COMP 210 – Data Structures and Analysis (Sec 1&2)

Assignment #3 – Linked Lists

Issue Date: October 2nd, 2023.

Due Date: Monday, October 9th, 2023, 11:55pm

Marks: 5

In this Assignment, you will be required to write Java methods to fulfill some tasks related to Linked Lists. Before continuing, make sure you're caught up on the lectures on the topic. You will use what we have learned about Linked Lists in this assignment, and how to put them into practice using Java.

Please don't stress if something doesn't work — you should come to the office hours and/or post on Piazza! If you feel entirely stuck and unable to resolve your issue, email Qiwei Zhao: qiwei@cs.unc.edu and the LAs team will work together to help you!

Part A. Complete the Prerequisites

Complete the following requirements as done in the previous assignments.

1. Prepare the Java Develop Environment.

This step should already have been completed in the previous assignments. So, you should have already installed the required software such as Java, and IntelliJ.

2. Set up your java project.

- Launch IntelliJ IDEA.
- In your earlier created **COMP210** Project, right click on the src directory and select "New Package." Name your package "**assn03**".
- The starter code is attached as assn03.zip which you should unzip. You should now be able to see the 4 starter code java file. Copy these 4 java files in the assn03 folder of the above (COMP210) package.

At this point, you should have a minimum viable setup for assn03. For the remainder of the exercise, you are free to design your program however you choose, so long as it meets the specifications outlined in part B. This means that given an input of the form described in part B, it outputs the information as required in part B. You can define any additional helper methods or helper classes that you think might be useful, and there may be a variety of ways to solve this assignment.

Part B. Write necessary methods to implement various tasks.

1. Overview of the java files.

In this assignment your task is to complete a series of methods using linked lists. You are being provided with the following 4 .java class files, with some preexisting methods and some methods that you are to implement:

Main.java

This is the main user program.
It uses the **LinkedList** class.

LinkedList.java

This is the Linked List class that uses **NodeImpl.**
Methods already implemented:

- *size*
- *isEmpty*
- *clear*
- *contains*
- *toArray*
- *add*
- *remove*
- *get*
- *set*
- *add*
- *indexOf*
- *lastIndexOf*
- *validIndex*
- *getHead*
- *toString*

Methods to be implemented:

- *removeAtIndex*
- *isEqual*
- *removeRepeats*
- *reverse*
- *merge*

Node.java

This is an interface class for Nodes.

NodeImpl.java

This is an implementation of the Node interface.
Methods already implemented:

- getValue
- setValue
- getNext
- setNext
- hasNext

The Node class is how we represent an individual Node that makes up the List and the LinkedList class has a full implementation of the LinkedList. The **Main.java** class gives examples of how to construct a list, and uses various predefined methods such as ‘add’, ‘toString’, ‘sizeof’, ‘contains’, and ‘remove’ to demonstrate the functions that can be performed on linked lists. The ‘toString’ method is provided in LinkedList.java to easily display the contents of the list. You should execute the Main class and understand the results of the operations that have been implemented.

2. List of Tasks to be implemented.

The LinkedList methods that you need to implement (which generally increase in difficulty) are to be modified in the **LinkedList.java** file. **You should not be changing any of the completed methods at the bottom, and only edit the required methods at the top of this file.** You should also not modify the Node and NodeImpl java files. Before starting the solution, reviewing, and understanding the entire contents of this package will help you to complete the tasks required by this assignment.

An example of the input and expected output is also provided for each task. Note that for these algorithms you should NOT be creating a new list and returning it, you should only be manipulating the *list* object that the method is called on.

The tasks that you should fulfill are the following:

Task 1: Let's remove the Node at a given Index!

In the first task, you are asked to delete the node at a given index of the linked list. In this part you can assume that the values in Nodes are not duplicated. if the given index is greater than the size of the list, your program should be able to throw an IndexOutOfBounds exception using:

```
throw new IndexOutOfBoundsException();
```

Hints: You can traverse the list as done in methods such as 'contains', 'get', etc. Or you can simply use the *get* and *remove* methods already provided. (Make sure boundary conditions work, such as for the first element at index 0).

Example:

```
*    List: 1 -> 4 -> 2
*    i: 1
*    Return: List: 1 -> 2

*    List: 1 -> 4 -> 2
*    i: 0
*    Return: List: 4 -> 2

*    List: 1 -> 4 -> 2
*    i: 3
*    Return: IndexOutOfBoundsException()
```

Task 2: Are two Linked Lists equal?

In this task, you will be asked to complete the public boolean **isEqual**() method, which should be able to determine whether two linkedlists are identical. Return true if this linked list is equal to the list argument, false otherwise.

Hints: Two lists are equal if they have the same size, and the same elements in the same order. Use the *size* method to return the size of a linkedlist, and you can use *get* to return the value. (At this stage the complexity of your algorithm does not matter.)

Example:
- List: 1 -> 4 -> 2
- List2: 1 -> 4 -> 2
- Return: true

- List: 1 -> 5
- List2: 2 -> 5
- Return false

Task 3: Find redundant nodes! And remove them!

In this task, you will be asked to complete the public void **removeRepeats()**, which should be able to remove the duplicate values from a given sorted linked list.

Tip: First, this is a sorted list, and duplicate nodes will be together. We need to move forward along this linked list, and when we find that the values of adjacent nodes are equal, we remove the redundant nodes. Then this task can use your prior methods. At the same time, we need to pay attention to the special case when the size of the linked list is zero.

Example:
- List: 1 -> 2 -> 2 -> 2 -> 2 -> 3 -> 3 -> 3 -> 4
- List after removing redundant nodes: 1 -> 2 -> 3 -> 4

Task 4: Reverse

In this task, you will be asked to complete the public void **reverse**() method, which needs to make the input linked list reversed.

Tip: You might be able to use 3 adjacent node pointers (say p, q, r) to adjust the order of the adjacent nodes of the original linked list to get a reversed linked list. In addition, when the size of the linked list is 1, the flipped linked list is the same as the original linked list.

Example:

- List: 1 -> 2 -> 3 -> 4 (use p ->1, q -> 2, r -> 3, etc).

- Reversed List: 4 -> 3-> 2 -> 1

Task 5: Merge two lists together!

In this task, you will be asked to complete the public void **merge**(LinkedList list2) method, which can merge the given linked list into the current list. The list2 will always be either the same size, or shorter than the current list.

Tip: we can visit two linkedlists together, and then connect their nodes in turn. This processing should end until the tail node of list2 is visited.

Example:
- List: 1 -> 2 -> 3
- List2: 4 -> 5 -> 6
- Return: 1 -> 4 -> 2 -> 5 -> 3 -> 6

- List: 1 -> 2 -> 3 -> 4
- List2: 5 -> 6
- Return 1 -> 5 -> 2 -> 6 -> 3 -> 4

Submit to Gradescope for Grading

All that’s left now is to hand in your work on Gradescope for grading!

Before doing so, you need to know that before an assignment’s deadline you can resubmit work as many times as you need to without penalty. Portions of assignments are autograded and will provide near-immediate feedback. We want you to resubmit as many times as it takes you in order to earn full autograding credit!

Login to Gradescope and select the assignment named “Assignment 3 - Linkedlist” You’ll see an area to upload a zip file. To produce a zip file for autograding, return to IntelliJ.

Mac Users

Along the bottom of your window, you should see an option to open a terminal integrated into IntelliJ. Type the following command (all on a single line, including the ‘.’):

```
zip -r assn03_submission.zip assn03 .
```

(You need to zip the whole file, including src file)

In the file explorer pane, look for the zip file named “assn03_submission.zip”. If you right click on this file “Open in -> Finder” on Mac, the zip file’s location on your computer will open. Upload this file to Gradescope to submit your work for this assignment.

Windows Users

Navigate to your course workspace in a File Explorer window. Then right click on the src folder in your exercises directory and compress the directory into a zip folder. You can name it “**assn03_submission.zip**”. Please pay attention, the level of the folder needs to be the same as the description. Compression software on Windows may introduce additional folder structure, and the file naming must be the same.

When you upload it to Gradescope, please delete any files that showed up in the **src/** folder that were not actually part of **assn03**. Specifically, in this assignment, the files that need to be kept are **Linkedlist.java, Main.java, Node.java, and NodeImpl.java.**

Autograding will take a few moments to complete. For this exercise there will be no “human graded” component. If there are issues reported, you are encouraged to try and resolve them and resubmit. If for any reason you are not receiving full credit and are not sure what to try next, come give us a visit in the office hour!