CS 2413: Data Structures – Programming Project 4 – Spring 2017
Due 11:59 PM, April 14, 2017

Objectives

1. [10 pts] Use STL stack and vector in the project
2. [30 pts] Implement ArrayBT class
3. [25 pts] Implement ArrayBST class
4. [35 pts] Implement main() that read and process the input according to the project description
   – 10 points for each of I, F, O commands, 5 points for A command.
5. [20 points] Bonus – remove method
6. Document your project thoroughly as the examples in the textbook. This includes but not limited to header comments for all classes/methods, explanatory comments for each section of code, meaningful variable and method names, and consistent indentation. Program that lack or weak in documentation will receive a *deduction* of up to 30% of the grade.

Project Description

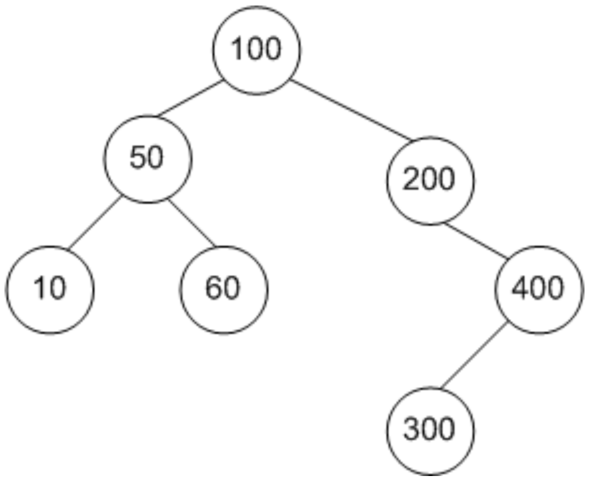
The binary search tree data structure was discussed class. The main objective of this project is to store the binary search tree using an array data structure as presented below.

ArrayBT

The binary 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 ArrayBT) approach. The ArrayBT is similar to the array implementation of a linked list discussed in the textbook.

In the ArrayBT, 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 100. 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	5	-1
5	400	8	-1
6	60	-1	-1
7	NULL	-1	-1
8	300	-1	-1



Please note that in the above table, we have assumed that the root is placed at position 0. The positions of various values in the table are arbitrary. 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. 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 BinaryTree (or 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. The order of pushing the values does not matter. 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.

The structures of these classes are given below. You need to implement all the methods given in the textbook for binary tree and binary search trees.

```
template <class DT>
class ArrayBTNode
{
    public:
        BT* _info;
        int _left; //the index position where the left ArrayBTNode will be available
        int _right;//the index position where the right ArrayBTNode will be available
//…… All the methods that is required
//…… Write the overloaded comparison operators >, <, ==, >=, <=, != for comparing
//……. the info field
};

template <class DT>
class ArrayBT
{
    public:
        vector<ArrayBTNode<DT>* >* myBinaryTree;
        int _root; //index position of the root the binary tree in the vector
        int _noNodes; //the number of nodes in the binary tree
        int _size; //the maximum size of the vector
        stack<int> freeLocations; //STL stack
//……..all the methods that are required
};
```

The binary search tree (ArrayBST) will inherit from ArrayBT.

```
template <class DT>
class ArrayBST: virtual public ArrayBT<BT>
{
    public:
        //insert, remove, inorder and preorder traversals
        //other methods that should be part of any class
};
```

search

Instead of returning a pointer to the subtree, search returns the index of the object in the array. Sequential search cannot be used to implement search method.

ostream operator and DisplayRaw

The overloaded ostream operator should print both the preorder and inorder traversals. The implementation of the remove method is optional and you will get a 20% bonus if you implement it. 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 File

We will insert/remove integers into/from an ArrayBST. The first line of the program will be the value for K (size of the array), followed by commands for I (insert), R (remove), F (search), O (output the tree using the ostream operator), A (output using DisplayRaw method). If you did not implement R command, you still need to process the command but only by displaying "Command not implemented." message.

Output

After each operation, display the outcome of the operation.

Constraints

1. This project must be implemented in C++. The only header file you will have <iostream>, <stack>, <vector>, and <cstdlib> (for atoi() which can be used to convert a C string to an integer in the input file). We will not use any other libraries.
2. None of the projects will be 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.