Lab 12

100 Points

Due date:
11:59pm, Monday 05/06/2019 for Tuesday labs.
11:59pm, Wednesday 05/08/2019 for Thursday labs.

Purpose:
The purpose of this lab is to implement a graph data structure to help a city to build an efficient road system.

General Requirements:
For this lab, you are required to implement a graph data structure using an adjacency matrix and then implement Kruskal's and Prim's algorithm to calculate the minimum spanning tree (MST), please refer to slides 18 and 21 in note package 8 for Kruskal's and Prim's algorithm, respectively. At the same time, you will implement depth-first search and breadth-first search traversals on the graph data structure, please refer to slide 25 in note package 8 for depth-first search and breadth-first search. These implementations will help you build an efficient road system.

Bus service system:
The map of the city is given to you and specifies the distances between each pair of points in the city. The city bus system needs to build an appropriate number of bus stops based on the cost computation required to go from one area to another. You need to construct a bus map with the help of Kruskal's and Prim's algorithms.

Operations on graph:
Implement the following operations for the graph data structure:

- **Buildgraph(costs, n)** - should build a graph, it accepts 2 arguments – a square 2D array of integer costs representing the cost of each edge and n, the number of nodes in the graph.
- **BFS()** - the method performs a breadth-first search on the graph. It should return an array of size 2 for which each entry contains a pointer that points to an array. The first array should contain the tree edges of the graph. The second array should contain the cross edges.
- **DFS()** - the method performs a depth-first search on the graph. It should return an array of size 2 for which each entry contains a pointer that points to an array. The first array should contain the tree edges of the graph. The second array should contain the back edges.
- **Kruskal()** - implement Kruskal's algorithm that computes a MST for a given graph. At every step of adding a new edge, you will need to ensure that no cycle is created. You may use the disjoint-set data structure and a priority queue—min-5 heap—that you have implemented in the previous lab for cycle detection and obtaining a minimum cost

edge. The algorithm should return an array of edges and the costs of the corresponding edges. The returned array will represent the MST.

- **Prim()** - implement Prim's algorithm that computes a MST for a given graph. You may use a priority queue —min-5 heap—that you implemented in the previous lab for finding a minimum cost edge. The algorithm should return an array of edges and the costs of the corresponding edges. The returned array will represent the MST.

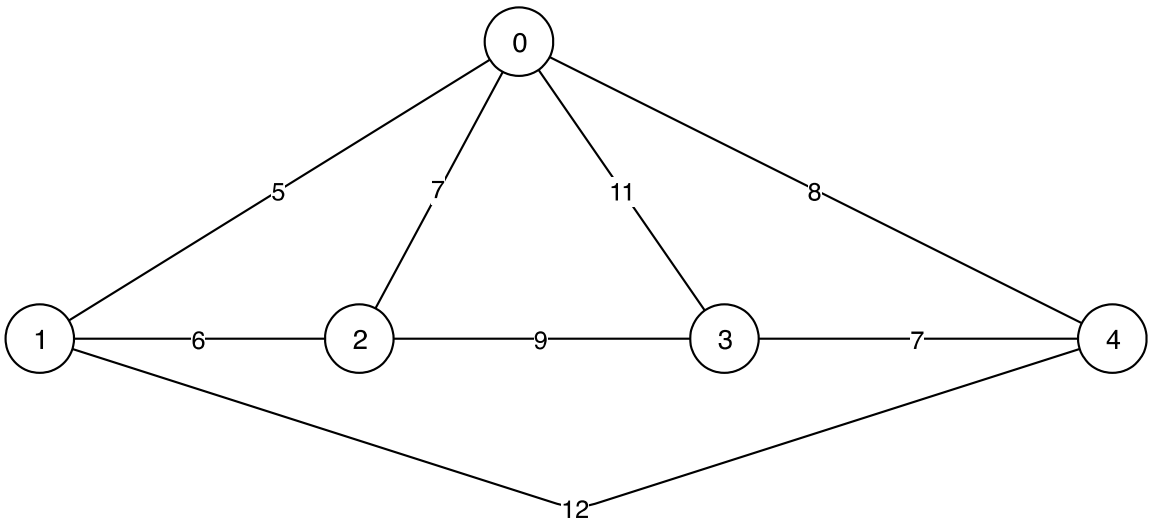
You will have to construct a simple UI like this:

Please choose one of the following commands:
1- BFS
2- DFS
3- Kruskal MST
4- Prim MST
5- Exit

Data file:
data.txt:
5
0 5 7 11 8
5 0 6 -1 12
7 6 0 9 -1
11 -1 9 0 7
8 12 -1 7 0

The first line mentions the number of bus stops (i.e., n) to be connected. The second line is the label of the bus stop followed by the costs to connect the bus stop to the remaining stops in numerical order. A value of -1 implies that no direct road can be built to connect the two bus stops.

On the next page, we demonstrate the graph for data.txt:



Expected output

Please choose one of the following commands:

1- BFS
2- DFS
3- Kruskal MST
4- Prim MST
5- Exit

> 1

Tree Edges: (0,1) (0,2) (0,3) (0,4)
Cross Edges: (1,2) (1,4) (2,3) (3,4)

Please choose one of the following commands:

1- BFS
2- DFS
3- Kruskal MST
4- Prim MST
5- Exit

> 2

Tree Edges: (0,1) (1,2) (2,3) (3,4)
Back Edges: (0,2) (0,3) (0,4) (1,4)

Please choose one of the following commands:

1- BFS
2- DFS
3- Kruskal MST
4- Prim MST

5- Exit
> 3
(0, 1){5} (1,2){6} (3,4){7} (0, 4){8}
Total cost = 26

Please choose one of the following commands:
1- BFS
2- DFS
3- Kruskal MST
4- Prim MST
5- Exit
> 4
(0, 1){5} (1,2){6} (0,4){8} (3,4){7}
Total cost = 26

Please choose one of the following commands:
1- BFS
2- DFS
3- Kruskal MST
4- Prim MST
5- Exit
> 5
Byebye!

Questions
Please answer the following questions in no more than 5 lines each and submit it in PDF format along with your implemented code.

1. What is the worst case algorithmic asymptotic complexity, i.e., O(?) of each of the operations that you have implemented? Assume, Graph G = (V, E), |V | = n, |E| = m.
    a. Finding a minimum spanning tree using Kruskal's algorithm.
    b. Finding a minimum spanning tree using Prim's algorithm.
2. Can you find a simple graph example where Kruskal's algorithm and Prim's algorithm both fail? What is the output of your implemented method for such a case? Does it fail to generate the correct MST?

Submission:
Follow the conventions below to facilitate grading:

Source Code

Place all your source files (*.cpp, *.hpp) and input files in a single folder with no subfolders.

- Name your folder using the convention Lastname_LabX (e.g., Smith_Lab12).
- Include a functioning Makefile inside the folder. (The makefile should also include the clean command.)
- **Verify that your code runs on the lab Linux machines before submission.**

Report

- **Please include your answers (answers.pdf) in your folder before compressing it.**

Compressed File

- Compress using .zip, .rar, or .tar.gz.
- Name your file using the convention Lastname_LabX (e.g., Smith_Lab12.zip).

Email

- Use the following subject for your email: Lastname_LabX (e.g., Smith_Lab12).
- Send your code to l290w868@ku.edu if you are in one of Lei's sections or to dhwanipandya1401@ku.edu if you are in one of Dhwani's sections.
- Anytime you have a question about the lab, put the word question in the subject of the email.