

LEZIONE 9

9.1 PROPRIETÀ DI SOTTOSTRUTTURA OTTIMA E DI SCELTA GREEDY NELLE DUE VERSIONI DEL PROBLEMA DELLO ZAINO (KNAPSACK)

Estrarrebbe la programmazione dinamica e la strategia greedy sfruttano le proprietà di sottostruzione ottime. Per ciò, potremmo essere tentati dello sviluppo di una soluzione basata sulle programmazioni dinamiche quando sarebbe sufficiente un approccio greedy; viceversa, potremmo erroneamente pensare che una soluzione greedy funzionerà quando invece è necessario usare la programmazione dinamica.

KNAPSACK (0-1 KNAPSACK o KNAPSACK INTERO)

- input:
- n oggetti $\{1, \dots, n\}$, tali che l'oggetto i abbia un valore v_i e un peso w_i (in realtà, abbiamo un input due array di lunghezza n (v_1, \dots, v_n) e (w_1, \dots, w_n));
 - un valore numerico W (peso massimo che lo zaino può contenere).

goal: scegliere un sottoinsieme di oggetti di velore complessivo massimo e con peso al più W , ovvero trovare

$$x_1, \dots, x_n \in \{0,1\}; \quad \max \left\{ \underbrace{\sum_{i=1}^n x_i v_i}_{\text{velore complessivo}} \mid \underbrace{\sum_{i=1}^n x_i w_i \leq W}_{\text{pero}} \text{ e } x_1, \dots, x_n \in \underline{\{0,1\}} \right\}$$

FRACTIONAL KNAPSACK

input:

- n oggetti $\{1, \dots, n\}$, tali che l'oggetto i abbia un velore v_i e un peso w_i (in realtà, abbiamo su input due array di lunghezza n (v_1, \dots, v_n) e (w_1, \dots, w_n));
- un velore numerico W (peso massimo che lo zaino può sostenere).

goal: scegliere un sottoinsieme di fratraci di oggetti di velore complessivo massimo e con peso al più W , ovvero trovare $x_1, \dots, x_n \in [0,1] \cap \mathbb{Q}$ tale che

$$\max \left\{ \sum_{i=1}^n x_i v_i \mid \sum_{i=1}^n x_i w_i \leq W \text{ e } x_1, \dots, x_n \in [0,1] \cap \mathbb{Q} \right\}$$

PROP: 0-1 KNAPSACK ha la proprietà di sottostituzion offuscate.

Prof.: Sia $(x_1^*, \dots, x_n^*) \in \{0,1\}^n$ una soluzione offuscata al problema, allora

$$\sum_{i=1}^n x_i^* w_i \leq W \quad e \quad \sum_{i=1}^n x_i^* v_i \text{ è massimo. Poco dopo siamo}$$

al sottoproblema in cui abbiamo trovare una

soluzione offuscata per $\{x_1, \dots, x_{j-1}, x_{j+1}, \dots, x_n\}$. Allora,

$$\sum_{\substack{i=1 \\ i \neq j}}^n x_i^* w_i \leq W - x_j^* w_j. \quad \text{Se } (x_1^*, \dots, x_{j-1}^*, x_{j+1}^*, \dots, x_n^*)$$

non fosse offuscata allora vorremmo che ci esiste un'altra

soluzione $(x_1', \dots, x_{j-1}', x_{j+1}', \dots, x_n')$ tale che

$$\sum_{\substack{i=1 \\ i \neq j}}^n x_i' w_i \leq W - x_j^* w_j \quad \text{e} \quad \sum_{\substack{i=1 \\ i \neq j}}^n x_i' v_i > \sum_{\substack{i=1 \\ i \neq j}}^n x_i^* v_i$$

Nei allora consideriamo le soluzioni al problema
frazionale $(x_1', \dots, x_{j-1}', x_j^*, x_{j+1}', \dots, x_n)$, abbiamo che

$$\sum_{\substack{i=1 \\ i \neq j}}^n x_i' w_i + x_j^* w_j \leq W$$

$$\sum_{\substack{i=1 \\ i \neq j}}^n x_i' v_i + x_j^* v_j > \sum_{\substack{i=1 \\ i \neq j}}^n x_i^* v_i + x_j^* v_j = \sum_{i=1}^n x_i^* v_i.$$

contradizione \square

Prop: FRACTIONAL KNAKPSACK ha la proprietà dell'ottimalità

Proof: Dimostrazione riguarda alle precedenti. \square

Possiamo applicare le proposte fatte di cui sopra per risolvere entro certi limiti i problemi.

Ma ci basterebbe applicare la strategia greedy?

Strategie greedy per il fractional knapsack:

- Si calcolano i valori per unità di peso (al kilo) per ogni oggetto, $r_i = \frac{v_i}{w_i} \quad \forall i \in \{1, \dots, n\}$.
- Si prende, a ogni scelta la frazione più grande possibile del r_i più grande possibile;

Supponiamo $r_1 \geq r_2 \geq \dots \geq r_n$

$$\left. \begin{array}{l} S_1 : W = q_1 + W_2, \quad q_1 \leq w_1 \\ S_2 : W_2 = q_2 + W_3 \\ \vdots \quad \vdots \\ S_m : W_m = q_m + W_{m+1} \end{array} \right\}, \quad \begin{array}{l} q_i = \text{frazione di peso} \\ \text{dell'oggetto } i \\ \text{prezzo} = (\text{quantità} \\ \text{di kilo che } i \\ \text{che prenderemo}) \end{array}$$

un ↓ ferme
 quando
 ○ esaurisco
 la disponibilità di i - $q_i = w_i$
 oppure quando
 raggiungo W

$$x_i = \frac{q_i}{w_i}, \quad \forall i \in \{1, \dots, n\}$$

poiché $0 \leq q_i \leq w_i \Rightarrow 0 \leq x_i \leq 1$.

Questo algoritmo (ordineando pure i k_i) risponde
 $\Theta(n \log(n))$ - time.

$$\sum_{i=1}^n x_i v_i = \sum_{i=1}^n \frac{q_i}{w_i} \cdot v_i = \sum_{i=1}^n q_i k_i$$

Prop: FRACTIONAL KNAPSACK GODE DELLA PROPRIETÀ DI SCELTA GREEDY.

Proof: Si a sua un sotto problema con liste di

pero W_m . Supponiamo $\{k_m \geq \dots \geq k_n\}$.

Consideriamo una soluzione ottima

$$\{q_m, \dots, q_n\}.$$

$$W_m \leq W_n$$

$$\left\{ \frac{x_m}{w_m}, \dots, \frac{x_n}{w_n} \right\}$$

Se k_m fa parte della soluzione ottima (i.e. $q_m \neq 0$)

Supponiamo che $q_m = 0$, ovvero che k_m non faccia parte delle soluzioni ottime. Allora

$$\sum_{i=m+1}^n q_i \leq W_m \quad e \quad \sum_{i=m+1}^n q_i k_i \text{ è massimo.}$$

$$\sum_{i=m+1}^n x_i v_i$$

$c = m + 1$

Allora posso sostituire k_{m+1} con k_m , ovvero

$$\left\{ \begin{array}{c} q_{m+1}^*, \\ \parallel \\ q_{m+1} \end{array}, \begin{array}{c} q_{m+2}^*, \\ \parallel \\ 0 \end{array}, \begin{array}{c} q_{m+2}^*, \\ \parallel \\ q_{m+2} \end{array}, \dots, \begin{array}{c} q_n^* \\ \parallel \\ q_n \end{array} \right\} \text{ allora}$$

$$\sum_{i=m}^n q_i^* = \sum_{i=m+1}^n q_i = \sum_{i=m}^n q_i \leq W_m \quad k_m > k_{m+1}$$

$$\sum_{i=m}^n q_i^* k_i = \underbrace{q_m^* k_m}_{\text{0}} + \underbrace{0 \cdot k_{m+1}}_{\text{0}} + \underbrace{\sum_{i=m+2}^n q_i k_i}_{\text{> m}} >$$

$$> q_{m+1} k_{m+1} + \sum_{i=m+2}^n q_i k_i = \sum_{i=m+1}^n q_i k_i$$

contraddizione.

Quindi ne conseguentemente le scelte greedy
fa parte delle α ottime.

□

CONTRO ESEMPIO: (0,1)- KNAKSACK non ha la proprietà di scelta greedy.

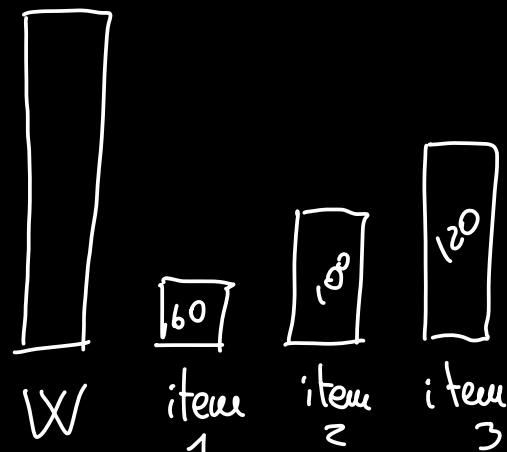
Consideriamo la seguente istanza di knapsack:

item 1 ($60\text{€}, 10\text{kg}$)

item 2 ($100\text{€}, 20\text{kg}$)

item 3 ($120\text{€}, 30\text{kg}$)

$W = 50 \text{ kg}$



$$k_1 = \frac{60}{10} = 6 \text{ €/kg} , \quad k_2 = \frac{100}{20} = 5 \text{ €/kg} , \quad k_3 = \frac{120}{30} = 4 \text{ €/kg}$$

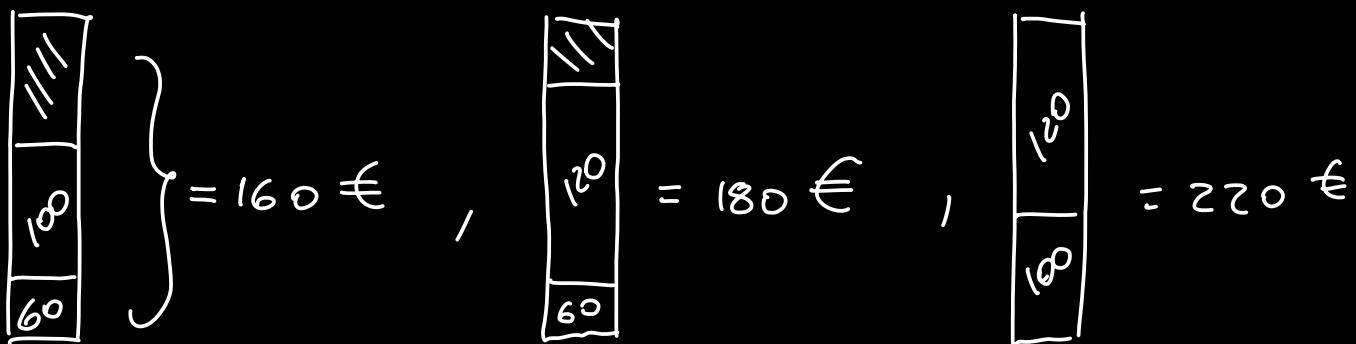
$$W = 50 = 40 + \underbrace{40}_{W_2} , \quad 40 = 20 + \underbrace{20}_{W_3} , \quad 20 = \frac{2}{3} \cdot 30$$

$$\{q_1, q_2, q_3\} = \{10, 20, 20\}$$

$$\{x_1, x_2, x_3\} = \left\{1, 1, \frac{2}{3}\right\}$$

$$\begin{aligned} \text{valore} &= 60 + 100 + \frac{2}{3} \cdot 120 = 240 \text{€} \\ &\downarrow \\ &\text{fractional} \end{aligned}$$

ma cosa succede nel caso futero 0-1 KNP SACK



Le scelte greedy per 0-1 KNP SACK è $x_1=1, x_2=1, x_3=0$
di valore 160€
che non è ottima.

Oss: Nel problema 0-1 KNP SACK, quando si considera
se scegliere o meno un oggetto bisogna confrontare
le soluzioni al problema che lo include con
quelle che non lo escludono, prima di poter
fare la scelta. Si vede come vengono generate

tanti sottoproblemi sovrapposti (overlapping subproblems),
 tra fra di loro diverso dei problemi risolvibili con le reg.
 altre cose.

9.2 PROPRIETÀ DELLA RICERCA

DEPTH-FIRST

DFS(G)

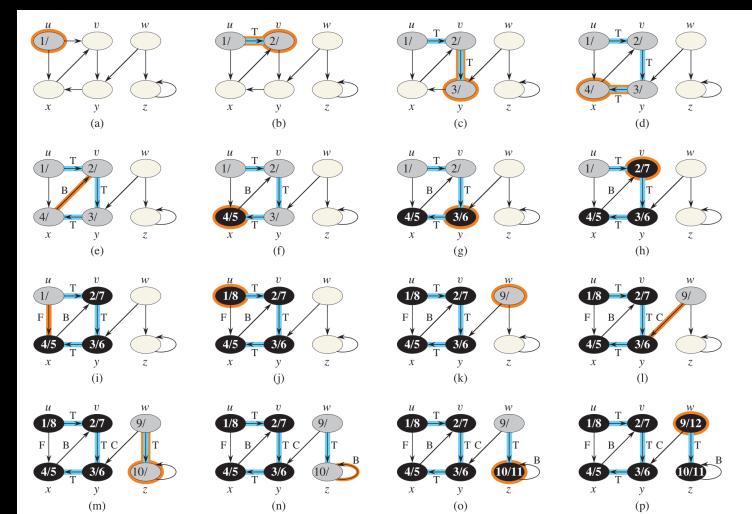
```

1 for each vertex  $u \in G.V$ 
2    $u.\text{color} = \text{WHITE}$ 
3    $u.\pi = \text{NIL}$ 
4    $\text{time} = 0$ 
5   for each vertex  $u \in G.V$ 
6     if  $u.\text{color} == \text{WHITE}$ 
7       DFS-VISIT( $G, u$ )
  
```

DFS-VISIT(G, u)

```

1  $\text{time} = \text{time} + 1$            // white vertex  $u$  has just been discovered
2  $u.d = \text{time}$ 
3  $u.\text{color} = \text{GRAY}$ 
4 for each vertex  $v$  in  $G.\text{Adj}[u]$  // explore each edge  $(u, v)$ 
5   if  $v.\text{color} == \text{WHITE}$ 
6      $v.\pi = u$ 
7     DFS-VISIT( $G, v$ )
8  $\text{time} = \text{time} + 1$ 
9  $u.f = \text{time}$ 
10  $u.\text{color} = \text{BLACK}$           // blacken  $u$ ; it is finished
  
```



La ricerca depth-first (DF) esplora gli archi che partono dal vertice v scoperto più recentemente e che sono ancora insospettati, la ricerca fa retroscorrere per esplorare

gli archi uscenti dal vertice v de cui avrà raggruppato.
Il processo continua finché non sono stati esplorati tutti
i vertici raggruppabili del vertice sorgente. Se sul grafo
ci sono ancora vertici non esplorati, la ricerca DF
seleziona un nuovo vertice sorgente e ripete le procedure.

Possiamo definire il sottografo $G_\pi = (V, \mathcal{E}_\pi)$, con
 $\mathcal{E}_\pi = \{(v, \pi, v) : v \in V \text{ e } v.\pi \neq \text{nil}\}$.

G_π forma una foresta depth-first che comprende diversi
alberi depth-first.

Gli archi di \mathcal{E}_π sono chiamati archi dell'albero
(tree edges).

La ricerca DF colora i vertici del grafo su input per indicare il loro stato.

All'finalizzazione tutti i vertici sono bianchi.

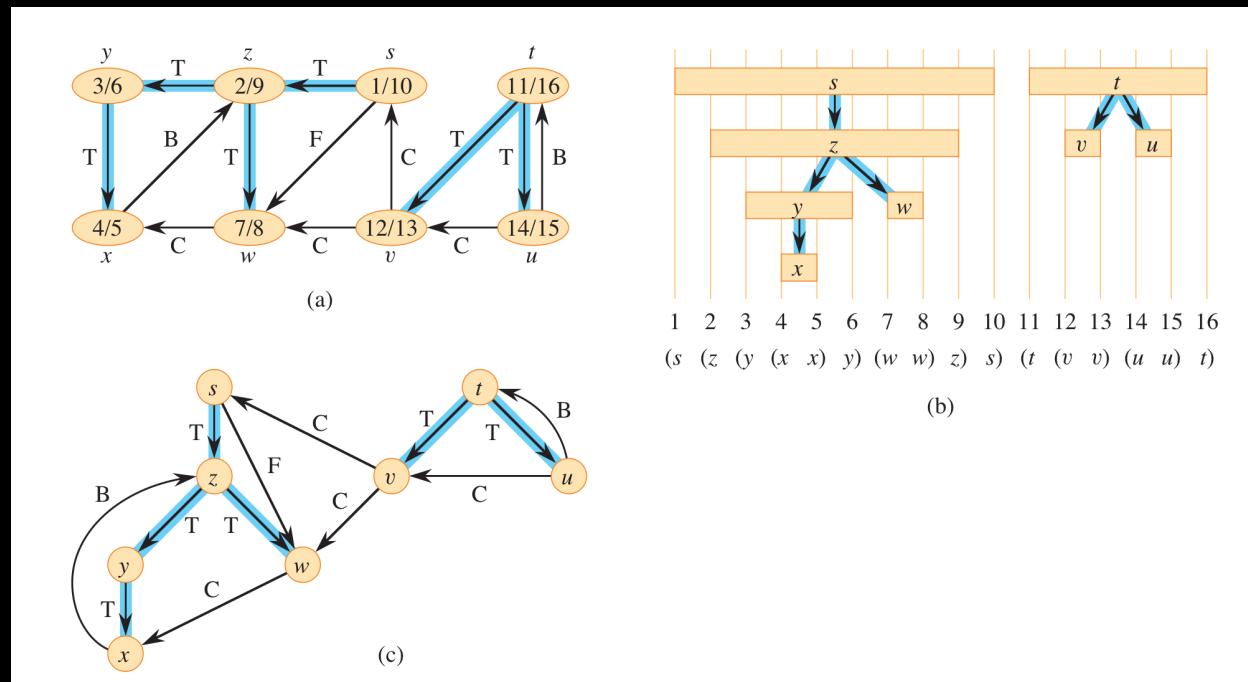
Quando un vertice viene scoperto diventa grigio e si fruisce di vento verso quando le sue esplorazioni terminate, ovvero quando tutti gli archi che dipendono da esso sono stati esplorati.

Abbiamo pure due timestamp per ogni vertice v :

- $v.d$ (discovery - true)
- $v.f$ (finish - true).

Durante un'esecuzione di $\text{DFS-VISIT}(G, v)$ il ciclo alle linee 4-7 viene eseguito $|\text{Adj}(v)|$ volte. Poiché DFS-VISIT viene

Invocato una volta per ogni vertice e $\sum_{v \in V} |\text{Adj}(v)| = O(|E|)$, il tempo richiesto da DFS è $O(|V| + |E|)$.



THM (PARENTHESIS THM): Se una ricerca DF di un grafo $G(V, E)$ (selettivo o meno), A passo di vertici $u, v \in V$ esattamente una delle seguenti tre condizioni si verifica:

- $[u.d, u.f] \cap [v.d, v.f] = \emptyset$ e nessuno tra u e v

è discendente dell'altro;

- $[u.d, u.f] \subseteq [v.d, v.f]$ e u è un discendente di v ;
- $[v.d, v.f] \subseteq [u.d, u.f]$ e v è un discendente di u .

COR (ANNIDAMENTO DEGLI INTERVALLI DEI DISCENDENTI): In una foresta DF di un grafo $G(V, E)$ (orientato o no), $u, v \in V$

v è un discendente $\iff u.d < v.d < v.f < u.f$.
proprio di u

THM (WHITE PATH THEOREM): In una foresta DF di un grafo

$G = (V, E)$ (orientato o no), $u, v \in V$

v è un discendente di $u \iff$ al tempo $u.d \exists$ un cammino
da u a v tutto fatto di vertici bianchi.

$G\pi$ può contenere 4 tipi di archi di G

- archi dell'albero: (u,v) è un arco dell'albero se è stato scoperto esplorando (u,v) .
- archi all'indietro: archi (u,v) che connettono un vertice u ad un suo antenato v in un albero DF. Tutti i self-loop sono archi all'indietro.
- archi in avanti: archi (u,v) che non sono archi dell'albero e connettono un vertice u a un discendente proprio dell'albero DF.
- archi d'attraversamento: tutti gli altri archi. Connellono vertici dello stesso albero fruttanto che nessun vertice è direttamente antenato dell'altro. Oppone connelloni vertici che sono in diversi alberi del bosco.

RHN: Si ricercare DF di un grafo non-orientato G , ogni
arco di G è un arco dell'albero o un arco all'indietro.