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

docenti: T. Calamoneri, A. Monti
Sapienza Università di Roma
13 Gennaio 2022

Esercizio 1 (10 punti):
Si consideri la seguente funzione:

```
funzione Exam(n):
    tot ← n;
    if n <= 1:  return tot;
    j ← 80;
    while j >= 3 do:
        tot ← tot + j;
        j ← j − 2;
    return tot+Exam(n − j)
```

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 dell'albero** dettagliando i passaggi del calcolo e giustificando ogni affermazione.

a) Il *while* della funzione viene iterato un numero costante di volte (esattamente 40). Al termine del while la j vale 2 e funzione viene quindi richiamata al più una volta e su un istanza di dimensione $n - 2$.
La ricorrenza è $T(n) = T(n-2) + \Theta(1)$ se $n > 1$ e $T(n) = \Theta(1)$ altrimenti.

b) **L'albero è in questo caso una catena infatti ha un solo nodo per livello inoltre, poiché il valore di n si decrementa di 2 ad ogni livello intermedio il numero di livelli della catena è $\lceil \frac{n}{2} \rceil = \Theta(n)$. In ogni livello dell'albero il nodo contribuisce per $\Theta(1)$ per cui sommando i contributi dei vari livelli abbiamo un costo totale $\Theta(n)$.**

Esercizio 2 (10 punti):

Abbiamo due array **ordinati** A e B di n interi distinti; si vuole sapere se esiste un valore x in A ed un valore y in B che differiscono al più 3 in valore assoluto (vale a dire $|x - y| \leq 3$).

Ad esempio:
per $A = [1, 2, 20, 30]$ e $B = [6, 7, 10]$ la risposta è negativa.
Per $A = [1, 2, 9, 10, 12]$ e $B = [6, 14, 16, 20]$ la risposta è positiva (grazie alla coppia $(9, 6)$ o anche $(12, 14)$).

Progettare un **algoritmo** che risolva il problema restituendo 1 se la risposta è positiva, 0 altrimenti. Il costo computazionale dell'algoritmo deve essere asintoticamente strettamente inferiore a $\Theta(n^2)$.

Dell'algoritmo proposto

a) si dia la descrizione a parole,

b) si scriva lo pseudocodice,

c) si giustifichi il costo computazionale.

a) **Entrambi gli array sono ordinati, quindi possiamo progettare il seguente algoritmo lineare:**
dati due indici, i che parte dal primo elemento di A e j che parte dal primo elemento di B, verifichiamo se $A[i]$ e $B[j]$ differiscono per meno di 3 ed, in tal caso, restituiamo 1 e terminiamo; altrimenti incrementiamo solo uno dei due indici, in modo che la differenza tra i due elementi vada a ridursi: se $A[i] < B[j]$ incrementiamo i, se invece $A[i] > B[j]$ incrementiamo j; iteriamo il ragionamento. Ovviamente il numero di iterazioni è al più $2n$ e quindi il costo computazionale è $O(n)$.

Notare che è meno efficiente l'algoritmo che scorre l'array A e, per ogni elemento x incontrato, verifica con una ricerca binaria

in B se è presente uno dei 7 valori $x-3, x-2, x-1, x, x+1, x+2, x+3$. Questo algoritmo ha infatti costo $O(n \log n)$.

Infine, non è accettabile l'algoritmo che per ogni elemento di A scorre l'intero array B, poiché questo ha costo quadratico, che non è ammesso dalla traccia (poco importa inserire delle euristiche che migliorino il caso migliore: il caso peggiore rimane quadratico).

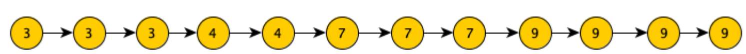
b)
```
def esercizio2(A, B):
    a = 0; b = 0
    while a < n and b < n:
        if |A[a] - B[b]| ≤ 3 :
            return 1
        if A[a] < B[b]:
            a++
        else b++
    return 0
```

c) Il costo computazionale dell'algoritmo dipende dal numero di iterazioni del *while*, che è dell'ordine di n (si dimostra in modo analogo a quanto fatto a lezione per la funzione di Fusione nel Merge Sort). Ogni iterazione ha un costo costante ed il costo complessivo risulta quindi essere $O(n)$.

Esercizio 3 (10 punti):

Si consideri una lista non vuota 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).

Gli interi nella lista sono ordinati in modo non decrescente e bisogna eliminare dalla lista i record contenenti duplicati. Si consideri ad esempio la lista L in figura; subito sotto viene riportato il risultato dell'operazione di cancellazione.



Progettare un **algoritmo iterativo** che, dato il puntatore r alla testa della lista effettui l'operazione di modifica in tempo $\Theta(n)$ dove n è il numero

di elementi presenti nella lista. Lo spazio di lavoro dell'algoritmo deve essere $O(1)$.
Dell'algoritmo proposto

a) si dia la descrizione a parole,

b) si scriva lo pseudocodice,

c) si giustifichi il costo computazionale.

d) si scriva lo pseudocodice di un algoritmo **ricorsivo** che risolve il problema

a) uso un indice che all'inizio punta al primo elemento della lista e, se questo è anche l'ultimo, l'algoritmo termina altrimenti, se l'elemento puntato contiene un valore uguale a quello dell'elemento seguente, quest'ultimo viene eliminato dalla lista aggiornando il puntatore altrimenti ci si sposta all'elemento successivo. In entrambi i casi si itera la procedura.

b)
```
def cancella_duplicati(r):
    p = r
    while p.next != None :
        if p.val != p.next.val:
            p = p.next
        else :
            p.next = p.next.next
```

c) Ad ogni iterazione del while o si avanza di un elemento nella lista o si cancella l'elemento che segue nella lista quindi dopo esattamente n iterazioni la procedura termina.

d)
```
def cancella_duplicati_ricorsivo(r):
    if r.next == None : return r
            p = cancella_duplicati_ricorsivo(r.next)
        if r.val == p.val:
            r.next = p.next
        else :
            r.next = p
    return r
```