Data Structures and Algorithms (CSE225) Laboratory

Assessment-Based Final Examination. Lab Instructor: Tonmoay Deb

Total Marks: 100 (25 marks per problem)

Submission Deadline: 4th June 2020

RULES TO FOLLOW:

- Answer four problems. One problem is linked to the other.
- Each and every topic in the question has already been discussed in the classes (online or in-person).
- Plagiarism is STRICTLY PROHIBITED. At most, you can discuss with your PAIR, but can not copy any content from anyone. You need to write everything yourself.
- Marks are counted based on your code writing and logic formulation, not based on your code's ability to run.
- You need to write all codes within ".CPP, .H FILES." If you have confusion on how to do this, take a look in the last class on the project and see where we made the option systems.
- You will submit only the ".CPP and .H" files. NO ".ZIP" FILES OR ANY OTHER EXTENSIONS ARE ALLOWED.
- Questions are very easy; I suggest you go through the online video recordings to recap everything.
- If you have any single queries about the question, directly post them into the Google Classroom. I will not answer any of your queries over email to keep things transparent.

Problem 1: Let’s get started

Suppose you are developing a transportation facility application. In this sense, you are supposed to design a Graph Network where the nodes will be the Divisions only. And the edges will be the distance from each division to another. The graph will be bi-directional, so there won’t be any arrows in the edge lines.

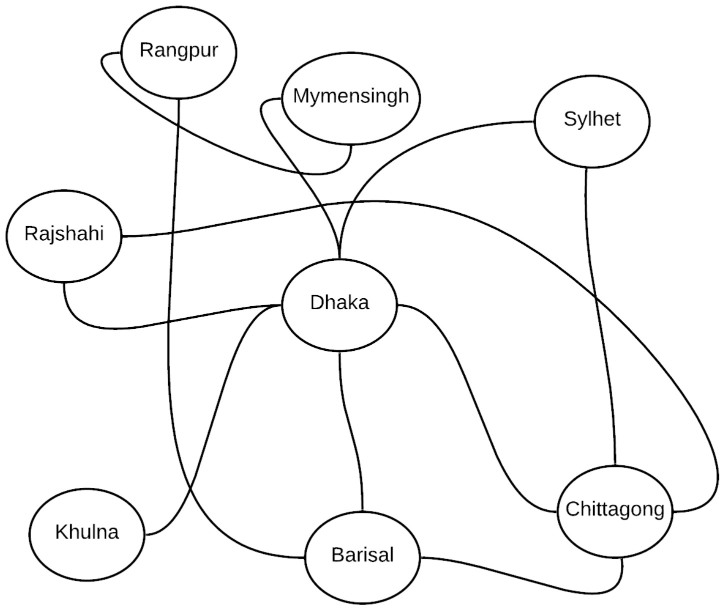
Step 1: Create a “TransportGraph.cpp” and “TransportGraph.h” file and set your favorite class name.

Step 2: Your graph Will be having a 2D Array (you know) to map the edges from one place to another.

Step 3: Write down a “main.cpp,” which will be used to enter information from one place to another. So, you need to enter 64 distances (edges) in total. Distance will be from each division to another. Initially, just write a random number of edge connections, don’t need to be 64.

Step 4: Print the Adjacency matrix (the 2D Array) in the main function. Remember, you need to write the function in your TransportGraph class, not in the main function.

Below is an **example** image of the graph of our divisions connected with random edges. However, your graph should take input of all the edges and the weights, respectively.



Problem 2: We need to find the best Route!

So, you have already built the graph. Congratulations! One simple task is left. Suppose, we need to deliver a product from Chittagong to Rangpur, now, which is the fastest way to reach there? Write two functions (DFSGraph and BFSGraph) for solving the task.

Step 1: Write a function, DFSGraph, on your TransportGraph class and run the function from your main function. The function should print the immediate paths (the DFS tree nodes we discussed in the online class) from Chittagong to Rangpur.

Step 2: Do the identical thing for the BFSGraph function.

Step 3: Upgrade the system, make this parameterized, so that we can give any two divisions as input we want and get the immediate path.

Problem 3: Let’s put some effort into learning the data structure!

We can now generate the path right now. That means we have the information about the immediate nodes from start to destination. Your task is to create a Linked List from the output you got and print the list. You need to do this into your TransportGraph class by writing a function.

For example, we have for the immediate path for Chittagong to Rangpur as:

Chittagong -> Sylhet -> Barisal -> Dhaka -> Rangpur.

You need to create a linked list from this information. Super simple!

Problem 4: Let us wrap the graph up!

So far, you are already highly skilled in learning how to make a graph, converting a graph output into the LinkedList. Now we will do one simple thing, which is a piece of cake if you already came up to this stage.

For a given node as input, you need to create a linked list of the adjacent nodes in sorted order. We already learned how to make a sorted list (Yay!)

Let's see an example:

For Dhaka, the Adjacent nodes are (from the picture):

Dhaka-> Chittagong -> Rajshahi -> Khulna -> Barisal -> Sylhet -> Mymensingh

The LinkedList output will be:

Dhaka (it will not change) -> Barisal -> Chittagong -> Khulna -> Mymensingh -> Rajshahi -> Sylhet

So, you need to make function, printSortedList() where the input will be the node (e.g., Dhaka), and the output will be a LinkedList, which will be sorted based on the first character value of the string (the division name).