CS 202, Summer 2021

Homework 2 – Binary Search Trees

Due: 23:55, July 7, 2021

Important Notes

Please do not start the assignment before reading these notes.

1. Before 23:55, July 7, upload your solutions in a single **ZIP** archive using Moodle submission form. Name the file as studentID_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, BinarySearchTree.cpp, main.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 and student id 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, Berat Biçer. Thus, please contact him directly (berat.bicer 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 – 15 points

You must draw the trees in this question by hand and embed the scans of your drawings in your report. Make sure that your drawings are clear. Ambiguities would make you lose points.

(a) [5 points] Insert 5, 3, 9, 4, 8, 6, 1, 7, 2, 10 into an empty binary search tree (BST) in the given order. Show the resulting BST **after every insertion**.

(b) [5 points] What are the preorder, inorder, and postorder traversals of the BST after (a)?

(c) [5 points] Delete 1, 5, 9, 10, 7 from the BST you have after (a) in the given order. Show the resulting BST **after every deletion**.

Question 2 – 70 points

(a) [20 points] Write a pointer-based implementation of Binary Search Tree named as `BST` for maintaining a collection of integer keys. Each node object keeps an integer data value together with left and right child pointers and must be implemented using a class named `BinaryNode`. Implement the following functions in your class:

`bool insert(const int key)`: Inserts a new node containing the given integer key to the BST. If the key already exists in the BST, do not do any insertion. Returns true if insertion is successful, and false if not.

`bool delete(const int key)`: Deletes the node containing the given integer key from the BST. Returns true if deletion is successful, and false if not.

`BinaryNode* retrieve(const int key)`: Returns the address of the node that contains the given integer key. Returns NULL if the key does not exist in the BST.

`void inorderTraversal(int*& array, int& length)`: Performs an inorder traversal of the BST and fills a dynamically allocated `array` with the keys in traversal order. The array must be dynamically allocated in this function. The `length` of the array is also returned.

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

(b) [15 points] Write a function with the following prototype that computes and returns the number of nodes that are height balanced in the BST. A node is considered height balanced if the heights of the left subtree and the right subtree of that node differ by at most 1.

```
int numHeightBalanced()
```

(c) [15 points] The level of a node is the number of nodes along the path from the node to the tree's root node. The level of the root node is 1. Write a function with the following prototype that computes and returns the number of nodes whose level is greater than or

equal to the given number named `level`.

```
int numNodesDeeper(int level)
```

(d) [10 points] Use the `BST` class to insert and delete the keys given in Question 1 into an empty tree. Declare a `BST` object and perform the insertion and deletion operations in Question 1 in the given order. Then, use the `inorderTraversal` function to return and display the contents of the resulting tree. Implement this step in the `main` function in a file named `main.cpp`.

(e) [10 points] Write a global function named **`heightAnalysis()`** that analyzes how the height of a BST changes after a sequence of insertion and deletion operations. This global function should be given in `main.cpp` and should perform the following operations:

(i) Creates an array of 20000 random integers and inserts them one by one into an empty BST starting from the beginning of the array. After each set of 1000 insertions, output the height of the tree.

(ii) Shuffles the array created above, and deletes the integers one by one from the BST starting from the beginning of the shuffled array. After each set of 1000 deletions, output the height of the tree.

Question 3 – 15 points

In this question, you are asked to perform additional analysis on the height of a BST after a sequence of insertion operations. You are given auxiliary global functions as part of the homework assignment (see `auxArrayFunctions.h` and `auxArrayFunctions.cpp` files). The function that should be used is named `createNearlySortedArrays`. This function creates an almost sorted array of length N in which each item is at most K locations away from its target location.

You can fix N as 20000, and use at least three different values of K (in a wide range) to generate different arrays. Then, as in part (e) of Question 2, you can use each array separately and insert the integers one by one into an empty BST starting from the beginning of the array. After each set of 1000 insertions, you can output the height of the tree.

After running your programs, prepare a single page report about the height analysis. This report should provide a single figure (prepared using a plotting program) on which you plot the height of the tree (y-axis) versus the number of items in the tree (x-axis) for the insertion results obtained in part (e) of Question 2 and for the three different arrays used in Question 3. Then, you should discuss how these height values relate to the theoretical values discussed in the class.