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

docenti: T. Calamoneri, A. Monti
Sapienza Università di Roma
31 Gennaio 2021

Esercizio 1 (8 punti):
Si consideri la seguente funzione:

```
funzione Exam(n):
    tot ← n;
    if n <= 1:  return tot;
    j ← 512;
    while j >= 2 do:
        k ← 0;
        while 3 * k <= n do:   k ← k + 1;
        tot ← tot + Exam(k);
        j ← j/2;
    return tot
```

a) Si imposti la relazione di ricorrenza che ne definisce il tempo di esecuzione giustificando dettagliatamente l'equazione ottenuta.

b) Si risolva la ricorrenza usando il **metodo principale** (o un altro metodo, ricordando che $\sum_{i=1}^{k} 3^i = \Theta(3^k)$) dettagliando i passaggi del calcolo e giustificando ogni affermazione.

a) Il *while* più esterno 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 while più esterno, quindi esattamente 9 **volte ed ogni volta su un'istanza di dimensione** k **, vale a dire** $\frac{n}{3}$**.**
La ricorrenza è dunque $T(n) = 9T\left(\frac{n}{3}\right) + \Theta(n)$ **se** $n > 1$ **e** $T(n) = \Theta(1)$ **altrimenti.**

b) **Per applicare il metodo principale notiamo che** $n^{\log_b a} = n^{log_3 9} = n^2$ **mentre** $f(n) = \Theta(n) = O(n^2)$ **siamo dunque nel primo caso del teorema principale e si ha** $T(n) = \Theta(n^2)$**.**

Esercizio 2 (12 punti):

Sia A un array di dimensione n e B un array **ordinato** di dimensione m, contenenti entrambi numeri interi. Si vuole trovare il numero di interi di A che non sono presenti in B. Progettare un algoritmo ricorsivo che risolva il problema con un costo computazionale asintotico strettamente inferiore a $\Theta(nm)$.

Ad esempio: per $A = [8, 1, 2, 12, 10, 11, 20, 2]$ e $B = [3, 3, 4, 8, 10, 10, 13, 20, 21, 22]$ l'algoritmo deve restituire 5 (i numeri in A e non in B sono infatti $1, 2, 2, 11, 12$.

Dell'algoritmo proposto

a) si dia la descrizione a parole,

b) si scriva lo pseudocodice,

c) si giustifichi il costo computazionale.

a) **Inizializziamo un contatore** `cont` **a 0. Scorriamo il vettore** A **e, per ogni elemento** x **incontrato, verifichiamo con la ricerca binaria se in** B **è presente** x**. In caso affermativo incrementiamo un contatore, e comunque continuiamo a scandire gli elementi di** A**. Dopo aver esaminato tutti gli elementi di** A **restituiamo il valore del contatore.**

b)
```
def esercizio2(A, B):
    i, n = 0, len(A)
    cont = 0
    while i < n:
        x = A[i]
            if RIC-BIN(B, x)! = -1  cont+ = 1
    return cont
```

dove con RIC-BIN(B, y) indichiamo la funzione di ricerca binaria che cerca x nell'array B e ne restituisce la posizione se lo trova, -1 altrimenti.

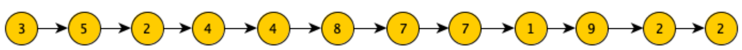
c) **il costo computazionale dell'algoritmo dipende dal numero di iterazioni del *while*, che è esattamente n. Il costo di ogni iterazione è dato da quello della ricerca binaria, $O(\log m)$. Il costo totale è dunque $O(n \log m)$.**

Alternativamente, si sarebbe potuto scegliere di ordinare l'array non ordinato A con un algoritmo efficiente, ed eseguire poi un algoritmo lineare che scorre con un indice ciascun vettore incrementando quello relativo all'elemento più piccolo. In tal caso, il costo sarebbe stato $\Theta(n \log n)$ più $O(n + m)$; in totale: se $m = O(n \log n)$ allora si ha $\Theta(n \log n)$ altrimenti $O(m)$. Poiché m potrebbe essere asintoticamente strettamente inferiore a $\log n$, questa soluzione non soddisfa la richiesta di avere un costo computazionale asintotico strettamente inferiore a $\Theta(nm)$.

Esercizio 3 (10 punti):

Si consideri una lista L, in cui ogni elemento è un record a due campi, il campo `val` contenente un intero ed il campo `next` con il puntatore al nodo seguente (`next` vale *None* per l'ultimo record della lista.

Bisogna contare i record della lista contenenti numeri pari. Si consideri ad esempio la lista L, per questa lista bisogna la risposta è 6



Progettare un **algoritmo ricorsivo** che, dato il puntatore r alla testa della lista effettui l'operazione di conteggio in tempo $\Theta(n)$ dove n è il numero di elementi presenti nella lista.

Dell'algoritmo proposto

a) si dia la descrizione a parole,

b) si scriva lo pseudocodice,

c) si giustifichi il costo computazionale risolvendo la ricorsione che viene fuori dall'algoritmo utilizzando uno dei metodi di soluzione visti a lezione.

a) **considero la testa della lista, se la lista è vuota restituisco** 0 **e termino. In caso contrario ricorsivamente conto i numeri pari presenti nella sottolista che parte dall'elemento successivo alla testa e restituisco semplicemente questo numero se la testa ha valore dispari, altrimenti restituisco questo numero incrementato di uno.**

b)
```
def conta_pari(r):
    if r == None : return 0
    if r.val%2 == 1:  return conta_pari(r.next)
    return conta_pari(r.next) +1
```

c) **la chiamata ricorsiva avviene su di una lista che ha un elemento in meno rispetto alla lista di partenza si ha dunque**
$T(n) = T(n-1) + \Theta(1)$ **se** $n > 0$**,** $T(0) = \Theta(1)$**.**
Posso risolvere la ricorsione usando il metodo iterativo.
Dopo k **iterazioni si ha** $T(n) = T(n-k) + \sum_{i=0}^{k} \Theta(1)$
e da questo deduciamo che dopo n **passi si ha:**
$T(n) = T(0) + \sum_{i=0}^{n} \Theta(1) = \Theta(n)$**.**

Notare che un algoritmo iterativo non soddisfa la richiesta del testo.