

# INTRODUZIONE AGLI ALGORITMI

## Esame Scritto a canali unificati

con idee per la soluzione

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

### Esercizio 1 (10 punti):

Si consideri la seguente funzione:

```
funzione Exam(n):  
  611 tot ← 1;  
  611 if n ≤ 1: return tot;  
  611 j ← 63;  
  while j > 0 do:  
    611 k ← 0; x=n → k=3n  
    while 3*k ≤ n do: k ← k+1;  
    tot ← tot + 2 * Exam(k); 9T(n/3)  
    611 j ← j - 7;  
    while k > 0 do: 6(n)  
      for i = 1 to n do: 6(n) = Θ(n²) tot ← tot - 1  
      k ← k - 1;  
  return tot
```

9 volte { T(n) = 9T(n/3) + Θ(n²)  
T(1) = Θ(1)

- Si imposti la relazione di ricorrenza che ne definisce il tempo di esecuzione giustificando dettagliatamente l'equazione ottenuta.
- Si risolva la ricorrenza usando il **metodo dell'albero** dettagliando i passaggi del calcolo e giustificando ogni affermazione.
- Il primo *while* della funzione viene iterato un numero costante di volte (esattamente 9) ed ogni iterazione costa  $\Theta(n)$  (il *while*

più interno richiede infatti  $\frac{n}{3} = \Theta(n)$  iterazioni).

La funzione viene ricorsivamente richiamata ad ogni iterazione del primo `while`, quindi esattamente 9 volte, ed ogni volta su un'istanza di dimensione  $k$  dopo l'uscita dal `while` interno, vale a dire  $\frac{n}{3}$ .

L'ultimo `while` della funzione viene iterato  $\Theta(n)$  volte (precisamente  $\lceil \frac{n}{3} \rceil$  volte) e ad ogni iterazione viene eseguito un `for` che richiede tempo  $\Theta(n)$ .

La ricorrenza è dunque  $T(n) = 9T(\frac{n}{3}) + \Theta(n^2)$  se  $n > 1$  e  $T(n) = \Theta(1)$  altrimenti.

- b) Qui bisogna rappresentare l'albero. Brevemente, in ogni livello  $i$  dell'albero tutti i  $9^i$  nodi danno un contributo di  $(\frac{n}{3^i})^2 = \frac{n^2}{9^i}$  per cui, sommando i contributi livello per livello, ogni livello  $i$  contribuisce per  $n^2$ . L'albero che si ottiene è completo ed ogni nodo interno ha 9 figli pertanto la sua altezza è logaritmica in  $n$ . Ne consegue che il costo totale è  $\Theta(n^2 \log n)$ .

**Esercizio 2 (10 punti):** Dato un array **ordinato**  $A$  di  $n$  interi ed un intero  $k$  vogliamo sapere quante coppie in  $A$  hanno somma  $k$ . Si progetti un algoritmo **iterativo** che risolva il problema in tempo  $\Theta(n)$ .

Ad esempio:

- se  $A = [1, 2, 2, 3, 4, 5, 5, 5, 8, 9, 9]$  e  $k = 7$  l'algoritmo deve restituire 7 (le coppie a somma 7 sono infatti  $(1, 5)$ ,  $(1, 6)$ ,  $(1, 7)$ ,  $(2, 5)$ ,  $(2, 6)$ ,  $(2, 7)$  e  $(3, 4)$ ).
- se  $A = [1, 5, 5, 5, 9]$  e  $k = 10$  l'algoritmo deve restituire 4 (le coppie a somma 10 sono infatti  $(0, 4)$ ,  $(1, 2)$ ,  $(1, 3)$ ,  $(2, 3)$ ).

Dell'algoritmo proposto:

- a) si dia la descrizione a parole,
  - b) si scriva lo pseudocodice,
  - c) si giustifichi il costo computazionale.
- a) Inizializziamo il contatore *count* delle coppie a 0 e due indici  $a$  e  $b$  al primo e all'ultimo elemento dell'array, rispettivamente.

Se la somma degli elementi puntati è superiore a  $k$  decremento l'indice  $b$ . Se la somma dell'elemento puntati è inferiore a  $k$  incremento l'indice  $a$ . Se la somma degli elementi puntati è  $k$  decremento  $b$  fino a raggiungere un elemento inferiore a quello puntato e incremento  $a$  fino a raggiungere un elemento maggiore di quello puntato. Sia  $x$  l'incremento subito da  $a$  e  $y$  il decremento subito da  $b$ . Se la coppia trovata aveva elementi uguali allora  $count$  viene incrementato di  $\binom{x+y}{2}$  altrimenti di  $x \cdot y$ . Mi fermo quando l'indice  $a$  raggiunge l'indice  $b$ .

b) di seguito l'algoritmo in Python

```
def es(A,k):
    a,b,count=0, len(A)-1,0
    while a<b:
        if A[a]+A[b]>k:b-=1
        if A[a]+A[b]<k:a+=1
        if A[a]+A[b]==k:
            x=1
            while b-1>a and A[b-1]==A[b]:
                b-=1;x+=1
            y=1
            while a+1<b and A[a+1]==A[a]:
                a+=1; y+=1
            if A[a]==A[b]: count+=(x+y)*(x+y-1)//2
            else: count+=x*y
            a=a+1
            b=b-1
    return count
```

c) il costo computazionale dell'algoritmo dipende dal numero di iterazioni dei *while*. Ad ogni iterazione dei *while* uno dei due puntatori  $a$  e  $b$  si avvicina all'altro; questo significa che dopo  $n$  iterazioni l'algoritmo termina. Il costo dell'algoritmo è dunque  $\Theta(n)$ .

### Esercizio 3 (10 punti):

Si consideri una lista a puntatori  $L$ , in cui ogni elemento è un record a tre campi: il campo `val` contenente un bit (cioè un valore 0 o 1), il campo `next` con il puntatore al nodo seguente (`next` vale `None` per l'ultimo record della lista) ed il campo `prec` con il puntatore al nodo precedente (`prec` vale `None` per il primo record della lista).

Bisogna verificare se la stringa che si ottiene considerando i bit dei vari nodi della lista è palindroma. Ad esempio, se la lista  $L$  in input è quella di sinistra nella figura che segue, la risposta è NO (la stringa binaria 010110 non

è palindroma, mentre se  $L$  è la lista di destra la risposta è SI (la stringa binaria 11011 è palindroma).



Progettare un algoritmo (iterativo o ricorsivo) che, dato il puntatore  $s$  alla testa della lista, risolva il problema in tempo  $\Theta(n)$ , dove  $n$  è il numero di nodi della lista a puntatori. Lo spazio di lavoro dell'algoritmo proposto deve essere  $O(1)$  (in altri termini NON è possibile definire e utilizzare altre liste).  
Dell'algoritmo proposto:

- a) si dia la descrizione a parole,
  - b) si scriva lo pseudocodice,
  - c) si giustifichi il costo computazionale.
- a) utilizzo un secondo puntatore  $r$  che parte dall'ultimo nodo della lista e si sposta verso sinistra mentre il puntatore  $s$  si sposterà verso destra. Per prima cosa, ricaviamo la posizione di partenza del puntatore  $r$  raggiungendo la fine della lista. A questo punto i due puntatori sono in posizione simmetrica nella lista. Se i due puntatori sono sullo stesso nodo allora termino restituendo 'SI' (la lista contiene una stringa di un unico simbolo); in caso contrario vengono confrontati i contenuti dei nodi puntati, se non sono uguali restituisco 'NO', se sono uguali ed  $s.next$  è uguale a  $r$  restituisco 'SI' (ho controllato tutti gli elementi simmetrici). Se non ho terminato puntatori si spostano allora al nodo successivo (che per il puntatore  $s$  è  $s.next$  mentre per il puntatore  $r$  è  $r.prec$ ) e l'operazione di controllo si ripete. Mi fermo dunque quando scopro simboli diversi in posizione simmetrica o i confronti terminano (questo si verifica quando i due puntatori sono sullo stesso nodo o su nodi adiacenti).
- b) `def palindrome(s):`  
`d = s`  
`while d.next != None : d = d.next`  
`while True:`  
`if s == r : return 'SI'`  
`if s.val != r.val : return 'NO'`

```
if s.next == r : return 'SI'  
s = s.next  
r = r.prec
```

- c) Bisogna eseguire il calcolo in modo formale. Brevemente, comunque, il primo while richiede di scorrere la lista e costa dunque  $\Theta(n)$ . Ad ogni iterazione del secondo while i due indici *r* ed *s* si avvicinano, questo significa che dopo al più  $n/2$  iterazioni anche il secondo while termina il suo lavoro. Il costo dell'algoritmo è dunque  $\Theta(n)$