Assignment 6: Binary Search Trees

Write functions in C to perform the following operations on BST. The keys will be positive integers. The struct (representing a node of the tree) will ***necessarily*** have

a) the key-value

b) pointer to the left child,

c) pointer to the right child and

d) pointer to parent.

Note that we assume distinct keys. (A minor tweak can allow for duplication as well).

The following functions have to be implemented (please NOTE the id number of the functions):

1. ***Insert(Tree*, key).*** The creation of a tree would, therefore, be a series of inserts. Returns true on successful insertion otherwise returns false!

2. ***Delete(Tree*,key)***. Deletes the particular key from the tree if it exists. Returns true on successful deletion otherwise returns false!

3. ***Find(Tree*, key).*** Returns true if exists and prints its depth otherwise returns false!

4. ***PrintTree(Tree*)***: Print the ***In-order, Pre-order, Post-order*** of the tree .

5. ***PrintSubTree(Tree*, key):*** Print the above three traversals for the subtree rooted at the node containing "key"

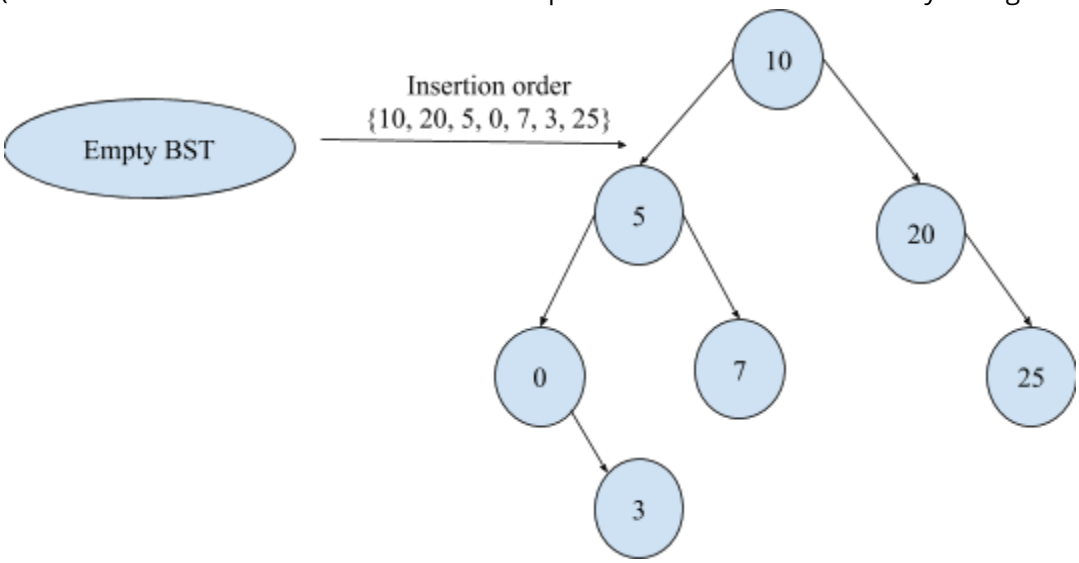
6. ***CalculateImbalance(Tree*, key):*** Calculates the difference between the number of nodes in the left subtree and the number of nodes in the right subtree rooted at *the key*. You can assume that this function is being invoked only for valid keys!

Here Tree* in the argument can be the pointer to the root node, some global variable, or some custom struct that you have that identifies the tree.

For more information about the sample I/O please refer to the following diagrams!

The following diagram shows the insertion in BST

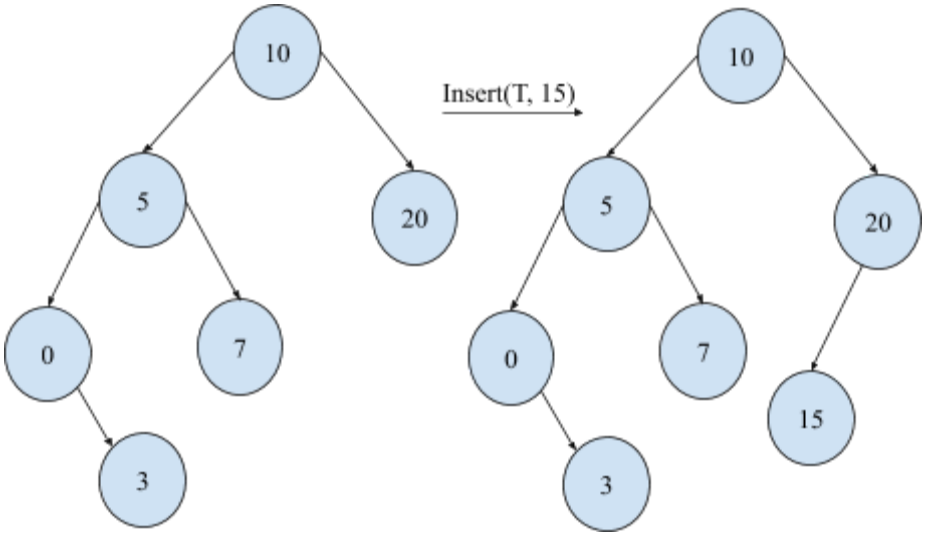
(Note that there can be more than one permutation of the same keys to generate the same BST)



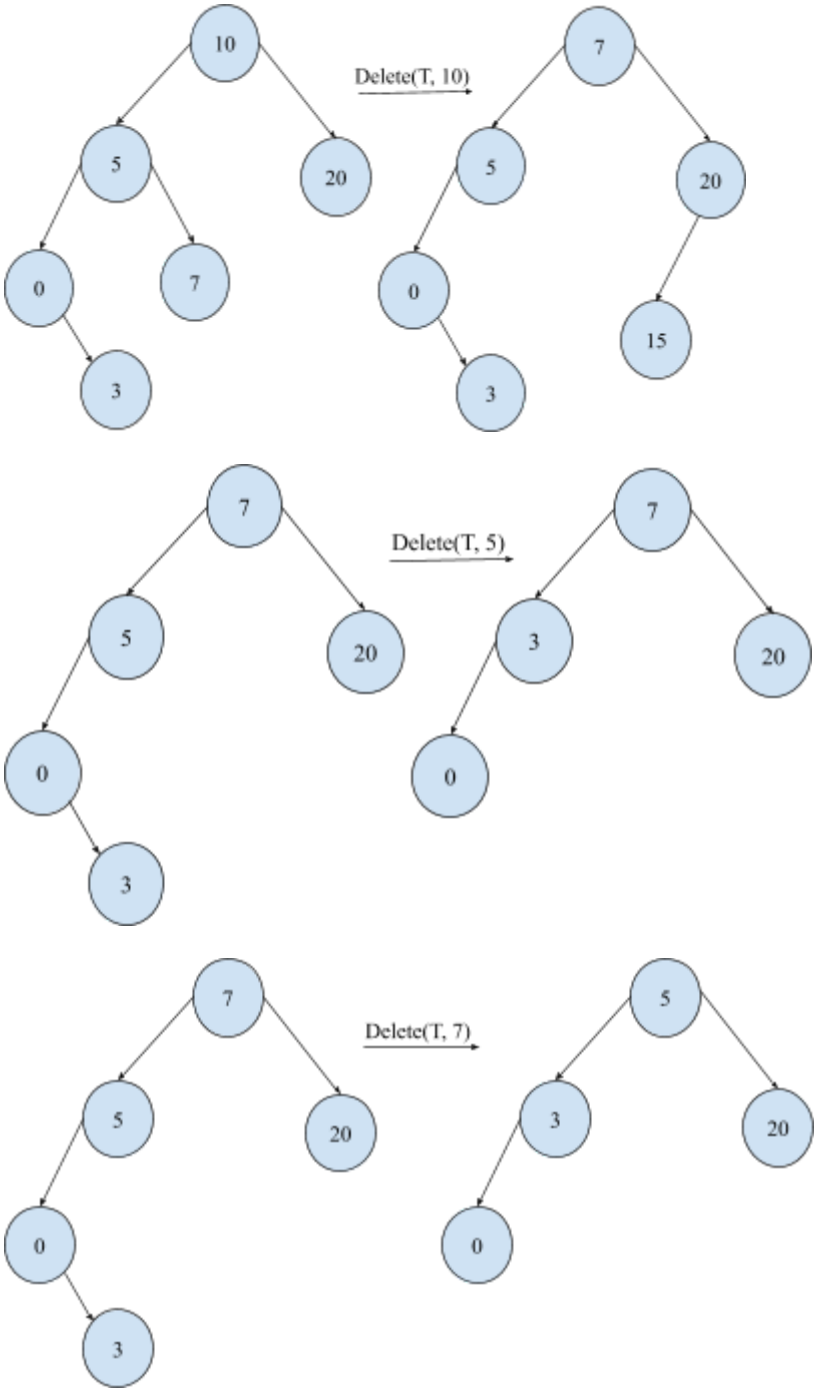
The above tree is made after a series of insertion as shown:

Insert(T, 10) → Insert(T,20) → Insert(T,5) → Insert(T,0) → Insert(T,7) → Insert(T,3) → Insert(T,25)

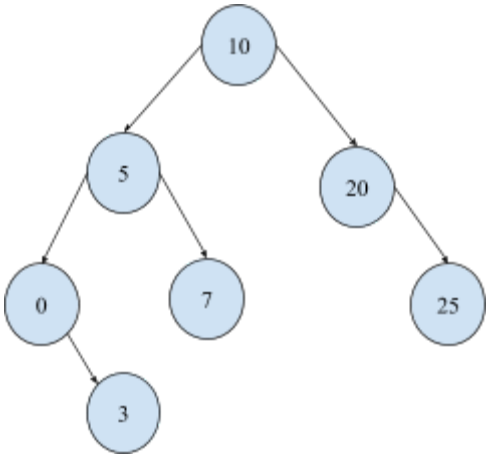
The following diagram shows a single insertion for your reference:



The following diagram shows a deletion (*we assume that deletion is implemented as a replacement with predecessor policy*)



Let us consider the following tree for the rest of the examples:



Sample function outputs:

Find(T, 10) → true, depth: 0

Find(T, 3) → true, depth: 3

Find(T, 100) → false

Im-balance(T, 10) → 2 (#left child - #right child = 4 - 2 = 2)

Im-balance(T,5) → 1 (#left child - #right child = 2 - 1 = 1)

Print(10) generates the following output:

In-order Traversal: 0 3 5 7 10 20 25

Pre-order Traversal: 10 5 0 3 7 20 25

Post-order Traversal: 3 0 7 5 25 20 10

Input/Output Format

input.txt

(*Please use relative paths in fopen(), not absolute paths like "C:\myname\ass.c" . Relative paths, like "ass1.c" means looking for a file called ass1.c in the current directory.)*

First line: **N,** Number of operations to be executed.

The next **N** lines: The operations to be executed.

Each operations is encoded as: <operation number, as mentioned in the beginning of the assignment> <argument>

Arguments are supplied where necessary, otherwise nothing is provided (just the operation number)

Sample input:

5

1 15

1 7

3 7

4

5 7

This input means adding 15, then 7 to the tree. Then find(7), PrintTree(), and PrintSubTree(7)

Sample Output (output.txt)

Note: Please do not print ANYTHING else, like "enter the input", "menu" etc.

true

true

true, depth=1

Inorder: 7 15

Preorder: 15 7

Postorder: 7 15

Inorder: 7

Preorder: 7

Postorder: 7

Output details:

For each function, print the result in a separate line:

1)Print the corresponding truth values for **Insert, Delete, Find (also depth if find is true)**

2) **CalculateImbalance(key)** should print the difference(self explanatory)

3)For **print tree,** please print a sequence of numbers
separated by spaces (consider the tree in assignment), for each of the traversals.
pre-order : 10 5 0 3 7 20 25
post-order : 3 0 7 5 25 20 10
In-order : 10 5 0 3 7 20 25