Il problema dello zaino

Programmazione Dinamica

Maria Rita Di Berardini, Emanuela Merelli[1]

[1]Dipartimento di Matematica e Informatica
Università di Camerino

25 gennaio 2010

Il problema dello zaino 0-1

- Un ladro entra in un magazzino e trova n oggetti $\{a_1, ..., a_n\}$ di peso $w_1, ..., w_n$ e valore $v_1, ..., v_n$
- Il ladro vuole realizzare il furto di maggior valore compatibile con la capacità W del suo zaino
- Ciascun oggetto può essere preso per intero o lasciato; il ladro non può prendere frazioni di oggetti
- Formalmente:

$$\max \sum_{i=1}^{n} x_i \cdot v_i$$

con vincoli

$$\sum_{i=1}^{n} x_i \cdot w_i \leq W$$

$$x_i \in \{0, 1\} \qquad \text{per ogni } i = 1, \ldots, n$$

Il problema dello zaino frazionario

- Un ladro entra in un magazzino e trova n oggetti $\{a_1, ..., a_n\}$ di peso $w_1, ..., w_n$ e valore $v_1, ..., v_n$
- Il ladro vuole realizzare il furto di maggior valore compatibile con la capacità W del suo
- Il ladro può prendere frazioni di oggetti
- Formalmente:

$$\max \sum_{i=1}^{n} x_i \cdot v_i$$

con vincoli

$$\sum_{i=1}^{n} x_i \cdot w_i \leq W$$

$$0 \leq x_i \leq 1 \qquad \text{per ogni } i = 1, \ldots, n$$

Problema dello zaino 0-1

Prima soluzione: la forza bruta

- Consiste nell'enumerate tutte le possibili combinazioni degli n oggetti, valutare l'ammissibilità di ciascuna di esse (la somma dei pesi degli oggetti selezionati non deve eccedere la capacità complessiva dello zaino) e scegliere una combinazione ammissibile con valore maggiore
- Quante sono tutte le possibili combinazioni di n oggetti? Abbiamo due possibili scelte per ogni oggetto (0 o 1), il che significa 2^n combinazioni
- Un algoritmo di questo tipo ha un costo computazionale $O(2^n)$ – non accettabile

Spazio dei sottoproblemi

Denotiamo con $S(i, w)$ il sottoproblema in cui il ladro può sceglire oggetti nell'insieme $\{a_1, \cdots a_i\}$ (con $1 \leq i \leq n$) per realizzare il furto di maggior valore compatibile con una capacità residua dello zaino pari a $1 \leq w \leq W$

Problema originale: $S(n, W)$

Casi banali:

- $S(0, w)$ (insieme degli oggetti vuoto), e
- $S(i, 0)$ (capacità residua pari a zero);

in entrambi i casi non possiamo scegliere alcun oggetto

Sottoproblemi generati

Sia $S(i, w)$ un sottoproblema non banale $(i, w > 0)$. Esaminiamo il peso w_i dell'i-esimo oggetto e consideriamo i seguenti possibili casi:

- $w_i > w$: l'oggetto a_i non può essere selezionato - risolviamo il sottoproblema $S(i-1, w)$
- $w_i \leq w$: abbiamo due possibili alternative
  1. scegliamo di prendere l'oggetto a_i e poi risolviamo $S(i-1, w-w_i)$, oppure
  2. ignoriamo l'oggetto a_i e risolviamo $S(i-1, w)$

  Tra le due, scegliamo quella con valore maggiore

In ogni caso, una soluzione ottima per $S(i, w)$ deve contenere una soluzione per uno dei sottoproblemi $S(i-1, w)$ e $S(i-1, w-w_i)$

Sottostruttura ottima

- Sia $A(i, w)$ una soluzione ottima per $S(i, w)$ (con $i, j > 0$)
- Consideriamo il caso in cui $A(i, w) = \{a_i\} \cup A(i - 1, w - w_i)$, con $A(i - 1, w - w_i)$ è una soluzione per $S(i - 1, w - w_i)$

Sottostruttura ottima

- Sia $A(i, w)$ una soluzione ottima per $S(i, w)$ (con $i, j > 0$)
- Consideriamo il caso in cui $A(i, w) = \{a_i\} \cup A(i - 1, w - w_i)$, con $A(i - 1, w - w_i)$ è una soluzione per $S(i - 1, w - w_i)$
- Assumiamo ora, per assurdo, che $A(i - 1, w - w_i)$ non sia ottima
- Deve esistere una soluzione alternativa $A'(i - 1, w - w_i)$ per lo stesso sottoproblema il cui valore (somma dei valori degli oggetti selezionati) è maggiore di quello di $A(i - 1, w - w_i)$

Sottostruttura ottima

- Sia $A(i, w)$ una soluzione ottima per $S(i, w)$ (con $i, j > 0$)
- Consideriamo il caso in cui $A(i, w) = \{a_i\} \cup A(i-1, w-w_i)$, con $A(i-1, w-w_i)$ è una soluzione per $S(i-1, w-w_i)$
- Assumiamo ora, per assurdo, che $A(i-1, w-w_i)$ non sia ottima
- Deve esistere una soluzione alternativa $A'(i-1, w-w_i)$ per lo stesso sottoproblema il cui valore (somma dei valori degli oggetti selezionati) è maggiore di quello di $A(i-1, w-w_i)$
- Sia $A'(i, w) = \{a_i\} \cup A'(i-1, w-w_i)$. Allora:

$$\begin{aligned} v(A'(i, w)) &= v_i + v(A'(i-1, w-w_i)) \\ &> v_i + v(A(i-1, w-w_i)) = v(A(i, w)) \end{aligned}$$

Sottostruttura ottima

- Sia $A(i, w)$ una soluzione ottima per $S(i, w)$ (con $i, j > 0$)
- Consideriamo ora il caso in cui $a_i \notin A(i, w)$ e (quindi) $A(i, w)$ contiene al suo interno una soluzione $A(i-1, w)$ per $S(i-1, w)$
- Assumiamo (per assurdo) che $A(i-1, w)$ non sia ottima
- Deve esistere una soluzione alternativa $A'(i-1, w)$ per $S(i-1, w)$ il cui valore (somma dei valori degli oggetti selezionati) è maggiore di quello di $A(i-1, w)$
- Allora:

$$v(A'(i, w)) = v(A'(i-1, w)) > v(A(i-1, w)) = v(A(i, w))$$

Numero di sottoproblemi

Abbiamo Wn sottoproblemi della forma $S(i, w)$ con $1 \leq i \leq n$ e $1 \leq w \leq W$

Problema originale: $S(n, W)$

Casi banali:

- W sottoproblemi della forma $S(0, w)$, e
- n sottoproblemi della forma $S(i, 0)$.

In totale $Wn + W + n = O(Wn)$ sottoproblemi

Calcolo del valore della soluzione ottima

Sia $v(i, w)$ il valore della soluzione ottima per il sottoproblema $S(i, w)$, dove $i = 0, \cdots, n$ e $w = 0, \cdots, W$

- (Casi banali) $i = 0$ oppure $w = 0$: $v(i, 0) = v(0, w) = 0$ per ogni $i = 0, \cdots, n$ e $w = 0, \cdots, W$
- (Casi non banali) $i, w > 0$. In questo caso:

$$v(i, w) = \begin{cases} v(i-1, w) & \text{se } w_i > w \\ \max\{v(i-1, w), v(i-1, w-w_i)\} & \text{se } w_i \leq w \end{cases}$$

Identificazione di uno schema Bottom-Up

Possiamo usare una matrice V di dimensione $n+1 \times W+1$ per memorizzare risultati parziali

$i \backslash w$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

Casi Base: $V[i,0] = V[0,j] = 0$ per ogni $i = 0, ..., n$ e $w = 0, ..., W$

Identificazione di uno schema Bottom-Up

$i \backslash w$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0	⊙	⊙	⊙	⊙	
3	0				⊙	
4	0					

$V[i, w]$ dipende dai valore $V[i-1, w]$e $V[i-1, w-w_i]$

Identificazione di uno schema Bottom-Up

$i \backslash w$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0	⊙	⊙	⊙	⊙	
3	0				⊙	
4	0					

$V[i, w]$ dipende dai valore $V[i-1, w]$e $V[i-1, w-w_i]$

Ci basta riempire la matrice riga per riga

Algoritmo

KNAPSACK01($\mathbf{w}, \mathbf{v}, W$)

```
 1  n ← length[v]
 2  for w ← 0 to W
 3  do V[0, w] ← 0
 4  for i ← 0 to n
 5  do V[i, 0] ← 0
 6  for i ← 1 to n
 7  do for w ← 1 to W
 8       do if w > w[i]
 9            then V[i, w] ← V[i − 1, w]
10                 A[i] ← 0
11            else  if V[i − 1, w] > V[i − 1, w − w[i]
12                    then V[i, w] ← V[i − 1, w]
13                         A[i] ← 0
14                    else  V[i, w] ← V[i − 1, w − w[i]]
15                         A[i] ← 1
16  return V, A
```