CS 202, Spring 2021

Homework 3 – Heaps, Priority Queues and AVL Trees

Due: 23:55, April 16, 2021

Important Notes

Please do not start the assignment before reading these notes.

1. Before 23:55, April 16, upload your solutions in a single **ZIP** archive using Moodle submission form. Name the file as studentID_secNo_hw3.zip.
2. Your ZIP archive should contain the following files:
   - **`hw3.pdf`**, the file containing the answers to Questions 1 and 3.
   - `simulator.cpp`, `simulator.h`, and `main.cpp`, the files containing the C++ source code of Question 2, and the **`Makefile`**.
   - Do not forget to put your name, student id, and section number in all of these files. Well comment your implementation. Add a header as given below to the beginning of each file:

```
/*
 * Title: Heaps
 * Author: Name Surname
 * ID: 21000000
 * Section: 1
 * Assignment: 3
 * Description: description of your code
 */
```

   - 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 (for example to measure the time).
   - You should upload handwritten answers for Q1 (in other words, do not submit answers prepared using a word processor).
   - Use the exact algorithms shown in lectures.
3. 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. Thus, you will lose a significant amount of points if your C++ code does not compile or execute on the `dijkstra` server.
4. This homework will be graded by your TA, Kemal Büyükkaya. Thus, please contact him directly (kemal.buyukkaya at bilkent.edu.tr) for any homework related questions.

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.

Question 1 – 30 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, 24, 20, 15, 19, 16, 17, 25, 14, 18, 22 and 13, 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) Show the result of inserting 29, 22, 19, 23, 18, 20, 27, 16, 15 and 17 in that order into an initially empty **AVL** tree. Show the tree after each insertion, clearly labeling which tree is which.

Question 2 – 60 points

In this programming assignment, you will estimate the minimum number of printers needed in Bilkent Library so that the average waiting time of a print request does not exceed a given amount of time.

In order to estimate the minimum number of printers needed, you will be given the logs of the print requests and use these data to simulate the processing of the requests for the printers and calculate the average waiting time for the requests. Using the calculated average waiting time, you will estimate the minimum number of printers needed for the future.

The log file is stored in a simple text file which is a unix-style file and the end of line character is denoted by “\n”. The first line of the file gives the number of requests in the log file. The subsequent lines include the id, priority, request time (denotes the time in minutes since the log file is created) and process time (denotes the processing time of the request in minutes). Each of these are stored as integer numbers and separated by a white space.

For example, from the file content given below, we see 4 requests sent to the printer and we can extract the information about each request. For instance, the first request has id 1 and priority 8. Additionally, we can see that it is sent after 7 mins of log file creation and can be processed in 4 mins. Similar information can also be obtained for other requests as well.

Sample input file:

```
4
1 8 7 4
2 5 70 10
3 3 82 70
5 1 100 5
```

In this assignment, you will assume that each printer follows the given protocol below:

- The request with the highest priority should be processed first.
- In case of having two requests with the same highest priority, the request which is sent earlier should be selected.
- If more than one printer is available, then the printer with lower id will process the request.
- Once a request is assigned to a printer, the printer immediately starts processing the request and is not available during the processing time for that request. After the process of that request ends, the printer becomes available immediately.

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 requests are sorted according to their request arrival times.
- There may be any number of requests in the log file. For example, in the input file given above this number is 4. (**Hint:** use dynamic memory for allocations, and release it before the program ends).

In your implementation, you MUST use a heap-based priority queue to store requests that are waiting for a printer (i.e., to store requests that were sent but have not been processed yet). If you do not use such a heap-based priority queue to store these requests, 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 as follows:

```
username@dijkstra:~>./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 log data. The second one is the maximum average waiting time. Your program should calculate the minimum number of printers required for meeting this `avgwaitingtime`. You may assume that the maximum average waiting time is given as a double value.

Hint

> Use the heap data structure to hold requests that are waiting for a printer and to find the request with the highest priority. Update the heap whenever a new print request arrives or a request processing starts. In order to find the optimum number of printers needed, repeat the simulation by increasing the number of printers and return the minimum number of printers that will achieve the maximum average waiting time constraint. Display the simulation for which you find the optimum number of printers.

SAMPLE OUTPUT: Suppose that you have the following input file consisting of the printer request data. Also suppose that the name of the file is `print_jobs.txt`.

```
10
1       2       1       10
2       4       1       12
3       1       1       6
4       1       1       5
5       2       4       10
6       4       7       14
7       2       9       10
8       4       11      14
9       1       13      6
10      1       14      5
```

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 print_jobs.txt 5

Minimum number of printers required: 4

Simulation with 4 printers:

Printer 0 prints print request 2 at minute 1 (wait: 0 mins)
Printer 1 prints print request 1 at minute 1 (wait: 0 mins)
Printer 2 prints print request 3 at minute 1 (wait: 0 mins)
Printer 3 prints print request 4 at minute 1 (wait: 0 mins)
Printer 3 prints print request 5 at minute 6 (wait: 2 mins)
Printer 2 prints print request 6 at minute 7 (wait: 0 mins)
Printer 1 prints print request 8 at minute 11 (wait: 0 mins)
Printer 0 prints print request 7 at minute 13 (wait: 4 mins)
Printer 3 prints print request 9 at minute 16 (wait: 3 mins)
Printer 2 prints print request 10 at minute 21 (wait: 7 mins)

Average waiting time: 1.6 minutes
```

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

Minimum number of printers required: 3

Simulation with 3 printers:

Printer 0 prints print request 2 at minute 1 (wait: 0 mins)
Printer 1 prints print request 1 at minute 1 (wait: 0 mins)
Printer 2 prints print request 3 at minute 1 (wait: 0 mins)
Printer 2 prints print request 6 at minute 7 (wait: 0 mins)
Printer 1 prints print request 8 at minute 11 (wait: 0 mins)
Printer 0 prints print request 5 at minute 13 (wait: 9 mins)
Printer 2 prints print request 7 at minute 21 (wait: 12 mins)
Printer 0 prints print request 10 at minute 23 (wait: 9 mins)
Printer 1 prints print request 9 at minute 25 (wait: 12 mins)
Printer 0 prints print request 4 at minute 28 (wait: 27 mins)

Average waiting time: 6.9 minutes
```

Question 3 – Scalability (10 points)

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