Western University
Department of Computer Science
CS 1027 (Computer Science Fundamentals II)
Assignment 3

Professor: Ali Bou Nassif, Ph.D., P.Eng.

Instructions: Read carefully before you start.

This assignment should be done individually. The deadline is Wednesday July 22, 2015 at 11:54 pm. Please refer to the course outline (**Assignments Submissions and Late Assignments Policy Section)** for the late penalty. Each student must submit a zip file (zip and NOT rar) that compresses the main folder of his/her assignment (check the last paragraph of the assignment for more details). All files should work under Eclipse. It is very important that you comment your code thoroughly. Marks will be deducted if the code is not commented properly. The overall mark is 120 (20 bonus marks)

Description

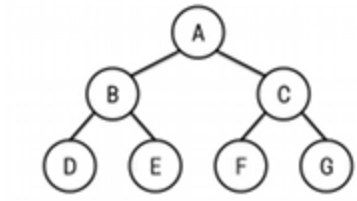
The purpose of this assignment is to get experience working with several topics presented in the course, including recursion, iterators, lists, binary trees and tree traversal.

Provided Files

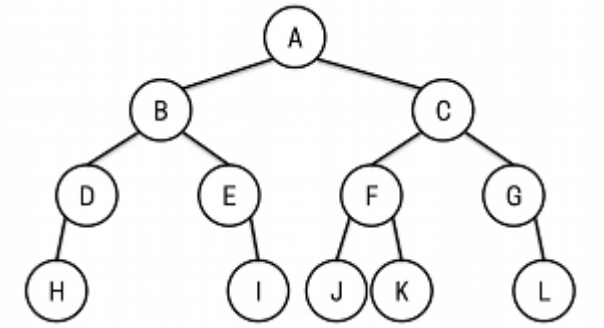
Asn3ProvidedFiles.zip
Assignment3_Tutorial (Includes helpful tips)

You are provided with a number of files to use for the assignment, including a TreeBuilder that builds LinkedBinaryTrees from text files written in a simplified Newick postorder format. There are two tree files provided: small_tree.txt and larger_tree.txt

Small Tree



Larger Tree



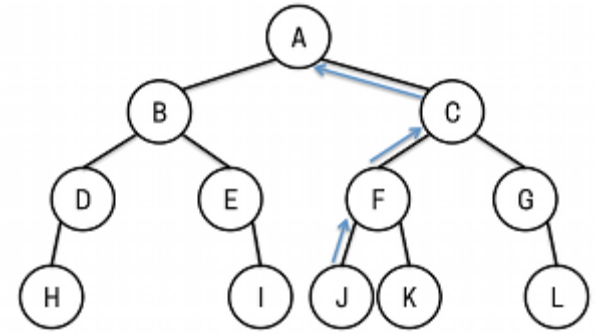
You must complete three methods in LinkedBinaryTree, and two programs TestPathToRoot.java and FindCommonAncestor.java We will separate this into two parts.

Part 1 - Path to Root

In the first part you will write a method, pathToRoot, that finds the path from any element in the tree to the root node. If the element is not in the tree, an exception must be thrown (an ElementNotFoundException).

This method will call a recursive method, pathToRootAgain (which you must also write), to actually find the element, and then add the elements on the path to a list (the one passed as a parameter to the recursive method) after it finds the element.

After the method executes, the iterator returned by the pathToRoot method must iterate over the elements found on the path from the target element to the root, *including* the target element and the root.



You will find that these methods have similarities to both the find method as well as the traversal methods in LinkedBinaryTree.java.

The headers of each method are present in the LinkedBinaryTree.java file that was provided to you. You will need to complete these methods.

You will also complete a program called TestPathToRoot.java, which will read a file from the command line, build a tree from the file using the TreeBuilder class, then, for all elements in the tree, print the path from that element to the root.

You must use an iterator to step through each element in the tree. You must also use an iterator to print each path.

This program has been started for you in the provided code.

Sample output for small_tree.txt

```
For element: D - the path to the root is: D B A
For element: B - the path to the root is: B A
For element: E - the path to the root is: E B A
For element: A - the path to the root is: A
For element: F - the path to the root is: F C A
For element: C - the path to the root is: C A
For element: G - the path to the root is: G C A
```

You must catch and print a useful message for the following exception types:

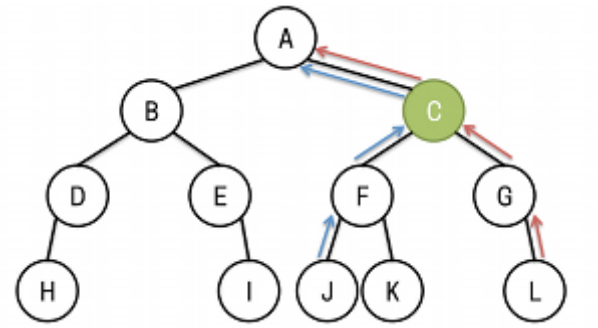
- Thrown by the buildTree method:
    - IOException
    - MalformedTreeFileException
- Thrown by the pathToRoot method:
    - ElementNotFoundException

Your program should also catch the exceptions as close to the possible source of the exception as possible, and try to recover. For example, if for some reason it can't find an element in the tree, it should inform the user, but then continue trying to go through the other elements.

Note: If everything is working properly, it should not be possible to throw the ElementNotFoundException, as you are iterating over elements you got from the tree structure.

Part 2 - Lowest Common Ancestor

Once you have completed Part 1, you can now use those methods to find the lowest common ancestor of two elements. In this part, you will complete the lowestCommonAncestor method which takes two elements as parameters, finds the paths from each element to the root (if the element is in the tree), and then finds the maximal-level node that is shared by these paths.



The method header has been given in the LinkedBinaryTree.java file. The approach you use is up to you. You can call either of the methods you wrote for the first part. If you are working with a ListADT type, you must use a "for each" loop to go through it, which takes advantage of the Iterable interface.

If either of the two elements passed as parameters cannot be found in the tree, you must throw an ElementNotFoundException.

For this part, you will also complete the program FindCommonAncestor.java.

This program will:

1. read a tree file from the command line
2. display the contents of the tree
3. ask the user for two string inputs
4. output the lowest common ancestor to the console, or an error message informing the user that the element could not be found in the tree

This program has been started for you in the provided code, with an example of how to prompt the user for information.

Sample runs with small_tree.txt

Example 1:

```
The tree contains: DBEAFCG
Enter first element: D
Enter second element: G
The lowest common ancestor is: A
```

Example 2:

```
The tree contains: DBEAFCG
Enter first element: F
Enter second element: G
The lowest common ancestor is: C
```

Example 3:

```
The tree contains: DBEAFCG
Enter first element: D
Enter second element: D
The lowest common ancestor is: D
```

You must catch and print a useful message for the following exception types:

- Thrown by the buildTree method:
  - IOException
  - MalformedTreeFileException
- Thrown by the readLine method:
  - IOException
- Thrown by the lowestCommonAncestor method:
  - ElementNotFoundException

For this program, these exceptions are all "fatal", in that we cannot continue if they occur. Here, you can use a single try/catch/finally structure for the bulk of the functionality, but make sure to clean up any resources you can in the finally block.

What to hand in

You will submit, via OWL:

- LinkedBinaryTree.java
- TestPathToRoot.java
- FindCommonAncestors.java

Functional Specifications

- Your program has to be compliable under Eclipse.
- Use Javadoc-style comments for each class and method. All significant variables must be commented.
- You do not need to submit the html files generated by running javadoc
- 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.