

Esercitazione

Dynamic Programming

10/04/2025

*si ringrazia il Dott. Maestro
Alessandro Straziota per le slide

Massima sottosstringa palindroma

Massima sottostringa palindroma

- **Input:** una stringa S di n caratteri.
- **Goal:** eliminare da S dei caratteri (anche nessuno) in modo tale che la sottostringa risultante S^* sia **palindroma** e di **lunghezza massima**.
- **OPT:** massima lunghezza sottostringa palindroma.

Massima sottostringa palindroma

$S =$

A	L	G	O	R	I	T	M	O
---	---	---	---	---	---	---	---	---

$S^* =$

A	L	G	O	R	I	T	M	O
---	---	---	---	---	---	---	---	---

$OPT = 3$

Massima sottostringa palindroma

- **Sottoproblema:**

$OPT[i, j] =$ massima lunghezza sottostringa palindroma di $S[i \dots j]$

Massima sottostringa palindroma

- **Sottoproblema:**

$OPT[i, j] =$ massima lunghezza sottostringa palindroma di $S[i \dots j]$

- **Soluzione:**

$OPT[1 , n]$

Massima sottostringa palindroma

- **Sottoproblema:**

$OPT[i, j]$ = massima lunghezza sottostringa palindroma di $S[i \dots j]$

- **Soluzione:**

$OPT[1, n]$

- **Caso base:**

If $i > j \rightarrow OPT[i, j] = 0$

$OPT[i, i] = 1$

Massima sottostringa palindroma

- **Formula ricorsiva:**

$$\text{OPT}[i, j] = \begin{cases} 2 + \text{OPT}[i + 1, j - 1] & \text{se } S[i] = S[j] \\ \max\{\text{OPT}[i + 1, j], \text{OPT}[i, j - 1]\} & \text{altrimenti} \end{cases}$$

Massima sottostringa palindroma

- **Formula ricorsiva:**

Aggiungo $S[i]$ ed $S[j]$ a S^* , e ricorsivamente osservo $S[i+1 \dots j-1]$

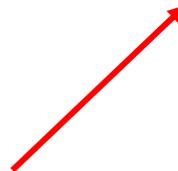
$$\text{OPT}[i, j] = \begin{cases} 2 + \text{OPT}[i + 1, j - 1] & \text{se } S[i] = S[j] \\ \max\{\text{OPT}[i + 1, j], \text{OPT}[i, j - 1]\} & \text{altrimenti} \end{cases}$$

Massima sottostringa palindroma

- **Formula ricorsiva:**

$$\text{OPT}[i, j] = \begin{cases} 2 + \text{OPT}[i + 1, j - 1] & \text{se } S[i] = S[j] \\ \max\{\text{OPT}[i + 1, j], \text{OPT}[i, j - 1]\} & \text{altrimenti} \end{cases}$$

Rimuovo $S[i]$



Massima sottostringa palindroma

- **Formula ricorsiva:**

$$\text{OPT}[i, j] = \begin{cases} 2 + \text{OPT}[i + 1, j - 1] & \text{se } S[i] = S[j] \\ \max\{ \text{OPT}[i + 1, j], \text{OPT}[i, j - 1] \} & \text{altrimenti} \end{cases}$$

Rimuovo $S[j]$



Massima sottostringa palindroma

	A	L	G	O	R	I	T	M	O
A	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0
O	0	0	0	0	0	0	0	0	0
R	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0
O	0	0	0	0	0	0	0	0	0

Inizializzazione
tabella

Massima sottostringa palindroma

	A	L	G	O	R	I	T	M	O
A	1	0	0	0	0	0	0	0	0
L	0	1	0	0	0	0	0	0	0
G	0	0	1	0	0	0	0	0	0
O	0	0	0	1	0	0	0	0	0
R	0	0	0	0	1	0	0	0	0
I	0	0	0	0	0	1	0	0	0
T	0	0	0	0	0	0	1	0	0
M	0	0	0	0	0	0	0	1	0
O	0	0	0	0	0	0	0	0	1

Casi base

Massima sottostringa palindroma

	A	L	G	O	R	I	T	M	O
A	1	1	0	0	0	0	0	0	0
L	0	1	1	0	0	0	0	0	0
G	0	0	1	1	0	0	0	0	0
O	0	0	0	1	1	0	0	0	0
R	0	0	0	0	1	1	0	0	0
I	0	0	0	0	0	1	1	0	0
T	0	0	0	0	0	0	1	1	0
M	0	0	0	0	0	0	0	1	1
O	0	0	0	0	0	0	0	0	1

Caso generale

Attenzione: da riempire
in diagonale! Perché?

Massima sottstringa palindroma

	A	L	G	O	R	I	T	M	O
A	1	1	1	1	1	1	1	1	3
L	0	1	1	1	1	1	1	1	3
G	0	0	1	1	1	1	1	1	3
O	0	0	0	1	1	1	1	1	3
R	0	0	0	0	1	1	1	1	1
I	0	0	0	0	0	1	1	1	1
T	0	0	0	0	0	0	1	1	1
M	0	0	0	0	0	0	0	1	1
O	0	0	0	0	0	0	0	0	1

Valore ottimo



Massima sottstringa palindroma

	A	L	G	O	R	I	T	M	O
A	1	1	1	1	1	1	1	1	3
L	0	1	1	1	1	1	1	1	3
G	0	0	1	1	1	1	1	1	3
O	0	0	0	1	1	1	1	1	3
R	0	0	0	0	1	1	1	1	1
I	0	0	0	0	0	1	1	1	1
T	0	0	0	0	0	0	1	1	1
M	0	0	0	0	0	0	0	1	1
O	0	0	0	0	0	0	0	0	1

Valore ottimo

Time $O(n^2)$

Massima sottostringa palindroma

```
def opt(s: str) -> int:
    n = len(s)
    M = [[0] * n for _ in range(n)]
    for i in range(n):
        M[i][i] = 1

    for d in range(1, n):
        for i in range(n - d):
            j = i + d
            if s[i] == s[j]:
                M[i][j] = 2 + M[i + 1][j - 1]
            else:
                M[i][j] = max(M[i + 1][j], M[i][j - 1])

    return M[0][n - 1]
```

Implementazione in python con approccio **tabulation** (bottom-up)

Massima sottostringa palindroma

Ma abbiamo veramente
bisogno di riempire TUTTA la
tabella ?



Studente di informatica

Massima sottosttringa palindroma

o	t	t	o		s	a	g	a	c	i		c	a	g	a	s	o	t	t	o
---	---	---	---	--	---	---	---	---	---	---	--	---	---	---	---	---	---	---	---	---

Massima sottostringa palindroma

o t t o s a g a c i c a g a s o t t o

	o	t	t	o	s	a	g	a	c	i	c	a	g	a	s	o	t	t	o	
o	1	1	2	4	4	4	4	4	4	4	5	5	5	7	9	11	13	15	17	19
t	0	1	2	2	2	2	2	3	3	3	5	5	5	7	9	11	13	15	17	17
t	0	0	1	1	1	1	1	3	3	3	5	5	5	7	9	11	13	15	15	15
o	0	0	0	1	1	1	1	3	3	3	5	5	5	7	9	11	13	13	13	13
s	0	0	0	0	1	1	1	3	3	3	3	3	5	7	9	11	11	11	11	11
a	0	0	0	0	0	1	1	3	3	3	3	3	5	7	9	9	9	9	9	9
g	0	0	0	0	0	0	1	1	1	1	1	3	5	7	7	7	7	7	7	7
a	0	0	0	0	0	0	0	1	1	1	1	3	5	5	5	5	5	5	5	5
c	0	0	0	0	0	0	0	0	1	1	1	3	3	3	3	3	3	3	3	4
i	0	0	0	0	0	0	0	0	0	1	1	1	1	1	3	3	3	3	3	4
c	0	0	0	0	0	0	0	0	0	0	1	1	1	3	3	3	3	3	3	4
a	0	0	0	0	0	0	0	0	0	0	0	1	1	3	3	3	3	3	3	4
g	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	2	4	4
a	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	2	4	4
s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	2	4	4
o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	2	4	4
t	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	2	2
t	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

Tante celle sono vuote o inutili. Riempiamo solo quelle necessarie!!



Studente di informatica

Massima sottostringa palindroma

```
def opt(s: str, i: int, j: int, cache: dict = dict()) -> int:
    if (i, j) in cache:
        return cache[i, j]

    if i > j:
        cache[i, j] = 0
    elif i == j:
        cache[i, j] = 1
    else:
        if s[i] == s[j]:
            cache[i, j] = 2 + opt(s, i + 1, j - 1, cache)
        else:
            cache[i, j] = max(opt(s, i + 1, j, cache), opt(s, i, j - 1, cache))
    return cache[i, j]
```

Implementazione in python con approccio **memoization** (top-down)

Massima sottostringa palindroma

o t t o s a g a c i c a g a s o t t o

	o	t	t	o		s	a	g	a	c	i		c	a	g	a	s	o	t	t	o	
o																						19
t																						17
t																						15
o																						13
s									5	5	5	7	9	11								
a				1	1	1	3	3	3	3	3	5	7	9	11							
g					1	1	3	3	3	3	3	5	7	9								
a						1	1	1	1	1	3	5										
c							1	1	1	3												
i								1	1													
									1													
c																						
a																						
g																						
a																						
s																						
o																						
t																						
t																						
o																						

Gotta Memoize 'Em All! *

(*) -2 punti all'esame per chi non capisce il riferimento

Massima sottosequenza palindroma

Come ricavare la soluzione dai valori dell'ottimo

```
def solution(s: str, OPT: dict, i: int, j: int) -> str:
    if i > j:
        return ""
    elif i == j:
        return s[i]
    elif s[i] == s[j]:
        return s[i] + solution(s, OPT, i + 1, j - 1) + s[j]
    elif OPT[i + 1, j] > OPT[i, j - 1]:
        return solution(s, OPT, i + 1, j)
    else:
        return solution(s, OPT, i, j - 1)
```

Il problema dei due scaffali

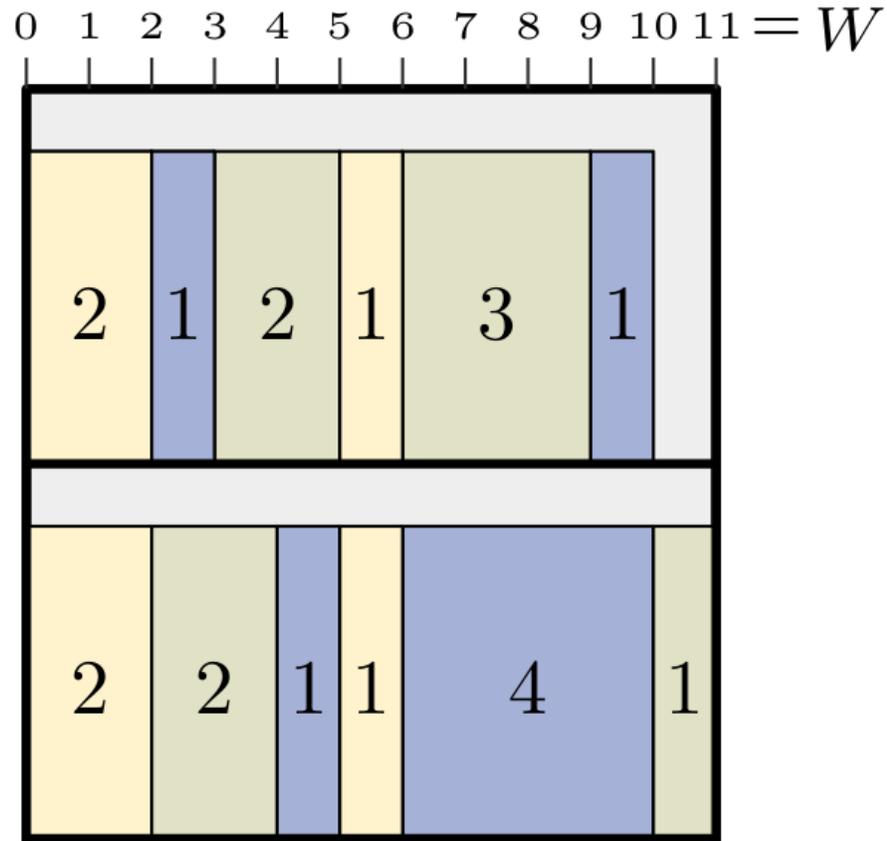
(*) Si ringrazia il prof. Stefano Leucci per l'esercizio

Il problema dei due scaffali

- **Input:** una collezione di n libri di spessore $t_1, \dots, t_n \in \mathbb{N}$.
- **Goal:** vuoi comprare due scaffali S_1 ed S_2 , **entrambi di larghezza** $W \in \mathbb{N}$, per disporre tutti i tuoi libri.
- **OPT:** larghezza **minima** necessaria per gli scaffali S_1 ed S_2 .

Il problema dei due scaffali

Inptu: { 2, 1, 2, 2, 1, 1, 2, 1, 4, 1, 3, 1 }



OPT: 11

Il problema dei due scaffali

- **Sottoproblema:**

$OPT[i, w]$ = minimo spazio usato sullo scaffale S_1 se sullo scaffale S_2 ho a disposizione spazio w e voglio piazzare i libri di spessore t_1, \dots, t_i .

- **Soluzione:**

min W tale che $OPT[n, W] \leq W$

$$W \leq T = \sum t_i$$


- **Caso base:**

$$OPT[0, w] = 0$$

Il problema dei due scaffali

- **Formula ricorsiva:**

$$\text{OPT}[i, w] = \begin{cases} t_i + \text{OPT}[i - 1, w] & \text{se } t_i > w \\ \min \{ t_i + \text{OPT}[i - 1, w], \text{OPT}[i - 1, w - t_i] \} & \text{altrimenti} \end{cases}$$

Il problema dei due scaffali

- **Formula ricorsiva:**

t_i non entra nello scaffale S_2 , quindi lo devo mettere per forza in S_1

$$\text{OPT}[i, w] = \begin{cases} t_i + \text{OPT}[i - 1, w] & \text{se } t_i > w \\ \min \{ t_i + \text{OPT}[i - 1, w], \text{OPT}[i - 1, w - t_i] \} & \text{altrimenti} \end{cases}$$

Il problema dei due scaffali

- **Formula ricorsiva:**

$$\text{OPT}[i, w] = \begin{cases} t_i + \text{OPT}[i - 1, w] & \text{se } t_i > w \\ \min \{ t_i + \text{OPT}[i - 1, w], \text{OPT}[i - 1, w - t_i] \} & \text{altrimenti} \end{cases}$$

Inserisco t_i nello scaffale S_2



Il problema dei due scaffali

- **Formula ricorsiva:**

$$\text{OPT}[i, w] = \begin{cases} t_i + \text{OPT}[i - 1, w] & \text{se } t_i > w \\ \min \{ t_i + \text{OPT}[i - 1, w], \text{OPT}[i - 1, w - t_i] \} & \text{altrimenti} \end{cases}$$

Inserisco t_i nello scaffale S_1

Il problema dei due scaffali

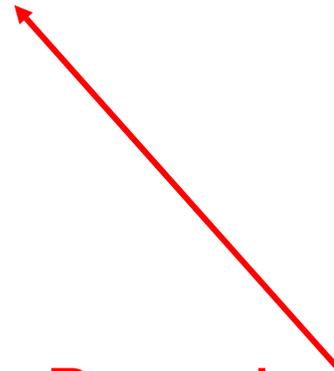
- **Space:** $O(n T)$ dove $T = \sum t_i$

Il problema dei due scaffali

- **Space:** $O(n T)$ dove $T = \sum t_i$
- **Time:** $O(n T)$, ogni volta devo combinare un numero costante di sottoproblemi.

Il problema dei due scaffali

- **Space:** $O(n T)$ dove $T = \sum t_i$
- **Time:** $O(n T)$, ogni volta devo combinare un numero costante di sottoproblemi.



Pseudo-polinomiale!

Il problema dei due scaffali

- **Esercizio 1:** implementare l'algoritmo con approccio bottom-up (tabulation) e top-down (memoization).

- **Esercizio 2:** adattare l'algoritmo nel caso in cui si hanno a disposizione 3 scaffali S_1 , S_2 ed S_3 .