CSCM12 – Coursework II
Due March 19th 11am

Cécilia Pradic

c.pradic@swansea.ac.uk

Guidelines

- By submitting this work, you state that you fully understand and are complying with the university's policy on Academic Integrity and Academic Misconduct. The policy can be found at `https://www.swansea.ac.uk/academic-services/academic-guide/assessment-issues/academic-integrity-academic-misconduct`.
- Use of "generative AI" to produce answers is forbidden.
- This is an individual assignment, and you must not collaborate with others or share solutions.
- In this assignment, you are given template java files defining a class implementing a doubly-linked lists with some functions missing. Your goal will be to implement those functions.
- For this coursework, you are **not** allowed to use import statements at the beginning of the file such as `import java.util.*;`
- You should **not** modify the type or name of the attributes and methods already in the file. You are however perfectly allowed to add your own methods if you want.
- There are 30 marks to be earned in total. Each of lab sheets 1-5 earns you 1 bonus mark (but 30 total is the maximum achievable). Following the submission instructions correctly is worth 4 marks, the questions are worth 56 marks.
- Submission instructions will be found at the end of this document.

Overview

The goal of this coursework will be to implement doubly-linked lists as described in the lectures, so as to familiarize yourself with programming with recursive structures in Java. To do so, we provide you with template files `Node.java` and `DLList.java` and a number of placeholder functions for you to fill in the latter one[1]. Then you should check that everything compiles without errors (i.e., `javac Node.java DLList.java` should terminate without errors or warnings) and upload `DLList.java` to autograder. Note that you should not upload `Node.java` to autograder; there is a local copy on the server identical to the one provided to you, so you should not modify it.

[1] and some dummy `return` statement that you should get rid of ultimately - they are there so that the code compiles even if it is unfinished.

I suggest that you test your code whenever you have finished writing a function so that you spend a reasonable amount of time debugging – to do so you should probaby create another file with a `main` function.

Before starting writing your code, please read the file `DLList.java` in full. The idea is that the class should ultimately implements doubly-linked lists with references at the front and the back in a principled way, as alluded to in the lecture.

Internally, essentially two kind of representations should be allowed during the run of any application that uses the `DLList<T>` class:

- either `first = last = null` and we consider the list empty
- or we have `first.prev = last.next = null` and for every other node n accessible from first and last, we should have `n.next.prev = n = n.prev.next` Furthermore, we should be able to reach last from first by taking `first.next. ... .next` a certain number of times. Note that in the particular case where we have a single element, we have `first = last`; otherwise, `first != last`

All of your functions should preserve those invariants by default. Additionally, this class has a length attribute that should correspond to the actual length of the list at all times. It is your responsibility to write your code so that this remains true when calling your method.

Some methods and constructors not mentioned in the handout are just there for your convenience if you want to run some tests on your code, which I strongly recommend before submitting!

Note that the classes `DLList` and `Node`, much like `ArrayList`, parameterized by the type of the labels. This uses a feature of java called **generic**. For the most part, since the signatures of the methods you have to implement are provided to you, you may code by pretending `T` is an abstract class given to you; if you want to understand things in more details, I strongly encourage you to take a read through `https://dev.java/learn/generics/`.

Questions (56 marks)

1. (8 marks) Write an implementation for the method

   ```
   public void push_back(T x);
   ```

   that adds the element `x` at the end of the list. This should run in constant time ($\mathcal{O}(1)$).

2. (8 marks) Write an implementation for the method

   ```
   public void push_front(T x);
   ```

   that adds the element `x` at the front of the list. This should run in constant time ($\mathcal{O}(1)$).

3. (8 marks) Write an implementation for the method

   ```
   public T pop_front();
   ```

   that removes the element `x` at the front of the list and returns it. This should run in constant time ($\mathcal{O}(1)$).

4. (8 marks) Write an implementation for the methode

   ```
   public void concatenate(DLList<T> xs);
   ```

   which takes an additional list `xs` as input and adds it at the end of the current list. You may break the invariant that `xs` is a correct doubly-linked list to do so. Your method should work in constant time $\mathcal{O}(1)$.

5. (8 marks) Implement a method

   ```
   public T get(int idx);
   ```

   which returns the ith element of the list without modifying it, assuming that we start counting at 0 (so that calling `get` from the list $[22, 1, 3, 5]$ with `idx = 2` returns 3). For full marks, your solution should run in $\mathcal{O}$(`idx`, `length - idx`).

6. (8 marks) Implement a method

   ```
   static public DLList<T> flatten(DLList<DLList<T>> xss);
   ```

   that takes as input a list of lists `xss` and concatenate all of the lists therein. For instance, calling `DLList.flatten(xs)` when `xs` corresponds to $[[1, 5, 2], [0, 2, 2], [], [10]]$ should return the list $[1, 5, 2, 0, 2, 2, 10]$. It is fine if your function breaks its input `xss` in such a way it is no longer a valid doubly-linked list afterwards, but the output should of course be a valid list. For full marks, your solution should run in $\mathcal{O}$(length of `xs`) (not the sum of the lengths of the elements of `xs`!).

7. (8 marks) Implement merge sort as a method

   ```
   static public DLList<Integer> mergeSort(<DLList<Integer> xs);
   ```

   that takes as input a list `xs` and returned a sorted version. It is fine if your function break its input `xs`. You function should run in $\mathcal{O}(n \log(n))$ and be space-efficient (i.e., it should be possible to run your algorithm with at most $\mathcal{O}(\log(n))$ space in addition to the input/output). You are **strongly** encouraged to introduce auxiliary methods to help you out with this question.

Submission instructions (4 marks)

- You submit your solutions by uploading your filled copy of the `DLList.java` file to `https://csautograder.swansea.ac.uk`. The submission page for this module will open from March 14th.
- Part of the grading will done automatically on the server by checking whether your implementations are correct on a large number of randomly drawn examples, but your submission will be also reviewed by a marker to allocate the remaining marks that are hard to check automatically (such as the fact that your solution are efficient).
- Before submitting, check that your file compiles as a stand-alone file. That is, issuing `javac Node.java DLList.java` should terminate without any errors or warning being issued. Something that often makes this go wrong is if students use an IDE that inserts a `package` directive at the top of a file - you should not have that. This is a necessary but not sufficient condition for your file to go through the grading setup on the server.
- Reminder that you are not allowed to import anything! If the system see you used the word `import` anywhere, it will simply reject your submission without testing it (it also means you should avoid this word in variable names).
- You will be allowed to submit your coursework multiple times, and you will get some automated feedback from the server; please try it out early to make sure it manages to process your submission

- Prior to the deadline, there is less automated feedback on questions 5 and 6, and no automated feedback for question 7; this is indentional and meant to encourage you to develop. You will get more feedback when the marks are released.
- For any other technical issues with autograder, please contact me. Dropping a message on the discussion board on canvas is the preferred option if your problem is of general interest.