CSCI 230 – Homework 3

Double Linked List

Objectives

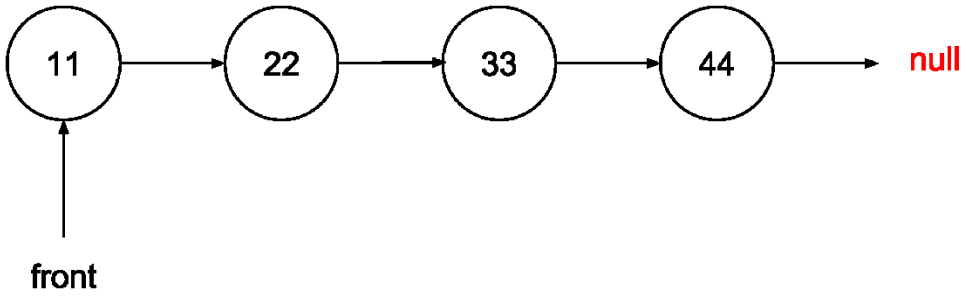
- Implement a DLL and its basic functionality.
- Test the DLL with detailed test cases.
- Analyze and compare DLLs vs SLLs.

Background information

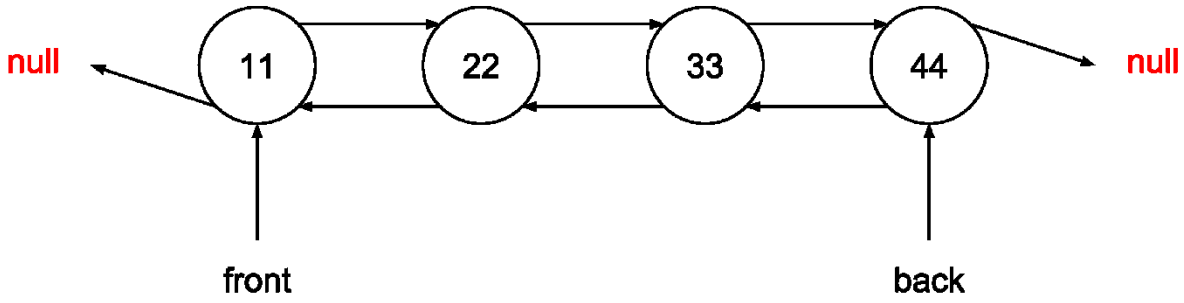
Ch. 10 – Linked Lists, Shaffer textbook

A doubly-linked list is a similar to the singly-linked lists, except in two crucial ways: your nodes now have pointers to *both* the previous and next nodes, and your linked list class has now have pointers to *both* the front and back of your sequence of list node objects.

Visually, the singly linked lists look like this:



Doubly-linked lists containing the same data will look like this:



Assignment

Your implementation should:

1. **Be generic** (e.g. you use generics to let the users store objects of any types in your list)
2. Implement the OrderedDList class based on the List interface in OpenDSA 3.1.1.
3. Methods to implement and additional instructions for some methods are given below:
    A. insert(E it) – Insert "it" at the proper location that ensures that the list is ordered. Note that the current pointer is not moved after an insert.
    B. remove() - Remove and return the current element if the element is in the list.
    C. clear(), moveToStart(), moveToEnd(), prev(), next(), length(), currPos(), moveToPos(), isAtEnd(), getValue(), isEmpty() – as described in OpenDSA
4. Methods not to implement: append(E it)
5. Be as asymptotically efficient as possible.
6. Contain exactly as many node objects as there are items in the list. (If the user inserts 5 items, you should have 5 nodes in your list).
7. Write a method indexOfMin that returns the index of the minimum item in a Double Linked List, assuming that each item in the list implements a Comparable interface.

Warning: correctly implementing a doubly-linked list will require you to pay careful attention to edge cases. Some tips and suggestions:

- Think carefully about the end cases (front and back) and what should happen when the list is empty or nearly empty.
- Write pseudocode for your methods before writing code. Avoid immediately thinking in terms of list node manipulation – instead, come up with a high-level plan and write helper methods that abstract your node manipulations. Then, flesh out how each helper method will work.

  Or to put it another way, figure out how to refactor your code **before** you start writing it. Your code will be significantly less buggy that way.

- Keep in mind the differences between objects and primitives (int, double, etc). This will come up in two ways: one, you'll need to remember that changing an object might change the reference in another place, and two, you'll need to remember to use == to compare equality for primitives and nulls, and use .equals() for object comparisons. Tip: you may use java.util.Objects to handle your equality checks with possibly-null values instead of juggling the == and .equals() yourself. Here's documentation for the relevant method.

What to submit:

- A zipped folder containing a folder named [LastName][FirstIntial]_HW[#], that contains your java files.
- One of the files must contain the **DoubleLinkedList** class in a file naturally named **DoubleLinkedList.java** The filename and class name must match exactly what is in bold.

- One class named **HW3** stored in a file named **HW3.java** that contains a main method that demos your ordered list and what works and specifies in comments what doesn't.
- Any other java files required of your solution. // There might not be any.
- Put **DoubleLinkedList.java and HW3.java (and any other java files needed) in a directory named YourLastNameFirstInitial_HW3. Then zip that folder. Then you will have a folder YourLastNameFirstInitial_HW3.zip. This is what you need to submit for HW3.**
- Name your folder: **YourLastNameFirstInitial_HW3.zip**. For me, this would be: MountrouidouX_HW3.zip.