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

docenti: T. Calamoneri, A. Monti
Sapienza Università di Roma
17 Gennaio 2024

Esercizio 1 (10 punti): Si risolva con il **metodo dell'albero** la seguente equazione di ricorrenza:

$$\begin{cases} T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + \Theta(n) \text{ se } n > 1 \\ T(1) = \Theta(1) \end{cases}$$

Si dettaglino i ragionamenti ed i passaggi del calcolo, giustificando ogni affermazione.

Si veda l'Esercizio 3 degli Esercizi ragionati.

Esercizio 2 (10 punti): Si scriva lo pseudocodice, opportunamente commentato, di una funzione **iterativa** che, preso in input un array A di interi non nulli (positivi e negativi), ne sposti gli elementi in modo che gli interi negativi precedano gli interi positivi e restituisca poi l'array così modificato.
Ad esempio per $A = [3, -5, -7, 1, -8]$ due possibili risposte corrette (ma ve ne sono altre) sono $A = [-8, -7, -5, 1, 3]$ o $A = [-5, -8, -7, 3, 1]$.
La funzione deve avere costo computazionale $O(n)$, dove n è il numero di elementi presenti nell'array. Il costo in termini di spazio oltre l'array A deve essere $\Theta(1)$ (in pratica non può far uso di array di appoggio).

Si motivi in dettaglio il costo computazionale dell'algoritmo proposto. Si risponda, inoltre, alle seguenti domande:

a) qual è il costo computazionale se l'array in input è ordinato in modo non decrescente?

b) qual è il costo computazionale se l'array in input è ordinato in modo non crescente?

c) qual è il costo computazionale se l'array in input è costituito da elementi tutti uguali?

E' sufficiente modificare la funzione Partiziona del Quicksort, in cui il pivot non sia un elemento dell'array, ma 0. Il codice python di una possibile soluzione dell'esercizio è il seguente:

```
def esercizio2(A):
  i,j = 0, len(A)-1
  while i<j:
      if A[i] > A[j]:
          A[i], A[j] = A[j], A[i]
          i+=1
          j-=1
      else:
          if A[i]<0: i+=1
          if A[i]>0: j-=1
  return A
```

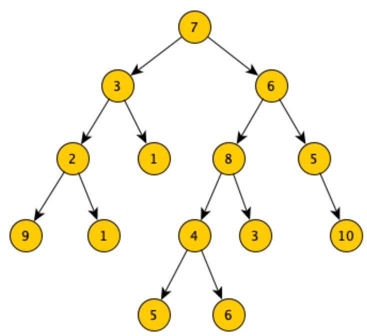
Usiamo due indici i e j per scorrere l'array, il primo si sposta da sinistra verso destra ed il secondo da destra verso sinistra e la procedura termina quando si incontrano. Il primo va alla ricerca del primo elemento positivo ed il secondo si sposta alla ricerca del primo elemento negativo se $i < j$ allora avviene lo scambio. Quando i due indici si incontrano alla loro sinistra ci sono solo elementi negativi ad ella loro destra solo elementi positivi.

**Per quanto riguarda il costo computazionale:
ad ogni iterazione del *while* almeno uno degli indici si sposta nella direzione dell'altro (se avviene lo scambio si spostano entrambi) questo significa che il numero di iterazioni (che è il numero di passi richiesti perchè i due indici si incontrino) è dell'ordine della loro distanza iniziale che**

è n. Questo significa che la procedura termina in $O(n)$ passi. Nota che servono Ω(n) passi prima che i due indici si incontrino e la procedura termini, questo significa che la risposta a tutte e tre domande finali è: $\Theta(n)$.

Esercizio 3 (10 punti): Si consideri un albero binario radicato T, i cui nodi abbiano un campo *val* contenente un intero e i campi *left* e *right* con i puntatori ai figli. Un nodo v dell'albero si dice *valido* se si verificano le seguenti tre condizioni:
- v ha entrambi i figli;
- il campo *val* di uno dei due figli è minore di quello padre;
- il campo *val* dell'altro figlio è maggiore di quello del padre.

Ad esempio nell' albero in figura ci sono esattamente 2 nodi validi (quelli con valore 6 e 2).



Si progetti una **funzione ricorsiva** che, dato il puntatore r alla radice di T restituisca il numero di nodi validi di T in tempo $\mathcal{O}(n)$ dove n è il numero di nodi presenti nell'albero.
Della funzione proposta:

a) si dia la descrizione a parole,
b) si scriva lo pseudocodice,
c) si giustifichi formalmente il costo computazionale.

NOTA BENE: nello pseudocodice della funzione ricorsiva **non** si deve far uso di variabili globali.

L'algoritmo prevede di visitare l'albero. Nel caso di albero vuoto viene restituito il valore 0, altrimenti nel generico

nodo r la funzione viene richiamata ricorsivamente sui sottoalberi di r in modo da ricavare un valore c che rappresenta la somma dei nodi validi presenti nel suo sottoalbero di sinistra e nel suo sottoalbero di destra. Il valore di c viene poi incrementato di 1 nel caso in cui il nodo r stesso sia un nodo valido. Il conteggio finale così ottenuto viene infine restituito al nodo padre.
Il codice python della procedura appena descritta è il seguente:

```
def es3(r):
    if r == None:
        return 0
    c= es3(r.left) + es3(r.right)
    if r.left and r.right and (r.left.val <r.val < r.right.val or r.right.val < r.val < r.left.val
        c+=1
    return c
```

Il costo computazionale è quello della visita, e la dimostrazione con il metodo di sostituzione va dettagliata.