Homework 4

CSE 274

Deadline 11/08/2022

A company has just started operating. As time goes by, the company might start doing business in different states of the country, or might stop doing business in some states. We need a Map data structure to keep a record of the number of employees the company has in each state of the country that the company is doing business in. In this Lab, the keyword “states” refers to the states of the country. In response to the above need of the company we develop a map data structure using trees. Such a data structure is called a TreeMap.

Import the `Application.java` file in Eclipse, run the application and make sure that you receive no error.

I. The TreeMap Class

A tree map is a Binary Search Tree (BST), where each node of the tree has a `key` variable and a `value` variable. In the above example, `key` variables store the states of the country where the company does business in, and the `value` variables store the number of employees in those states. Accordingly, the `Node` class is as follows:

```
class Node {
String key;
int data;
Node down;
Node up;}
```

The `TreeMap` class has the following method to insert new nodes to the TreeMap:

```
 public void add(String key, int data)
```

The above method uses the same logic in BSTs for inserting new nodes to the TreeMap. Especially, in placing a new node in the TreeMap, the `key` variable of the new node is compared with the `key` variables of the existing nodes in the TreeMap. In the above example, `keys` are `String` variables. Since Java does not have > and < operator for comparison of `Strings`, the `TreeMap` class uses the following method to compare the `keys`:

```
public boolean IsGreaterThan(String key1, String key2)
{
if (LetterToNumber(key1.charAt(0))>LetterToNumber(key2.charAt(0)))
      return true;

if (LetterToNumber(key1.charAt(0))<LetterToNumber(key2.charAt(0)))
      return false;

if (LetterToNumber(key1.charAt(1))>LetterToNumber(key2.charAt(1)))
      return true;

if (LetterToNumber(key1.charAt(1))<LetterToNumber(key2.charAt(1)))
      return false;

return false;}
```

In above, `LetterToNumber` is the same method that we saw in previous labs.
The `TreeMap` class has the following method to search for a `key` in the nodes of the TreeMap:

```
public int doMapping(String key) {
```

The above method uses the same logic in BSTs for searching the nodes. If there is a node having the given `key` in the TreeMap, the method `return` the `value` inside the node.
Also, the `TreeMap` class has the following method to delete a `node` from the TreeMap:

```
public int contain(String key) {
```

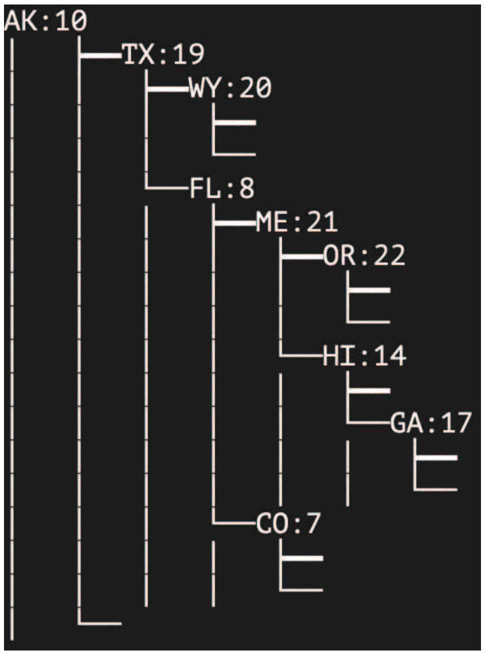
The above method uses the same logic in BSTs for deleting nodes.

Use the following lines of code to test the `TreeMap` class:

```
TreeMap myBinaryTree = new TreeMap();
myBinaryTree.add("AK",10);
myBinaryTree.add("TX",19);
myBinaryTree.add("FL",8);
myBinaryTree.add("CO",7);
myBinaryTree.add("ME",21);
myBinaryTree.add("LA",15);
myBinaryTree.add("CA",16);
myBinaryTree.add("HI",14);
myBinaryTree.add("GA",17);
myBinaryTree.add("OR",22);
myBinaryTree.add("WY",20);

myBinaryTree.delete("LA");
myBinaryTree.delete("CA");
myBinaryTree.display();
```

The expected output is printed below:



```
AK:10
│   ├──TX:19
│   │   ├──WY:20
│   │   │   ├──
│   │   │   └──
│   │   └──FL:8
│   │   │   ├──ME:21
│   │   │   │   ├──OR:22
│   │   │   │   │   ├──
│   │   │   │   │   └──
│   │   │   │   └──HI:14
│   │   │   │   │   ├──
│   │   │   │   │   └──GA:17
│   │   │   │   │   │   ├──
│   │   │   │   │   │   └──
│   │   │   └──CO:7
│   │   │   │   ├──
│   │   │   │   └──
│   └──
│
```

II. A Bug in `add` method

There is a bug in the `add` method of `TreeMap` class. Because of this bug, one cannot update the `value` variable of a node that already exists in the TreeMap. Fix this bug and use the following lines of code to test the `add` method:

```
TreeMap myBinaryTree = new TreeMap();
myBinaryTree.add("AK",10);
myBinaryTree.add("TX",19);
myBinaryTree.add("FL",8);
myBinaryTree.add("CO",7);
myBinaryTree.add("ME",21);
myBinaryTree.add("LA",15);
myBinaryTree.add("CA",16);
myBinaryTree.add("HI",14);
myBinaryTree.add("GA",17);
myBinaryTree.add("OR",22);
myBinaryTree.add("WY",20);
myBinaryTree.add("FL",17);

System.out.println(myBinaryTree.doMapping("FL"));
```

The expected output after fixing the bug is as follows:

```
17
```

III. Submitting the Assignment

Insertion, deletion and searching operations on TreeMap all have time complexities of O(log(n)). For a HashMap the time complexities of these operations are O(1), but one needs to design an ideal `HashFunc` for the HashMap to achieve O(1) time complexities.

When submitting your response to this homework, keep the above lines of code in the body of the `main` method.