[Binary Tree Problem]

- Data Structure 5th Homework

Here are 11 binary tree problems in increasing order of difficulty. Some of the problems operate on binary search trees while others work on plain binary trees with no special ordering.

1. Size

This problem demonstrates simple binary tree traversal. Given a binary tree, count the number of nodes in the tree.

```
int size(Nptr T);
```

2. getHeight

Given a binary tree, compute its "height" -- the number of nodes along the longest path from the root node down to the farthest leaf node.

```
int getHeight(Nptr T);
```

3. minValue

Given a non-empty binary search tree (an ordered binary tree), return the minimum data value found in that tree. Note that it is not necessary to search the entire tree. A maxValue() function is structurally very similar to this function. This can be solved with recursion or with a simple while loop.

```
int minValue(Nptr T);
```

4. maxValue

Given a non-empty binary search tree (an ordered binary tree), return the maximum data value found in that tree.

```
int maxValue(Nptr T);
```

5. printInorder

Given a binary search tree, iterate over the nodes to print them out in increasing order. So the tree...

```
      4
     / \
    2   5
   / \
  1   3
```

produces the output "1 2 3 4 5". This is known as an "inorder" traversal of the tree.

```
void printInorder(Nptr T);
```

6. printPostorder

Given a binary tree, print out the nodes of the tree according to a bottom-up "postorder" traversal -- both subtrees of a node are printed out completely before the node itself is printed, and each left subtree is printed before the right subtree. So the tree...

```
      4
     / \
    2   5
   / \
  1   3
```

produces the output "1 3 2 5 4".

void printPostorder(Nptr T);

7. hasPathSum

We'll define a "root-to-leaf path" to be a sequence of nodes in a tree starting with the root node and proceeding downward to a leaf (a node with no children). We'll say that an empty tree contains no root-to-leaf paths. So for example, the following tree has exactly four root-to-leaf paths:

```
          5
         / \
        4   8
       /   / \
      11  13  4
     /  \      \
    7    2      1
```

Root-to-leaf paths:
path 1: 5 4 11 7
path 2: 5 4 11 2
path 3: 5 8 13
path 4: 5 8 4 1

For this problem, we will be concerned with the sum of the values of such a path -- for example, the sum of the values on the 5-4-11-7 path is 5 + 4 + 11 + 7 = 27. Given a binary tree and a sum, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum. Return false if no such path can be found.

int hasPathSum(Nptr T, int sum);

8. sameTree

Given two binary trees, return true if they are structurally identical -- they are made of nodes with the same values arranged in the same way.

int sameTree(Nptr T1, Nptr T2);

9. printPaths

Given a binary tree, print out all of its root-to-leaf paths as defined above. This problem is a little harder than it looks, since the "path so far" needs to be communicated between the recursive calls. Hint: probably the best solution is to create a recursive helper function printPathsRecur(node, int path[], int pathLen), where the path array communicates the sequence of nodes that led up to the current call. Alternately, the problem may be solved bottom-up, with each node returning its list of paths. Given a binary tree, print out all of its root-to-leaf paths, one per line.

void printPaths(Nptr T);

10. mirror

Change a tree so that the roles of the left and right pointers are swapped at every node. So the tree...

```
     4
    / \
   2   5
  / \
 1   3
```

is changed to...

```
   4
  / \
 5   2
    / \
   3   1
```

The solution is short, but very recursive. As it happens, this can be accomplished without changing the root node pointer, so the return-the-new-root construct is not necessary.

void mirror(Nptr T);

11. isBST

Suppose you have helper functions minValue() and maxValue() that return the min or max int value from a non-empty tree (see problems 3&4 above). Write an isBST() function that returns true if a tree is a binary search tree and false otherwise. Returns true if a binary tree is a binary search tree.

int isBST(Nptr T);

note: you may make a new function 'insertFalse' to make a non-binary search tree and test your isBST function.

* Due Date
- June 5, 2007 (Tuesday), 23:59:59

Good Luck!