Laboratorio di Algoritmi e Strutture Dati

Esercitazioni del 22 Gennaio 2013

Esercizio 1: zaino

Si tratta di risolvere il classico problema dello zaino tramite tecniche di programmazione dinamica.
Vengono proposte due varianti.

Ripetizione illimitata di oggetti

Un ladro con uno zaino in spalla penetra in un magazzino.
Lo zaino ha capacità (nel senso che regge al massimo un peso) C, per C un intero positivo.
Nel magazzino vi sono $t_1, \ldots, t_k$ *tipi di oggetti*. Si suppone che per ogni tipo t_i di oggetti il magazzino contenga un numero illimitato di oggetti di quel tipo.
A ogni tipo di oggetto t_i è associato un peso p_i e un valore v_i, con p_i, v_i interi positivi.
Il ladro deve mettere nello zaino una collezione di oggetti tali che la somma dei valori di tutti gli oggetti scelti sia massima rispetto a tutte le collezioni di oggetti il cui peso complessivo non ecceda C.
Formalmente:

- Una collezione di oggetti è una k-pla $(q_1, \ldots, q_k) \in \mathbb{N}^k$, dove q_i, per q_i un intero ≥ 0, indica quanti oggetti di tipo t_i appartengono alla collezione.
- il peso di una collezione $(q_1, \ldots, q_k)$ è la somma $\sum_{i=1}^{k} q_i p_i$, mentre il valore della collezione è la somma $\sum_{i=1}^{k} q_i v_i$.
- Una *soluzione ammissibile* al problema dello zaino (con ripetizione illimtata di oggetti) è una collezione $(z_1, \ldots, z_k)$ tale che valga $\sum_{i=1}^{k} z_i p_i \leq C$.
- Una *soluzione ottimale* al problema dello zaino (con ripetizione illimtata di oggetti) è una soluzione ammissibile $(z_1, \ldots, z_k)$ tale che per il suo valore $\sum_{i=1}^{k} z_i v_i$ valga

  $$\sum_{i=1}^{k} z_i v_i \geq \sum_{i=1}^{k} q_i v_i$$

  per ogni possibile collezione ammissibile $(q_1, \ldots, q_k) \in \mathbb{N}^k$.

Il programma deve poter essere lanciato come:

`zaino` *file*

dove *file* è un file che descrive un'istanza del problema nel seguente formato: (si vedano i file `zaino.txt` e `zaino2.txt` nel materiale predisposto).

```
C k

nome1 p1 v1
```

```
nome2 p2 v2
... ... ...
nomek p2 vk
```

dove `C` è la capacità dello zaino; `k` è il numero di tipi di oggetti; `nome1,nome2,...,nomek` sono stringhe di lunghezza massima 20 caratteri che contengono i nomi dei tipi di oggetto, `p1,p2,...,pk` e `v1,v2,...,vk` sono rispettivamente i pesi e i valori dei tipi di oggetto.

Il programma deve:

1. Leggere il file e collocare i valori letti in opportune strutture (array).
2. Calcolare una soluzione ottimale per l'istanza del problema.
3. Stampare a video la soluzione nel formato descritto nell'esempio:

   Il file `zaino.txt` contiene i seguenti dati:

   ```
   20  3

   frutta 5  2
   tonno  3  3
   acqua  7  8
   ```

   Dunque il programma deve stampare la seguente soluzione:

   ```
   Il valore totale dello zaino e' 22
   Lo zaino contiene:
   2 di tonno
   2 di acqua
   ```

Considerazioni implementative

La soluzione ingenua consisterebbe nel generare *tutte* le collezioni il cui peso sia $\leq C$ e di scegliere fra queste una collezione di valore massimo.

Ma se la capacità C e il numero k di tipi di oggetti è elevato, questa soluzione è impraticabile.

Ma: **osservazioni sulla struttura dei sottoproblemi**:

Se conosco una soluzione ottimale $(q_1, \ldots, q_k)$ per uno zaino di capacità C allora, per ogni tipo di oggetto t_i di peso p_i e tale che $q_i > 0$, la k-pla $(q_1, \ldots, q_i - 1, \ldots, q_k)$ è una soluzione *ottimale* per uno zaino di capacità $C - p_i$.

Se conosco una soluzione ottimale $(q_1, \ldots, q_k)$ per uno zaino di capacità $C' < C$ allora, per ogni tipo di oggetto t_i di peso $p_i \leq (C - C')$, ho che $(q_1, \ldots, q_i + 1, \ldots, q_k)$ è una soluzione *ammissibile* per uno zaino di capacità $C' + p_i$.

Possiamo sfruttare queste osservazioni come segue.

Supponiamo per esempio che $C = 20$ e che vi siano 3 tipi di oggetti con pesi e valori dati da:

```
p1 = 5, v1 = 2
p2 = 3, v2 = 3
p3 = 7, v2 = 8
```

- Predisponiamo un array `zaino` di $C+1$ interi tali che per ogni $i = 0, 1, \ldots, C$, l'elemento `zaino[i]` contenga alla fine della procedura il valore di una soluzione ottimale per uno zaino di capacità i. All'inizio tutte le posizioni dell'array sono poste a 0.

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16,17,18,19,20      (indice i)
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0      (zaino[i])
```

- Dato che conosciamo a priori la soluzione per uno zaino di capacità 0 (esso contiene 0 oggetti di ogni tipo e dunque il suo valore è 0), l'elemento `zaino[0]` già contiene il valore appropriato 0. Possiamo utilizzare questo valore per costruire soluzioni ammissibili (non ancora necessariamente ottimali) per tutte le posizioni di `zaino` ottenute sommando a `zaino[0]` il peso di un oggetto: vale a dire, se $p_i \leq C$ allora poniamo in `zaino[pi]` il valore v_i.

```
0 | 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16,17,18,19,20      (indice i)
0 | 0, 0, 3, 0, 2, 0, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0      (zaino[i])
        t2    t1    t3
```

- Proseguiamo in questo modo considerando una soluzione per uno zaino di capacità 1. Notiamo che `zaino[1]` deve rimanere 0 perchè non possiamo aggiungere oggetti all'attuale valore di `zaino[1]` senza eccedere C, ma possiamo però propagare le soluzioni ammissibili ottenute, sommando a 1 i pesi di ogni tipo di oggetto e collocando nelle posizioni ottenute `zaino[1 + pi]` il valore `zaino[1]` sommato al valore dell'oggetto.

```
0, 1 | 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16,17,18,19,20      (indice i)
0, 0 | 0, 3, 3, 2, 2, 8, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0      (zaino[i])
           t2    t1     t3
```

- Al passo successivo ragioniamo allo stesso modo per uno zaino di capacità 2. Dunque `zaino[2] = 0` e propaghiamo le soluzioni ammissibili attuali nelle posizioni `zaino[2 + pi]`, *se e solo se* le soluzioni propagate sono migliori di quelle attuali. Ad esempio `zaino[2 + p2]` viene aggiornata, dato che $0 + v_2 >$`zaino[2 + p2]`, mentre `zaino[2 + p1]` non viene aggiornata dato che $0 + v_1 <$`zaino[2 + p1]`.

```
0, 1, 2 | 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16,17,18,19,20      (indice i)
0, 0, 0 | 3, 3, 3, 2, 8, 8, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0      (zaino[i])
              t2    t1    t3
```

- Al passo 3 ragioniamo allo stesso modo. Si noti che `zaino[6]` viene posto a `6`, il valore corrispondente a 2 oggetti di tipo t_2, dato che $p_2 + p_2 \leq 6$ e $2v_2$ è maggiore del vecchio valore di `zaino[6]`. Si osservi che $6 = 3 + p_2$ e $2v_2 =$ `zaino`$[3] + v_2$.

```
0, 1, 2, 3 | 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16,17,18,19,20      (indice i)
0, 0, 0, 3 | 3, 3, 6, 8, 8, 8,11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0      (zaino[i])
                 t2    t1    t3
```

- Al generico passo i e per ogni tipo di oggetto j si propaga la soluzione `zaino[i]` alle posizioni `zaino[i + pj]`, a patto che $i + p_j \leq C$ e che `zaino[i] + vj` sia migliore dell'attuale soluzione in `zaino[i + pj]`.

- A questo punto troviamo in `zaino[20]` il valore totale di una soluzione ottimale per lo zaino (nel nostro esempio `zaino[20]` conterrà 22).

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16,17,18,19,20 |     (indice i)
0, 0, 0, 3, 3, 3, 6, 8, 8, 9,11,11,12,14,16,16,17,19,19,20,22 |     (zaino[i])
```

Abbiamo ottenuto il valore (ottimo) della soluzione ottimale, ma la soluzione ottimale trovata come è fatta? Bisogna ricostruirla.

- Per ricostruire la soluzione, al momento della creazione dell'array `zaino`, si crei un array di $C+1$ interi `quale`, inizializzato a 0.
- Al generico passo i dell'algoritmo sopra descritto, non appena si modifica il valore di `zaino` in una sua posizione `k`, diciamo `zaino[k] = zaino[i] + vj` per qualche `j`, si pone `quale[k] = j`, in tal modo registrando che l'ultimo oggetto aggiunto in uno zaino di capacità k è stato un oggetto di tipo t_j.
- Ottenuto il valore della soluzione si consideri l'array `quale`.

  ```
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16,17,18,19,20      (indice i)
  0, 0, 0, 2, 2, 2, 2, 3, 3, 2, 3, 3, 2, 3, 3, 3, 3, 3, 3, 3 3      (quale[i])
  |        |        |                     |                     |
  |---p2---|---p2---|---------p3---------|---------p3---------|
  ```

  In particolare `quale[C]`: il suo valore indica l'ultimo oggetto aggiunto allo zaino — nel nostro esempio `quale[20]` contiene 3, dunque almeno un oggetto di tipo t_3 appartiene alla soluzione.
- Si sottragga adesso a C il peso dell'oggetto trovato — nel nostro esempio si deve sottrarre $p_3 = 7$, dunque il nuovo valore di C è 13.
  Di nuovo `quale[C]` — nel nostro esempio `quale[13]` —, indica un oggetto appartenente alla soluzione — nel nostro esempio un altro oggetto di tipo t_3.
- Si itera con le sottrazioni successive sino ad arrivare a 0. Nel nostro esempio la soluzione ottenuta di valore 22, contiene 2 oggetti di tipo t_3 e 2 oggetti di tipo t_2.

L'algoritmo descritto calcola una soluzione ottimale al problema dello zaino in tempo $O(Ck)$ (Nota: non è comunque un algoritmo polinomiale nella dimensione dell'input (la dimensione del numero C è di $O(\log C)$ bits).

Oggetti non ripetibili

Questa variante del problema riceve lo stesso tipo di input e deve calcolare una soluzione ottimale al problema dello zaino, *sotto il vincolo* che il magazzino contenga esattamente un esemplare di ogni tipo di oggetto.

Dunque una soluzione a questa variante del problema dello zaino è una k-pla $(z_1, \ldots, z_k) \in \{0,1\}^k$ (dunque $z_i = 0$ o $z_i = 1$) di valore $\sum_{i=1}^{k} z_i v_i$ massimo fra tutte le k-ple $(q_1, \ldots, q_k) \in \{0,1\}^k$ per le quali $\sum_{i=1}^{k} q_i p_i \leq C$. Ovviamente $q_i = 1$ indica che l'oggetto t_i appartiene alla collezione, mentre $q_i = 0$ indica che l'oggetto t_i non appartiene alla collezione.

Le osservazioni sui sottoproblemi devono essere modificate di conseguenza. In particolare se ho una soluzione ottimale $(q_1, \ldots, q_i = 1, \ldots, q_k)$ per uno zaino di capacità C, non è detto che togliendo un oggetto di peso p_i ottenga una soluzione ottimale $(q_1, \ldots, q_i - 1 = 0, \ldots, q_k)$ per uno zaino di capacità $C - p_i$. Infatti potrebbe esserci una soluzione ammissibile per la capacità $C - p_i$ che contiene l'oggetto i e che è migliore di $(q_1, \ldots, q_i - 1 = 0, \ldots, q_k)$.

Dobbiamo considerare entrambe le possibilità. Supponiamo di avere una soluzione ottimale $(q_1, \ldots, q_k)$ di capacità C costruita scegliendo gli oggetti da $t_1, t_2, \ldots, t_k$. Allora:

- Se l'oggetto t_k appartiene alla soluzione, dunque $q_k = 1$, allora ho una soluzione $(q_1, \ldots, q_{k-1}, 0)$ per uno zaino di capacità $C - p_k$ che è ottimale se l'insieme degli oggetti da considerare è $t_1, t_2, \ldots, t_{k-1}$.
- Se l'oggetto t_k non appartiene alla soluzione, dunque $q_k = 0$, allora ho una soluzione $(q_1, \ldots, q_{k-1}, 0)$ per uno zaino di capacità C che è ottimale se l'insieme degli oggetti da considerare è $t_1, t_2, \ldots, t_{k-1}$.

Si tratta di aggiungere un oggetto alla volta all'insieme degli oggetti prelevabili.

Per calcolare la soluzione ottimale per la capacità C e l'insieme di oggetti $\{t_1, \ldots, t_k\}$, bisogna dunque usare le soluzioni ottimali ai sottoproblemi caratterizzati da capacità $C' \leq C$ e insieme di oggetti $\{t_1, \ldots, t_h\}$ con $h < k$.

Un vettore non basta più, bisogna utilizzare una matrice `zaino[C + 1][k + 1]` di interi.
`zaino[j][i]` conterrà il valore di una soluzione ottimale per uno zaino di capacità j costruito a partire dall'insieme di oggetti $\{t_1, \ldots, t_i\}$.

La matrice `zaino` è inizializzata a 0 ovunque.

La zeresima riga e la zeresima colonna contengono già le soluzioni ottimali ai problemi corrispondenti, infatti uno zaino di capacità 0, o uno zaino che rimane vuoto perchè non vi sono oggetti disponibili, ha valore massimo 0.

Ciclando su ogni oggetto i e, più internamente, su ogni capacità j, si consulti la soluzione ottimale in `zaino[j][i-1]` e la si confronti con il valore ottenuto sommando alla soluzione ottimale in `zaino[j - pi][i-1]` il valore dell'i-esimo oggetto `vi`. Si scelga la soluzione migliore fra le due.

Si prosegua fino a calcolare il valore di `zaino[C][k]`. Questo è il valore di una soluzione ottimale per uno zaino di capacità `C` costruito a partire dall'insieme di oggetti $\{t_1, \ldots, t_k\}$.

Esempio. Il file `zaino2.txt` contiene i seguenti dati:

```
10  5

rame        3  3
argento     2  4
platino     5  4
litio       2  1
oro         6  7
```

Dunque il programma deve stampare la seguente soluzione:

```
Il valore totale dello zaino e' 12
Lo zaino contiene:
oro
litio
argento
```

L'algoritmo descritto calcola una soluzione ottimale a questa variante del problema dello zaino in tempo $O(Ck)$ (Nota: non è comunque un algoritmo polinomiale nella dimensione dell'input (la dimensione del numero C è di $O(\log C)$ bits).