Programmazione dinamica

Violetta Lonati

Università degli studi di Milano
Dipartimento di Scienze dell'Informazione

Laboratorio di algoritmi e strutture dati
Corso di laurea in Informatica

Argomenti

Scheduling di intervalli pesati

Il problema dello zaino

Argomenti

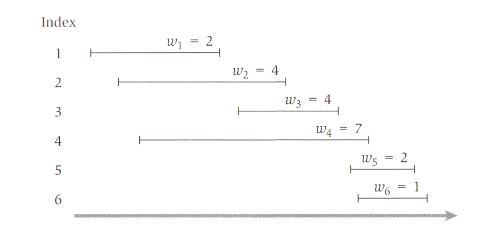
Scheduling di intervalli pesati

Il problema dello zaino

Scheduling di intervalli pesati - il problema

Intervallo: (i, f, v), dove

- i è il tempo di inizio,
- f è il tempo di fine,
- v è il valore (o peso) dell'intervallo.



Dato un insieme I di intervalli, una soluzione al problema dello scheduling è data da un sottoinsieme $S \subseteq I$ di intervalli che non si sovrappongono fra loro.
Il valore di una soluzione S è dato dalla somma dei valori degli intervalli contenuti in S

Scheduling di intervalli pesati - continua

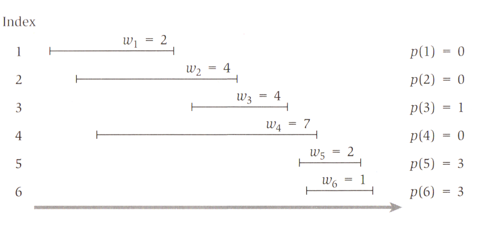
Ordiniamo gli elementi di i in base al tempo di fine:

$$I = \{(i_1, f_1, v_1), (i_2, f_2, v_2), \ldots, (i_n, f_n, v_n)\}$$

$$f_1 \leq f_2 \leq \ldots \leq f_n$$

Definizione

Per ogni indice j tra 1 e n, sia $p(j)$ come il più grande indice $i < j$ tale che l'intervallo di indice i non si sovrappone all'intervallo di indice j.



Scheduling di intervalli pesati - soluzione ricorsiva

Detto $Opt(j)$ il valore di una qualsiasi soluzione ottimale S_j costruita usando gli intervalli di indici $\{1, 2, \ldots, j\}$, vale questa relazione ricorsiva:

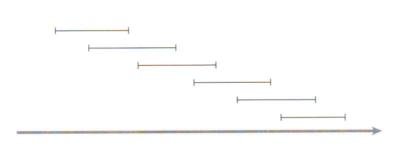
$$Opt(j) = max\{v_j + Opt(p(j)), Opt(j-1)\}$$

- se $j \in S_j$, allora $Opt(j) = v_j + Opt(p(j))$,
- altrimenti $Opt(j) = Opt(j-1)$.

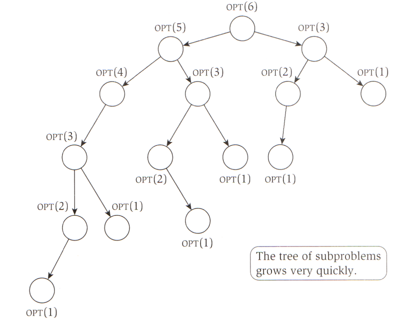
```
Compute-Opt(j)
  If j=0 then
    Return 0
  Else
    Return max(v_j+Compute-Opt(p(j)), Compute-Opt(j−1))
  Endif
```

Scheduling di intervalli pesati - complessità?

Ogni sottoproblema può venire calcolato molte volte!!



Su questa istanza del problema



le chiamate ricorsive "esplodono"!

Scheduling di intervalli pesati - *Memoization*

Memorizziamo le soluzioni dei sottoproblemi in un vettore, riducendo le chiamate ricorsive:

```
M-Compute-Opt(j)
  If j = 0 then
    Return 0
  Else if M[j] is not empty then
    Return M[j]
  Else
   Define M[j] = max(v_j+M-Compute-Opt(p(j)), M-Compute-Opt(j-1))
    Return M[j]
  Endif
```

Scheduling di intervalli pesati - Ricostruire la soluzione

Non c'è bisogno di aggiungere informazioni. C'è già tutto nel vettore!

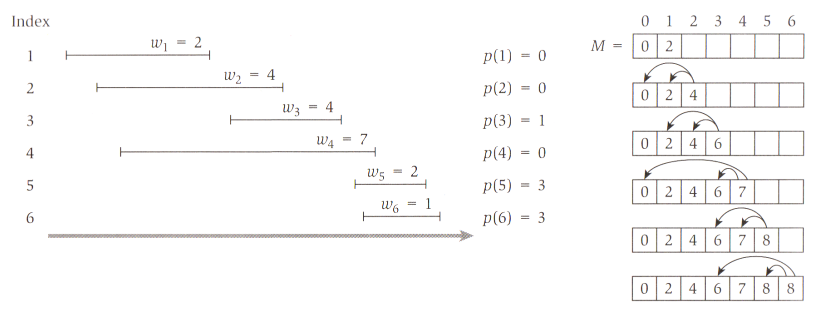
```
Find-Solution(j)
  If j = 0 then
    Output nothing
  Else
    If v_j + M[p(j)] ≥ M[j − 1] then
      Output j together with the result of Find-Solution(p(j))
    Else
      Output the result of Find-Solution(j − 1)
    Endif
  Endif
```

Scheduling di intervalli pesati - Programmazione dinamica

Costruiamo il vettore delle soluzioni dei sottoproblemi senza ricorsione:

```
Iterative-Compute-Opt
  M[0] = 0
  For j = 1, 2, ..., n
    M[j] = max(v_j + M[p(j)], M[j - 1])
  Endfor
```

Scheduling di intervalli pesati - Esempio di esecuzione



Argomenti

Scheduling di intervalli pesati

Il problema dello zaino

Il problema dello zaino - versione semplificata

Dati:

- uno zaino che sopporta un peso massimo P,
- un insieme $T = \{1, 2, \ldots, n\}$ di *tipi* di oggetti. Ogni tipo i di oggetti ha un peso p_i e un valore v_i, entrambi interi positivi. (Per ogni tipo è disponibile una fornitura illimitata di oggetti.)

Problema:

Vogliamo riempire lo zaino non superando P con il peso complessivo degli oggetti, ma allo stesso tempo massimizzando la somma dei valori degli oggetti nello zaino.

Una soluzione S è data da una lista di tipi, eventualmente ripetuti. Es: 3 oggetti di tipo 1 + 2 oggetti di tipo 4 + 1 oggetto di tipo 5.

$$\sum_{i \in S} p_i =\leq P, \qquad \text{MAX} \left\{ \sum_{i \in S} v_i \mid S \right\}$$

Il problema dello zaino (sempl.) - ricerca esaustiva

È chiaro che in linea di principio potremmo provare a enumerare tutti gli insiemi di oggetti che stanno nello zaino, e cercare quello con valore massimo. Se però i tipi sono molti, e la capacità dello zaino grande, il numero di soluzioni da esaminare diventa improponibile.

Il problema dello zaino (sempl.) - sottoproblemi

Struttura ricorsiva del problema - sottoproblemi

Se ho una soluzione ottima per uno zaino che regge P, e tolgo dalla soluzione un oggetto qualsiasi di tipo t, ottengo una soluzione ottima per uno zaino che regge $P - p_t$.

Dimostrazione per assurdo:

Infatti, se per assurdo esistesse una soluzione migliore con peso inferiore a $P - p_t$, potrei aggiungerle un oggetto di tipo t e ottenere così una soluzione migliore per il problema originale (il che è impossibile, avendo assunto che la soluzione fosse ottima).

Il problema dello zaino (sempl.) - progr. dinamica

- Se conosciamo la soluzione ottima per uno zaino di peso Q, possiamo ottenere nuove soluzioni per zaini di grandezza superiore aggiungendo un oggetto di tipo t, per ogni tipo t in T.
- In particolare, se ho una soluzione di valore V per uno zaino di peso Q, allora so che esiste una soluzione di valore $V + v_t$ per uno zaino di peso $Q + p_t$, per ogni t in T.
- *Tutte le soluzioni ottime si ottengono in questo modo.*

Il problema dello zaino (sempl.) - vettore di supporto

- Costruisco un vettore s di lunghezza P, tale che $s[i]$ contenga il valore delle soluzioni ottime per uno zaino di peso i.
- Il vettore si può costruire partendo da $i = 0$ e incrementando i, sfruttando la relazione vista prima.
- Una volta completato il vettore s, il valore ottimo di una soluzione per P è chiaramente dato da $s[P]$.

Il problema dello zaino (sempl.) - esempio

Tabella dei pesi e dei valori dei tipi:

Tipo	Peso	Valore
1	5	2
2	3	3
3	7	8

Nel seguente schema: la prima riga riporta gli indici, ciascuna delle righe successive rappresenta un passo dell'esecuzione:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0															
0	0		3		2		8								
0	0	0	3	3	2	2	8	8							
0	0	0	3	3	3	2	8	8	8						
0	0	0	3	3	3	6	8	8	8	11					

...

Il problema dello zaino - versione generale

- uno zaino che sopporta un peso massimo P,
- un insieme $E = \{1, 2, \ldots, m\}$ di oggetti. Ogni oggetto ha un peso p_i e un valore v_i, entrambi interi positivi.

Problema:

Vogliamo riempire lo zaino non superando P con il peso complessivo degli oggetti, ma allo stesso tempo massimizzando la somma dei valori degli oggetti nello zaino.

Riduzione?

Non possiamo più utilizzare la riduzione precedente: infatti, se tolgo un oggetto e da una soluzione ottima per uno zaino che porta P, *non è detto che quanto rimane sia una soluzione ottima per uno zaino che porta $P - p_e$*. Infatti, utilizzando e potrei ottenere una soluzione migliore per quel peso, e a questo punto non potrei aggiungerlo nuovamente: la dimostrazione per assurdo non funziona più.

Il problema dello zaino - riduzione a sottoproblemi

Riduco rispetto a due parametri: peso e numero di oggetti considerati!

Cerco soluzione ottima $S_{P,j}$ per tutti gli zaini di peso inferiore a P e *per tutti gli insiemi di oggetti* $1, 2, \ldots, j$ *con* $j \leq m$.

Struttura ricorsiva

Sia $S_{P,j}$ soluzione ottima di valore V per uno zaino di peso P che utilizza gli oggetti $1, 2, \ldots, j$. Ci sono due possibilità:

- se $j \in S_{P,j}$, allora esiste soluzione ottima $S_{P-p_j,j-1}$ di valore $V - v_j$
- se $j \notin S_{P,j}$, allora esiste soluzione ottima $S_{P,j-1}$ di valore V (data dallo stesso sottoinsieme!)

Il problema dello zaino - riduzione a sottoproblemi

Un vettore non sarà sufficiente!

Uso una matrice

che contiene nella posizione di indici i e j il valore della sottosoluzione ottima per uno zaino di peso i che utilizza al più i primi j oggetti di E (chiaramente, $0 \leq i \leq P$ e $0 \leq j \leq m$).

Costruisco la matrice

a partire dalla posizione $(0, 0)$.

Il valore ottimo

sarà nella posizione (P, m).