CS 202 Fundamental Structures of Computer Science II
Assignment 3 – Heaps, Priority Queues and AVL Trees

Assigned on: July 11, 2019
Due Date: Juy 18 2019, 23:55

1) Question Part (40 points)

a) (10 points) This problem gives you some practice with the basic operations on binary min heaps. Make sure to check your work.

- Starting with an empty binary min heap, show the result of inserting, in the following order, 13, 9, 4, 8, 5, 6, 14, 3, 7, 11 and 2, one at a time, into the heap. By show, we mean, "draw the tree resulting from each insert."
- Now perform two deleteMin operations on the binary min heap you constructed in part (a). Show the binary min heaps that result from these successive deletions, again by drawing the binary tree resulting from each delete

b) (10 points) Consider a binary min heap. Print the keys as encountered in preorder traversal. Is the output sorted? Justify your answer. Attempt the same question for inorder and postorder traversals.

c) (10 Points) Write a function in pseudocode that determines whether a given binary tree is a min heap.

d) (10 points) Show the result of inserting 18, 11, 8, 12, 7, 9, 16, 5, 4 and 6 in that order into an initially empty **AVL** tree. Show the tree after each insertion, clearly labeling which tree is which.

2) Programming Part (60 points)

In this programming assignment, you will try to find a software solution to the problem of the director of the Bilkent Health Center. She is trying to figure out how many doctors should work in the emergency service of the center. For each doctor in the emergency service, the expense of the health center increases; however, according to the standards of Bilkent, average waiting time for all patients should not exceed a given amount of time. So, she needs to optimize this number and asks for your help in this task. The center has the data of past patients. Your program should use these data to calculate average waiting times and find out the minimum number of doctors needed to meet the average waiting time requirement.

The data are stored in a plain text file[1]. The first line of the file contains the number of patients. The subsequent lines contain four integers, each separated by one or more whitespace characters (space or tab). These denote, respectively, the patient id, the patient priority (from 1 to 10, higher is more urgent), arrival time (in minutes from a given point [e.g. 12:00 am]) and service time (in minutes).

[1] The file is a UNIX-style text file with the end-of-line denoted by a single \n (ASCII 0x0A)

For example, from the file content given below, we understand there are 3 patients. The first patient with id 1 has priority of 9, arrives at the center at minute 1, and his treatment lasts for 5 minutes. The second patient with id 14 has priority of 3, arrives at the center at minute 70, and his treatment lasts 10 minutes. The third patient with id 5 has priority of 3, arrives at the center at minute 82, and his treatment lasts 70 minutes.

Sample input file:

```
3
1     9     1     5
14    3     70    10
5     3     82    70
```

In this assignment, you are asked to write a simulation program that reads patient data from an input file and calculates the minimum number of doctors required for a given maximum average waiting time.

In your implementation, you may make the following assumptions:

- The data file will always be valid. All data are composed of integers.
- In the data file, the patients are sorted according to their arrival times.
- There may be at most 200 patients in the data file.

Your implementation must obey the following requirements:

- The patient with the highest priority should be examined first.
- In case of having two patients with the same highest priority, the patient who has waited longer should be selected first.
- If more than one doctor is available at a given time; the patient is assigned to the doctor with a lower id.
- Once a patient is assigned to a doctor, the doctor immediately starts treating that patient and is not available during the treatment time given for that patient. After the treatment of that patient ends, the doctor becomes available immediately.
- The waiting time of a patient is the duration (difference) between the arrival time of the patient and the time he is assigned to a doctor.

In your implementation, you **MUST** use a heap-based priority queue to store patients who are waiting for a doctor (i.e., to store patients who have arrived at the center but have not been treated yet). If you do not use such a heap-based priority queue to store these patients, then you will get no points from this question.

The name of the input file and the maximum average waiting time will be provided as command line arguments to your program. Thus, your program should run using two command line arguments. Thus, the application interface is simple and given as follows:

`username@hostname:~>./simulator <filename> <avgwaitingtime>`

Assuming that you have an executable called “simulator”, this command calls the executable with two command line arguments. The first one is the name of the file from which your program reads the patient data. The second one is the maximum average waiting time; your program should calculate the minimum number of doctors required for meeting this **`avgwaitingtime`**. You may assume that the maximum average waiting time is given as an integer.

Hint

Use the heap data structure to hold patients that are waiting for a doctor and to find the patient with the highest priority. Update the heap whenever a new patient arrives or a patient treatment starts. In order to find the optimum number of doctors needed, repeat the simulation for increasing number of doctors and return the minimum number of doctors that will achieve the maximum average waiting time constraint. Display the simulation for which you find the optimum number of doctors.

SAMPLE OUTPUT:

Suppose that you have the following input file consisting of the patient data. Also suppose that the name of the file is `patients.txt`.

```
12
1       20      1       10
2       40      1       14
3       10      1       6
4       10      1       5
5       20      4       10
6       40      7       14
7       20      9       10
8       40      11      14
9       10      13      6
10      10      14      5
11      20      15      10
12      40      17      14
```

The output for this input file is given as follows for different maximum average waiting times. Please check your program with this input file as well as the others that you will create. Please note that we will use other input files when grading your assignments.

```
username@hostname:~>./simulator   patients.txt   5

Minimum number of doctors required: 4

Simulation with 4 doctors:

Doctor 0 takes patient 2 at minute 1 (wait: 0 mins)
Doctor 1 takes patient 1 at minute 1 (wait: 0 mins)
Doctor 2 takes patient 3 at minute 1 (wait: 0 mins)
Doctor 3 takes patient 4 at minute 1 (wait: 0 mins)
Doctor 3 takes patient 5 at minute 6 (wait: 2 mins)
Doctor 2 takes patient 6 at minute 7 (wait: 0 mins)
Doctor 1 takes patient 8 at minute 11 (wait: 0 mins)
Doctor 0 takes patient 7 at minute 15 (wait: 6 mins)
Doctor 3 takes patient 11 at minute 16 (wait: 1 mins)
Doctor 2 takes patient 12 at minute 21 (wait: 4 mins)
Doctor 0 takes patient 9 at minute 25 (wait: 12 mins)
Doctor 1 takes patient 10 at minute 25 (wait: 11 mins)

Average waiting time: 3 minutes
```

```
username@hostname:~>./simulator   patients.txt   10

Minimum number of doctors required: 3

Simulation with 3 doctors:

Doctor 0 takes patient 2 at minute 1 (wait: 0 mins)
Doctor 1 takes patient 1 at minute 1 (wait: 0 mins)
Doctor 2 takes patient 3 at minute 1 (wait: 0 mins)
Doctor 2 takes patient 6 at minute 7 (wait: 0 mins)
Doctor 1 takes patient 8 at minute 11 (wait: 0 mins)
Doctor 0 takes patient 5 at minute 15 (wait: 11 mins)
Doctor 2 takes patient 12 at minute 21 (wait: 4 mins)
Doctor 0 takes patient 7 at minute 25 (wait: 16 mins)
Doctor 1 takes patient 11 at minute 25 (wait: 10 mins)
Doctor 0 takes patient 4 at minute 35 (wait: 34 mins)
Doctor 1 takes patient 9 at minute 35 (wait: 22 mins)
Doctor 2 takes patient 10 at minute 35 (wait: 21 mins)

Average waiting time: 9.83333 minutes
```

Question about Scalability (5 pnts)

Now suppose that your simulation was to be run for the emergency service of a very large hospital with many potential doctors (*N*) and many, many more patients. Would it still be a good idea to try the simulation starting from 1 doctor and increasing until you found the right number $K \leq N$? What is a better strategy for finding the optimum number of doctors in such a case?

HAND-IN

- You should prepare the answers of 1) Question Part and the discussion part of 2) Programming Part using a word processor (in other words, do not submit images of handwritten answers).
- Before 23:55 of July 18, 2019, upload your solutions using Moodle. You should upload a single zip file that contains
  - `hw3.pdf`, the file containing the answers to Question 1 & about scalability in Question 2.
  - `simulator.cpp`, `simulator.h`, and `main.cpp`, the files containing the C++ source code of Question 2.
  - `readme.txt`, the file containing anything important on the compilation and execution of your program in Question 2.
  - Do not put any unnecessary files such as the auxiliary files generated from your favorite IDE. Be careful to avoid using any OS dependent utilities.
  - Do not forget to put your name and student id in all of these files. Well comment your implementation.
  - Name your zip file as follows: **`NAME_SURNAME_hw3.zip`**.
  - Keep all the files before you receive your grade.
  - This homework will be graded by your TA, Ege Berkay Gulcan (berkay.gulcan at bilkent edu tr). Thus, you may ask your homework related questions directly to him.

IMPORTANT:

- Although you may use any platform or any operating system to implement your algorithms and obtain your experimental results, your code should work on the **dijkstra** server (dijkstra.ug.bcc.bilkent.edu.tr). We will compile and test your programs on that server. If your C++ code does not compile or execute in that server, you will lose points.
- **Attention:** For this assignment, you are allowed to use the codes given in our textbook and/or our lecture slides. However, you ARE NOT ALLOWED to use any codes from other sources (including the codes given in other textbooks, found on the Internet, belonging to your classmates, etc.). Furthermore, you ARE NOT ALLOWED to use any data structure or algorithm related function from the C++ standard template library (STL).

Do not forget that plagiarism and cheating will be heavily punished. Please do the homework yourself.