ID: 12859191938
Name: Arda Tarım
Section: 1
Assignment number: 3

Problem Statement and Code Design

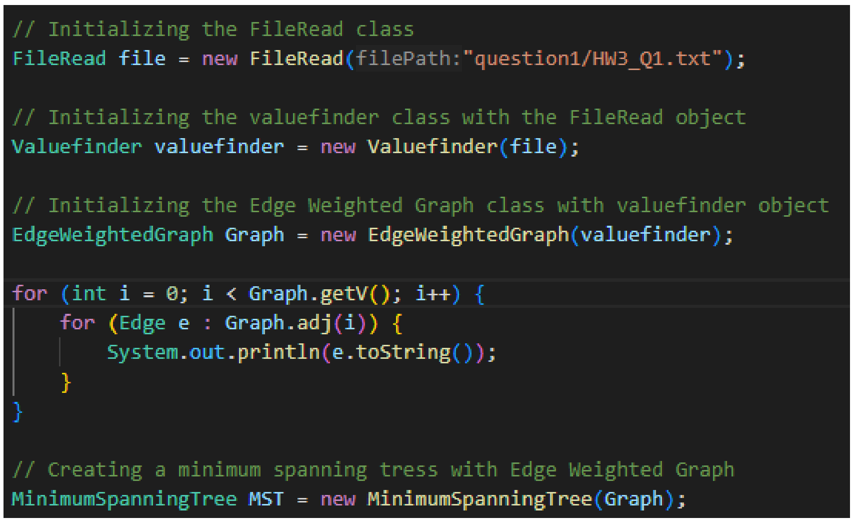
There are two questions in the assignment and they both require a weighted graph data structure. Firstly I created an Edge class. I used my custom IterableList class to implement a Weighted Graph for question 1. Then I created a DirectedEdge class to implement a Directed Weighted Graph for question 2

For question 1, I implemented a MinimumSpanningTree class, using the lazy implementation of Prim's Algorithm. I was able to create a minimum spanning tree that includes the source vertex with this class. Then for the solution of question 2, I implemented the Dijktras class. In this class I used Dijktra's Algorithm for finding the shortest paths. With this algorithm I was able to find shortest paths from 0 to 1,2,3 vertices.

Implementation and Functionality

HW3_Q1_solution

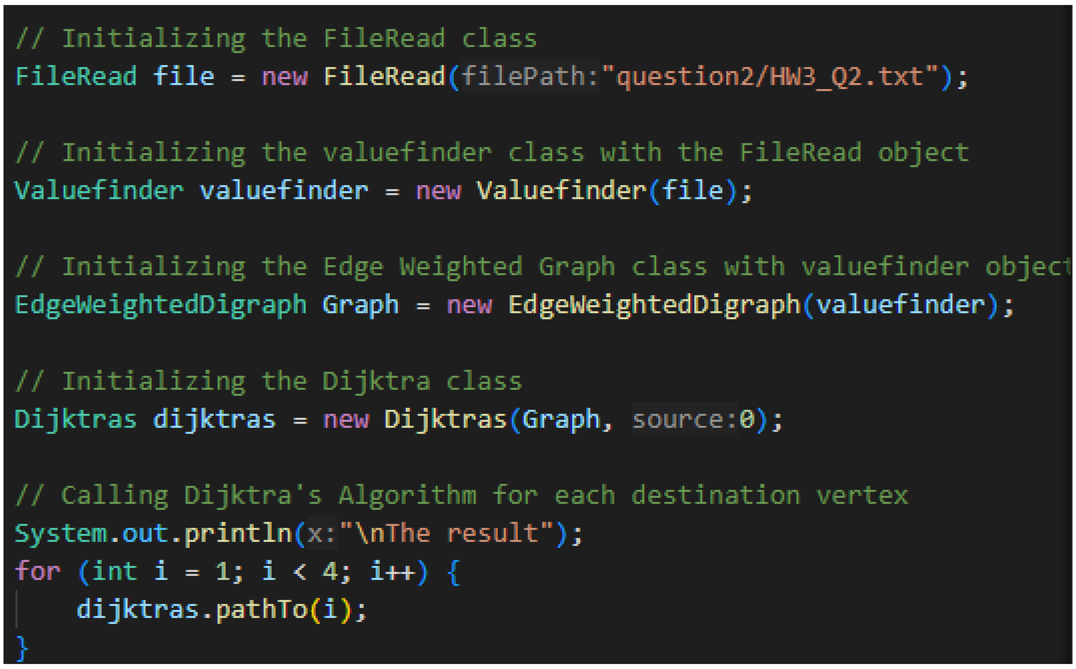
I initialized FileRead and Valuefinder classes to read input from txt file. Then I created a Weighted Graph from the input data. Then I initialized the Minimum Spanning Tree class from my Weighted Graph. Finally, MST data is printed to the command line.



```
// Initializing the FileRead class
FileRead file = new FileRead(filePath:"question1/HW3_Q1.txt");

// Initializing the valuefinder class with the FileRead object
Valuefinder valuefinder = new Valuefinder(file);

// Initializing the Edge Weighted Graph class with valuefinder object
EdgeWeightedGraph Graph = new EdgeWeightedGraph(valuefinder);

for (int i = 0; i < Graph.getV(); i++) {
    for (Edge e : Graph.adj(i)) {
        System.out.println(e.toString());
    }
}

// Creating a minimum spanning tress with Edge Weighted Graph
MinimumSpanningTree MST = new MinimumSpanningTree(Graph);
```

HW3_Q2_solution

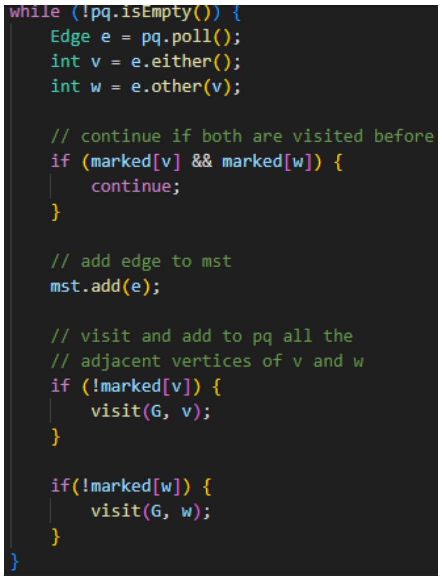
I initialized FileRead and Valuefinder classes to read input from txt file. Then I created a Weighted Digraph from the input data. Then I initialized the Dijktra’s and chose vertex 0 as the source vertex. Then, I printed the shortest path from 0 to 1,2,3 vertices to the command line with their total weight values.



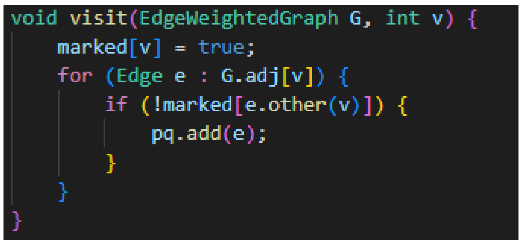
```
// Initializing the FileRead class
FileRead file = new FileRead(filePath:"question2/HW3_Q2.txt");

// Initializing the valuefinder class with the FileRead object
Valuefinder valuefinder = new Valuefinder(file);

// Initializing the Edge Weighted Graph class with valuefinder object
EdgeWeightedDigraph Graph = new EdgeWeightedDigraph(valuefinder);

// Initializing the Dijktra class
Dijktras dijktras = new Dijktras(Graph, source:0);

// Calling Dijktra's Algorithm for each destination vertex
System.out.println(x:"\nThe result");
for (int i = 1; i < 4; i++) {
    dijktras.pathTo(i);
}
```

Minimum Spanning Tree

This class creates a MinimumSpanningTree from an Edge Weighted Graph object. The algorithm starts from the source vertex and visits the adjacent vertices. Then the vertex with the smallest value is added to the tree. Then the algorithm visits the new adjacent vertices to expand the tree until all vertices are visited.



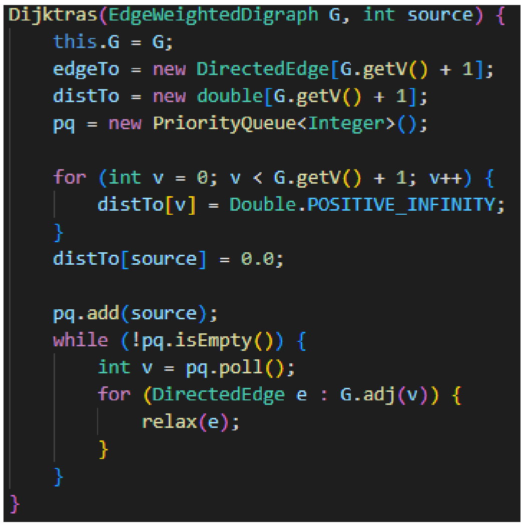
```
while (!pq.isEmpty()) {
    Edge e = pq.poll();
    int v = e.either();
    int w = e.other(v);

    // continue if both are visited before
    if (marked[v] && marked[w]) {
        continue;
    }

    // add edge to mst
    mst.add(e);

    // visit and add to pq all the
    // adjacent vertices of v and w
    if (!marked[v]) {
        visit(G, v);
    }

    if(!marked[w]) {
        visit(G, w);
    }
}
```



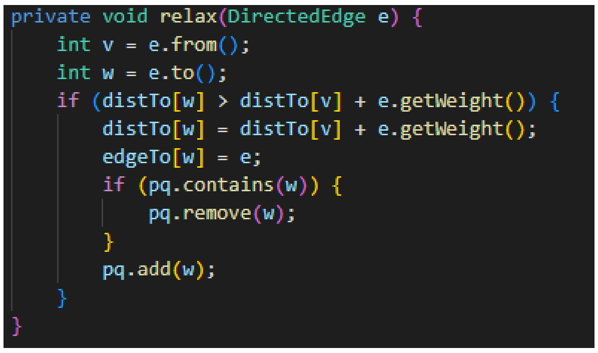
```
void visit(EdgeWeightedGraph G, int v) {
    marked[v] = true;
    for (Edge e : G.adj[v]) {
        if (!marked[e.other(v)]) {
            pq.add(e);
        }
    }
}
```

DijktraShortestPath

edgeTo and distTo arrays and priority queue are initialized in the constructor. All the distances except the source vertex in distTo class are set to positive infinite value. Source vertex is added to the pq. Algorithm picks the smallest edge from the pq and applies edge relaxation to all adjacent vertices. Edge relaxation: If there is a shorter path to the next vertex, the longer path to vertex is removed and shorter one is used. For example if A to C is 300 and A to B is 100 and B to C is 100. Edge relaxation removes the A to C option because there is a shorter way.



```
Dijktras(EdgeWeightedDigraph G, int source) {
    this.G = G;
    edgeTo = new DirectedEdge[G.getV() + 1];
    distTo = new double[G.getV() + 1];
    pq = new PriorityQueue<Integer>();

    for (int v = 0; v < G.getV() + 1; v++) {
        distTo[v] = Double.POSITIVE_INFINITY;
    }
    distTo[source] = 0.0;

    pq.add(source);
    while (!pq.isEmpty()) {
        int v = pq.poll();
        for (DirectedEdge e : G.adj(v)) {
            relax(e);
        }
    }
}
```



```
private void relax(DirectedEdge e) {
    int v = e.from();
    int w = e.to();
    if (distTo[w] > distTo[v] + e.getWeight()) {
        distTo[w] = distTo[v] + e.getWeight();
        edgeTo[w] = e;
        if (pq.contains(w)) {
            pq.remove(w);
        }
        pq.add(w);
    }
}
```

EdgeWeightedGraph & Edge

Implementation of an Edge Weighted Graph data structure. Includes the basic methods getV, getE, addEdge and adj. Can be initialized with a valuefinder object or manually adding edges from the input stream. Uses an array of custom made IterableList class for holding adjacency lists.

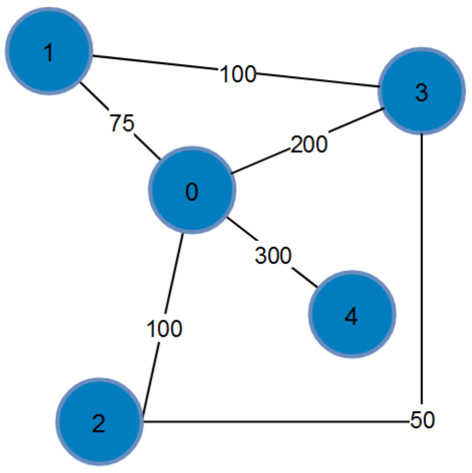
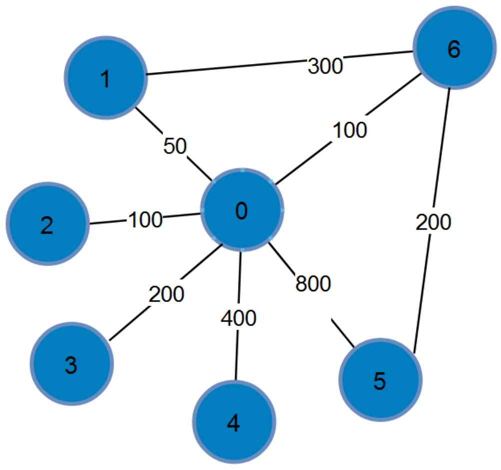
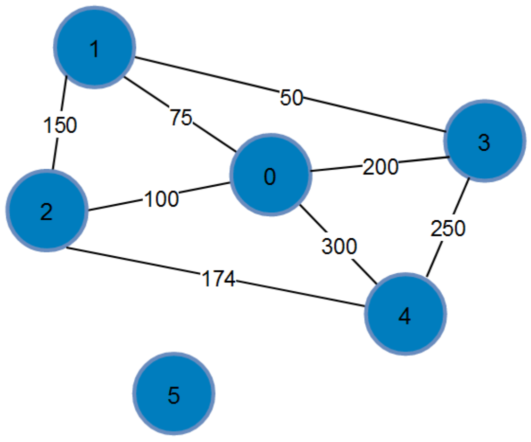
Edge class from vertex v to w with a weight value. Using either() method will return one of the endpoints of the edge and the other(v) method will return the other endpoint of the edge. Also this class implements the Comparable interface for comparing the weight value with other edges.

EdgeWeightedDigraph & DirectedEdge

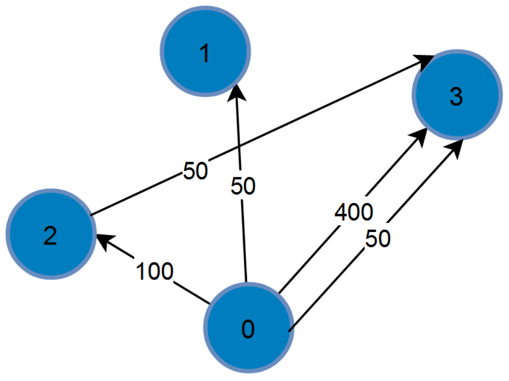
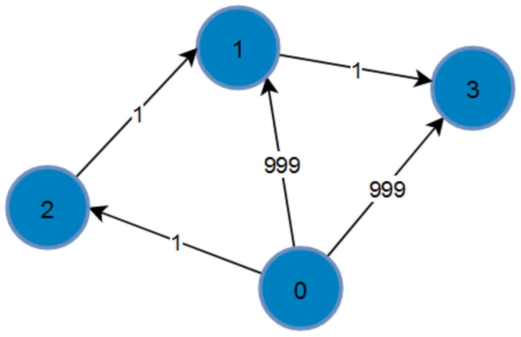
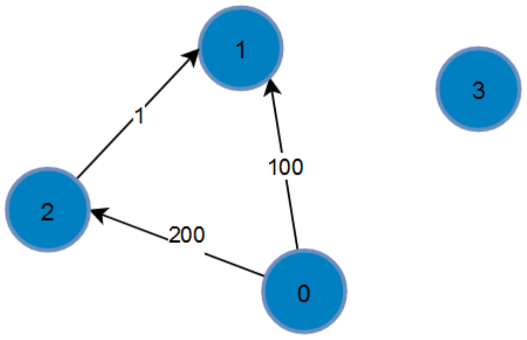
These classes are almost the same classes as Edge Weighted Graph and Edge. The differences are in EdgeWeightedDiagraph we only add the edge to the adjacency list of one vertice and DirectedEdge has from() and to() methods instead of either() and other() methods.

Testing

Testing for Q1

 1 3 0 4 2 100 75 200 300 100 50	The Minimum Spanning Tree Path 0 1 75 0 2 100 2 3 50 0 4 300 Expected Output
 1 6 2 0 3 4 5 300 50 100 100 200 800 400 200	The Minimum Spanning Tree Path 0 1 50 0 2 100 0 6 100 0 3 200 6 5 200 0 4 400 Expected Output
 1 3 0 2 4 5 50 150 75 200 100 300 250 174	Exception in thread "main" java.lang.NullPointerException: Cannot invoke "java.lang.Iterable.iterator()" because the return value of "question1.EdgeWeightedGraph.adj(int)" is null Unconnected vertex causes an error. Because MST must include all vertices.

Testing for Q2

	The result 0 1 50 0 2 100 0 3 50 Expected output
	The result 0 2 1 2 0 2 1 0 2 1 3 3 Expected Output
	The result 0 1 100 0 2 200 Expected Output Unconnected vertex didn’t cause an error.

Final Assessments

Which parts were the most challenging for you?

I had troubles implementing the Dijktra's algorithm.

What did you like about the assignment? What did you learn from it?

The assignment helped me to understand the logic behind the minimum spanning tree and shortest path algorithms. I found it helpful for my studies.

References

Algorithms, FOURTH EDITION, Robert Sedgewick and Kevin Wayne