Algoritmi e Strutture Dati (Complementi)

Appello dell' 8/07/2003 – Prof. Giancarlo Mauri

Soluzioni

Esercizio 1.
Un ente pubblico ha ricevuto un finanziamento di K euro per effettuare degli acquisti. Gli oggetti che vorrebbe acquistare sono n, e hanno un valore commerciale pari a $v_1, v_2, \ldots, v_n$ euro. Purtroppo i soldi non bastano per acquistare tutti gli n oggetti; inoltre, i soldi che non vengono spesi vanno restituiti allo Stato. L'ente vorrebbe pertanto sapere se è possibile acquistare un sottoinsieme degli n oggetti in modo da spendere esattamente K euro, così da non dover restituire nulla allo Stato.

Progettare un algoritmo di programmazione dinamica che aiuti l'ente a decidere se ciò è possibile. In particolare, scrivere prima le equazioni di ricorrenza che esprimono una soluzione del problema e poi lo pseudo–codice dell'algoritmo. Infine, si dica qual è la complessità in tempo dell'algoritmo proposto.

Soluzione. Il problema può essere formalizzato come segue: dato un insieme $V = \{v_1, v_2, \ldots, v_n\}$ di interi positivi, e un intero positivo K, esiste un sottoinsieme $V' \subseteq V$ tale che $\sum_{v \in V'} v = K$?

Definiamo una matrice $M[1..n, 1..K]$ a valori in $\{0, 1\}$, dove per $i \in \{1, \ldots, n\}$ e $j \in \{1, \ldots, K\}$ vale $M[i, j] = 1$ se e solo se esiste un sottoinsieme A di $\{v_1, \ldots, v_i\}$ tale che $\sum_{a \in A} a = j$. Calcolati i valori di M, risponderemo SI se $M[n, K] = 1$, e NO altrimenti. I valori di M verranno calcolati per righe, a partire dalla prima; all'interno di ogni riga, i valori verranno calcolati da sinistra verso destra (e quindi, in particolare, l'elemento $M[n, K]$ sarà l'ultimo ad essere calcolato).

Per calcolare $M[i, j]$, osserviamo che affinché sia $M[i, j] = 1$ ci sono due possibilità:

1. esiste un insieme $A \subseteq \{v_1, \ldots, v_i\}$ tale che $\sum_{a \in A} a = j$ e $v_i \in A$, oppure
2. v_i *non* appartiene a *nessun* insieme $A \subseteq \{v_1, \ldots, v_i\}$ tale che $\sum_{a \in A} a = j$.

Nel primo caso deve esistere un insieme $B \subseteq \{v_1, \ldots, v_{i-1}\}$ tale che $\sum_{b \in B} b = j - v_i$, e quindi deve valere $M[i-1, j-v_i] = 1$ (si noti che deve essere $i > 1$ e $j > v_i$, altrimenti l'elemento $M[i-1, j-v_i]$ non esiste). Nel secondo caso deve esistere un insieme $B \subseteq \{v_1, \ldots, v_{i-1}\}$ tale che $\sum_{b \in B} b = j$, e quindi deve valere $M[i-1, j] = 1$ (sempre con la richiesta che valga $i > 1$).

Pertanto, le equazioni di ricorrenza che consentono di calcolare il valore di $M[i,j]$ sono:

$$M[i,j] = \begin{cases} \max\{M[i-1, j-v_i], M[i-1,j]\} & \text{se } i > 1 \text{ e } j > v_i \\ M[i-1,j] & \text{se } i > 1 \text{ e } j \leq v_i \\ 1 & \text{se } i = 1 \text{ e } v_1 = j \\ 0 & \text{se } i = 1 \text{ e } v_1 \neq j \end{cases}$$

mentre lo pseudocodice dell'algoritmo che calcola i valori della matrice M è il seguente:

```
SUBSET SUM({v_1, v_2, ..., v_n})
for j ← 1 to K
   do M[1, j] ← 0
M[1, v_1] ← 1
for i ← 2 to n
    do for j ← 1 to K
           do M[i, j] ← M[i − 1, j]
              if j > v_i and M[i − 1, j − v_i] > M[i, j]
                 then M[i, j] ← M[i − 1, j − v_i]
return M[n, K]
```

Il tempo di calcolo dell'algoritmo è $\Theta(nK)$.

Esercizio 2.
Sia $S = \{1, \ldots, 1000\}$ l'insieme dei primi 1000 interi positivi, e sia I la famiglia di sottoinsiemi di S tale che $A \subseteq S$ appartiene a I se e solo se $|A| \leq 5$ (dove $|A|$ denota la cardinalità di A).

- La coppia (S, I) è un sistema di indipendenza?
- La coppia (S, I) è un matroide?

Per ciascuna delle due domande dare una dimostrazione se la risposta è affermativa, oppure fornire un controesempio se la risposta è negativa.

Soluzione. La coppia (S, I) proposta è un matroide. Infatti, valgono le seguenti proprietà.

- S è un insieme finito e non vuoto.
- I è un insieme finito (contiene al più tutti i sottoinsiemi di S) e non vuoto (ad esempio, $\emptyset \in I$).

- Vale la **proprietà della ereditarietà**: dati $A \in I$ e $B \subseteq A$, vale $B \in I$.

  Infatti $A \in I$ significa che $|A| \leq 5$. Se $B \subseteq A$ allora $|B| \leq |A|$, da cui $|B| \leq 5$, e quindi $B \in I$.

- Vale la **proprietà dello scambio**: dati $A, B \in I$ tali che $|A| < |B|$, $\exists\, x \in B \setminus A$ tale che $A \cup \{x\} \in I$.

  Infatti $B \in I$ significa che $|B| \leq 5$. Poiché $|A| < |B|$, vale $|A| < 5$, e inoltre esiste sicuramente un numero x che appartiene a B ma non appartiene ad A. Allora $|A \cup \{x\}| = |A| + 1 < 5 + 1$, da cui $|A \cup \{x\}| < 6$, ovvero $|A \cup \{x\}| \leq 5$, e quindi $A \cup \{x\} \in I$.

Si osservi che gli elementi linearmente indipendenti massimali di I sono i sottoinsiemi di S contenenti *esattamente* 5 elementi.

Esercizio 3.

Scrivere un algoritmo efficiente che, dato un grafo $G = (V, E)$ non orientato (rappresentato in memoria tramite liste di adiacenza), prende in ingresso un vertice $v \in V$ e stampa l'elenco dei vertici che hanno distanza pari da v.

Soluzione. L'idea è quella di eseguire una visita in ampiezza del grafo (algoritmo BFS a pag. 451 del libro di testo) utilizzando il vertice v dato in ingresso come sorgente, e di utilizzare l'informazione contenuta nel vettore delle distanze d alla fine della visita.

Assumiamo che sia $V = \{1, 2, \ldots, n\}$, ovvero che i vertici di V siano numerati da 1 a $n = |V|$. Ricordiamo che il vettore d ha n componenti, e che per ogni vertice $u \in V$ l'elemento $d[u]$ contiene la distanza di u dalla sorgente. Poiché la sorgente è v, avremo $d[v] = 0$. Inoltre, se u non è raggiungibile da v (ovvero u e v appartengono a due componenti connesse differenti) avremo $d[u] = \infty$.

Pertanto, l'algoritmo che risolve il problema proposto è il seguente:

```
PRINT–EVEN-DISTANCE(G)
BFS(G, v)
for u ← 1 to n
    do if d[u] < ∞ and d[u] mod 2 = 0
          then print u
```

Lo pseudo–codice dell'algoritmo BFS è esattamente quello di pag. 451 del libro di testo. Il tempo di esecuzione di BFS è $O(V + E)$, mentre la stampa dei vertici che hanno distanza pari dalla sorgente richiede una scansione del vettore d, che può essere fatta in $O(V)$ passi. Quindi il tempo di esecuzione totale dell'algoritmo che risolve il problema proposto è $O(V + E)$.