

Programmazione dinamica: sottoproblemi

- ▷ Dobbiamo tagliare un asta in tante parti per poi venderle. Ogni parte dell'asta ha un prezzo di vendita che è funzione della sua lunghezza. Dobbiamo decidere come tagliare l'asta per massimizzare il ricavo.
- ▷ Input: lunghezza n dell'asta (intero positivo) e tabella dei prezzi p_1, p_2, \dots, p_n , dove p_i è il prezzo di un pezzo lungo i . I pezzi possono solo avere una lunghezza intera.
- ▷ Output: $s = \langle l_1, l_2, \dots, l_k \rangle$ con $1 \leq l_i \leq n$ e $\sum_{i=1}^k l_i = n$ che rende massimo $\sum_{i=1}^k p_{l_i}$
- ▷ Se i sottoproblemi condividono altri sottoproblemi, con la tecnica divide et impera risolveremmo molti sottoproblemi molte volte

Rod cutting: definizione

- ▷ Dobbiamo tagliare un asta in tante parti per poi venderle. Ogni parte dell'asta ha un prezzo di vendita che è funzione della sua lunghezza. Dobbiamo decidere come tagliare l'asta per massimizzare il ricavo.
- ▷ Input: lunghezza n dell'asta (intero positivo) e tabella dei prezzi p_1, p_2, \dots, p_n , dove p_i è il prezzo di un pezzo lungo i . I pezzi possono solo avere una lunghezza intera.
- ▷ Output: $s = \langle l_1, l_2, \dots, l_k \rangle$ con $1 \leq l_i \leq n$ e $\sum_{i=1}^k l_i = n$ che rende massimo $\sum_{i=1}^k p_{l_i}$

$$G(s) = \sum_{i=1}^k p_{l_i} = \text{QUANTO RENDE}$$

Programmazione dinamica: sottoproblemi

Rod cutting: esempio

lunghezze	1	2	3	4
prezzi	1	5	8	9
1 + 1 + 1 + 1				
5 + 1 + 1				
1 + 5 + 1				
1 + 1 + 5				
8 + 1				
5 + 5				
1 + 8				
9				

- ▷ Con la programmazione dinamica si risolve ogni sottoproblema una sola volta e si memorizza la sua soluzione in una struttura dati opportuna
- ▷ Prima di risolvere un sottoproblema si controlla nella struttura dati se è già memorizzata la sua soluzione
- ▷ Se la soluzione è presente nella tabella non si calcola nuovamente la soluzione, si usa quella memorizzata in tabella

Calcolare quanto guadagno al metro e prendere il max non ottimizza al meglio i partiti rimanenti

$$\max \left(g_i = \frac{p_i}{l_i} \right)$$

SOLUZIONE CAUSA: buone ed efficiente ma non ottima

Rod cutting: struttura delle soluzioni ottime

Struttura delle soluzioni ottime
Il problema del Rod cutting soddisfa la proprietà della sottostruttura ottima

Rod cutting: numero di soluzioni



Possa confermare che anche la soluzione ottima? **DIM PER ASSUNTO**

Il numero di possibili soluzioni per un asta lunga n è

$$|S(n)| = 2^{n-1}$$

Se non fosse ottima, ci sarebbero altri tagli che mi daranno di più (es: > 500), ma in quel caso tutto il problema non sarebbe ottimo, contro l'ipotesi.

Rod cutting: proprietà della sottostruttura ottima

- ▷ Sia $s = \langle l_1, l_2, \dots, l_k \rangle$ una soluzione ottima
 - ▷ $\sum_{i=1}^k l_i = n$
 - ▷ Sia h un indice compreso tra 1 e k
 - ▷ $s_1 = \langle l_1, l_2, \dots, l_h \rangle$ è una soluzione ottima per un asta lunga $L_1 = \sum_{i=1}^h l_i$
 - ▷ $s_2 = \langle l_{h+1}, l_{h+2}, \dots, l_k \rangle$ è una soluzione ottima per un asta lunga $L_2 = \sum_{i=h+1}^k l_i$
- Se non soddisfa, in genere sì ha tempo esponenziale*

Cut-Rod: algoritmo top down

```

Cut-Rod(p, n)
1   if n == 0
2       return 0
3   q = -∞
4   for i = 1 to n
5       q = MAX(q, p[i] + Cut-Rod(p, n - i))
6   return q

```

soddisfa proprietà sottostruttura ottima

Cut-Rod: proprietà della sottostruttura ottima

Rod cutting: proprietà della sottostruttura ottima

- ▷ Supponiamo per assurdo che s_1 non sia ottima per un asta lunga L_1
- ▷ Esisterebbe una soluzione s'_1 tale che $G(s'_1) > G(s_1)$
- ▷ Sia $s' = s'_1 \cup s_2$
- ▷ $G(s') = G(s'_1) + G(s_2) > G(s_1) + G(s_2) = G(s)$
- ▷ Ma s è ottima per ipotesi \implies assurdo

Cut-Rod: costo computazionale

$$\begin{aligned}
T(n) &= 1 + \sum_{i=1}^n T(n-i) \\
&= 1 + \sum_{i=0}^{n-1} T(i)
\end{aligned}$$

Cut-Rod: proprietà della sottostruttura ottima

$$T(n) = \Theta(2^n)$$

Extended-Cut-Rod: programmazione dinamica

EXTENDED-BOTTOM-UP-CUT-ROD(p, n)

```

1 let  $r[1..n]$  and  $s[1..n]$  be new arrays
2  $r[0] = 0$ 
3 for  $j = 1$  to  $n$ 
4    $q = -\infty$ 
5     for  $i = 1$  to  $j$ 
6       if  $q < p[i] + r[j-i]$ 
7          $q = p[i] + r[j-i]$ 
8          $s[j] = i$  Lunghezza primo taglio
9          $r[j] = q$  asta j
10    return  $r$  and  $s$ 
```

Esercizio

$$T(n) = 1 + \sum_{i=0}^{n-1} T(i)$$

$$T(0) = 1$$

Dimostrare che $T(n) = \Theta(2^n)$ **INDUZIONE**

$$2^0 = 1$$

$$T(n) = 1 + \sum_{i=0}^{n-1} 2^i = 1 + \Theta(2^n) = \Theta(2^n)$$

→ tutti < n ⇒ ipotesi induzione

Cut-Rod: programmazione dinamica

Print-Cut-Rod-Solution

lunghezza	1	2	3	4	5	6	7	8	9	10
prezzo	1	5	8	9	10	13	17	18	22	25
	$r[i]$	0	1	5	8	10	13	17	18	22

BOTTOM-UP-CUT-ROD(p, n)

```

1 let  $r[0..n]$  be a new array
2  $r[0] = 0$ 
3 for  $j = 1$  to  $n$  fissar una lunghezza
4    $q = -\infty$ 
5     for  $i = 1$  to  $j$  calcola soluzione ottima
6        $q = \text{MAX}(q, p[i] + r[j-i])$ 
7        $r[j] = q$  salva soluzione
8     return  $r[n]$ 
```

$\underbrace{q}_{\text{taglio } 1}$	$\underbrace{p_1 = 1}$	$\underbrace{p_2 = 5}$	$\underbrace{p_3 = 8}$	$\underbrace{p_4 = 10}$	$\underbrace{p_5 = 13}$	$\underbrace{p_6 = 17}$	$\underbrace{p_7 = 18}$	$\underbrace{p_8 = 22}$	$\underbrace{p_9 = 25}$	$\underbrace{p_{10} = 30}$
$r[i]$	0	1	5	8	10	13	17	18	22	25
$s[i]$	0	1	2	3	2	2	6	1	2	3

PRINT-CUT-ROD-SOLUTION(p, n)

```

1 ( $r, s$ ) = EXTENDED-BOTTOM-UP-CUT-ROD( $p, n$ )
2 while  $n > 0$ 
3   print  $s[n]$  tagli in sequenza
4    $n = n - s[n]$ 
```

Moltiplicazione di matrici non quadrate

Moltiplicazione di 3 matrici: esempio

MATRIX-MULTIPLY(A, B)

```
1  n = A.rows
2  m = A.columns // equal to B.rows
3  p = B.columns
4  let C be a new  $n \times p$  matrix
5  for i = 1 to n
6    for j = 1 to p
7       $c_{ij} = 0$ 
8      for k = 1 to m
9         $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$ 
10   return(C)
```

Moltiplicazione di 3 matrici: esempio

Moltiplicazione di 3 matrici: esempio

- ▷ Calcoliamo $B = A_1 \cdot A_2$, costo $10 \cdot 100 \cdot 5 = 5000$
- ▷ $B \in \mathbb{R}^{10,5}$
- ▷ Calcoliamo $C = B \cdot A_3$, costo $10 \cdot 5 \cdot 50 = 2500$
- ▷ Costo per calcolare $(A_1 \cdot A_2) \cdot A_3 = 7500$

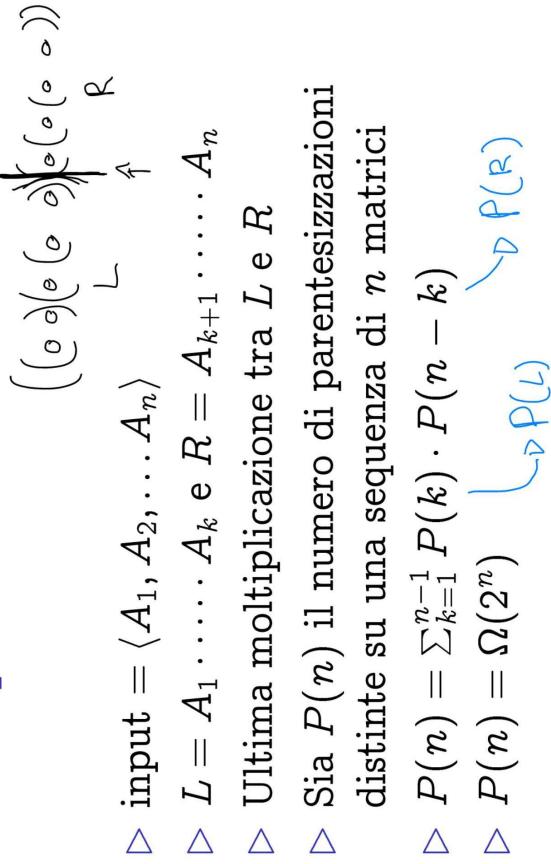
- ▷ Calcoliamo $B = A_2 \cdot A_3$, costo $100 \cdot 5 \cdot 50 = 25.000$
- ▷ $B \in \mathbb{R}^{100,50}$
- ▷ Calcoliamo $C = A_1 \cdot B$, costo $10 \cdot 100 \cdot 50 = 50.000$
- ▷ Costo per calcolare $A_1 \cdot (A_2 \cdot A_3) = 75.000$

Moltiplicazione di una sequenza di matrici: definizione

- ▷ Siano A_1, A_2, \dots, A_n matrici tali per cui:
numero di colonne di A_i = numero di righe
di A_{i+1} per $i = 1, \dots, n - 1$, quindi sono
moltiplicabili tra loro
- ▷ Siano $M = A_1 \cdot A_2, \dots, A_n$
- ▷ Trovare il modo (parentesizzazione) più
rapido per calcolare M
- ▷ Input: A_1, A_2, \dots, A_n
- ▷ Output: una parentesizzazione di costo
minimo per calcolare M

Moltiplicazione di una sequenza di matrici: numero di parentesizzazioni

- ▷ input = $\langle A_1, A_2, \dots, A_n \rangle$
- ▷ $L = A_1 \cdot \dots \cdot A_k$ e $R = A_{k+1} \cdot \dots \cdot A_n$
- ▷ Ultima moltiplicazione tra L e R
- ▷ Sia $P(n)$ il numero di parentesizzazioni
distinte su una sequenza di n matrici
- ▷ $P(n) = \sum_{k=1}^{n-1} P(k) \cdot P(n-k)$
- ▷ $P(n) = \Omega(2^n)$
- ▷ $P(n) = \Theta(n^2)$



Numero di possibili parentesizzazioni: esempio

Moltiplicazione di una sequenza di matrici: numero di sottoproblemi

*Non essendo comutativa i sottoproblemi possono essere
soltanto combinazione di matrici*

- ▷ input = $\langle A_1, A_2, \dots, A_n \rangle$
 - ▷ Sia $SubProb(n)$ il numero di sottoproblemi
distinti su una sequenza di n matrici
 - ▷ Sottoproblema = $\langle A_i, A_{i+1}, \dots, A_j \rangle$ con
 $1 \leq i \leq n$ e $i < j \leq n$
 - ▷ $SubProb(n) = \Theta(n^2)$
- Input = $\langle A_1, A_2, A_3, A_4 \rangle$
1. $(A_1(A_2(A_3A_4)))$
 2. $(A_1((A_2A_3)A_4))$
 3. $((((A_1A_2)A_3)A_4)$
 4. $((A_1(A_2A_3))A_4)$
 5. $((A_1A_2)(A_3A_4))$

Sequenza di moltiplicazioni di matrici: struttura delle soluzioni ottime

Sequenza di moltiplicazioni di matrici: proprietà della sottostruutura ottima

- ▷ Supponiamo per assurdo che $L(k)$ non sia ottima per $\langle A_1, \dots, A_k \rangle$
- ▷ Esisterebbe una parentesizzazione $L'(k)$ tale che $\text{costo}(L'(k)) < \text{costo}(L(k))$
- ▷ Sia $P'(n) = L'(k) \cdot R(n - k)$
- ▷ $\text{costo}(P'(n)) = \text{costo}(L'(n)) + \text{costo}(R(n - k)) + \text{costo}(\text{ultima moltiplicazione}) < \text{costo}(L(k)) + \text{costo}(R(n - k)) + \text{costo}(\text{ultima moltiplicazione}) = \text{costo}(P(n))$
- ▷ Ma $P(n)$ è ottima per ipotesi \implies assurdo

Struttura delle soluzioni ottime

Il problema della sequenza di moltiplicazioni di matrici soddisfa la proprietà della sottostruutura ottima

Sotto ragionamento del culto

Sequenza di moltiplicazioni di matrici: proprietà della sottostruutura ottima

Sequenza di moltiplicazioni di matrici: programmazione dinamica

- ▷ Sia $P(n)$ una parentesizzazione ottima per $\langle A_1, A_2, \dots, A_n \rangle$
- ▷ Sia $P(n) = L(k) \cdot R(n - k)$ ovvero $P(n)$ prevede una ultima moltiplicazione tra le prime k matrici (moltiplicate tra loro in qualche modo) e le altre $n - k$ matrici (moltiplicate tra loro in qualche modo).
- ▷ $L(k)$ è una parentesizzazione ottima per $\langle A_1, \dots, A_k \rangle$
- ▷ $R(n - k)$ è una parentesizzazione ottima per $\langle A_{k+1}, \dots, A_n \rangle$

A_1	A_2	A_3	A_4
A_1	A_2	A_3	A_4
A_1	A_2	A_3	A_4

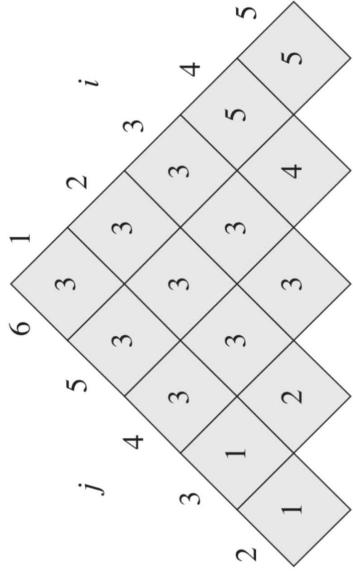
Sequenza di moltiplicazioni di matrici: esempio

matrice	dimensione
A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25

Sequenza di moltiplicazioni di matrici: esempio

s (equivalente "ultimo taglio")

posizione ultima moltiplicazione



Sequenza di moltiplicazioni di matrici: esempio

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

Sequenza di moltiplicazioni di matrici:

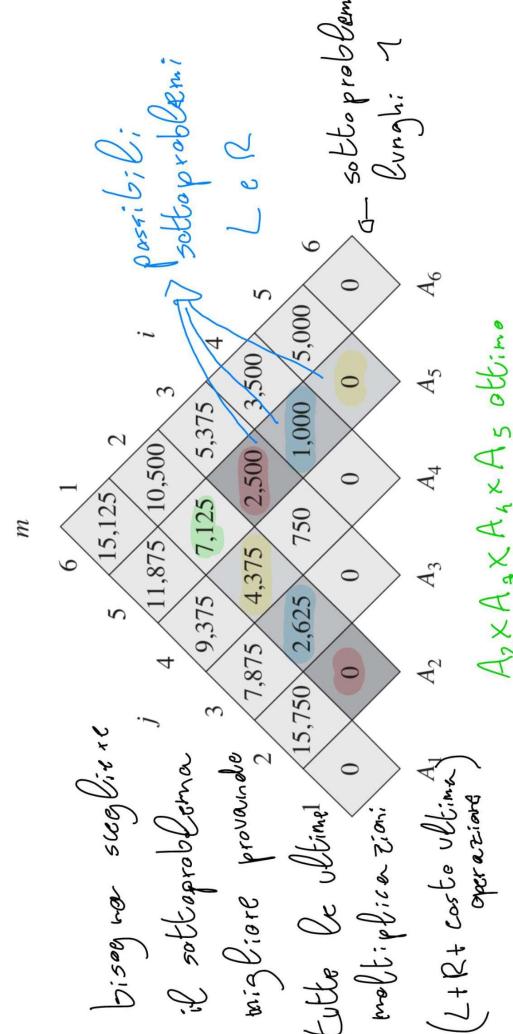
MATRIX-CHAIN-ORDER(p)

1 $n = p.length - 1$
2 let $m[1..n, 1..n]$ and $s[1..n - 1, 2..n]$ be new tables
3 for $i = 1$ to n :
4 $m[i, i] = 0$ *prime livello*
5 for $l = 2$ to n // l is the chain length

*piramidi
mossa
mat.*

6 for $i = 1$ to $n - l + 1$
7 $j = i + l - 1$
8 $m[i, j] = \infty$
9 for $k = i$ to $j - 1$
10 $q = m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j$
11 if $q < m[i, j]$
12 $m[i, j] = q$
13
14 return m and s

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25



$A_2 \times A_3 \times A_4 \times A_5$ ottime

($L+R+coste ultime$)
operazioni

Sequenza di moltiplicazioni di matrici:
stampa della soluzione

```
PRINT-OPTIMAL-PARENTS( $s, i, j$ )
1  if  $i == j$ 
2    print "A"i
3  else print "("
4    PRINT-OPTIMAL-PARENTS( $s, i, s[i, j]$ )
5    PRINT-OPTIMAL-PARENTS( $s, s[i, j] + 1, j$ )
6    print ")"
```

Matrix-Chain-Order: costo computazionale

tempo: $T(n) = \Theta(n^3)$

spazio: $S(n) = \Theta(n^2)$