CS 202 Fundamental Structures of Computer Science II

Assignment 3 – Heaps and AVL Trees

Date Assigned: November 14, 2016
Due Date: November 24, 2016

1) Question Part (30 points)

a) (5 points) Insert numbers ‘7’, ‘12’, ‘15’, ‘22’, ‘13’, ‘27’, ‘16’, ‘24’, ‘29’, ‘25’, ‘18’ 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).

b) (7,5 points) Give a pseudocode (in a C++ like syntax) for efficiently computing the median of a given AVL tree (consists of all integer values as keys) called **computeMedian**. Feel free to make changes in classes for AVL node and/or AVL tree (indicate what you changed), to facilitate easy computation of the median. You may also assume that the tree contains all unique elements; that is, there is no repetition of elements. Your algorithm should take the root of the tree as input. Give the running time complexity of your algorithm in Big-O notation with a brief explanation of the algorithm logic.

c) (7,5 points) Give a pseudocode (in a C++ like syntax) for an efficient algorithm 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 may assume a pointer based tree implementation with left and right child pointers as well as the key value in the node structure. Give the running time complexity of your algorithm in Big-O notation with a brief explanation of the algorithm logic.

d) (10 points) State the following propositions as **true** or **false**. Prove if the statement is true; give a counter example if it is false.

- i- All heaps are AVL trees.
- ii- A complete AVL tree is also a red-black tree with all black nodes.
- iii- In a tree with n nodes, if in every node, the difference between heights of two child nodes is at most c (i.e. a constant), than the height of the tree is O (log n).

2) Programming Part (70 points)

In this programming assignment, you are asked to find a software solution to the scheduling problem of Bilkent Bank. The bank’s manager is trying to figure out how many cashiers should work in the customer service of the bank. For each cashier in the service, the expense of the bank increases; however according to the standards of the bank, the average waiting time for all customers should not exceed a given amount of time. So, she needs to optimize this number and asks for your help in this task. The bank has the data to predict service time of customers. Your program should use these data to calculate average waiting times and find out the minimum number of cashiers 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 customers. The subsequent lines contain four integers, each separated by one or more whitespace characters (space or tab). These denote, respectively, the customer id, the registration year of the customer to the bank[2], arrival time for transaction (in minutes from a given point [e.g. 12:00 am]) and service time (in minutes).

For example, from the file content given below, we understand there are 3 customers. The first customer with id 1 arrives at the bank at minute 1, and his transaction lasts for 5 minutes. He has been a customer of the bank for 15 years. The second customer with id 14 arrives at the bank at minute 70, and his transaction lasts 10 minutes. The third customer with id 5 arrives at the bank at minute 82, and his transaction lasts 70 minutes. The second and third customers have been a customer of the bank for 11 years, which means that their priorities are the same.

1 The file is a UNIX-style text file with the end-of-line denoted by a single \n (ASCII 0x0A)
2 The bank was established in 1968

Sample input file:

```
3
1     1999 1    5
14    2003 70   10
5     2003 82   70
```

In this assignment, you are asked to write a simulation program that reads customer data from an input file and calculates the minimum number of cashiers 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 customers are sorted according to their arrival times.
- There may be at most 200 customers in the data file.

The bank assigns priority to its customers in order to provide them some special benefits. One of them is that they do not have to wait in the queue for transactions. The priority is defined considering the period of time that a customer has been with the bank. Long-time customers of the bank have higher priority than others. Your implementation must obey the following requirements:

- The customer with the highest priority should be examined first.
- In case of having two customers with the same highest priority, the customer who has waited longer should be selected first.
- If more than one cashier is available at a given time; the customer is assigned to the cashier with a lower id.
- When a cashier starts giving a service to a customer, the cashier should finish his service with this customer even though another customer with a higher priority comes to the bank.
- Once a customer is assigned to a cashier, the cashier immediately starts carrying out the transaction of that customer and is not available during the transaction time given for that customer. After the transaction of that customer carries out, the cashier becomes available immediately.
- The waiting time of a customer is the duration (difference) between the arrival time of the customer and the time he is assigned to a cashier.

In your implementation, you MUST use a heap-based priority queue to store customers who are waiting for a cashier (i.e., to store customers who have arrived at the bank but their transactions have not been conducted yet). If you do not use such a heap-based priority queue to store these customers, then you will get no points from this homework.

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 customer data. The second one is the maximum average waiting time; your program should calculate the minimum number of cashiers 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 customers that are waiting for a cashier and to find the customer with the highest priority. Update the heap whenever a new customer arrives or a customer's transaction is conducted. In order to find the optimum number of cashiers needed, repeat the simulation for increasing number of cashiers and return the minimum number of cashiers that will achieve the maximum average waiting time constraint. Display the simulation for which you find the optimum number of cashiers.

SAMPLE OUTPUT:

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

```
12
1      1994   1      10
2      1974   1      14
3      2004   1      6
4      2004   1      5
5      1994   4      10
6      1974   7      14
7      1994   9      10
8      1974   11     14
9      2004   13     6
10     2004   14     5
11     1994   15     10
12     1974   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@dijkstra:~>./simulator   customers.txt   5`

```
Minimum number of cashiers required: 4

Simulation with 4 cashiers:

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

Average waiting time: 3 minutes
```

`username@dijkstra:~>./simulator   customers.txt   10`

```
Minimum number of cashiers required: 3

Simulation with 3 cashiers:

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

Average waiting time: 9.83333 minutes
```

IMPORTANT NOTES

Please do not start the assignment before reading these notes:

- Before 23:55 on due date as specified above upload your solutions to Moodle.
- Don’t forget to write down your id, name, section, assignment number or any other information relevant to your program in the beginning of your main file.
- Don’t forget to write comments at important parts of your code.
- Be careful about indentation.
- You are free to write your programs in any environment (you may use either Linux or Windows).
- The name of the archive file must be in the following format.

Surname_Name_StudentId_Section_HW3.zip

- In this assignment, you must submit a single archive file that contains your implementation. The name of the archive file must be in the following format. You should upload a **single zip** file that contains your solution to first part as **pdf** (do NOT send photos of your solution, type it if it is possible, scan if it is not), code files (only the "**.cpp**" and "**.h**" files that you write for the homework) of the second part and a "**Makefile**" for the compilation of your code that produces the executable named "**simulator**". In the end, your zip file should contain ONLY **code files**, "**Makefile**" and **pdf** of first part; any violation of these causes a significant loss from your grade. Name of the pdf should be "***Section_ID_SurnameName.pdf***".
    - *Surname_Name_StudentId_Section_HW3.zip*

- **IMPORTANT: Although you may use any platform and any operating system in implementing your algorithms and obtaining your experimental results, your code should work in a Linux environment with the g++ compiler. We will test your codes in a Linux environment. Thus, you may lose significant amount of points, if your C++ code does not compile or execute in a Linux environment.**
- Keep all the files until you receive your grade.
- If you do not know how to write a Makefile, you can find lots of tutorials on the Internet. Basically they are files that contain a list of rules for building an executable that is used by "make" utility of Unix/Linux environments. "make" command should build an executable called "simulator", so write your Makefile accordingly (i.e. at the end, when you type "make" in terminal on your working directory, it should produce "simulator" executable)
- Late submissions will **not** be graded.
- This homework will be graded by TA Miray Ayşen. Thus, you may ask your homework related questions directly to her. (miray.aysen at bilkent edu tr)
- For this assignment, you cannot use any existing code from other sources such as the heap class in the C++ standard template library (STL). However, 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 somewhere else (e.g., from the internet, other text books, other slides ...).
- Pay attention to these instructions, otherwise you may lose some points even though your code has no error.

DO THE HOMEWORK YOURSELF. PLAGIARISM AND CHEATING ARE HEAVILY PUNISHED!!!