Algoritmi e Strutture Dati 2

Esercizi

Francesco Pasquale

12 ottobre 2020

Esercizio 1. Si consideri il seguente algoritmo *greedy* per il problema *chain matrix multiplication*

Algorithm 1 `Alg`$(M_1, \dots, M_n)$

INPUT: Matrici $M_1, \dots, M_n$, dove M_i è una matrice $a_{i-1} \times a_i$, con $a_0, a_1, \dots, a_n \in \mathbb{N}$
OUTPUT: La matrice prodotto $M = M_1 \times \cdots \times M_n$

  if $n = 2$ **then**
    return $M_1 \times M_2$
  $k = \arg\min\{a_{i-1}a_ia_{i+1} \,:\, i = 1, \dots, n-1\}$
  $P = M_k \times M_{k+1}$
  return `Alg`$(M_1, \dots, M_{k-1}, P, M_{k+2}, \dots, M_n)$

---

1. Fornire un controesempio che mostri come il numero totale di moltiplicazioni numeriche eseguite dall'algoritmo per calcolare $M_1 \times \cdots \times M_n$ non è ottimo.
2. Implementare `Alg` e l'algoritmo di programmazione dinamica visto a lezione (si veda il Cap. 6.5 in [1]) in un linguaggio di programmazione a piacere.
3. Scrivere un programma che prenda in input n numeri $a_0, a_1, \dots, a_n \in \mathbb{N}$ e calcoli quante moltiplicazioni farebbe l'algoritmo greedy *Alg* e quante ne farebbe l'algoritmo di programmazione dinamica, con input matrici M_i di dimensioni $a_{i-1} \times a_i$, per $i = 1, \dots, n$.
4. Eseguire più volte il programma del punto precedente con numeri $a_0, a_1, \dots, a_n \in \mathbb{N}$ scelti a caso e valutare in media quante operazioni si risparmiano con l'algoritmo di programmazione dinamica rispetto all'algoritmo greedy.

Esercizio 2. Avete n monete non bilanciate: se lanciate la moneta i-esima ottenete `Testa` con probabilità $p_i \in [0,1]$ e `Croce` con probabilità $1 - p_i$. Volete calcolare la probabilità $P(n,k)$ che escano esattamente k teste quando lanciate tutte le n monete.
Per esempio, se $p_1 = 1/2$, $p_2 = 1/3$, $p_3 = 1/4$, allora

$$P(3,0) = (1-p_1)(1-p_2)(1-p_3) = \frac{1}{2} \cdot \frac{2}{3} \cdot \frac{3}{4} = \frac{1}{4},$$

$$\begin{aligned} P(3,1) &= p_1(1-p_2)(1-p_3) + (1-p_1)p_2(1-p_3) + (1-p_1)(1-p_2)p_3 \\ &= \frac{1}{2} \cdot \frac{2}{3} \cdot \frac{3}{4} + \frac{1}{2} \cdot \frac{1}{3} \cdot \frac{3}{4} + \frac{1}{2} \cdot \frac{2}{3} \cdot \frac{1}{4} = \frac{11}{24}\,. \end{aligned}$$

Osservate che in generale $P(n,k)$ coinvolge una somma di $\binom{n}{k}$ termini, ognuno dei quali è un prodotto di n numeri. Quindi un algoritmo ingenuo per calcolare $P(n,k)$ farebbe $n\binom{n}{k}$ moltiplicazioni e $\binom{n}{k}$ somme.

Progettare un algoritmo che prende in input n valori $p_1, \ldots, p_n \in [0,1]$ e un intero non negativo $k \leqslant n$ e restituisce $P(n,k)$ facendo soltanto $\mathcal{O}(nk)$ operazioni.

Esercizio 3. Stiamo pianificando un viaggio dalla località A alla località B. Idealmente ci piacerebbe fare delle tappe da 200 km al giorno, ma questo potrebbe non essere possibile perché i punti dove possiamo sostare per la notte sono solo nelle posizioni $a_1 < a_2 < \cdots < a_n$, dove $a_i \in \mathbb{N}$ indica la distanza in *km* dell'i-esimo punto di sosta dalla località di origine A.

Per esempio, supponiamo che B è a 600 km da A. Se $a_1 = 100$, $a_2 = 200$, $a_3 = 400$ allora possiamo sostare in a_2 e a_3 e fare esattamente tre tappe da 200 km al giorno. Ma se $a_1 = 150$, $a_2 = 300$, $a_3 = 450$ allora non possiamo fare tappe da 200 km al giorno e dobbiamo scegliere fra varie opzioni: per esempio, potremmo (1) fare due soste, in a_1 e a_3, oppure (2) fare tutte e tre le soste in a_1, a_2 e a_3, o una delle altre opzioni. Per quantificare il nostro grado di soddisfazione rispetto alle varie opzioni supponiamo di usare questa misura: se in un giorno facciamo x km, allora paghiamo un *costo* pari a $(x-200)^2$. La soluzione (1) quindi, in cui faremmo tre tappe, di cui due da 150 km e una da 300 km, avrebbe un costo totale di $2(150-200)^2 + (300-200)^2 = 15$mila; la soluzione (2) invece, in cui faremmo quattro tappe da 150 km l'una, avrebbe un costo totale di $4(150-200)^2 = 10$mila.

Progettare un algoritmo efficiente che prende in input i numeri $a_1, a_2, \ldots, a_n$ (per semplicità, supponiamo che l'ultimo punto di sosta a_n è la destinazione B) e restituisce i punti di sosta in cui fermarsi per minimizzare il costo totale.

Esercizio 4. Una sequenza ordinata di simboli di un alfabeto Σ si dice *palindroma* se resta invariata leggendola da destra verso sinistra. Più formalmente, $x_1x_2\ldots x_n \in \Sigma^n$ è palindroma se $x_1x_2\ldots x_n = x_n\ldots x_2x_1$. Per esempio, `angelalavalalegna` è una sequenza palindroma. Una *sottosequenza* di una sequenza di simboli $x_1x_2\ldots x_n$, è una sequenza $x_{i_1}x_{i_2}\ldots x_{i_k}$ con $1 \leqslant i_1 < i_2 < \cdots < i_k \leqslant n$. Per esempio, `cba` è una sottosequenza della sequenza `abcbca`.

Progettare un algoritmo che prende in input una sequenza di simboli e restituisce una sottosequenza palindroma di lunghezza massima. Per esempio, se la sequenza in input è `abcbca` l'algoritmo deve restituire `abcba`.

Riferimenti bibliografici

[1] S. Dasgupta, C. Papadimitriou e U. Vazirani. *Algorithms.* McGraw-Hill, 2006.