

# Programmazione Dinamica

Analizziamo un primo problema di programmazione dinamica, in modo tale da capire come sono formati questi problemi e il funzionamento degli algoritmi.

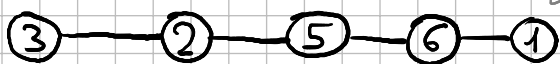
## - Massimo Sottinsieme indipendente (MSI)

### Problema:

Un gestore di un locale deve decidere a quali clienti far affittare un'intera sala del locale.

Quando la sala viene affittata il locale il giorno dopo deve restare vuoto per motivi di pulizie.

Il gestore vuole massimizzare il lucro.



il valore all'interno dei nodi è quanto il gestore viene pagato per l'affitto della sala in quel giorno.

### Soluzione

Dobbiamo determinare un sottoinsieme di nodi ( $S$ ) tale che se  $u, v \in S$ ,  $u, v \notin E$  che sia massimo, quindi  $\sum_{i \in S} \text{peso}(i)$  massimo  
significa che non possono essere vicini

Non puoi utilizzare una soluzione greedy, quindi prendendo sempre il massimo, perché non terrebbe conto del vincolo  $u, v \in S$ ,  $u, v \notin E$

Dobbiamo pensare ad una nuova soluzione.

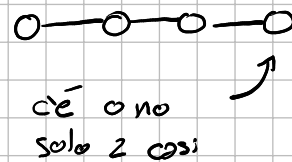
## Teorema Sottostuttura Ottima (TSO)

ovvero il problema con  $n$  nodi.

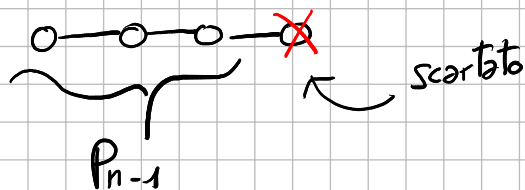
Proviamo a supporre di avere già una soluzione ottimale ( $S$ ) di  $P_n$ .

Abbiamo 2 possibili scenari

- 1) Sicuramente il nodo  $n-1 \notin S$
- 2) Sicuramente il nodo  $n-1 \in S$



- 1) L'ultimo nodo fa parte della soluzione ottimale, quindi possiamo sicuramente dire che è una soluzione ottimale anche per  $P_{n-1}$



Se non fosse così avremmo un assurdo, infatti esisterebbe una soluzione

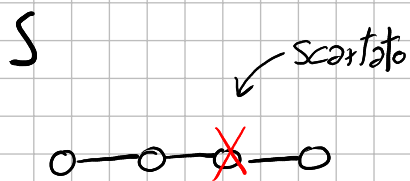
$S'$  per  $P_{n-1}$  con peso maggiore di  $S$ . Ma tale soluzione varrebbe

per  $P_n$  e quindi migliore anche di  $S$ , ma  $S$  è per definizione

ottimale. Quindi abbiamo dimostrato per assurdo che la soluzione  $S$

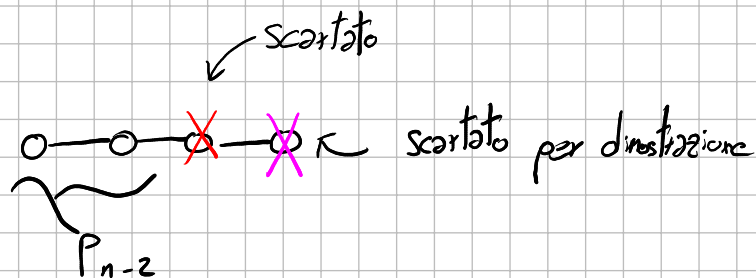
è ottimale anche per  $P_{n-1}$ .

- 2) Nel secondo caso l'ultimo nodo fa parte della soluzione ottimale, quindi possiamo dire che  $S' = S \setminus \{v_{n-1}\}$  è valido per  $P_{n-2}$ .



$S' = S \setminus \{v_{n-1}\}$ , quindi se scartiamo da  $S$

l'ultimo nodo



Come puoi vedere se scarti l'ultimo nodo ottieni la soluzione anche per  $P_{n-2}$ , ovvero  $S'$ , se non fosse ottimo esisterebbe una soluzione  $S''$  per  $P_{n-2}$  di peso maggiore di  $S'$ , ma allora  $S'' \cup \{v_{n-1}\}$  sarebbe migliore di  $S$ , cosa impossibile.

Come puoi notare la Programmazione Dinamica si basa sul fatto che la soluzione del Problema  $P_n$  si basa su quelle precedenti:  $P_{n-1}$  e  $P_{n-2}$

Versione Ricorsiva

$Val(S_n)\{$

se  $n=1$   $ris = peso(0)$

se  $n=2$   $ris = \max(peso(0), peso(1))$

else  $ris = \max(Val(S_{n-1}), Val(S_{n-2}) + peso(n-1))$

return  $ris;$

Costo  $O(2^n)$ , perché deve risolvere di nuovo problemi che ha già risolto

Versione Ricorsiva con Memoization (usiamo un Array per salvare i risultati appena calcolati)

prima del caso base

se  $(A[n] = \text{null}) \{ A[n] = \text{ris} \}$

else  $\text{ris} = A[n]$

$O(n)$ , ma costo elevato in spazio

Versione Iterativa

Funzionamento:

Val  $(S_n) \{$

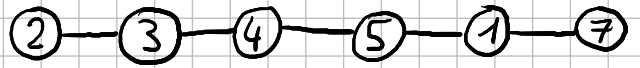
$A[1] = \text{peso}(0)$

$A[2] = \max(\text{peso}(0), \text{peso}(1))$

for  $i = 3$  to  $n$

$A[i] = \max(A[i-1], A[i-2] + \text{peso}(i-1))$

return  $A[n]$ ;



$A[1] = 2$

$A[2] = 3$

$A[3] = \max(A[2], A[1] + 4) = 6$

$A[4] = \max(6, 3 + 5) = 8$

$A[5] = \max(8, 6 + 1) = 8$

$A[6] = \max(8, 8 + 7) = 15 \checkmark$

Dimostrazione:

- Induzione

ogni volta che calcolo  $A[i]$  ho la soluzione al problema fino ad  $i$  nodi.

Base: 1 elemento  $A[1]$

2 elementi:  $A[2]$

Passo induttivo:  $k$

Ipotesi:  $\forall i < k \quad A[i] \text{ ok}$

$\rightarrow \text{ok} = \text{valore della sol. ottimale per } P_i$

Tesi:  $A[k]$  è ancora valore S ottimale per  $P_j$  ed  $\Rightarrow \text{ok}$ .

Questo è dovuto dal fatto che grazie al teorema di sottostruttura ottimale (TSO) sappiamo che arrivati a  $k$  avremo la Soluzione ottimale, perché scegliamo ogni

Volta una delle due soluzioni ottimali.

Tutti gli algoritmi Greedy sono anche algoritmi di Programmazione Dinamica, con l'unica differenza è che gli algoritmi greedy prendono sempre la migliore decisione che sono in grado di vedere in quel momento, e non ci tornano mai indietro.

Invece Divide et impera suddivide in problemi disgiunti e non si basano sulla sotto struttura ottima.

↳ in Prog dinamica i problemi si sovrappongono.

Come ottenere la soluzione effettiva, ovvero i nodi che sono stati scelti per ottenere quel valore?

Dobbiamo effettuare una ricostruzione (dopo aver eseguito l'algoritmo) della soluzione.

$i = n$

while  $i > 1$

se  $AC[i] > AC[i-1]$

$sol \leftarrow i-1$

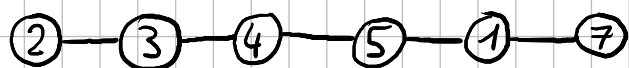
$i = i-2$  // sicuramente quello vicino non è stato scelto

else

$i--$

if  $i == 1$   $sol \leftarrow 0$

Funzionamento



Val = 15

$i: 5$

sol: 7

$i: 3$

sol: 7, 5

$i: 1$

sol: 7, 5, 3

$i: -1$

Fine iterazioni