Data Structures - Assignment no. 2, 2008

Remarks:

- **Write both your name and your ID number very clearly on the top of the exercise. Write your exercises in pen, or in clearly visible pencil. Please write *very* clearly.**
- **Recall that 80% of the theoretical exercises must be submitted. The exercises can and must be worked on and submitted alone.**
- **Give correctness and complexity proofs for every algorithm you write.**
- **For every problem, find the most efficient algorithm. A non efficient algorithm will be considered as an incomplete answer.**
- **For every question where you are required to write pseudo-code, also explain your solution in words.**

1. The pre-order read of a binary tree is the sequence of elements that are printed by the following procedure:

```
PRE-ORDER(v)
IF (v = null) RETURN
ELSE
    PRINT v.key
    PRE-ORDER(v.left)
    PRE-ORDER(v.right)
    RETURN
```

The post-order read of a binary tree is the sequence of elements that are printed by the following procedure:

```
POST-ORDER(v)
IF (v = null) RETURN
ELSE
    POST-ORDER(v.left)
    POST-ORDER(v.right)
    PRINT v.key
    RETURN
```

The in-order read of a binary tree is the sequence of elements that are printed by the following procedure:

```
IN-ORDER(v)
IF (v = null) RETURN
ELSE
    IN-ORDER(v.left)
    PRINT v.key
    IN-ORDER(v.right)
    RETURN
```

(a) Give a tree with at least 3 nodes (all nodes must have different keys) such that both its in-order read and its pre-order read are the same, or prove that there is no such tree.

(b) Give a tree with at least 3 nodes (all nodes must have different keys) such that both its in-order read and its post-order read are the same, or prove that there is no such tree.

(c) Give a tree with at least 3 nodes (all nodes must have different keys) such that both its pre-order read and its post-order read are the same, or prove that there is no such tree.

2. In Figure 2 you are shown a Red-Black Tree, whose elements are stored at the leaves (the values in the inner nodes are "pivot" elements which are not really elements of the data structure, and whose purpose is to allow searching).

   (a) Explain how to implement the operation $Insert(x)$ and $Delete(x)$ on this kind of RB-tree in $O(\log n)$ time.

   (b) Insert the keys 20 and 26 to the tree, then delete key 19, then insert 21,24 and delete 17, and finally insert 14. Now draw the resulting tree.

3. Suppose that we start from an empty Red-Black tree, and that we perform n insert operations, where $n > 1$. Prove that we necessarily end up with a Red-Black tree which has at least one red vertex. Assume that we are dealing with a Red-Black tree that stores its elements at the leaves.
(Hint: One way to prove this is: (1) prove that the second insert operation creates a red node; (2) then prove that for any insert operation, if there is a red node before applying it, then there is also a red node after applying it. Note that there may be other ways to prove the claim.)

4. Describe a data structure that implements a dictionary ADT. (The dictionary ADT maintains a set of keys, S, and supports the operations $insert(x)$, $delete(x)$ and $find(x)$). Let n be the number of operations performed on the data structure since it was created. The data structure should implement insert and delete in time $O(1)$ worst-case, and find in time $O(n \log n)$ worst case. Also, the amortized complexity of all operations should be $O(\log n)$. (In other words, the worst-case time of performing n operations should be $O(n \log n)$). Describe the data structure (no need to give pseudocode), and prove your claims about the running time.

5. Define the vertex-depth of a tree to be the distance between its root and the furthest leaf, measured in vertices, not in edges. For example, the depth of a tree which contains a single vertex is 1. The depth of an empty tree is 0.

   A binary search tree is called a *valid AVL tree* if for each node v the following condition holds: Let v_1, v_2 be v's children. Then we require that the difference between the depth of the subtree whose root is v_1 and the depth of the subtree whose root is v_2 is -1, 0, or $+1$. For example, Figure 1 depicts a valid AVL tree (the keys are not listed).

   (a) Prove that a valid AVL tree of depth d always has at least F_d vertices, where F_d is the d'th Fibonacci number. ($F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$). (Hint: use induction on d).

   (b) Let T be a valid AVL tree with n vertices. Prove that the depth of T is $O(\log n)$. You may use item (a).

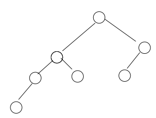

Figure 1: A valid AVL Tree

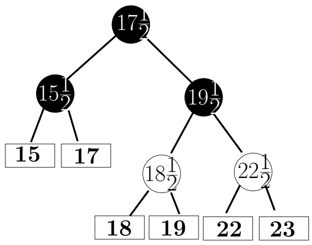

Figure 2: A red-black tree with elements at the leaves. (the white nodes are considered red. Recall that all leaves are considered black).