Lab 12

CSE 274

Import the `Application.java` file into Eclipse, run the application and make sure that you receive no error. In this lab we study BSTs. Since a BST is still a binary tree, for creating BSTs we can use the same `BinaryTree` class that we developed in previous labs. The only difference is that we need to develop new `add`, `delete` and `contain` methods that keep the nodes of the BST sorted/organized.

I. The `add` Method

The `add` method of the `BinaryTree` class is used to add new values to the BST:

```
public void add(int value) {
      root = addRecursive(root, value);
 }
```

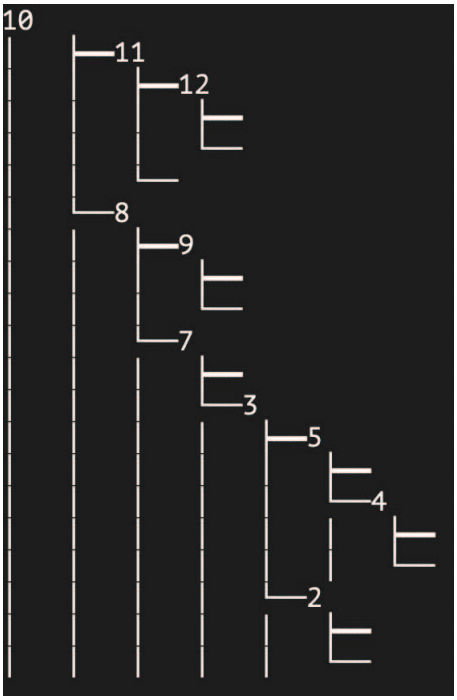
From above, the `add` method calls `addRecursive(root, value)` to add the new value to BST, **recursively**. Develop the `addRecursive` method for `BinaryTree` class using the following logic:

```
private Node addRecursive(Node current, int value)

      if (current == null)
            return new Node(value)

      if (value < current.value)
            current.down = addRecursive(current.down, value)
            return current

      if (value > current.value)
            current.up = addRecursive(current.up, value)
            return current

      if (value==current.value)
            return current

      return current
```

Notice that, in the above logic the last line is not needed logically as the last line is never executed. Still the last line should be included, otherwise Java compiler unnecessarily complains that the `addRecursive` method might not `return`. Use the following lines of code to test the developed method:

```
BinaryTree myBinaryTree = new BinaryTree();
myBinaryTree.add(10);
myBinaryTree.add(11);
myBinaryTree.add(8);
myBinaryTree.add(7);
myBinaryTree.add(3);
myBinaryTree.add(5);
myBinaryTree.add(4);
myBinaryTree.add(12);
myBinaryTree.add(9);
myBinaryTree.add(2);
myBinaryTree.display();
```

The expected output is printed below:



II. THE CONTAIN METHOD

The `contain` method of the `BinaryTree` class receives a `value` as an argument and checks whether the tree has a node with the given `value`:

```
public boolean contain(int value) {
      return containRecursive(root, value);
}
```

From above, the `contain` method calls `containRecursive(root, value)` method to check the existence of the given `value` in the BST, **recursively**. Develop the `containRecursive` method for the `BinaryTree` class using the following logic:

```
private boolean containRecursive(Node current, int value)
      if (current == null)
            return false;

      if (value == current.value)
            return true;

      if (value<current.value)
            return containRecursive(current.down, value);

      if (value>current.value)
            return containRecursive(current.up, value);

      return false
```

Notice that, in the above logic the last line is not needed logically as the last line is never executed. Still the last line should be included, otherwise Java compiler unnecessarily complains that the `containRecursive` method might not `return`. Use the following lines of code to test the developed method:

```
BinaryTree myBinaryTree = new BinaryTree();
myBinaryTree.add(10);
myBinaryTree.add(11);
myBinaryTree.add(7);
myBinaryTree.add(3);
myBinaryTree.add(5);
myBinaryTree.add(4);
myBinaryTree.add(12);
myBinaryTree.add(9);
myBinaryTree.add(2);

System.out.println(myBinaryTree.contain(7));
System.out.println(myBinaryTree.contain(8));
System.out.println(myBinaryTree.contain(12));
System.out.println(myBinaryTree.contain(2));
System.out.println(myBinaryTree.contain(4));
System.out.println(myBinaryTree.contain(14));
System.out.println(myBinaryTree.contain(17));
System.out.println(myBinaryTree.contain(6));
System.out.println(myBinaryTree.contain(1));
```

The expected output is printed below:

```
true
false
true
true
true
false
false
false
false
```

III. The `delete` method

The `delete` method of `BinaryTree` class receives a `value` as an argument and deletes the node that has the given `value`:

```
public void delete(int value)
{
if (root ==null)
      return;

root=deleteRecursive(root, value);
return;
}
```

From above, the `delete` method calls `deleteRecursive` method which performs several actions recursively to find and delete the node with the given `value`. A programmer has developed the following `deleteRecursive` method for `BinaryTree` class:

```
private Node deleteRecursive(Node current, int value)
{
      if (current == null)
      {
            return null;
      }

      if (value == current.value)
      {
            if (current.down==null && current.up ==null)
                  return null;

            if (current.down==null)
                  return current.up;

            if (current.up==null)
                  return current.down;

            int smallestValue = findSmallestValue(current.up);
            current.value = smallestValue;
            current.up = deleteRecursive(current.up,smallestValue);
            return current;
      }

      if (value < current.value)
      {
            current.down=deleteRecursive(current.down, value);
            return current ;
      }

      if (value > current.value)
      {
            current.up=deleteRecursive(current.up, value);
            return current;
      }

      return null;
}
```

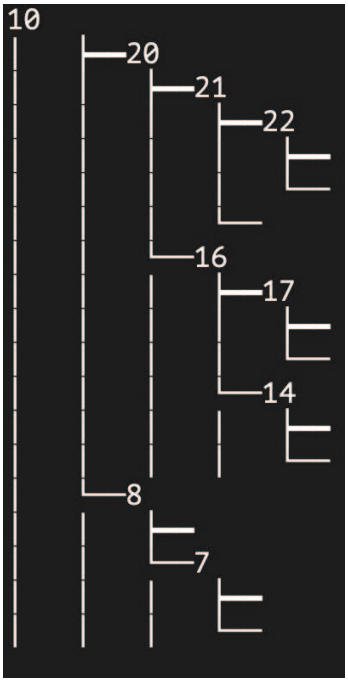
Notice that, in the above code the last line is not needed logically as the last line is never executed. Still the last line should be included, otherwise Java compiler unnecessarily complains that the `deleteRecursive` method might not `return`. The body of the `deleteRecursive` method uses the following method:

```
private int findSmallestValue(Node mynode)
```

The above method receives `mynode` as an argument and gives the smallest value in the subtree that starts at `mynode`. Develop the above method for the `BinaryTree` class. Use the following lines of code to test the developed method and the `delete` method:

```
BinaryTree myBinaryTree = new BinaryTree();
myBinaryTree.add(10);
myBinaryTree.add(19);
myBinaryTree.add(8);
myBinaryTree.add(7);
myBinaryTree.add(21);

myBinaryTree.add(15);
myBinaryTree.add(16);
myBinaryTree.add(14);
myBinaryTree.add(17);
myBinaryTree.add(17);
myBinaryTree.add(22);
myBinaryTree.add(20);

myBinaryTree.delete(19);
myBinaryTree.delete(15);

myBinaryTree.display();
```

The expected output is printed below:



IV. Submitting the Assignment

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