue analisi di Taglia(3)

Taglia(3) Costo: $p_2 + \text{Taglia}(1) = 5 + 1 = 6$

$i = 3$ per Taglia(3)

Taglia(3) Costo: $p_3 + \text{Taglia}(0) = 8 + 0 = 8$

@2020
giuseppe.precipe@unipi.it

Taglio della corda: esempio

Corda da 4 **Taglia(4)** $p[] = [1, 5, 8, 9]$

$i = 1$ **Taglia(4)** Costo: $p_1 + \text{Taglia}(3)$ Ricorsivamente, ripartiamo da $i=1$

$i = 1$ **Taglia(3)** Costo: $p_1 + \text{Taglia}(2) = 1 + 5 = 6$ Ricorsivamente, ripartiamo da $i=1$

$i = 1$ **Taglia(2)** Costo: $p_1 + \text{Taglia}(1) = 1 + 1 = 2$ ➔ Ottimo Taglia(2) è quindi 5

$i = 2$ **Taglia(2)** Costo: $p_2 + \text{Taglia}(0) = 5 + 0 = 5$ Prosegue analisi di Taglia(3)

$i = 2$ per Taglia(3) **Taglia(3)** Costo: $p_2 + \text{Taglia}(1) = 5 + 1 = 6$ ➔ Ottimo Taglia(3) è ????

$i = 3$ per Taglia(3) **Taglia(3)** Costo: $p_3 + \text{Taglia}(0) = 8 + 0 = 8$

@2020
giuseppe.prencipe@unipi.it

Taglio della corda: esempio

Corda da 4 **Taglia(4)** $p[] = [1, 5, 8, 9]$

$i = 1$ **Taglia(4)** Costo: $p_1 + \text{Taglia}(3)$ Ricorsivamente, ripartiamo da $i=1$

$i = 1$ **Taglia(3)** Costo: $p_1 + \text{Taglia}(2) = 1 + 5 = 6$ Ricorsivamente, ripartiamo da $i=1$

$i = 1$ **Taglia(2)** Costo: $p_1 + \text{Taglia}(1) = 1 + 1 = 2$ ➔ Ottimo Taglia(2) è quindi 5

$i = 2$ **Taglia(2)** Costo: $p_2 + \text{Taglia}(0) = 5 + 0 = 5$ Prosegue analisi di Taglia(3)

$i = 2$ per Taglia(3) **Taglia(3)** Costo: $p_2 + \text{Taglia}(1) = 5 + 1 = 6$ ➔ Ottimo Taglia(3) è quindi 8

$i = 3$ per Taglia(3) **Taglia(3)** Costo: $p_3 + \text{Taglia}(0) = 8 + 0 = 8$

@2020
giuseppe.prencipe@unipi.it

Taglio della corda: esempio

Corda da 4 **Taglia(4)** $p[] = [1, 5, 8, 9]$

$i = 1$ **Taglia(4)** Costo: $p_1 + \text{Taglia}(3)$ Ricorsivamente, ripartiamo da $i=1$

$i = 1$ **Taglia(3)** Costo: $p_1 + \text{Taglia}(2) = 1 + 5 = 6$ Ricorsivamente, ripartiamo da $i=1$

$i = 1$ **Taglia(2)** Costo: $p_1 + \text{Taglia}(1) = 1 + 1 = 2$ ➡ Ottimo Taglia(2) è quindi 5

$i = 2$ **Taglia(2)** Costo: $p_2 + \text{Taglia}(0) = 5 + 0 = 5$

----- Prosegue analisi di Taglia(3)

$i = 2$ per Taglia(3) **Taglia(3)** Costo: $p_2 + \text{Taglia}(1) = 5 + 1 = 6$ ➡ Ottimo Taglia(3) è quindi 8

$i = 3$ per Taglia(3) **Taglia(3)** Costo: $p_3 + \text{Taglia}(0) = 8 + 0 = 8$

E così via....

@2020
giuseppe.precipec@unipi.it

n	i	p [i]	CUT-ROD (p, n-i)	p [i] + CUT-ROD (p, n-i)	max
4	1	1	$(p, 4 - 1) = (p, 3) = 8$	$1 + 8 = 9$	10
	2	5	$(p, 4 - 2) = (p, 2) = 5$	$5 + 5 = 10$	
	3	8	$(p, 4 - 3) = (p, 1) = 1$	$8 + 1 = 9$	
	4	9	$(p, 4 - 4) = (p, 0) = 0$	$9 + 0 = 9$	
3	1	1	$(p, 3 - 1) = (p, 2) = 5$	$1 + 5 = 6$	8
	2	5	$(p, 3 - 2) = (p, 1) = 1$	$5 + 1 = 6$	
	3	8	$(p, 3 - 3) = (p, 0) = 0$	$8 + 0 = 8$	
2	1	1	$(p, 2 - 1) = (p, 1) = 1$	$1 + 1 = 2$	5
	2	5	$(p, 2 - 2) = (p, 0) = 0$	$5 + 0 = 5$	
1	1	1	$(p, 1 - 1) = (p, 0) = 0$	$1 + 0 = 1$	1

@2020
giuseppe.precipec@unipi.it

Implementazione top-down

```
TAGLIO(p,n)
  if n==0
    return 0
  q =  $-\infty$ 
  for i=1 to n
    q=max{q, p[i]+TAGLIO(p,n-i)}
  return q
```

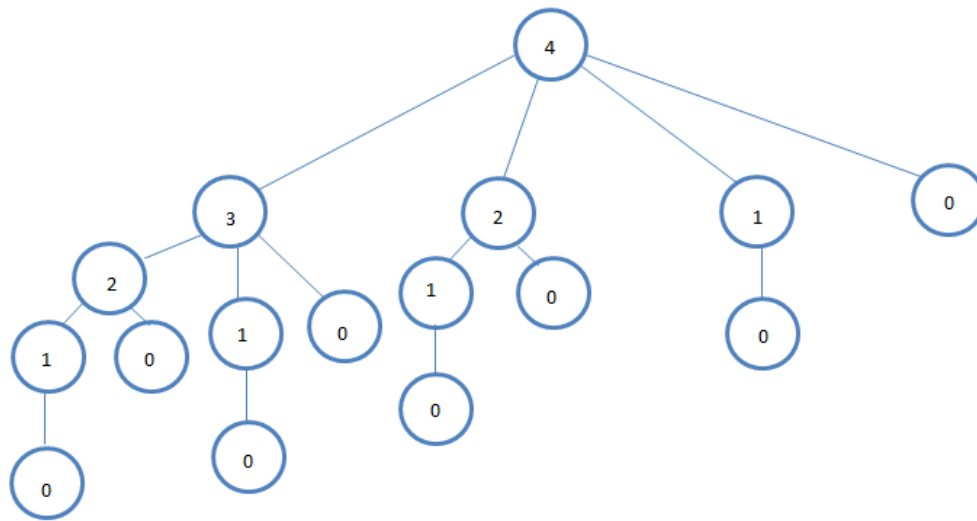
- Tempo: $T(n)=1+????$

Implementazione top-down

```
TAGLIO(p,n)
  if n==0
    return 0
  q =  $-\infty$ 
  for i=1 to n
    q=max{q, p[i]+TAGLIO(p,n-i)}
  return q
```

- Tempo: $T(n)=1+T(1)+T(2)+\dots+T(n-1)$
 - $T(n)=O(2^n)$

Implementazione top-down



Albero ricorsione di TAGLIO(p, n), n = 4.

@2020
giuseppe.prencepi@unipi.it

Programmazione Dinamica — **bottom-up**

L'idea, come fatto per Fibonacci, è partire dai **valori che già conosciamo**, che sono quelli più piccoli

Su quelli più piccoli, che memorizziamo nella **tabella di programmazione dinamica**, calcoliamo quelli via via più grandi, fino a trovare la soluzione del problema

@2020
giuseppe.prencepi@unipi.it

Attenzione ora!



@2020
giuseppe.precipe@unipi.it

Programmazione Dinamica — **bottom-up**

@2020
giuseppe.precipe@unipi.it

Programmazione Dinamica — **bottom-up**

Taglia(4)

$p = [1, 5, 8, 9]$

Programmazione Dinamica — **bottom-up**

$r = [0, 0, 0, 0, 0]$

Taglia(4)

$p = [1, 5, 8, 9]$

Programmazione Dinamica — **bottom-up**

$r = [0, 0, 0, 0, 0]$

-----j=1

Taglia(4)

$p = [1, 5, 8, 9]$

Programmazione Dinamica — **bottom-up**

$r = [0, 0, 0, 0, 0]$

-----j=1



Taglia(4)

$p = [1, 5, 8, 9]$

Programmazione Dinamica — **bottom-up**

$r = [0, 0, 0, 0, 0]$

Taglia(4)

.....j=1
■ → Ottimo = ????

$p = [1, 5, 8, 9]$

Programmazione Dinamica — **bottom-up**

$r = [0, 1, 0, 0, 0]$

Taglia(4)

.....j=1
■ → Ottimo = 1

$p = [1, 5, 8, 9]$

Programmazione Dinamica — **bottom-up**

$r = [0, 1, 0, 0, 0]$

Taglia(4)

----- $j=1$
 → Ottimo = 1

Iterativamente, passiamo a $j=2$



Programmazione Dinamica — **bottom-up**

$r = [0, 1, 0, 0, 0]$

Taglia(4)

----- $j=1$
 → Ottimo = 1

Iterativamente, passiamo a $j=2$



E ricordiamo anche la
definizione generale del
problema di ottimizzazione

$$r_n = \max\{p_i + r_{n-i}, 1 \leq i \leq n\}$$

**Vanno sfruttati i valori ottimi calcolati
per i sotto-problemi**

Al momento conosciamo solo $r[1]$

Programmazione Dinamica — **bottom-up**


$r = [0, 1, 0, 0, 0]$

Taglia(4)

 → Ottimo = 1 $p = [1, 5, 8, 9]$

----- $j=1$

----- Iterativamente, passiamo a $j=2$

 2 possibilità

E ricordiamo anche la definizione generale del problema di ottimizzazione

$$r_n = \max\{p_i + r_{n-i}, 1 \leq i \leq n\}$$

Vanno sfruttati i valori ottimi calcolati per i sotto-problemi

Al momento conosciamo solo $r[1]$

@2020
giuseppe.precipe@unipi.it

Programmazione Dinamica — **bottom-up**


$r = [0, 1, 0, 0, 0]$

Taglia(4)

 → Ottimo = 1 $p = [1, 5, 8, 9]$

----- $j=1$

----- Iterativamente, passiamo a $j=2$

 2 possibilità

E ricordiamo anche la definizione generale del problema di ottimizzazione

$$r_n = \max\{p_i + r_{n-i}, 1 \leq i \leq n\}$$

Vanno sfruttati i valori ottimi calcolati per i sotto-problemi

Al momento conosciamo solo $r[1]$

@2020
giuseppe.precipe@unipi.it

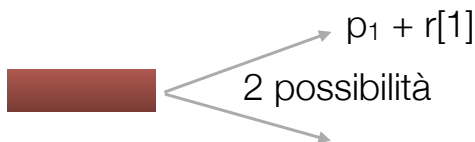
Programmazione Dinamica — **bottom-up**

$r = [0, 1, 0, 0, 0]$

Taglia(4)

----- $j=1$ -----
 → Ottimo = 1

Iterativamente, passiamo a $j=2$



Vanno sfruttati i valori ottimi calcolati per i sotto-problemi

Al momento conosciamo solo $r[1]$

E ricordiamo anche la definizione generale del problema di ottimizzazione

$$r_n = \max\{p_i + r_{n-i}, 1 \leq i \leq n\}$$

@2020
giuseppe.precipe@unipi.it

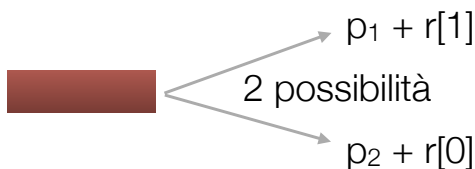
Programmazione Dinamica — **bottom-up**

$r = [0, 1, 0, 0, 0]$

Taglia(4)

----- $j=1$ -----
 → Ottimo = 1

Iterativamente, passiamo a $j=2$



Vanno sfruttati i valori ottimi calcolati per i sotto-problemi

Al momento conosciamo solo $r[1]$

E ricordiamo anche la definizione generale del problema di ottimizzazione

$$r_n = \max\{p_i + r_{n-i}, 1 \leq i \leq n\}$$

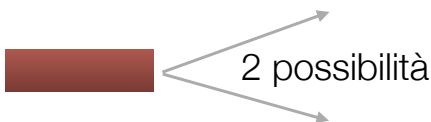
@2020
giuseppe.precipe@unipi.it

Programmazione Dinamica — **bottom-up**

$r = [0, 1, 0, 0, 0]$

Taglia(4)

..... $j=1$ $p = [1, 5, 8, 9]$
..... Iterativamente, passiamo a $j=2$



E ricordiamo anche la definizione generale del problema di ottimizzazione

$$r_n = \max\{p_i + r_{n-i}, 1 \leq i \leq n\}$$

Vanno sfruttati i valori ottimi calcolati per i sotto-problemi

Al momento conosciamo solo $r[1]$

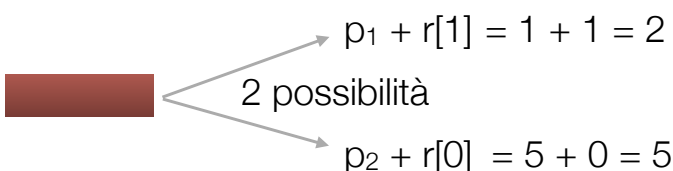
@2020
giuseppe.precipe@unipi.it

Programmazione Dinamica — **bottom-up**

$r = [0, 1, 0, 0, 0]$

Taglia(4)

..... $j=1$ $p = [1, 5, 8, 9]$
..... Iterativamente, passiamo a $j=2$



E ricordiamo anche la definizione generale del problema di ottimizzazione

$$r_n = \max\{p_i + r_{n-i}, 1 \leq i \leq n\}$$

Vanno sfruttati i valori ottimi calcolati per i sotto-problemi

Al momento conosciamo solo $r[1]$

@2020
giuseppe.precipe@unipi.it

Programmazione Dinamica — **bottom-up**

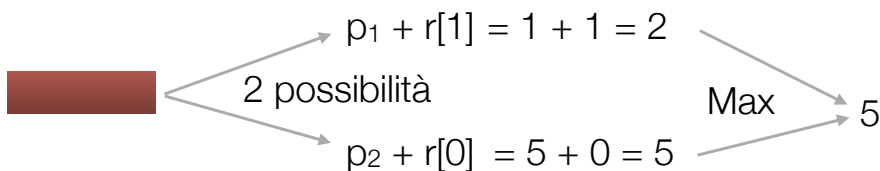
$r = [0, 1, 0, 0, 0]$

Taglia(4)

$p = [1, 5, 8, 9]$

----- $j=1$
[] → Ottimo = 1

----- Iterativamente, passiamo a $j=2$



Vanno sfruttati i valori ottimi calcolati per i sotto-problemi

E ricordiamo anche la definizione generale del problema di ottimizzazione

$$r_n = \max\{p_i + r_{n-i}, 1 \leq i \leq n\}$$

Al momento conosciamo solo $r[1]$

Programmazione Dinamica — **bottom-up**

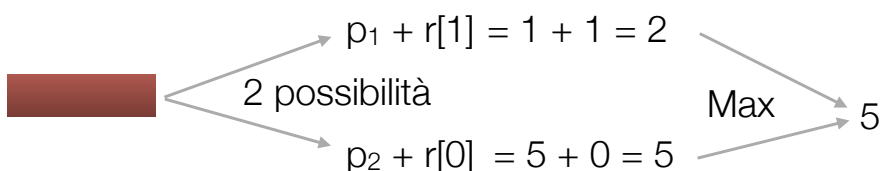
$r = [0, 1, 5, 0, 0]$

Taglia(4)

$p = [1, 5, 8, 9]$

----- $j=1$
[] → Ottimo = 1

----- Iterativamente, passiamo a $j=2$

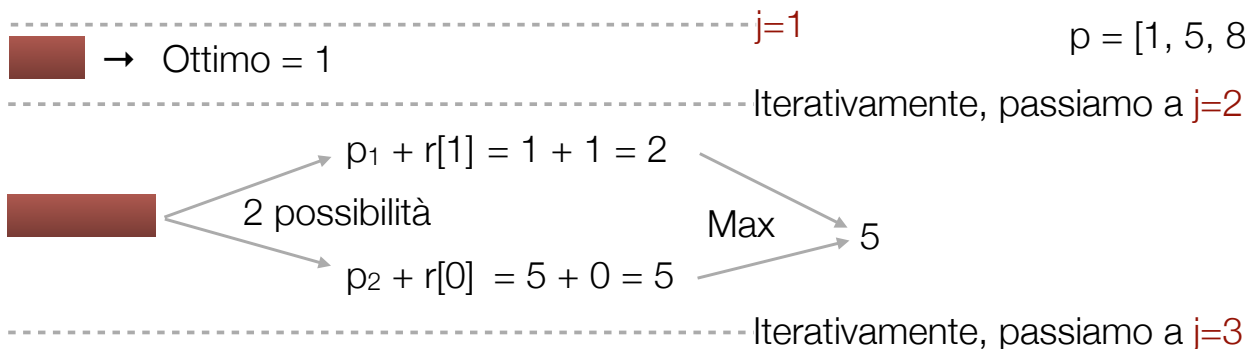


Programmazione Dinamica — **bottom-up**

$r = [0, 1, 5, 0, 0]$

Taglia(4)

$p = [1, 5, 8, 9]$

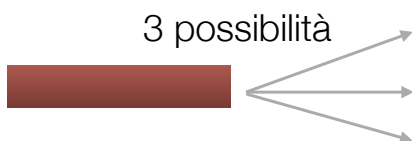
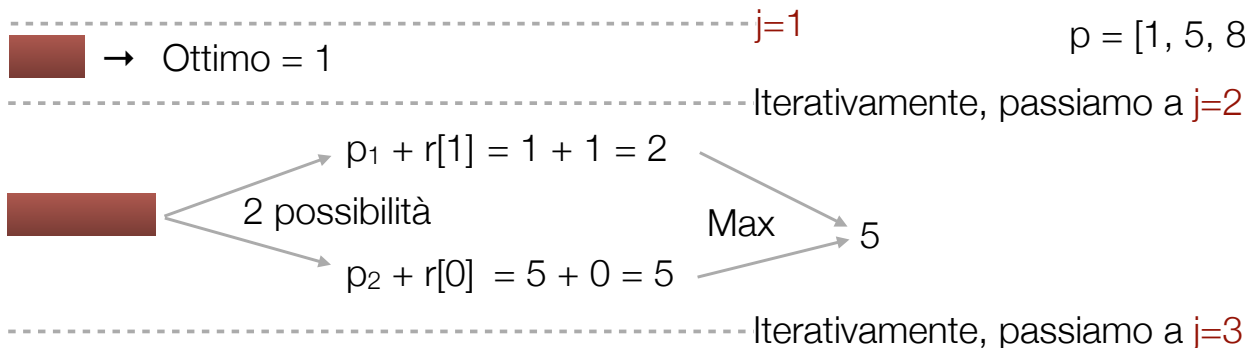


Programmazione Dinamica — **bottom-up**

$r = [0, 1, 5, 0, 0]$

Taglia(4)

$p = [1, 5, 8, 9]$

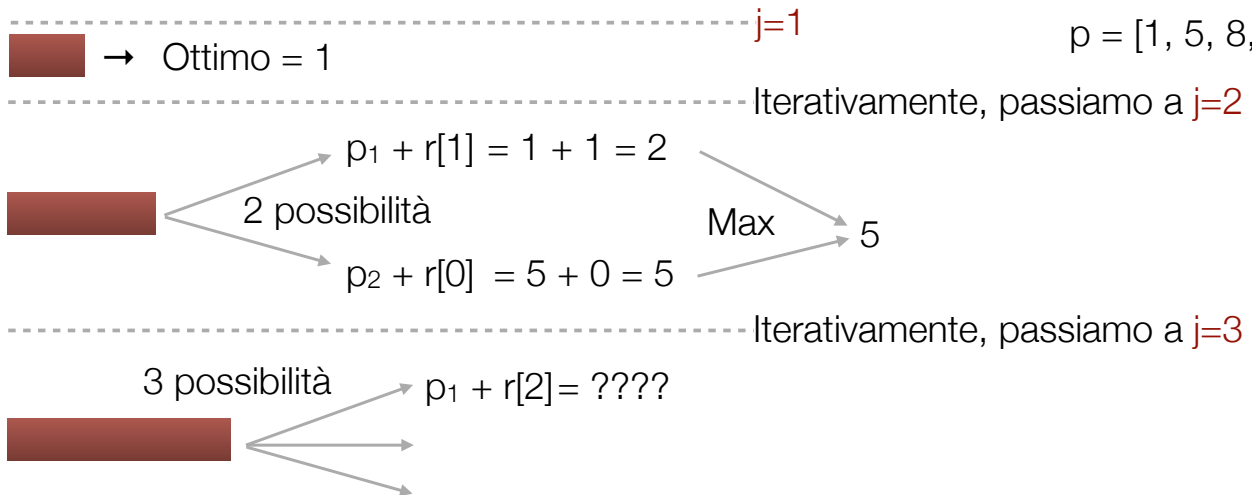


Programmazione Dinamica — **bottom-up**

$r = [0, 1, 5, 0, 0]$

Taglia(4)

$p = [1, 5, 8, 9]$



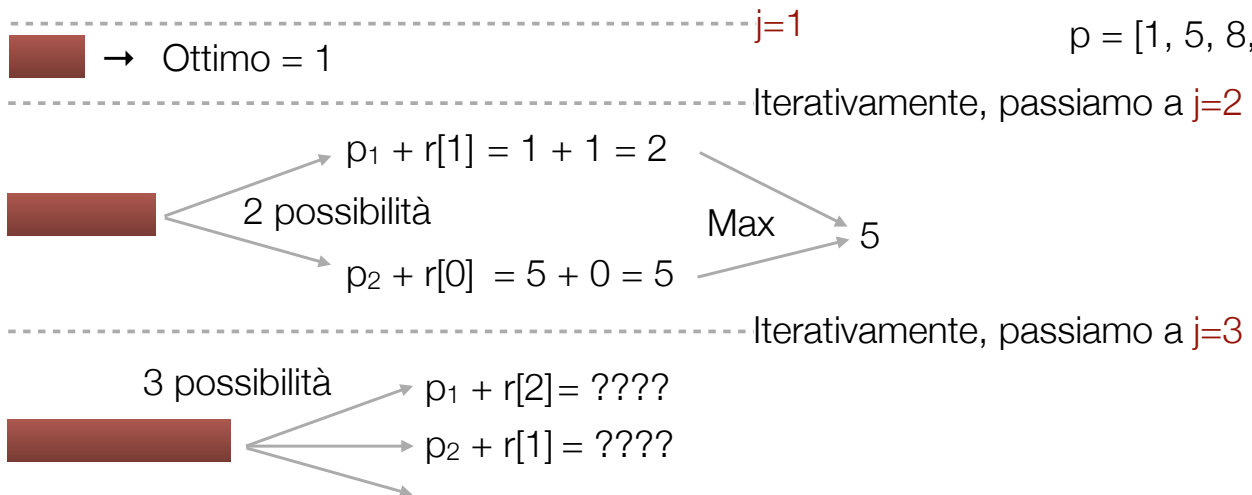
@2020
giuseppe.prencipe@unipi.it

Programmazione Dinamica — **bottom-up**

$r = [0, 1, 5, 0, 0]$

Taglia(4)

$p = [1, 5, 8, 9]$



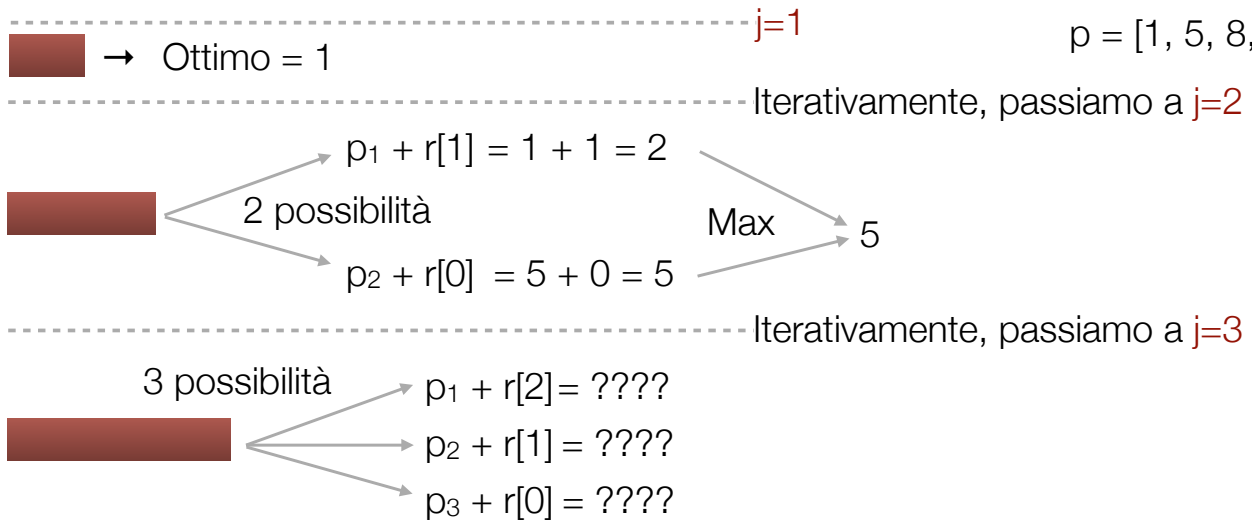
@2020
giuseppe.prencipe@unipi.it

Programmazione Dinamica — **bottom-up**

$r = [0, 1, 5, 0, 0]$

Taglia(4)

$p = [1, 5, 8, 9]$



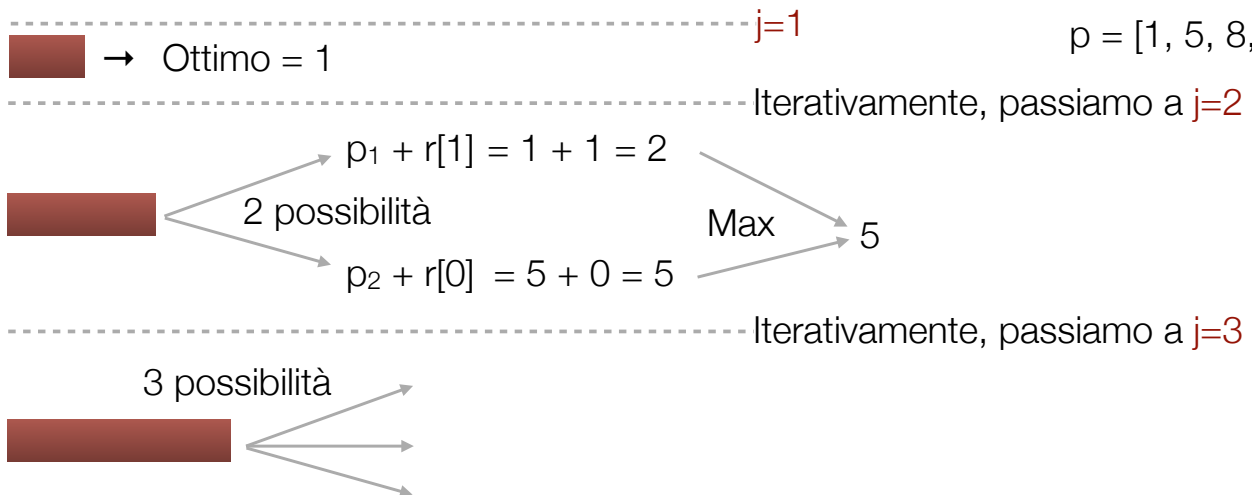
@2020
giuseppe.prencipe@unipi.it

Programmazione Dinamica — **bottom-up**

$r = [0, 1, 5, 0, 0]$

Taglia(4)

$p = [1, 5, 8, 9]$



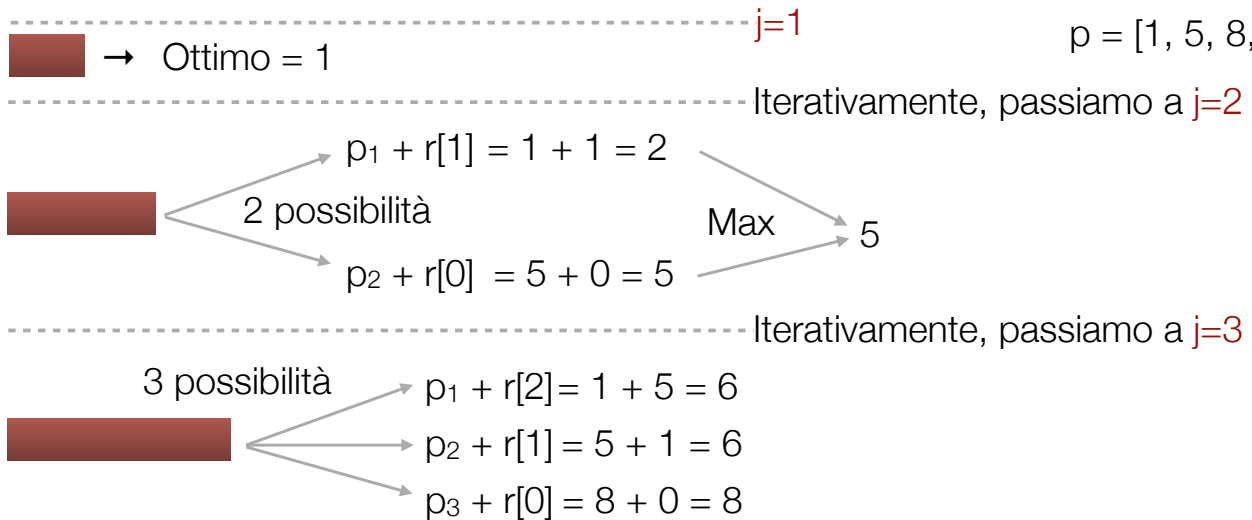
@2020
giuseppe.prencipe@unipi.it

Programmazione Dinamica — **bottom-up**

$r = [0, 1, 5, 0, 0]$

Taglia(4)

$p = [1, 5, 8, 9]$



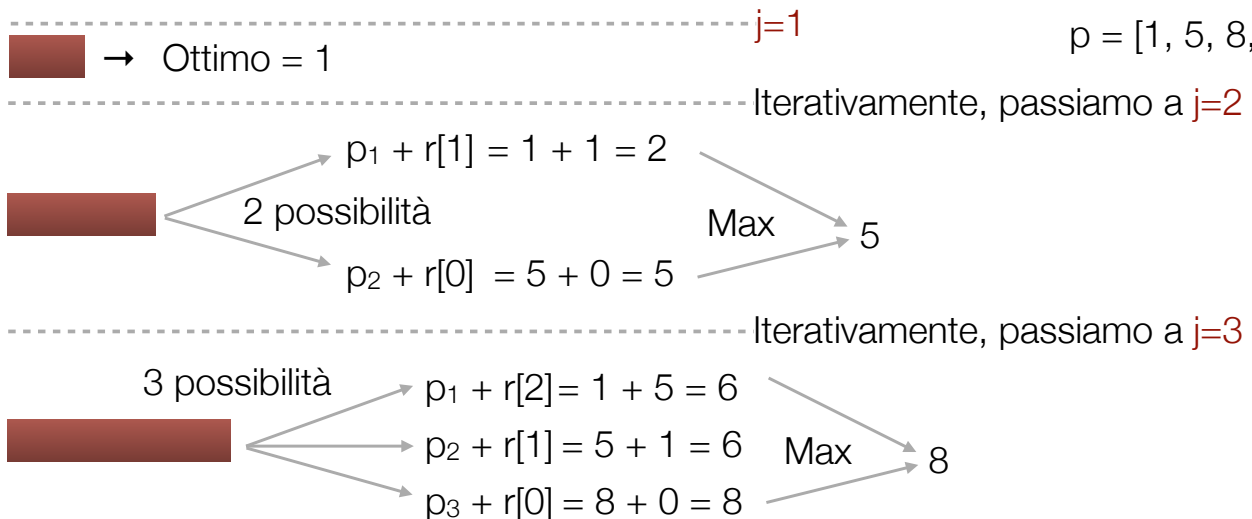
@2020
giuseppe.prencipe@unipi.it

Programmazione Dinamica — **bottom-up**

$r = [0, 1, 5, 0, 0]$

Taglia(4)

$p = [1, 5, 8, 9]$



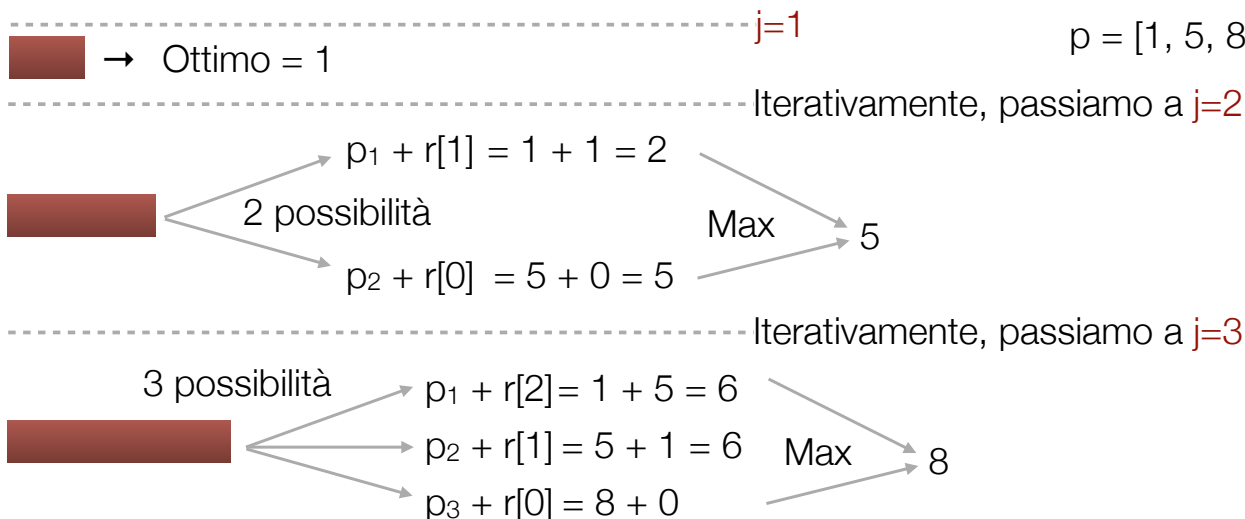
@2020
giuseppe.prencipe@unipi.it

Programmazione Dinamica — **bottom-up**

$r = [0, 1, 5, 8, 0]$

Taglia(4)

$p = [1, 5, 8, 9]$



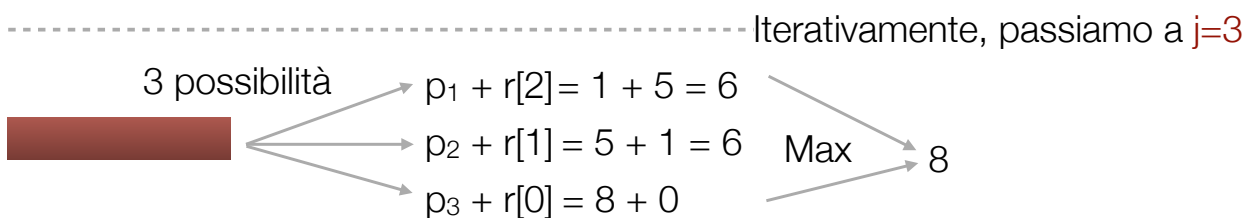
@2020
giuseppe.prencipe@unipi.it

Programmazione Dinamica — **bottom-up**

$r = [0, 1, 5, 8, 0]$

Taglia(4)

$p = [1, 5, 8, 9]$



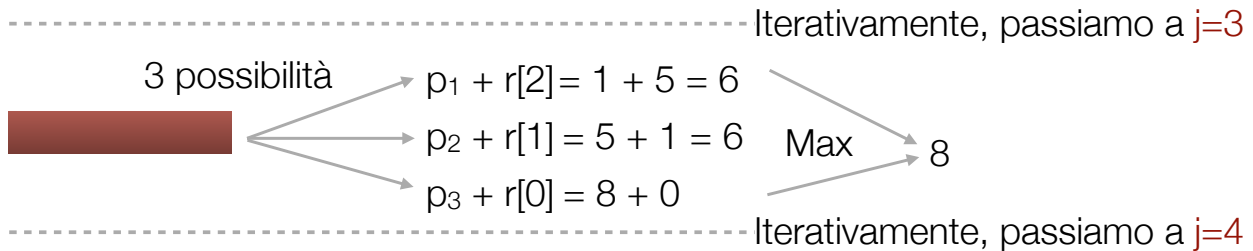
@2020
giuseppe.prencipe@unipi.it

Programmazione Dinamica — **bottom-up**

$r = [0, 1, 5, 8, 0]$

Taglia(4)

$p = [1, 5, 8, 9]$

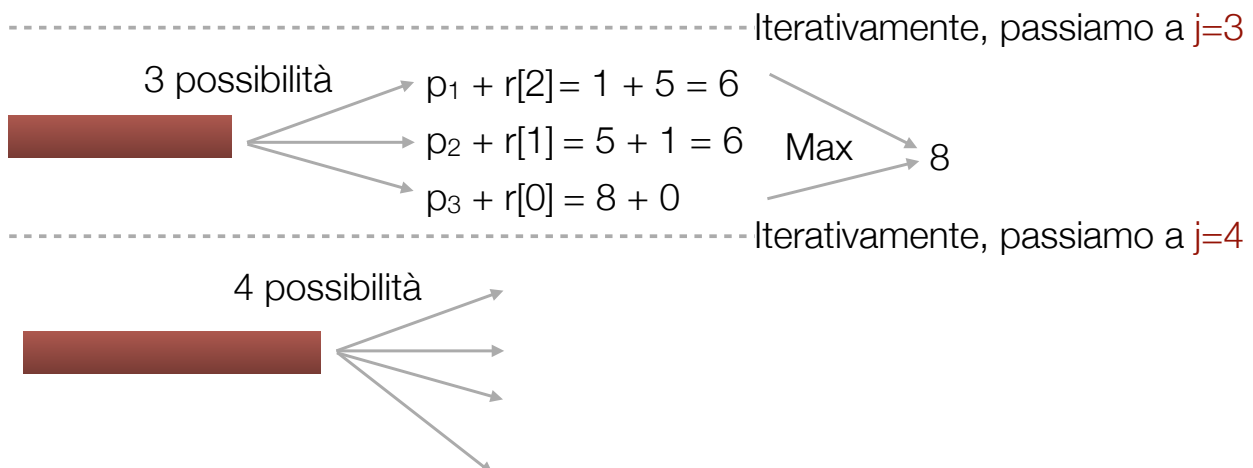


Programmazione Dinamica — **bottom-up**

$r = [0, 1, 5, 8, 0]$

Taglia(4)

$p = [1, 5, 8, 9]$

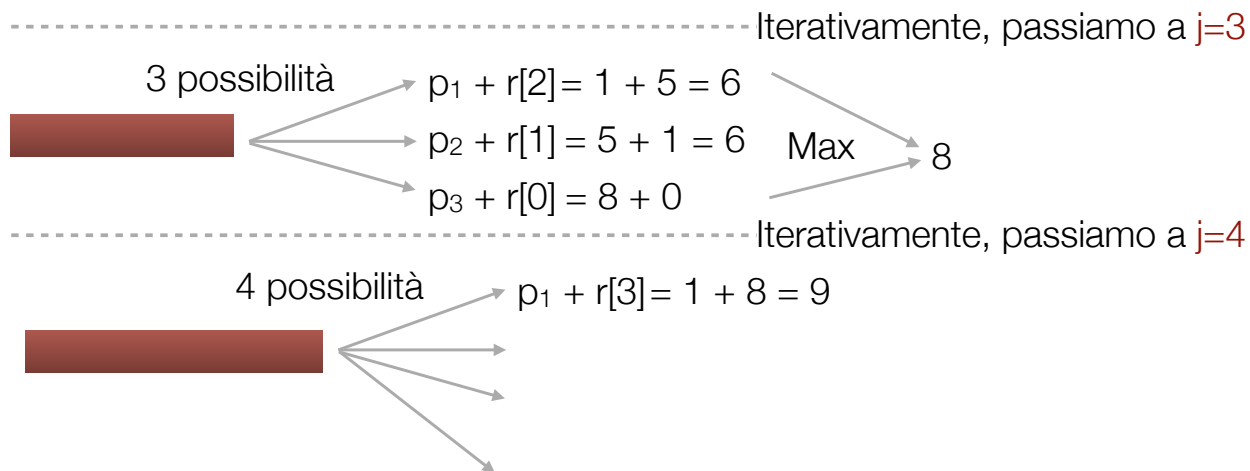


Programmazione Dinamica — **bottom-up**

$r = [0, 1, 5, 8, 0]$

Taglia(4)

$p = [1, 5, 8, 9]$



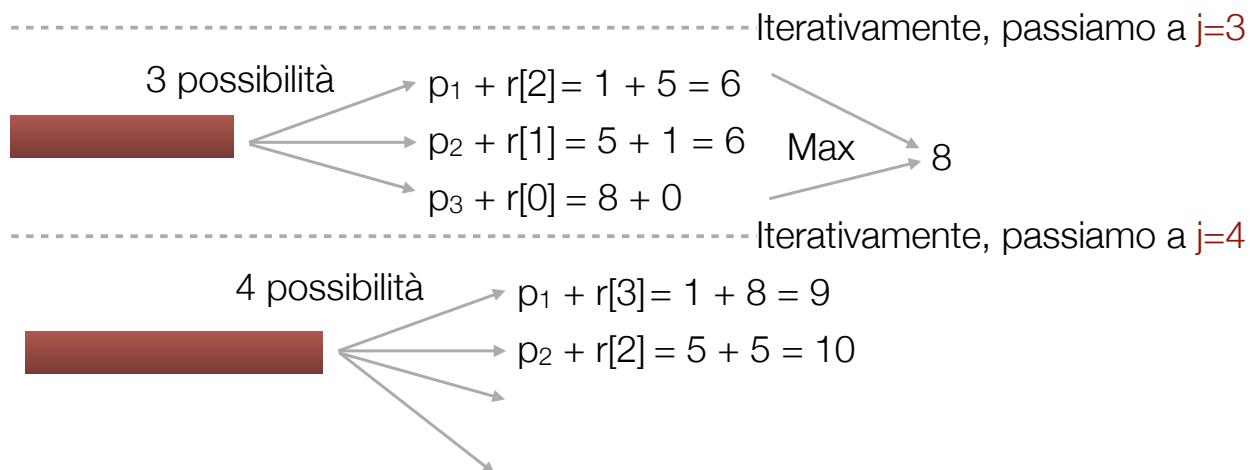
@2020
giuseppe.prencipe@unipi.it

Programmazione Dinamica — **bottom-up**

$r = [0, 1, 5, 8, 0]$

Taglia(4)

$p = [1, 5, 8, 9]$



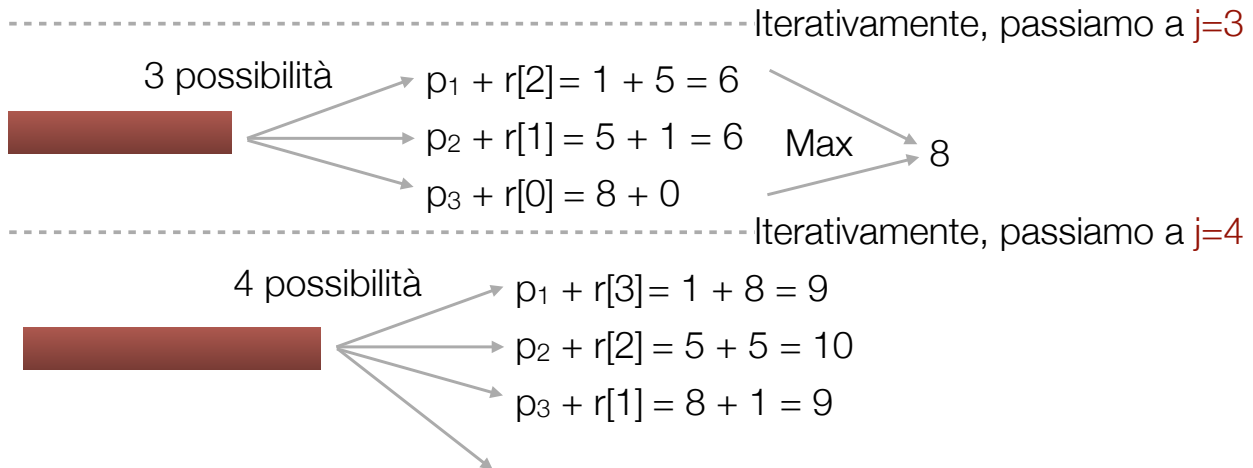
@2020
giuseppe.prencipe@unipi.it

Programmazione Dinamica — **bottom-up**

$r = [0, 1, 5, 8, 0]$

Taglia(4)

$p = [1, 5, 8, 9]$



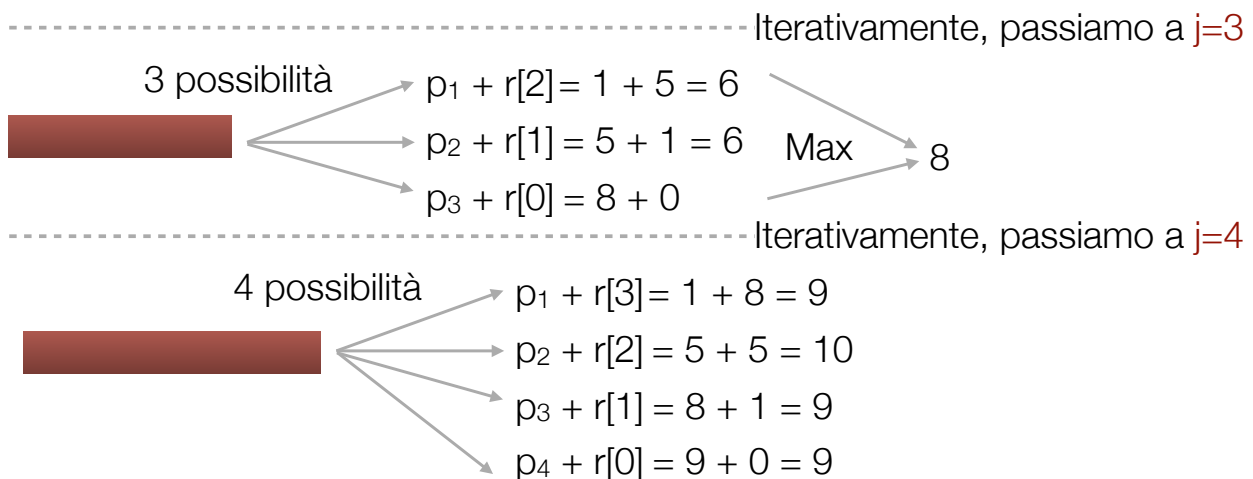
@2020
giuseppe.prencipe@unipi.it

Programmazione Dinamica — **bottom-up**

$r = [0, 1, 5, 8, 0]$

Taglia(4)

$p = [1, 5, 8, 9]$



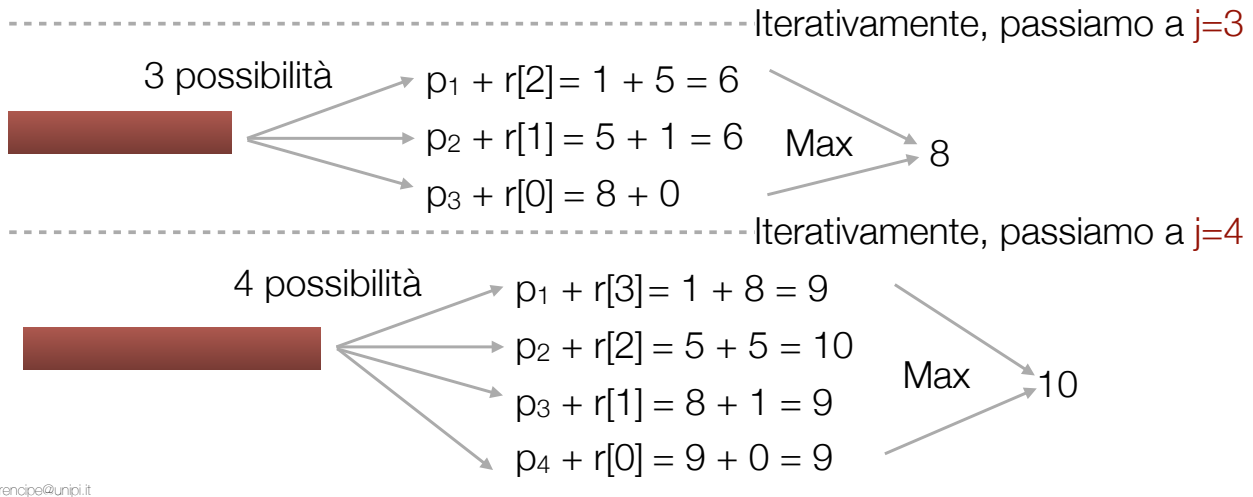
@2020
giuseppe.prencipe@unipi.it

Programmazione Dinamica — **bottom-up**

$r = [0, 1, 5, 8, 0]$

Taglia(4)

$p = [1, 5, 8, 9]$



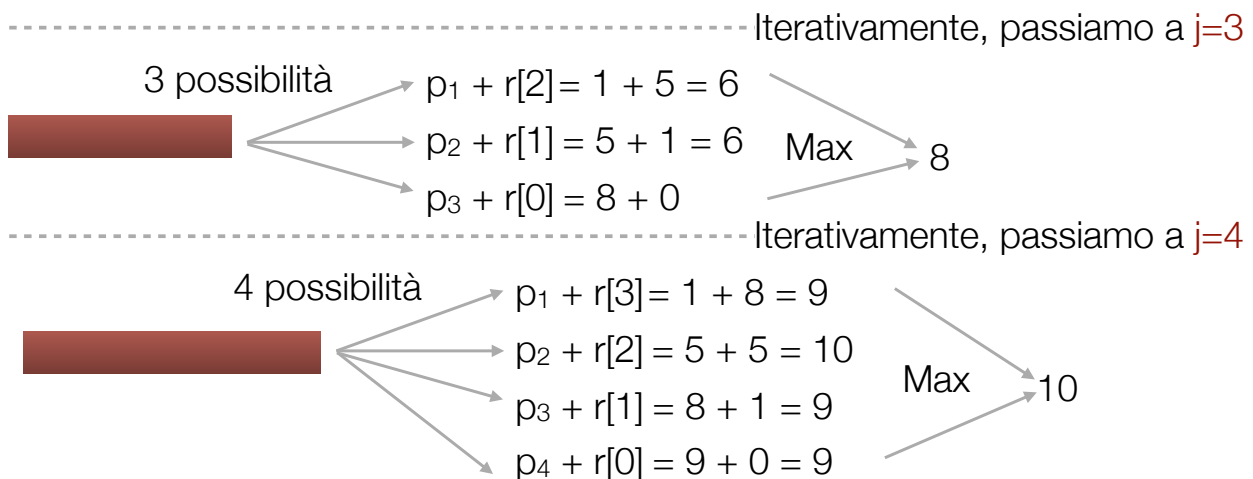
@2020
giuseppe.prencipe@unipi.it

Programmazione Dinamica — **bottom-up**

$r = [0, 1, 5, 8, 10]$

Taglia(4)

$p = [1, 5, 8, 9]$



@2020
giuseppe.prencipe@unipi.it

Programmazione Dinamica — **bottom-up**

TAGLIO-Dinamico(p,n)

$r[0..n]$

$r[0]=0$

for $j=1$ to n

$q=-\infty$

 for $i=1$ to j

 if $q < p[i]+r[j-i]$

$q = p[i]+r[j-i]$

$r[j]=q$

return r

E ricordiamo anche la
definizione generale del
problema di ottimizzazione

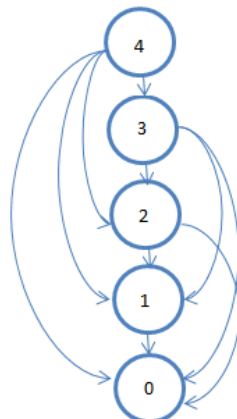
$$r_n = \max\{p_i + r_{n-i}, 1 \leq i \leq n\}$$

@2020
giuseppe.prencipe@unipi.it

Programmazione Dinamica — **bottom-up**

Ogni sotto-problema viene risolto una volta sola, le sotto-soluzioni ottime memorizzate in un array, e utilizzate quando necessario

Il tempo **esponenziale** della soluzione ricorsiva diventa così **polinomiale**



@2020
giuseppe.prencipe@unipi.it

Recuperare la soluzione ottima

```
TAGLIO-Dinamico(p,n)
  r[0..n]
  r[0]=0
  for j=1 to n
    q=-∞
    for i=1 to j
      if q < p[i]+r[j-i]
        q= p[i]+r[j-i]
    r[j]=q
  return r
```

@2020
giuseppe.prencipe@unipi.it

L'array r ci permette di
calcolare il valore ottimo

**In quale posizione di r si
trova l'ottimo del problema
di dimensione n?**

Recuperare la soluzione ottima

```
TAGLIO-Dinamico(p,n)
  r[0..n]
  r[0]=0
  for j=1 to n
    q=-∞
    for i=1 to j
      if q < p[i]+r[j-i]
        q= p[i]+r[j-i]
    r[j]=q
  return r
```

@2020
giuseppe.prencipe@unipi.it

L'array r ci permette di
calcolare il valore ottimo

**In quale posizione di r si
trova l'ottimo del problema
di dimensione n?**

r = [0, 1, 5, 8, 10]



Recuperare la soluzione ottima

```
TAGLIO-Dinamico(p,n)
  r[0..n]
  r[0]=0
  for j=1 to n
    q=-∞
    for i=1 to j
      if q < p[i]+r[j-i]
        q= p[i]+r[j-i]
    r[j]=q
  return r
```

@2020
giuseppe.prencipe@unipi.it

L'array r ci permette di
calcolare il valore ottimo

In quale posizione di r si trova
l'ottimo del problema di
dimensione n?

**Come facciamo se
vogliamo sapere dove
tagliare la corda per avere
il valore ottimo?**

Recuperare la soluzione ottima

```
TAGLIO-Dinamico(p,n)
  r[0..n]
  r[0]=0
  for j=1 to n
    q=-∞
    for i=1 to j
      if q < p[i]+r[j-i]
        q= p[i]+r[j-i]
    r[j]=q
  return r
```

@2020
giuseppe.prencipe@unipi.it

**Come facciamo se
vogliamo sapere dove
tagliare la corda per avere
il valore ottimo?**

Recuperare la soluzione ottima

```
TAGLIO-Dinamico(p,n)
  r[0..n]
  r[0]=0
  for j=1 to n
    q=-∞
    for i=1 to j
      if q < p[i]+r[j-i]
        q= p[i]+r[j-i]
    r[j]=q
  return r
```

@2020
giuseppe.prencipe@unipi.it

Come facciamo se vogliamo sapere dove tagliare la corda per avere il valore ottimo?

Ogni volta che troviamo un ottimo per un sotto-problema, dobbiamo ricordarci la posizione del taglio

Come modifichiamo l'algoritmo?

Recuperare la soluzione ottima

```
TAGLIO-Dinamico(p,n)
  r[0..n] and s[0..n] arrays
  r[0]=0
  for j=1 to n
    q=-∞
    for i=1 to j
      if q < p[i]+r[j-i]
        s[j]=i; q= p[i]+r[j-i]
    r[j]=q
  return r and s
```

@2020
giuseppe.prencipe@unipi.it

Ogni volta che troviamo un ottimo per un sotto-problema, dobbiamo ricordarci la posizione del taglio

Come modifichiamo l'algoritmo?

Utilizziamo un secondo array, s[]

Recuperare la soluzione ottima

```
STAMPA-TAGLIO(n)
  (r,s) = TAGLIO-Dinamico(p,n)
  while n>0
    print s[n]
    n=n-s[n]
```

@2020
giuseppe.prencipe@unipi.it

Recuperare la soluzione ottima

```
STAMPA-TAGLIO(n)
  (r,s) = TAGLIO-Dinamico(p,n)
  while n>0
    print s[n]
    n=n-s[n]
```

Esempio:

p = [1, 5, 8, 9]	i	0	1	2	3	4
	r[i]	0	1	5	8	10
	s[i]	0	1	2	3	2

@2020
giuseppe.prencipe@unipi.it

Recuperare la soluzione ottima

STAMPA-TAGLIO(n)

```
(r,s) = TAGLIO-Dinamico(p,n)
while n>0
    print s[n]
    n=n-s[n]
```

Esempio:

p = [1, 5, 8, 9]

i	0	1	2	3	4
r[i]	0	1	5	8	10
s[i]	0	1	2	3	2

n = 4  →

Recuperare la soluzione ottima

STAMPA-TAGLIO(n)

```
(r,s) = TAGLIO-Dinamico(p,n)
while n>0
    print s[n]
    n=n-s[n]
```

Esempio:

p = [1, 5, 8, 9]

i	0	1	2	3	4
r[i]	0	1	5	8	10
s[i]	0	1	2	3	2

n = 4  ^{s[4]}
→

Recuperare la soluzione ottima

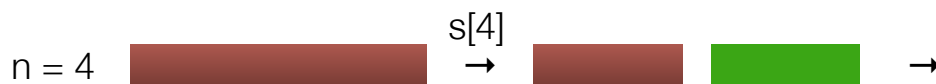
STAMPA-TAGLIO(n)

```
(r,s) = TAGLIO-Dinamico(p,n)
while n>0
    print s[n]
    n=n-s[n]
```

Esempio:

p = [1, 5, 8, 9]

i	0	1	2	3	4
r[i]	0	1	5	8	10
s[i]	0	1	2	3	2



Recuperare la soluzione ottima

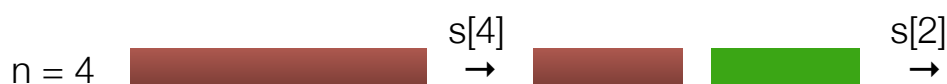
STAMPA-TAGLIO(n)

```
(r,s) = TAGLIO-Dinamico(p,n)
while n>0
    print s[n]
    n=n-s[n]
```

Esempio:

p = [1, 5, 8, 9]

i	0	1	2	3	4
r[i]	0	1	5	8	10
s[i]	0	1	2	3	2



Recuperare la soluzione ottima

STAMPA-TAGLIO(n)

```
(r,s) = TAGLIO-Dinamico(p,n)
while n>0
    print s[n]
    n=n-s[n]
```

Esempio:

p = [1, 5, 8, 9]

i	0	1	2	3	4
r[i]	0	1	5	8	10
s[i]	0	1	2	3	2



Recuperare la soluzione ottima

STAMPA-TAGLIO(n)

```
(r,s) = TAGLIO-Dinamico(p,n)
while n>0
    print s[n]
    n=n-s[n]
```

Esempio:

p = [1, 5, 8, 9]

i	0	1	2	3	4
r[i]	0	1	5	8	10
s[i]	0	1	2	3	2



Recuperare la soluzione ottima

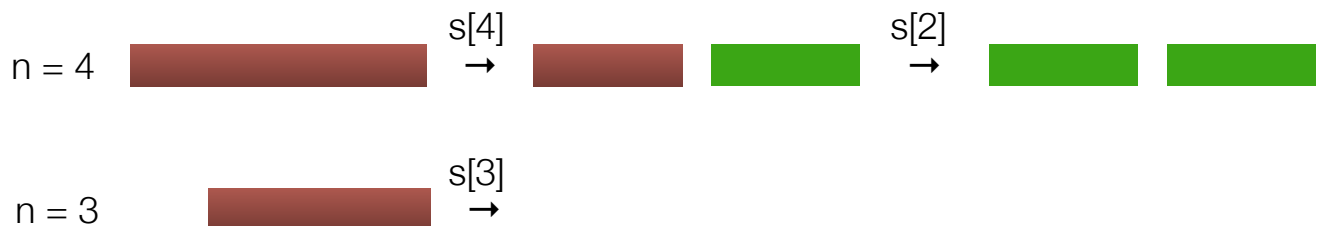
STAMPA-TAGLIO(n)

```
(r,s) = TAGLIO-Dinamico(p,n)
while n>0
  print s[n]
  n=n-s[n]
```

Esempio:

p = [1, 5, 8, 9]

i	0	1	2	3	4
r[i]	0	1	5	8	10
s[i]	0	1	2	3	2



Recuperare la soluzione ottima

STAMPA-TAGLIO(n)

```
(r,s) = TAGLIO-Dinamico(p,n)
while n>0
  print s[n]
  n=n-s[n]
```

Esempio:

p = [1, 5, 8, 9]

i	0	1	2	3	4
r[i]	0	1	5	8	10
s[i]	0	1	2	3	2



Coefficienti binomiali

Coefficienti binomiali

Coefficienti binomiali

- Il coefficiente binomiale $C(n, k)$ è il numero di modi che si hanno per scegliere un sottoinsieme di k elementi da un insieme di n elementi
- Per definizione,

Coefficienti binomiali

- Il coefficiente binomiale $C(n, k)$ è il numero di modi che si hanno per scegliere un sottoinsieme di k elementi da un insieme di n elementi
- Per definizione,

$$C(n,k) = n! / ((n-k)! * k!)$$

Coefficienti binomiali

- Il coefficiente binomiale $C(n, k)$ è il numero di modi che si hanno per scegliere un sottoinsieme di k elementi da un insieme di n elementi
- Per definizione,

$$C(n, k) = n! / ((n-k)! * k!)$$

- Questa formula non viene utilizzata per il calcolo effettivo, perché, anche per piccoli valori di n , **calcolare $n!$ è estremamente costoso**
- Si utilizza invece la seguente formula per il calcolo di $C(n, k)$

Coefficienti binomiali

- Il coefficiente binomiale $C(n, k)$ è il numero di modi che si hanno per scegliere un sottoinsieme di k elementi da un insieme di n elementi
- Per definizione,

$$C(n, k) = n! / ((n-k)! * k!)$$

- Questa formula non viene utilizzata per il calcolo effettivo, perché, anche per piccoli valori di n , **calcolare $n!$ è estremamente costoso**
- Si utilizza invece la seguente formula per il calcolo di $C(n, k)$

$$C(n, k) = C(n-1, k-1) + C(n-1, k)$$

$$C(n, 0) = 1$$

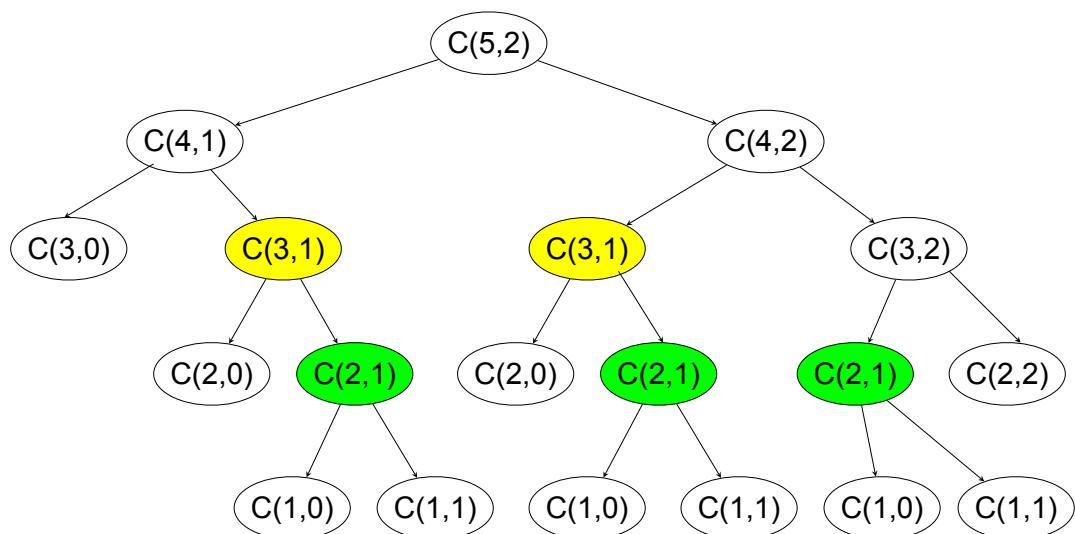
$$C(n, n) = 1$$

Coefficienti binomiali — top-down

```
C(n: int, k: int): float {  
    if ((k==0) || (k==n))  
        return 1;  
    else  
        return C(n - 1, k) + C(n - 1, k - 1);  
}
```

@2020
giuseppe.prencipe@unipi.it

Coefficienti binomiali – albero ricorsione per $C(5,2)$



@2020
giuseppe.prencipe@unipi.it

$$T(n, k) = C(n, k) = \frac{n!}{k!(n-k)!}$$

Caso peggiore con $k=n/2$

$$T(n, n/2) = \frac{n!}{(n/2)!(n/2)!} \Rightarrow O\left(\frac{2^n}{\sqrt{n}}\right)$$

Programmazione Dinamica: Memoization

- Il primo passo che possiamo compiere è quello di evitare di calcolare più volte gli stessi valori
- Usiamo una tabella di **programmazione dinamica**
 - Questa tecnica viene chiamata anche **memoization**
 - **Memoization** (non memorization): il termine deriva da memo (memorandum)

Coefficienti binomiali – memoization

```
ResultEntry {  
    done: boolean,  
    value: float  
}
```

```
ResultEntry[n+1][k+1] result;
```

Risultati sotto-problemi in una **tabella**:
result[i][j] rappresenta C(i,j)
Tutte le posizioni della tabella sono **inizializzate**
con result[i][j].done=false

@2020
giuseppe.precipe@unipi.it

Coefficienti binomiali – memoization (sempre ricorsivo)

```
C(n: int, k: int): float {
```

```
C(n,k)=C(n-1, k-1)+C(n-1, k)  
C(n,0)=1  
C(n,n)=1
```

```
}
```

@2020
giuseppe.precipe@unipi.it

Coefficienti binomiali – memoization (sempre ricorsivo)

```
C(n: int, k: int): float {
```

```
    if (valore già in ResultEntry????)
        ????
```

```
    if (valori n, k per cui C(n,k) si sa fare????) {
        ????
```

```
        per gli altri casi?
```

```
    }
    @2020
    giuseppe.prencipe@unipi.it
```

```
C(n,k)=C(n-1, k-1)+C(n-1, k)
C(n,0)=1
C(n,n)=1
```

Coefficienti binomiali – memoization (sempre ricorsivo)

```
C(n: int, k: int): float {
```

```
    if (result[n][k].done == true)
        return result[n][k].value;
```

```
    if (valori n, k per cui C(n,k) si sa fare????) {
        ????
```

```
        per gli altri casi?
```

```
    }
    @2020
    giuseppe.prencipe@unipi.it
```

```
C(n,k)=C(n-1, k-1)+C(n-1, k)
C(n,0)=1
C(n,n)=1
```

Coefficienti binomiali – memoization (sempre ricorsivo)

```
C(n: int, k: int): float {  
  
    if (result[n][k].done == true)  
        return result[n][k].value;  
  
    if ((k == 0) || (k == n)) {  
        ???  
    }  
}
```

$C(n,k)=C(n-1, k-1)+C(n-1, k)$
 $C(n,0)=1$
 $C(n,n)=1$

per gli altri casi?

@2020
giuseppe.precipe@unipi.it

Coefficienti binomiali – memoization (sempre ricorsivo)

```
C(n: int, k: int): float {  
  
    if (result[n][k].done == true)  
        return result[n][k].value;  
  
    if ((k == 0) || (k == n)) {  
        result[n][k].done = true;  
        result[n][k].value = 1;  
        return result[n][k].value;  
    }  
}
```

$C(n,k)=C(n-1, k-1)+C(n-1, k)$
 $C(n,0)=1$
 $C(n,n)=1$

per gli altri casi?

@2020
giuseppe.precipe@unipi.it

Coefficienti binomiali – memoization (sempre ricorsivo)

```
C(n: int, k: int): float {  
  
    if (result[n][k].done == true)  
        return result[n][k].value;  
  
    if ((k == 0) || (k == n)) {  
        result[n][k].done = true;  
        result[n][k].value = 1;  
        return result[n][k].value;  
    }  
  
    result[n][k].done = true;  
    result[n][k].value = C(n - 1, k) + C(n - 1, k - 1);  
    return result[n][k].value;  
}
```

@2020
giuseppe.prence@unipi.it

$C(n,k)=C(n-1, k-1)+C(n-1, k)$
 $C(n,0)=1$
 $C(n,n)=1$

Coefficienti binomiali – memoization (sempre ricorsivo)

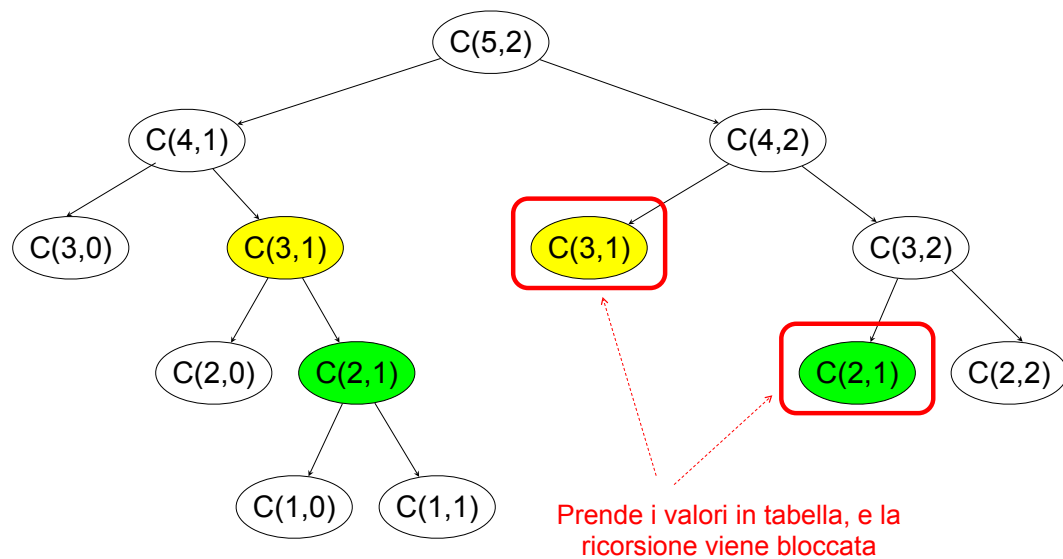
```
C(n: int, k: int): float {  
  
    if (result[n][k].done == true)  
        return result[n][k].value;  
  
    if ((k == 0) || (k == n)) {  
        result[n][k].done = true;  
        result[n][k].value = 1;  
        return result[n][k].value;  
    }  
  
    result[n][k].done = true;  
    result[n][k].value = C(n - 1, k) + C(n - 1, k - 1);  
    return result[n][k].value;  
}
```

@2020
giuseppe.prence@unipi.it

$C(n,k)=C(n-1, k-1)+C(n-1, k)$
 $C(n,0)=1$
 $C(n,n)=1$

Non ricalcola, ma
accede valori in tabella

Coefficienti binomiali – albero ricorsione con memoization



@2020
giuseppe.prencipe@unipi.it

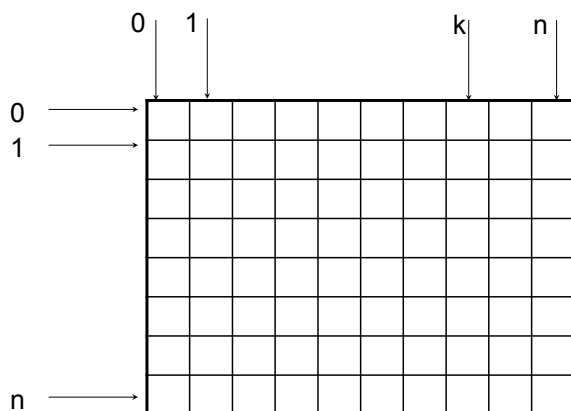
Programmazione Dinamica

- Eliminiamo ora la ricorsione, e sostituiamola con iterazione
- Bisogna capire come inserire i valori in tabella in modo che la tabella sia utile
 - **Programmazione Dinamica**

@2020
giuseppe.prencipe@unipi.it

Coefficienti binomiali — bottom-up, riempire la tabella

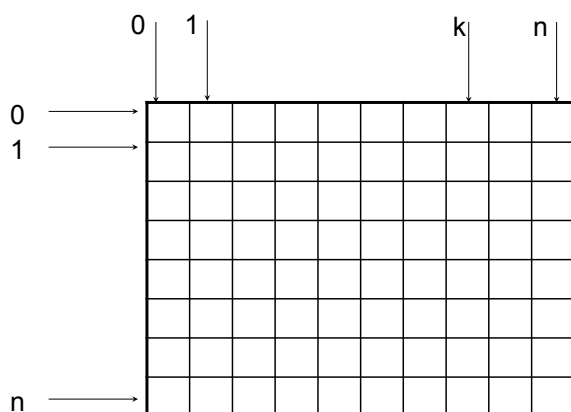
- `result[i][j]` memorizza il valore di $C(i,j)$
- La tabella ha $n+1$ righe and $k+1$ colonne, $k \leq n$
- **Inizializzazione:** Quali valori possiamo inserire già all'inizio? In altre parole, di quali sotto-problemi conosciamo già la soluzione?



@2020
giuseppe.precipe@unipi.it

Coefficienti binomiali — bottom-up, riempire la tabella

- `result[i][j]` memorizza il valore di $C(i,j)$
- La tabella ha $n+1$ righe and $k+1$ colonne, $k \leq n$
- **Inizializzazione:** Quali valori possiamo inserire già all'inizio? In altre parole, di quali sotto-problemi conosciamo già la soluzione?

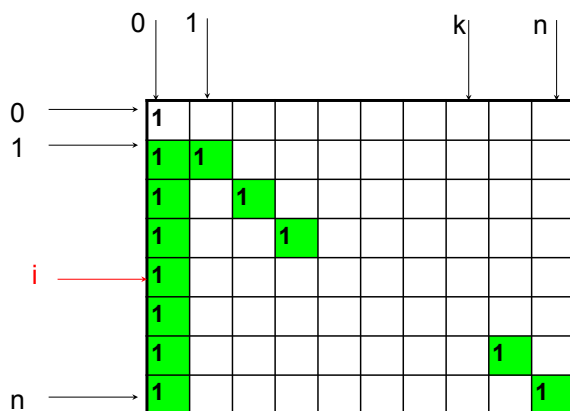


$$\begin{aligned} C(n,k) &= C(n-1, k-1) + C(n-1, k) \\ C(n,0) &= 1 \\ C(n,n) &= 1 \end{aligned}$$

@2020
giuseppe.precipe@unipi.it

Coefficienti binomiali — bottom-up, riempire la tabella

- `result[i][j]` memorizza il valore di $C(i,j)$
- La tabella ha $n+1$ righe and $k+1$ colonne, $k \leq n$
- **Inizializzazione:** $C(i,0)=1$ e $C(i,i)=1$, per $1 \leq i \leq n$

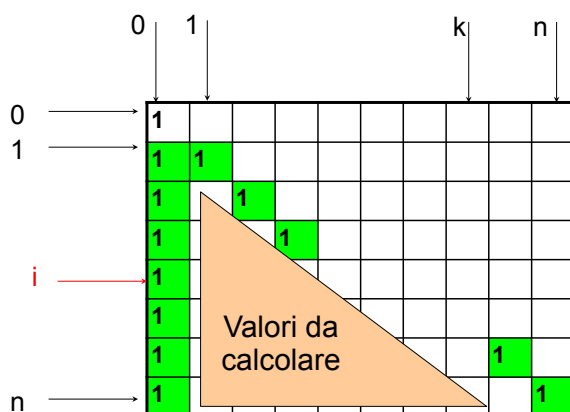


$$C(n,k)=C(n-1, k-1)+C(n-1, k)$$
$$C(n,0)=1$$
$$C(n,n)=1$$

@2020
giuseppe.precipe@unipi.it

Coefficienti binomiali — bottom-up, riempire la tabella

- `result[i][j]` memorizza il valore di $C(i,j)$
- La tabella ha $n+1$ righe and $k+1$ colonne, $k \leq n$
- **Inizializzazione:** $C(i,0)=1$ e $C(i,i)=1$, per $1 \leq i \leq n$

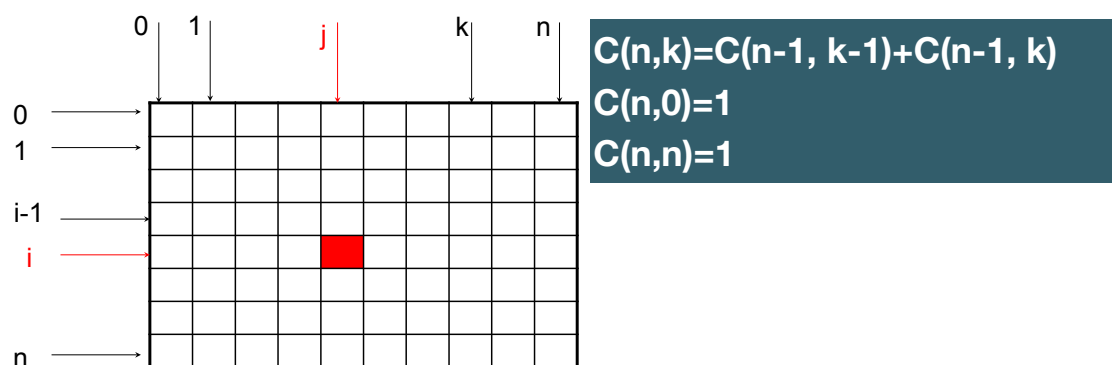


$$C(n,k)=C(n-1, k-1)+C(n-1, k)$$
$$C(n,0)=1$$
$$C(n,n)=1$$

@2020
giuseppe.precipe@unipi.it

Coefficienti binomiali — bottom-up, riempire la tabella

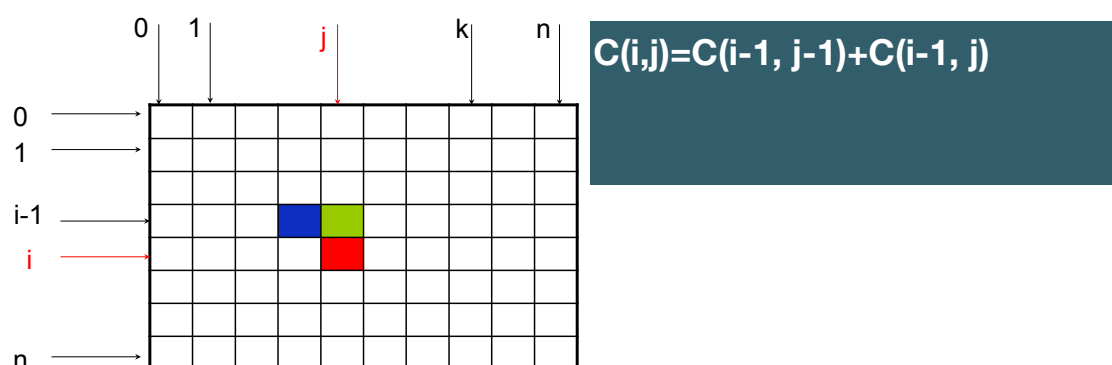
- `result[i][j]` memorizza il valore di $C(i,j)$
- Gli altri elementi (i,j) come li calcoliamo?



@2020
giuseppe.precipe@unipi.it

Coefficienti binomiali — bottom-up, riempire la tabella

- `result[i][j]` memorizza il valore di $C(i,j)$
- Il resto dei valori (i,j) , $2 \leq i \leq n$ e $1 \leq j \leq i-1$ sono calcolati utilizzando i valori $(i-1, j-1)$ e $(i-1, j)$



@2020
giuseppe.precipe@unipi.it

Coefficienti binomiali — bottom-up

```
result: float[][][];  
  
C(n: int, k: int): float {  
    result: float[n + 1][n + 1];  
    int i, j;  
    inizializzazione
```

riempimento tabella prog. din.

```
    return result[n][k];
```

```
    }  
    @2020  
    giuseppe.prencipe@unipi.it
```

```
C(i,j)=C(i-1, j-1)+C(i-1, j)  
C(n,k)=C(n-1, k-1)+C(n-1, k)  
C(n,0)=1  
C(n,n)=1
```

Coefficienti binomiali — bottom-up

```
result: float[][][];  
  
C(n: int, k: int): float {  
    result: float[n + 1][n + 1];  
    int i, j;  
    for (i=0; i<=n; i++) {  
        result[i][0]=1;  
        result[i][i]=1;  
    }  
    riempimento tabella prog. din.
```

```
    return result[n][k];
```

```
    }  
    @2020  
    giuseppe.prencipe@unipi.it
```

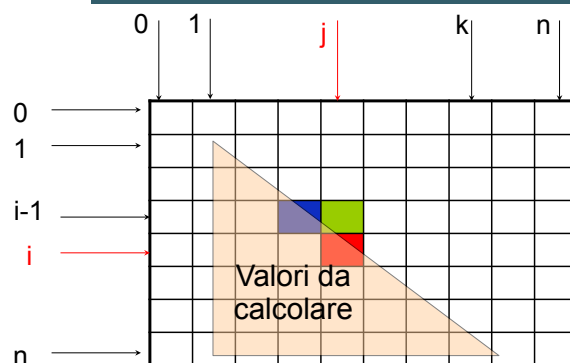
```
C(i,j)=C(i-1, j-1)+C(i-1, j)  
C(n,k)=C(n-1, k-1)+C(n-1, k)  
C(n,0)=1  
C(n,n)=1
```

Coefficienti binomiali — bottom-up

```
result: float[][][];  
  
C(n: int, k: int): float {  
    result: float[n + 1][n + 1];  
    int i, j;  
    for (i=0; i<=n; i++) {  
        result[i][0]=1;  
        result[i][i]=1;  
    }  
    riempimento tabella prog. din.  
    Complete!!!!  
  
    return result[n][k];  
}
```

@2020
giuseppe.prencipe@unipi.it

$C(i,j)=C(i-1,j-1)+C(i-1,j)$
 $C(n,k)=C(n-1,k-1)+C(n-1,k)$
 $C(n,0)=1$
 $C(n,n)=1$



Coefficienti binomiali — bottom-up

```
result: float[][][];  
  
C(n: int, k: int): float {  
    result: float[n + 1][n + 1];  
    int i, j;  
    for (i=0; i<=n; i++) {  
        result[i][0]=1;  
        result[i][i]=1;  
    }  
    for (i=2; i<=n; i++)  
        for (j=1; j<i; j++)  
            result[i][j]=result[i-1][j-1]+result[i-1][j];  
  
    return result[n][k];  
}
```

@2020
giuseppe.prencipe@unipi.it

$C(i,j)=C(i-1,j-1)+C(i-1,j)$
 $C(n,k)=C(n-1,k-1)+C(n-1,k)$
 $C(n,0)=1$
 $C(n,n)=1$

Coefficienti binomiali — bottom-up

```
result: float[][];  
  
C(n: int, k: int): float {  
    result: float[n + 1][n + 1];  
    int i, j;  
    for (i=0; i<=n; i++) {  
        result[i][0]=1;  
        result[i][i]=1;  
    }  
    for (i=2; i<=n; i++)  
        for(j=1; j<i; j++)  
            result[i][j]=result[i-1][j-1]+result[i-1][j];  
  
    return result[n][k];  
}
```

@2020
giuseppe.prencipe@unipi.it

Coefficienti binomiali — bottom-up

```
result: float[][];  
  
C(n: int, k: int): float {  
    result: float[n + 1][n + 1];  
    int i, j;  
    for (i=0; i<=n; i++) {  
        result[i][0]=1;  
        result[i][i]=1;  
    }  
    for (i=2; i<=n; i++)  
        for(j=1; j<i; j++)  
            result[i][j]=result[i-1][j-1]+result[i-1][j];  
  
    return result[n][k];  
}
```

Tempo: ????
Spazio: ????

@2020
giuseppe.prencipe@unipi.it

Coefficienti binomiali — bottom-up

```
result: float[][];
```

```
C(n: int, k: int): float {
```

```
    result: float[n + 1][n + 1];
```

```
    int i, j;
```

```
    for (i=0; i<=n; i++) {
```

```
        result[i][0]=1;
```

```
        result[i][i]=1;
```

```
    }
```

```
    for (i=2; i<=n; i++)
```

```
        for(j=1; j<i; j++)
```

```
            result[i][j]=result[i-1][j-1]+result[i-1][j];
```

```
    return result[n][k];
```

```
}
```

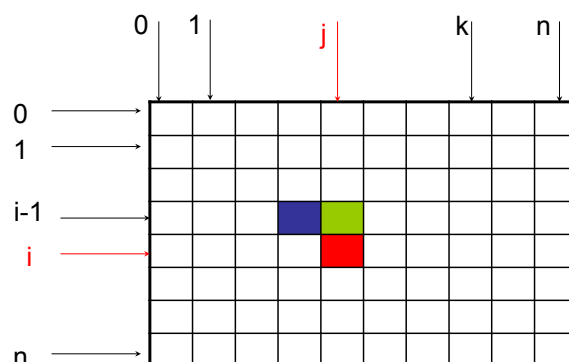
@2020
giuseppe.prencipe@unipi.it

Tempo: $O(n*n)$ (o $O(n*k)$)

Spazio: $O(n*n)$ (o $O(n*k)$)

Coefficienti binomiali — spazio

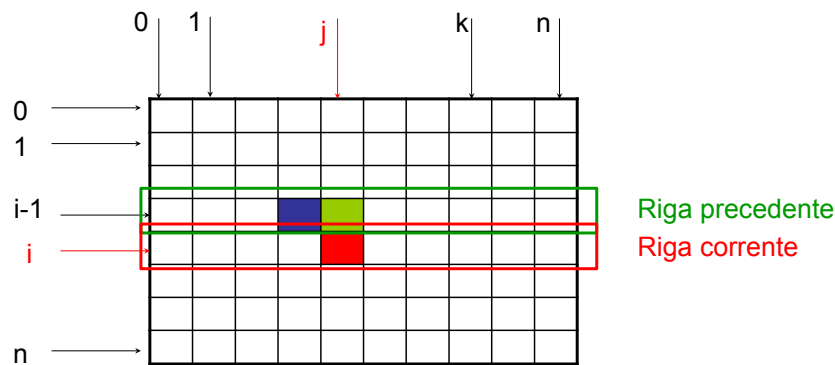
- Di quanti valori c'è effettivamente bisogno per calcolare ogni (i,j) ?
- Abbiamo bisogno di tutta la tabella in ogni momento?



@2020
giuseppe.prencipe@unipi.it

Coefficienti binomiali — spazio

- Di quanti valori c'è effettivamente bisogno per calcolare ogni (i,j) ?
- Abbiamo bisogno di tutta la tabella in ogni momento?



@2020
giuseppe.prencipe@unipi.it

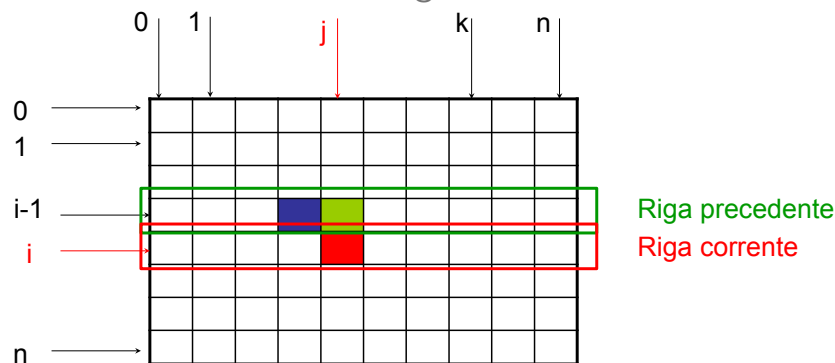
Programmazione Dinamica — migliorare uso memoria

- In molti algoritmi di Programmazione Dinamica, non è davvero necessario memorizzare **TUTTI** i valori della tabella
- Ogni passo (sotto-problema) generalmente dipende solo da un ridotto insieme di sotto-problemi (e non da tutti)
- Quindi, è possibile in genere sostituire la tabella intera con un **buffer di memoria più piccolo** che viene aggiornato durante la computazione

@2020
giuseppe.prencipe@unipi.it

Coefficienti binomiali — spazio

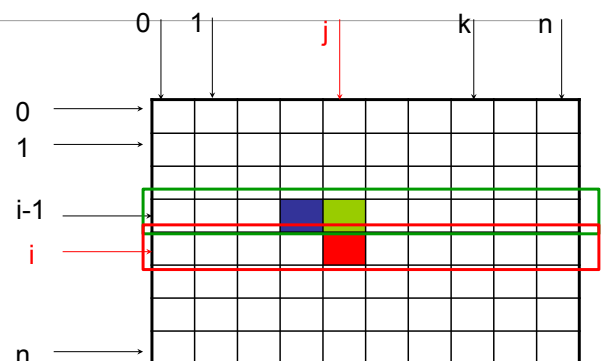
- Ad ogni iterazione `for i`, per calcolare i valori della riga corrente abbiamo bisogno solo dei valori della riga precedente
- Quindi, 2 righe sono sufficienti come buffer di memoria
- Questi buffer sono *riutilizzati* ad ogni iterazione



@2020
giuseppe.precipe@unipi.it

Coefficienti binomiali — bottom-up, spazio efficiente

```
C(n: int, k: int): float {  
    result1: float[n + 1];  
    result2: float[n + 1];  
    result1[0] = 1;  
    result1[1] = 1;  
  
    for (int i = 2; i <= n; i++) {  
        ????  
        for (int j = 1; j < i; j++)  
            ????  
  
        ????  
        ????  
        ????  
        ????  
    }  
    return result1[k];  
}
```

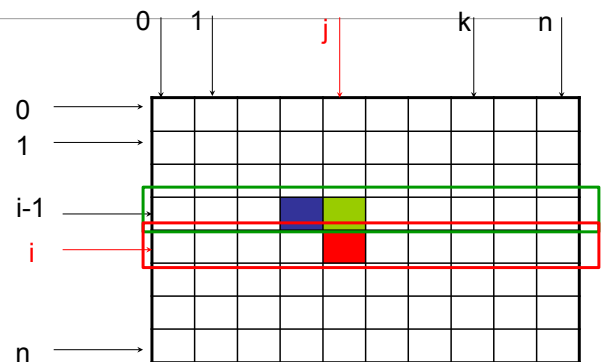


@2020
giuseppe.precipe@unipi.it

Coefficienti binomiali — bottom-up, spazio efficiente

```
C(n: int, k: int): float {  
    result1: float[n + 1];  
    result2: float[n + 1];  
    result1[0] = 1;  
    result1[1] = 1;  
  
    for (int i = 2; i <= n; i++) {  
        result2[0] = 1;  
        for (int j = 1; j < i; j++)  
            result2[j] = result1[j - 1] + result1[j];  
        result2[i] = 1;  
        auxi: float[] = result1;  
        result1 = result2;  
        result2 = auxi;  
    }  
    return result1[k];  
}
```

@2020
giuseppe.precipe@unipi.it



Coefficienti binomiali — bottom-up, spazio efficiente

```
C(n: int, k: int): float {  
    result1: float[n + 1];  
    result2: float[n + 1];  
    result1[0] = 1;  
    result1[1] = 1;  
  
    for (int i = 2; i <= n; i++) {  
        result2[0] = 1;  
        for (int j = 1; j < i; j++)  
            result2[j] = result1[j - 1] + result1[j];  
        result2[i] = 1;  
        auxi: float[] = result1;  
        result1 = result2;  
        result2 = auxi;  
    }  
    return result1[k];  
}
```

@2020
giuseppe.precipe@unipi.it

Tempo: ????

Spazio: ????

Coefficienti binomiali — bottom-up, spazio efficiente

```
C(n: int, k: int): float {
```

```
    result1: float[n + 1];  
    result2: float[n + 1];  
    result1[0] = 1;  
    result1[1] = 1;
```

Tempo: $O(n*n)$ (or $O(n*k)$)
Spazio: $O(n)$ (or $O(k)$)

```
    for (int i = 2; i <= n; i++) {  
        result2[0] = 1;  
        for (int j = 1; j < i; j++)  
            result2[j] = result1[j - 1] + result1[j];  
        result2[i] = 1;  
        auxi: float[] = result1;  
        result1 = result2;  
        result2 = auxi;  
    }  
    return result1[k];
```

```
@2020 }  
giuseppe.precipe@unipi.it
```

Programmazione Dinamica in 4 passi

1. **Caratterizzare** la struttura di una soluzione ottima
2. **Ricorsivamente** definire il valore di una soluzione ottima
3. **Calcolare** il valore di una soluzione ottima in modo bottom-up
4. **Costruire** gli elementi della soluzione ottima dalle informazioni calcolate (non sempre richiesto)

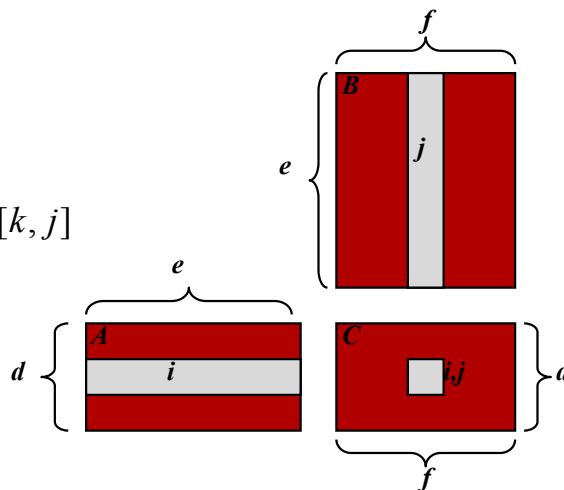
Concatenazione del prodotto di matrici

Sequenza ottima di moltiplicazioni di matrici

- Moltiplicazione di matrici

- $C = A * B$
- $A: d \times e$ e $B: e \times f$
- $O(d \cdot e \cdot f)$ tempo

$$C[i, j] = \sum_{k=0}^{e-1} A[i, k] * B[k, j]$$



Sequenza ottima di moltiplicazioni di matrici

Vogliamo calcolare il prodotto di n matrici A_i di taglia $d_i \times d_{i+1}$

$$A^* = A_0 \times A_1 \times \cdots \times A_{n-1}$$

Hp. semplificativa: il costo di moltiplicazione di due matrici di taglia $r \times s$ e $s \times t$ è pari a $r \times s \times t$

Sequenza ottima di moltiplicazioni di matrici

- **Sequenza di moltiplicazione di matrici:**
 - Calcolare $A = A_0 * A_1 * \dots * A_{n-1}$
 - A_i è $d_i \times d_{i+1}$
 - Problema: come moltiplicare efficientemente?
- Esempio
 - B è 3×100
 - C è 100×5
 - D è 5×5
 - $(B * C) * D$ abbiamo **quante operazioni????**
 - $B * (C * D)$ abbiamo **quante operazioni????**

Sequenza ottima di moltiplicazioni di matrici

- **Sequenza di moltiplicazione di matrici:**
 - Calcolare $A = A_0 * A_1 * \dots * A_{n-1}$
 - A_i è $d_i \times d_{i+1}$
 - Problema: come moltiplicare efficientemente?
- Esempio
 - B è 3×100
 - C è 100×5
 - D è 5×5
 - $(B * C) * D$ abbiamo $1500 + 75 = 1575$ operazioni
 - $B * (C * D)$ abbiamo **quante operazioni???**

Sequenza ottima di moltiplicazioni di matrici

- **Sequenza di moltiplicazione di matrici:**
 - Calcolare $A = A_0 * A_1 * \dots * A_{n-1}$
 - A_i è $d_i \times d_{i+1}$
 - Problema: come moltiplicare efficientemente?
- Esempio
 - B è 3×100
 - C è 100×5
 - D è 5×5
 - $(B * C) * D$ abbiamo $1500 + 75 = 1575$ operazioni
 - $B * (C * D)$ abbiamo $1500 + 2500 = 4000$ operazioni

Sequenza ottima di moltiplicazioni di matrici

Vogliamo calcolare il prodotto di n matrici A_i di taglia $d_i \times d_{i+1}$

$$A^* = A_0 \times A_1 \times \dots \times A_{n-1}$$

Hp. semplificativa: *il costo di moltiplicazione di due matrici di taglia $r \times s$ e $s \times t$ è pari a $r \times s \times t$*

Esempio: $d_0=100, d_1=20, d_2=1000, d_3=2, d_4=50$

$(A_0 \times (A_1 \times (A_2 \times A_3)))$	$d_2 d_3 d_4 + d_1 d_2 d_4 + d_0 d_1 d_4 = 1.200.000$
$(A_0 \times ((A_1 \times A_2) \times A_3))$	$d_1 d_2 d_3 + d_1 d_3 d_4 + d_0 d_1 d_4 = 142.000$
$((A_0 \times A_1) \times (A_2 \times A_3))$	$d_0 d_1 d_2 + d_2 d_3 d_4 + d_0 d_2 d_4 = 7.100.000$
$((A_0 \times A_1) \times A_2) \times A_3$	$d_0 d_1 d_2 + d_0 d_2 d_3 + d_0 d_3 d_4 = 2.210.000$
$((A_0 \times (A_1 \times A_2)) \times A_3)$	$d_1 d_2 d_3 + d_0 d_1 d_3 + d_0 d_3 d_4 = 54.000$

@2020
giuseppe.precipe@unipi.it

Approccio enumerativo

- **Algoritmo:**
 - Prova tutti i possibili modi per moltiplicare $A=A_0 \times A_1 \times \dots \times A_{n-1}$
 - Calcola il numero di operazioni per ogni possibile modo
 - Scegli il migliore
- **Costo:**
 - Il numero di possibili modi è uguale al numero di alberi binari con n nodi
 - Questo è **esponenziale!**
 - Si chiama numero di Catalan, ed è circa 4^n
 - **Pessimo** algoritmo!

@2020
giuseppe.precipe@unipi.it

Sequenza ottima di moltiplicazioni

- Il **costo varia** a seconda di come **associamo** il prodotto utilizzando la moltiplicazione tra due matrici: in generale, qual è il **costo minimo**?
- **Nuovo problema:**
 - **input:** sequenza di interi positivi d_0, d_1, \dots, d_n
(dove $d_i \times d_{i+1}$ è la taglia della matrice A_i)
 - **output:** minimo numero di operazioni per calcolare

$$A^* = A_0 \times A_1 \times \dots \times A_{n-1}$$

- (detto **costo minimo** per A^*) con l'ipotesi che la moltiplicazione di due matrici di taglia $r \times s$ e $s \times t$ richieda $r \times s \times t$ operazioni
- **Nota:** il tutto si applica anche con **algoritmi più veloci**

1. Caratterizzare i sotto-problemi ottimi

- Metodo più **efficiente**: per $0 \leq i \leq j \leq n-1$, sia

$$M(i,j) = \text{costo minimo per } A_i \times A_{i+1} \times \dots \times A_j$$

(la cui taglia risultante è $d_i \times d_{j+1}$)

- Il **costo minimo** per A^* è quindi dato da....**????**

1. Caratterizzare i sotto-problemi ottimi

- Metodo più **efficiente**: per $0 \leq i \leq j \leq n-1$, sia

$$M(i,j) = \text{costo minimo per } A_i \times A_{i+1} \times \dots \times A_j$$

(la cui taglia risultante è $d_i \times d_{j+1}$)

- Il **costo minimo** per A^* è quindi dato da **$M(0,n-1)$**

1. Caratterizzare i sotto-problemi ottimi

- Quali sono i sotto-problemi “banali”? Per quali valori di i e j ????

1. Caratterizzare i sotto-problemi ottimi

- **$M(i,i) = 0$ per $0 \leq i \leq n-1$** (per ottenere A_i non dobbiamo moltiplicare)

1. Caratterizzare i sotto-problemi ottimi

- $M(i,i) = 0$ per $0 \leq i \leq n-1$ (per ottenere A_i non dobbiamo moltiplicare)
- **Caso generale:** come possiamo definire i sotto-problemi?

Come definire **ricorsivamente** $A_i \times A_{i+1} \times \dots \times A_j$
in termini di **problemi più piccoli?**

1. Caratterizzare i sotto-problemi ottimi

- $M(i,i) = 0$ per $0 \leq i \leq n-1$ (per ottenere A_i non dobbiamo moltiplicare)
- **Caso generale:** per ogni $i \leq r < j$, vale

$$\underbrace{A_i \times A_{i+1} \times \dots \times A_j}_{\text{dim: ???}} = \underbrace{\phantom{A_i \times A_{i+1} \times \dots \times A_r}}_{\text{dim: ???}} \times \underbrace{\phantom{A_{r+1} \times A_{r+2} \times \dots \times A_j}}_{\text{dim: ???}}$$

↑
costo: ???

Come possiamo dividere $A_i \times A_{i+1} \times \dots \times A_j$????

@2020
giuseppe.precipec@unipi.it

2. Definire la struttura ricorsiva

- $M(i,i) = 0$ per $0 \leq i \leq n-1$ (per ottenere A_i non dobbiamo moltiplicare)
- **Caso generale:** per ogni $i \leq r < j$, vale

$$\underbrace{A_i \times A_{i+1} \times \dots \times A_j}_{\text{dim: ???}} = \underbrace{(A_i \times A_{i+1} \times \dots \times A_r)}_{\text{dim: ???}} \times \underbrace{(A_{r+1} \times A_{r+2} \times \dots \times A_j)}_{\text{dim: ???}}$$

↑
costo: ???

@2020
giuseppe.precipec@unipi.it

2. Definire la struttura ricorsiva

- $M(i,i) = 0$ per $0 \leq i \leq n-1$ (per ottenere A_i non dobbiamo moltiplicare)
- **Caso generale:** per ogni $i \leq r < j$, vale

$$\underbrace{A_i \times A_{i+1} \times \dots \times A_j}_{\text{dim: } d_i \times d_{j+1}} = \underbrace{(A_i \times A_{i+1} \times \dots \times A_r)}_{\text{dim: } \text{????}} \times \underbrace{(A_{r+1} \times A_{r+2} \times \dots \times A_j)}_{\text{dim: } \text{????}}$$

\uparrow
costo: ????

2. Definire la struttura ricorsiva

- $M(i,i) = 0$ per $0 \leq i \leq n-1$ (per ottenere A_i non dobbiamo moltiplicare)
- **Caso generale:** per ogni $i \leq r < j$, vale

$$\underbrace{A_i \times A_{i+1} \times \dots \times A_j}_{\text{dim: } d_i \times d_{j+1}} = \underbrace{(A_i \times A_{i+1} \times \dots \times A_r)}_{\text{dim: } d_i \times d_{r+1}} \times \underbrace{(A_{r+1} \times A_{r+2} \times \dots \times A_j)}_{\text{dim: } \text{????}}$$

\uparrow
costo: ????

2. Definire la struttura ricorsiva

- $M(i,i) = 0$ per $0 \leq i \leq n-1$ (per ottenere A_i non dobbiamo moltiplicare)
- **Caso generale:** per ogni $i \leq r < j$, vale

$$\underbrace{A_i \times A_{i+1} \times \dots \times A_j}_{\text{dim: } d_i \times d_{j+1}} = \underbrace{(A_i \times A_{i+1} \times \dots \times A_r)}_{\text{dim: } d_i \times d_{r+1}} \times \underbrace{(A_{r+1} \times A_{r+2} \times \dots \times A_j)}_{\text{dim: } d_{r+1} \times d_{j+1}}$$

↑
costo: ????

2. Definire la struttura ricorsiva

- $M(i,i) = 0$ per $0 \leq i \leq n-1$ (per ottenere A_i non dobbiamo moltiplicare)
- **Caso generale:** per ogni $i \leq r < j$, vale

$$\underbrace{A_i \times A_{i+1} \times \dots \times A_j}_{\text{dim: } d_i \times d_{j+1}} = \underbrace{(A_i \times A_{i+1} \times \dots \times A_r)}_{\text{dim: } d_i \times d_{r+1}} \times \underbrace{(A_{r+1} \times A_{r+2} \times \dots \times A_j)}_{\text{dim: } d_{r+1} \times d_{j+1}}$$

↑
costo: $d_i \times d_{r+1} \times d_{j+1}$

$$M(i,j) = \boxed{} \quad \boxed{} \quad \boxed{}$$

2. Definire la struttura ricorsiva

- $M(i,i) = 0$ per $0 \leq i \leq n-1$ (per ottenere A_i non dobbiamo moltiplicare)
- **Caso generale:** per ogni $i \leq r < j$, vale

$$\underbrace{A_i \times A_{i+1} \times \dots \times A_j}_{\text{dim: } d_i \times d_{j+1}} = \underbrace{(A_i \times A_{i+1} \times \dots \times A_r)}_{\text{dim: } d_i \times d_{r+1}} \times \underbrace{(A_{r+1} \times A_{r+2} \times \dots \times A_j)}_{\text{dim: } d_{r+1} \times d_{j+1}}$$

↑
costo: $d_i \times d_{r+1} \times d_{j+1}$

$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}$$

Per ottenere il **minimo** per $M(i,j)$, bisogna trovare r che lo minimizza

@2020
giuseppe.precipe@unipi.it

2. Definire la struttura ricorsiva

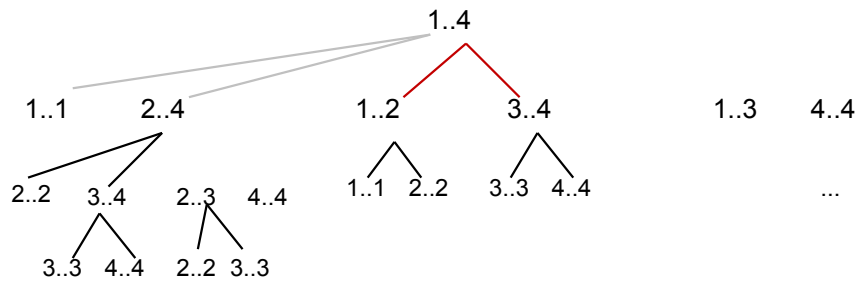
```
1 CostoMinimoRicorsivo( i, j ): ⟨pre:
2   IF (i >= j) RETURN 0;
3   minimo = +∞;
4   FOR (r = i; r < j; r = r+1) {
5     costo = CostoMinimoRicorsivo( i, r );
6     costo = costo + CostoMinimoRicorsivo( r+1, j );
7     costo = costo + d[i] × d[r+1] × d[j+1];
8     IF (costo < minimo) minimo = costo;
9   }
10  RETURN minimo;
```

$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}$$

Per ottenere il **minimo** per $M(i,j)$, bisogna trovare r che lo minimizza

@2020
giuseppe.precipe@unipi.it

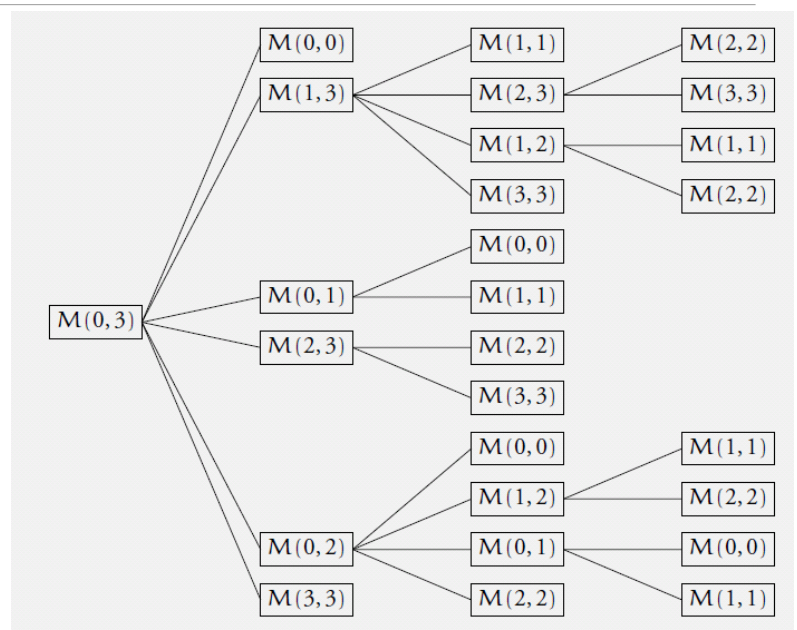
2. Definire la struttura ricorsiva — top-down (overlap delle soluzioni)



@2020
giuseppe.prencipe@unipi.it

2. Definire la struttura ricorsiva — top-down (overlap delle soluzioni)

Numero esponenziale di chiamate ricorsive, ciascuna invocata molte volte con gli stessi parametri d'ingresso



@2020
giuseppe.prencipe@unipi.it

2. Definire la struttura ricorsiva — top-down (overlap delle soluzioni)

$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}$$

- Quindi, indicando con $T(n)$ la complessità del calcolo di $M(0,n-1)$,
 - Qual è l'equazione di ricorrenza? (*ricordate che $M(i,j)$ prova tutti gli r*)

$$T(n) = c' + \text{????} =$$

↑
Inizializzazioni

@2020
giuseppe.prencipe@unipi.it

2. Definire la struttura ricorsiva — top-down (overlap delle soluzioni)

$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}$$

- Quindi, indicando con $T(n)$ la complessità del calcolo di $M(0,n-1)$,
 - Qual è l'equazione di ricorrenza? (*ricordate che $M(i,j)$ prova tutti gli r*)

$$T(n) = c' + \sum_{r=0, \dots, n-2} (T(r+1) + T(n-r-1) + c'')$$

- Esponenziale!!

@2020
giuseppe.prencipe@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

- Il problema, come visto, è che i valori vengono calcolati più volte
- Ricorriamo alla programmazione *dinamica*
 - Memorizziamo i risultati parziali in una matrice
 - Il calcolo di $M[i,j]$ si basa su valori di M già calcolati
 - Cerchiamo di capire quali sono i valori *base*
 - **Domanda:** In questo caso, quali elementi della matrice possiamo riempire subito?

3. Calcolare valore ottimo della soluzione — bottom-up

- Il problema, come visto, è che i valori vengono calcolati più volte
- Ricorriamo alla programmazione *dinamica*
 - Memorizziamo i risultati parziali in una matrice
 - Il calcolo di $M[i,j]$ si basa su valori di M già calcolati
 - Cerchiamo di capire quali sono i valori *base*
 - **$M[i,i]=0$ (era il caso base della ricorsione)**
 - Calcolare il prodotto della sequenza composta da una sola matrice ha costo nullo

3. Calcolare valore ottimo della soluzione — bottom-up

- Il problema, come visto, è che i valori vengono calcolati più volte
- Ricorriamo alla programmazione *dinamica*
 - Memorizziamo i risultati parziali in una matrice
 - Il calcolo di $M[i,j]$ si basa su valori di M **già calcolati**
 - Cerchiamo di capire quali sono i valori *base*

– $M[i,i]=0$ (**era il caso base della ricorsione**)

- Calcolare il problema su una sola matrice

Quindi, al passo 0, possiamo riempire con 0 la diagonale della matrice M

3. Calcolare valore ottimo della soluzione — bottom-up

tabella: $\text{costi}[i][j] = M(i,j)$

$i - j$	0	1	2	3
0	0			
1		0		
2			0	
3				0

$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

3. Calcolare valore ottimo della soluzione — bottom-up

tabella: $\text{costi}[i][j] = M(i,j)$

Domanda: noti i valori della diagonale (tutti 0), quali elementi della matrice possiamo calcolare?

In altre parole, quali elementi della matrice riempire al passo 1?

i - j	0	1	2	3
0	0			
1		0		
2			0	
3				0

$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

@2020
giuseppe.precipe@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

tabella: $\text{costi}[i][j] = M(i,j)$

Domanda: noti i valori della diagonale (tutti 0), quali elementi della matrice possiamo calcolare?

In altre parole, quali elementi della matrice riempire al passo 1?

i - j	0	1	2	3
0	0			
1		0		
2			0	
3				0

$M[0,3]$ di quali valori ha bisogno?

$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

@2020
giuseppe.precipe@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

tabella: $\text{costi}[i][j] = M(i,j)$

Domanda: noti i valori della diagonale (tutti 0), quali elementi della matrice possiamo calcolare?

In altre parole, quali elementi della matrice riempire al passo 1?

$M[0,3]$ di quali valori ha bisogno?

$i - j$	0	1	2	3
0				
1		0		
2			0	
3				0

$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

@2020
giuseppe.precipe@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

tabella: $\text{costi}[i][j] = M(i,j)$

Domanda: noti i valori della diagonale (tutti 0), quali elementi della matrice possiamo calcolare?

In altre parole, quali elementi della matrice riempire al passo 1?

$M[0,3]$ di quali valori ha bisogno?

$i - j$	0	1	2	3
0				
1		0		
2			0	
3				0

$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

@2020
giuseppe.precipe@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

tabella: $\text{costi}[i][j] = M(i,j)$

Domanda: noti i valori della diagonale (tutti 0), quali elementi della matrice possiamo calcolare?

In altre parole, quali elementi della matrice riempire al passo 1?

$M[0,3]$ di quali valori ha bisogno?

$i - j$	0	1	2	3
0				
1		0		
2			0	
3				

$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

@2020
giuseppe.precipe@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

tabella: $\text{costi}[i][j] = M(i,j)$

Domanda: noti i valori della diagonale (tutti 0), quali elementi della matrice possiamo calcolare?

In altre parole, quali elementi della matrice riempire al passo 1?

$M[0,3]$ di quali valori ha bisogno?

$i - j$	0	1	2	3
0				
1		0		
2			0	
3				

$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

@2020
giuseppe.precipe@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

tabella: $\text{costi}[i][j] = M(i,j)$

Domanda: noti i valori della diagonale (tutti 0), quali elementi della matrice possiamo calcolare?

In altre parole, quali elementi della matrice riempire al passo 1?

i - j	0	1	2	3
0	0			
1		0		
2			0	
3				0

$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

@2020
giuseppe.precipe@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

tabella: $\text{costi}[i][j] = M(i,j)$

Domanda: noti i valori della diagonale (tutti 0), quali elementi della matrice possiamo calcolare?

In altre parole, quali elementi della matrice riempire al passo 1?

i - j	0	1	2	3
0	0			
1		0		
2			0	
3				0

$M[2,3]$ di quali valori ha bisogno?

$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

@2020
giuseppe.precipe@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

tabella: $\text{costi}[i][j] = M(i,j)$

Domanda: noti i valori della diagonale (tutti 0), quali elementi della matrice possiamo calcolare?

In altre parole, quali elementi della matrice riempire al passo 1?

$M[2,3]$ di quali valori ha bisogno?

$i - j$	0	1	2	3
0	0			
1		0		
2				
3				

$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

@2020
giuseppe.precipe@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

tabella: $\text{costi}[i][j] = M(i,j)$

Domanda: noti i valori della diagonale (tutti 0), quali elementi della matrice possiamo calcolare?

In altre parole, quali elementi della matrice riempire al passo 1?

SI!!!!

$M[2,3]$ di quali valori ha bisogno?

$i - j$	0	1	2	3
0	0			
1		0		
2				
3				

$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

@2020
giuseppe.precipe@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

tabella: $\text{costi}[i][j] = M(i,j)$

Domanda: noti i valori della diagonale (tutti 0), quali elementi della matrice possiamo calcolare?

In altre parole, quali elementi della matrice riempire al passo 1?

i - j	0	1	2	3
0	0			
1		0		
2			0	
3				0

In generale, come è possibile riempire la matrice?

$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

@2020
giuseppe.precipe@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

tabella: $\text{costi}[i][j] = M(i,j)$

Domanda: noti i valori della diagonale (tutti 0), quali elementi della matrice possiamo calcolare?

In altre parole, quali elementi della matrice riempire al passo 1?

i - j	0	1	2	3
0	0			
1		0		
2			0	
3				0

In generale, come è possibile riempire la matrice?

$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

@2020
giuseppe.precipe@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

tabella: $\text{costi}[i][j] = M(i,j)$

Domanda: noti i valori della diagonale (tutti 0), quali elementi della matrice possiamo calcolare?

In altre parole, quali elementi della matrice riempire al passo 1?

In generale, come è possibile riempire la matrice?

$i - j$	0	1	2	3
0	0			
1		0		
2			0	
3				0

$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

@2020
giuseppe.precipe@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

tabella: $\text{costi}[i][j] = M(i,j)$

Domanda: noti i valori della diagonale (tutti 0), quali elementi della matrice possiamo calcolare?

In altre parole, quali elementi della matrice riempire al passo 1?

In generale, come è possibile riempire la matrice?

$i - j$	0	1	2	3
0	0			
1		0		
2			0	
3				0

$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

@2020
giuseppe.precipe@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

tabella: $\text{costi}[i][j] = M(i,j)$

Domanda: noti i valori della diagonale (tutti 0), quali elementi della matrice possiamo calcolare?

In altre parole, quali elementi della matrice riempire al passo 1?

i - j	0	1	2	3
0	0			
1		0		
2			0	
3				0

Per diagonali!!!!

$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

©2020
giuseppe.prencipe@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

- L'approccio bottom-up riempie la matrice per diagonali
- Valore $[i,j]$ calcolato con i valori precedenti in riga i-esima e colonna j-esima

risultato finale

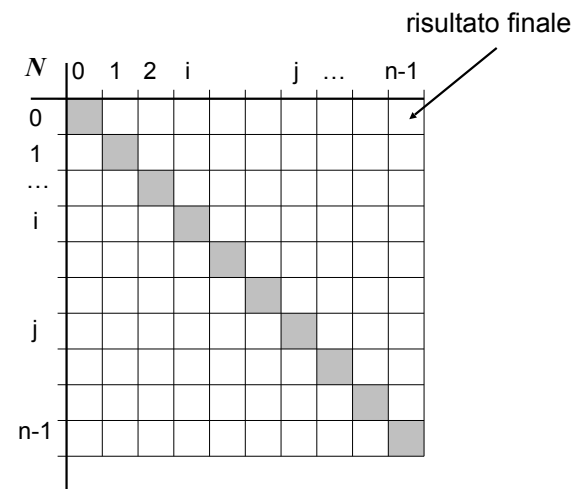
N	0	1	2	i	j	...	n-1
0							
1							
...							
i							
j							
n-1							

$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

©2020
giuseppe.prencipe@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

- L'approccio bottom-up riempie la matrice per diagonal
- Valore $[i,j]$ calcolato con i valori precedenti in riga i -esima e colonna j -esima

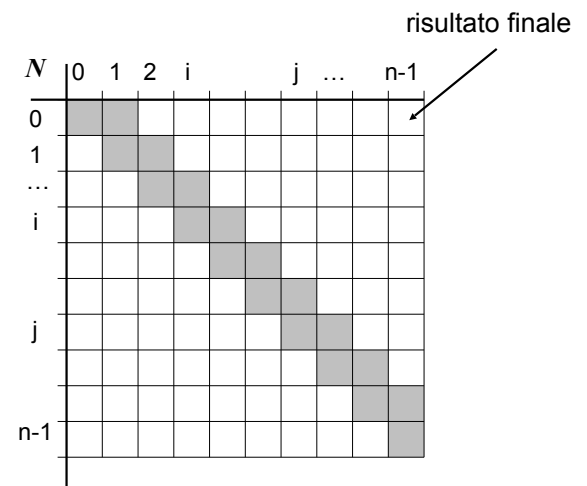


$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

@2020
giuseppe.preci@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

- L'approccio bottom-up riempie la matrice per diagonal
- Valore $[i,j]$ calcolato con i valori precedenti in riga i -esima e colonna j -esima

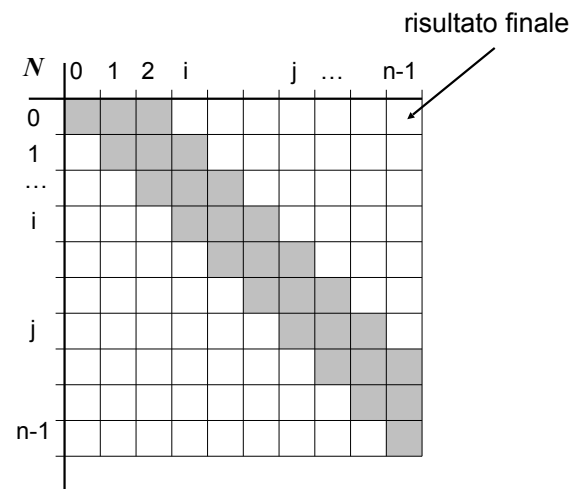


$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

@2020
giuseppe.preci@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

- L'approccio bottom-up riempie la matrice per diagonal
- Valore $[i,j]$ calcolato con i valori precedenti in riga i -esima e colonna j -esima

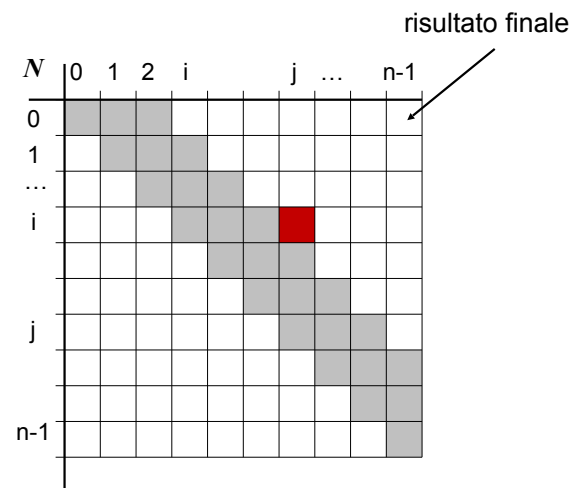


$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

@2020
giuseppe.preci@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

- L'approccio bottom-up riempie la matrice per diagonal
- Valore $[i,j]$ calcolato con i valori precedenti in riga i -esima e colonna j -esima

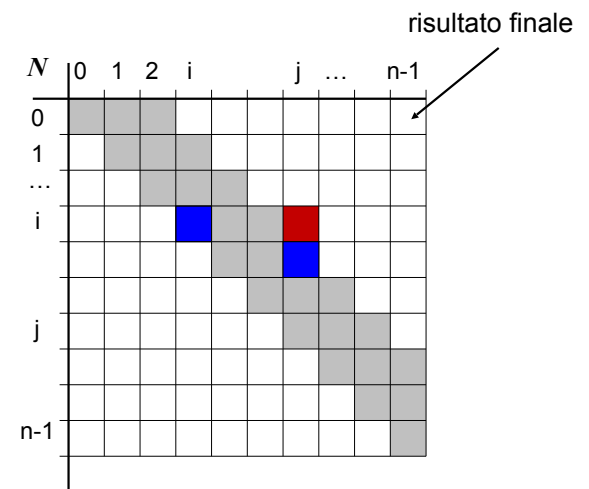


$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

@2020
giuseppe.preci@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

- L'approccio bottom-up riempie la matrice per diagonal
- Valore $[i,j]$ calcolato con i valori precedenti in riga i -esima e colonna j -esima

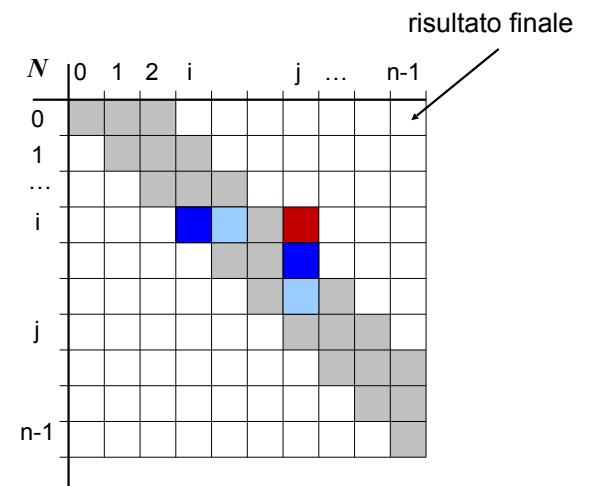


$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

@2020
giuseppe.preci@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

- L'approccio bottom-up riempie la matrice per diagonal
- Valore $[i,j]$ calcolato con i valori precedenti in riga i -esima e colonna j -esima

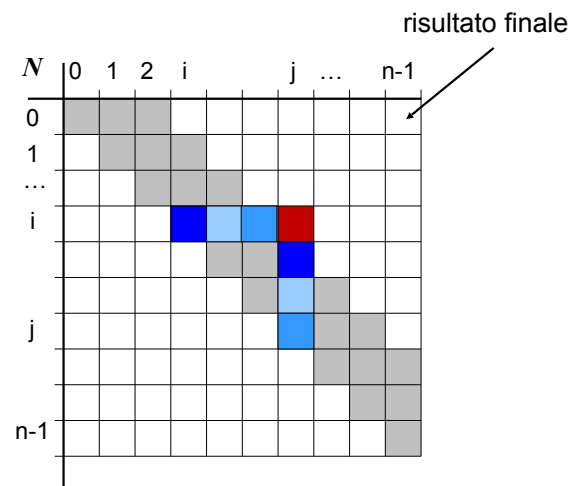


$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

@2020
giuseppe.preci@unipi.it

3. Calcolare valore ottimo della soluzione — bottom-up

- L'approccio bottom-up riempie la matrice per diagonal
- Valore $[i,j]$ calcolato con i valori precedenti in riga i -esima e colonna j -esima



$$M(i,j) = M(i,r) + M(r+1,j) + d_i d_{r+1} d_{j+1}, i < j$$

@2020
giuseppe.precipe@unipi.it

Soluzione

```

1 CostoMinimoIterativo( ):
2   FOR (i = 0; i < n; i = i+1) {
3     costi[i][i] = 0;
4   }
5   FOR (diagonale = 1; diagonale < n; diagonale = diagonale+1) {
6     FOR (i = 0; i < n-diagonale; i = i+1) {
7       j = i + diagonale;
8       costi[i][j] = +∞;
9       FOR (r = i; r < j; r = r+1) {
10        costo = costi[i][r] + costi[r+1][j];
11        costo = costo + d[i] × d[r+1] × d[j+1];
12        IF (costo < costi[i][j]) {
13          costi[i][j] = costo;
14          indice[i][j] = r;
15        }
16      }
17    }
18  }
19  RETURN costi[0][n-1];

```

Sostituiscono le chiamate ricorsive

Serve a ricostruire la soluzione ottima

i - j	0	1	2	3
0	0			
1		0		
2			0	
3				0

@2020
giuseppe.precipe@ur

Analisi di complessità

i-j	0	1	2	3
0	0			
1		0		
2			0	
3				0

- Quanto costa riempire ogni casella della tabella?

```

1 CostoMinimoIterativo( ):
2   FOR (i = 0; i < n; i = i+1) {
3     costi[i][i] = 0;
4   }
5   FOR (diagonale = 1; diagonale < n; diagonale = diagonale+1) {
6     FOR (i = 0; i < n-diagonale; i = i+1) {
7       j = i + diagonale;
8       costi[i][j] = +∞;
9       FOR (r = i; r < j; r = r+1) {
10        costo = costi[i][r] + costi[r+1][j];
11        costo = costo + d[i] × d[r+1] × d[j+1];
12        IF (costo < costi[i][j]) {
13          costi[i][j] = costo;
14          indice[i][j] = r;
15        }
16      }
17    }
18  }
19  RETURN costi[0][n-1];

```

Domanda: complessità??

@2020
giuseppe.prencipe@unipi.it

Analisi di complessità

i-j	0	1	2	3
0	0			
1		0		
2			0	
3				0

- La tabella **costi** ha taglia $n \times n$ e il calcolo di ciascuno dei suoi elementi richiede $O(n)$ tempo (ciclo **for** più interno)
- CostoMinimoIterativo** richiede $O(n^3)$ tempo (laddove la versione ricorsiva è esponenziale in n)
- Lo **spazio occupato** è $O(n^2)$ celle

- Anche qui ottimizzazione in spazio possibile

```

1 CostoMinimoIterativo( ):
2   FOR (i = 0; i < n; i = i+1) {
3     costi[i][i] = 0;
4   }
5   FOR (diagonale = 1; diagonale < n; diagonale = diagonale+1) {
6     FOR (i = 0; i < n-diagonale; i = i+1) {
7       j = i + diagonale;
8       costi[i][j] = +∞;
9       FOR (r = i; r < j; r = r+1) {
10        costo = costi[i][r] + costi[r+1][j];
11        costo = costo + d[i] × d[r+1] × d[j+1];
12        IF (costo < costi[i][j]) {
13          costi[i][j] = costo;
14          indice[i][j] = r;
15        }
16      }
17    }
18  }
19  RETURN costi[0][n-1];

```

@2020
giuseppe.prencipe@unipi.it

4. Costruire la soluzione ottima

- Utilizzando la tabella **indice**, si ricava la sequenza ottima di moltiplicazioni per il calcolo di A^*

- Scrivere una procedura

StampaSequenza(indice, i, j)

che restituisca la sequenza di prodotti di costo minimo per moltiplicare le matrici in A_i, \dots, A_j

```
1 CostoMinimoIterativo( ):  
2   FOR (i = 0; i < n; i = i+1) {  
3     costi[i][i] = 0;  
4   }  
5   FOR (diagonale = 1; diagonale < n; diagonale = diagonale+1) {  
6     FOR (i = 0; i < n-diagonale; i = i+1) {  
7       j = i + diagonale;  
8       costi[i][j] = +∞;  
9       FOR (r = i; r < j; r = r+1) {  
10        costo = costi[i][r] + costi[r+1][j];  
11        costo = costo + d[i] × d[r+1] × d[j+1];  
12        IF (costo < costi[i][j]) {  
13          costi[i][j] = costo;  
14          indice[i][j] = r;  
15        }  
16      }  
17    }  
18  }  
19  RETURN costi[0][n-1];
```

@2020
giuseppe.prencipe@unipi.it

4. Costruire la soluzione ottima

StampaSequenza(indice, i, j)

IF (i==j) {

STAMPA A_i ;

}

else {

STAMPA "("

r=indice[i][j];

STAMPA StampaSequenza(indice, i, r);

STAMPA StampaSequenza(indice, r+1, j);

STAMPA ")"

}

```
1 CostoMinimoIterativo( ):  
2   FOR (i = 0; i < n; i = i+1) {  
3     costi[i][i] = 0;  
4   }  
5   FOR (diagonale = 1; diagonale < n; diagonale = diagonale+1) {  
6     FOR (i = 0; i < n-diagonale; i = i+1) {  
7       j = i + diagonale;  
8       costi[i][j] = +∞;  
9       FOR (r = i; r < j; r = r+1) {  
10        costo = costi[i][r] + costi[r+1][j];  
11        costo = costo + d[i] × d[r+1] × d[j+1];  
12        IF (costo < costi[i][j]) {  
13          costi[i][j] = costo;  
14          indice[i][j] = r;  
15        }  
16      }  
17    }  
18  }  
19  RETURN costi[0][n-1];
```

@2020
giuseppe.prencipe@unipi.it

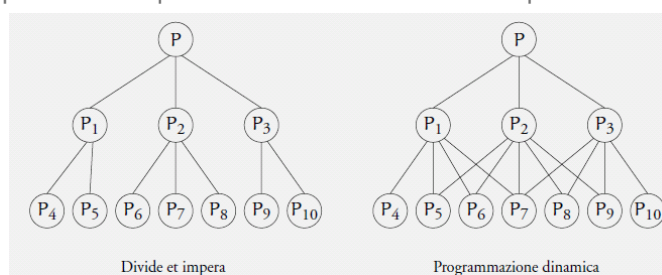
Paradigma della programmazione dinamica

Similitudine con *divide et impera*:

- Decomposizione di problemi in sotto-problemi
- definizione ricorsiva della soluzione

Differenza rispetto al *divide et impera*:

- Utile in **problemi di ottimizzazione** in cui ogni soluzione ammissibile ha un costo: soluzione di **costo ottimo**
- uno stesso sotto-problema può rientrare come componente nella definizione di diversi altri



@2020
giuseppe.precipe@unipi.it

Regole auree per la programmazione dinamica

1. Soluzione **ottima** di un problema deriva dalla **ottimalità** delle soluzioni dei suoi **sotto-problemi**: Definizione di una **regola ricorsiva** di calcolo della soluzione ottima (punto 1.)
2. **Tabella di programmazione dinamica** da riempire **iterativamente** con la regola ricorsiva (punto 2.)
3. **Ordine di riempimento** della tabella (punto 3.)
4. Cercare eventualmente la **soluzione di costo ottimo**

@2020
giuseppe.precipe@unipi.it

Esercizio

A_0 : 30 X 35

A_1 : 35 X15

A_2 : 15X5

A_3 : 5X10

A_4 : 10X20

A_5 : 20 X 25

@2020
giuseppe.precipice@unipi.it

Esercizio

- A_0 : 30 X 35; A_1 : 35 X15; A_2 : 15X5;
 A_3 : 5X10; A_4 : 10X20; A_5 : 20 X 25

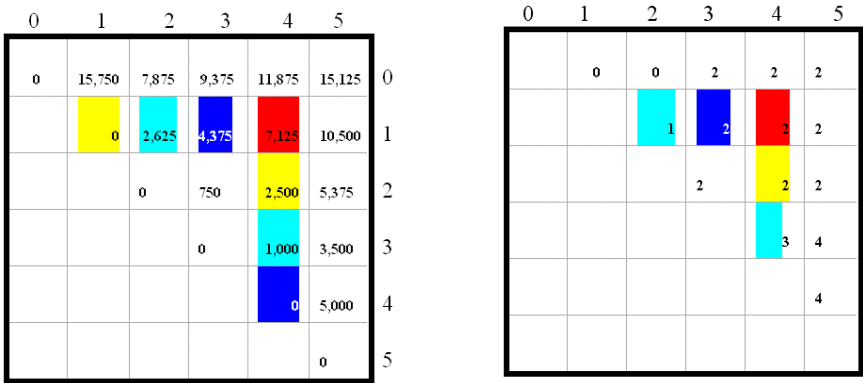
	0	1	2	3	4	5	
0	0	15,750	7,875	9,375	11,875	15,125	0
1		0	2,625	4,375	7,125	10,500	1
2			0	750	2,500	5,375	2
3				0	1,000	3,500	3
4					0	5,000	4
5						0	5

Esempio per calcolo $M[1,4]$

@2020
giuseppe.precipice@unipi.it

Esercizio

$(A_0 * (A_1 * A_2)) * ((A_3 * A_4) * A_5)$



Longest Common Subsequence

Sicurezza dei sistemi e sotto-sequenza comune più lunga (LCS)

- Traccia dei comandi eseguiti (file di *log*) per monitorare eventuali intrusioni (*intrusion detection*)
- Intrusione: **sotto-sequenza** di un file di log

Operazioni rappresentate come etichette: **A**, **B**, ...

Esempio:

log = F =

B, A, A, B, D, C, D, C, A, A, C, A, C, B, A

intrusione = S =

A, D, C, A, A, B

@2020
giuseppe.prencipe@unipi.it

Sotto-sequenza comune più lunga

- Una sequenza S di lunghezza k è **sotto-sequenza** di una sequenza A di lunghezza n se e solo se può essere ottenuta da A *cancellando alcuni elementi*

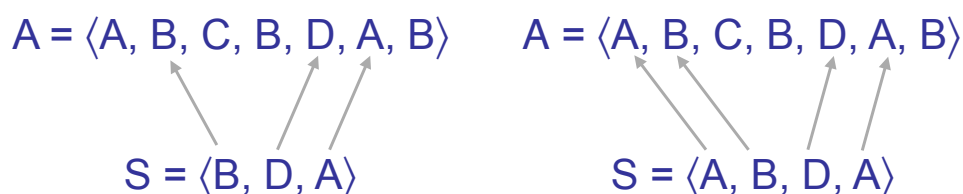
@2020
giuseppe.prencipe@unipi.it

Sotto-sequenza comune più lunga

- Una sequenza S di lunghezza k è **sotto-sequenza** di una sequenza A di lunghezza n se e solo se può essere ottenuta da A *cancellando alcuni elementi*
- **Più formalmente**, se esistono delle posizioni $0 \leq i_0 < i_1 < \dots < i_{k-1} \leq n-1$ in A tali che
$$S[j] = A[i_j] \text{ per } j = 0, 1, \dots, k-1$$

Sotto-sequenza comune più lunga

- Una sequenza S di lunghezza k è **sotto-sequenza** di una sequenza A di lunghezza n se e solo se può essere ottenuta da A *cancellando alcuni elementi*
- **Più formalmente**, se esistono delle posizioni $0 \leq i_0 < i_1 < \dots < i_{k-1} \leq n-1$ in A tali che
$$S[j] = A[i_j] \text{ per } j = 0, 1, \dots, k-1$$



Sotto-sequenza comune più lunga

- Una sequenza S di lunghezza k è **sotto-sequenza** di una sequenza A di lunghezza n se esistono delle posizioni $0 \leq i_0 < i_1 < \dots < i_{k-1} \leq n-1$ in A tali che
 $S[j] = A[i_j]$ per $j = 0, 1, \dots, k-1$
- S è **sotto-sequenza comune** ad A e B , se è sotto-sequenza comune sia di A che di B

$A = \langle A, B, C, B, D, A, B \rangle$ $A = \langle A, B, C, B, D, A, B \rangle$

$B = \langle B, D, C, A, B, A \rangle$ $B = \langle B, D, C, A, B, A \rangle$

@2020
giuseppe.precipec@unipi.it

Sotto-sequenza comune più lunga

- Una sequenza S di lunghezza k è **sotto-sequenza** di una sequenza A di lunghezza n se esistono delle posizioni $0 \leq i_0 < i_1 < \dots < i_{k-1} \leq n-1$ in A tali che
 $S[j] = A[i_j]$ per $j = 0, 1, \dots, k-1$
- S è **sotto-sequenza comune** ad A e B , se è sotto-sequenza comune sia di A che di B

$A = \langle A, \mathbf{B}, \mathbf{C}, B, D, A, B \rangle$ $A = \langle A, \mathbf{B}, \mathbf{C}, B, D, A, B \rangle$

$S = \langle B, C, A \rangle$ $S' = \langle B, C, B, A \rangle$

$B = \langle \mathbf{B}, D, \mathbf{C}, \mathbf{A}, B, A \rangle$ $B = \langle \mathbf{B}, D, \mathbf{C}, A, \mathbf{B}, \mathbf{A} \rangle$

@2020
giuseppe.precipec@unipi.it

Sotto-sequenza comune più lunga

- Una sequenza S di lunghezza k è **sotto-sequenza** di una sequenza A di lunghezza n se esistono delle posizioni $0 \leq i_0 < i_1 < \dots < i_{k-1} \leq n-1$ in A tali che
 $S[j] = A[i_j]$ per $j = 0, 1, \dots, k-1$
- S è **sotto-sequenza comune** ad A e B , se è sotto-sequenza comune sia di A che di B
- $\text{LCS}(A, B) = \text{lunghezza}$ della **sotto-sequenza comune più lunga** (*longest common subsequence*)

Sotto-sequenza comune più lunga

- $\text{LCS}(A, B) = \text{lunghezza}$ della **sotto-sequenza comune più lunga** (*longest common subsequence*)

Sotto-sequenza comune più lunga

- $\text{LCS}(A, B) = \text{lunghezza}$ della **sotto-sequenza comune più lunga** (*longest common subsequence*)

$A = \langle A, B, C, B, D, A, B \rangle$ $A = \langle A, B, C, B, D, A, B \rangle$

$S = \langle B, C, A \rangle$ $S' = \langle B, C, B, A \rangle$

$B = \langle B, D, C, A, B, A \rangle$ $B = \langle B, D, C, A, B, A \rangle$

S è LCS di A e B?

Sotto-sequenza comune più lunga

- $\text{LCS}(A, B) = \text{lunghezza}$ della **sotto-sequenza comune più lunga** (*longest common subsequence*)

$A = \langle A, B, C, B, D, A, B \rangle$ $A = \langle A, B, C, B, D, A, B \rangle$

$S = \langle B, C, A \rangle$ $S' = \langle B, C, B, A \rangle$

$B = \langle B, D, C, A, B, A \rangle$ $B = \langle B, D, C, A, B, A \rangle$

S è LCS di A e B?



Sotto-sequenza comune più lunga

- $LCS(A, B) = \text{lunghezza}$ della **sotto-sequenza comune più lunga** (*longest common subsequence*)

$A = \langle A, B, C, B, D, A, B \rangle$ $A = \langle A, B, C, B, D, A, B \rangle$

$S = \langle B, C, A \rangle$ $S' = \langle B, C, B, A \rangle$

$B = \langle B, D, C, A, B, A \rangle$ $B = \langle B, D, C, A, B, A \rangle$

S' è LCS di A e B?

Sotto-sequenza comune più lunga

- $LCS(A, B) = \text{lunghezza}$ della **sotto-sequenza comune più lunga** (*longest common subsequence*)

$A = \langle A, B, C, B, D, A, B \rangle$ $A = \langle A, B, C, B, D, A, B \rangle$

$S = \langle B, C, A \rangle$ $S' = \langle B, C, B, A \rangle$

$B = \langle B, D, C, A, B, A \rangle$ $B = \langle B, D, C, A, B, A \rangle$

S' è LCS di A e B?



Sotto-sequenza comune più lunga

- $LCS(A, B) =$ **lunghezza** della **sotto-sequenza comune più lunga** (*longest common subsequence*)
- Questa **formulazione generica** del problema copre anche il problema (definito inizialmente) di scoprire se S di lunghezza k compare come sotto-sequenza (**intrusione**) di un file di log F
- **Domanda**: perché?

Sotto-sequenza comune più lunga

- $LCS(A, B) =$ **lunghezza** della **sotto-sequenza comune più lunga** (*longest common subsequence*)
- Questa **formulazione generica** del problema copre anche il problema (definito inizialmente) di scoprire se S di lunghezza k compare come sotto-sequenza (**intrusione**) di un file di log F
- **Domanda**: perché?

È **sufficiente** verificare che $k = LCS(S, F)$

Altre applicazioni di LCS

- **Biologia**
 - Sequenze di DNA sono rappresentate come sequenze di sotto-molecole, ognuna appartenente a un tipo: A C G T. In genetica, è di forte interesse calcolare le similarità tra due DNA mediante LCS
- **Confronto tra file**
 - **Sistemi di versionamento: esempio - "diff"** è utilizzato per confrontare due differenti versioni dello stesso file, per determinare quali cambiamenti sono stati fatti al file. Funziona trovando la LCS delle righe dei due file

@2020
giuseppe.prencipe@unipi.it

Soluzione di forza brut(t)a

- **Come potremmo procedere?**

@2020
giuseppe.prencipe@unipi.it

Soluzione di forza brut(t)a

@2020
giuseppe.prencipe@unipi.it

Soluzione di forza brut(t)a

- Siano A e B due sequenze
 - $LCS(A,B) = L(m,n)$, dove $m = |A|$ e $n = |B|$

@2020
giuseppe.prencipe@unipi.it

Soluzione di forza brut(t)a

- Siano A e B due sequenze
 - $LCS(A,B) = L(m,n)$, dove $m = |A|$ e $n = |B|$
- Per ogni sotto-sequenza di A, controlla se è sotto-sequenza di B
 - Ci sono 2^m sotto-sequenze di A da controllare
- Per controllare ogni sotto-sequenza, si impiega $\Theta(n)$ tempo
 - Scandisci B per cercare la prima lettera della sotto-sequenza in esame, e da lì cerca la seconda lettera, e così via
- Tempo: $\Theta(n2^m)$

@2020
giuseppe.prencipe@unipi.it

1. e 2. Struttura ricorsiva

LCS(A,B) può essere ricavato analizzando i prefissi di A e B....

@2020
giuseppe.prencipe@unipi.it

1. e 2. Struttura ricorsiva

LCS(A,B) può essere ricavato analizzando i prefissi di A e B....



@2020
giuseppe.precipe@unipi.it

1. e 2. Struttura ricorsiva

LCS(A,B) può essere ricavato analizzando i prefissi di A e B....

- Immaginiamo di aver trovato l'ottimo tra due sotto-problemi
 - **Prefisso** x di A e **prefisso** y di B
 - $|x| = i, |y| = j$



@2020
giuseppe.precipe@unipi.it

1. e 2. Struttura ricorsiva

LCS(A,B) può essere ricavato analizzando i prefissi di A e B....

- Immaginiamo di aver trovato l'ottimo tra due sotto-problemi
 - **Prefisso** x di A e **prefisso** y di B
 - $|x| = i, |y| = j$

Conosciamo $\text{LCS}(x,y) = 3$



@2020
giuseppe.prencipe@unipi.it



1. e 2. Struttura ricorsiva

LCS(A,B) può essere ricavato analizzando i prefissi di A e B....

- Immaginiamo di aver trovato l'ottimo tra due sotto-problemi
 - **Prefisso** x di A e **prefisso** y di B
 - $|x| = i, |y| = j$

Conosciamo $\text{LCS}(x,y) = 3$



@2020
giuseppe.prencipe@unipi.it



**Ora, cosa facciamo per andare avanti
nella ricerca di $\text{LCS}(A,B)$?**

1. e 2. Struttura ricorsiva

LCS(A,B) può essere ricavato analizzando i prefissi di A e B....

- Immaginiamo di aver trovato l'ottimo tra due sotto-problemi
 - **Prefisso** x di A e **prefisso** y di B
 - $|x| = i, |y| = j$

Conosciamo $LCS(x,y) = 3$



@2020
giuseppe.precipe@unipi.it



Analizziamo il carattere successivo ad entrambi i prefissi....

1. e 2. Struttura ricorsiva

LCS(A,B) può essere ricavato analizzando i prefissi di A e B....

- Immaginiamo di aver trovato l'ottimo tra due sotto-problemi
 - **Prefisso** x di A e **prefisso** y di B
 - $|x| = i, |y| = j$

Conosciamo $LCS(x,y) = 3$



@2020
giuseppe.precipe@unipi.it



Quali sono i possibili casi?

1. e 2. Struttura ricorsiva

LCS(A,B) può essere ricavato analizzando i prefissi di A e B....

- Immaginiamo di aver trovato l'ottimo tra due sotto-problemi
 - Prefisso** x di A e **prefisso** y di B
 - $|x| = i, |y| = j$

Conosciamo $LCS(x,y) = 3$



@2020
giuseppe.precipe@unipi.it



$c==c'$ oppure $c!=c'$

1. e 2. Struttura ricorsiva

LCS(A,B) può essere ricavato analizzando i prefissi di A e B....

- Immaginiamo di aver trovato l'ottimo tra due sotto-problemi
 - Prefisso** x di A e **prefisso** y di B
 - $|x| = i, |y| = j$

Conosciamo $LCS(x,y) = ?$



@2020
giuseppe.precipe@unipi.it



Se $c==c'$ qual è il nuovo LCS()?

1. e 2. Struttura ricorsiva

LCS(A,B) può essere ricavato analizzando i prefissi di A e B....

- Immaginiamo di aver trovato l'ottimo tra due sotto-problemi
 - Prefisso** x di A e **prefisso** y di B
 - $|x| = i, |y| = j$

Conosciamo $LCS(x,y) = 4$



@2020
giuseppe.precipe@unipi.it



Se $c==c'$ qual è il nuovo LCS()?

1. e 2. Struttura ricorsiva

LCS(A,B) può essere ricavato analizzando i prefissi di A e B....

- Immaginiamo di aver trovato l'ottimo tra due sotto-problemi
 - Prefisso** x di A e **prefisso** y di B
 - $|x| = i, |y| = j$

$$LCS(i,j) = LCS(x, y) + 1$$

Conosciamo $LCS(x,y) = 4$



@2020
giuseppe.precipe@unipi.it



Se $c==c'$ qual è il nuovo LCS()?

1. e 2. Struttura ricorsiva

LCS(A,B) può essere ricavato analizzando i prefissi di A e B....

- Immaginiamo di aver trovato l'ottimo tra due sotto-problemi
 - Prefisso** x di A e **prefisso** y di B
 - $|x| = i, |y| = j$



Conosciamo $LCS(x,y) = ?$



@2020
giuseppe.precipe@unipi.it

Se $c \neq c'$ qual è il nuovo LCS?

1. e 2. Struttura ricorsiva

LCS(A,B) può essere ricavato analizzando i prefissi di A e B....

- Immaginiamo di aver trovato l'ottimo tra due sotto-problemi
 - Prefisso** x di A e **prefisso** y di B
 - $|x| = i, |y| = j$

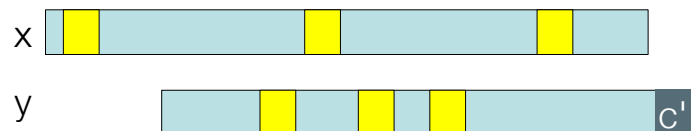
Dobbiamo considerare i due possibili scenari

Scenario S1: $LCS(x+c, y)$



@2020
giuseppe.precipe@unipi.it

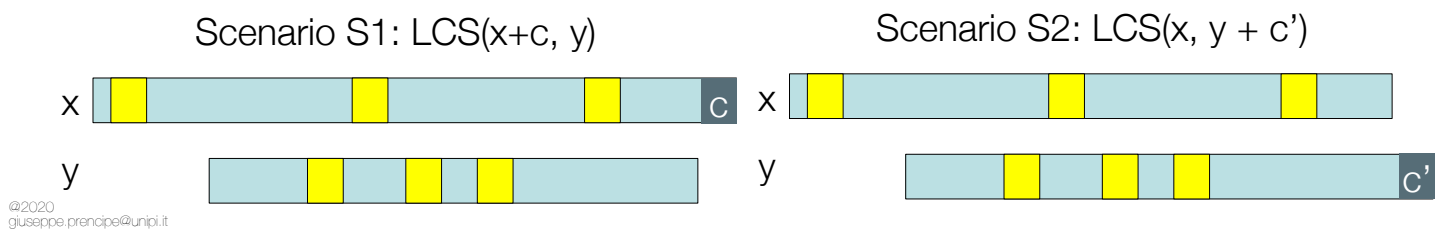
Scenario S2: $LCS(x, y + c')$



1. e 2. Struttura ricorsiva

LCS(A,B) può essere ricavato analizzando i prefissi di A e B....

- Immaginiamo di aver trovato l'ottimo tra due sotto-problemi
 - **Prefisso** x di A e **prefisso** y di B
 - $|x| = i, |y| = j$

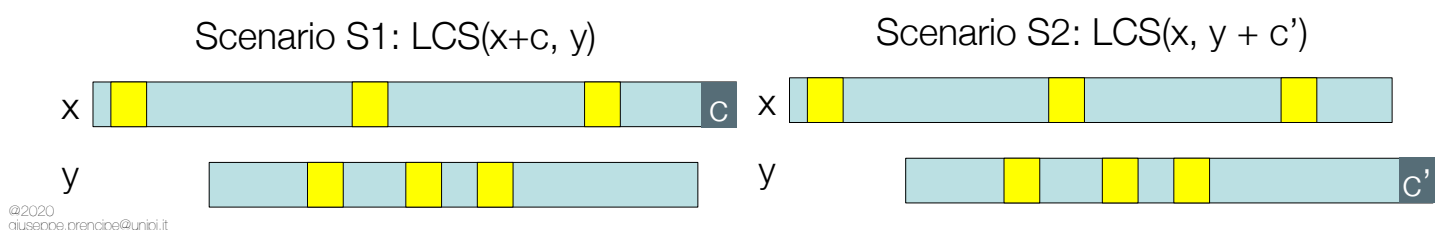


1. e 2. Struttura ricorsiva

LCS(A,B) può essere ricavato analizzando i prefissi di A e B....

- Immaginiamo di aver trovato l'ottimo tra due sotto-problemi
 - **Prefisso** x di A e **prefisso** y di B
 - $|x| = i, |y| = j$

$$\text{LCS}(i,j) = \max(\text{S1}, \text{S2})$$



1. e 2. Struttura ricorsiva

LCS(A,B) può essere ricavato analizzando i prefissi di A e B....

- Immaginiamo di aver trovato l'ottimo tra due sotto-problemi
 - Prefisso** x di A e **prefisso** y di B
 - $|x| = i, |y| = j$

Provate a definire la formulazione ricorsiva (prima i casi facili e poi gli altri)....

$$\text{LCS}(i,j) = \max(S1, S2)$$

Scenario S1: $\text{LCS}(x+c, y)$

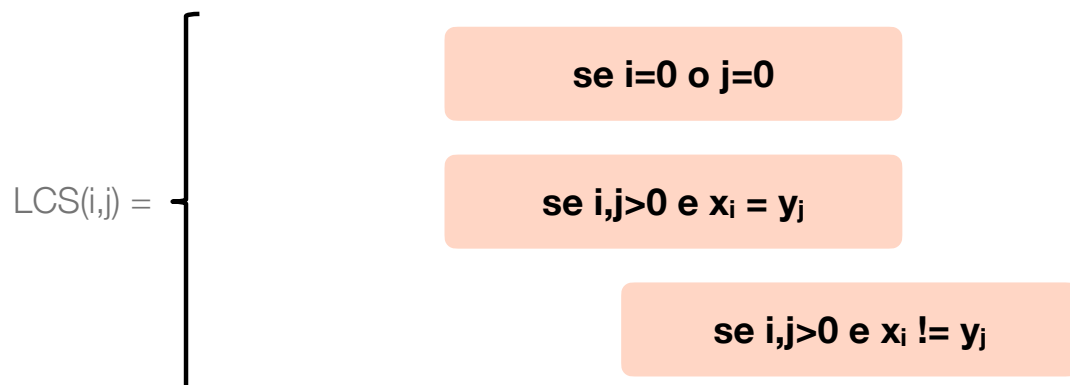


@2020
giuseppe.prencipe@unipi.it

Scenario S2: $\text{LCS}(x, y + c')$



1. e 2. Struttura ricorsiva



1. e 2. Struttura ricorsiva

$$\text{LCS}(i,j) = \begin{cases} 0 & \text{se } i=0 \text{ o } j=0 \\ \text{se } i,j>0 \text{ e } x_i = y_j \\ \text{se } i,j>0 \text{ e } x_i \neq y_j \end{cases}$$

@2020
giuseppe.prencipe@unipi.it

1. e 2. Struttura ricorsiva

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

@2020
giuseppe.prencipe@unipi.it

1. e 2. Struttura ricorsiva

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

3. Calcolo bottom-up

Come definiamo la tabella di programmazione dinamica?

3. Calcolo bottom-up

Come definiamo la tabella di programmazione dinamica?

$$\text{LCS}(i,j) = \text{LCS}(x, y) + 1$$

$$\text{LCS}(i,j) = \max(S1, S2)$$

3. Calcolo bottom-up

Come definiamo la tabella di programmazione dinamica?

$$\text{LCS}(i,j) = \text{LCS}(x, y) + 1$$

$$\text{LCS}(i,j) = \max(S1, S2)$$

Matrice **L** di dimensione **mxn**

dove $L(i,j)$ rappresenta $\text{LCS}(A[0, i-1], B[0, j-1])$

3. Calcolo bottom-up

Quali sono i casi base (cioè i valori che posso già inserire in L)?

Matrice L di dimensione **$m \times n$**

dove $L(i,j)$ rappresenta $LCS(A[0, i-1], B[0, j-1])$

@2020
giuseppe.prencipe@unipi.it

3. Calcolo bottom-up

Quali sono i casi base (cioè i valori che posso già inserire in L)?

Se una delle due sequenze è vuota....e quindi?

Matrice L di dimensione **$m \times n$**

dove $L(i,j)$ rappresenta $LCS(A[0, i-1], B[0, j-1])$

@2020
giuseppe.prencipe@unipi.it

3. Calcolo bottom-up

Quali sono i casi base (cioè i valori che posso già inserire in L)?

Se una delle due sequenze è vuota....e quindi?

Se $i = 0$ o $j = 0 \rightarrow$????

Matrice L di dimensione **mxn**

dove $L(i,j)$ rappresenta $LCS(A[0, i-1], B[0, j-1])$

@2020
giuseppe.precipe@unipi.it

3. Calcolo bottom-up

Quali sono i casi base (cioè i valori che posso già inserire in L)?

Se una delle due sequenze è vuota....e quindi?

Se $i = 0$ o $j = 0 \rightarrow L(i,j) = 0$

Matrice L di dimensione **mxn**

dove $L(i,j)$ rappresenta $LCS(A[0, i-1], B[0, j-1])$

@2020
giuseppe.precipe@unipi.it

3. Calcolo bottom-up

Quali sono i casi base (cioè i valori che posso già inserire in L)?

Se una delle due sequenze è vuota....e quindi?

Se $i = 0$ o $j = 0 \rightarrow L(i, j) = 0$

	0	1	2		n
0	0	0	0	0	0
1	0				
2	0				
	0				
	0				
m	0				

Matrice L di dimensione **mxn**

dove $L(i, j)$ rappresenta $LCS(A[0, i-1], B[0, j-1])$

3. Calcolo bottom-up

Quali sono i casi base (cioè i valori che posso già inserire in L)?

Se una delle due sequenze è vuota....e quindi?

Se $i = 0$ o $j = 0 \rightarrow L(i, j) = 0$

[A[0,-1] = B[0,-1] = sequenza vuota]

	0	1	2		n
0	0	0	0	0	0
1	0				
2	0				
	0				
	0				
m	0				

Matrice L di dimensione **mxn**

dove $L(i, j)$ rappresenta $LCS(A[0, i-1], B[0, j-1])$

3. Calcolo bottom-up

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i-1, j-1) + 1 & \text{se } i, j > 0 \text{ e } a[i-1] = b[j-1] \\ \max \{ L(i, j-1), L(i-1, j) \} & \text{se } i, j > 0 \text{ e } a[i-1] \neq b[j-1] \end{cases}$$

	0	1	2		n
0	0	0	0	0	0
1	0				
2	0				
	0				
	0				
m	0				

@2020
giuseppe.prencipe@unipi.it

3. Calcolo bottom-up

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i-1, j-1) + 1 & \text{se } i, j > 0 \text{ e } a[i-1] = b[j-1] \\ \max \{ L(i, j-1), L(i-1, j) \} & \text{se } i, j > 0 \text{ e } a[i-1] \neq b[j-1] \end{cases}$$

Tabella di programmazione dinamica di taglia
(m+1)*(n+1)

$$L[i][j] = \text{LCS}(i, j)$$

	0	1	2		n
0	0	0	0	0	0
1	0				
2	0				
	0				
	0				
m	0				

@2020
giuseppe.prencipe@unipi.it

3. Calcolo bottom-up

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i-1, j-1) + 1 & \text{se } i, j > 0 \text{ e } a[i-1] = b[j-1] \\ \max \{ L(i, j-1), L(i-1, j) \} & \text{se } i, j > 0 \text{ e } a[i-1] \neq b[j-1] \end{cases}$$

Tabella di programmazione dinamica di taglia $(m+1) \times (n+1)$

$$L[i][j] = \text{LCS}(i, j)$$

Ordine riempimento?

	0	1	2		n
0	0	0	0	0	0
1	0				
2	0				
	0				
	0				
m	0				

3. Calcolo bottom-up

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i-1, j-1) + 1 & \text{se } i, j > 0 \text{ e } a[i-1] = b[j-1] \\ \max \{ L(i, j-1), L(i-1, j) \} & \text{se } i, j > 0 \text{ e } a[i-1] \neq b[j-1] \end{cases}$$

Tabella di programmazione dinamica di taglia $(m+1) \times (n+1)$

$$L[i][j] = \text{LCS}(i, j)$$

Ordine riempimento?

	0	1	2	j	n
0	0	0	0	0	0
1	0				
2	0				
i	0				
	0				
m	0				

3. Calcolo bottom-up

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i-1, j-1) + 1 & \text{se } i, j > 0 \text{ e } a[i-1] = b[j-1] \\ \max \{ L(i, j-1), L(i-1, j) \} & \text{se } i, j > 0 \text{ e } a[i-1] \neq b[j-1] \end{cases}$$

Tabella di programmazione dinamica di taglia $(m+1) \times (n+1)$

$$L[i][j] = \text{LCS}(i, j)$$

Ordine riempimento?

	0	1	2	j	n
0	0	0	0	0	0
1	0				
2	0				
i	0				
	0				
m	0				

3. Calcolo bottom-up

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i-1, j-1) + 1 & \text{se } i, j > 0 \text{ e } a[i-1] = b[j-1] \\ \max \{ L(i, j-1), L(i-1, j) \} & \text{se } i, j > 0 \text{ e } a[i-1] \neq b[j-1] \end{cases}$$

Tabella di programmazione dinamica di taglia $(m+1) \times (n+1)$

$$L[i][j] = \text{LCS}(i, j)$$

Ordine riempimento?

	0	1	2	j	n
0	0	0	0	0	0
1	0				
2	0				
i	0				
	0				
m	0				

3. Calcolo bottom-up

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i-1, j-1) + 1 & \text{se } i, j > 0 \text{ e } a[i-1] = b[j-1] \\ \max \{ L(i, j-1), L(i-1, j) \} & \text{se } i, j > 0 \text{ e } a[i-1] \neq b[j-1] \end{cases}$$

Tabella di programmazione dinamica di taglia $(m+1) \times (n+1)$

$$L[i][j] = \text{LCS}(i, j)$$

Ordine riempimento?

per riga o
colonna

	0	1	2	j	n
0	0	0	0	0	0
1	0				
2	0				
i	0				
	0				
m	0				

@2020
giuseppe.precipe@unipi.it

3. Calcolo bottom-up

```

1  LCS( a, b ):                                     <pre: a e b sono di lunghezza m e n>
2      FOR (i = 0; i <= m; i = i+1)
3          lunghezza[i][0] = 0;
4      FOR (j = 0; j <= n; j = j+1)
5          lunghezza[0][j] = 0;
6      FOR (i = 1; i <= m; i = i+1)
7          FOR (j = 1; j <= n; j = j+1) {
8              IF (a[i-1] == b[j-1]) {
9                  lunghezza[i][j] = lunghezza[i-1][j-1] + 1;
10             } ELSE IF (lunghezza[i][j-1] > lunghezza[i-1][j]) {
11                 lunghezza[i][j] = lunghezza[i][j-1];
12             } ELSE {
13                 lunghezza[i][j] = lunghezza[i-1][j];
14             }
15         }
16     RETURN lunghezza[m][n];


```

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i-1, j-1) + 1 & \text{se } i, j > 0 \text{ e } a[i-1] = b[j-1] \\ \max \{ L(i, j-1), L(i-1, j) \} & \text{se } i, j > 0 \text{ e } a[i-1] \neq b[j-1] \end{cases}$$

Complessità

- **Domanda:** (solita)....complessità??


```
1 LCS( a, b ):                                     <pre: a e b sono di lunghezza m e n>
2   FOR (i = 0; i <= m; i = i+1)
3     lunghezza[i][0] = 0;
4   FOR (j = 0; j <= n; j = j+1)
5     lunghezza[0][j] = 0;
6   FOR (i = 1; i <= m; i = i+1)
7     FOR (j = 1; j <= n; j = j+1) {
8       IF (a[i-1] == b[j-1]) {
9         lunghezza[i][j] = lunghezza[i-1][j-1] + 1;
10      } ELSE IF (lunghezza[i][j-1] > lunghezza[i-1][j]) {
11        lunghezza[i][j] = lunghezza[i][j-1];
12      } ELSE {
13        lunghezza[i][j] = lunghezza[i-1][j];
14      }
15    }
16   RETURN lunghezza[m][n];
```



Complessità

- **Domanda:** (solita)....complessità??

```
1 LCS( a, b ):                                     <pre: a e b sono di lunghezza m e n>
2   FOR (i = 0; i <= m; i = i+1)
3     lunghezza[i][0] = 0;
4   FOR (j = 0; j <= n; j = j+1)
5     lunghezza[0][j] = 0;
6   FOR (i = 1; i <= m; i = i+1)
7     FOR (j = 1; j <= n; j = j+1) {
8       IF (a[i-1] == b[j-1]) {
9         lunghezza[i][j] = lunghezza[i-1][j-1] + 1;
10      } ELSE IF (lunghezza[i][j-1] > lunghezza[i-1][j]) {
11        lunghezza[i][j] = lunghezza[i][j-1];
12      } ELSE {
13        lunghezza[i][j] = lunghezza[i-1][j];
14      }
15    }
16   RETURN lunghezza[m][n];
```



- **Tempo** $O(mn)$
- **Spazio** $O(mn)$

Complessità

- **Domanda:** (solita)....complessità??

```
1 LCS( a, b ):                                <pre: a e b sono di lunghezza m e n>
2   FOR (i = 0; i <= m; i = i+1)
3     lunghezza[i][0] = 0;
4   FOR (j = 0; j <= n; j = j+1)
5     lunghezza[0][j] = 0;
6   FOR (i = 1; i <= m; i = i+1)
7     FOR (j = 1; j <= n; j = j+1) {
8       IF (a[i-1] == b[j-1]) {
9         lunghezza[i][j] = lunghezza[i-1][j-1] + 1;
10      } ELSE IF (lunghezza[i][j-1] > lunghezza[i-1][j]) {
11        lunghezza[i][j] = lunghezza[i][j-1];
12      } ELSE {
13        lunghezza[i][j] = lunghezza[i-1][j];
14      }
15    }
16   RETURN lunghezza[m][n];
```

- Tempo $O(mn)$
- Spazio $O(mn)$

È possibile ridurre lo spazio ????

Complessità

- **Domanda:** (solita)....complessità??

```
1 LCS( a, b ):                                <pre: a e b sono di lunghezza m e n>
2   FOR (i = 0; i <= m; i = i+1)
3     lunghezza[i][0] = 0;
4   FOR (j = 0; j <= n; j = j+1)
5     lunghezza[0][j] = 0;
6   FOR (i = 1; i <= m; i = i+1)
7     FOR (j = 1; j <= n; j = j+1) {
8       IF (a[i-1] == b[j-1]) {
9         lunghezza[i][j] = lunghezza[i-1][j-1] + 1;
10      } ELSE IF (lunghezza[i][j-1] > lunghezza[i-1][j]) {
11        lunghezza[i][j] = lunghezza[i][j-1];
12      } ELSE {
13        lunghezza[i][j] = lunghezza[i-1][j];
14      }
15    }
16   RETURN lunghezza[m][n];
```

- Tempo $O(mn)$
- Spazio $O(mn)$

È possibile ridurre lo spazio a $O(n)$,
usando solo le ultime due righe (o
colonne)

4. Come ricostruire la sequenza più lunga?

Bisogna utilizzare una matrice **indice** simile a quella utilizzata per la sequenza ottima di moltiplicazioni

@2020
giuseppe.prencepi@unipi.it

Domanda: dove inseriamo il calcolo di **indice** nel codice visto?

Calcolo di **indice**

```
1  LCS( a, b ):                                     <pre: a e b sono di lunghezza m e n>
2      FOR (i = 0; i <= m; i = i+1)
3          lunghezza[i][0] = 0;
4      FOR (j = 0; j <= n; j = j+1)
5          lunghezza[0][j] = 0;
6      FOR (i = 1; i <= m; i = i+1)
7          FOR (j = 1; j <= n; j = j+1) {
8              IF (a[i-1] == b[j-1]) {
9                  lunghezza[i][j] = lunghezza[i-1][j-1] + 1;
10             } ELSE IF (lunghezza[i][j-1] > lunghezza[i-1][j]) {
11                 lunghezza[i][j] = lunghezza[i][j-1];
12             } ELSE {
13                 lunghezza[i][j] = lunghezza[i-1][j];
14             }
15         }
16     RETURN lunghezza[m][n];
```

@20
giuseppe.prencepi@unipi.it

Calcolo di **indice**

Domanda: dove inseriamo il calcolo di **indice** nel codice visto?

```
1  LCS( a, b ):                                     <pre: a e b sono di lunghezza m e n>
2      FOR (i = 0; i <= m; i = i+1)
3          lunghezza[i][0] = 0;
4      FOR (j = 0; j <= n; j = j+1)
5          lunghezza[0][j] = 0;
6      FOR (i = 1; i <= m; i = i+1)
7          FOR (j = 1; j <= n; j = j+1) {
8              IF (a[i-1] == b[j-1]) {
9                  lunghezza[i][j] = lunghezza[i-1][j-1] + 1;
10             } ELSE IF (lunghezza[i][j-1] > lunghezza[i-1][j]) {
11                 lunghezza[i][j] = lunghezza[i][j-1];
12             } ELSE {
13                 lunghezza[i][j] = lunghezza[i-1][j];
14             }
15         }
16     RETURN lunghezza[m][n];
```

Diagram illustrating the assignment of the **indice** value:

- For the case where $a[i-1] == b[j-1]$, the assignment is $\text{indice}[i][j] = \langle i-1, j-1 \rangle$.
- For the case where $\text{lunghezza}[i][j-1] > \text{lunghezza}[i-1][j]$, the assignment is $\text{indice}[i][j] = \langle i, j-1 \rangle$.
- For the case where $\text{lunghezza}[i-1][j] > \text{lunghezza}[i][j-1]$, the assignment is $\text{indice}[i][j] = \langle i-1, j \rangle$.

@2020
giuseppe.precipe@unipi.it

Procedura di Stampa

```
Stampa_LCS(i, j) {
    IF ((i > 0) && (j > 0)) {
        <i', j'> = indice[i][j];
        Stampa_LCS(i', j');
        PRINT a[i-1];
    }
}
```

In quale caso stampo i caratteri in sequenza?

Procedura di Stampa

```

Stampa_LCS(i, j) {
  IF ((i > 0) && (j > 0)) {
    <i', j'> = indice[i][j];
    Stampa_LCS(i', j');
    IF ((i' == i - 1) && (j' == j - 1)) PRINT a[i - 1];
  }
}

```

Caso in cui i caratteri sono uguali!

@2020
giuseppe.precipec@unipi.it

Esempio

$X = \langle A, B, C, B, D, A \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

If $x_i = y_j$

$\text{indice}[i, j] = "\diagdown"$

Else if $L[i - 1, j] \geq L[i, j - 1]$

$\text{indice}[i, j] = "\uparrow"$

else

$\text{indice}[i, j] = "\leftarrow"$

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i - 1, j - 1) + 1 & \text{se } i, j > 0 \text{ e } a[i - 1] = b[j - 1] \\ \max \{ L(i, j - 1), L(i - 1, j) \} & \text{se } i, j > 0 \text{ e } a[i - 1] \neq b[j - 1] \end{cases}$$

Riempiamo per righe....

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

@2020
giuseppe.precipec@unipi.it

Esempio

$X = \langle A, B, C, B, D, A \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

If $x_i = y_j$

$\text{indice}[i, j] = "\diagdown"$

Else if $L[i - 1, j] \geq L[i, j - 1]$

$\text{indice}[i, j] = "\uparrow"$

else

$\text{indice}[i, j] = "\leftarrow"$

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i - 1, j - 1) + 1 & \text{se } i, j > 0 \text{ e } a[i - 1] = b[j - 1] \\ \max \{ L(i, j - 1), L(i - 1, j) \} & \text{se } i, j > 0 \text{ e } a[i - 1] \neq b[j - 1] \end{cases}$$

Riempiamo per righe....

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

Esempio

$X = \langle A, B, C, B, D, A \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

If $x_i = y_j$

$\text{indice}[i, j] = "\diagdown"$

Else if $L[i - 1, j] \geq L[i, j - 1]$

$\text{indice}[i, j] = "\uparrow"$

else

$\text{indice}[i, j] = "\leftarrow"$

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i - 1, j - 1) + 1 & \text{se } i, j > 0 \text{ e } a[i - 1] = b[j - 1] \\ \max \{ L(i, j - 1), L(i - 1, j) \} & \text{se } i, j > 0 \text{ e } a[i - 1] \neq b[j - 1] \end{cases}$$

Riempiamo per righe....

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

Esempio

$X = \langle A, B, C, B, D, A \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

If $x_i = y_j$

$\text{indice}[i, j] = "\diagdown"$

Else if $L[i - 1, j] \geq L[i, j - 1]$

$\text{indice}[i, j] = "\uparrow"$

else

$\text{indice}[i, j] = "\leftarrow"$

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i - 1, j - 1) + 1 & \text{se } i, j > 0 \text{ e } a[i - 1] = b[j - 1] \\ \max \{ L(i, j - 1), L(i - 1, j) \} & \text{se } i, j > 0 \text{ e } a[i - 1] \neq b[j - 1] \end{cases}$$

Riempiamo per righe....

		0	1	2	3	4	5	6
		Y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	\uparrow 0					
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Esempio

$X = \langle A, B, C, B, D, A \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

If $x_i = y_j$

$\text{indice}[i, j] = "\diagdown"$

Else if $L[i - 1, j] \geq L[i, j - 1]$

$\text{indice}[i, j] = "\uparrow"$

else

$\text{indice}[i, j] = "\leftarrow"$

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i - 1, j - 1) + 1 & \text{se } i, j > 0 \text{ e } a[i - 1] = b[j - 1] \\ \max \{ L(i, j - 1), L(i - 1, j) \} & \text{se } i, j > 0 \text{ e } a[i - 1] \neq b[j - 1] \end{cases}$$

Riempiamo per righe....

		0	1	2	3	4	5	6
	y_j	B	D	C	A	B	A	
0	x_i	0	0	0	0	0	0	
1	A	0	\uparrow 0	\uparrow 0				
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Esempio

$X = \langle A, B, C, B, D, A \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

If $x_i = y_j$

$\text{indice}[i, j] = "\diagdown"$

Else if $L[i - 1, j] \geq L[i, j - 1]$

$\text{indice}[i, j] = "\uparrow"$

else

$\text{indice}[i, j] = "\leftarrow"$

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i - 1, j - 1) + 1 & \text{se } i, j > 0 \text{ e } a[i - 1] = b[j - 1] \\ \max \{ L(i, j - 1), L(i - 1, j) \} & \text{se } i, j > 0 \text{ e } a[i - 1] \neq b[j - 1] \end{cases}$$

Riempiamo per righe....

		0	1	2	3	4	5	6
	y_j	B	D	C	A	B	A	
0	x_i	0	0	0	0	0	0	
1	A	0	\uparrow 0	\uparrow 0	\uparrow 0			
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Esempio

$X = \langle A, B, C, B, D, A \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

If $x_i = y_j$

$\text{indice}[i, j] = "\diagdown"$

Else if $L[i - 1, j] \geq L[i, j - 1]$

$\text{indice}[i, j] = "\uparrow"$

else

$\text{indice}[i, j] = "\leftarrow"$

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i - 1, j - 1) + 1 & \text{se } i, j > 0 \text{ e } a[i - 1] = b[j - 1] \\ \max \{ L(i, j - 1), L(i - 1, j) \} & \text{se } i, j > 0 \text{ e } a[i - 1] \neq b[j - 1] \end{cases}$$

Riempiamo per righe....

		0	1	2	3	4	5	6
		Y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	\uparrow 0	\uparrow 0	\uparrow 0	\diagdown 1		
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Esempio

$X = \langle A, B, C, B, D, A \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

If $x_i = y_j$

$\text{indice}[i, j] = "\diagdown"$

Else if $L[i - 1, j] \geq L[i, j-1]$

$\text{indice}[i, j] = "\uparrow"$

else

$\text{indice}[i, j] = "\leftarrow"$

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i-1, j-1) + 1 & \text{se } i, j > 0 \text{ e } a[i-1] = b[j-1] \\ \max \{ L(i, j-1), L(i-1, j) \} & \text{se } i, j > 0 \text{ e } a[i-1] \neq b[j-1] \end{cases}$$

Riempiamo per righe....

		0	1	2	3	4	5	6
		Y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	\uparrow 0	\uparrow 0	\uparrow 0	\diagdown 1	\leftarrow 1	
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Esempio

$X = \langle A, B, C, B, D, A \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

If $x_i = y_j$

$\text{indice}[i, j] = "\diagdown"$

Else if $L[i - 1, j] \geq L[i, j-1]$

$\text{indice}[i, j] = "\uparrow"$

else

$\text{indice}[i, j] = "\leftarrow"$

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i-1, j-1) + 1 & \text{se } i, j > 0 \text{ e } a[i-1] = b[j-1] \\ \max \{ L(i, j-1), L(i-1, j) \} & \text{se } i, j > 0 \text{ e } a[i-1] \neq b[j-1] \end{cases}$$

Riempiamo per righe....

		0	1	2	3	4	5	6
	y_j	B	D	C	A	B	A	
0	x_i	0	0	0	0	0	0	
1	A	0	\uparrow 0	\uparrow 0	\uparrow 0	\diagdown 1	\leftarrow 1	
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Esempio

$X = \langle A, B, C, B, D, A \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

If $x_i = y_j$

$\text{indice}[i, j] = "\diagdown"$

Else if $L[i - 1, j] \geq L[i, j-1]$

$\text{indice}[i, j] = "\uparrow"$

else

$\text{indice}[i, j] = "\leftarrow"$

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i - 1, j - 1) + 1 & \text{se } i, j > 0 \text{ e } a[i - 1] = b[j - 1] \\ \max \{ L(i, j - 1), L(i - 1, j) \} & \text{se } i, j > 0 \text{ e } a[i - 1] \neq b[j - 1] \end{cases}$$

Riempiamo per righe....

		0	1	2	3	4	5	6
		Y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1
2	B	0	\swarrow 1					
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Esempio

$X = \langle A, B, C, B, D, A \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

If $x_i = y_j$

$\text{indice}[i, j] = "\diagdown"$

Else if $L[i - 1, j] \geq L[i, j-1]$

$\text{indice}[i, j] = "\uparrow"$

else

$\text{indice}[i, j] = "\leftarrow"$

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i - 1, j - 1) + 1 & \text{se } i, j > 0 \text{ e } a[i - 1] = b[j - 1] \\ \max \{ L(i, j - 1), L(i - 1, j) \} & \text{se } i, j > 0 \text{ e } a[i - 1] \neq b[j - 1] \end{cases}$$

Riempiamo per righe....

		0	1	2	3	4	5	6
	y_j	B	D	C	A	B	A	
0	x_i	0	0	0	0	0	0	
1	A	0	\uparrow 0	\uparrow 0	\uparrow 0	\diagdown 1	\leftarrow 1	
2	B	0	\diagdown 1	\leftarrow 1				
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Esempio

$X = \langle A, B, C, B, D, A \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

If $x_i = y_j$

$\text{indice}[i, j] = "\diagdown"$

Else if $L[i - 1, j] \geq L[i, j - 1]$

$\text{indice}[i, j] = "\uparrow"$

else

$\text{indice}[i, j] = "\leftarrow"$

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i - 1, j - 1) + 1 & \text{se } i, j > 0 \text{ e } a[i - 1] = b[j - 1] \\ \max \{ L(i, j - 1), L(i - 1, j) \} & \text{se } i, j > 0 \text{ e } a[i - 1] \neq b[j - 1] \end{cases}$$

Riempiamo per righe....

		0	1	2	3	4	5	6
		Y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1
2	B	0	\swarrow 1	\leftarrow 1	\leftarrow 1			
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Esempio

$X = \langle A, B, C, B, D, A \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

If $x_i = y_j$

$\text{indice}[i, j] = "\diagdown"$

Else if $L[i - 1, j] \geq L[i, j - 1]$

$\text{indice}[i, j] = "\uparrow"$

else

$\text{indice}[i, j] = "\leftarrow"$

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i - 1, j - 1) + 1 & \text{se } i, j > 0 \text{ e } a[i - 1] = b[j - 1] \\ \max \{ L(i, j - 1), L(i - 1, j) \} & \text{se } i, j > 0 \text{ e } a[i - 1] \neq b[j - 1] \end{cases}$$

Riempiamo per righe....

		0	1	2	3	4	5	6
	y_j	B	D	C	A	B	A	
0	x_i	0	0	0	0	0	0	
1	A	0	\uparrow 0	\uparrow 0	\uparrow 0	\diagdown 1	\leftarrow 1	
2	B	0	\diagdown 1	\leftarrow 1	\leftarrow 1	\uparrow 1		
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Esempio

$X = \langle A, B, C, B, D, A \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

If $x_i = y_j$

$\text{indice}[i, j] = "\diagdown"$

Else if $L[i - 1, j] \geq L[i, j - 1]$

$\text{indice}[i, j] = "\uparrow"$

else

$\text{indice}[i, j] = "\leftarrow"$

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i - 1, j - 1) + 1 & \text{se } i, j > 0 \text{ e } a[i - 1] = b[j - 1] \\ \max \{ L(i, j - 1), L(i - 1, j) \} & \text{se } i, j > 0 \text{ e } a[i - 1] \neq b[j - 1] \end{cases}$$

Riempiamo per righe....

		0	1	2	3	4	5	6
		Y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1
2	B	0	\swarrow 1	\leftarrow 1	\leftarrow 1	\uparrow 1	\swarrow 2	
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Esempio

$X = \langle A, B, C, B, D, A \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

If $x_i = y_j$

$\text{indice}[i, j] = "\diagdown"$

Else if $L[i - 1, j] \geq L[i, j - 1]$

$\text{indice}[i, j] = "\uparrow"$

else

$\text{indice}[i, j] = "\leftarrow"$

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i - 1, j - 1) + 1 & \text{se } i, j > 0 \text{ e } a[i - 1] = b[j - 1] \\ \max \{ L(i, j - 1), L(i - 1, j) \} & \text{se } i, j > 0 \text{ e } a[i - 1] \neq b[j - 1] \end{cases}$$

Riempiamo per righe....

		0	1	2	3	4	5	6
	y_j	B	D	C	A	B	A	
0	x_i	0	0	0	0	0	0	
1	A	0	\uparrow 0	\uparrow 0	\uparrow 0	\diagdown 1	\leftarrow 1	
2	B	0	\diagdown 1	\leftarrow 1	\leftarrow 1	\uparrow 1	\diagdown 2	
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Esempio

$X = \langle A, B, C, B, D, A \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

If $x_i = y_j$

$\text{indice}[i, j] = "\diagdown"$

Else if $L[i - 1, j] \geq L[i, j - 1]$

$\text{indice}[i, j] = "\uparrow"$

else

$\text{indice}[i, j] = "\leftarrow"$

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i - 1, j - 1) + 1 & \text{se } i, j > 0 \text{ e } a[i - 1] = b[j - 1] \\ \max \{ L(i, j - 1), L(i - 1, j) \} & \text{se } i, j > 0 \text{ e } a[i - 1] \neq b[j - 1] \end{cases}$$

Riempiamo per righe....

		0	1	2	3	4	5	6
		Y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1
2	B	0	\swarrow 1	\leftarrow 1	\leftarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2
3	C	0	\uparrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\uparrow 2	\uparrow 2
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Esempio

$X = \langle A, B, C, B, D, A \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

If $x_i = y_j$

$\text{indice}[i, j] = "\diagdown"$

Else if $L[i - 1, j] \geq L[i, j - 1]$

$\text{indice}[i, j] = "\uparrow"$

else

$\text{indice}[i, j] = "\leftarrow"$

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i - 1, j - 1) + 1 & \text{se } i, j > 0 \text{ e } a[i - 1] = b[j - 1] \\ \max \{ L(i, j - 1), L(i - 1, j) \} & \text{se } i, j > 0 \text{ e } a[i - 1] \neq b[j - 1] \end{cases}$$

Riempiamo per righe....

		0	1	2	3	4	5	6
	y_j	B	D	C	A	B	A	
0	x_i	0	0	0	0	0	0	
1	A	0	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1
2	B	0	\swarrow 1	\leftarrow 1	\leftarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2
3	C	0	\uparrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\uparrow 2	\uparrow 2
4	B	0	\swarrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow 3	\leftarrow 3
5	D	0						
6	A	0						
7	B	0						

Esempio

$X = \langle A, B, C, B, D, A \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

If $x_i = y_j$

$\text{indice}[i, j] = "\diagdown"$

Else if $L[i - 1, j] \geq L[i, j - 1]$

$\text{indice}[i, j] = "\uparrow"$

else

$\text{indice}[i, j] = "\leftarrow"$

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i - 1, j - 1) + 1 & \text{se } i, j > 0 \text{ e } a[i - 1] = b[j - 1] \\ \max \{ L(i, j - 1), L(i - 1, j) \} & \text{se } i, j > 0 \text{ e } a[i - 1] \neq b[j - 1] \end{cases}$$

Riempiamo per righe....

		0	1	2	3	4	5	6
	Y_j	B	D	C	A	B	A	
0	x_i	0	0	0	0	0	0	
1	A	0	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1
2	B	0	\swarrow 1	\leftarrow 1	\leftarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2
3	C	0	\uparrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\uparrow 2	\uparrow 2
4	B	0	\swarrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow 3	\leftarrow 3
5	D	0	\uparrow 1	\swarrow 2	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 3
6	A	0						
7	B	0						

Esempio

$X = \langle A, B, C, B, D, A \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

If $x_i = y_j$

$\text{indice}[i, j] = "\diagdown"$

Else if $L[i - 1, j] \geq L[i, j - 1]$

$\text{indice}[i, j] = "\uparrow"$

else

$\text{indice}[i, j] = "\leftarrow"$

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i - 1, j - 1) + 1 & \text{se } i, j > 0 \text{ e } a[i - 1] = b[j - 1] \\ \max \{ L(i, j - 1), L(i - 1, j) \} & \text{se } i, j > 0 \text{ e } a[i - 1] \neq b[j - 1] \end{cases}$$

Riempiamo per righe....

		0	1	2	3	4	5	6
	y_j	B	D	C	A	B	A	
0	x_i	0	0	0	0	0	0	
1	A	0	\uparrow 0	\uparrow 0	\uparrow 0	\diagdown 1	\leftarrow 1	\diagdown 1
2	B	0	\diagdown 1	\leftarrow 1	\leftarrow 1	\uparrow 1	\diagdown 2	\leftarrow 2
3	C	0	\uparrow 1	\uparrow 1	\diagdown 2	\leftarrow 2	\uparrow 2	\uparrow 2
4	B	0	\diagdown 1	\uparrow 1	\uparrow 2	\uparrow 2	\diagdown 3	\leftarrow 3
5	D	0	\uparrow 1	\diagdown 2	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 3
6	A	0	\uparrow 1	\uparrow 2	\uparrow 2	\diagdown 3	\uparrow 3	\diagdown 4
7	B	0						

Esempio

$X = \langle A, B, C, B, D, A \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

If $x_i = y_j$

$\text{indice}[i, j] = "\diagdown"$

Else if $L[i - 1, j] \geq L[i, j - 1]$

$\text{indice}[i, j] = "\uparrow"$

else

$\text{indice}[i, j] = "\leftarrow"$

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i - 1, j - 1) + 1 & \text{se } i, j > 0 \text{ e } a[i - 1] = b[j - 1] \\ \max \{ L(i, j - 1), L(i - 1, j) \} & \text{se } i, j > 0 \text{ e } a[i - 1] \neq b[j - 1] \end{cases}$$

Riempiamo per righe....

		0	1	2	3	4	5	6
	y_j	B	D	C	A	B	A	
0	x_i	0	0	0	0	0	0	
1	A	0	\uparrow 0	\uparrow 0	\uparrow 0	\diagdown 1	\leftarrow 1	\diagdown 1
2	B	0	\diagdown 1	\leftarrow 1	\leftarrow 1	\uparrow 1	\diagdown 2	\leftarrow 2
3	C	0	\uparrow 1	\uparrow 1	\diagdown 2	\leftarrow 2	\uparrow 2	\uparrow 2
4	B	0	\diagdown 1	\uparrow 1	\uparrow 2	\uparrow 2	\diagdown 3	\leftarrow 3
5	D	0	\uparrow 1	\diagdown 2	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 3
6	A	0	\uparrow 1	\uparrow 2	\uparrow 2	\diagdown 3	\uparrow 3	\diagdown 4
7	B	0	\diagdown 1	\uparrow 2	\uparrow 2	\uparrow 3	\diagdown 4	\uparrow 4

Esempio

$X = \langle A, B, C, B, D, A \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

If $x_i = y_j$

$\text{indice}[i, j] = "\diagdown"$

Else if $L[i - 1, j] \geq L[i, j - 1]$

$\text{indice}[i, j] = "\uparrow"$

else

$\text{indice}[i, j] = "\leftarrow"$

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i - 1, j - 1) + 1 & \text{se } i, j > 0 \text{ e } a[i - 1] = b[j - 1] \\ \max \{ L(i, j - 1), L(i - 1, j) \} & \text{se } i, j > 0 \text{ e } a[i - 1] \neq b[j - 1] \end{cases}$$

Riempiamo per righe....

		0	1	2	3	4	5	6
	y_j	B	D	C	A	B	A	
0	x_i	0	0	0	0	0	0	
1	A	0	\uparrow 0	\uparrow 0	\uparrow 0	\diagdown 1	\leftarrow 1	\diagdown 1
2	B	0	\diagdown 1	\leftarrow 1	\leftarrow 1	\uparrow 1	\diagdown 2	\leftarrow 2
3	C	0	\uparrow 1	\uparrow 1	\diagdown 2	\leftarrow 2	\uparrow 2	\uparrow 2
4	B	0	\diagdown 1	\uparrow 1	\uparrow 2	\uparrow 2	\diagdown 3	\leftarrow 3
5	D	0	\uparrow 1	\diagdown 2	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 3
6	A	0	\uparrow 1	\uparrow 2	\uparrow 2	\diagdown 3	\uparrow 3	\diagdown 4
7	B	0	\diagdown 1	\uparrow 2	\uparrow 2	\uparrow 3	\diagdown 4	\uparrow 4

Dove si trova
la **lunghezza**
LCS?

Esempio

$X = \langle A, B, C, B, D, A \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

If $x_i = y_j$

$\text{indice}[i, j] = \text{"\backslash"}$

Else if $L[i - 1, j] \geq L[i, j - 1]$

$\text{indice}[i, j] = \text{"\uparrow"}$

else

$\text{indice}[i, j] = \text{"\leftarrow"}$

$$L(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ L(i - 1, j - 1) + 1 & \text{se } i, j > 0 \text{ e } a[i - 1] = b[j - 1] \\ \max \{ L(i, j - 1), L(i - 1, j) \} & \text{se } i, j > 0 \text{ e } a[i - 1] \neq b[j - 1] \end{cases}$$

Riempiamo per righe....

		0	1	2	3	4	5	6
	y_j	B	D	C	A	B	A	
0	x_i	0	0	0	0	0	0	
1	A	0	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1
2	B	0	\swarrow 1	\leftarrow 1	\leftarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2
3	C	0	\uparrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\uparrow 2	\uparrow 2
4	B	0	\swarrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow 3	\leftarrow 3
5	D	0	\uparrow 1	\swarrow 2	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 3
6	A	0	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow 3	\uparrow 3	\swarrow 4
7	B	0	\swarrow 1	\uparrow 2	\uparrow 2	\uparrow 3	\swarrow 4	\uparrow 4

Dove si trova
la **lunghezza**
LCS?

Procedura di Stampa

		0	1	2	3	4	5	6
	y_j	B	D	C	A	B	A	
0	x_i	0	0	0	0	0	0	
1	A	0	\uparrow 0	\uparrow 0	\uparrow 0	\diagdown 1	\leftarrow 1	
2	B	0	\diagdown 1	\leftarrow 1	\leftarrow 1	\uparrow 1	\diagdown 2	
3	C	0	\uparrow 1	\uparrow 1	\diagdown 2	\leftarrow 2	\uparrow 2	
4	B	0	\diagdown 1	\uparrow 1	\uparrow 2	\uparrow 2	\diagdown 3	
5	D	0	\uparrow 1	\diagdown 2	\uparrow 2	\uparrow 2	\uparrow 3	
6	A	0	\uparrow 1	\uparrow 2	\uparrow 2	\diagdown 3	\uparrow 3	
7	B	0	\diagdown 1	\uparrow 2	\uparrow 2	\uparrow 3	\diagdown 4	

Procedura di Stampa

- Inizia da **indice**[m, n] e segui le frecce
- Quando si incontra un “ \ ” in **indice**[i, j]
 $\Rightarrow x_i = y_j$ è un elemento della LCS

LCS:

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

Procedura di Stampa

- Inizia da **indice**[m, n] e segui le frecce
- Quando si incontra un “ \ ” in **indice**[i, j]
 $\Rightarrow x_i = y_j$ è un elemento della LCS

LCS:

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

Procedura di Stampa

- Inizia da **indice**[m, n] e segui le frecce
- Quando si incontra un “ \ ” in **indice**[i, j]
 $\Rightarrow x_i = y_j$ è un elemento della LCS

LCS:

A

@2020
giuseppe.prencipe@unipi.it

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

Procedura di Stampa

- Inizia da **indice**[m, n] e segui le frecce
- Quando si incontra un “ \ ” in **indice**[i, j]
 $\Rightarrow x_i = y_j$ è un elemento della LCS

LCS:

A

@2020
giuseppe.prencipe@unipi.it

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

Procedura di Stampa

- Inizia da **indice[m, n]** e segui le frecce
- Quando si incontra un “ \ ” in **indice[i, j]**
 $\Rightarrow x_i = y_j$ è un elemento della LCS

LCS: **B A**

@2020
giuseppe.prencipe@unipi.it

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

Procedura di Stampa

- Inizia da **indice[m, n]** e segui le frecce
- Quando si incontra un “ \ ” in **indice[i, j]**
 $\Rightarrow x_i = y_j$ è un elemento della LCS

LCS: **B A**

@2020
giuseppe.prencipe@unipi.it

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

Procedura di Stampa

- Inizia da **indice**[m, n] e segui le frecce
- Quando si incontra un “ \ ” in **indice**[i, j]
 $\Rightarrow x_i = y_j$ è un elemento della LCS

LCS: **C B A**

@2020
giuseppe.prencipe@unipi.it

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

Procedura di Stampa

- Inizia da **indice**[m, n] e segui le frecce
- Quando si incontra un “ \ ” in **indice**[i, j]
 $\Rightarrow x_i = y_j$ è un elemento della LCS

LCS: **C B A**

@2020
giuseppe.prencipe@unipi.it

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

Procedura di Stampa

- Inizia da **indice[m, n]** e segui le frecce
- Quando si incontra un “\” in **indice[i, j]**
 $\Rightarrow x_i = y_j$ è un elemento della LCS

LCS: **B C B A**

@2020
giuseppe.prencipe@unipi.it

		0	1	2	3	4	5	6
	Y_j	B	D	C	A	B	A	
0	x_i	0	0	0	0	0	0	
1	A	0	↑ 0	↑ 0	↑ 0	↘ 1	←1	↘ 1
2	B	0	↖1	↖1	←1	↑ 1	↘ 2	←2
3	C	0	↑ 1	↑ 1	↖2	↖2	↑ 2	↑ 2
4	B	0	↘ 1	↑ 1	↑ 2	↑ 2	↖3	←3
5	D	0	↑ 1	↘ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↘ 3	↑ 3	↖4
7	B	0	↘ 1	↑ 2	↑ 2	↑ 3	↘ 4	↑ 4

Procedura di Stampa

- Inizia da **indice[m, n]** e segui le frecce
- Quando si incontra un “\” in **indice[i, j]**
 $\Rightarrow x_i = y_j$ è un elemento della LCS

LCS: **B C B A**

@2020
giuseppe.prencipe@unipi.it

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

Procedura di Stampa Alternativa

```
1.  if i = 0 or j = 0
2.    then return
3.  if b[i, j] = "\\"
4.    then Stampa_LCS(indice, A, i - 1, j - 1)
5.    print xi
6.  elseif indice[i, j] = "↑"
7.    then Stampa_LCS(indice, A, i - 1, j)
8.    else Stampa_LCS(indice, A, i, j - 1)
```

Tempo: $\Theta(????)$

@2020 Chiamata iniziale: Stampa_LCS(indice, A, length[X], length[Y])
giuseppe.preciupo@unipi.it

Procedura di Stampa Alternativa

```
1.  if i = 0 or j = 0
2.    then return
3.  if b[i, j] = "\\"
4.    then Stampa_LCS(indice, A, i - 1, j - 1)
5.    print xi
6.  elseif indice[i, j] = "↑"
7.    then Stampa_LCS(indice, A, i - 1, j)
8.    else Stampa_LCS(indice, A, i, j - 1)
```

Tempo: $\Theta(m+n)$

@2020 Chiamata iniziale: Stampa_LCS(indice, A, length[X], length[Y])
giuseppe.preciupo@unipi.it

Migliorare il codice

@2020
giuseppe.prencipe@unipi.it

Migliorare il codice

- Necessario tenere tutta la tabella **L** per calcolare LCS?

@2020
giuseppe.prencipe@unipi.it

Migliorare il codice

- Necessario tenere tutta la tabella **L** per calcolare LCS?
- Anche qui possiamo tenere solo le ultime due righe/colonne

Migliorare il codice

Migliorare il codice

- Non è necessario tenere tutta la tabella **L**, ma solo le ultime due righe/colonne

Migliorare il codice

- Non è necessario tenere tutta la tabella **L**, ma solo le ultime due righe/colonne
- Ma la tabella **L** serve per restituire la sequenza!

Migliorare il codice

- Non è necessario tenere tutta la tabella **L**, ma solo le ultime due righe/colonne
- Ma la tabella **L** serve per restituire la sequenza!
- In questo caso, possiamo risparmiare qualcosaCome calcoliamo ogni $L[i, j]$?
 - $L[i, j]$ dipende solo da $L[i - 1, j - 1]$, $L[i - 1, j]$, e $L[i, j - 1]$
 - Quindi, possiamo eliminare la tabella **indice** e calcolare in tempo costante quale dei tre valori è stato utilizzato per calcolare $L[i, j]$, durante la procedura di stampa stessa
 - Si risparmia lo spazio della tabella **indice**
 - In ogni caso la necessità di spazio non è ridotta **asintoticamente**: infatti, c'è sempre bisogno della tabella **L**