Assignment #4: ADTs, Trees, and Recursion

This assignment is to be done individually and submitted (using the drop box) before 8:30AM on Wednesday, November 07. You must attach the appropriate cover page for this assignment.

Question 1: Traversals

a. **[4 marks]** Give the preorder, postorder, inorder and levelorder (breadth first traversal) listings of the elements in the tree of Figure 2. (Ignore the shape of nodes.)

b. **[2 marks]** Construct the single binary tree that corresponds to the following traversals:
   - **inorder**: `B F G H P R S T W Y Z`
   - **postorder**: `B G H F R W T Z Y S P`

c. **[3 mark]** Construct all trees which have the following preorder traversal: `A B C D`.

d. **[2 mark]** Prove by counter-example that a given pair of preorder and postorder tree traversals does not specify a unique tree.

Question 2: Recursion and Deques

[8 marks] Assume the Deque ADT includes the following operators: `addToFront(), addToRear(), peekFront(), peekRear(), contains(), removeFromFront(), and removeFromRear(), and size().`

Two deques are reverse if they have the same values in the reverse ordering.

Consider the following four deques:

- **a**: `A B C D E F G H`
- **b**: `H G F E D C B A`
- **c**: `H G F E D C B`
- **d:** `H G F D E C B A`

For example, deque **a** is a reverse of **b** but is not a reverse of **c** or **d**. Empty (and also equal) deques are reverse too!. **e** is a reverse of **f**, and **g** is a reverse of **h**

- **e**: `H H H H`
- **f:** `H H H H`
- **g:**
- **h:**

Develop the following **recursive** method that we could add to the Deque ADT to check if it is the reverse of a passed in deque. No marks will be given for an iterative solution.

```
public boolean dequeReverse(DequeInterface d2)
{  // pre: d2 is non-null, this and d2 do not have any null items,
   //      this is the same size as d2.
   // post: return true if this and d2 are reverse, otherwise return false.

   // your code here
}
```

Question 3: Balanced Strings

[6 marks] Write a Java method to check if an input string is balanced or not. The string may have letters and the following characters "(", ")", "{", "}", "[", "]", "<", and ">". This is an augmentation to balanced strings introduced in lecture notes (page 46). You can use all public methods of ADTs taught in the lecture.

Question 4: Recursion and Trees

[6 marks] Develop the following **recursive** method that we could add to the BinaryTreeInterface ADT which removes all **leaf nodes** that match the object passed in. By remove, we mean the node is replaced by an empty subtree. Again, we will use the passed in object's '`.equals`' method to determine equality.

```
public void removeMatchingLeaves( Object obj )
   {  // pre: obj is non-null.
      // post: all leaves in this containing obj are replaced
      // by empty subtrees

      // your code here.
   }
```

As an example, if we remove all the leaves containing 'A' in the following tree (on the left) we get the one on the right (Note that the empty trees are represented with little boxes in this picture just for better visualization.)

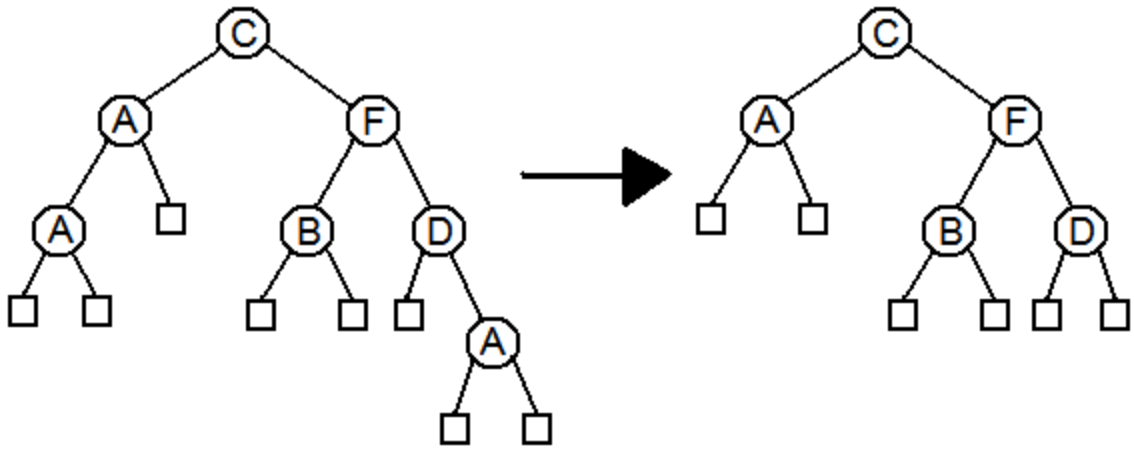

Figure 1

Question 5: Recursion and Trees (Again)

Consider a nonempty binary tree with two types of nodes: **min** nodes and **max** nodes. Each node has an integer value initially associated with it. You can define the **value** of such a **minimax** tree as follows:

- If the root is a min node, the value of the tree is equal to the *minimum* of
  - The integer stored in the root
  - The value of the left subtree, but only if it is nonempty
  - The value of the right subtree, but only if it is nonempty
- If the root is a max node, the value of the tree is equal to the *maximum the above three values.*

a. **[2 marks]** Compute the value of the following minimax tree (Figure 2). Each node is labeled with its initial value.

b. **[8 marks]** Write pseudocode for evaluating these trees.

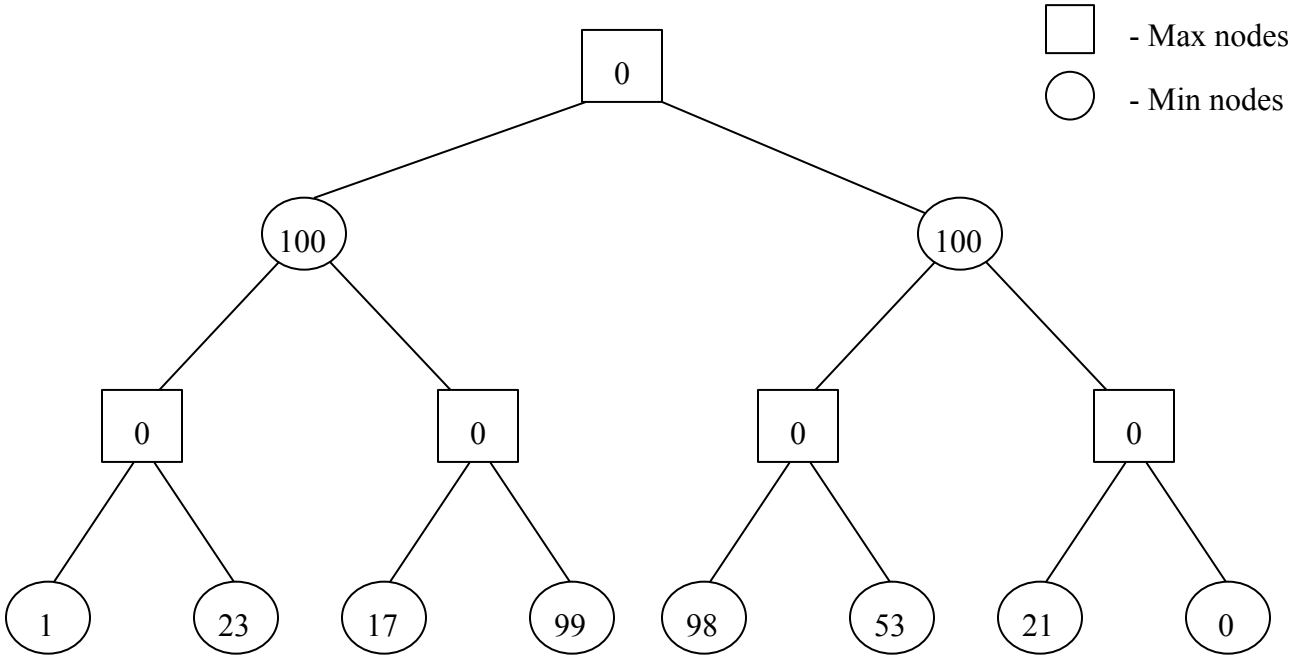

Figure 2