FIT2004 S2/2017: Assessment questions for week 10

THIS PRAC IS ASSESSED! (5 Marks)

DEADLINE: Sunday, 01-Oct-2017 23:55:00 AEST
CLASS: You will be interviewed during your lab by your demonstrator who 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 time and space complexity 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/before your demonstrators have interviewed you, you are expected to work towards the programming competition the details of which will be released soon.
SUBMISSION REQUIREMENT: You will need to submit a zipped file containing your Python program (named navigation.py) as well as a PDF file briefly describing your solution and its space and time complexity. 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. Penalties will be applied if you fail to submit the PDF file. The zipped file is to be submitted on Moodle before the deadline.
Important: The assignments will be checked for plagiarism and collusion using an advanced plagiarism detector and the students will be interviewed by tutors to demonstrate the understanding of their code. Last year, many students were detected by the plagiarism detector and almost all got zero mark for the assignment and, as a result, many failed the unit. "Helping" others is NOT okay. Please do not share your solutions with others. If someone asks you for help, ask them to visit us during consultation hours for help.

Minimum Detour Path

Alice enters in your room with a box of sweets and hands it over to you saying "Here are some sweets for a sweet friend. Your algorithm helped me to optimize my ebay listings and maximize the profit. I am so grateful to you."

You are happy to hear that your hardwork helped her, "Thanks. I must admit that it took me a while to understand the dynamic programming and solve the problems. And I still think I need to practice more to master the dynamic programming. Anyway, never mind Alice".

Alice smiles and says, "Practice makes perfect! :)'.

You nod, "Yes, you are right! Anyhoo, I hope your ebay business is not taking too much of your time and you are having fun."

Alice exhales, "The competition is cut throat but I am having fun. I enjoy coming up with new business strategies to compete with other businesses. For example, I am selling some heavy products for which the postage cost is quite high. To attract more customers, I have introduced a promotion that promises free same-day delivery to one lucky customer everyday. To keep the same-day delivery promise and to save the postage cost, I deliver the order myself, take a selfie while delivering the product and post it on our Facebook page to promote my business. This has significantly increased the number of orders I receive."

You: "That's interesting. But it must be quite hectic to deliver the order yourself.".

Alice: "Exactly! That's why I need your help in choosing the "lucky" customer. Everyday, I receive many orders from local customers (customers living in Melbourne) and my plan is to choose a customer such that delivering to him/her minimizes the distance I need to drive. For example, if I am going from home to work, I want to find the shortest route from home to work that passes through the delivery address of at least one customer. That will be the "lucky" customer ;)"

You get sarcastic: "That's so transparent Alice :P"

Alice ignores your sarcasm and continues: "Well, I have downloaded the road network for Melbourne from OpenStreetMap. I have also written a script that obtains all recent orders from ebay and maps the delivery addresses for each order on the map. However, I do not know much about graph algorithms and am not sure how to find the shortest route."

You: "What a coincidence! We just learned shortest path algorithms yesterday. I was planning to solve some graph problems to improve my understanding. I will be happy to help. Give me the details".

Alice is so happy to hear this and explains the detailed requirements which are formally described below.

Input

The input files `edges.txt` and `customers.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 some lines from `edges.txt`.

```
746 779 1390
747 759 590
747 763 700
```

Each line corresponds to an edge in the road network: first two columns are the IDs of vertices and the third column is the length of the edge connecting the two vertices. E.g., the first line indicates that there is an edge between the vertices with IDs 746 and 779 and the edge length is 1390 meters. All roads in the network are bidirectional, i.e., the graph is **not** directional.

The file `customers.txt` contains the locations (delivery addresses) of customers. We assume that each customer is located on one of the vertices in the graph. We also assume that a vertex cannot have more than one customers. Below are the first 3 lines from `customer.txt` denoting that we have a customer on each of the vertices with IDs $54, 145$ and 200.

```
54 customer
145 customer
200 customer
```

Your program will be tested on a file containing the same road network (same edges and distances) but with different locations of customers (i.e., a different `customers.txt`).

Task 1: Computing shortest path

The first task is to compute the shortest path and shortest distance between any two given vertices s and t. The time complexity of your algorithm must be $O(E \log V)$ and the space complexity must be $O(V + E)$.

Task 2: The minimum detour path

Alice will enter a source vertex s and a target vertex t. The minimum detour path is the shortest route from s to t that passes through at least one customer. Your algorithm must print the minimum detour path and its total length (called minimum detour distance). The following is a naive approach to find the minimum detour path and minimum detour distance.

```
# naive algorithm
minDetourDist = infinity
luckyCustomer = -1
for each customer c
    detourDist = dist(s,c) + dist(c,t) #use Disjkstra's algorithm
    if detourDist < minDetourDist:
        luckyCustomer = c
        minDetourDist = detourDist

minDetourPath = path from s to luckyCustomer + path from luckyCustomer to t
print minDetourDist and shortestDetourPath
```

In the above algorithm the shortest distance/path between two vertices is computed using Dijkstra's algorithm. The for loop in the above algorithm requires calling Dijkstra's algorithm $2C$ times where C is the total number of customers. Therefore, the time complexity of the above algorithm is $O(C \cdot E \log V)$ where E is the total number of edges and V is the total number of vertices in the graph. Your goal is to compute the minimum detour path and minimum detour distance in $O(E \log V)$ time complexity and $O(V + E)$ space complexity.

Output

Your program will ask the user to enter the source vertex s and the target vertex t. It must then print the shortest distance, shortest path, minimum detour distance and minimum detour path. Below are some sample outputs.

```
Enter source vertex: 1710
Enter target vertex: 1730

Shortest Path and Distance:
1710 --> 1718 --> 1724 --> 1727 --> 1730
Total route distance:  3300

Minimum Detour Path and Distance:
1710 --> 1718 --> 1724 --> 1726 --> 1731 --> 1732 (C) --> 1731 --> 1730
Total route distance:  3970
```

In the above example, the shortest distance between vertex 1710 and vertex 1730 is 3300 meters and the minimum detour distance is 3970 meters. The shortest path and the minimum detour path are also shown. In the minimum detour path, the vertex which has a customer located on it is denoted as (C), i.e., the vertex 1732 has a customer located on it.

Below is another example.

```
Enter source vertex: 1741
Enter target vertex: 1753

Shortest Path and Distance:
1741 --> 1737 --> 1753
Total route distance:  1840

Minimum Detour Path and Distance:
1741 --> 1737 --> 1753 --> 1773 --> 1771 (C) --> 1773 --> 1753
Total route distance:  3420
```

Note that it is possible that there are more than one minimum detour paths. In this case, you can print any path of your choice. Your answer will be considered correct if the detour path is the minimal, i.e., there does not exist any other path between s to t that has a smaller distance and contains at least one customer.

In some cases, a minimum detour path may contain more than one customers. In such case, Alice will manually look at the route and decide the "lucky" customer. Your task is only to return the minimum detour path containing at least one customer. In the example below, the minimum detour path contains two customers (1732 and 1771).

```
Enter source vertex: 1256
Enter target vertex: 4977

Shortest Path and Distance:
1256 --> 1258 --> 1275 --> 1266 --> 1272 --> 1601 --> 1631 --> 1706 --> 1707
 --> 1710 --> 1718 --> 1724 --> 1727 --> 1729 --> 1734 --> 1741 --> 1745
 --> 1755 --> 4977
Total route distance:  13110

Minimum Detour Path and Distance:
1256 --> 1263 --> 1270 --> 1287 --> 1557 --> 1562 --> 1565 --> 1569 --> 1572
 --> 1573 --> 1578 --> 1580 --> 1581 --> 1584 --> 1597 --> 1606 --> 1613
 --> 1621 --> 1626 --> 1637 --> 1644 --> 1669 --> 1743 --> 1736 --> 1728
 --> 1733 --> 1732 (C) --> 1739 --> 1758 --> 1761 --> 1767 --> 1771 (C)
 --> 1773 --> 1778 --> 1781 --> 1790 --> 4976 --> 4977
Total route distance:  13550
```

It is also possible that the shortest path may be the same as the minimum detour path. See the example below.

```
Enter source vertex: 1700
Enter target vertex: 1900

Shortest Path and Distance:
1700 --> 1692 --> 1683 --> 1677 --> 1685 --> 1682 --> 1678 --> 1681 --> 1694
 --> 1696 --> 1697 --> 1702 --> 1706 --> 1670 --> 1648 --> 1666 --> 1673
 --> 1768 --> 1762 --> 1736 --> 1744 --> 1746 --> 1759 --> 1763 --> 2436
 --> 2434 --> 2432 --> 2430 --> 2429 --> 2426 --> 2425 --> 2435 --> 2445
 --> 2446 --> 2453 --> 2457 --> 1050 --> 1036 --> 1027 --> 1022 --> 1013
 --> 1006 --> 974 --> 973 --> 966 --> 960 --> 956 --> 952 --> 950 --> 949
 --> 951 (C) --> 959 --> 965 --> 985 --> 1009 --> 1020 --> 1054 --> 4676
 --> 4677 --> 4678 --> 3422 --> 3428 --> 3432 --> 3438 --> 3448 --> 4852
 --> 1903 --> 1881 --> 1865 --> 1888 --> 1900
Total route distance:  40890

Minimum Detour Path and Distance:
1700 --> 1692 --> 1683 --> 1677 --> 1685 --> 1682 --> 1678 --> 1681 --> 1694
 --> 1696 --> 1697 --> 1702 --> 1706 --> 1670 --> 1648 --> 1666 --> 1673
 --> 1768 --> 1762 --> 1736 --> 1744 --> 1746 --> 1759 --> 1763 --> 2436
 --> 2434 --> 2432 --> 2430 --> 2429 --> 2426 --> 2425 --> 2435 --> 2445
 --> 2446 --> 2453 --> 2457 --> 1050 --> 1036 --> 1027 --> 1022 --> 1013
 --> 1006 --> 974 --> 973 --> 966 --> 960 --> 956 --> 952 --> 950 --> 949
 --> 951 (C) --> 959 --> 965 --> 985 --> 1009 --> 1020 --> 1054 --> 4676
 --> 4677 --> 4678 --> 3422 --> 3428 --> 3432 --> 3438 --> 3448 --> 4852
 --> 1903 --> 1881 --> 1865 --> 1888 --> 1900
Total route distance:  40890
```

Note that it is also possible that a customer is located on a source or a target vertex. In the example below, the customer is located on the source vertex.

```
Enter source vertex: 2186
Enter target vertex: 2200

Shortest Path and Distance:
2186 (C) --> 2166 --> 2157 --> 2149 --> 2148 --> 2150 --> 2152 --> 2158
 --> 2173 --> 2185 --> 2198 --> 2200
Total route distance:  9050

Minimum Detour Path and Distance:
2186 (C) --> 2166 --> 2157 --> 2149 --> 2148 --> 2150 --> 2152 --> 2158
 --> 2173 --> 2185 --> 2198 --> 2200
Total route distance:  9050
```

The user may enter the same ID for both source and the target vertices, i.e., Alice may want to start from her home, deliver an item and then return to her home. Below are some examples.

```
Enter source vertex: 1700
Enter target vertex: 1700

Shortest Path and Distance:
1700
Total route distance:  0

Minimum Detour Path and Distance:
1700 --> 1692 --> 1683 --> 1677 --> 1675 --> 1676 --> 1643 --> 1618 (C)
 --> 1643 --> 1676 --> 1675 --> 1677 --> 1683 --> 1692 --> 1700
Total route distance:  9380
```

In the example below, the target vertex is the same as the source vertex and a customer is located on the source vertex. Therefore, the shortest distance and minimum detour distance both are 0.

```
Enter source vertex: 1618
Enter target vertex: 1618

Shortest Path and Distance:
1618 (C)
Total route distance:  0

Minimum Detour Path and Distance:
1618 (C)
Total route distance:  0
```

Things to note

It is strongly recommended that you implement the naive algorithm discussed above. This can be implemented quite easily once you have completed Task 1. You must then verify your algorithm for Task 2 by comparing its results with the results of the naive algorithm. Also, make sure that you check your algorithm for special cases (e.g., when the source and target are the same and a customer is located on it).

You will lose all marks for a given task, if your algorithm does not produce correct results for the task. Also, heavy penalties are applied if your complexity is worse than the required complexity and you may lose all marks if the complexity is similar to that of the naive algorithm. This is because our focus is on developing efficient algorithms and data structures. Task 2 is the critical part of the assignment and is worth majority of the marks for this assignment. Therefore, do not assume that you will get around 50% marks by completing only the task 1.

-=o0o=-
END
-=o0o=-