Computing Structures – Fall 2022
Project 5
Due: November 22nd 2022 at 11:59 PM

Objective:

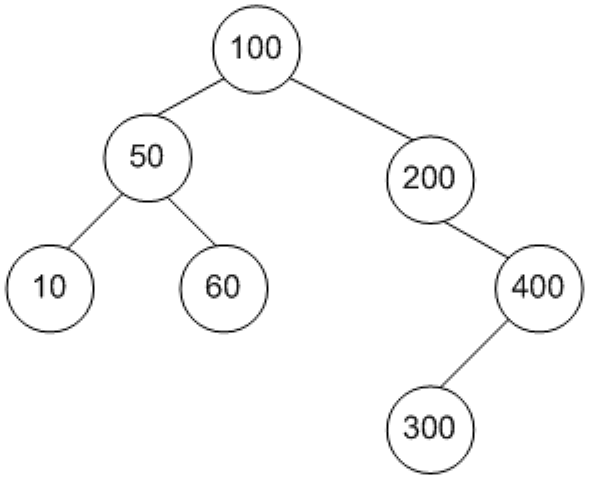
1. Use STL stack and vector in the project.
2. Implement ArrayBST class.
3. Implement main() that read and process the input according to the project description
   - for each of I, F, O and A commands.
   - R command is extra credit of 20%.

Description:
The main objective of this project is to store the binary search tree using an array data structure as presented below.

ArrayBST:
The binary search tree data structure can be stored using a number of ways including pointers (_left and _right tree pointers), with a single dimensional array (mainly used for storing a complete binary tree), and the array of left and right index (call this ArrayBST) approach. The ArrayBST is similar to the array implementation of a linked list discussed in the textbook.

In the ArrayBST, we will have an array (or vector) of ArrayBTNode objects. The size of the array or vector is K, which can be specified in the constructor with default value of lets say 20. The root of the binary tree is placed at an index position say *x* with a value of say 100. Let us also assume that the root node has a left child with a value of 50 and a right child with a value of 200. The left child (or the right child) can be at any index position of the array. Let these positions be *y* (left index) and *z* (right index), respectively. We have to store this information at the root node located at index position *x*. An example of this structure is given below.

position	_info	_left	_right
0	100	2	4
1	NULL	-1	-1
2	50	3	6
3	10	-1	-1
4	200	-1	5
5	400	8	-1
6	60	-1	-1
7	NULL	-1	-1
8	300	-1	-1



The left child (resp. right child) of the root is at index position 2 (resp. 4). The left child of node with value 50 is at index position 3 while its right child with a value of 60 is at index position 6. Positions 1 and 7 are empty slots (because _info is set to NULL) wherein new nodes can be placed. The algorithms to remove a node will place NULL at appropriate index positions. When the left (resp. right) index is a -1 then it implies that the left child of that node is empty.

Insertion and Removal

In order to insert a new node, first we need to find an index position with _info set to NULL. The time to complete this task in the worst case will be equal to the size of the vector. But a simple technique to keep track of the free positions is to use a stack. You can use the STL stack (call it freeLocations) for this purpose. When the BinarySearchTree is created you will initialize this stack with empty positions 0 through maximum number of nodes minus one. That is, will push values 0, 1, 2, 3, …, n-1, onto the stack where n is the maximum number of nodes allowed in the binary tree. So, you will have n – 1 as the first index position to store the first insertion node. Whenever you need to insert a node into the binary search tree you pop an index position from the stack. When a node is removed, we need to push the corresponding position onto the stack.

search

Search returns the index of the ArrayBTNode object in the vector. Sequential search cannot be used to implement search method. Binary Search is the way to go about this search.

ostream operator and DisplayRaw

The overloaded ostream operator should print both the **preorder** and **inorder** traversals. You will also implement a DisplayRaw method that prints out the following information stored in the data structure:

(a) Position, *_info, _left, _right in the array (only for the rows with data stored)
(b) The content of the stack

Input explanation:

We will insert/remove integers into/from an ArrayBST object. The first line of the program will be the value for K (size of the vector), followed by commands for I (insert), R (remove), F (search), O (output the tree using the ostream operator), A (output using DisplayRaw method).

You can only insert a maximum of K nodes into the tree. Anything over K, you cannot insert into the BST until you remove another node. Similar to the previous project, a separate input will be provided for remove (**project5_extra.cpp**).

Class definition:

A boiler plate source file has been provided along with this project specification.

A sample output file will be provided (in the next few days) for the given corresponding input file. Your output is supposed to exactly be the same as this output. You need to essentially follow this format of the output. This is necessary for your program to pass the autograder.

Redirected input:

Redirected input provides you a way to send a file to the standard input of a program without typing it using the keyboard. To use redirected input in Visual Studio environment (on windows), follow these steps: After you have opened or created a new project, on the menu go to project, project properties, expand configuration properties until you see Debugging, on the right you will see a set of options, and in the command arguments type <"input filename". The < sign is for redirected input and the input filename is the name of the input file (including the path if not in the working directory).

If you use macOS or linux (or windows using powershell), you may use the terminal to compile and run your program using g++. This is the compiler that we use in the autograder on GradeScope (you may need to install g++ on your computer). Once you installed g++, you may compile your program using the following command:

```
g++ project5.cpp -o p5
```

You may then run the program with the redirected input using the following command:

```
./p5 < input1.txt
```

Make sure your program and your input file are in the same folder.

Submission:

Submission will be through GradeScope. Your program will be autograded with test cases and then hand graded to check for documentation and if you have followed the specifications given. You may submit how many ever times to check if your program passes the test cases provided. Test cases will be released at the beginning and later other test cases will be released while grading. For the autograder to work, the program you upload must be named as **project5.cpp**.

Your final submission must contain your source file named **project5.cpp**. You access GradeScope using the tab on the left in our course page on canvas.
If you have attempted the extra credit, you will submit **project5_extra.cpp** as well along side the project5.cpp file.

Sample output file for corresponding input files will be released. The input1.txt file given is a simple input file that will help you understand the project, more complicated ones will be released later and used for grading.

Constraints:

1. In this project, the only header you will use the header files that are given to you in the boiler plate code. Using other or excess header files will be subject to a heavy grade penalty for the project.
2. None of the projects is a group project. Consulting with other members of this class on programming projects is strictly not allowed and plagiarism charges will be imposed on students who do not follow this.
    - Please review academic integrity policies in the modules.

How to ask for help:

1. Check FAQ discussion board of the project first.
2. Email the TA with your precise questions.
3. Upload your well commented program to CodePost.
    a. This is a website which is used to share code.
    b. You will upload your program and I can view it and comment on it.
    c. Here is the invite link to our class for the summer.
    d. Once you join the class on CodePost, you can upload your program to the Project 1 assignment.

        Note: Your program will not be auto graded at CodePost, this is just for when you want to ask a question and a place where I can look at your program and comment on it. CodePost is great for live feedback. GradeScope is the place where you should submit and where your program will be autograded.