COMP 282 - Summer 2020

Homework 02 - Due: Sunday, 08/09/2020 by midnight

The purpose of this project is to introduce you to building and utilizing trees. It will consist of multiple parts, each corresponding to a lecture.

There are to be absolutely no packages used in your submissions. The use of the default package is, generally, not advised for enterprise-level work; this, however, is not that.

The following files **should** be present in your submission:

- Tree.java
- BinaryTree.java

You may include any interfaces described here, but they will not be examined. A successful project need only include the list of files mentioned above – with appropriate implementations, of course.

Part 1 - The Tree Class

In order to build more complex tree-based structures, you need to start with the basics. Namely, you will need to build a Tree class to store some arbitrary set of elements. We will extend this class throughout the rest of the project.

The basic interface definition is as follows:

```
public interface ITree<T> {
  public T getItem();
  public ITree<T> find(T item);
  public ITree<T> insert(T item);
}
```

This should be included in your project as ITree.java. You must provide an implementation for this interface in form of a Tree class – to be defined in Tree.java:

```
public class Tree<T> implements ITree<T> {
  // ...
  public Tree(T item) {
    // ...
  }
  // ...
}
```

This is the only file required from this part of the project.

Part 2 - Binary Trees

Now it's time to take the general implementation of a tree, and extend from it a binary tree. To do this, we will need to be able to only accept items that are ordinal:

```
public class BinaryTree<T extends Comparable<T>> extends Tree<Comparable<T>> {
  // ...
}
```

You will want to override the find and insert methods of your Tree class in order to ensure you are using this new tree as efficiently as possible. The BinaryTree.java file will be the only required file from this part of the project.

PART 3 - TRAVERSAL

In this part, we will supply four methods for traversing our tree structures. The goal is to implement the following interface in your BinaryTree class:

```
import java.util.*;

public interface ITraversable<T> {
  public ArrayList<T> nlr(); // Pre-order
  public ArrayList<T> lnr(); // In-order
  public ArrayList<T> lrn(); // Post-order
  public ArrayList<T> bfs(); // Breadth-first
}
```

Each method should return an ArrayList of node values, based on the appropriate traversal.

PART 4 - MEASUREMENT

In order to implement some more sophisticated trees, we will need an easy way of determining their heights. In order for to do this, we will implement another interface:

```
public interface IMeasurable {
  public int size();
  public int height();
  public int balance-factor(Node nd);
}
```

These should be fairly self descriptive: the size method will return the total number of elements in the tree, while the height method returns its height. **Remember: the height of a tree is the longest path from the root to any of its leaves.** The balance-factor method returns the balance factor of the node nd, you can call height() method inside balance-factor() method.

PART 5 - ROTATION

We have seen where the use of binary search trees can go wrong if we aren't careful about their inputs. In order to be more robust against these bad situations, we will implement left and right rotations in the BinaryTree class. In order to accomplish this, you will have to implement a new interface:

```
  public interface IRotatable<T> {
  public ITree<T> rotateLeft();
  public ITree<T> rotateRight();
  }
```

Each function will operate on the root node of its tree, and perform the appropriate rotation. The return value should be the new root of the rotated tree. The pseudocode the left rotation has been added at the end of this file.

Note: It is OK to implement this homework using other programing languages like C++ and Python. In addition, you can implement the above tree/binary tree in your own way. For example, you can implement only binary tree without implementing the tree. Or, you can define binary tree with integer keys (values).

INSTRUCTIONS for SUBMISSION

Submit EVERYTHING EVERY TIME. With each submission, including the current version of all work products to-date, even if incomplete, even if un-changed. This includes:

Source Code with complete explanation and comments

Executable program

Entire software project files if you use any IDE

Picture files that show you successfully compile your Java program with a set of output results.

Software Project files (example: if MS Visual Studio is used, include the ENTIRE project folder containing all project files – such as .vcxproj.)

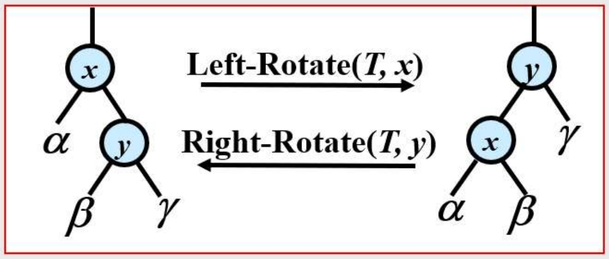
Note: All deliverables must be put in a folder, then Zip the folder and name it as HW02_your CSUN ID and submit it via Canvas.

If you plan to use GitHub, you can only send me the URL of your project. Make sure it is accessible.

LEFT ROTATION PSEUDOCODE:

```
LEFT_ROTATE(T, x)
    y = x.right
    x.right = y.left
    if y.left != NULL
        y.left.parent = x
    y.parent = x.parent
    if x.parent == NULL //x is root
        T.root = y
    elseif x == x.parent.left // x is left child
        x.parent.left = y
    else // x is right child
        x.parent.right = y
    y.left = x
    x.parent = y

    x.height = 1 + MAX(HEIGHT(x.left), HEIGHT(x.right))
    y.height = 1 + MAX(HEIGHT(y.left), HEIGHT(y.right))
```


x
y
α
β
γ
Left-Rotate(T, x)
Right-Rotate(T, y)
y
x
γ
α
β