CS 202, Spring 2022

Homework 3 – Heaps, Priority Queues and AVL Trees

Due: 23:55, April 15, 2022

Important Notes

Please do not start the assignment before reading these notes.

1. Before 23:55, April 15, 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.
   - **`all .h files and .cpp files`** files which contain the C++ source codes, and the **`Makefile`**. "make" command should build an executable called "simulator", so write your Makefile accordingly (i.e. at the end, when you type "make" on a terminal in your working directory, it should produce "simulator" executable) .
   - Do not forget to put your name, student id, and section number in all of these files. Well comment your code. Add a header as given below to the beginning of each file:

```
/*
 * Title: Heaps, Priority Queues
 * 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.
   - You should upload handwritten answers for Question 1 and 3 (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, Salman Mohammad. Thus, please contact him directly (salman.mohammad 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) Insert letters ‘F’, ‘I’, ‘L’, ‘S’, ‘J’, ‘V’, ‘M’, ‘T’, ‘Z’, ‘U’, ‘O’ to an AVL tree (by hand, no codes) and show the resulting tree after each insertion (if a rotation is needed show the tree before and after the rotation, and state the type of the rotation). Assume that the ordering between letters is according to their ASCII codes, i.e. A < B.

(b) (10 points) Give a pseudocode (in a C++ like syntax) for computing the median of a given AVL tree (consists of all integer values as keys) called computeMedian. You can make changes in the node structure of the tree (indicate what you have added), you can assume a pointer based tree, you can assume that the tree contains all unique elements, that is, there is no repetition of elements. Your algorithm should take the root as input. Give the running time complexity of your algorithm in Big-O notation with a brief explanation of the algorithm logic.

(c) (10 points) Give a pseudocode (in a C++ like syntax) for checking whether a given binary search tree is an AVL tree or not, called checkAVL. Your function should take the root of the tree as input, you can assume a pointer based tree with left and right child pointers as well as the key value in node structure. Give the running time complexity of your algorithm in Big-O notation with a brief explanation of the algorithm logic.

Question 2 – 65 points

In this programming assignment, you will estimate the minimum number of computers needed in a web hosting firm so that the average waiting time of a client for each HTTP request does not exceed a given amount of time.

In order to estimate the minimum number of computers needed, you will be given the logs of the HTTP requests and use this data to simulate the processing of the requests by the computers and calculate the average waiting time for the requests. Using the calculated average waiting time, you will estimate the minimum number of computers 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 milliseconds since the log file is created) and process time (denotes the processing time of the request in milliseconds). 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 made to the server and we can extract the information about each request. For instance, the first request has id 1 and priority 9. Additionally, we can see that it is sent after 3 ms of log file creation and can be processed in 5 ms. Similar information can also be obtained for other requests as well.

Input file:

```
4
1 9 3 5
2 5 70 10
3 3 82 70
5 1 100 5
```

In this assignment, you will assume that each computer 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 computer is available, then the computer with lower id will process the request. - Once a request is assigned to a computer, the computer immediately starts processing the request and is not available during the processing time for that request. After the process of that request ends, the computer 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 sent 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).

Important Note: In your implementation, you MUST use a heap-based priority queue to store requests that are waiting for a computer (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 assignment.

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@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 computers 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 computer and to find the request with the highest priority. Update the heap whenever a new request arrives or a request processing starts. In order to find the optimum number of computers needed, repeat the simulation for an increasing number of computers and return the minimum number of computers that will achieve the maximum average waiting time constraint. Display the simulation for which you find the optimum number of computers.

SAMPLE OUTPUT:

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

```
21
1 20 1 10
2 5 1 10
3 10 1 34
4 10 10 21
5 60 20 30
6 10 35 60
7 10 41 70
8 10 41 40
9 10 41 50
10 10 41 60
11 10 49 60
12 40 50 100
13 5 55 10
14 40 61 14
15 10 81 60
16 10 98 50
17 20 105 10
18 40 120 14
19 10 130 50
20 60 150 10
21 40 150 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@dijkstra:~>./simulator log.txt 50

Minimum number of computers required: 3

Simulation with 3 computers:

Computer 0 takes request 1 at ms 1 (wait: 0 ms)
Computer 1 takes request 3 at ms 1 (wait: 0 ms)
Computer 2 takes request 2 at ms 1 (wait: 0 ms)
Computer 0 takes request 4 at ms 11 (wait: 1 ms)
Computer 2 takes request 5 at ms 20 (wait: 0 ms)
Computer 0 takes request 6 at ms 35 (wait: 0 ms)
Computer 1 takes request 7 at ms 41 (wait: 0 ms)
Computer 2 takes request 12 at ms 50 (wait: 0 ms)
Computer 0 takes request 14 at ms 95 (wait: 34 ms)
Computer 0 takes request 17 at ms 109 (wait: 4 ms)
Computer 1 takes request 10 at ms 111 (wait: 70 ms)
Computer 0 takes request 8 at ms 119 (wait: 78 ms)
Computer 2 takes request 20 at ms 150 (wait: 0 ms)
Computer 0 takes request 18 at ms 159 (wait: 39 ms)
Computer 2 takes request 21 at ms 160 (wait: 10 ms)
Computer 1 takes request 9 at ms 171 (wait: 130 ms)
Computer 0 takes request 11 at ms 173 (wait: 124 ms)
Computer 2 takes request 15 at ms 174 (wait: 93 ms)
Computer 1 takes request 16 at ms 221 (wait: 123 ms)
Computer 0 takes request 19 at ms 233 (wait: 103 ms)
Computer 2 takes request 13 at ms 234 (wait: 179 ms)

Average waiting time: 47.0476 ms
```

```
username@dijkstra:~>./simulator log.txt 20

Minimum number of computers required: 5

Simulation with 5 computers:

Computer 0 takes request 1 at ms 1 (wait: 0 ms)
Computer 1 takes request 3 at ms 1 (wait: 0 ms)
Computer 2 takes request 2 at ms 1 (wait: 0 ms)
Computer 3 takes request 4 at ms 10 (wait: 0 ms)
Computer 0 takes request 5 at ms 20 (wait: 0 ms)
Computer 1 takes request 6 at ms 35 (wait: 0 ms)
Computer 2 takes request 7 at ms 41 (wait: 0 ms)
Computer 3 takes request 10 at ms 41 (wait: 0 ms)
Computer 4 takes request 9 at ms 41 (wait: 0 ms)
Computer 0 takes request 12 at ms 50 (wait: 0 ms)
Computer 4 takes request 14 at ms 91 (wait: 30 ms)
Computer 1 takes request 8 at ms 95 (wait: 54 ms)
Computer 3 takes request 11 at ms 101 (wait: 52 ms)
Computer 4 takes request 17 at ms 105 (wait: 0 ms)
Computer 2 takes request 15 at ms 111 (wait: 30 ms)
Computer 4 takes request 16 at ms 115 (wait: 17 ms)
Computer 1 takes request 18 at ms 135 (wait: 15 ms)
Computer 1 takes request 19 at ms 149 (wait: 19 ms)
Computer 0 takes request 20 at ms 150 (wait: 0 ms)
Computer 0 takes request 21 at ms 160 (wait: 10 ms)
Computer 3 takes request 13 at ms 161 (wait: 106 ms)

Average waiting time: 15.8571 ms
```

Question 3 – about Scalability – 5 points

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