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

docenti: T. Calamoneri, A. Monti
Sapienza Università di Roma
7 Giugno 2023

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

```
def Es1(n):
    if n < 5:
        return n
    s = k = n
    while k > 1:
        s+ = k
        k = k//3
    return s + Es1(n − 1)
```

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

b) Si risolva la ricorrenza utilizzando uno dei metodi studiati, dettagliando sia i passaggi matematici che quelli logici.

a) **Il caso base si ha quando $n < 5$, ed ha costo costante. La funzione ricorsiva su input $n \geq 5$ richiama se stessa 1 volta su un'istanza di dimensione $n - 1$. Il ciclo** *while* **viene eseguito** $\Theta(\log n)$ **volte (in quanto si parte con** $k = n$ **e ad ogni iterazione del while il valore di** k **diventa un terzo di** k**; la base non è rilevante nella notazione asintotica). Perciò, la relazione di ricorrenza da risolvere è:**

- $T(n) = T(n-1) + \Theta(\log n), \ \ n \geq 5$
- $T(n) = \Theta(1), \ \ n < 5.$

b) **Risolvendola otteniamo** $T(n) = \Theta(n \log n)$. **È necessario dettagliare tutti i passaggi per arrivare all'equazione di ricorrenza e per risolverla.**

Esercizio 2 (10 punti):

Dato un array A di n interi compresi tra 0 a 50, sapendo che nell'array sono certamente presenti dei duplicati, si vuole determinare la distanza massima tra le posizioni di due elementi duplicati in A.

Ad esempio per $A = [3, 3, 4, 6, 6, 3, 5, 5, 5, 6, 6, 9, 9, 1]$ i soli elementi che in A si ripetono sono 3, 6 e 9.

La distanza massima tra duplicati del 3 è 5,
la distanza massima tra duplicati del 6 è 7,
la distanza massima tra duplicati del 9 è 1.
quindi la risposta per l'array A è 7.

Progettare un algoritmo che, dato A, in tempo $\Theta(n)$ restituisca la distanza massima tra le posizioni con elementi duplicati. Dell'algoritmo proposto:

a) si scriva lo pseudocodice opportunamente commentato;

b) si giustifichi il costo computazionale.

a) **Osserviamo preliminarmente che la distanza massima tra le posizioni di due elementi uguali si verifica tra la prima e l'ultima occorrenza. Utilizziamo quindi un array** C **di** 51 **locazioni, le cui celle sono inizializzate**

tutte a -1; mentre scorriamo l'array A, in $C[x]$ inseriamo la prima occorrenza del valore x nell'array. Se in A incontriamo in posizione i l'intero x, se $C[x] = -1$ allora scriviamo i in $C[x]$, se invece $C[x]$ è un valore non negativo, possiamo calcolare in tempo $O(1)$ la distanza tra il primo elemento uguale ad x incontrato (che era in posizione $C[x]$) e l'elemento corrente; basterà infatti calcolare $i - C[x]$). Se in una variabile m teniamo traccia del massimo delle distanze via via calcolate, alla fine la risposta dell'algoritmo sarà proprio m.

Segue un possibile codice Python dell'idea proposta.

```
def es2(A):
    C = [-1] * 51
    m = 1
    for i in range (len(A)):
        if C[A[i]] == -1:
            C[A[i]] = i
        else:
            m = max(m, i - C[A[i]])
    return m
```

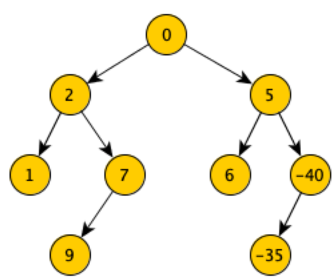
b) **Il costo dell'algoritmo è dato principalmente dal contributo del ciclo *for* che è $\Theta(n)$. È necessario dettagliare i contributi di ciascuna istruzione.**

È anche possibile implementare l'algoritmo utilizzando un dizionario senza modificare il costo computazionale nel caso peggiore $\Theta(n)$ (infatti, il costo di inserimento o ricerca in un dizionario nel caso peggiore è data dal numero di chiavi presenti nel dizionario che, nel nostro caso, è costante poiché il numero di elementi presenti nel dizionario è in ogni momento al più 51).

Segue il codice Python dell'implementazione tramite dizionario.

```
def es2(A):
    d = {}
    m = 1
    for i in range (len(A)):
        if A[i] in d:
            m = max(m, i - d[A[i]])
        else:
            d[A[i]] = i
    return m
```

Esercizio 3 (10 punti): Dato il puntatore r al nodo radice di un albero binario non vuoto, progettare un algoritmo *ricorsivo* che in tempo $\Theta(n)$ calcoli il numero di nodi che hanno esattamente 2 figli e chiave pari.
Ad esempio, per l'albero in figura, l'algoritmo deve restituire 2, per la presenza dei nodi con chiavi 2 e 0.
L'albero è memorizzato tramite puntatori e record di tre cam-



pi: il campo key contenente il valore ed i campi $left$ e $right$ con i puntatori al figlio sinistro e al figlio destro, rispettivamente (questi puntatori valgono $None$ in mancanza del figlio).
Dell'algoritmo proposto:

a) si scriva lo pseudocodice opportunamente commentato;

b) si giustifichi il costo computazionale.

NOTA BENE: nello pseudocodice dell'algoritmo ricorsivo **non** si deve far uso di variabili globali.

a) **Si utilizzi una visita dell'albero in postorder. Ogni nodo riceve dai suoi due figli il numero di nodi con le proprietà richieste presenti nei loro sottoalberi, somma questi valori, aggiunge eventualmente 1 nel caso sia esso stesso uno dei nodi da contare e restituisce al padre il valore del suo sottoalbero.**

Ecco un possibile codice Python dell'algoritmo che non utilizza variabili globali:

```
def es3(r):
    if r == None:
        return 0
    a = 0
    if r.key%2 == 0 and r.left! = None and r.right! = None:
        a+ = 1
    return a + es3(r.left) + es3(r.right)
```

b) **Il costo computazionale è quello della visita di un albero con n nodi. L'equazione di ricorrenza relativa alla visita è:**

- $T(n) = T(k) + T(n-1-k) + \Theta(1)$
- $T(0) = \Theta(1)$

dove k, è il numero di nodi presenti nel sottoalbero sinistro, $0 \leq k < n$. L'equazione si può risolvere con il metodo di sostituzione (che va esplicitamente svolto) dando come soluzione $\Theta(n)$.
Ogni passaggio va dettagliato.