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

docenti: T. Calamoneri, A. Monti
Sapienza Università di Roma
12 Settembre 2023

Esercizio 1 (10 punti):

Siano:

$$T(n) = 8T(n/2) + \Theta(n^2)$$

la funzione di costo di un algoritmo ricorsivo $\mathcal{A}$, e

$$T(n) = bT(n/4) + \Theta(n^2)$$

la funzione di costo di un altro algoritmo ricorsivo $\mathcal{A}'$, dove b è una costante intera positiva, e per entrambe le ricorrenze vale $T(1) = \Theta(1)$.

Qual è il minimo valore intero della costante b che rende $\mathcal{A}$ asintoticamente più veloce di $\mathcal{A}'$?

Dettagliare il ragionamento ed i passaggi del calcolo, giustificando ogni affermazione.

Applicando il teorema principale, $T(n) = O(n^3)$**.**
Il secondo algoritmo è asintoticamente meno efficiente solo se se $n^{\log_4 b} > n^3$**, vale a dire** $log_4 b > 3$ **cioè** $b > 4^3 = 64$**; quindi, il minimo intero per cui questo accade è** $b = 65$**.**
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, si scriva un algoritmo **iterativo** *MaxSequenzaElementiUguali* che calcoli il numero di elementi della più lunga porzione di A costituita interamente da elementi consecutivi uguali tra loro.

Ad esempio, se $A = [5, 7, 3, 3, 8, 9, 9, 9, 5, 3, 2, 2]$, allora la risposta è 3 in quanto la porzione $[9, 9, 9]$ è la più lunga formata da elementi consecutivi tutti uguali.

Dell'algoritmo proposto:

a) si scriva lo pseudocodice opportunamente commentato;

b) si calcoli formalmente il costo computazionale.

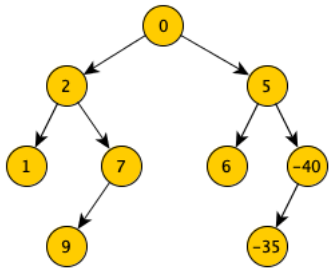
Basta scorrere l'array e tener traccia del numero massimo di ripetizioni trovate fino a quel momento, contando di volta in volta i numeri consecutivi uguali e ricominciando il conteggio ogni volta che si trova un numero diverso; l'unica cosa a cui stare attenti è di tener conto del numero di ripetizioni dell'ultima occorrenza (che non ha un successivo) e del numero di ripetizioni della prima occorrenza (che non ha elementi precedenti da controllare).
Segue un possibile codice Python, che nel compito va ben commentato e ne va dimostrato il costo computazionale.

```python
def MaxSequenzaElementiUguali(A):
    massimo=count=0
    for i in range(len(A)):
        if i==0 or A[i]==A[i-1]:
            count+=1
        else:
            massimo=max(massimo,count)
            count=1
    return max(massimo,count)
```

Esercizio 3 (10 punti):
Dato un albero binario non vuoto a valori interi T ed un suo nodo v, il *costo del cammino radice-v* è definito come la somma

dei valori dei nodi nel percorso che va dalla radice al nodo v (estremi inclusi).

Vogliamo calcolare il costo del massimo cammino radice-foglia di T.
Ad esempio nell'albero binario in figura, la risposta è 18, infatti nell'albero sono presenti quattro diversi cammini radice-foglia di costo 3, 18, 11 e -70, rispettivamente.



Dato il puntatore r al nodo radice di un albero binario non vuoto a valori interi T, progettare un algoritmo **ricorsivo** che, in tempo $\Theta(n)$, risolva il problema.
L'albero è memorizzato tramite puntatori e record a tre campi: 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 formalmente il costo computazionale.

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

Si utilizza una visita dell'albero. Ogni nodo terminata la sua visita restituisce il valore massimo del cammino radice foglia del suo sottoalbero.
Ecco un possibile codice Python dell'algoritmo che non utilizza variabili globali:

```
def es3(r):
    a = b = 0
    if r.left:
        a += es3(r.left)
    if r.right:
        b += es3(r.right)
    return max(a, b) + r.key
```

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 $\Theta(n)$.
Di conseguenza il costo dell'algoritmo è, come richiesto, $\Theta(n)$.
Nel compito, ogni passaggio va dettagliato.