CS 202, Spring 2021

Homework 2 – Binary Search Trees

Due: 23:55, March 18, 2021

Important Notes

Please do not start the assignment before reading these notes.

1. Before 23:55, March 18, upload your solutions in a single **ZIP** archive using Moodle submission form. Name the file as studentID_secNo_hw2.zip.
2. Your ZIP archive should contain the following files:
   - **`hw2.pdf`**, the file containing the answers to Question 1 and 3.
   - **BinaryNode.h, BinaryNode.cpp, BinarySearchTree.h, and BinarySearchTree.cpp** files as well as any additional class and its header files which contain the C++ source codes, and the **`Makefile`**.
   - Do not forget to put your name, student id, and section number in all of these files. Well comment your implementation. Add a header as given below to the beginning of each file:

```
/*
 * Title: Binary Search Trees
 * Author: Name Surname
 * ID: 21000000
 * Section: 1
 * Assignment: 2
 * Description: description of your code
 */
```

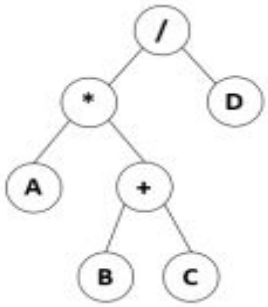
   - Do not put any unnecessary files such as the auxiliary files generated from your favorite IDE. Be careful to avoid using any OS dependent utilities (for example to measure the time).
   - You should upload handwritten answers for Q1 (in other words, do not submit answers prepared using a word processor).
   - Use the exact algorithms shown in lectures.
3. Although you may use any platform or any operating system to implement your algorithms and obtain your experimental results, your code should work on the `dijkstra` server (dijkstra.ug.bcc.bilkent.edu.tr). We will compile and test your programs on that server. Thus, you will lose a significant amount of points if your C++ code does not compile or execute on the `dijkstra` server.
4. This homework will be graded by your TA, Mert Sayar. Thus, please contact him directly (mert.sayar at bilkent.edu.tr) for any homework related questions.

Attention: For this assignment, you are allowed to use the codes given in our textbook and/or our lecture slides. However, you ARE NOT ALLOWED to use any codes from other sources (including the codes given in other textbooks, found on the Internet, belonging to your classmates, etc.). Furthermore, you ARE NOT ALLOWED to use any data structure or algorithm related function from the C++ standard template library (STL).

Do not forget that plagiarism and cheating will be heavily punished. Please do the homework yourself.

Question 1 – 20 points

(a) (6 points) Give the prefix, infix, and postfix expressions obtained by preorder, inorder, and postorder traversals, respectively, for the expression tree below:



(b) (9 points) Insert 31, 7, 56, 2, 1, 41, 45, 10, 70, 42, 38, 9 to an empty Binary Search Tree, and then delete 1, 45, 56, 7 in the given order. Show the evolution of the BST after each insertion and deletion operation.

(c) (5 points) A binary search tree has a preorder traversal of 18, 5, 11, 9, 7, 13, 12, 15, 21, 19, 28, 23, 25, 24, 26. Give the corresponding binary search tree. What is its postorder traversal?

Question 2 – 65 points

You will write a pointer-based implementation of Binary Search Tree with various functions. Each node object will keep an integer data value together with left and right child pointers.

You are required to implement following functions:

`bool isEmpty()`: Tests whether the BST is empty. True if BST is empty; otherwise false.

`int getHeight()`: Returns the height of the BST.

`int getNumberOfNodes()`: Returns the number of nodes in the BST.

`bool add(const int newEntry)`: Adds a new node containing a given data item to the BST. If the data item already exists in the current BST, then don't insert. Returns true if the addition is successful, or false if not.

`bool remove(const int anEntry)`: Removes the node containing the given data item from BST. True if the removal is successful, or false if not.

`bool contains(const int anEntry)`: Tests whether the given data item occurs in the

BST. True if the BST contains the given data item, or false if not.

`void preorderTraverse():` Traverses the BST in preorder and prints data item values in traversal order.

`void inorderTraverse():` Traverses the BST in inorder and prints data item values in traversal order.

`void postorderTraverse():` Traverses the BST in postorder and prints data item values in traversal order.

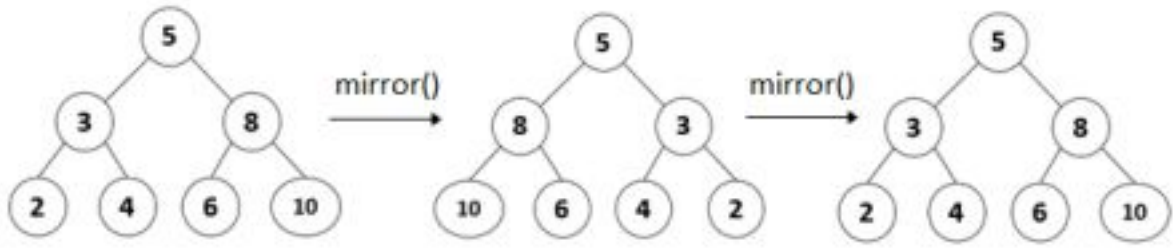
`void levelorderTraverse():` Traverses the BST in level-order and prints data item values in traversal order. A level-order traversal of a tree processes (visits) nodes one level at a time, from left to right, beginning with the root.

For this solution, if you need to use any additional data structure other than binary search trees, you should define and implement a class for that data structure. Note that you are allowed to use the C++ codes for any data structure such as ADT stack and ADT queue that are given in your textbook but are not allowed to use any other source implementation. For example, you cannot use the stack class defined in another book or on a web site. Similarly, you are not allowed to use any functions in the C++ standard template library (STL). You should upload this additional class as well as its header file in your solution.

`int span(const int a, const int b):` Returns the number of nodes in the BST with data values within a specific range. In other words, it is the number of nodes with the data value >=a and <=b.

You should implement an efficient function -- that is, your function should not traverse any parts of the tree that it does not need to traverse. If you write a function that visits all nodes in the tree, you will get NO points from this question.

`void mirror():` Changes the BST so that the roles of the left and right pointers are swapped at every node. When you apply this function to a BST, the data value in a node will be less than the data values in its left subtree and greater than the data values in its right subtree. If you apply this function to the mirrored tree, you should obtain the original BST.



- Design the node structure together with its set and get functions (and maybe some additional functions if necessary) in **BinaryNode.h** and **BinaryNode.cpp** properly.
- Implement the BST structure with above functions and their interfaces in **BinarySearchTree.h** and **BinarySearchTree.cpp** properly. You are free to write helper functions to accomplish the tasks required from the above functions. **Use the given file names and function signatures during implementation.** You will not submit a main.cpp file, instead we will use our main.cpp file during evaluation. Therefore, it is important to use given file names and function signatures.

Question-3 – (15 Points)

Explain briefly how you implemented above **levelorderTraverse**, **span** and **mirror** functions and analyze their worst-case running time complexities. Give the time complexities. **Discuss** whether or not each can be implemented to be asymptotically faster.