Algoritmi e Strutture Dati (Mod. B)

Programmazione Dinamica
(Parte I)

Numeri di Fibonacci

Definizione ricorsiva (o induttiva)

- $F(1) = F(0) = 1$
- $F(n) = F(n\text{-}1) + F(n\text{-}2)$

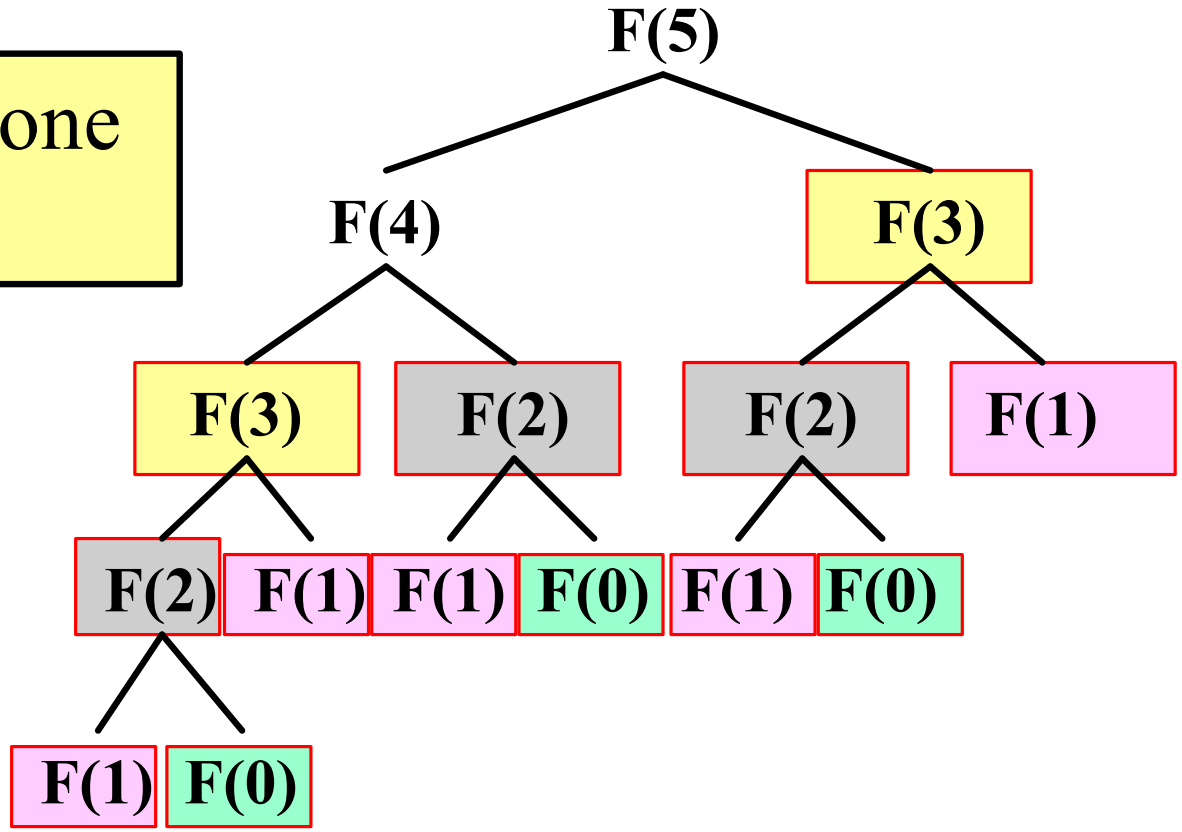
Algoritmo ricorsivo

```
Fib(n: intero)
  if n = 0 or n = 1
   then return 1
   else return Fib(n-1) + Fib(n-2)
```

Tempo di esecuzione dell'algoritmo

$$T(n) = \begin{cases} O(1) & \text{se } n \leq 1 \\ T(n-1) + T(n-2) & \text{se } n \geq 2 \end{cases}$$

Tempo di esecuzione è $O(2^n)$



Algoritmo II

```
Fib(n:intero)
   f[0] = 1
   f[1] = 1
   for i=2 to n
      f[i] = f[i-1] + f[i-2]
   return f[n]
```

Un array $f[]$ di dimensione n.

La complessità in tempo è $O(n)$.

La complessità in spazio è $O(n)$.

n	0	1	2	3	4	5	6	7
$f[]$	1	1	2	3	5	8	13	21

Algoritmo II

```
Fib(n:intero)
   f[0] = 1
   f[1] = 1
   for i=2 to n
      f[i] = f[i-1] + f[i-2]
   return f[n]
```

Un array $f[]$ di dimensione n.

La complessità in tempo è $O(n)$.

La complessità in spazio è $O(n)$.

n	0	1	2	3	4	5	6	7
$f[]$	1	1	2	3	5	8	13	21

Algoritmo II

```
Fib(n:intero)
   f[0] = 1
   f[1] = 1
   for i=2 to n
      f[i] = f[i-1] + f[i-2]
   return f[n]
```

Un array $f[]$ di dimensione n.

La complessità in tempo è $O(n)$.

La complessità in spazio è $O(n)$.

n	0	1	2	3	4	5	6	7
$f[]$	1	1	2	3	5	8	13	21

Algoritmo II

```
Fib(n:intero)
   f[1] = f[2] = 1
   for i=2 to n
         f[0] = f[1]
         f[1] = f[2]
         f[2] = f[0] + f[1]
   return f[2]
```

Un array $f[]$ di dimensione 2.

La complessità in spazio è $O(1)$.

La complessità in tempo è $O(n)$.

n	0	1	2	3	4	5	6	7
$f[0]$	-	-	1	1	2	3	5	8
$f[1]$	1	1	1	2	3	5	8	13
$f[2]$	1	1	2	3	5	8	13	21

Programmazione Dinamica

- **Strategia sviluppata intorno agli anno ‘50 nel campo dei *problemi di ottimizzazione***

- **Applicazione nei casi in cui:**
  - **ci sia più di una soluzione al problema**
  - **alle soluzioni è associabile un indice di “*bontà*” (ad esempio: costo, preferenza, etc.)**
  - **si vuole determinare la soluzione con *indice ottimo* (la *soluzione ottima* del problema, rispetto all’indice di “bontà”)**

Programmazione Dinamica

- ***Caratterizzare*** **la** ***struttura*** **di una** ***soluzione ottima***
- ***Definire*** ***ricorsivamente*** **il valore di una** ***soluzione ottima***
  - **La soluzione ottima ad un problema contiene le soluzioni ottime ai sottoproblemi**

- ***Calcolare*** **il** ***valore di una soluzione*** **ottima "*bottom-up*" (cioè calcolando prima le soluzioni ai casi più semplici)**
  - **Si usa una tabella per memorizzare le soluzioni dei sottoproblemi**
  - **Evitare di ripetere il lavoro più volte: non ricalcolare le soluzioni di sottoproblemi già calcolate.**
- ***Costruire*** **la (una) soluzione ottima.**

Catena di moltiplcazione tra matrici

Problema: **Data una sequenza di matrici *compatibili 2 a 2 al prodotto*** $A_1, A_2, A_3, \ldots, A_n$, ***vogliamo calcolare il loro prodotto*****.**

La moltiplicazione di matrici si basa sulla *moltiplicazione scalare* come operazione elementare.

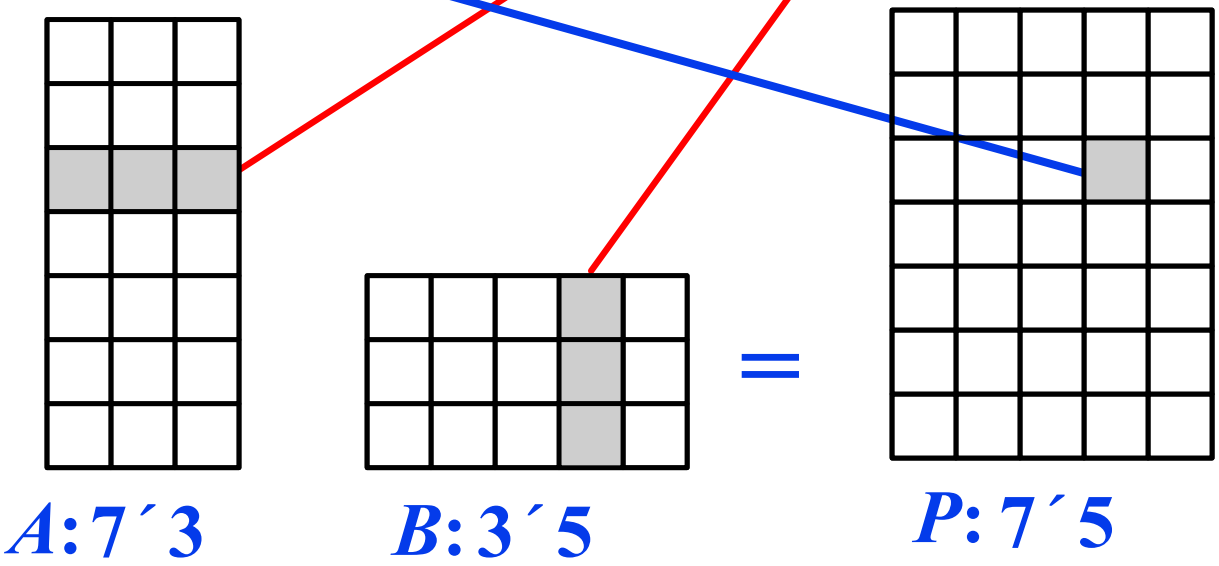
Vogliamo calcolare il prodotto impiegando il numero minore possibile di moltiplicazioni

Il prodotto di matrici *non è commutativo*...

...ma *è associativo* **[** $(A_1 \times A_2) \times A_3 = A_1 \times (A_2 \times A_3)$ **]**

Moltiplicazione tra matrici

$$P[i,j] = \sum_{k=1}^{c} A[i,k] \times B[k,j]$$



A: r ´ c

B: p ´ q **(ma deve valere $c = p$)**

A ´ B: r ´ q **: richiede r ´ c ´ q (r ´ p ´ q) moltiplicazioni scalari**

Moltiplicazione tra 2 matrici

```
Prod-Matrici(A[r,c],B[p,q],P[r,q]: matrice)
  if c ¹ p
    then ERRORE "dimensioni non compatibili"
         return
    else for i=1 to r do
             for j=1 to q do
                 sum=0
                 for k=1 to c do
                 sum=sum+A[i,k]´ B[k,j]
                 P[i,j]=sum
```

$$P[i,j] = \sum_{k=1}^{c} A[i,k] \times B[k,j]$$

Tempo di esecuzione = $\Theta(r \cdot c \cdot q)$ ⟹ $\Theta(n^3)$

Catena di moltiplcazione tra matrici

- **3 matrici:** **A** **B** **C**
- **Dimensioni:** **100´1 , 1´100 , 100´1**

• *((A B) C)*	*Num Moltiplicazioni*	*Memoria*
(A B)	100´1´100 = 10000	10000
((A B) C)	100´100´1 = 10000	100
	20000	10100
• *(A (B C))*		
(B C)	1´100´1 = 100	1
(A (B C))	100´1´1 = 100	100
	200	101

Catena di moltiplcazione tra matrici

- **4 matrici:** **A** **B** **C** **D**
- **Dimensioni:** **50´10, 10´40, 40´30, 30´5**

- ***(((A B) C) D)*** **: 87500 moltiplicazioni**

(A B)	**50´10´40 = 20000**
((A B) C)	**50´40´30 = 60000**
((A B) C) D	**50´30´ 5 = 7500**
	87500

Catena di moltiplcazione tra matrici

- **4 matrici:** **A** **B** **C** **D**
- **Dimensioni:** **50´10**, **10´40**, **40´30**, **30´5**

- ***(((A B) C) D)*** : **87500 moltiplicazioni**
- ***((A (B C)) D)*** : **34500 moltiplicazioni**
- ***((A B)(C D))*** : **36000 moltiplicazioni**
- ***(A ((B C) D))*** : **16000 moltiplicazioni**
- ***(A (B (C D)))*** : **10500 moltiplicazioni**

Criterio di scelta

- **Determinare il *numero di moltiplicazioni* scalari necessari per i prodotti tra le matrici *in ogni "parentesizzazione"***
- **Scegliere la parentesizzazione che richiede il *numero minimo di moltiplicazioni* (*criterio di ottimalità*)**

- **Ma quante sono le parentesizzazioni possibili?**

  per $n = 3$ sono 2

  per $n = 4$ sono 5

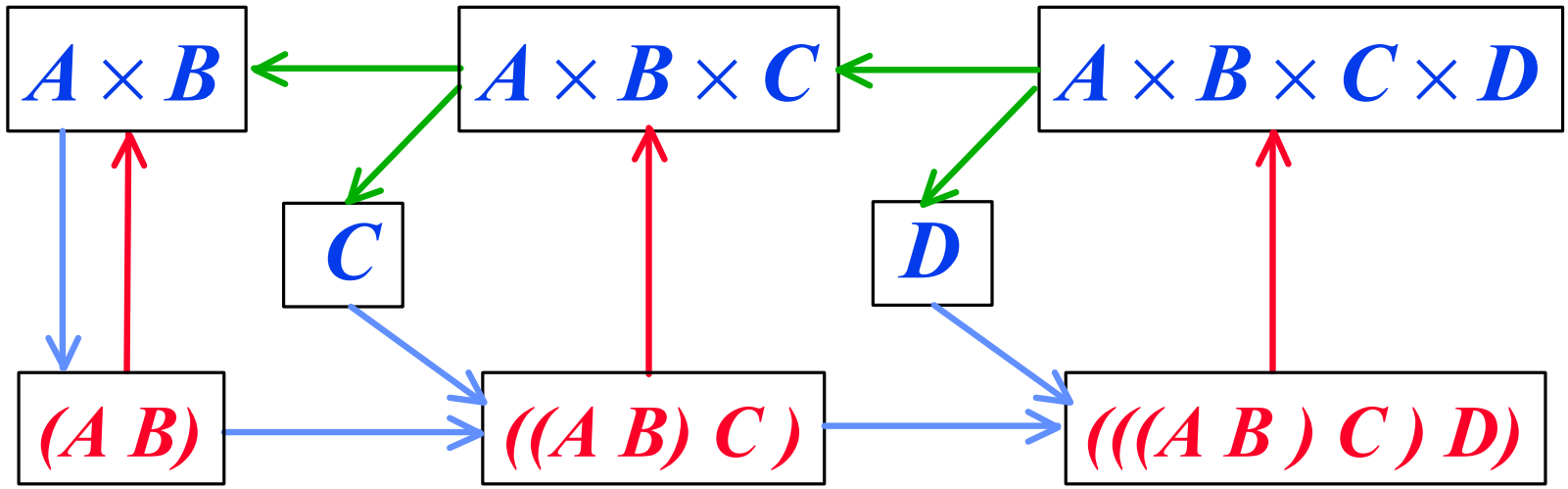
  per $n > 4$ quante sono?

Definizione di parentesizzazione

Definizione: **Un prodotto di matrici $A_1 \cdot A_2 \cdot A_3 \cdot \ldots \cdot A_n$ si dice *completamente parentesizzato* se:**

- **consiste di una unica matrice ($n = 1$) oppure**
- **per qualche $1 \leq k \leq n-1$, è il prodotto, delimitato da parentesi, tra i prodotti *completamente parentesizzati***

$$A_1 \cdot A_2 \cdot A_3 \cdot \ldots \cdot A_k \quad \text{e} \quad A_{k+1} \cdot A_2 \cdot A_3 \cdot \ldots \cdot A_n$$

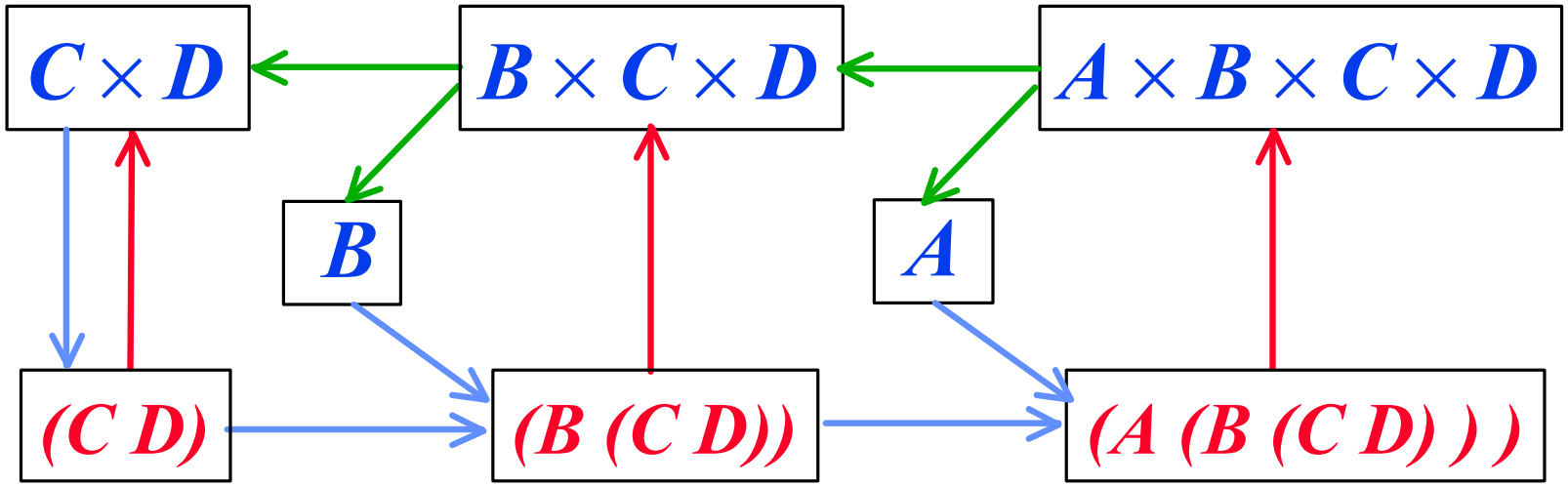


Definizione di parentesizzazione

Definizione: **Un prodotto di matrici $A_1 \cdot A_2 \cdot A_3 \cdot \ldots \cdot A_n$ si dice *completamente parentesizzato* se:**

- **consiste di una unica matrice ($n = 1$) oppure**
- **per qualche $1 \leq k \leq n-1$, è il prodotto, delimitato da parentesi, tra i prodotti *completamente parentesizzati***

$$A_1 \cdot A_2 \cdot A_3 \cdot \ldots \cdot A_k \quad \text{e} \quad A_{k+1} \cdot A_2 \cdot A_3 \cdot \ldots \cdot A_n$$

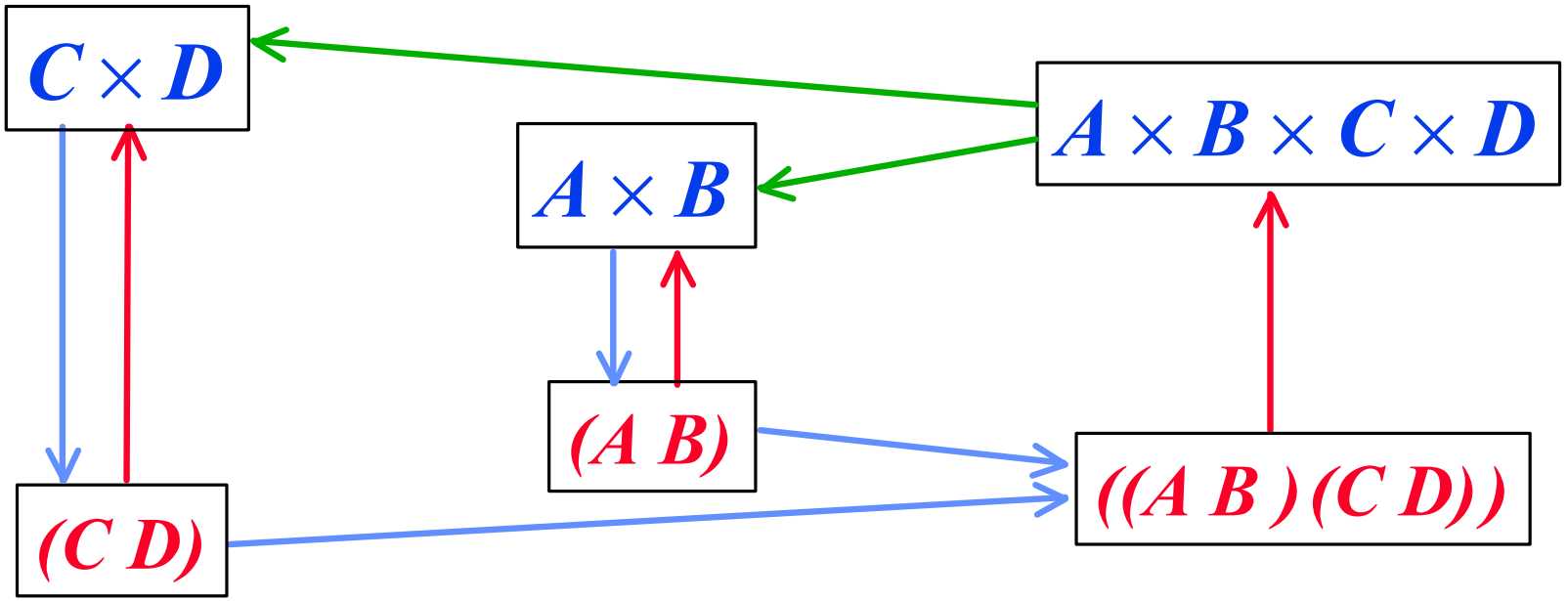


Definizione di parentesizzazione

Definizione: **Un prodotto di matrici $A_1 \cdot A_2 \cdot A_3 \cdot \ldots \cdot A_n$ si dice *completamente parentesizzato* se:**

- **consiste di una unica matrice ($n = 1$) oppure**
- **per qualche $1 \leq k \leq n-1$, è il prodotto, delimitato da pare-ntesi, tra i prodotti *completamente parentesizzati***

$$A_1 \cdot A_2 \cdot A_3 \cdot \ldots \cdot A_k \quad \text{e} \quad A_{k+1} \cdot A_2 \cdot A_3 \cdot \ldots \cdot A_n$$



Quanti modi ci sono di parentesizzare?

- $A_1, A_2, A_3, \ldots, A_n$
- **Sia $P(n)$ il *numero di modi* per calcolare il prodotto di n matrici.**
- **Supponiamo che l'ultima moltiplicazione sia**
  - $(A_1, A_2, \ldots, A_k) \times (A_{k+1}, \ldots, A_n)$ $\quad 1 \leq k \leq n-1$

- **per ogni scelta di k per $(A_1,A_2,\ldots,A_k)$ ci sono $P(n-k)$ possibili parentesizzazioni della seconda porzione $(A_{k+1},\ldots,A_n)$ e**
- **per ogni scelta di k di $(A_{k+1},\ldots,A_n)$ ci sono $P(k)$ possibili parentesizzazioni della prima porzione $(A_1,A_2,\ldots,A_k)$.**

Quanti modi ci sono di parentesizzare?

- $A_1, A_2, A_3, \ldots, A_n$
- **Sia $P(n)$ il *numero di modi* di calcolare il prodotto di n matrici.**
- **Supponiamo che l'ultima moltiplicazione sia**
  - $(A_1, A_2, \ldots, A_k) \times (A_{k+1}, \ldots, A_n)$ $\quad 1 \le k \le n-1$

- **Dunque, fissato un k, ci sono $P(k) \cdot P(n-k)$ modi.**
- $P(n) = \sum_{1 \le k \le n-1} P(k) \cdot P(n-k)$ $\qquad P(1) = 1$

n	1	2	3	4	5	6	7	8	9	10
$P(n)$	1	1	2	5	14	42	132	429	1430	4862

Quanti modi ci sono di parentesizzare?

Dunque, fissato un *k*, ci sono *P(k)*·*P(n-k)* modi.

$$P(n) = \begin{cases} 1 & \text{se } n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{altrimenti} \end{cases}$$

Questa è una equazione di ricorrenza

n	1	2	3	4	5	6	7	8	9	10
P(n)	1	1	2	5	14	42	132	429	1430	4862

Quanti modi ci sono di parentesizzare?

Dunque, fissato un *k*, ci sono *P(k)·P(n-k)* modi.

$$P(n) = \begin{cases} 1 & \text{se } n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{altrimenti} \end{cases}$$

Questa è una equazione di ricorrenza...

… la cui soluzione è la sequenza dei *numeri catalani*

$$P(n) = C(n-1) \rightarrow C(n) = \frac{1}{n+1}\binom{2n}{n} = \Omega\left(\frac{4^n}{n^{3/2}}\right)$$

Quanti modi ci sono di parentesizzare?

$$P(n) = \begin{cases} 1 & \text{se } n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{altrimenti} \end{cases}$$

Questa è una equazione di ricorrenza...

… la cui soluzione è la sequenza dei *numeri catalani*

$$P(n) = C(n-1) \rightarrow C(n) = \frac{1}{n+1}\binom{2n}{n} = \Omega\left(\frac{4^n}{n^{3/2}}\right)$$

Quindi: "enumerare tutte le possibilità, calcolare il numero di moltiplicazioni" e "scegliere la parentesizzazione a costo minore" *non è praticabile* (il numero di possibilità è *esponenziale*)!

Soluzione con programmazione dinamica

① ***Caratterizzare*** **la** ***struttura*** **di una** ***soluzione ottima***

② ***Definire*** ***ricorsivamente*** **il valore di una** ***soluzione ottima***

③ **Calcolare il** ***valore di una soluzione*** **ottima "*bottom-up*" (dal basso verso l'alto)**

④ **Cotruzione di una soluzione ottima.**

Vediamo ora *una ad una* le *4 fasi* del processo di sviluppo

Notazione

Denoteremo nel seguito con:

c_0 : **numero di righe della matrice** A_1

c_{i-1} : **numero di righe della prima matrice** A_i

c_i : **numero di colonne della matrice** A_i

$A_{1\ldots n}$: **sia una parentesizzazione che il risultato del prodotto** $A_1 \cdot A_2 \cdot \ldots \cdot A_n$

$A_{l\ldots r}$: **sia una parentesizzazione che il risultato del prodotto** $A_l \cdot \ldots \cdot A_r$

① *Caratterizzare della soluzione ottima*

- **Una soluzione al problema della parentesizzazione ottima di *n* matrici *divide il problema* nei due *sottoproblemi*:**
  - **il prodotto delle prime *k matrici* $A_{1\ldots k}$ e**
  - **il prodotto delle rimanenti *n-k* matrici $A_{k+1\ldots n}$ (per qualche *k*).**
- **La *soluzione finale* ($A_{1\ldots n}$) è il risultato del prodotto delle due matrici $A_{1\ldots k}$ e $A_{k+1\ldots n}$.**

① *Caratterizzare della soluzione ottima*

- **La *soluzione finale* ($A_{1...n}$) è il risultato del prodotto delle due matrici $A_{1...k}$ e $A_{k+1...n}$.**
- **Il *costo di una soluzione* (i.e. del prodotto $A_{1...n}$) è allora la *somma*:**
  - **"costo del prodotto $A_{1...k}$", più**
  - **"costo del prodotto $A_{k+1...n}$", più l**
  - **"costo del prodotto finale" tra le due matrici, cioè $c_0 c_k c_n$.**

$$c[A_{1...n}] = c[A_{1...k}] + c[A_{k+1...n}] + c_0 c_k c_n$$

① *Caratterizzare della soluzione ottima*

- **La *soluzione finale* ($A_{1...n}$) è il risultato del prodotto delle due matrici $A_{1...k}$ e $A_{k+1...n}$.**
- **Il *costo* del prodotto $A_{1...n}$ è la *somma* del costo del prodotto di $A_{1...k}$ più il costo di $A_{k+1...n}$, più il costo del prodotto finale tra le due matrici risultanti, cioè $c_0 c_k c_n$.**

Ma come devono essere fatte le soluzioni ai due sottoproblemi $A_{1...k}$ e $A_{k+1...n}$ per garantire che la *soluzione complessiva* ($A_{1...n} = A_{1...k} \times A_{k+1...n}$) sia anch'essa *ottima*?

① *Caratterizzare della soluzione ottima*

- **Quello che ci serve che valga è che la *struttura delle soluzioni ai sottoproblemi* sia *analoga* a quella del *problema complessivo*.**
- **Cioè che *soluzioni ottime* ai *sottoproblemi* permettano di costruire la *soluzione ottima* al *problema complessivo*.**

Teorema: **Se $A_{1\ldots n}=A_{1\ldots k}\,´\,A_{k+1\ldots n}$ è una *parentesizzazione ottima* del prodotto $A_1´A_2´\ldots´A_n$, allora $A_{1\ldots k}$ e $A_{k+1\ldots n}$ sono *parentesizzazioni ottime* dei prodotti $A_1´\ldots´A_k$ e $A_{k+1}´\ldots´A_n$, rispettivamente.**

① Caratterizzare della soluzione ottima

Teorema: Se $A_{1\ldots n} = A_{1\ldots k} \cdot A_{k+1\ldots n}$ è una *parentesizzazione ottima* del prodotto $A_1 \cdot A_2 \cdot \ldots \cdot A_n$, allora $A_{1\ldots k}$ e $A_{k+1\ldots n}$ sono *parentesizzazioni ottime* dei prodotti $A_1 \cdot \ldots \cdot A_k$ e $A_{k+1} \cdot \ldots \cdot A_n$, rispettivamente.

Dimostrazione: Supponiamo che $A_{1\ldots n} = A_{1\ldots k} \cdot A_{k+1\ldots n}$ sia una *parentesizzazione ottima* del problema $A_1 \cdot A_2 \cdot \ldots \cdot A_n$ ma che almeno uno tra $A_{1\ldots k}$ e $A_{k+1\ldots n}$ non sia una parentesizzazione ottima del *rispettivo sottoproblema*.

Il costo $c[A_{1\ldots n}] = c[A_{1\ldots k}] + c[A_{k+1\ldots n}] + c_0 c_k c_n$

Supponiamo che esista una parentesizzazione migliore $A'_{1\ldots k}$ delle stesse prime k matrici (cioè $c[A'_{1\ldots k}] < c[A_{1\ldots k}]$).

Allora basterebbe sostituire $A'_{1\ldots k}$ al posto $A_{1\ldots k}$ per ottenere anche una parentesizzazione migliore per $A_{1\ldots n}$.

① *Caratterizzare della soluzione ottima*

Teorema: **Se $A_{1\ldots n}=A_{1\ldots k}\,´\,A_{k+1\ldots n}$ è una *parentesizzazione ottima* del prodotto $A_1´A_2´\ldots´A_n$, allora $A_{1\ldots k}$ e $A_{k+1\ldots n}$ sono *parentesizzazioni ottime* dei prodotti $A_1´\ldots´A_k$ e $A_{k+1}´\ldots´A_n$, rispettivamente.**

Questo teorema fornisce la *caratterizzazione della struttura della soluzione ottima*.

Ci dice che *ogni soluzione ottima* al problema della parentesizzazione *contiene al suo interno* le *soluzioni ottime dei due sottoproblemi*.

L'esistenza di *sottostrutture ottime nella soluzione ottima* di un problema è una delle *caratteristiche* che vanno ricercate *per decidere* se la tecnica di *Programmazione Dinamica è applicabile*.

② *Definizione del valore di una soluzione ottima*

Il secondo passo consiste nel *definire ricorsivamente* il *valore della soluzione ottima* (alla parentesizzazione) *in termini* delle *soluzioni ottime* (alle parentesizzazioni) *dei sottoproblemi*.

Notazione

- **Sia** $C(l,r)$ **il** *numero ottimo di moltiplicazioni* **necessario per calcolare il prodotto**

  $A_{l \ldots r}$ **dove** $1 \leq l \leq r \leq n$

- **Definiamo** $C(1,n)$ **ricorsivamente come segue:**
  - *Caso Base:*

    $C(l,r) = 0$ **se** $l = r$

② *Definizione del valore di una soluzione ottima*

- **Definiamo $C(1,n)$ ricorsivamente cone segue:**
  - ***Caso Base:***

    $C(l,r) = 0$ **se** $l = r$
  - ***Caso Induttivo***

    Supponiamo che l'ultima moltiplicazione sia

    $A_{l\ldots k} \times A_{k+1\ldots l}$ **dove** $l \leq k \leq r-1$ **allora**

    $C(l,r) = C(l,k) + C(k+1,r) + c_{l-1}\, c_k\, c_r$

② *Definizione del valore di una soluzione ottima*

- ***Caso Base:***

  $C(l,r) = 0$ **se** $l = r$

- ***Caso Induttivo***

  $A_{l\ldots k} \cdot A_{k+1\ldots l}$ **dove** $l \leq k \leq r-1$ **allora**

  $C(l,r) = C(l,k) + C(k+1,r) + c_{l-1}\, c_k\, c_r$

- **Ma per risolvere il nostro problema ci interessa sapere per quale valore di k si ottiene il valore minimo per $C(l,r)$**

② *Definizione del valore di una soluzione ottima*

- ***Caso Base:***

  $C(l,r) = 0$ **se** $l = r$

- ***Caso Induttivo***

  $A_{l\ldots k} \cdot A_{k+1\ldots l}$ **dove** $l \leq k \leq r-1$

  $C(l,r) = C(l,k) + C(k+1,r) + c_{l-1}\, c_k\, c_r$

Ma non conosciamo il valore di *k*...

… quindi dobbiamo tentarli tutti!

$$C(l,r) = \min_{l \le k < r} \{ C(l,k) + C(k+1,r) + c_{l-1} c_k c_r \}$$

③ *Calcolo del valore di una soluzione ottima*

Il terzo passo consiste nel *calcolare* il *valore della soluzione ottima* (alla parentesizzazione) *in termini* delle *soluzioni ottime* (alle parentesizzazioni) *dei sottoproblemi*.

③ *Calcolo del valore di una soluzione ottima*

A partire dall'equazione sotto (risultato del passo 2), sarebbe facile definire un *algoritmo ricorsivo* che calcola il *costo minimo C(1,n)* di $A_{1\ldots n}$

$$C(l,r)=\begin{cases}0 & \text{se } l \geq r \\ \min\limits_{l\leq k<r}\{C(l,k)+C(k+1,r)+c_{l-1}c_k c_r\} & \text{se } l < r\end{cases}$$

Vedremo che tale approccio porta ad un *algoritmo di costo esponenziale*, non migliore della *enumerazione esaustiva*.

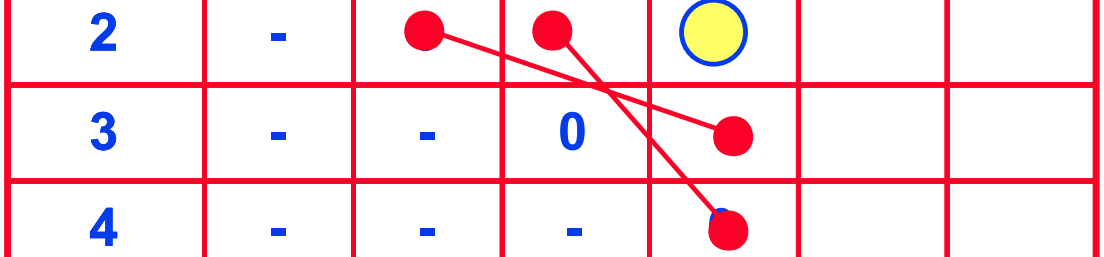
L \ R	1	2	3	4	5	6
1						
2	-					
3	-	-	0			
4	-	-	-	0		
5	-	-	-	-	0	
6	-	-	-	-	-	0

$C(l,r) = 0$ se $l = r$,

$C(l,r) = min_{l \le k < r}\{ C(l,k) + C(k+1,r) + c_{l-1}c_k c_r \}$ altrimenti

$$C(1,2) = min_{1 \le k < 2}\{ C(1,k) + C(k+1,2) + c_1 c_k c_2 \}$$
$$= C(1,1) + C(2,2) + c_0 c_1 c_2$$

L \ R	1	2	3	4	5	6
1	0					
2	-					
3	-	-	0			
4	-	-	-			
5	-	-	-	-	0	
6	-	-	-	-	-	0

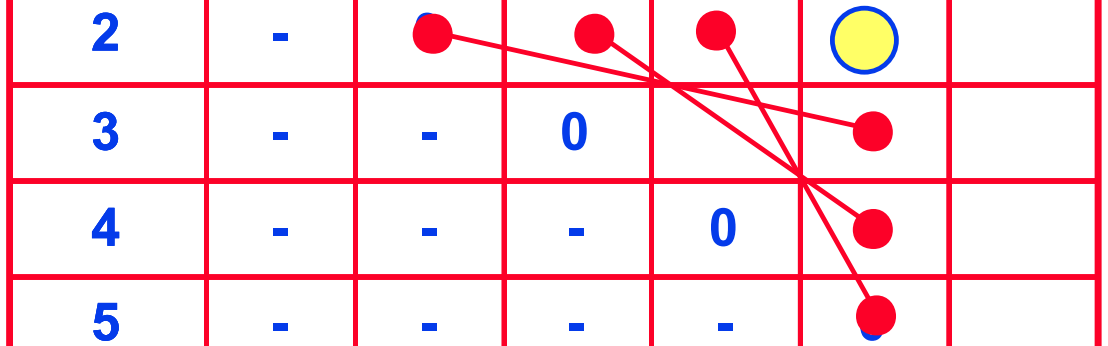


$C(l,r) = 0$ se $l = r$,

$C(l,r) = min_{l \le k < r}\{ C(l,k) + C(k+1,r) + c_{l-1}c_k c_r \}$ altrimenti

$$C(2,4) = min_{2 \le k < 4}\{ C(2,k) + C(k+1,4) + c_1 c_k c_4 \}$$
$$= min\{ C(2,2) + C(3,4) + c_1 c_2 c_4 ,$$
$$C(2,3) + C(4,4) + c_1 c_3 c_4 \}$$

L \ R	1	2	3	4	5	6
1	0					
2	-					
3	-	-	0			
4	-	-	-	0		
5	-	-	-	-		
6	-	-	-	-	-	0

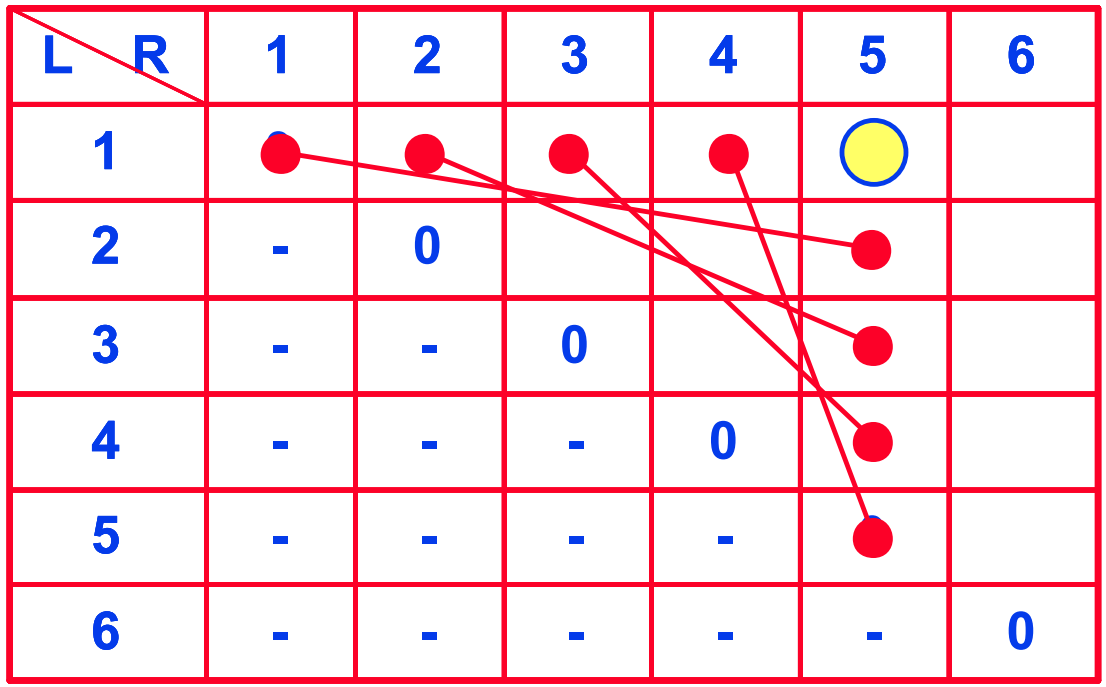


$C(l,r) = 0$ se $l = r$,

$C(l,r) = min_{l \le k < r}\{ C(l,k) + C(k+1,r) + c_{l-1}c_k c_r \}$ altrimenti

$$C(2,5) = min_{2 \le k < 5}\{ C(2,k) + C(k+1,5) + c_1 c_k c_5 \}$$
$$= min\{ C(2,2) + C(3,5) + c_1 c_2 c_5 ,$$
$$C(2,3) + C(4,5) + c_1 c_3 c_5 ,$$
$$C(2,4) + C(5,5) + c_1 c_4 c_5 \}$$

L \ R	1	2	3	4	5	6
1						
2	-	0				
3	-	-	0			
4	-	-	-	0		
5	-	-	-	-		
6	-	-	-	-	-	0



$C(l,r) = 0$ **se** $l = r$,

$C(l,r) = min_{l \le k < r}\{ C(l,k) + C(k+1,r) + c_{l-1}c_k c_r \}$ **altrimenti**

$$C(1,5) = min_{1 \le k < 5}\{ C(1,k) + C(k+1,5) + c_0 c_k c_5 \}$$
$$= min\{ C(1,1) + C(2,5) + c_0 c_1 c_5 ,$$
$$C(1,2) + C(3,5) + c_0 c_2 c_5 ,$$
$$C(1,3) + C(4,5) + c_0 c_3 c_5 ,$$
$$C(1,4) + C(5,5) + c_0 c_4 c_5 \}$$

L \ R	1	2	3	4	5	6
1	0					
2	-	0				
3	-	-	0			
4	-	-	-	0		
5	-	-	-	-	0	
6	-	-	-	-	-	0

$C(l,r) = 0$ **se** $l = r$,

$C(l,r) = min_{l \le k < r}\{ C(l,k) + C(k+1,r) + c_{l-1}c_k c_r \}$ **altrimenti**

$$C(1,6) = min_{1 \le k < 6}\{ C(1,k) + C(k+1,6) + c_0 c_k c_6 \}$$
$$= min\{ C(1,1) + C(2,6) + c_0 c_1 c_6 ,$$
$$C(1,2) + C(3,6) + c_0 c_2 c_6 ,$$
$$C(1,3) + C(4,6) + c_0 c_3 c_6 ,$$
$$C(1,4) + C(5,6) + c_0 c_4 c_6 ,$$
$$C(1,5) + C(6,6) + c_0 c_5 c_6 \}$$