COMP 1406Z
Fall 2021 - Tutorial #5

Objectives

- To gain more practice using recursion.
- To write recursive methods for self-referencing data structures.

Getting Started:

Download the **T05-Start-Code.zip** file from the Brightspace website. Open the IntelliJ project contained in the zip file. This project contains the classes you need to start this tutorial.

Tutorial Problems:

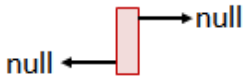
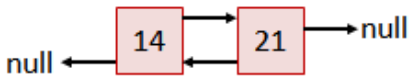
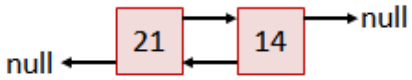
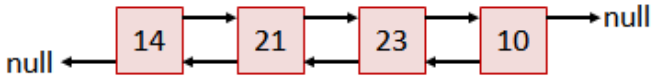
1) Consider the **LinkedList** class provided in the starting code.

Write the code for the instance method in the **LinkedList** class called **isInIncreasingOrder()** which returns a **boolean** indicating whether or not the data in the list is in increasing order.

i. The method must be recursive. It makes sense to have a directly recursive method that takes an **Item** from the list as a parameter and checks the remaining items after it.

ii. You will need to check to make sure that there is at least one **Item** in the list before calling the recursive method.

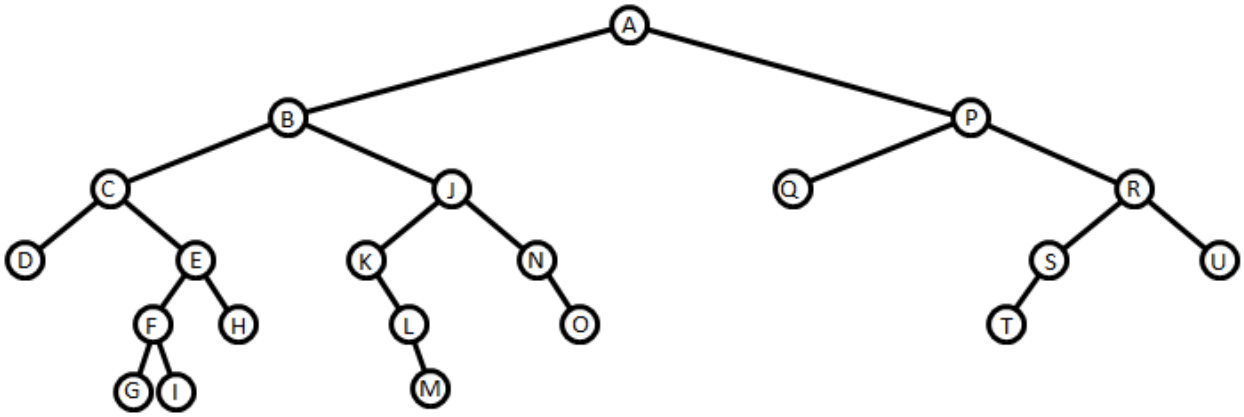
Below are some examples and the expected results. Use the **SortedListTestProgram** to test your results to make sure that your method works properly:

TRUE	
TRUE	
TRUE	
FALSE	
FALSE	
TRUE	

2) In the tutorial code provided, there is a **BinaryTree** class that represents the sentinel version of the binary tree data structure described in the notes and lecture (in the notes/lecture it is called **BinaryTree2**, whereas here it is simply called **BinaryTree**).

A **BinaryTreeTestProgram** class has been provided which creates the tree shown below and also prints out the height of the tree as well as the leaves of the tree.

A. Run the code to make sure that it works correctly.



B. Write a recursive instance method in the **BinaryTree** class called **contains(String d)** which returns a **boolean** indicating whether or not the tree contains the data passed in as a parameter. Note: this is not a search tree, so you will have to search all nodes in the tree before you can conclude the item is not present.

C. Add the following lines to the end of the **BinaryTreeTestProgram** to test your code:

```
System.out.println("Tree contains A = " + root.contains("A")); //true
System.out.println("Tree contains E = " + root.contains("E")); //true
System.out.println("Tree contains M = " + root.contains("M")); //true
System.out.println("Tree contains U = " + root.contains("U")); //true
System.out.println("Tree contains Z = " + root.contains("Z")); //false
```

---

3) In the tutorial code provided, there are classes called **Maze** and **Rat** that represent a rat moving through a maze. There is also a class called **MazeTestProgram** that allows you to test out a method called **canFindCheeseIn(Maze m)** which was discussed in the readings and lecture.

A. Run the code to make sure that it works. Try to understand the method if you are not sure what it is doing.

B. Add the following lines to the end of the **MazeTestProgram** code:

```
m = Maze.sampleMaze();
r.moveTo(1,1);
m.display(1,1);
System.out.println("Free Spaces = " + r.freeSpaces(m));
m.display(1,1);
```

C. Write a recursive method in the **Rat** class called **freeSpace(Maze m)** that determines the number of unique spaces that the rat can move around in from its starting location (i.e., (1,1) in this case).

Do not worry about marking the locations in the maze as being unvisited, simply leave them as being visited once the rat reaches that location.

Note that the method will be similar in structure to the **canFindCheeseIn(Maze m)** method but it must return *an integer* instead. The result for our test case should be **104**. Make sure that your code works.

Submission

Zip your completed tutorial #5 project and submit your **tutorial5.zip** file to the Tutorial #5 submission on Brightspace. Make sure you download and test your submission after you have submitted. Submitting a corrupt zip or a zip file that does not have the correct files will result in a loss of marks.