ITU Computer Engineering Department
BLG 223E Data Structures, Spring 2022
Recitation #4

Problem Definition

In this recitation, you are asked to implement a Binary Search Tree (BST) with some additional operations. As you may remember from your classes, BST data structure is mainly composed of add, traverse, search and remove operations. However, for this recitation, we are only interested in the add, traverse and the maximum sub-tree methods. You can see the content of the header files corresponding the BSTNode and BST as below.

Code Listing 1: BST Node Header

```
#ifndef BST_NODE
#define BST_NODE

class Node{
    private:
        int data;
        Node* left;
        Node* right;
    public:
        Node(int);
        void set_data(int);
        int get_data();
        void set_left(Node*);
        Node* get_left();
        void set_right(Node*);
        Node* get_right();
};

#endif
```

Code Listing 2: BST Header

```
#ifndef BST_DEF
#define BST_DEF

#include <vector>
#include "BSTNode.h"

using namespace std;

class BST{
    private:
        Node* root;

        void postorder_traverse(Node*);
        void postorder_destruct(Node*);

    public:
        BST();
        ~BST();

        Node* get_root();
        void add(int);
        void corrupted_add(vector<int>);
        bool contains(int);
        int maxSumBST(Node*);

        void printPostOrder();
};

#endif
```

You are asked to fill in the constructors and destructors, add, contains, maxSumBST, postorder_traverse, postorder_destruct and printPostOrder methods of the BST class. The purpose of all of these functions mentioned in BST class are provided below:

- BST::BST()
  - Class constructor for the BST class.
  - You need to provide your implementation for this method.
- BST::BST()
  - Class destructor for the BST class.
  - You are not obligated to use postorder_destruct method for the removal of the nodes.
  - You need to provide your implementation for this method.
- Node* BST::get_root()
  - Returns the root note.
- void BST::add(int add)
  - This method adds an integer in order to construct a BST.
  - If the integer exists within the tree, do not add it.
  - You need to provide your implementation for this method.

- void BST::corrupted_add(vector¡int¿ data)
  - corrupted_add method assigns a Node to a Binary Tree (BT), without considering the requirements of the BST. This suggests that considering a node in a BT, its left neighbour may have a **greater** value than itself, or its right neighbour may have **lower** value than itself. This makes the method incorrect to be used for constructing a BST and has nothing to do with your add implementation.
  - Breadth First Search (BFS) like queue mechanism is used for node assignment.
  - corrupted_add method has been **pre-implemented for you** to be used along with the maxSumBST method.
- bool BST::contains(int data)
  - This method checks whether a node with the specified integer input is present within the tree.
  - You need to provide your implementation for this method.
- int BST::maxSumBST(Node* node)
  - Used for finding the maximum sum of the nodes of a sub-tree that is a BST.
  - Nodes may take negative values.
  - You need to provide your implementation for this method.
- void BST::printPostOrder()
  - This method traverses the BST and prints the content of the nodes in post order.
  - You are not obligated to use postorder_traverse method for the removal of the nodes.
  - You need to provide your implementation for this method.
- void BST::postorder_traverse(Node*)
  - Used for recursively traversing the tree.
  - Implement the function only if you had used in somewhere.
- void BST::postorder_destruct(Node*)
  - Used for recursively deleting the nodes of the tree.
  - Implement the function only if you had used in somewhere.

You are not allowed to modify the content of 'main.cpp', but you are allowed to include new functions where necessary. You are free to use the member functions of the BST and Node classes in your code.

Note: The number of entries may reach upto a million, hence make sure that there are no memory leaks present in your code.

Sample Cases

```
Adding 6
Adding 3
Adding 3
Adding 5
Adding 9
Adding 7
Adding 12
Printing the tree with postorder traversal
5 3 7 12 9 6
```

Figure 1: Adding 7 nodes, only 6 of them are shown in the post-order traversal due to duplicate entry.

```
BST contains 3  : true
BST contains 15 : false
BST contains 12 : true
BST contains 6  : true
BST contains 7  : true
BST contains 5  : true
```

Figure 2: Checking whether nodes exist in somewhere within the tree or not.

```
Corrupted Adding 1
Corrupted Adding 4
Corrupted Adding 3
Corrupted Adding 2
Corrupted Adding 4
Corrupted Adding 2
Corrupted Adding 5
Corrupted Adding null value
Corrupted Adding null value
Corrupted Adding null value
Corrupted Adding null value
Corrupted Adding null value
Corrupted Adding null value
Corrupted Adding 4
Corrupted Adding 6
2 4 4 2 4 6 5 3 1
Sub-BST with Maximum Sum: 20
```

Figure 3: Post-order representation of the binary tree where its maximum sum is found to be 20.

Submission Rules

- You may need to sync the VSCode configuration files for this recitation, which you can find them on Ninova. You may want to change the ”args” line in launch.json, of you want to test with a different input file.
- Use comments wherever necessary in your code to explain what you did.
- You can use STL containers (vector, list, etc.) wherever needed.
- Your program will be checked by using **Calico(https://bitbucket.org/uyar/calico)** automatic checker.
- Do not share any code or text that can be submitted as a part of an assignment (discussing ideas is okay).
- Only electronic submissions through Ninova will be accepted no later than deadline.
- You may discuss the problems at an abstract level with your classmates, but you should not **share or copy code** from your classmates or from the Internet. You should submit your **own, individual** homework.
- Academic dishonesty, including cheating, plagiarism, and direct copying, is unacceptable.
- If you have any question about the recitation, you cand send e-mail to Caner Özer (ozerc@itu.edu.tr).
- Note that **YOUR CODES WILL BE CHECKED WITH THE PLAGIARISM TOOLS!**