Algoritmi e Strutture Dati

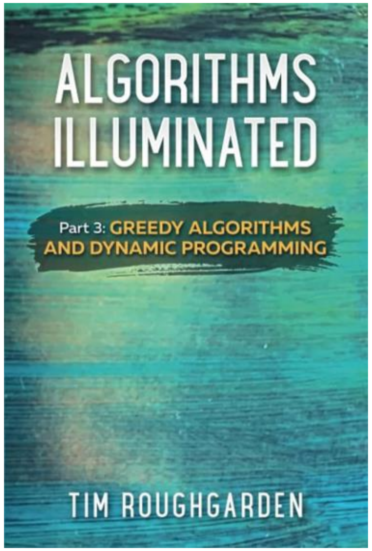
Luciano Gualà

guala@mat.uniroma2.it

www.mat.uniroma2.it/~guala

Programmazione dinamica

una tecnica di progettazione algoritmica molto potente



Capitolo 16

Sommario

- La tecnica della programmazione dinamica all'opera
- Un problema interessante: insieme indipendente di peso massimo (per un grafo a cammino)
  - perché le altre tecniche non funzionano
  - ragionare sulla struttura/proprietà della soluzione
- Un algoritmo di programmazione dinamica con complessità lineare
- Principi generali della programmazione dinamica
  - sottoproblemi, relazioni fra sottoproblemi, tabelle

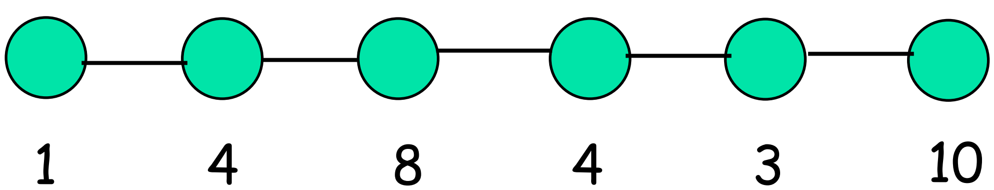
Insiime Indipendente di peso massimo (su grafi a cammino)

Input: Un cammino G di n nodi. Ogni nodo v_i ha un peso w_i.

Goal: trovare un insieme indipendente di peso massimo, ovvero un insieme S di nodi tale che:

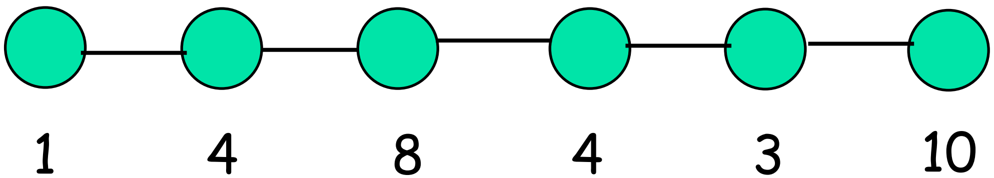
(i) S è un II,

(ii) $w(S) = \sum_{v_i \in S} w_i$ è più grande possibile.

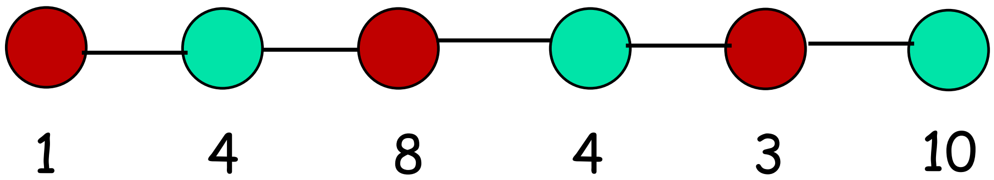


un insieme indipendente (II) di G è un sottoinsieme di nodi che non contiene due nodi adiacenti, ovvero per ogni coppia di nodi dell'insieme i due nodi non sono collegati da un arco.

esempio:



esempio:

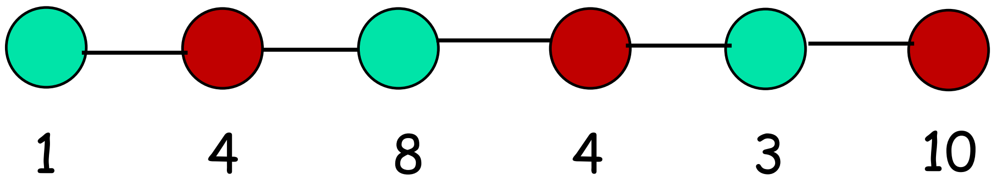


$S=\{v_1, v_3, v_5\}$

$w(S)=12$

un insieme indipendente

esempio:

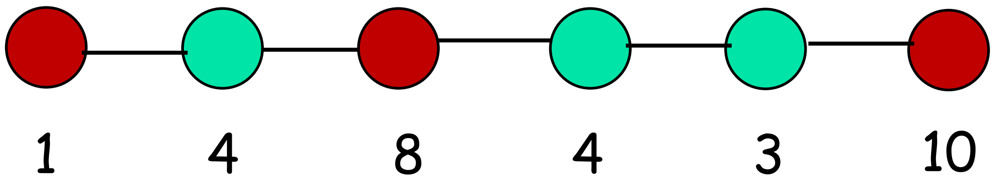


$S=\{v_2, v_4, v_6\}$

$w(S)=18$

un insieme indipendente migliore

esempio:



$S=\{v_1, v_3, v_6\}$

$w(S)=19$

un insieme indipendente ancora migliore (è un II di peso massimo!)

progettiamo un algoritmo:
che approccio utilizzare?

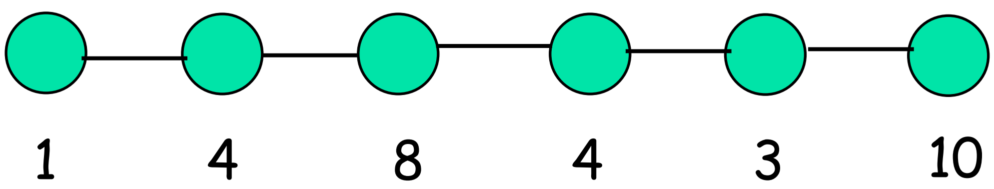
Forza bruta: enumerazione

idea: enumeriamo tutti i sottoinsiemi degli n nodi, per ognuno verifichiamo che è un insieme indipendente, ne calcoliamo il peso e teniamo quello di peso massimo.

domanda: quanti sottoinsiemi guardiamo?

risposta: tanti! (troppi)

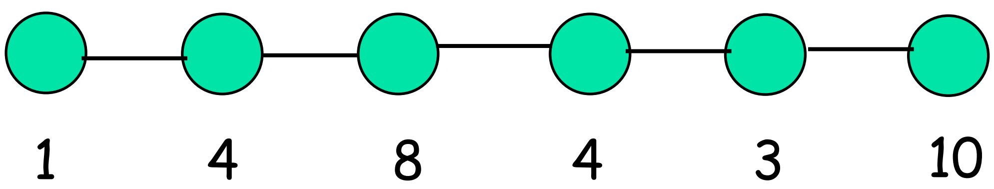
... sono 2^n !!!



approccio goloso (greedy)

idea: costruisco la soluzione in modo incrementale scegliendo ogni volta il nodo indipendente di valore massimo.

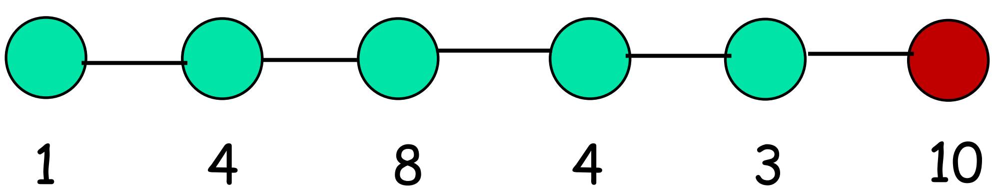
domanda: funziona?



approccio goloso (greedy)

idea: costruisco la soluzione in modo incrementale scegliendo ogni volta il nodo indipendente di valore massimo.

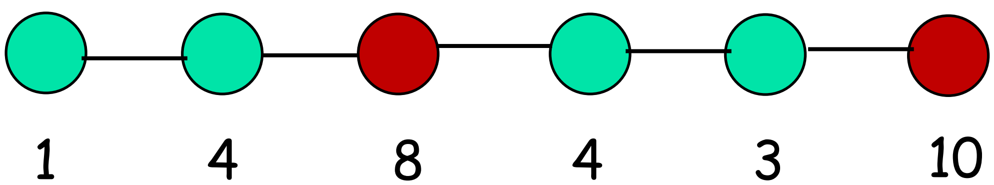
domanda: funziona?



approccio goloso (greedy)

idea: costruisco la soluzione in modo incrementale scegliendo ogni volta il nodo indipendente di valore massimo.

domanda: funziona?

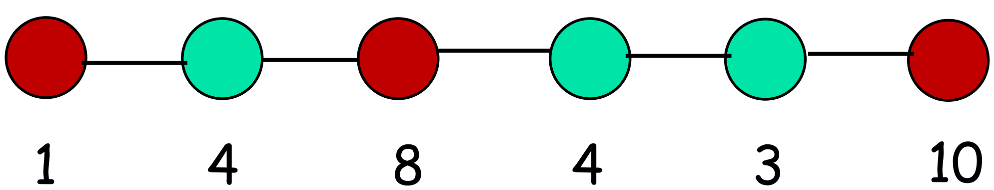


approccio goloso (greedy)

idea: costruisco la soluzione in modo incrementale scegliendo ogni volta il nodo indipendente di valore massimo.

domanda: funziona?

risposta: …su questa istanza l'algoritmo se l'è cavata bene!

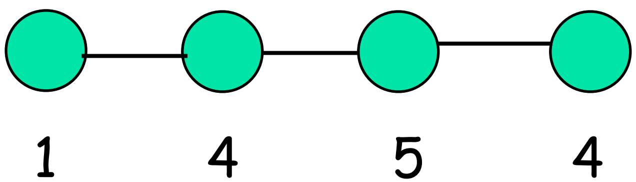


…sarà corretto davvero???

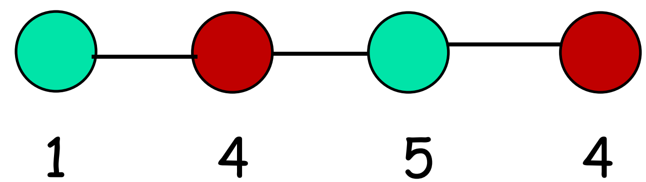
approccio goloso (greedy)

idea: costruisco la soluzione in modo incrementale scegliendo ogni volta il nodo indipendente di valore massimo.

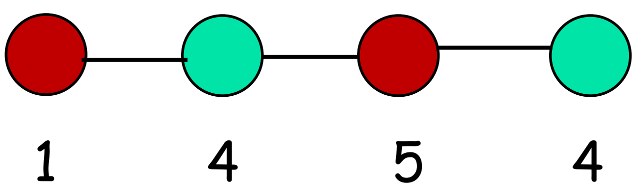
domanda: funziona? NO!!!!!



istanza



soluzione ottima

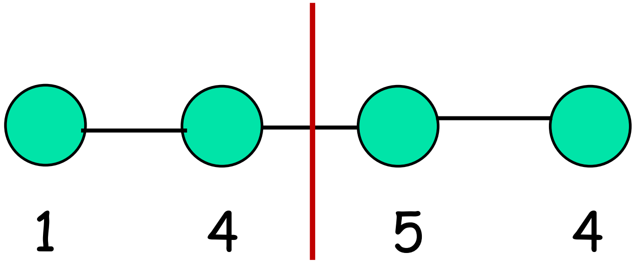


soluzione algoritmo greedy

divide et impera

idea: divido il cammino a metà, calcolo ricorsivamente l'II di peso massimo sulle due metà e poi ricombino le soluzioni.

domanda: è corretto?



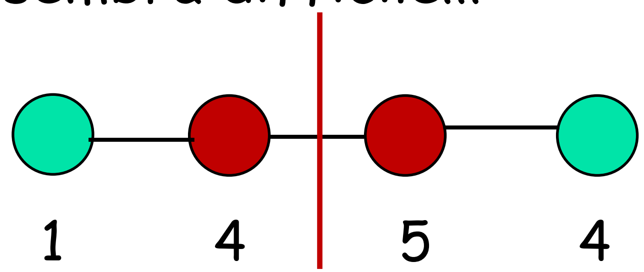
divide et impera

idea: divido il cammino a metà, calcolo ricorsivamente l'II di peso massimo sulle due metà e poi ricombino le soluzioni.

domanda: è corretto?

domanda: posso risolvere (efficientemente) i conflitti che ho quando ricombino?

... sembra difficile!!!



difficile ricombinare le soluzioni!!!!

Cosa non sta funzionando?

…non stiamo capendo davvero la struttura del problema.

…la comprensione della struttura del problema ci porterà a sviluppare un nuovo approccio.

cercando un nuovo approccio

passaggio critico: ragionare sulla struttura/proprietà della soluzione (ottima) del problema.

in termini di soluzioni (ottime) di sottoproblemi più "piccoli"

non davvero diverso da come si ragiona implicitamente quando si usa la tecnica del divide-et-impera

obiettivo: esprimere la soluzione del problema come combinazione di soluzioni di (opportuni) sottoproblemi. Se le combinazioni sono "poche" possiamo cercare la combinazione giusta per forza bruta.

ragionando sulla struttura della soluzione

sia S^* la soluzione ottima, ovvero l'II di peso massimo di G.
Considera l'ultimo nodo v_n di G.

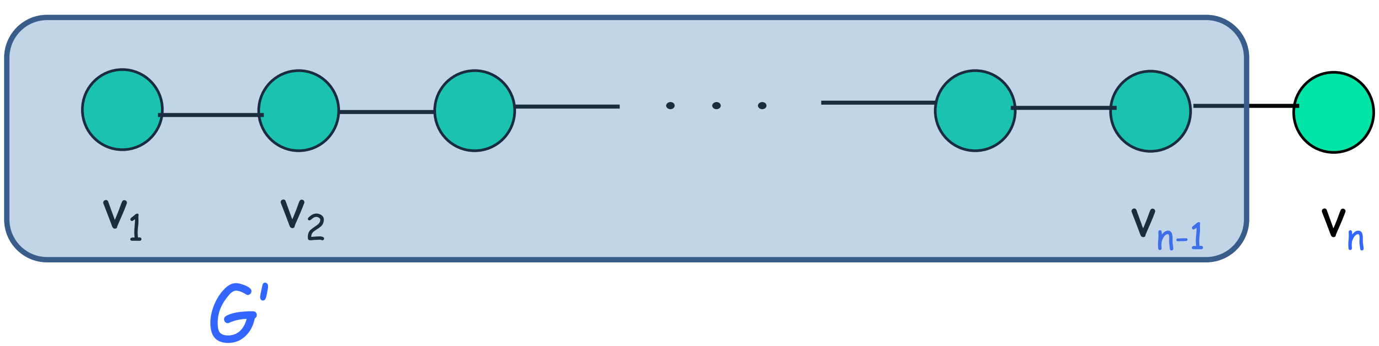
osservazione: $v_n \notin S^*$ o $v_n \in S^*$

caso 1: $v_n \notin S^*$

considera $G' = G - \{v_n\}$.

allora S^* è una soluzione ottima per G'.

se esistesse una soluzione S migliore per G', S sarebbe migliore anche per G: assurdo!



ragionando sulla struttura della soluzione

sia S^* la soluzione ottima, ovvero l'II di peso massimo di G.
Considera l'ultimo nodo v_n di G.

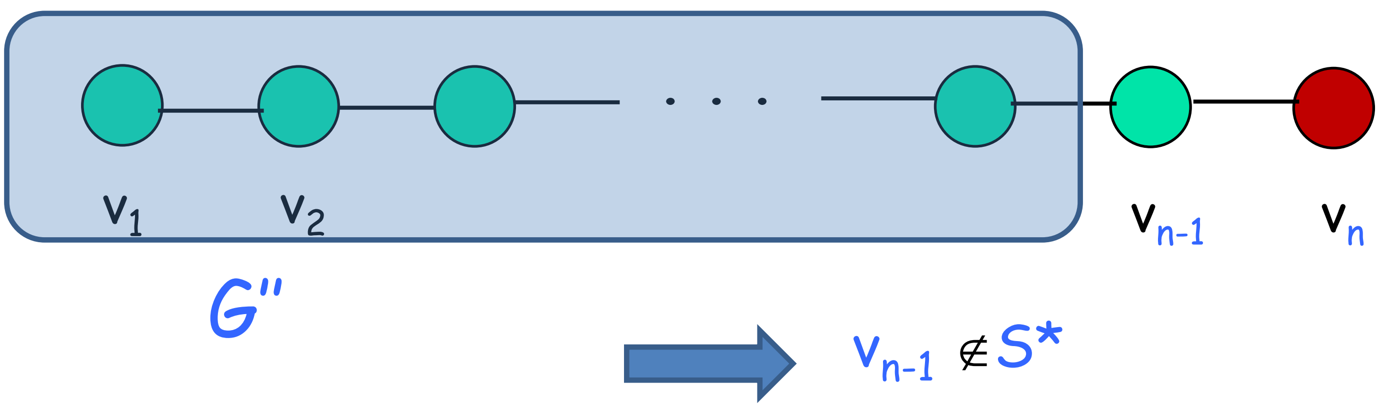
osservazione: $v_n \notin S^*$ o $v_n \in S^*$

caso 2: $v_n \in S^*$

considera $G''=G - \{v_{n-1}, v_n\}$.

allora $S^* \setminus \{v_n\}$ è una soluzione ottima per G''.

se esistesse una soluzione S migliore per G'', $S\cup\{v_n\}$ sarebbe migliore di S^* per G: assurdo!



verso un algoritmo

proprietà: l'II di peso massimo per G deve essere o:

(i) l'II di peso massimo per G',

(ii) v_n unito all'II di peso massimo per G''.

Idea (forse folle): calcolare tutte e due le soluzioni e ritornare la migliore delle due.

quale è il tempo dell'algoritmo se calcolo le due soluzioni *ricorsivamente*?

$$T(n)=T(n-1)+T(n-2)+O(1)$$

(è quella di Fibonacci2)

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

esponenziale!!!

...però forse non tutto è perduto

domanda fondamentale: quanti problemi distinti sono risolti dall'algoritmo ricorsivo?

$$\Theta(n)$$

c'è un sottoproblema per ogni prefisso di G

Idea: procediamo iterativamente considerando prefissi di G dai più piccoli verso i più grandi.

esempio

G_j: sottocammino composto dai primi j vertici di G

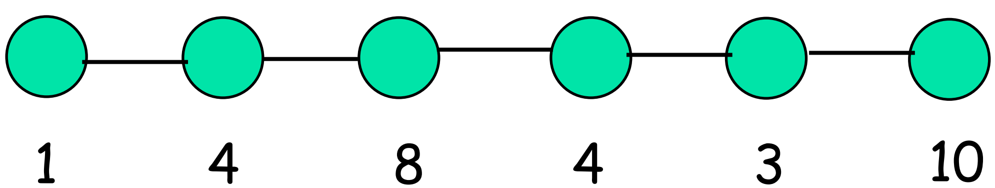
Sottoproblema j: calcolare il peso del miglior II per G_j

OPT[j]: valore soluzione sottoproblema j, ovvero peso dell'II di peso massimo di G_j

$OPT[1]=w_1$; $OPT[2]= \max \{w_1, w_2\}$

$OPT[j]= \max \{OPT[j-1], w_j+OPT[j-2]\}$

OPT: 1 4 9 9 12 19



l'algoritmo

G_j: sottocammino composto dai primi j vertici di G
OPT[]: vettore di n elementi;
dentro OPT[j] voglio mettere il peso dell'II di peso massimo di G_j

1. OPT[1]=w_1; OPT[2]= max $\{w_1, w_2\}$
2. **for** j=3 **to** n **do**
3. OPT[j]= max {OPT[j-1], w_j+OPT[j-2]}
4. **return** OPT[n]

$\boxed{T(n)=\Theta(n)}$

Oss: l'algoritmo calcola il valore della soluzione ottima, ma non la soluzione.

possiamo trovare in tempo lineare
anche l'II di peso massimo?

Ricostruire la soluzione (in tempo lineare)

ricostruire la soluzione

Idea semplice: mentre calcoliamo i valori OPT[j] possiamo mantenere esplicitamente anche la soluzione.

corretta ma non ideale: spreco di tempo e spazio

un'idea migliore: ricostruire la soluzione solo alla fine sfruttando il vettore OPT[].

proprietà chiave:

$v_j \in$ II di peso massimo di G_j $\Longleftrightarrow$ $w_j + OPT[j-2] \geq OPT[j-1]$

un algoritmo per ricostruire la soluzione

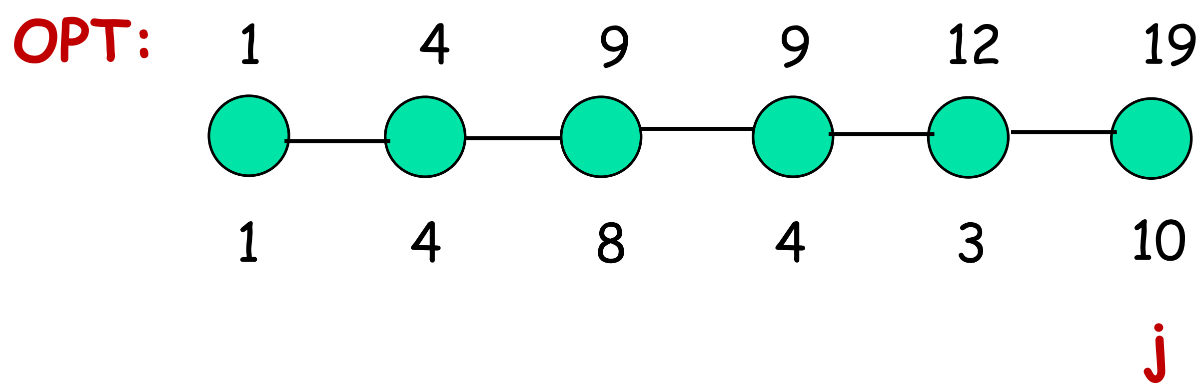
1. $S^*=\emptyset$; $j=n$;
2. **while** $j \geq 3$ **do**
3. **if** $OPT[j-1] \geq w_j+OPT[j-2]$
   then $j=j-1$;
   else $S^*=S^* \cup \{ v_j \}$; $j=j-2$;
4. **if** $j=2$ e $w_2>w_1$ **then** $S^*=S^* \cup \{ v_2 \}$
   else $S^*=S^* \cup \{ v_1 \}$;
5. **return** S^*

complessità temporale?

$$\boxed{T(n)=\Theta(n)}$$

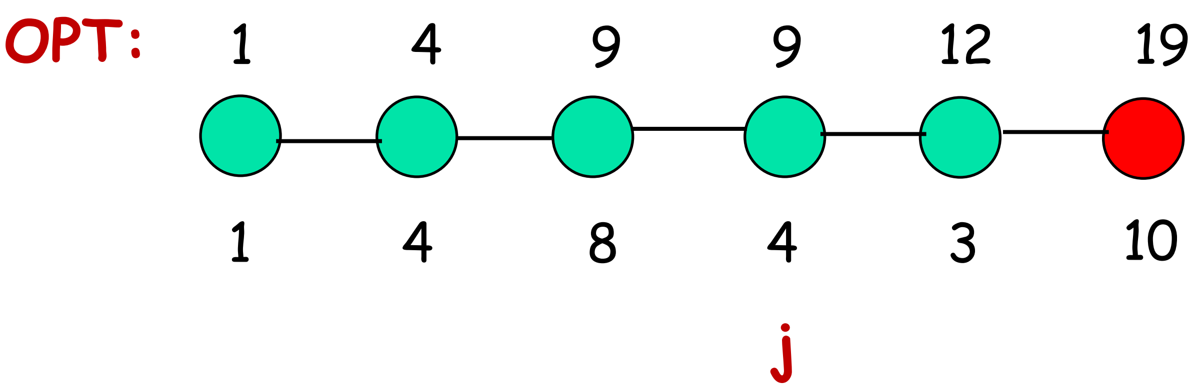
un algoritmo per ricostruire la soluzione

```
S*=∅; j=n;
while j≥3 do
    if OPT[j-1] ≥ w_j+OPT[j-2]
    then j=j-1;
    else S*=S*∪{ v_j }; j=j-2;
if j=2 e w_2>w_1  then S*=S*∪{ v_2 }
                  else S*=S*∪{ v_1 };
return S*
```



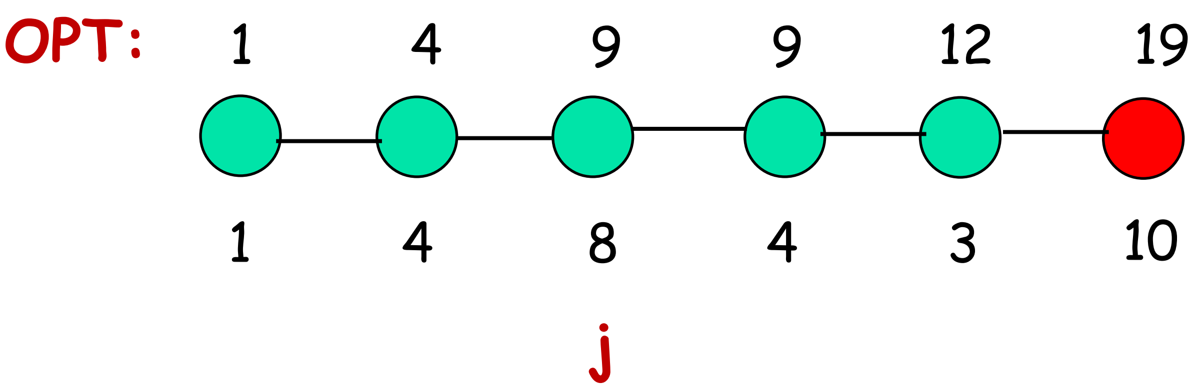
un algoritmo per ricostruire la soluzione

1. $S^*=\emptyset$; $j=n$;
2. **while** $j \geq 3$ **do**
3. **if** $OPT[j-1] \geq w_j + OPT[j-2]$
   then $j=j-1$;
   else $S^*=S^* \cup \{ v_j \}$; $j=j-2$;
4. **if** $j=2$ e $w_2>w_1$ **then** $S^*=S^* \cup \{ v_2 \}$
   else $S^*=S^* \cup \{ v_1 \}$;
5. **return** S^*



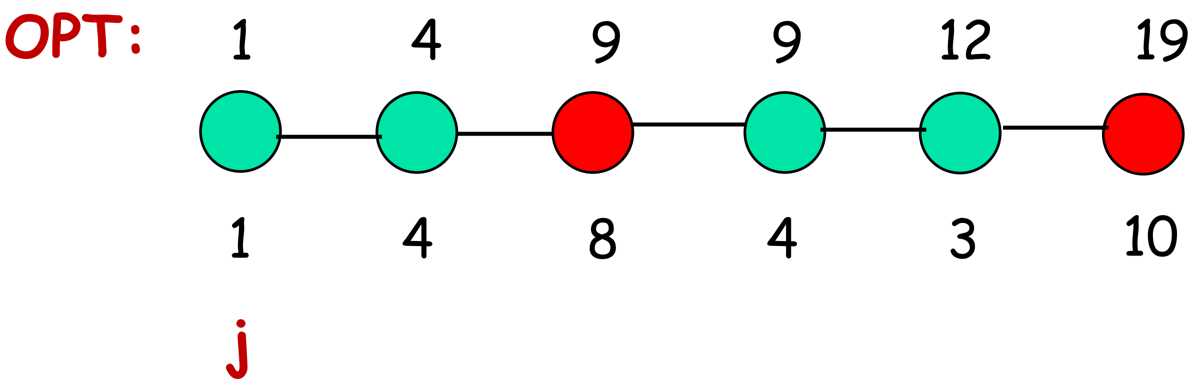
un algoritmo per ricostruire la soluzione

1. $S^* = \emptyset$; $j = n$;
2. **while** $j \geq 3$ **do**
3. **if** $OPT[j-1] \geq w_j + OPT[j-2]$
   then $j = j-1$;
   else $S^* = S^* \cup \{ v_j \}$; $j = j-2$;
4. **if** $j = 2$ e $w_2 > w_1$ **then** $S^* = S^* \cup \{ v_2 \}$
   else $S^* = S^* \cup \{ v_1 \}$;
5. **return** S^*



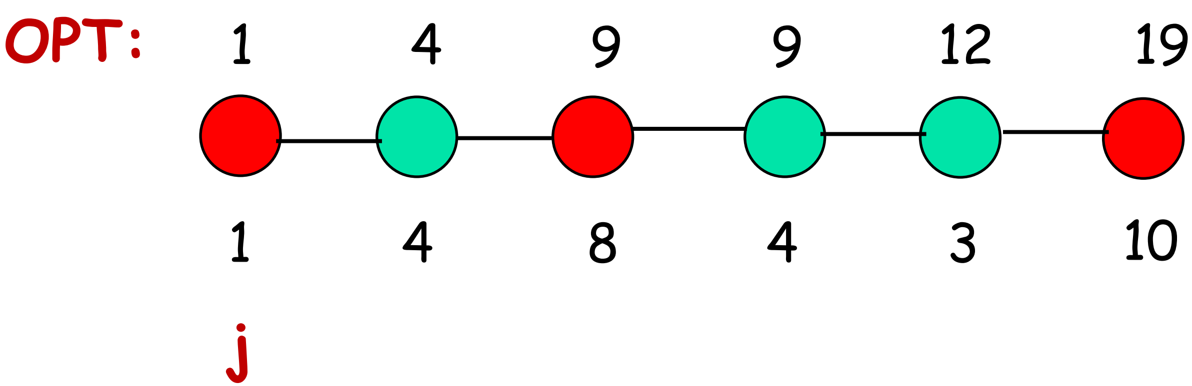
un algoritmo per ricostruire la soluzione

```
1.  S*=∅; j=n;
2.  while j≥3 do
3.      if OPT[j-1] ≥ w_j+OPT[j-2]
        then j=j-1;
        else S*=S*∪{ v_j }; j=j-2;
4.  if j=2 e w_2>w_1  then S*=S*∪{ v_2 }
                      else S*=S*∪{ v_1 };
5.  return S*
```



un algoritmo per ricostruire la soluzione

1. $S^* = \emptyset$; $j = n$;
2. **while** $j \geq 3$ **do**
3.   **if** $OPT[j-1] \geq w_j + OPT[j-2]$
    then $j = j-1$;
    else $S^* = S^* \cup \{ v_j \}$; $j = j-2$;
4. **if** $j = 2$ e $w_2 > w_1$ **then** $S^* = S^* \cup \{ v_2 \}$
    else $S^* = S^* \cup \{ v_1 \}$;
5. **return** S^*



Programmazione Dinamica: principi generali

1) identificare un numero piccolo di sottoproblemi

es: calcolare l'II di peso massimo di G_j, $j=1,\dots,n$

2) descrivere la soluzione di un generico sottoproblema in funzione delle soluzioni di sottoproblemi più "piccoli"

es: $OPT[j]=\max\{OPT[j-1], w_j+OPT[j-2]\}$

3) le soluzioni dei sottoproblemi sono memorizzate in una tabella

4) avanzare opportunamente sulla tabella, calcolando la soluzione del sottoproblema corrente in funzione delle soluzioni di sottoproblemi già risolti.

Proprietà che devono avere i sottoproblemi

1) essere pochi

2) risolti tutti i sottoproblemi si può calcolare velocemente la soluzione al problema originale

spesso la soluzione cercata è semplicemente quella del sottoproblema più grande

3) ci devono essere sottoproblemi "piccoli"

casi base

4) ci deve essere un ordine in cui risolvere i sottoproblemi

e quindi un modo di avanzare nella tabella e riempirla

ancora sul ruolo dei sottoproblemi

(breve discussione con avvertimenti)

…maledetti, favolosi sottoproblemi!

La chiave di tutto è la definizione dei "giusti" sottoproblemi

La definizione dei "giusti" sottoproblemi è un punto di arrivo

Solo una volta definiti i sottoproblemi si può verificare che l'algoritmo è corretto

Se la definizione dei sottoproblemi è un punto di arrivo, come ci arrivo?

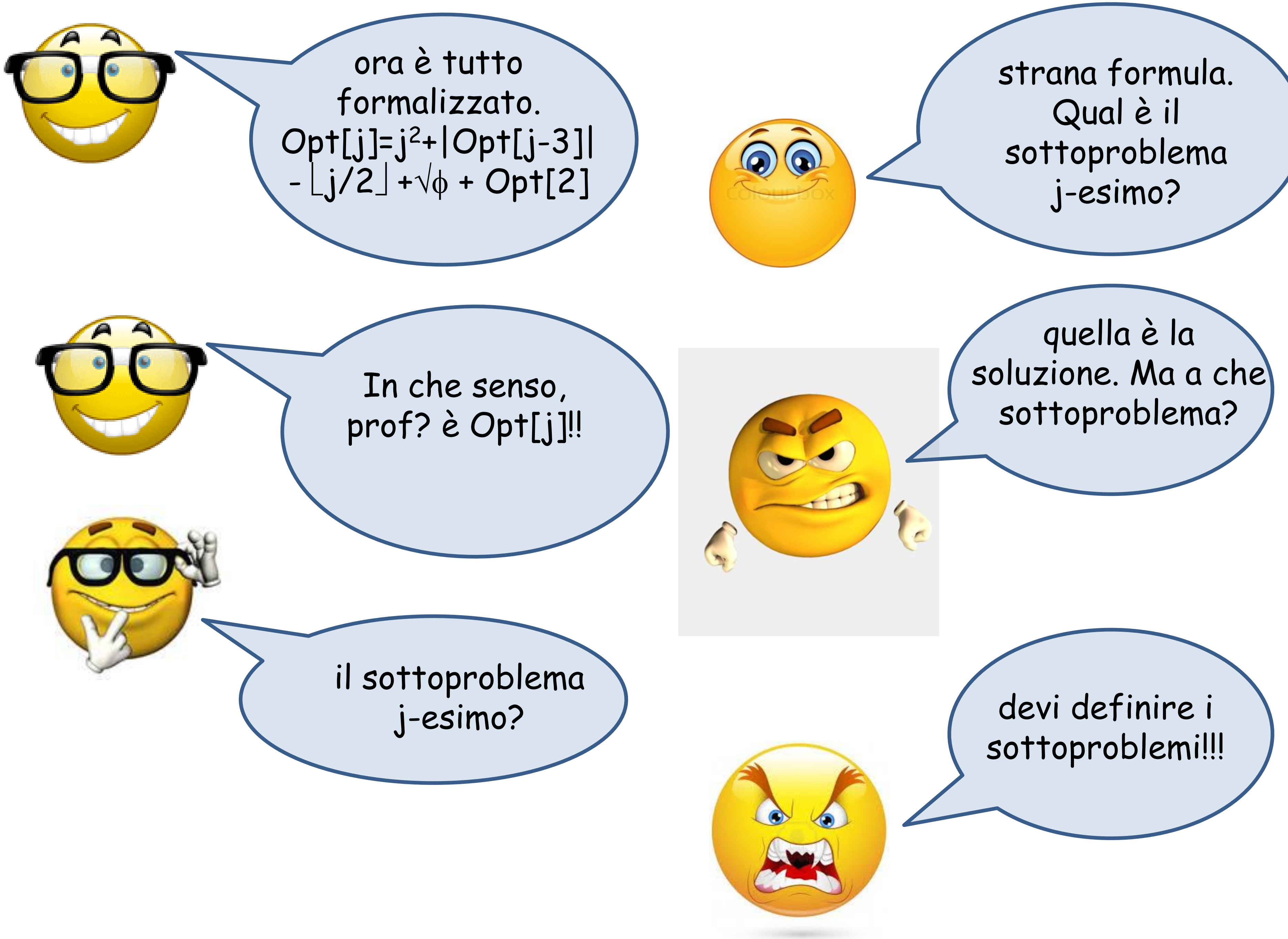
… ragionando sulla struttura della soluzione (ottima) cercata.

La struttura della soluzione può suggerire i sottoproblemi e l'ordine in cui considerarli

…e qualche avvertimento.

(brevi dialoghi ricorrenti)


salve, professore, volevo farle vedere questo algoritmo di programmazione dinamica, per capire se è corretto.
Bene. Come hai definito i sottoproblemi?
Sottoproblemi? Che sottoproblemi?
devi definire i sottoproblemi!!!



ora è tutto formalizzato. $Opt[j]=j^2+|Opt[j-3]| - \lfloor j/2 \rfloor + \sqrt{\phi} + Opt[2]$
strana formula. Qual è il sottoproblema j-esimo?
In che senso, prof? è Opt[j]!!
quella è la soluzione. Ma a che sottoproblema?
il sottoproblema j-esimo?
devi definire i sottoproblemi!!!


…e questo sipariettosi chiude con un…

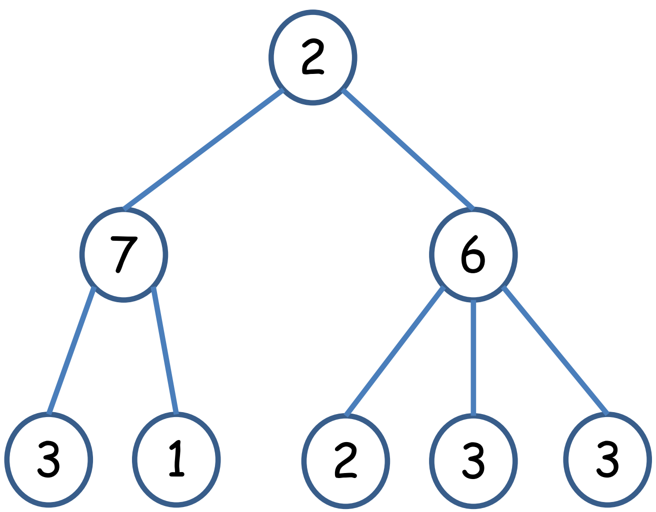
Esercizio: II di peso massimo su alberi (il problema della festa aziendale)

problema: invita i dipendenti alla festa aziendale

massimizza: il divertimento totale degli invitati

vincolo: tutti devono divertirsi

⇒ non invitare un dipendente e il suo boss diretto!



input: un albero con pesi sui nodi

goal: un II di peso totale massimo

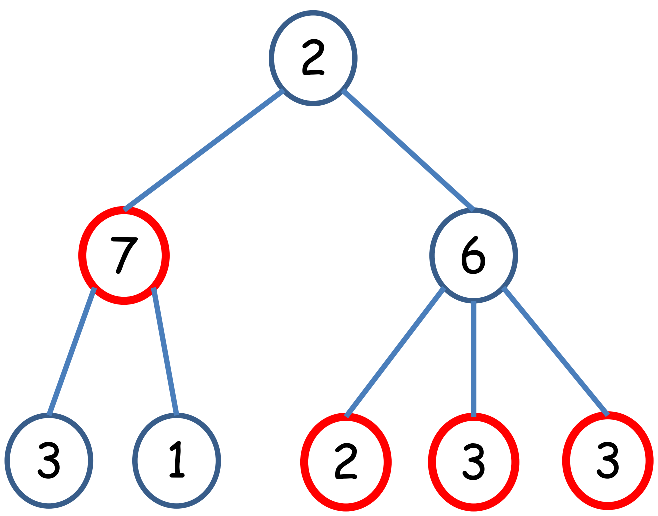
Esercizio: II di peso massimo su alberi (il problema della festa aziendale)

problema: invita i dipendenti alla festa aziendale

massimizza: il divertimento totale degli invitati

vincolo: tutti devono divertirsi

$\Rightarrow$ non invitare un dipendente e il suo boss diretto!



input: un albero con pesi sui nodi

goal: un II di peso totale massimo

OPT= 15