Zülal Karın **6/05/2022**
12622989076 **Section 02**

CMPE 343 HW-2 REPORT

Problem Statement and Code Design

Task 1: In the first question, we are asked to connect the given computers first, then answer the queries. I created an edge weighted graph by connecting computers together. For this, I used the Edge class and the EdgeWeightedGraph class. I took the distance between computers as the edge weight. In order for all computers to be connected to each other, I used the minimum spanning tree, and finally, to answer the queries, I created a new graph consisting of mst edges and found the distance between the computers with the bredth first search algorithm. I have put a Structure Chart below for you to better understand the steps I followed.

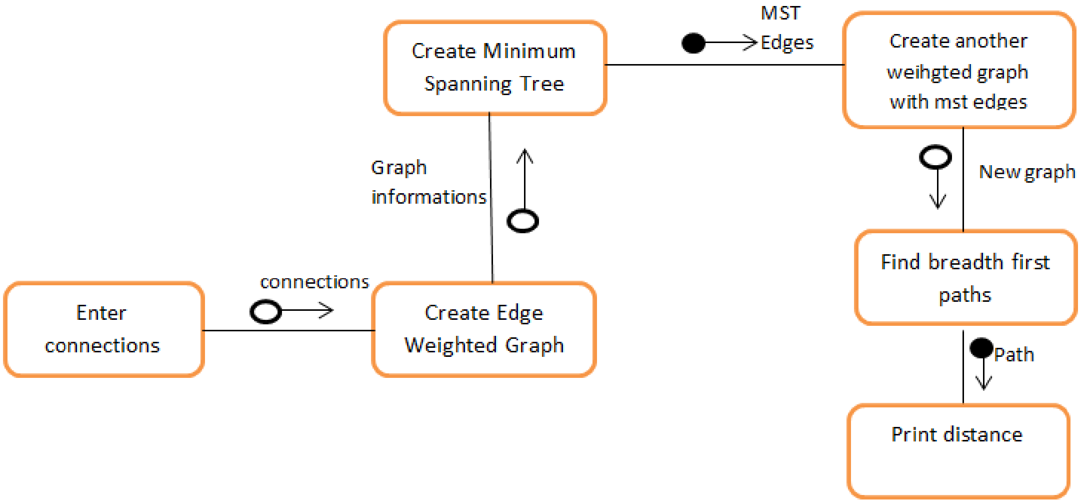


figure 1

Task 2: In the second question, I am asked to make a weighted graph consisting of auditoriums. Each auditorium has a seating capacity. I am asked to place the given partipant number in the auditoriums in a way that will create the shortest path. To do this, I used the Dijkstra Shortest Path algorithm slightly modified. For example, instead of starting the source vertex in weight from 0, I started from the given distance. Because this path was common for all participants. You can see the structure I created more clearly in the structural diagram below.

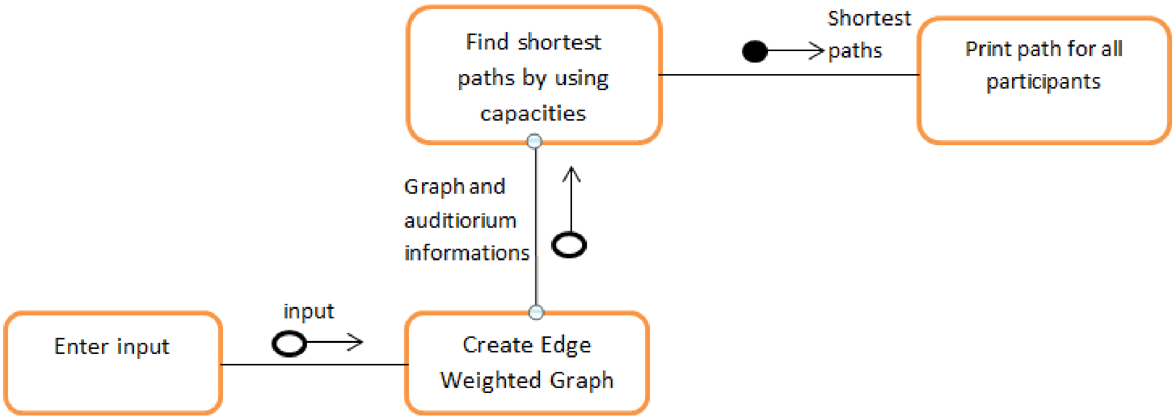

Find shortest paths by using capacities
Shortest paths
Print path for all participants
Graph and auditiorium informations
input
Enter input
Create Edge Weighted Graph

figure 2

Implementation and Functionality

Task 1: For this task, mainly I created Edge, EdgeWeightedGraph which contains main method, KruskalMST class and BreathFirstPaths class. I will describe these classes and the methods within them using the Structure Diagram in figure1.

1)Create edge weighted graph: Inside the EdgeWeightedGraph class, I first created weighted edges using the Edge class by separating the input according to the space character. Then I created a graph with these edges.

2)Create minimum spanning tree: Since all computers must be connected to each other with LAN cable, I had to use a minimum spanning tree. Because MST covers all vertexes. I used the Kruskal algorithm for this.

3) Create another graph with MST edges: Now, I had an MST. but I could only respond to the given queries using MST edges. So I created a new graph (newGraph) that contains the same vertices but only the MST edges.

4) find distances with bfs algorithm : Finally, I modified the bfs algorithm to find the distance between two computers. For this, I took advantage of the eiter() and other(v) methods of the Edge class.

Task 2: For this task, I created Edge, EdgeWeightedGraph, Driver class, DijkstraSP class and IndexMinPQ classes. I will describe these classes and the methods within them using the Structure Diagram in figure2.

1)Create edge weighted graph: Just like in part1, I created a graph object from the EdgeWeightedGraph class using the given u, v and w values.

2)Find Shortest Paths : The fact that all vertexes have a capacity made my job very difficult in this part. I was asked to find the seat at the shortest distance for each participant. I used the Kruskal shortest path algorithm with the KruskalSP class for this. I changed many things while using the algorithm. The original algorithm was created for directed graphs. I converted it to undirected graph. Then I changed the relax method with its parameters. I added the number of participants to the parameter and Graph to reach the capacity of vertexes. I reduced the capacity and the number of partipants by 1 for each path I found.

3)Print paths for all participants: In the relax method I modified, I printed the distance I found using the distTo method for all participants. Later, I reduced the number of participants and the auditorium's capacity by 1.

Testing

Task 1: My code is successfully passed all the tests. Here, I want to talk about the key points that make my code pass through the test. The first is to find the MST edges as you can see in Figure-3 and create a new graph from them. The second is to modify the BFS algorithm according to Edge Weight Graph, as you can see in Figure-4. In Figure-5, you can see the queries I entered for the graph in Figure-3 and their outputs.

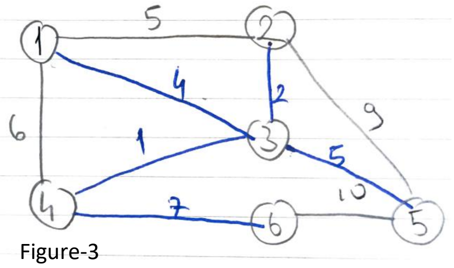
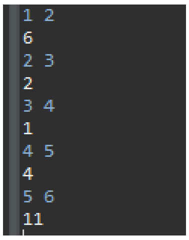
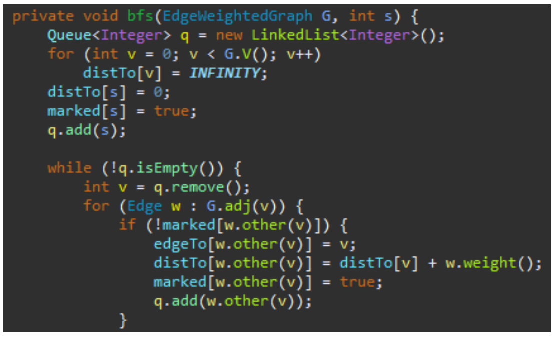


Figure-3



```
1 2
6
2 3
2
3 4
1
4 5
4
5 6
11
```

Figure-5

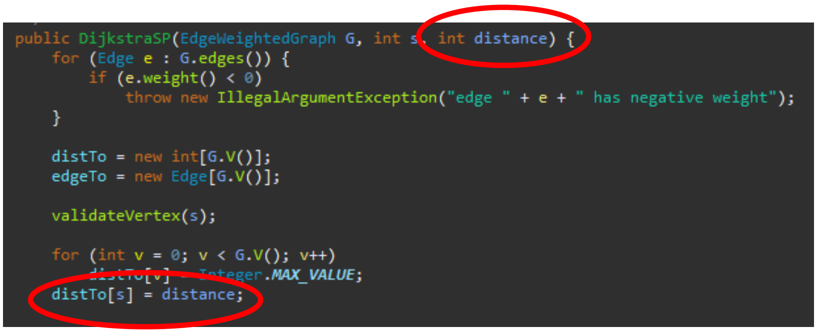


```
private void bfs(EdgeWeightedGraph G, int s) {
    Queue<Integer> q = new LinkedList<Integer>();
    for (int v = 0; v < G.V(); v++)
        distTo[v] = INFINITY;
    distTo[s] = 0;
    marked[s] = true;
    q.add(s);

    while (!q.isEmpty()) {
        int v = q.remove();
        for (Edge w : G.adj(v)) {
            if (!marked[w.other(v)]) {
                edgeTo[w.other(v)] = v;
                distTo[w.other(v)] = distTo[v] + w.weight();
                marked[w.other(v)] = true;
                q.add(w.other(v));
            }
```

Figure-4

Task 2: In my code, I correctly generated the edge weighted graph based on the inputs provided. For each test case, I was able to calculate the distance between the auditorium and the conference hall correctly. You can see this process in figure-6. The most crucial part in my code was my relax method. I have explained my relax method in detail below.



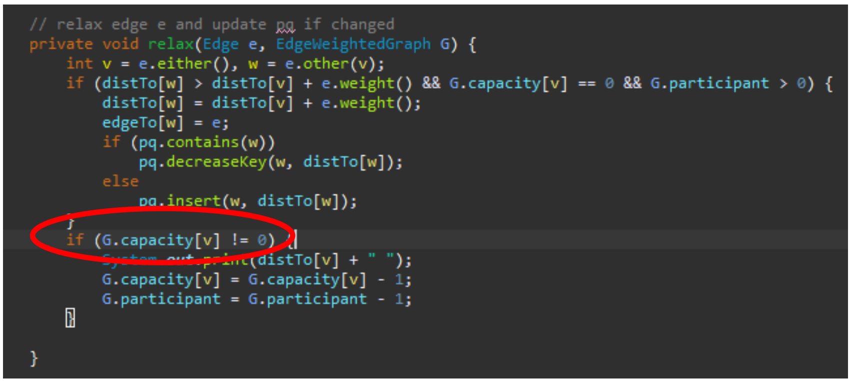
```
public DijkstraSP(EdgeWeightedGraph G, int s, int distance) {
    for (Edge e : G.edges()) {
        if (e.weight() < 0)
            throw new IllegalArgumentException("edge " + e + " has negative weight");
    }

    distTo = new int[G.V()];
    edgeTo = new Edge[G.V()];

    validateVertex(s);

    for (int v = 0; v < G.V(); v++)
        distTo[v] = Integer.MAX_VALUE;
    distTo[s] = distance;
```

Figure-6



```
// relax edge e and update pq if changed
private void relax(Edge e, EdgeWeightedGraph G) {
    int v = e.either(), w = e.other(v);
    if (distTo[w] > distTo[v] + e.weight() && G.capacity[v] == 0 && G.participant > 0) {
        distTo[w] = distTo[v] + e.weight();
        edgeTo[w] = e;
        if (pq.contains(w))
            pq.decreaseKey(w, distTo[w]);
        else
            pq.insert(w, distTo[w]);
    }
    if (G.capacity[v] != 0) {
        System.out.print(distTo[v] + " ");
        G.capacity[v] = G.capacity[v] - 1;
        G.participant = G.participant - 1;
    }

}
```

Figure-7

Inside the if statement, I checked the capacity of the auditoriums. If the capacity is still not 0, I stayed on the same vertex without applying the relaxation operation.

Final Assessments

- For the first part, the output was wrong at first because I forgot that the vertex number starts from 0 while graphing. Later, I noticed the error and fixed it. Small mistakes can have big consequences.

-For part 2, having all the auditoriu's capacity made my job very difficult. For this, I had to modify the Kruskal algorithm. I had a hard time doing this.

-In conclusion, both questions were instructive enough for me. Especially in the second part, I pushed the limits of my brain.