Lezione 4 maggio

Monday, May 4, 2020 11:01 AM

Programmazione Dinamica

Zaino (Bisaccia, Ladro) Knapsack

n oggetti : S insieme

valore $p_1 \dots p_n$ W

peso $w_1 \dots w_n$

Il ladro vuole determinare un sottoinsieme $S' \subseteq S$:

$$\max \sum_{i \in S'} p_i \text{ col vincolo } \sum_{i \in S'} w_i \leq W$$

0-1 Knapsack è un problema "difficile" è NP-hard

Knapsack frazionabile è facile

	peso	valore	valore/peso
X_1	5	10	2
X_2	4	6	1.5
X_3	4	5	1.2

$W = 8$

Algoritmo greedy

- ordina gli oggetti per valore decrescente
- Seleziona in ordine finché entrano nello zaino

$\{X_1\}$ val = 10 sol. alg. Greedy

$\{X_2, X_3\}$ val 11 sol. ottima per questo esempio

Alg. Greedy complessità $\Theta(n \log n)$

Anche considerando l'ordinamento

Anche considerando l'ordinamento in funzione di val/peso, l'alg. Greedy dà nell'esempio il medesimo risultato.

Zaino frazionabile

$\forall i$ si può inserire nello zaino una quantità (dose) di a_i

$$0 \leq \text{dose}(a_i) \leq 1$$

la tecnica greedy funziona

$\Theta(n \log n)$

greedy < goloso / avido

1) $Z(n, W)$ n oggetti e peso W

$Z(i, j)$ è il max valore ottenuto considerando $a_1 \dots a_i$ (un sottoinsieme di) con peso massimo $= j$. $0 \leq j \leq W$

2) Sottoproblemi elementari

$Z(0, j) = \emptyset$

$Z(i, \emptyset) = \emptyset$

3) Regola ricorsiva:

$Z(i, j) = \max$ < $Z(i-1, j)$; $Z(i-1, j-w_i) + v_i$

$$Z(i,j) = \max \begin{cases} \\ Z(i-1, j-w_i) + p_i \end{cases}$$

$j - w_i \geq 0$

- selezione l'elemento i-esimo
- non selezione " "

$a_i \quad p_i \quad w_i$

4) Si alloca una tabella bidimensionale

$Z(0..n, 0..W)$

Esempio:

	a_1	a_2	a_3
p	60	100	120
w	1	2	3

$W = 5$

Sol. ottima $\{a_2, a_3\}$

valore 220

peso $5 \leq 5$

Z

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	60	60	60	60	60
2	0	60	100	160	160	160
3	0	60	100	160	180	220

$Z(2,4) = \max \{ Z(1,4) = 60 \quad Z(1,2) + p_2 = 160 \}$

$$Z(i,j) = \max \begin{cases} Z(i-1, j) & \text{non sel. } a_i \\ Z(i-1, j-w_i) + p_i \end{cases}$$

↑ seleziono a_i

$Z(n,W)$ contiene il valore della soluzione ottima.

Un alg. di programmazione dinamica si può applicare se è vera la ottimalità delle sottostrutture.

Complessità tempo $\Theta(n \cdot W)$

spazio $\Theta(n \cdot W)$

0-1 Zaino è un problema difficile

è NP - hard

nessuno ha mai trovato un algoritmo polinomiale di soluzione

La complessità è sì esponenzialmente polinomiale non lo è

è PSEUDO POLINOMIALE

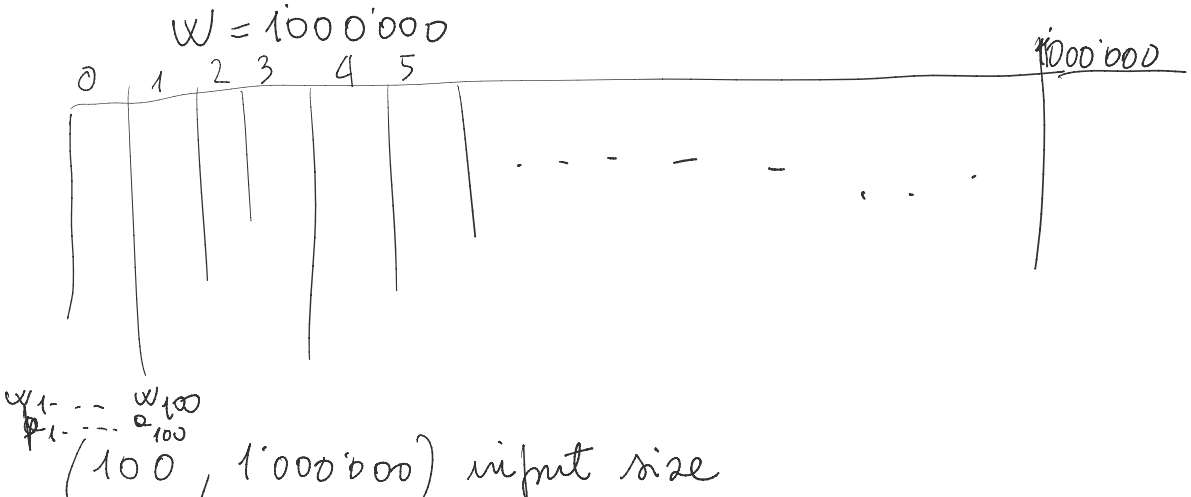
$\Theta(n \cdot W)$ qual'è la dim. dell'input

$n, \log W$

W è esponenziale rispetto a $\log W$

$\log W = t \qquad W = 10^t$

L'algoritmo di programmazione dinamica è esponenziale rispetto alla capacità dello zaino



$w_1 \dots w_{100}$

$v_1 \dots v_{100}$

(100, 1'000'000) input size

$\psi_1 \dots \psi_{100}$

$(100, 1'000'000)$ input size

la tabella ha un milione di colonne!

Algoritmo <u>brute-force</u> per lo Zaino.

S $\quad$ S' è un sottoinsieme

$w_1 \dots w_n$ $\quad$ di oggetti

$p_1 \dots p_n$

Uso <u>vettore caratteristico</u> $\quad n = 4$

0	1	1	0
a_1	a_2	a_3	a_4

$\{a_2, a_3\}$

1	0	1	1

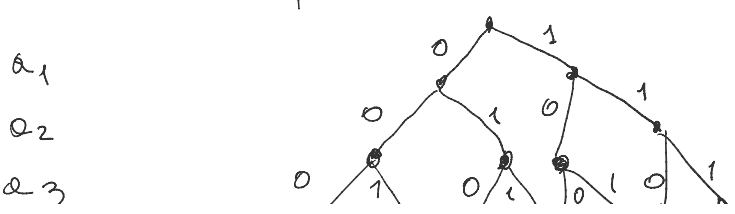
$\{a_1, a_3, a_4\}$

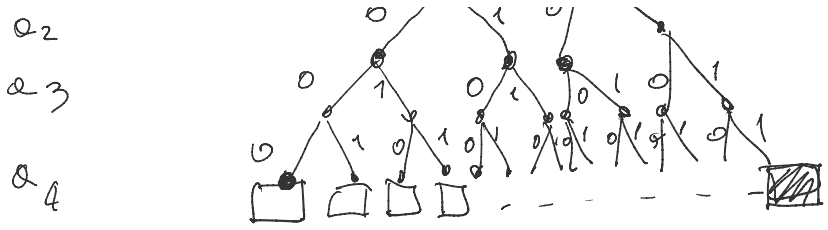
Alg. brute force

- Genera tutti i sottoinsiemi (stringhe binarie)
- Per ogni sottoinsieme generato controlla se è una possibile soluzione e se è la max trovata finora. Nel caso la ricorda

0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
⋮
1 1 1 1

→ $a_2\ a_4$ si controlla che $w_2 + w_4 \leq W$ $p_2 + p_4$ sia max corrente.





- Genera soluzione
- Controllo se la soluzione è "buona"

Quanto costa? input size $(n, \log W)$
come le foglie di un albero binario di altezza n = ? = $2^n \cdot n$

Algoritmo "brute-force" esp. in n = $\Theta(2^n)$

Algoritmo PD è esponenziale in $\log W$

Se abbiamo un problema 0-1 knapsack con n "piccolo" e W "grande" conviene Brute-force.

Se invece n è grande e W relativamente piccolo allora usa PD

$n = 100$ e $W = 200$.