Algoritmi 2 (A.A. 2005/2006)
Esercitazione 4

Irene Finocchi [§]

1 M-cammino

Sia data una matrice $M = [m_{i,j}]$ di dimensione $n \times n$ contenente interi positivi. Un M-cammino è una sequenza $(m_{i_1,1}, m_{i_2,2}, ...m_{i_n,n})$ tale che $i_{k+1} \in \{i_k - 1, i_k, i_k + 1\}$. Il costo di un M-cammino è dato dalla somma dei valori degli elementi che lo compongono. Descrivere un algoritmo che trova un M-cammino di costo minimo in tempo $O(n^2)$.

Risolviamo prima il problema di calcolare il costo μ dell'M-cammino ottimo (ovvero, avente costo minimo). Mostreremo in seguito come ricostruire un M-cammino ottimo a partire dall'informazione sul valore di μ.

Il sottoproblema che risolveremo è il seguente: per ogni i, j tali che $1 \leq i, j \leq n$, calcolare il costo dell'M-cammino ottimo che termina nella cella $M[i, j]$. Manterremo le informazioni su tali sottoproblemi in una matrice bidimensionale C di costi:

$$C[i, j] \ = \ \text{costo minimo di un M-cammino che termina nella cella } M[i, j]$$

Una volta costruita tutta la matrice C, μ può essere ottenuto come il valore minimo nella sua ultima colonna:

$$\mu = \min_{1 \leq i \leq n} C[i, n]$$

Per come è definito il problema dell'M-cammino, è infatti possibile "uscire" dalla matrice M attraverso una qualunque cella dell'ultima colonna.

E' facile convincersi che $C[i, 1] = m_{i,1}$ per ogni $i \in [1, n]$. Inoltre, osserviamo che la cella $M[i, j]$ può essere raggiunta solo in tre possibili modi, ovvero passando per $M[i - 1, j - 1]$ (se esiste), per $M[i, j - 1]$, o per $M[i + 1, j - 1]$ (se esiste). Quindi il costo minimo di un M-cammino per raggiungere $M[i, j]$ può essere ottenuto come segue:

$$C[i, j] = m_{i,j} + \min\{C[i - 1, j - 1], C[i, j - 1], C[i + 1, j - 1]\}$$

Per semplicità, assumiamo che C sia provvista di una riga 0 ed una riga $n + 1$ tali che $C[0, j] = C[n+1, j] = +\infty$ per ogni $j \in [1, n]$: ciò garantisce che $C[i-1, j-1]$ e $C[i+1, j-1]$

[§]Dipartimento di Informatica, Università degli Studi di Roma "La Sapienza", Via Salaria 113, 00198 Rome, Italy. E-mail: {finocchi}@di.uniroma1.it.

```
    Algoritmo CalcolaCosto
1.  Input Una matrice M di dimensione n × n contenente interi positivi
2.  Output Il costo μ di un M-cammino di costo minimo in M
3.  begin
4.      for i = 1 to n do
5.          C[i, 1] ← M[i, 1]
6.          C[0, i] ← +∞
7.          C[n + 1, i] ← +∞
8.      for j = 2 to n do
9.          for i = 1 to n do
10.             C[i, j] ← M[i, j] + min{C[i − 1, j − 1], C[i, j − 1], C[i + 1, j − 1]}
11.     return min_{1≤i≤n} C[i, n]
12. end
```

Figure 1: Algoritmo per il calcolo del costo minimo di un M-cammino.

siano sempre ben definite (anche nel caso in cui $i = 1$ oppure $i = n$). Otteniamo quindi la seguente ricorrenza:

$$C[i,j] = \begin{cases} m_{i,1} & \textit{se } \text{j=1} \\ m_{i,j} + \min\{C[i-1,j-1], C[i,j-1], C[i+1,j-1]\} & \textit{altrimenti} \end{cases}$$

La matrice C, e quindi il valore μ, possono pertanto essere calcolati in tempo $O(n^2)$ come mostrato dallo pseudocodice in Figura 1.

Per calcolare un M-cammino di costo minimo possiamo sfruttare le informazioni contenute nella matrice C. Sia i tale che $C[i, n] = \mu$. Sia $i' \in \{i - 1, i, i + 1\}$ tale che $C[i', n - 1] = C[i, n] - m_{i,n}$. Costruiamo ricorsivamente l'M-cammino di costo minimo per raggiungere la cella $M[i', n - 1]$ e restituiamo la concatenazione di tale cammino con $m_{i,n}$.

2 Furto

Una banda di tre ladri deve spartirsi il frutto di una rapina di n oggetti $\{a_1, ..., a_n\}$ ciascuno caratterizzato da un proprio valore intero positivo v_i. Sapendo che l'ammontare totale del bottino M è divisibile per tre, descrivere un algoritmo che verifichi se è possibile spartire gli n oggetti in parti di ugual valore e, in caso affermativo, produca la partizione. L'algoritmo deve avere tempo di esecuzione $O(n \cdot M^2)$.

Consideriamo innanzitutto il problema di verificare se è possibile spartire gli n oggetti in tre parti ciascuna di valore $M/3$. Osserviamo che, se i ladri sono professionisti, nessun oggetto sarà scartato né tanto meno restituito al legittimo proprietario! La parte assegnata al terzo ladro sarà quindi univocamente determinata dalle parti assegnate al primo e al secondo. Sarà pertanto sufficiente verificare se sia possibile spartire gli oggetti in modo tale che i bottini del primo e del secondo ladro siano entrambi pari a $M/3$.

Per trovare una soluzione a questo problema, risolveremo il seguente sottoproblema $\mathcal{P}(i, j, k)$:

$\mathcal{P}(i,j,k)$: esiste una partizione degli oggetti $\{a_1,...,a_k\}$ che assegna al primo ladro un valore i e al secondo ladro un valore j?

Tali sottoproblemi sono definiti per k tale che $0 \leq k \leq n$ e per i,j tali che $0 \leq i,j \leq M/3$. Memorizzeremo le soluzioni per i sottoproblemi $\mathcal{P}(i,j,k)$ in una matrice Booleana tridimensionale P, di dimensioni $[0,M/3] \times [0,M/3] \times [0,n]$. La soluzione del problema originario sarà in $P[M/3,M/3,n]$.

E' facile verificare che $P[0,0,0] = true$ e che $P[i,j,0] = false$ se $i \neq 0$ o $j \neq 0$. Questi casi rappresentano le condizioni al contorno. Per calcolare il valore $P[i,j,k]$ a partire dai valori $P[\_,\_,k-1]$ consideriamo i tre casi in cui l'oggetto a_k venga assegnato al primo, al secondo o al terzo ladro:

- Assegnando l'oggetto a_k al terzo ladro, $P[i,j,k]$ risulterebbe vera se e solo se $P[i,j,k-1] = true$.
- Assegnando l'oggetto a_k al primo ladro, $P[i,j,k]$ risulterebbe vera se e solo se $P[i-v_k,j,k-1] = true$. Osserviamo che deve essere $i \geq v_k$ affinché $P[i-v_k,j,k-1]$ sia propriamente definita.
- Assegnando l'oggetto a_k al secondo ladro, $P[i,j,k]$ risulterebbe vera se e solo se $P[i,j-v_k,k-1] = true$. Osserviamo che deve essere $j \geq v_k$ affinché $P[i,j-v_k,k-1]$ sia propriamente definita.

La ricorrenza che descrive i passi base e il passo di avanzamento nella costruzione della matrice P è pertanto:

$$P[i,j,k] = \begin{cases} true & se\ i=j=k=0 \\ false & se\ k=0\ e\ i,j \neq 0 \\ P[i,j,k-1]\ OR\ P[i-v_k,j,k-1]\ OR\ P[i,j-v_k,k-1] & se\ i \geq v_k\ e\ j \geq v_k \\ P[i,j,k-1]\ OR\ P[i-v_k,j,k-1] & se\ i \geq v_k\ e\ j < v_k \\ P[i,j,k-1]\ OR\ P[i,j-v_k,k-1] & se\ i < v_k\ e\ j \geq v_k \\ P[i,j,k-1] & se\ i < v_k\ e\ j < v_k \end{cases}$$

La matrice P può essere calcolata in tempo $O(n \cdot M^2)$ come mostrato dallo pseudocodice in Figura 2.

Nel caso in cui $P[M/3,M/3,n] = true$, vogliamo calcolare la partizione vera e propria degli oggetti. Per fare ciò, usiamo lo pseudocodice in Figura 3.

```
    Algoritmo VerificaSpartizione
1.  Input n oggetti {a_1, ..., a_n} con valori positivi {v_1, ..., v_n}, valore totale M
2.  Output Valore Booleano
3.  begin
4.      for i = 0 to M/3 do
5.          for j = 0 to M/3 do
6.              P[i, j, 0] ← false
7.      P[0, 0, 0] ← true
8.      for k = 1 to n do
9.          for i = 0 to M/3 do
10.             for j = 0 to M/3 do
11.                 case:
12.                     i ≥ v_k e j ≥ v_k:
13.                         P[i, j, k] ← P[i, j, k − 1] OR P[i − v_k, j, k − 1] OR P[i, j − v_k, k − 1]
14.                     i ≥ v_k e j < v_k:
15.                         P[i, j, k] ← P[i, j, k − 1] OR P[i − v_k, j, k − 1]
16.                     i < v_k e j ≥ v_k:
17.                         P[i, j, k] ← P[i, j, k − 1] OR P[i, j − v_k, k − 1]
18.                     i < v_k e j < v_k:
19.                         P[i, j, k] ← P[i, j, k − 1]
20.                 endcase
21.     return P[M/3, M/3, n]
22. end
```

Figure 2: Algoritmo per verificare se esiste una partizione in tre parti uguali.

```
    Algoritmo Spartisci
1.  Input matrice P
2.  Output partizione degli oggetti tra i tre ladri
3.  begin
4.      if P[M/3, M/3, n] = true
5.          i, j ← M/3
6.          for k = n downto 1 do
7.              if i ≥ v_k and P[i − v_k, j, k − 1] = true
8.                  assegna oggetto a_k al primo ladro
9.                  i ← i − v_k
10.             else if j ≥ v_k and P[i, j − v_k, k − 1] = true
11.                 assegna oggetto a_k al secondo ladro
12.                 j ← j − v_k
13.             else assegna oggetto a_k al terzo ladro
14.     return partizione degli oggetti
15. end
```

Figure 3: Algoritmo per calcolare la partizione degli oggetti.