California State University, Sacramento
College of Engineering and Computer Science

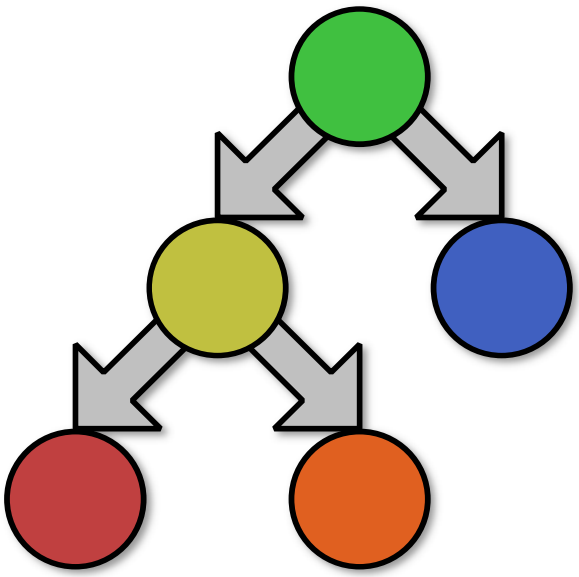
Computer Science 130: Data Structures and Algorithm Analysis

Fall 2022 – Project 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: Key-Value Class

To store the values in the array, we need to create a key-value class. In this case, I called it "Entry", but you are welcome any other name that you like.

```
class Entry
   public int key
   public String value
end class
```

Part 2: BinarySearchTree Class

Interface

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

public class BinarySearchTree		
	BinarySearchTree()	Constructor.
string	**about()**	Returns text about you – the author of this class.
void	**print()**	Start recursion from the root node. Note: It starts with an indent of 0.
void	**add(Entry entry)**	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 the Entry's value. If the node is not found, you can return an empty string.

Pseudocode

```
class BinarySearchTree
    private Node root
    ... All your methods go here
end class
```

Part 3: Node Class

Interface

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.

Create a very basic node class. All this requires is a left and right link (to another node) and a generic data field. It should also have a few constructors. In particular, you need one that will create a node with links to two other nodes.

public class Node		
	Node(Entry entry)	Constructor that assigns the value and key.
Entry	**value**	The value that the node contains.
Node	**left**	
Node	**right**	
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(Entry entry)**	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 an empty string.

Pseudocode

Note, this is far different from the Node class used by the linked list. Your Node class should be modified as follows:

```
class Node
    public Entry value
    public Node left
    public Node right

    ... All your methods go here
end class
```

Adding an Entry

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 (Entry entry)
    if entry.Key < this node's key then
        if left isn't null
            left.add(entry)
        else
            create a new left child for this node.
        end if
    end if

    if entry.Key > this node's key then
        if right isn't null
            right.add(entry)
        else
            create a right child for this node.
        end if
    end if
end method
```

Printing the Tree

To print the tree, you will use recursion using a **preorder depth-first traversal**. Don't worry, it is not hard.

```
method Print (int indent)
    Print spaces for indent
    Print the "+---"
    Print this.value and a newline
    left.Print(indent + 1)
    right.Print(indent + 1)
End Function
```

Part 4: Input File Format

A number of test files will be provided to you for testing your code. The format is designed to be easy to read in multiple programming languages. You need to use the classes, built in your programming language, to read the source files.

The first line of the data contains the total digits in the key. You should save this value and use it to control the number of passes the Radix Sort will perform. Naturally, this can also be inferred from the data itself, but it's useful to have it explicitly stated. The records end with an entry called "END".

```
Digits in the key
Key 1, Value 1
Key 2, Value 2
...
Key n, Value n
END
```

The following is one of the most basic test files on the CSc 35 Website. It consists of only 13 records.

File: years.txt

```
4
2020,Great Toilet Paper Shortage
1839,Sutter's Fort founded
1846,Bear Flag Revolt
1947,Sacramento State founded
1977,Star Wars was born
2017,Star Wars franchise almost destroyed
1964,Buffalo wings invented
1783,United States Constitution enacted
1850,California joins the United States
1848,Gold Rush begins
1776,American Revolution
1980,Pacman was released
1953,Cheese Whiz was invented (Nacho Era began)
END
```

13 records

Part 5: 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, if the following file is added to the Binary Search Tree.

File: halloween calories.txt

```
2
73,M&M's Fun size
60,Tootsie Pop
40,Starburst Fun Size
70,Kit Kat Snack Size
30,Laffy Taffy
80,Snickers Fun Size
50,Nerds Mini Box
77,Hershey's Milk Chocolate Fun Size
82,Almond Joy Snack Size
END
```

It will result in the following tree.

```
+--- 73: M&M's fun size
   +--- 60: Tootsie Pop
      +--- 40: Starburst Fun Size
         +--- 30: Laffy Taffy
         +--- 50: Nerds Mini Box
      +--- 70: Kit Kat Snack Size
   +--- 80: Snickers Fun Size
      +--- 77: Hershey's Milk Chocolate Fun Size
      +--- 82: Almond Joy Snack Size
```

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 entries, but I have switched the position of the first two.

File: halloween calories 2.txt

```
2
60,Tootsie Pop
73,M&M's Fun size
40,Starburst Fun Size
70,Kit Kat Snack Size
30,Laffy Taffy
80,Snickers Fun Size
50,Nerds Mini Box
77,Hershey's Milk Chocolate Fun Size
82,Almond Joy Snack Size
END
```

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.

```
+--- 60: Tootsie Pop
   +--- 40: Starburst Fun Size
      +--- 30: Laffy Taffy
      +--- 50: Nerds Mini Box
   +--- 73: M&M's fun size
      +--- 70: Kit Kat Snack Size
      +--- 80: Snickers Fun Size
         +--- 77: Hershey's Milk Chocolate Fun Size
         +--- 82: Almond Joy Snack Size
```

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 **must** use recursion in the Node class. The BinarySearchTree only starts recursion on the root.
- 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).
- Proper style.

Grading

1	Correct use of recursion – it must happen in the Node class.	20%
2	Correct interfaces	10%
3	Correct print method	*20%*
4	Correct add/find	*20%*
5	Proper Style	*10%*
6	Reading from the test file.	*20%*

Due Date

Due **November 17, 2022** by 11:59 pm.

Given you did a good job on the Tree Evaluator, then 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.

Proper Style

Well-formatted code

Points will be deducted if your program doesn't adhere to basic programming style guidelines. The requirements are below:

1. If programming C++, Java, or C#, I don't care where you put the starting curly bracket. Just be consistent.
2. Use the proper naming convention for your programming language. The interfaces, above, are using Microsoft's C# standard, but it's different in other languages.
3. Indentation must be used.
4. Indentation must be consistent. Three or four spaces works. **Beware the tab character**. It might not appear correctly on my computer (tabs are inconsistent in size).
5. Proper commenting. Not every line needs a comment, but sections that contain logic often do. Add a comment before every section of code – such as a loop or If Statement. Any complex idea, such as setting a link, must have a comment. Please see below.

The following code is well formatted and commented.

```
void foo()
{
    int x;

    //Print off the list
    x = 0;
    while (x < this.count)
    {
       items.Add(this[x]);  //Add the item to the temporary list
       x++;
    }
}
```

Poorly-formatted code

The following code is poor formatted and documented.

```
void foo()
{
    int x;

x = 0;
while (x < this.count)   {
     items.Add(this[x]);  //Call add on items.
       x++;
}
}
```