**Lab Assignment 1:**

**Binary Trees**

|  |
| --- |
| **Due Date**  **(a one-week LA)** |
| **Wednesday Labs: 02/03 11:59pm**  **Thursday Labs: 02/04 11:59pm** |

**Objectives**

* Recursion
* Interfaces
* Basic Trees

**Problem Specification**

For this lab assignment, you will be implementing a tree data structure. A tree structure starts from the top node, called the root. Each node in the tree has a set of children, which are also nodes of the tree that can have children of their own, and so on. This keeps on going till we get to the bottom of the tree, to the “leaf” nodes. Each node in the tree, except for the root, has a parent. For this assignment, you will implement a binary tree data structure by “implementing” the interface given. A binary tree enforces the requirement that a node can have at most 2 children.

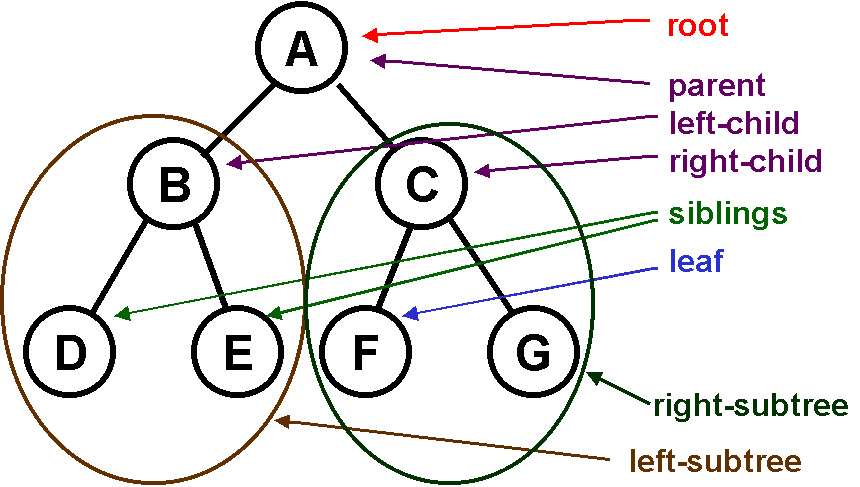
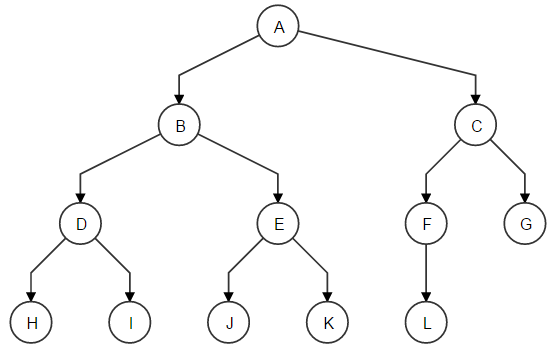


Figure 1 Image taken from www.mamtatutorial.com

**Input Data:** Our tree can be represented by the example below. Each line contains at most three characters. The first character is the ID of the node itself. The next two characters are the child nodes. If the line has only 2 characters, then that node contains only one child. If the line has one character only, then it is a leaf node with no children. To save you time, you can skip the file reading and use the provided main function directly.



*Example*

A B C

B D E

C F G

D H I

E J K

F L

G

H

I

J

K

L

Figure 2 an example of what it might like. The above is a complete binary tree, but it is not a requirement.

**Your tree does not have to be complete. It does not have to be full and it does not have to be balanced. The point of this assignment is to think recursively, and to learn about interfaces.**

**User Interaction**

User interacts with the program through the menu options. Here is the example of user interaction.

*1.Add Node*

*2.Tree Size*

*3.Find Node*

*0.Exit*

*->1*

*Please input the Node you want to add-> P*

*Please input the parent node of “P”-> F*

*Node successfully Added*

*///// Call printTree here to display the updated tree to the user.*

*1.Add Node*

*2.Tree Size*

*3.Find Node*

*0.Exit*

*->2*

*Please input the root node ->B*

*There are 7 nodes in that tree.*

*///// Counting will start from entered node. SO the size of the sub-tree from B is 7.*

*1.Add Node*

*2.Tree Size*

*3.Find Node*

*0.Exit*

*->3*

*Please input the node you want to look for -> X*

*Node X does not exist.*

*1.Add Node*

*2.Tree Size*

*3.Find Node*

*0.Exit*

*->0*

**Updated Tree Structure**

After the program terminates, you can see the original data has been updated since the tree has been changed. See the example below. You can see that the node P has been successfully added.

A B C

B D E

C F G

D H I

E J K

F L P

G

H

I

J

K

L

**Design** **Requirements**

***You are required to finish the design report before you leave the lab.***

Your lab report should contain the following two parts.

1. Basic Structure

List all the methods you plan to have for this project. Give each method a one-sentence description explaining what this method does.

The following methods are required. You can copy the following to your report.

**public static** **void** printMenu()

//Print the Interaction Menu to the screen.

You must also implement the following interface:

To do so, copy the code into an Interface file called INode.Java (Right Click Package Name -> New -> Interface -> Name=INode -> OK)

Copy the code in the box below to the file under the package declaration.

Save and add a new class (“TreeDataStructure”). Next to the box where it says Interfaces, click add, and search for INode. Add your Node interface and click ok.

Implement the functions in the class and add variables as needed. Think about how you will represent the children (hint: possibly an array of Nodes), how you will store the parent, and how you’ll walk through the tree.

The addChild function, the find function, the printTree function, and the size function must be implemented recursively. Everything else is up to you.

**package** edu.wmich.cs1120.spring16.interfaces;

**public** **interface** Node {

/\*\*

\* This method creates a new Node and

\* adds it to the children of the node it was called on

\* if its ID matches the parentID, otherwise it looks through

\* all its children

\* **@param** data A String containing the data

\* **@return** true if successful, false otherwise

\*/

**public** **boolean** addChild(String ID, String parentID);

/\*\*

\* Looks within the tree to find if value is contained

\* in that subtree. The node you use to call the find function

\* acts as the root of the tree.

\* **@param** value a string to be found in the tree

\* **@return** the node if found.

\*/

**public** Node find(String value);

/\*\*

\* Gets the parent of that node.

\* **@return** the parent node of the Node that is used to call this function.

\*/

**public** Node getParent();

/\*\*

\* Gets the size of the tree.

\* **@return** the size of the tree starting from the node that is used to call

\* this function, all the way down to the leaf nodes.

\*/

**public** **int** size();

/\*\*

\* Converts the data into a string and returns it

\* For example, Node A has 2 children B and C.

\* A.toString() will return A B C

\* **@return** String representation of the node.

\*/

**public** String toString();

/\*\*

\* Function to get the ID of the node.

\* **@return** String representation of the node ID

\*/

**public** String getId();

/\*\*

\* Prints the node upon which this method is called

\* as well as all the children node.

\* Use the toString() format to print.

\*/

**public** **void** printTree();

}

1. Pseudocode

Write pseudocode for all the required methods.

**Main function**

**public** **static** **void** main(String[] args) {

TreeDataStructure root = **new** TreeDataStructure("A");

root.addChild("B", "A");

root.addChild("C", "A");

root.addChild("D", "B");

root.addChild("E", "B");

root.addChild("F", "C");

root.addChild("G", "C");

root.addChild("H", "D");

root.addChild("I", "D");

root.addChild("J", "E");

root.addChild("K", "E");

root.addChild("L", "F");

root.printTree();

System.***out***.println(root.size());

/\*

\*

\* IMPLEMENT user options here.

\*/

}

**Additional Requirements**

**Coding Standards**

You must adhere to all conventions in the CS 1120 Java coding standard. This includes the use of white spaces for readability and the use of comments to explain the meaning of various methods and attributes. Be sure to follow the conventions for naming classes, variables, method parameters and methods.

**Assignment Submission**

* Generate a .zip file that contains all your files, including:
  + Program Files
    - including any input or output files