California State University, Sacramento
College of Engineering and Computer Science

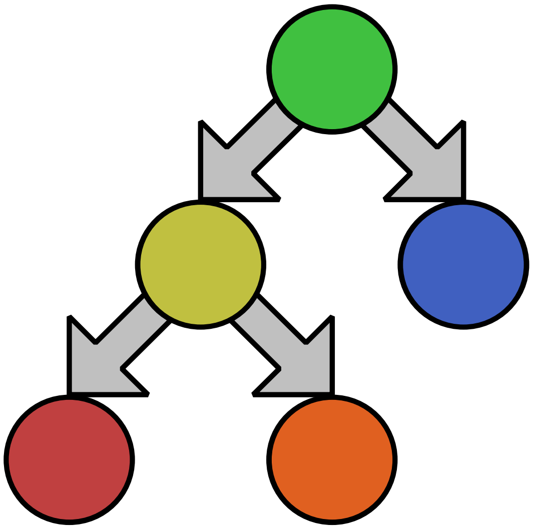
Computer Science 130: Data Structures and Algorithm Analysis

Assignment #3 – Binary Search Tree

Overview

For this assignment, you are going to create a Binary Search Tree (BST) with a minimal interface. You don't have to balance the tree. In fact, don't even try it yet. We are creating a very basic tree class.

For the sake of testing, we are going to restrict this tree to use Integers rather than the generic Object class. Like all programming assignments, work on it in parts.



Part 1: Node Class

Overview

In recursively defined structures, like trees, all the coding (and complexity) is found in the recursive structure itself. In the case of trees, the node will contain the vast amount of the logic and behavior.

The node will use a Key-Value system. The key is used to store and find nodes. The value field is completely passive.

Interface

public class Node		
Node	**left**	
Node	**right**	
int	**key**	The key. This key will be used to store the node into the tree.
string	**value**	The value that the node contains.
void	**print(int indent)**	This method will the structure of the tree. One node will be printed per line. You should use preorder tree traversal. Feel free to redirect the stream if you like. Please see the pseudocode below.
void	**add(int key, String value)**	Adds the key to the correct position in the BST. If the key already exists, do nothing.
string	**find(int key)**	Finds a node with the key and returns its value. If the node is not found, you can return either null or the empty string.

Pseudocode

```
Class Node
    Node left
    Node right
    int key
    string value

    ... All your methods go here
End Class
```

Adding a Key

To add a key, you will write a recursive method that will either recurse to the left or right – depending on the key. Whenever it can no longer recurse left or right, a new node is simply added.

```
Method add (int newKey, string newValue)
    If this node's key < newKey then
        If left isn't null
            left.add(newkey, newValue)
        else
            create a new left child for this node.
        End If
    End If

    If this node's key > newKey then
        If right isn't null
            right.add(newkey, newValue)
        else
            create a right left child for this node.
        End If
    End If
End Method
```

Printing the Tree

To print the tree, you will use also use recursion. In the examples below, I'm using spaces followed by a string of "**+--**" for readability. You can use spaces, dashes, etc… You can use any size of indentation you like (2, 3, etc…).

```
Method print (int indent)
    Print spaces for indent (3 or 4 times the indent)
    Print "+--" or some sort of text that looks nice.
    Print key, value, and a newline

    left.print(indent + 1)
    right.print(indent + 1)
End Method
```

Part 2: BinarySearchTree Class

Overview

The main class will contain virtually node code. Instead, this class is used to start recursion on the root node. All key/values are added to the using the add() method. It will find the correct position in the tree and store it there.

Interface

Just like before, the BinarySearchTree's methods will start recursion on the Node class.

public class BinarySearchTree		
string	**about()**	Returns text about you – the author of this class.
void	**print ()**	Starts recursion from the root node.
void	**add(int key, string value)**	Adds the key to the correct position in the BST. If the key already exists, do nothing. So, basically, you are creating a proper-set of numbers.
string	**find(int key)**	Finds a node with the key and returns its value. If the node is not found, you can return either null or the empty string.

Pseudocode

```
Class BinarySearchTree
    private Node root

    ... All your methods go here
End Class
```

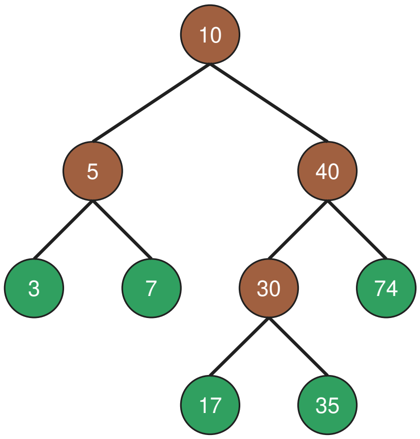
Some Examples

For example, if the following numbers are added to the Binary Search Tree.

10, 40, 30, 5, 74, 7, 17, 35, 3

It will result in the following tree. I've displayed the tree in both the text (using the Print method) and graphical version.

```
+-- 10
  +-- 5
    +-- 3
    +-- 7
  +-- 40
    +-- 30
      +-- 17
      +-- 35
    +-- 74
```

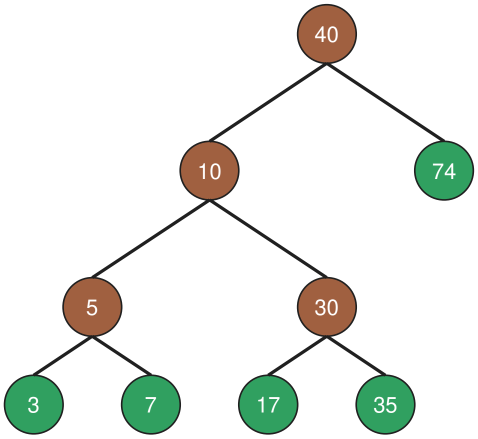


Binary Search Trees are extremely sensitive to the order that data is fed into them. In fact, once node is added, it's position in the tree will never change. In the example below, I've added the same numbers, but I have switched the position of the first two.

40, **10**, 30, 5, 74, 7, 17, 35, 3

Observe that, this minor change of order, has had a profound impact on the structure of the tree. The first key added will always become the root. And it will remain the root.

```
+-- 40
  +-- 10
    +-- 5
      +-- 3
      +-- 7
    +-- 30
      +-- 17
      +-- 35
  +-- 74
```



Part 3: Testing

Once you have finished your code, you need to test it using some good test data. Now you can see why you wrote the PrintTree method. It is vital to seeing if your methods are working correctly.

For example, the following is a basic demonstration of the add() method.

```
BinarySearchTree tree = new BinarySearchTree();


tree.add(10, "Buffalo Wings");
tree.add(43, "Cheez Whiz");
tree.add(18, "Root beer");
tree.add(6,  "Pringles");
tree.add(50, "Ice Cream");
tree.add(8,  "Chocolate");


tree.printTree();
```

Should produce the following tree:

```
+-- 10:  Buffalo Wings
  +-- 6: Pringles
    +-- 8: Chocolate
  +-- 43: Cheez Whiz
    +-- 18: Root beer
    +-- 50: Ice Cream
```

Requirements

- This **must** be completely all your code. If you share your solution with another student or re-use code from another class, you will receive a zero..

- You may use any programming language you are comfortable with. I strongly recommend not using C (C++, Java, C#, Visual Basic are all good choices).

> ⚠
>
> **You must use your linked-list and queue (modified) from Assignment #1.**
>
> **Do not use any built-in queue library class, etc… If you do, you will receive a zero.**
> **No exceptions. No resubmissions.**

Due Date

Due **November 4, 2021** by 11:59 pm.

Given you already have developed excellent programming skills in CSc 20, this shouldn't be a difficult assignment. **Do not send it to canvas.** E-Mail the following to dcook@csus.edu:

- The source code.

- The main program that runs the tests.

- Output generated by your tests

> ⚠
>
> **The e-mail server will delete all attachments that have file extensions it deems dangerous. This includes .py, .exe, and many more.**
>
> **So, please send a ZIP File containing all your files.**