Algoritmi e Strutture Dati
&
Laboratorio di Algoritmi e Programmazione

— Appello del 4 Settembre 2007 —

Esercizio 1 (ASD)

Descrivete a parole le proprietà espresse dalle seguenti equazioni e per ciascuna dite se è sempre vera oppure se vale solo sotto particolari condizioni.

- $f(n) + O(f(n)) = \Theta(f(n))$
- $f(n) + \Omega(f(n)) = \Theta(f(n))$

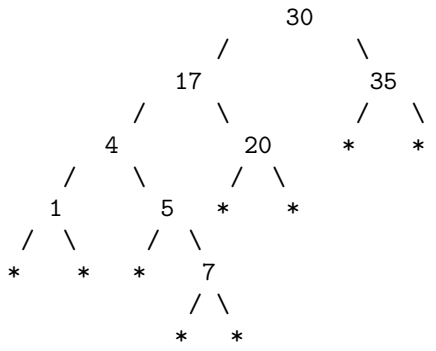
Esercizio 2 (ASD)

Risolvete le seguenti ricorrenze, giustificando la risposta.

- $T(n) = 2T(\frac{n}{3}) + 2n \log n + 3n$
- $T(n) = T(\frac{n}{2}) + T(\sqrt{n}) + n$

Esercizio 3 (ASD)

- Dite se il seguente albero binario di ricerca può essere colorato in modo da diventare un albero R/B. Giustificare la risposta. (Il simbolo * rappresenta la foglia-NIL.)



- Disegnate l'albero che si ottiene applicando una rotazione destra alla radice dell'albero precedente.

Esercizio 4 (ASD)

Sia $A[1..n]$, $n \geq 1$ un array di interi che soddisfa la seguente proiprietà:

$$\begin{aligned} \exists m: \quad & 1 \leq m \leq n \\ & \forall i: \ 1 \leq i < m \implies A[i] < A[i+1] \\ & \forall i: \ m \leq i < n \implies A[i] > A[i+1] \end{aligned}$$

Sviluppate un algoritmo che trovi l'indice m in tempo $O(\log n)$ e dimostrarne la correttezza.

[continua sul retro]

Esercizio 5 (LAB)

Sia data la seguente interfaccia per alberi binari generalizzati, con struttura a nodi in cui ogni nodo ha un colore ed un riferimento alla lista dei figli. L'interfaccia `ColorTree` ha un metodo `root()` che restituisce la radice dell'albero e due metodi che, dato un nodo, restituiscono rispettivamente il colore del nodo ed un iteratore che permette di scorrere la lista dei sui figli.

```
interface ColorTree  {
      Node root();               // la radice dell'albero
  Color color(Node p);       // il colore del nodo p
      Iterator figli(Node p);    // lista di figli di p
}
```

Diciamo che `T:ColorTree` è un BV-albero se `T` ha tutti i nodi di colore Blue (`Color.BLUE`) o Verde (`Color.GREEN`) ogni nodo Blue in `T` ha tutti i suoi figli Blue.

Fornite una implementazione del metodo `count()` specificato qui di seguito. La vostra implementazione deve garantire una complessità $O(k)$ dove k è il numero dei nodi verdi dell'albero.

```
public static void count(Tree T) {
  // PRE: T != null e' un BV-albero
  // POST: il numero dei nodi verdi in T.
}
```

Esercizio 6 (ASD+LAB)

Dato un albero binario T e due nodi u e v in T, definiamo il *minimo antenato comune* di u e v il nodo a di T di altezza minima che è antenato dei due nodi. Data l'interfaccia

```
interface BinTree  {
      Node root();               // la radice dell'albero
      Node left(Node p);         // figlio sinistro di p nell'albero
      Node right(Node p);        // figlio destro di p nell'albero
  int key(Node p);           // la chiave associata a p
}
```

fornite l'implementazione del metodo `mac` specificato qui di seguito e determinate la complessità della vostra implementazione.

```
public static Node mac(BinTree T, Node u, Node v) {
  // PRE: T != null e' un albero binario di ricerca (BST)
  con chiavi tutte distinte.
  // POST: restituisce il minimo antenato comune di u e v
}
```

Nota bene: l'albero `T` è un BST e **non** ha puntatore al padre.