Binary Search Tree in c

Shengbo lou
28530995
1/28/2016

1. **structs:**
t_node is the struct for the innner nodes, it has four fields: data, three t_node pointers: its parent, leftnode and rightnode.(Using typedef makes my life easier)
bst_t is the struct for the actual binary Search tree, it only has one field: the root of the tree.

2. **void bst_insert(bst_t* tree, int data):**

This method uses the help method: node_insert(t_node* node, int data).

For the help method:

First I check whether the root is NULL, if it's null, then create a root.

If it's not null, I compare the data and root's data. If data > tree->data, then
means it should be inserted at the right branch, I use recursion to do it. When
I reach the leaf, means it's time to insert(Yeah~ :)). create a node and insert it.
The same structure for data < tree->data.

3. **int bst_find(bst_t* tree, int data, int* found):**

This method is the easiest one(I think), it uses the help method node_find(t_node* node, int data, int* found)

For the help method:

First I check if tree is NULL, if it's null, just return -1(undefined).

If it's not null, I compare the data and tree's data. If data > tree->data, I keep
searching its right branch, if data < tree->data, I keep searching its left branch.
I use recursion too.

If I still haven't found it when I reach the leaf, then I set * found to 0 and return -1.

If I find it on my way to leaves, I set * found to one, and return 1;

4. **void bst_delete(bst_t* tree, int data):**

Last and hardest one, it uses the help method node_delete(t_node* node, int data)

For the help method:

First I check if tree is NULL, if it's null, just return.

If it's not null, I compare the data and tree's data. If data > tree->data, I go to its right branch and searching. If data < tree->data, I go to left branch and searching. I use recursion again.

If I find it, then there are three different cases: it has two children. it only has leftnode, it only has right node, it'a a leaf. The easiest one is the leaf one, just remove it. If it has only one child, it's also easy, just connect its child with its parent and delete itself.

The complicated one is it has two children, I will take the largest one (tmp) at its left branch and make it root. Then if tmp has only one child or it's a leaf(if I'm lucky enough), they will be one of cases I mentioned before.

And I free the node at the end.

That's it!