Fondamenti di Programmazione 2

2 Dicembre 2022

Esercizio 1

Scrivere un programma che presi in input un intero $n > 0$, un intero k ($0 < k < n$), e una sequenza di interi $x_1, ..., x_n$ determini una sottosequenza di esattamaente k elementi con somma massima.

In particolare, il programma dovrà stampare la somma della sottosequenza, gli elementi che ne fanno parte e gli indici in cui gli elementi appaiono nella sequenza in input.

Esempio Consideriamo la sequenza $9, 13, -50, 3, 2, 9$ e $k = 3$. In questo caso, la sottosequenza ottima è costituita dagli elementi $9, 13, 9$ che compaiono rispettivamente agli indici $0, 1, 5$ e hanno come somma 31.

Esercizio 2

Scrivere un programma che letto un array A di n elementi da input, calcoli gli indici $0 \leq i \leq j < n$ tali che la somma $S[i:j] = \sum_{k=i}^{k=j} A[k]$ risulti massima.

Esempio Consideriamo l'array $9, 13, -5, 3, 2$. In questo caso, gli indici ottimi sono $i = 0, j = 1$ in quanto $S[0:1] = A[0] + A[1] = 9 + 13 = 22$.

Consideriamo l'array $9, 60, -50, 60, -5, -10$. In questo caso, gli indici ottimi sono $i = 0, j = 3$, in quanto $S[0:3] = A[0]+A[1]+A[2]+A[3] = 9+60-50+60 = 79$.

Suggerimento: Per l'approccio dinamico, considerare la ricorrenza: $S[i] = max(S[i-1] + A[i], A[i])$ con $S[0] = A[0]$.

Esercizio 3

Un grafo si dice k-colorabile se è possibile associare ad ogni nodo un colore $c \in \{c_1, ..., c_k\}$ in modo tale che ogni arco del grafo abbia gli estremi colorati con colori diversi: non è possibile colorare gli archi u, v dello stesso colore se esiste un arco (u, v) nel grafo.

Scrivere un programma che preso in input un grafo non orientato restituisca una $(D+1)$-colorazione, dove D è il grado massimo nel grafo.

Questa colorazione esiste sempre per ogni grafo (teorema di Brook) e può essere calcolata utilizzando una strategia greedy, da non confondere con il problema di colorare un grafo con esattamente k colori - indipendentemente dal grafo in input - più difficile computazionalmente.

Esercizio 6

Scrivere una funzione che restituisca una sottosequenza *palindroma* di lunghezza massima di una stringa in input. Una stringa si dice palindroma se è uguale alla propria inversa, ad esempio la stringa `OTTO`.

Esempio Consideriamo la stringa `SCACCIAPENSIERI`. `CCC` è una sottosequenza palindroma, `CAPRI` è una sottosequenza non palindroma. `SACCAS`, `IENEI` sono esempi di sottosequenze palindrome, in questo caso di lunghezza massima.

Suggerimento: Per l'approccio dinamico, sia $S[i][j]$ la lunghezza della sottosequena palindroma di lunghezza massima nella sottostringa della stringa s che inizia all'indice i e finisce all'indice j. Per i, j generici, vale:

- $S[i][i] = 1$
- $S[i][j] = 0$ quando $i > j$
- $S[i][j] = 2 + S[i+1][j-1]$ quando $i < j, s[i] = s[j]$
- $S[i][j] = \max S[i+1][j], S[i][j-1]$ quando $i < j, s[i] \neq s[j]$

Esercizio 4

Sia M una matrice di interi positivi di dimensione $n \times m$. Ad ogni percorso dalla cella di coordinate $(0,0)$ alla cella di coordinate $(n-1, m-1)$ possiamo associare un costo, definito come la somma dei valori delle celle facenti parte del percorso. Siamo interessati a raggiungere la cella $(n-1, m-1)$ (in basso a destra) con il minor costo possibile, partendo da $(0,0)$ (in alto a sinitra).

Scrivere un programma che presa in input una matrice di interi calcoli una soluzione ottima.

Esempio Consideriamo la matrice di dimensioni 4×3:

```
10 30 90
50 30 90
10 90 90
10 10 10
```

Uno dei possibili percorsi per raggiungere $(3, 2)$ da $(0, 0)$ è →→↓↓↓, che corrisponde al costo di $10 + 30 + 90 + 90 + 90 + 10 = 320$.

Il percorso ottimo è invece ↓↓↓→→, con un costo di $10+50+10+10+10+10 = 80$.

Suggerimento: Per l'approccio dinamico, sia $S[i][j]$ il cammino a costo minimo per raggiungere $M[i][j]$. Per ogni j, $S[0][j] = M[0][0] + ... + M[0][j]$ e per ogni i $S[i][0] = M[0][0] + ... + M[i][0]$. Quando ci troviamo in i, j generico, $S[i][j] = M[i][j] + \min S[i-1][j], S[i][j-1]$.

Esercizio 5

Consideriamo un array di n interi A. Da una generica posizione i, possiamo fare un *salto* ad una qualsiasi posizione $i + 1, ..., i + A[i]$.

Scrivere una funzione che calcoli in quali posizioni è necessario saltare, partendo dalla posizione 0, per raggiungere la posizione $n - 1$ con il minor numero di salti.

Esempio Consideriamo l'array $3, 1, 1, 2, 0, 0$. Partendo dalla posizione 0, possiamo saltare in posizione $1, 2, 3$, poiché $A[0] = 3$:

Per raggiungere l'ultima cella dell'array, possiamo effettuare il salto `0 -> 3` e `3 -> 5`, raggiungendo la destinazione in 2 salti. Un'alternativa, non ottima, è effettuare i salti `0 -> 1 -> 2 -> 3 -> 5`.

```
Solzione ottima:
[3]  1   1   2   0   0
 3   1   1  [2]  0   0
 3   1   1   2   0  [0]

Un'altra soluzione,
non ottima:
[3]  1   1   2   0   0
 3  [1]  1   2   0   0
 3   1  [1]  2   0   0
 3   1   1  [2]  0   0
 3   1   1   2   0  [0]
```

Suggerimento: Per l'approccio dinamico, sia $S[i]$ il numero ottimo di salti per raggiungere n da i. $S[i] = 0$ quando $i \geq n$, mentre $S[i] = 1 + \min S[i + 1], S[i + 2], S[i + 3], ..., S[i + A[i]]$.