Data Structure Lab Assignment (CS 2172)
Assignment 7: Binary Search Tree

Time: 2 weeks

1. Write functions to support Binary Search Tree (BST)

In this assignment you are required to write C functions for performing different operations on a binary search tree (BST). The specifications of those functions along with their signatures are given below.

1. ***int isEmpty(treenode *t)***

   t points to the root node of a BST (may be null if the BST is empty). This function tests whether the tree is empty or not. It returns 1 if the BST is empty, otherwise it returns 0.

2. ***treenode *insertNode(treenode *t, int val)***

   t points to the root node of a BST (may be null if the BST is empty). This function will insert the data *"**val**"* at its proper place in the BST. The function returns a pointer to the root node of the resulting BST (that is, the BST resulted after the insertion of the node). This is the function which can be called repeatedly to construct a BST.

3. ***int getCount(treenode *t)***

   t points to the root node of a BST (may be null if the BST is empty). This function counts the number of nodes in the BST and returns the count.

4. ***int inorder(treenode *t)***

   t points to the root node of a BST (may be null if the BST is empty). This function prints the data associated with the nodes when they are visited in **inorder** traversal of the BST. It returns 1 after successfully printing the information or returns 0 if the tree is empty.

5. ***int preorder(treenode *t)***

   t points to the root node of a BST (may be null if the BST is empty). This function prints the data associated with the nodes when they are visited in **preorder** traversal of the BST. It returns 1 after successfully printing the information or returns 0 if the tree is empty.

6. ***int postorder(treenode *t)***

   t points to the root node of a BST (may be null if the BST is empty). This function prints the data associated with the nodes when they are visited in **postorder** traversal of the BST. It returns 1 after successfully printing the information or returns 0 if the tree is empty.

You may use the following structure for the nodes of the trees.

```
typedef struct node {
    int data;
    struct node *lchild;
    struct node *rchild;
    struct node *parent;
} treenode;
```

Where *lchild, rchild,* and *parent* are the pointers respectively to the left child, right child and to the parent and *data* is the information stored in the node. For the root node *parent* is always *NULL*.

2. Extension of the earlier assignment on binary search trees (BST)

In this assignment you have to define C functions which take binary (search) tree(s) as parameter(s) and perform different task(s) as specified below. You can use the function(s) you have created in the earlier assignment(s) on binary search tree to facilitate demonstration of the functions you are defining today. For example, the function ***treenode *insertNode(treenode *t, int val)***, that you have defined then can be used today to create binary (search) tree(s) on which you can test today's functions.

Functions to be defined today

1. ***int height (treenode *t)*** - This function returns the height of the tree whose root node is pointed to by "***t***".
2. ***int max(treenode *t)*** - This function returns the maximum key in the tree whose root node is pointed to by "***t***".
3. ***int min(treenode *t)*** - This function returns the minimum key in the tree whose root node is pointed to by "***t***".
4. ***int equal(treenode *t1, treenode *t2)*** - This function returns 1 (true) if the trees whose root nodes are pointed to by "***t1***" and "***t2***", respectively, are ***equal***. Otherwise the function returns 0 (false). Recall that two trees are said to be equal if they both contains same keys and they are structurally same.
5. ***treenode * insucc(treenode *t, int key)*** - This function returns the pointer to the inorder successor of the node containing the item “key” in the tree whose root node is pointed to by “***t***”. It returns NULL if no such node could be found.
6. ***int deleteNode(treenode *t, int val) - t*** points to the root node of a BST (may be null if the BST is empty). This function will delete a node if the data “***val”*** matches with any node. It returns True if successfully deletes a node and return False if a node with “***val***” is not found.

Optional Assignment: [Attempt number 3 only if 1 and 2 are completed]

3. Creation of Binary trees from their traversals

Given certain traversals of a particular **binary tree**, in this assignment, you are required to recreate the binary tree.

It can be observed that different binary trees with the same set of values may produce the same traversal (inorder /preorder/ postorder). Therefore, given a traversal it is not possible to recreate the underlying binary tree. But if for a binary tree both inorder and preorder traversals are given then it is possible to recreate the tree from those traversals. In other words two different binary trees may not have same inorder traversals and same preorder traversals. Same is the case if you are given inorder and postorder traversal.

Write a function getTreeInPre() which takes two traversals say inorder and preoder and creates the unique tree that produces the these traversals.

The prototype for getTreeInPre() is as follows.

treenode* getTreeInPre(int a[], int asize, int b[], int bsize)

a[] and b[] are the arrays which contains inoder and preorder traversal. The function returns the pointer to root of the tree that has been created by it.

Similarly, write a function ***treenode* getTreeInPost(int a[], int asize, int b[], int bsize)*** which takes two traversals say inorder (in a[]) and postoder (in b[]) and creates the unique tree that produces the these traversals.

Write a supporting main function to demonstrate your functions. Your program should be well-documented and properly indented.