CSCI 1933 Lab 11

Midterm II Review

Rules

To get credit for this lab, you must submit your answers to the questions below via Canvas by **11:55 p.m. on Wednesday, April** 15th. We will accept any form of document, whether it be pdf, jpg, png, txt, etc.; whatever works best for you. The solutions to the lab will be released after Wednesday. If you have any questions or would like feedback on your answers, we highly encourage you to either attend lab on Tuesday, attend one of the office hours throughout the week, or email the TAs.

> **Note:** The problems in this review lab are meant to reinforce the concepts you have learned so far throughout the course. Do not assume the layout, difficulty, length, or style of these questions are or are not similar to the questions on the upcoming midterm.

Lab Overview

As you all are hopefully already aware at this point, you have your second midterm this week. But if this is news, don't panic! This review lab is here to help you out! The problems are sectioned off as follows:

1. Linked List
2. Queues
3. Stacks
4. Exceptions
5. Complexity Analysis
6. Other Concepts

> **Note:** There is a topics list for the midterm posted here:

1 Linked List

These problems will require you to recall your `LinkedList` implementation in project 3. Remember that you will have limited time and resources during the midterm, so it is best to remember how `Node`'s work and how you implemented your `LinkedList` in project 3.

1.1 You get removed! You get removed! Every-two gets removed!

For this problem, implement `public void removeEvery(int n)` in your `LinkedList` class. This method should remove every n^{th} element from your list. You may assume that $n \geq 0$; if $n = 0$ or $n > size()$, do nothing.

Example: Suppose we have the following list: $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F$ and we call `removeEvery(2)`. We should end up with the following list: $A \rightarrow C \rightarrow E$.

1.2 Alright, break it up!

For this problem, implement `public LinkedList<LinkedList<T>> extractGroupsOf(int n)` in your `LinkedList` class. This method will return a list of lists, each of $size \leq n$. You may assume that $n \geq 0$; if $n = 0$ return an empty `List`.

Example: Suppose we have the following list: $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$ and we call `extractGroupsOf(2)`. The following list is returned: $\langle A \rightarrow B,\ C \rightarrow D,\ E\rangle$.

Constraint: You **cannot** use your `get(int index)` method or any method that gets a specific node. You must do this problem by traversing the links between the nodes in your list

2 Queues

These problems will require you to recall your `Queue` implementation from lecture. Remember that you will have limited time and resources during the midterm, so it is best to remember how `Queues` can be implemented using an `array` and `Nodes`

2.1 A Stacked Queue

You know how you've seen a `Queue` being implemented using an `array` and `Nodes`? Well, now you have to implement it using a `Stack`. Create a `StackedQueue` class and implement the following methods: `enqueue(T element), dequeue(), peek(), isEmpty()`. You can use any of the `Stack` implementations from the lecture examples or use Java's built in `Stack`.

Constraint: If you are using an `array`-based `Stack`, you **cannot** touch the underlying `array`. Similarly, if you are using a `Node`-based `Stack`, you **cannot** touch the underlying `Nodes`

2.2 Palindrome Check

A palindrome is a word that reads the same forward as backwards. Some examples are 'racecar', 'tacocat', and 'kayak'. Using a Queue, implement a `public boolean palindromeTest(String word)` method that takes in a word as a parameter and returns true if the word is a palindrome and false if it is not.

Constraint: Your **MUST** use a `Queue` as the underlying data structure. You can use Java's built-in `Queue`, one that you implemented yourself, or any of the ones in the lecture examples.

Constraint: If you are using an `array`-based `Queue`, you **cannot** touch the underlying `array`. Similarly, if you are using a `Node`-based `Queue`, you **cannot** touch the underlying `Nodes`

3 Stacks

These problems will require you to recall your Stack implementation from lecture. Remember that you will limited time and resources during the midterm, so it is best to remember how Stacks can be implemented using an array and Nodes

3.1 A Queued Stack

You know how you've seen a Stack being implemented using an array and Nodes? Well, now you have to implement it using a Queue. Create a QueuedStack class and implement the following methods:
push(T element), pop(), top(), isEmpty(). You can use any of the Queue implementations from the lecture examples or use Java's built in Queue.

> **Constraint:** If you are using an array-based Queue, you **cannot** touch the underlying array. Similarly, if you are using a Node-based Queue, you **cannot** touch the underlying Nodes

3.2 A Hack Stack

You know how some times you just want stuff from the middle of a pile but you can't take it because it will cause absolute mayhem, e.g. playing Jenga? Fortunately, we can take something from the middle of a pile with our handy-dandy HackStack. Create a HackStack class and implement the following methods:

- public void push(T element) – This just pushes the element onto the stack.
- public T pop() – This just pops the top element from the stack and returns it; if empty, return null.
- public T pop(int n) – This pops the n^{th} element from the top of the stack. You may assume that $n \geq 0$ where 0 corresponds to the top of the stack. If $n \geq$ the size of the stack, return null.
- public T peek() – This returns the item at the top of the stack.

> **Constraint:** Your **MUST** use a Stack as the underlying data structure. You can use Java's built-in Stack, one that you implemented yourself, or any of the ones in the lecture examples.

> **Constraint:** If you are using an array-based Stack, you **cannot** touch the underlying array. Similarly, if you are using a Node-based Stack, you **cannot** touch the underlying Nodes

4 Exceptions

4.1 From Bad to Worse

What is the difference between returning a value to signify bad input versus throwing an exception?

4.2 To Check Or Not to Check?

What is the difference between a checked and unchecked Java exception?

4.3 Exception Types

Name two subclasses of an `Exception` and give a situation in which they would be thrown.

4.4 Are You the Exception?

Given the following Java code, what exceptions could be produced? Explain your reasoning.

```java
public class MatchingStrings {
    public static boolean containsMatch(String[] array, String match) {
        boolean containsMatch = false;
        for (int i = 0; i < 10; i++) {
            if (array[i].equals(match)) {
              containsMatch = true;
            }
        }
      return containsMatch;
    }

    public static void main(String[] args) {
      //...
    }
}
```

4.5 Covering Your Bases (And Your Edges)

Suppose you have implemented your own version of `LinkedList`, as you did in project 3. You have all the required methods in place, your code is not producing any compile time errors, and given inputs that you expected, your code runs beautifully. Great! But you are not finished quite yet; it is time to test your code for edge cases. Identify at least three edge cases you should test for to ensure your code is functioning. (In particular, consider the add(int index, T element) and remove(T element) methods for LinkedList. What edge cases should you be certain to test?)

5 Complexity Analysis

5.1 Big-O

What is Big-O for the method `containsMatch(int[] array, int n)` in the following Java code? Explain your answer.

```
public class MatchingSubsequences {
    //checks int array for subsequences of n matching ints
    public static int containsMatch(int[] array, int n) {
        int countMatches = 0;
        for (int i = 0; i < array.length - (n - 1); i++) {
            boolean containsMatch = true;
            for (int j = i + 1; j < i + n; j++) {
                if (array[i] != array[j]) {
                    containsMatch = false;
                }
            }
            if (containsMatch) {
                countMatches++;
            }
        }
        return countMatches;
    }

    public static void main(String[] args) {
        int[] array = {2, 2, 2, 3, 4, 4, 4, 4, 4, 5, 5};
        int n = 5;
        System.out.println("The array contains " + containsMatch(array, n) +
                         " subsequence(s) of " + n + " matching ints.");
    }
}
```

5.2 More Big-O

What is Big-O for the method `add(T element)` in the following Java code? Explain your answer.

```
public class ArrayList<T extends Comparable<T>> implements List<T> {
    private T[] list;
    private int size;

    public ArrayList() {
        list = (T[]) new Comparable[2];
        size = 0;
    }
    private void doubleListLength() {
        T[] newList = (T[]) new Comparable[list.length * 2];
        for (int i = 0; i < size; i++) {
          newList[i] = list[i];
        }
        list = newList;
    }

    public boolean add(T element) {
        if (element == null) {
          return false;
        }
        if (size == list.length) {
          doubleListLength();
        }
        list[size] = element;
        size++;
        return true;
    }

    //...

    public static void main(String[] args) {
        //...
    }
}
```

5.3 Trade-Offs

When might you want to use an ArrayList over a LinkedList? What about a LinkedList over an ArrayList?

5.4 Show Some Improvement

How can you make add() in LinkedList operate in constant time O(1)?

6 Other Concepts

6.1 Definitions

What does a T indicate in Java? When would you want to use it?

6.2 Apples and Oranges, Abstracts and Inheritance

What is the difference between an abstract class and an interface? When might you want to use one over the other?

6.3 Making Comparisons

What is Comparable, what is Comparator, and how are they different? When might you want to use one over the other?

6.4 Equality

What is the difference between "==" and ".equals()"? When would you want to use one over the other?

6.5 Inheritance

Suppose there is an interface called `Vehicle`. The `Car` class and `Truck` class implement the `Vehicle` interface, but the `Bicycle` does not implement the `Vehicle` interface . Which of the following lines of code would cause a compile-time error?

1. Car accord = new Car();
2. Car lambo = new Vehicle();
3. Vehicle camry = new Car();
4. Vehicle f150 = new Truck();
5. Truck garbageTruck = new Truck();
6. Vehicle trainingWheels = new Bicycle();