FIT2004 S2/2016: Assessment questions for week 10

THIS PRAC IS ASSESSED! (6 Marks)

DEADLINE: Monday, 03-Oct-2016 10:00:00 AM
CLASS: This programming exercise has to be completed before the deadline. Your demonstrator will check your submission during the lab. He will ask you a series of questions to assess your understanding of this exercise, and gauge how you implemented it. It is required that you implement this exercise strictly using **Python programming language**. Practical work is marked on the performance of your program **and also on your understanding of the program**. A *perfect* program with zero understanding implies you will get **zero** marks! "Forgetting" is not an acceptable explanation for lack of understanding. Demonstrators are not obliged to mark programs that do not run or that crash.
After/befor your demonstrators have interviewed you, you are expected to work towards the programming competition or other questions that will be provided to you on the day.
SUBMISSION REQUIREMENT: You will need to submit a zipped file containing your Python programs (named CamDetector.py) as well as a PDF file briefly describing your solutions and their space and time complexities. The PDF file must give an outline of your solution (e.g., a high level idea of how did you solve it) and the **worst-case** space and time complexity of your solution. The PDF file must also include your solution to the problems related to AVL tree. The zipped file is to be submitted on Moodle before the deadline.
Important: You should carefully consider the border cases and make sure that you return correct results for all cases. Although you are not required to include the proof of correctness in your submissions, you are encouraged to formally prove the correctness of your programs. This will help you in identifying and fixing the bugs in your program if present.

1 Task 1: Red-light Camera Detector

Alice bursts into your room and is visibly upset. She waves a paper and screams, "I cannot believe this. They sent me a ticket for violating the traffic lights. I am sure the light was amber when I crossed the intersection. This is unfair." She then takes a long deep breath and firmly says, "Okay, I have decided that I will never drive through an intersection with red-light cameras again. Never! Can you please help me to write an app to detect the red-light cameras?".

"Well, I would love to help but I am quite busy with studies and ...", you murmur. Alice says, "Oh come on! I will provide you all the data and you just have to write an algorithm. It will not take much time and I heard you will have holidays in a week or so. I can wait!". You desperately try to explain, "Holidays? You mean mid-semester break? These are not holidays Alice! We are supposed to use the break to revise the content covered earlier in the semester and I have plenty of material to revise".

Alice does not give up so easily, "All I am asking is to implement an algorithm. Remember the time when I built a website for you. I am taking the algorithms and data structure course next semester anyway so I will not bother you after that. Pleaaasee! ". And before you could say something, she starts explaining what she wants.

"I will provide you two text files named vertices.txt and edges.txt. The file vertices.txt contains the IDs of the vertices/intersections that have red-light cameras. The file edges.txt contains the information of edges. Specifically, the first two values in each row correspond to the vertices that the edge connects and the third value corresponds to the distance between the vertices. Some rows in edges.txt have four values where the fourth value is `TOLL` denoting that this edge is a toll road. I want the application to take as input a source vertex v and a value of k and return me the k closest cameras and the shortest paths to each of these cameras. I do not want to travel on a toll road or drive through an intersection that has a red-light camera. Therefore, none of the paths must contain a toll road or *pass through* a red-light camera."

1.1 Input

The input files vertices.txt and edges.txt can be downloaded from Moodle. The total number of vertices are 6105 and you can assume that their IDs are in the range 0 to 6104. Below are the first 5 lines from vertices.txt denoting that the veritces $1, 5, 12, 16$ and 20 have red-light cameras.

```
1 camera
5 camera
12 camera
16 camera
20 camera
```

Below are some lines from edges.txt.

```
122 127 526.950012
123 125 228.396591
124 138 544.010193 TOLL
125 126 259.437286
```

The third line above shows that there is an edge between vertices with IDs 124 and 138. The length of this road segment is 544.010193 and this is a toll road.

Important: Your program must read data from the input files named "vertices.txt" and "edges.txt" (provided in the zipped folder). You will lose marks if the program does not correctly read from the provided files. You must test your program extensively. Your program will be tested on a file containing the same road network but with different toll roads and different locations of cameras.

1.2 Output

The program must ask the users to enter their location (vertex ID) and the value of k and then display the information of k-closest cameras. Below is a sample output.

```
Enter your location: 1609
Enter k: 3
Camera 1 : 1595   Distance from your location: 89.02904600000001
Shortest path:  1609 --> 1600 --> 1593 --> 1595

Camera 2 : 1624   Distance from your location: 143.87629700000002
Shortest path:  1609 --> 1600 --> 1607 --> 1624

Camera 3 : 1648   Distance from your location: 170.821331
Shortest path:  1609 --> 1622 --> 1616 --> 1625 --> 1627 --> 1636 --> 1648
```

Note that there is a camera at vertex 1580 with distance 166.19017100000002 but it is not returned in the above output because the shortest path to this camera passes through another camera (at intersection 1595). Below are the details of the camera that must be ignored by your algorithm.

```
Camera : 1580   Distance from your location: 166.19017100000002
Shortest path:  1609 --> 1600 --> 1593 --> 1595 --> 1578 --> 1580
```

Below is another sample output.

```
Enter your location: 2011
Enter k: 2
Camera 1 : 1999   Distance from your location: 21.692297
Shortest path:  2011 --> 1999

Camera 2 : 1991   Distance from your location: 302.152092
Shortest path:  2011 --> 2016 --> 4563 --> 2003 --> 1985 --> 1976 --> 1991
```

There is a path to camera at intersection 4529 with distance 238.061092 but it is ignored because the path contains a toll road.

```
Camera : 4529   Distance from your location: 238.061092
Shortest path:  2011 --> 2016 --> 4563 --> 4556 --> 4549 --> 4543 --> 4541
 --> 4529
```

For the above camera, the road segment `4543 --> 4541` is a toll road so this path must not be considered by your algorithm. The shortest path to 4529 that does not pass through any toll road or other camera is shown below.

```
Camera : 4529   Distance from your location: 474.407198
Shortest path:  2011 --> 2016 --> 4563 --> 4556 --> 3433 --> 3446 --> 4557
 --> 4545 --> 4542 --> 4529
```

This camera is not among the 2-closest cameras so it is not shown in the sample output above.

If the user is already on an intersection with red-light camera (i.e., the entered location is a vertex with camera), the algorithm must return a warning. Below is a sample output.

```
Enter your location: 2010
Enter k: 5
Oops! Too late to help!!! Please smile for the camera!
```

In some cases, it is possible that the k-closest cameras cannot be found (e.g., there may be less than k cameras that can be reached without passing through any toll road or any other camera). In this case, your algorithm must return only the cameras that can be reached. For example, assume that you are on a vertex v that has only two adjacent vertices u and x and both u and x have red-light cameras. If $k > 2$, the algorithm will return only u and x because no other cameras can be reached without passing through an intersection with a camera.

1.3 Implementation Requirements

Let V and E denote the number of vertices and edges, respectively. Your program must take $O(V + E)$ space and $O(V \log V + E \log V)$ time (in the worst-case) to return all k cameras. To achieve this time complexity, you will need to modify min-heap as described in the lecture slides. You must implement your own heap data structure.

2 Task 2: AVL Tree

Task 2a: Insert $20, 25, 32, 35, 40, 28$ in an empty AVL tree in this order.
Task 2b: Delete $30, 20, 25$ from the AVL tree shown in Fig, 1.

For each task, you need to show AVL tree after each insertion/deletion (and some explanation if necessary). If an update requires rotation, you need to identify the case (e.g., left-left case) and show how rotations are done. You should also include the balance factors of each node in your figures and argue why the tree is balanced or unbalanced.

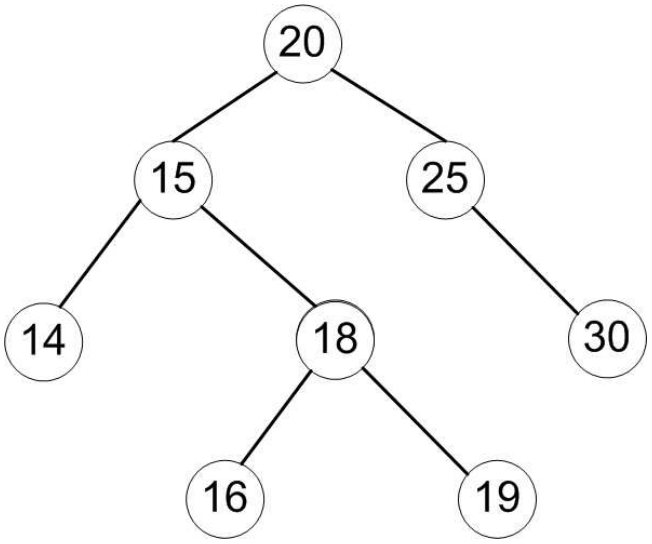

Figure 1: Task 2b