Data Structures and Object Oriented Programming

Homework #6

HEAPS, PRIORITY QUEUES AND SCHEDULING

Project 6 is due February 20th 2017, at 23:59:59.

This project has one milestone. Please read the Submission section carefully for more details.

Grading: 150 points

Submit your files using Vocareum. Make sure that you follow directions on how to submit files precisely. Read the submission report after you submitted your files.

Contents

1 Heaps and Heap Tester

1.1 Preamble

Array-based binary heaps are one very efficient implementation of priority queues, with O(1) time findMin() and O(log n) time insert() and deleteMin(). Because the **branching factor** (the number of children a node can have) in a binary heap is 2, the base of the logarithm in O(log n) is 2 as well. **In this project you will implement a d-ary heap, where d is the branching factor of the heap**.

Let us consider why it might be a good idea to use d-ary heaps over binary heaps : The height of a complete binary tree containing 10,000 items is 13 ($\log_2 10000 = 13$), meaning that if the items are stored in a binary heap, insertions and deletions might take up to 13 swaps. If the same items are stored in a heap with branching factor 16, no more than 3 swaps will be required. The disadvantage of having 16 children for one node is that, for example, when percolating down you have to find the smallest of the children, and the cost of this operation grows linearly with the number of children

For a d-ary heap with root at index 0 of the array,

(a) the children of the node at array index j are at indices: **d*j +1 to d*j +d**

(b) the parent of a node at array index j (except the root) is at: **floor((j - 1) / d)**

1.2 Implementation

(a) You are given an interface dHeapInterface.java, which your dHeap class must implement.

(b) Your dHeap **must use a heap-structured array** as its backing store.

(c) Your heap will store data of a generic type T. Since heap relies on the order relation defined among elements, the heap will require that the type T implement the Comparable interface. Any particular T must satisfy the constraint that *T extends Comparable<? super T >*
Hence, when defining your class dHeap, you should use the following syntax to ensure that it compiles correctly:
public class dHeap< T extends Comparable <? super T >> implements dHeapInterface < T >

(d) To instantiate the array that you will use to implement a heap, you will need to downcast into the type T[ ] in a similar manner as you did for your MyQueue class in HW4. However, now you cannot just instantiate an Object[] array because the heap requires that the data all be Comparable; hence, instead you should instantiate the underlying array as:
T[ ] array = (T[ ]) new Comparable[123];
(The choice of 123 was arbitrary)

(e) Your dHeap must define a public default constructor which creates a binary max-heap, using an array of length 5 as its initial backing store.

(f) You must implement both max-heap and min-heap in the dHeap class. **However, I strongly suggest that you first write and test a max-heap implementation before beginning your min-heap implementation.** Two of your constructors will specify whether a max-heap or min-heap will be instantiated via a boolean value. (Refer to the starter code for more details).

(g) You will implement 5 methods in dHeap - size(), clear(), element(), add(T) and remove(). Since the implementation can be long and complex, we require that you write private helper methods for both add() and remove() in dHeap, to make the overall logic clear and easy to read. Think about what helper methods might be useful to define. **At minimum, you must have the following helper methods in your dHeap.java**:

trickleDown(int index)
bubbleUp(int index)
resize()
parent(int index)

Helper methods are not part of the public interface of your dHeap class and so they should have a private or protected visibility. However, you are allowed to have public getters for testing purposes.

(h) Create a dHeapTester that tests all the public methods in dHeap. Make sure you test your heap thoroughly for different cases such adding a null data, removing from an empty heap, adding large number of elements, alternately add and remove etc. Once you finish with part (i) below, make sure to test both heaps: max and min. (TIP: It is very easy to test an Integer heap. A min-heap will return all numbers in an ascending order while a max-heap will return them in a descending order). Also make sure your heap works for different values of d, such as d=2, 3, 4 and so on.

(i) Once you finish and test your max-heap, you can now make the modifications required for a min heap. You need to differentiate between a min-heap and max-heap. If you find yourself doing a lot of copying and pasting, your implementation may be very inefficient. Stop there and think again. We will deduct style points for an inefficient implementation.

Hints:

- Think about a helper method that wraps around compareTo to reduce the amount of code needed to differentiate between a min-heap and a max-heap.
- Then, think about how comparisons would need to change to turn this code into a min-heap. This is where a simple wrapper around compareTo can be useful. If you

approach this correctly, you should only need slightly recode existing comparisons in your bubbleUp() and trickleDown() methods to use a comparison helper method.

2 Priority Queue and Tester

In this part you will create a MyPriorityQueue class that uses the dHeap class to provide the priority queue operations defined as follows:

(a) *public boolean offer(T e)* : Inserts the specified element 'e' into this priority queue. Throws NullPointerException if the specified element is null

(b) *public T poll()*: Retrieves the head of this Priority Queue (smallest element), or null if the queue is empty.

(c) *public void clear()* : Removes all of the elements from the queue. The queue will be empty after this call returns.

(d) *public T peek()* : Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.

Your priority queue should a use **binary max-heap** for its implementation. (Hint: Adapter Design Pattern). Constructor for MyPriorityQueue takes one argument: the initial size of a queue. Tester file is not required for submission, but make sure to test your class methods as well, since we will use our own tester to check your work.

3 Course Registration Simulator

(Thank you Pooja and J. Skrentny for the assignment idea)

In this section, you will develop a Course Scheduler that uses a Priority Queue to enroll students into desired courses

Introduction

As CS enrollments continue to grow at a fast rate, getting the CS courses you want is becoming more and more challenging. The current registration system for a course is essentially a queue - first-come, first-served. Once a course fills, then you're put on a waiting list - another queue.

One of the problems with this system is that students have no ability to say which courses are most important to them. With your programming knowledge, you can make a substantial improvement to registration systems by giving students more input. You'll be implementing a modified registration system that has the potential to make students happier.

The New System

Our new system enables students to express the importance of courses they wish to take using *course coins*. Every student has a limited number of course coins. You can allot different amounts of course coins for each course you wish to take in your enrollment request. The more coins you allot, the higher your chances of getting that course. Like the current registration system, each course has its own registration queue. However, students who allocated more course coins are ranked higher. To get a better understanding of this system, let's develop our supporting classes and talk about details.

3.1 The Student Class

You will first create a Student class that holds all the basic information about a Student. The Student class must implement the Student_Interface provided in the starter files. Each student is associated with a Student ID, a name and the total number of course coins the student has. Refer to the starter code and the interface for more details.

Note: The Student class contains two List variables to hold the enrolled and waitlisted courses. You may use either Java's LinkedList or the ArrayList.

3.2 The Course Class

You will now create a class Course that holds all the information about a course. The Course class must implement the Course_Interface. Each course is associated with a course name, course code and a capacity i.e. the maximum number of students that can be enrolled in this course. Refer to the given starter code for more details. Implement all the methods except for *addToWaitlist* and *processWaitlist* for now. You may implement these methods once you understand the Registration class explained in the next section.

Once you read through the next section you can now finish the remaining methods of the Course class. The method *addToWaitlist* takes a Registration object as a parameter and adds it to the Priority Queue waitlist. The method *processWaitlist* removes the next Registration from the Priority Queue and enrolls the Student into the Course. Note that you must update the Course Roster as well as the Student's Enrollment List when you enroll the student into the course. Refer to the interface for more detailed description of these methods.

3.3 The Registration Class

A Registration refers to a registration request made by a Student for a particular course. Each registration is associated with a Student, a Course and the number of course coins the student is willing to offer for that course. A larger value for the coins indicates that the course is important to the student.

The Course class has a PriorityQueue of Registrations as an instance variable i.e. each course has its own priority queue of waitlisted registrations. Hence the Registration class must implement the Comparable interface and override the compareTo method so that objects of this class can be added to the dHeap. Refer to the starter code and complete the implementation of the constructor, getters and the compareTo method in this class. Now go back to Section 3.2 and complete the methods of the Course class.

3.4 Understanding the Input File

Before implementing the scheduling algorithm, let us look into the format of the input files. There are two files for your program -the Information file and the Input file. The Information file contains the Courses and Students information, and the methods to parse this file is already provided in the starter code.

The Input file (which your algorithm will parse) will have lines of two forms:

a) **register** PID CourseID CourseCoins
OR
b) **enroll** quantity

where

- **register** is a preset word that indicates that a new student is a about to register
- **PID** Student's PID
- **CourseID** CourseID of a desired course
- **CourseCoins** number of coins the student is willing to allot for this course
- **enroll** is also a present word and indicates that you are ready to start an enrollment process.
- **quantity** represents a number of students that you should enroll in each course. If the course queue is currently empty do nothing. There is also an enrollment limit per course (capacity) and once the enrollment exceeds the limit, you must not enroll students to this course anymore.

Input file example:

register A11111 CSE12 40
register A65545 CSE12 20
enroll 2

```
register A12345 CSE20 10
```

3.5 Implementation

Read till Section 3.8 before beginning your implementation

You will now develop your algorithm to enroll students into courses in the file **CourseScheduling.java**. Refer to the given starter code. Many helper methods are already provided to you to make implementation easier. The main method receives two command-line arguments - the Information file as args[0] and the Input file as args[1]. The method *populateCourseandStudents* parses the Information file and generates lists of Course and Student objects. Your algorithm must parse the Input file line by line and process each line as follows* :

- Each line beginning with **register** tells you to create a new Registration for a student and a course with the specified number of coins, and add it to the Priority Queue waitlist of that course.
- Before adding a registration to the waitlist, you must check if the student is already waitlisted/enrolled into the course and if so, you must not add the registration request again to the queue.
- You must also check if the student has the specified number of coins left to register for the course. If a student has less number of course coins left than what he/she wants to offer to the course, you must not add the student request to the priority queue.
- Once you add a Registration to a Course queue, you must also deduct the coins used from the total number of coins for that Student.
- Each line beginning with **enroll** tells you to start a batch enrollment process for all courses.
- The quantity tells you the number of students that must be enrolled in each course. You can run a loop, go through each course in the course list and call the appropraite method on each course.
- You must not enroll students into a course whose capacity is full
- If there are no registrations in the waiting list then do nothing.

***Refer to the Appendix section to see tips on File IO. For example,**

3.6 How to process a sample input file

Consider the input file as given below:

```
register A11111 CSE12 40
register A65545 CSE12 20
register A76548 CSE11 45
register A55555 CSE11 40
register A12345 CSE12 15
enroll 2
register A12345 CSE20 10
```

- Upon reading the line *register A11111 CSE12 40* your algorithm must make a Registration with Student with PID A11111, Course with code CSE12 and coins 40 and add it to the waitlist queue of CSE12. You can get the Student corresponding to A11111 and Course corresponding to CSE12 using the getCourse and getStudent methods you will implement.
- Note that you must first check if Student A11111 has already been enrolled/waitlisted in this course or if the student has 40 coins left to offer.
- Upon reading the line *enroll 2*, your algorithm must enroll 2 students into every course. If the course waitlist is empty do nothing. You must also check if the course has reached its capacity.

3.7 Output requirements

You have been given many print statements in the starter code. These may look redundant to you, but this is to ensure that your output **exactly** matches the expected output. You can call these print statements at appropriate points in your code to correctly match your output with the expected one.

Once the parsing completes, the main method makes a call to *generateReport* which prints a summary of all Students and Courses to the console.

3.8 Sample Input and Output

Consider the sample input below:

```
register A65545 CSE12 20
register A11111 CSE12 40
register A76548 CSE11 45
register A55555 CSE11 40
register A12345 CSE12 15
enroll 2
register A12345 CSE20 10
```

The complete output for this input is given in the file ConsoleOutput1.txt. Below is a snapshot of the scheduling part of the output:

```
####START COURSE SCHEDULING####

Waitlisting Marina(A65545) to Course CSE12 with coins 20
Waitlisting Pooja(A11111) to Course CSE12 with coins 40
Waitlisting Don(A76548) to Course CSE11 with coins 45
Waitlisting Max(A55555) to Course CSE11 with coins 40
Waitlisting Haoran(A12345) to Course CSE12 with coins 15

####STARTING BATCH ENROLLMENT####

Enrolling Pooja(A11111) to Course CSE12 with coins 40
Enrolling Don(A76548) to Course CSE11 with coins 45
Enrolling Marina(A65545) to Course CSE12 with coins 20
Enrolling Max(A55555) to Course CSE11 with coins 40
####ENDING BATCH ENROLLMENT####

Waitlisting Haoran(A12345) to Course CSE20 with coins 10

####END COURSE SCHEDULING####
```

The first five lines in the output correspond to the first five lines in the input starting with **register** Notice that when the batch enrollment starts, Pooja gets enrolled into CSE12 first as she offers the maximum number of coins for the course (40). Similarly Don gets enrolled into CSE11 first as he offers 45 coins which is more than what the rest of the students offer for this course. Since you implemented your Priority Queue using a max-heap, returning the Registration with max coins is already taken care of by your dHeap! A few more sample inputs and outputs are provided in the starter files.

3.9 Grading Part 3

We will be testing your code against a few test input files and verify the output. Please be aware that you will not receive any credit for a test if your program throws exceptions for it. I highly recommend that you test your code thoroughly against all examples provided, and also come up with your own test cases to test corner cases, as the given sample inputs may not be comprehensive.

4 Extra credit, 5 points each

1. Write a method to find the min value in a max-heap in your dHeap class. The algorithm should find and return the min value in the heap efficiently without modifying the heap. Hint: you do not have to scan the entire heap.

   public T findMaxinMin()

2. Write a method that returns all the occurrences of the values greater than a parameter 'k' in a max-heap. You can use built-in Java's Linked List.

   public LinkedList<T> findGreaterThanK(T k)

5 Submission

You have one milestone for this homework: February 16th, 11:59pm. (extra 10 points)

Files to submit:

1. dHeap.java
2. dHeapTester.java
3. MyPriorityQueue.java

Final submission: The following files must be submitted on Vocareum for the final submission.

1. dHeap.java
2. dHeapTester.java
3. MyPriorityQueue.java
4. Course.java
5. Student.java
6. Registration.java
7. CourseScheduling.java

6 Appendix

6.1 File Reading using Scanner

Below is a simple example program for reading a file using a Scanner.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class ScannerReadFile {
    public static void main(String[] args) {
        // Create an instance of File.
        File file = new File("data.txt");
        try {
            // Create a new Scanner object which will read the data
            // from the file passed in. To check if there are more
            // line to read from it we check by calling the
            // scanner.hasNextLine() method. We then read line one
            // by one till all line is read.
            Scanner scanner = new Scanner(file);
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                System.out.println(line);
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

You may find the following methods from the Scanner class more useful to implement your algorithm.

Numeric and String methods

Method	*Returns*
`int nextInt()`	Returns the next token as an `int`. If the next token is not an integer, `InputMismatchException` is thrown.
`long nextLong()`	Returns the next token as a `long`. If the next token is not an integer, `InputMismatchException` is thrown.
`float nextFloat()`	Returns the next token as a `float`. If the next token is not a float or is out of range, `InputMismatchException` is thrown.
`double nextDouble()`	Returns the next token as a `long`. If the next token is not a float or is out of range, `InputMismatchException` is thrown.
`String next()`	Finds and returns the next complete token from this scanner and returns it as a string; a token is usually ended by whitespace such as a blank or line break. If not token exists, `NoSuchElementException` is thrown.
`String nextLine()`	Returns the rest of the current line, excluding any line separator at the end.
`void close()`	Closes the scanner.

Boolean methods

Method	*Returns*
`boolean hasNextLine()`	Returns `true` if the scanner has another line in its input; `false` otherwise.
`boolean hasNextInt()`	Returns `true` if the next token in the scanner can be interpreted as an `int` value.
`boolean hasNextFloat()`	Returns true if the next toke in the scanner can be interpreted as a `float` value.

6.2 Debugging with Eclipse

The most important features of IDEs like Eclipse are the Debugging features. Here are a few tutorials below that help you learn how to Debug your program in Eclipse. I strongly recommend that you become familiar with debugging as it will be very very useful in your future CS classes.
https://www.tutorialspoint.com/eclipse/eclipse_debugging_program.htm
http://eclipsesource.com/blogs/2013/01/08/effective-java-debugging-with-eclipse/