Assignment 3 SU18 CS1027

Due: Friday July 27th at 11:55 pm via OWL

Your final assignment deals with a collection of related problems working with binary trees. It involves three tasks, which each build upon the solution of the previous task.

Total weight of Assignment 3: 15%

This assignment is to be done individually. You may talk to other students on a high level, but all work must be done independently. All assignments will be run through anti-plagiarism software. First offense penalty will be recorded as an act of academic dishonesty, and a mark of -100% will be awarded. You can also help students work on bugs in their code, but the solution is never to "just copy what I did, here look."

Learning Outcomes:

By completing this assignment, you will gain skills related to

- Recursion
- Binary Trees
- Tree Algorithms

Classes from Lectures you will need

- BinaryTreeADT.java
- LinkedBinaryTree.java
- BinaryTreeNode.java
- ElementNotFoundException.java
- EmptyCollectionException.java
- *Others as you want – you may use any you deem necessary!*

Provided Classes for this Assignment

We will test your code using files very similar to this. Ensure your solution works with this files unmodified.

- *TestPathToRoot.java*
- *TestLowestCommonAncestor.java*
- *TestShortestPath.java*

Task 1 [8%] : Find the Path To Root from a given tree node.

Here, you will be creating a recursive method which is a specialized kind of find. You will be given an element in the tree to find, and once found, you will need to create an iterator which can traverse the elements along the path from the found element to the root of the tree. So, you will find the "path to root" for the node given as a parameter.

This method will be added to the LinkedBinaryTree.java file, and has the following header:

```
public Iterator<T> pathToRoot(T targetElement)
                                throws ElementNotFoundException{
```

Hint: You will need to create a recursive helper method, `pathToRootAgain`, as you did with the `findAgain` method.

You can test this method with the TestPathToRoot.java file.

Task 2 [3%]: Find the Lowest Common Ancestor given two tree nodes.

The problem here is, given two nodes, where is the lowest ancestor in the tree -- where the root is considered the highest -- that the two elements have in common. You can use the path to root of each element to help find this.

This method will be added to the LinkedBinaryTree.java file, and has the following header:

```
public T lowestCommonAncestor( T targetOne, T targetTwo)
                                throws ElementNotFoundException{
```

You can test this method with the TestLowestCommonAncestor.java file.

Task 3 [4%]: Find the Shortest Path from one given tree node to another given tree node

For the last problem, you will find two elements, and return an iterator that can traverse the elements along the shortest possible path (following the connections that exist in the tree between parent and child nodes) to get from one given tree node to another given tree node. You can use both the path to root and lowest common ancestor in your solution.

This method will be added to the LinkedBinaryTree.java file, and has the following header:

```
public Iterator<T> shortestPath(T targetOne, T targetTwo)
                                throws ElementNotFoundException{
```

You can test this method with the TestShortestPath.java file.

Implementation notes

Exceptions

- Your code must correctly handle thrown exceptions, including the ones you have written.
- The methods you are writing will throw `ElementNotFoundExceptions` as appropriate

Command Line Argument

You must read the Tree file from the command line as `arg[0]`, as you did with the labyrinth file in Asn2.

Non-functional Specifications

- Your program has to be compilable under Eclipse.
- Use Javadoc comments for each class and method.
- Make comments on the major steps of your algorithm
- Use Java conventions and good Java programming techniques (meaningful variable names, conventions for variable and constant names, etc). Indent your code properly.
- Remember that assignments are to be done individually and must be your own work.

Classes to Submit to OWL

- LinkedBinaryTree.java with you new methods.
- Any other files required for your solution to run with the test files, other than the ones specifically listed above. We will add these to the run directory when we test.