BLG336E – Analysis of Algorithms II Homework 2

Release Date: April 1, 2020
Due Date: April 22, 2020, 23.59

Late submissions will NOT be accepted. Please submit your homeworks only through Ninova. This is an individual homework and in case of inappropriate exchange of information is detected, disciplinary actions will be taken.

- You should write your code in C++ language and use object-oriented approach.
- Write your name and ID on the top of each document that you will upload.
- Compile your code on Linux using g++. You can test your code through ssh.
- Your codes must be compiled with a command like **g++ student_id.cpp** and run with **./a.out test.txt.** If you use another instruction please write it in your report.
- You are supposed to prepare a report file where you explain your algorithm and share your test case results in pdf format.
- Use comments to explain your code.
- If you have any questions, please contact me via akti15@itu.edu.tr
- **Submit your homework in a .zip file where it contains all of the necessary .h and .cpp files together with a report in .pdf format.**

Problem Description

Joseph and Lucy are best friends and they have arranged a travel together two months ago and reserved their flying tickets. However unfortunately, in this two months, they have fought over a small issue and they are not in good terms, so do not want to face each other in any case. Even so, they did not cancel the travel since everything was planned before. They decided to do check-in separately and stay at different hotels during the travel. Also, when going around the city, they do not want to run against each other. That is why we must arrange their routes from their hotels to the their destinations and from destinations to the hotels for one day. The rules are listed below:

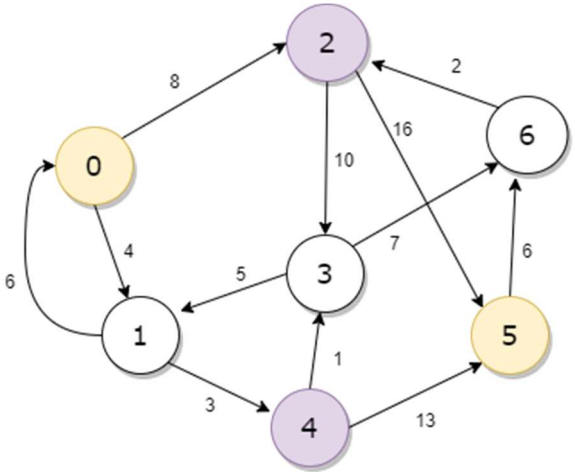
- The city map is represented as a **directed** and **weighted** graph where nodes are places and edges are roads. Edge weights are the lengths of roads.
- Two of the nodes are Joseph's hotel (JH) and Lucy's hotel (LH). The other two of the nodes are the destination for Joseph (JD) and destination for Lucy (LD).
- If there is a path from node A to B with length of n, then it takes n minutes to get to B from A. Assume that the speed of the Lucy and Joseph are the same.
- Joseph and Lucy leave their hotels at the same time and they must not be in the same node at the same time.
- They spend 30 minutes at the destination before returning their hotels.

You are supposed to find the shortest paths from hotels to destinations and destinations to the hotels that are not intersecting considering the instant locations of Joseph and Lucy.

PART 1 – Constructing the Graph

There is an example graph given below where the node 0 is JH, node 5 is JD, node 2 is LH and node 4 is LD. At the right of the graph, the format of the input files is given as the first line gives the number of nodes in the city (N) and the following line contains the hotels and destinations in following order: JH JD LH LD

The lines below gives the edge information as: source node, target node, weight respectively.




```
7
0 5 2 4
0 1 4
0 2 8
1 0 6
1 4 3
2 3 10
2 5 16
...
```

You must use the adjacency matrix representation for the graph construction. The txt file for graph input must be given as a command line argument.

PART 2 – Implementing the algorithm

You are supposed to implement a shortest path algorithm (i.e. Dijkstra). You need to find 4 shortest paths for each input as:

- **JH to JD** and **JD to JH**
- **LH to LD** and **LD to LH**

Remember that edge weights represent how many minutes it takes to get from one node to another and Lucy and Joseph cannot be at the same node at the same time.

So what to do step by step is like this:

- Find the shortest paths from hotels to destinations.
- Check if there are any case that Joseph and Lucy are at the same node considering the durations.
- If there is an intersection, find an alternative path which is valid and shortest. You should try to change either Joseph's or Lucy's path and choose the better one. If there is no alternative paths for both of them, then print "No solution!" on the screen and terminate the program.
- Then, find the shortest paths from destinations to hotels in the same manner. Note that they have waited at the destination for 30 minutes.

For the example graph given above we can find the shortest paths to destinations as follows:

JH (0) to JD (5) :

Node	0	1	4	5
Time	0	4	7	20

LH (2) to LD (4):

Node	2	3	1	4
Time	0	10	15	18

There is no intersection where Joseph and Lucy are at the same node as you can see in the tables. After waiting for 30 minutes at destinations they go back to the hotels using shortest paths again.

JD (5) to JH (0):

Node	5	6	2	3	1	0
Time	50	56	58	68	73	79

LD (4) to LH (2):

Node	4	3	6	2
Time	48	49	56	58

As you can observe in highlighted rows, they met at node 6 at minute 56. So we need to find an alternative second shortest path for Lucy which is 4-3-1-0-2. Because we have no choices for Joseph other than 5-6.

LD (4) to LH (2):

Node	4	3	1	0	2
Time	48	49	54	60	68

Sample output:

```
Joseph's Path, duration: 79
Node: 0  Time: 0
Node: 1  Time: 4
Node: 4  Time: 7
Node: 5  Time: 20
-- return --
Node: 5  Time: 50
Node: 6  Time: 56
Node: 2  Time: 58
Node: 3  Time: 68
Node: 1  Time: 73
Node: 0  Time: 79

Lucy's Path, duration: 68
Node: 2  Time: 0
Node: 3  Time: 10
Node: 1  Time: 15
Node: 4  Time: 18
-- return --
Node: 4  Time: 48
Node: 3  Time: 49
Node: 1  Time: 54
Node: 0  Time: 60
Node: 2  Time: 68
```

Hint: When comparing the alternative paths for Joseph and Lucy, please consider

(1) if there is an alternative path
(2) whether if this new path has any intersections
(3) if there are two alternative solutions without intersection, choose the shortest one overall (J + L).

Note: The test cases will be relatively simple where it is enough to find one alternative path for each person in case of intersection. This will give the result so you do not have to check further. (i.e there will be **no cases** where there are also intersections for all of the alternative paths, requiring a second iteration for alternative path search.)