COMP 2140 Assignment 2: Linked Lists

Cuneyt Akcora

Due: Monday, February 24, 2020 at 09:00 a.m.

Hand-in Instructions

Go to COMP2140 in UM Learn; then, under "Assessments" in the navigation bar at the top, click on "Assignments". You will find an assignment folder called "Assignment 2". Click the link and follow the instructions. Please note the following:

- In this assignment we have a driver class (Assignment2.java) and a structure class (LinkedList.java). You will implement methods in these two files. Do not rename the existing methods in classes. You will submit both of these files.
- As they stand, some methods are returning false. These returns must be rewritten by you to return results.
- Submit two .java files only. The .java files must contain all the source code that you wrote and the small report (in the comments at the end of the file). The first .java file must be named `A1<your last name><your first name>.java` (e.g., `A2AkcoraCuneyt.java`). The second file will be named LinkedList.java.
- Please do not submit anything else.
- We only accept homework submissions via UM Learn. Please DO NOT try to email your homework to the instructors or TAs or markers — it will not be accepted.
- We reserve the right to refuse to grade the homework or to deduct marks if you do not follow instructions.
- **Assignments become late immediately after the posted due date and time.** Late assignments will be accepted up to 49 hours after that time, at a penalty of 2% per hour or portion thereof. After 49 hours, an assignment is worth 0 marks and will no longer be accepted.

How to Get Help

Your course instructor is helpful: For our office hours and email addresses, see the course website on UM Learn (on the right side of the front page).

For email, please remember to put "[COMP2140]" in the subject and add a meaningful phrase after that to the subject, and to send from your UofM email account.

Course discussion groups on UM Learn: A discussion group for this assignment is available in the COMP 2140 course site on UM Learn (click on "Discussions" under "Communications"). Post questions and comments related to Assignment 1 there, and we will respond. We will also post questions related to the assignment that we receive by email, and answer them there. Please do not post solutions, not even snippets of solutions there, or anywhere else. **We expect you to read the assignment discussion group.**

Computer science help centre: The staff in the help centre can help you (but not give you assignment solutions!). See their website at http://www.cs.umanitoba.ca/undergraduate/computer-science-help-centre.php for location and hours. You can also email them at helpctr@cs.umanitoba.ca.

Programming Standards

When writing code for this course, **follow the programming standards**, available on this course's website on UM Learn. Failure to do so will result in the loss of marks.

Question

Remember: You will need to read this assignment many times to understand all the details of the program you need to write.

Goal: The purpose of this assignment is to write Java programs and implement various linked list algorithms. Then you will write a brief report describing your results, in the comments of your program; see the end of this document.

Code you can use: You are permitted to use and modify the code your instructor gave you in class for the various sorting algorithms, and code from Labs 1, 2 and 3. Make sure that you are using the LinkedList.java file that is uploaded for this assignment. Of course, you must use the code we provided to you below. You are NOT permitted to use code from any other source, You are NOT permitted to show or give your code to any other student. Any other code you need for this assignment, you must write for yourself. We encourage you to write ALL the code for this assignment yourself, based on your understanding of the algorithms — you will learn the material much better if you write the code yourself.

What you should implement: Implement the following algorithms and methods (you can add any necessary `private` helper methods):

1. **add():** In the LinkedList class implement an add method that takes a list2 and adds its values (not nodes) to the current list list1. Please consider these points:
   - Both lists are circular single linked list (As in Lab 2).
   - Both lists are ordered. Order of list1 must be preserved while adding values from list2.
   - The list2 should not change.

   The following three methods are designed to familiarize you with linked list improvements.

2. **convertCircularToOrdinary()** In the LinkedList class convert a circular list to an ordinary linked list (a single linked list without dummy header or trailer).
   - The existing linkedList.java class is implemented for a circular linked list.
   - This conversion will only change address pointers; we will not rewrite insert or search functions for an ordinary linked list.
   - When the function returns, the last pointer of the LinkedList class will point to the first node in the list, and the last node will have a null for the forward (next) node.

3. **convertOrdinaryToCircular()** In the LinkedList class convert an ordinary linked list to a circular linked list.
   - This conversion will only change address pointers.
   - When the function returns, the last pointer of the LinkedList class will point to the last node in the list, and the last.next will point to the first node of the linked list.

4. **reset()**: Reset the linked list to an empty circular link list.

5. **addDummies():** In the LinkedList class add a dummy header and a dummy trailer to the linked list. Use Integer.MIN_VALUE and Integer.MAX_VALUE as values.

6. **hasDummies()**: In the LinkedList class check if the linked list has a dummy header with Integer.MIN_VALUE and a dummy trailer with Integer.MAX_VALUE.

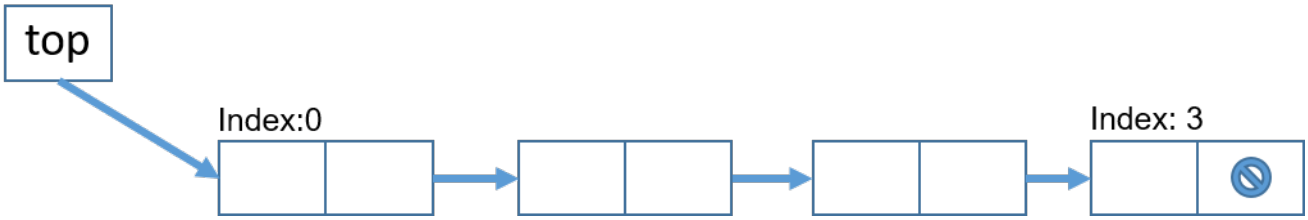

Figure 1: An ordinary linked list

7. **deleteOddNodes():** In the LinkedList class delete all nodes that have an odd item value.
   - Delete by reassigning pointers.
   - Do not create a new linked list.

   The following two methods are designed to show the efficiency of array operations when the data size is fixed.

8. **testInsertion():** In `A1<your last name><your first name>.java`, write a method to insert 'size' new integers into the list (for example, you can insert a dummy 1 every time), and record the time it takes to complete these insertions.

9. **testInsertion()(version 2):** In `A1<your last name><your first name>.java`, write a method to insert 'size' new integers into the array (you can insert a dummy 1 every time), and record the time it takes to complete these insertions.

   Reminder: These two insertion methods are overloaded; they have the same name but different signatures.

   (`https://www.geeksforgeeks.org/different-ways-method-overloading-java/`)

   The following two methods are designed to show the efficiency of Linked Lists when data size changes in time.

10. **insertValue()**: In `A1<your last name><your first name>.java`, insert the given value into the link list.

11. **insertValue() (version 2)**: In `A1<your last name><your first name>.java`, insert the given value into the array. If the array size is exceeded, you must create a new array and copy the old array into the new array. Afterwards, insert the new item value to the new array.

    The following two methods are designed to familiarize you with a well known linked list algorithm.

12. **corrupt()**: In the LinkedList.java write a method that corrupts a given ordinary linked list by changing the last node's forward pointer (which is supposed to be null) to point to a node at the given index. In Figure 1 a clean list is shown. In Figure 2, its corrupted version is shown. The task in `corrupt()` is to convert a clean linked list to a corrupted one. The position to corrupt is determined by the line

    ```
    index = r1.nextInt(list1.getSize()/2);
    ```

    which chooses the corruption index from the first half of the linked list.

13. **findCorruption()**: In the LinkedList.java write a method that finds if the linked list contains a loop.
    - In the main class, a sorted list is given as the parameter to this method (for ease of debugging). Do not use the sorted nature of the linked list to detect the corruption.
    - Do not store visited nodes to detect a loop. That would be a very lazy solution. If the linked list is big, you would store too much information.

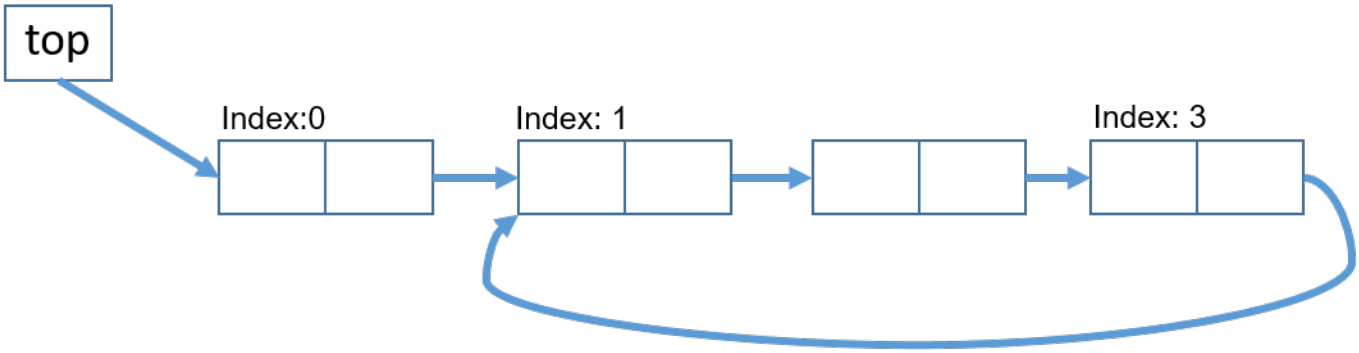

Figure 2: A corrupted linked list

Concerns.
It is not optimal to have hard-coded parameters in your class such as

```
final int list1Size = 1500;
```

Ideally, your main class should be passed an array of arguments. In this assignment, we are receiving list sizes (lis1Size=1500 and list2Size=1300) from the argument array `args` and parsing them to integers with the following lines.

```
int list1Size = Integer.parseInt(args[0]);
int list2Size = Integer.parseInt(args[1]);
```

Please avoid magical numbers in your code. There should be only ONE return per method. See the programming standards (under content/course documents) file on UMLearn, and follow the suggestions. Assignments will be graded by considering these standards. Your code must not give any warnings or errors when compiled and run.

Report

Write a small report in comments at the end of your program.
Answer the following questions (use short sentences, ideally only one sentence):

1. In value insertion to the array, how many times did your code create a new array?
2. In value insertion to the array, how many array elements are left unused after the very last insertion? This is the memory cost of using arrays.
3. When is it useful to store data in arrays vs linked lists?