January 2023 CSE 208
Offline Assignment on Balanced Binary Search Tree

Deadline: Friday, 28 July 2023, 11:45 PM
Last modified : Friday 21st July, 2023 22:04

Current Version	Time of Modification	Modification
1.0	Friday 21st July, 2023 17:50	Assignment published

Table 1: list of modifications

A **Binary Search Tree** (BST) is a binary tree data structure which has the following properties:

1. The left subtree (if it exists) of a node contains only nodes with keys **less** than the node's key.
2. The right subtree (if it exists) of a node contains only nodes with keys **greater** than the node's key.
3. The left and right subtrees each must also be a binary search tree.

We assume that all the keys will be **unique**.

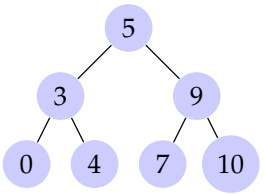

Figure 1: A sample BST

You have implemented BST in your DSA-I sessional. Now it is time to upgrade ourselves. We are now more concerned about achieving the optimal $\mathcal{O}(\log n)$ complexity for insertion, deletion and search operations. For this, we have learned about 3 variations of balanced BST (AVL tree, Red Black tree and Splay tree) in the class. To have a more grasp on them we will implement AVL tree in this assignment.

Our AVL tree will have the following functions:

1. **Insert:** It inserts a key in the BST.
2. **Delete:** It deletes the key in the BST if it exists in the BST.
3. **Find:** Searches whether a key is present in our BST.
4. **Traversal:** We only traverse the tree in order.

Please note, amortized complexity of each operation (except traversal) should be $\mathcal{O}(\log n)$.

You need to report the total amount of time (in milliseconds) each operation takes. Please have a look at sample report_splay.txt for clarification.

Input

After taking input from a file named *in.txt*, you will generate output on two files named *out_avl.txt*, *report_avl.txt*. Each line in the input will specify one of the following operations:

1. Insert (I) followed by an integer denoting the value of the node to be inserted

2. Delete (D) followed by an integer denoting the value of the node to be deleted

3. Find (F) followed by an integer denoting the value of the node to be searched for

4. Traversal (T).

Output

After,

1. Insert: the current state of the tree after insertion will be printed in the nested parentheses format. For example, the nested parentheses format for the tree in Figure 1 will be written as:

```
5(3(0,4),9(7,10))
```

2. Delete: same as insert
3. Find: print “found“ or “not found“
4. Traversal: print the keys in order mode separated by space. An in-order traversal in 1 would be:

```
0 3 4 5 7 9 10
```

Sample I/O

Input: in.txt

```
F 1
I 1
I 2
I 3
I 4
I 5
I 6
I 7
I 8
I 10
T
D 5
D 7
F 7
D 6
D 4
F 1
T
```

Ouput file: out_avl.txt

```
not found
1
1(,2)
2(1,3)
2(1,3(,4))
2(1,4(3,5))
4(2(1,3),5(,6))
4(2(1,3),6(5,7))
4(2(1,3),6(5,7(,8)))
4(2(1,3),6(5,8(7,10)))
1 2 3 4 5 6 7 8 10
```

```
4(2(1,3),8(6(,7),10))
4(2(1,3),8(6,10))
not found
4(2(1,3),8(,10))
8(2(1,3),10)
found
1 2 3 8 10
```

Ouput file: report_avl.txt

```
operation time(ms)
insert 0.00488281
delete 0
search 0
trav 0.00585938
total 0.0107422
```

Submission Guideline

1. Create a directory with your 7 digit student id as its name
2. Put the source files only into the directory created in step 1
3. Zip the directory (compress in .zip format; .rar, .7z or any other format is not acceptable)
4. Upload the .zip file on moodle.

For example, if your student id is 205xxx, create a directory named 2005xxx. Put only your source files (.c, .cpp, .java, .h, etc.) into 2005xxx. Compress 2005xxx into 2005xxx.zip and upload the 2005xxx.zip on moodle.

Mark Distribution

	Task Detail	Marks
1	Proper implementation of insert	2
1	Proper implementation of delete	4
3	Height balancing using rotation	10
4	Find and traversal functions	3
5	Reporting time	1

For Queries

If you have any questions related to the assignment please first check the queries thread in Moodle. You should post your confusion in the thread. If your query goes unanswered for 24 hours, please mail me.

Special Instructions

When writing code, it is essential to ensure readability, reusability, and good structure. This involves using appropriate functions to implement algorithms, giving variables meaningful names, adding suitable comments when necessary, and maintaining proper indentation. You will need to use your offline implementation to solve the online. There will be a viva too. So, please understand the concepts before you proceed to code.

Please note that you must rely on your own implementation to solve the assigned tasks. It is strictly prohibited to copy code from any source, including friends, seniors, or the internet. **Any form of plagiarism, regardless of its origin or destination, will result in a deduction of 100% marks for the offline assessment**. Moreover, repeated instances of plagiarism may lead to stricter consequences in accordance with departmental policies.