## Introduzione agli Algoritmi Esame Scritto a canali unificati con idee per la soluzione

docenti: T. CALAMONERI, A. MONTI Sapienza Università di Roma Giugno 2022

## Esercizio 1 (10 punti):

Per la soluzione di un certo problema disponiamo di un algoritmo iterativo con costo computazionale  $\Theta\left(n^{2}\right)$ . Ci viene proposto in alternativa un algoritmo ricorsivo il cui costo è catturato dalla seguente ricorrenza:

$$T(n) = a \cdot T\left(\frac{n}{4}\right) + \Theta(1) \text{ per } n \geq 4$$
 
$$T(n) = \Theta(1) \text{ altrimenti}$$

Dove a è una certa costante intera positiva con  $a \ge 2$ .

Determinare qual è il valore massimo che la costante intera a può avere perché l'algoritmo ricorsivo risulti asintoticamente più efficiente dell'algoritmo iterativo di cui disponiamo. **Motivare bene la vostra risposta.** 

## La risposta è 15.

Cominciamo col risolvere la ricorrenza. Applicando ad esempio il metodo principale abbiamo  $f(n) = \Theta(1)$  mentre  $n^{\log_4 a} \geq n^{\log_4 2} = n^{\frac{1}{2}}$  si ha quindi  $f(n) = O(n^{\log_4 a - \epsilon})$ . Siamo pertanto nel primo caso del metodo e la soluzione della ricorrenza è  $\Theta\left(n^{\log_4 a}\right)$ .

Ora se a=16 la ricorrenza ha soluzione  $\Theta\left(n^{\log_4 16}\right)=\Theta\left(n^2\right)$  quindi perché l'algoritmo ricorsivo abbia costo computazionale inferiore a quella dell'algoritmo iterativo deve aversi  $a\leq 15$ .

## Esercizio 2 (10 punti):

Sia A un array di n interi. Con la coppia ordinata (i, j),  $0 \le i \le j < n$ , rappresentiamo il suo sottoarray che parte dall'elemento in posizione i e termina con

l'elemento in posizione j, definiamo valore di un sottoarray come la somma dei suoi elementi.

Progettare un algoritmo che, dato un array A di interi positivi ed un intero positivo s, restituisce la coppia ordinata che rappresenta il sottoarray di A più a sinistra che ha valore s. Se un tale sottoarray non esiste, la funzione deve restituire None. L'algoritmo deve avere costo computazionale O(n).

Ad esempio, per A = [1, 3, 5, 2, 9, 3, 3, 1, 6]

- con s=7 l'algoritmo deve restituire la coppia (2,3) (ci sono infatti in A tre sottoarray con valore 7 le cui coppie nell'ordine da sinistra a destra sono (2,3), (5,7), (7,8)).
- ullet con s=21 l'algoritmo deve restituire None in quanto A non ha sottoarray con valore 21 .

Dell'algoritmo proposto:

- a) si dia la descrizione a parole;
- b) si scriva lo pseudocodice;
- c) si giustifichi il costo computazionale.
- a) Consideriamo i vari sottoarray di A utilizzando due indici i e j al primo e all'ultimo elemento del sottoarray rispettivamente e una variabile tot con il valore del sottoarray. Inizializziamo i due indici a 0 e di conseguenza tot a A[0]. Incrementiamo di volta in volta gli indici distinguendo 3 possibili casi:
  - il valore del sottoarray in esame è inferiore ad s, bisognerà in questo caso incrementare j aggiungendo un altro elemento al sottoarray. Il valore del nuovo array sarà tot + A[j].
  - il valore del sottoarray in esame è s. In questo caso abbiamo trovato il sottoarray e restituiamo i due indici i e j.
  - ullet il valore del sottoarray è superiore ad s. In questo caso dobbiamo incrementare i escludendo il primo elemento dal sottoarray. Il valore del nuovo array sarà tot-A[i-1]

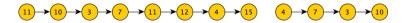
```
def es2(A,s):
    i=j=tot=0
    while j<len(A):
        print(i,j)
        tot+=A[j]
        while tot>s:
            tot-=A[i]
        i+=1
        if tot==s: return i,j
        j+=1
    return None
```

Ovviamente con le varie operazione di incremento di i e j dobbiamo far attenzione che si abbia sempre  $i \le j$  e j < n.

- b) di seguito una possibile implementazione dell'algoritmo in Python:
- c) il costo computazionale dell'algoritmo dipende dal numero di iterazioni dei while. Ad ogni iterazione del while più interno si incrementa l'indice i e ad ogni iterazione del while più esterno si incrementa l'indice j; questo significa che il numero di iterazioni è inferiore a 2n (infatti deve aversi sempre  $i \leq j < n$  con all'inizio i = j = 0). Il costo dell'algoritmo è dunque  $\Theta(n)$ .

Esercizio 3 (10 punti): Si consideri una lista concatenata dove ogni nodo ha 2 campi, il campo key contenente un intero ed il campo next con il puntatore al nodo seguente (next vale None per l'ultimo nodo della lista).

Bisogna aggiornare i puntatori della lista in modo da creare una nuova lista priva dei nodi con valore superiore a  $10\,\mathrm{e}$  in cui i nodi rimanenti appaiono in ordine inverso rispetto all'originale. Ad esempio per la lista di seguito a sinistra la funzione deve restituire la lista di seguito a destra:



Progettare un algoritmo che, dato il puntatore p alla testa della lista, risolve il problema in tempo  $\Theta(n)$  dove n è il numero di nodi della lista originaria. Lo spazio di lavoro dell'algoritmo proposto deve essere  $\Theta(1)$  (in altri termini non è possibile definire e utilizzare altre liste o nodi).

Dell'algoritmo proposto:

- a) si dia la descrizione a parole,
- b) si scriva lo pseudocodice,
- c) si giustifichi il costo computazionale.

a) Inizializzo un puntatore q ad una lista vuota questo sara' alla fine il puntatore alla nuova lista da restituire. Scorro la lista originaria e per ogni nodo che incontro, se la chiave del nodo è superiore a 10 semplicemente lo trascuro, in caso contrario lo aggancio in testa alla lista q.

```
b)
def es3(p):
   q = None
   while p:
      if p.key > 10:
        p = p.next
      else:
        t = p.next
        p.next = q
        q = p
        p = t
   return q
```