CS2001 Practical 4 Report

Tutor: Michael Young

Matriculation ID: 210021783

Overview

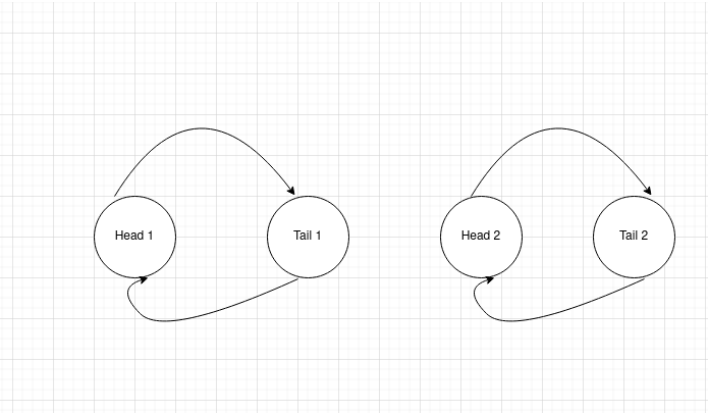
In this practical, we were tasked with implementing methods, both iteratively and recursively, for a program which manipulated doubly linked lists. I was successful in completing all the required functionality in this other than 2 methods.

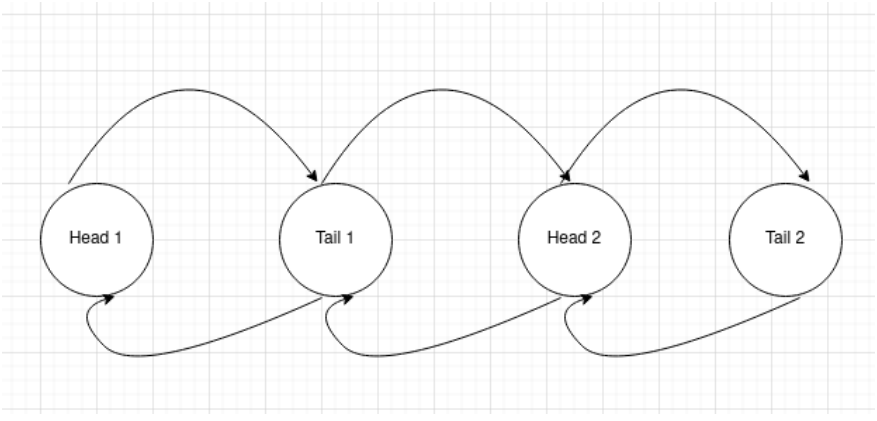
Design

A reoccurring theme for most of my methods in this class is that I used do while loops and slowly iterated through the ListNode temp until temp was equal to head again. I chose to use do whiles rather than for loops because although I could of use the size method to return the size each time, I found it would be more efficient reiterating through the ListNode Head as minimal times as possible. I also constantly check to see if the ListNode passed in is empty so that the program acts appropriately rather than returning something that it should not or just crashing.

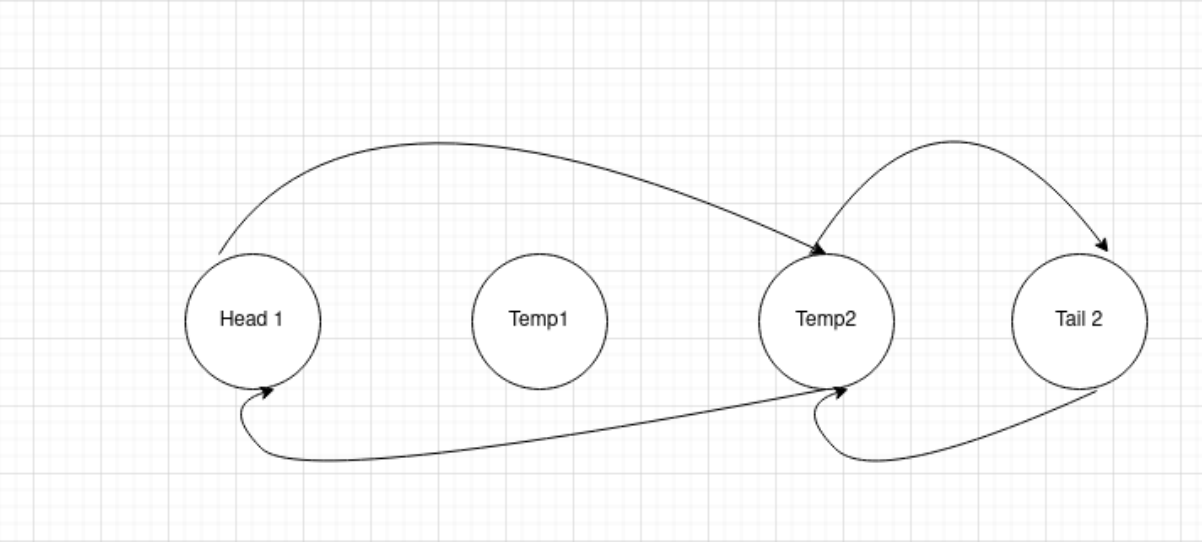
For all my methods in the recursive class that I did recursively, I chose to use two methods, the first one to validate the list and sometimes check the head with the second one being called to check each element. This is because it allowed me to essentially do what I would of done in the iterative process in the other class, like what I would of done in the do while.

My append method one of the ones I managed to do without using iteration or recursion, I managed to do this by essentially pointing the ListNodes to each other, this can be seen in the diagram below (before & after).

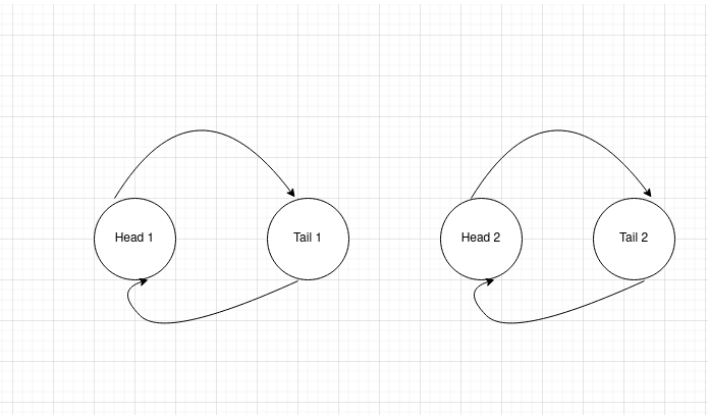


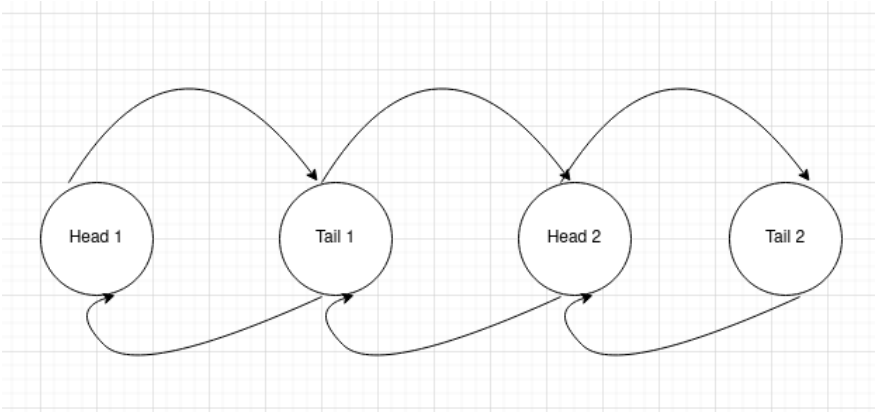


For my filter method, similar to append I did change the direction that the nodes were pointing however I made it so that the element to be removed had no nodes pointing to it hence how I removed it. Using Node.previous.next, this gets whatever the previous node's next is and I change that to Node.previous so I skip the current Node completely, I do the same to ensure that the next node is not pointing to it by setting temp.next.previous equal to temp.previous.



For my split method also managed to do that without using iteration or recursion, I did that by pointing the nodes to each other, as shown below. As you can see it is literally my append but in reverse.





Testing

I ensured all my methods passed all the tests given, however, when getting ready to submit I found I was not throwing something for my split method in recursive and I could not figure out what it was so this left me passing 36 out of 38 tests.

Other than doing the tests that we were given, my testing consisted of using different lists in the tests we were given, such as putting list 9 instead of list 6 in some of them and doing that I found no errors. However, this testing was not recorded and by the time my program was complete, it was too late to include proper through testing.

Evaluation

I concluded that my program is fit for purpose as it achieves all of the required functionality and is appropriately robust, although it could have been more secure, the specification and FAQ did specify that we should mostly assume that the input/node given is valid.

Conclusion

Overall, I have achieved all of the required functionality as my program:

- Achieves all required functionality iteratively and recursively, and without either where possible
- Successfully manipulates data within doubly linked lists without the use of standard Java collections

With more time I would have fixed my reverse method and recorded some individual tests so that I could check my program in a variety of situations.