CMPE 224/343 HW2 REPORT

I) INFORMATION

ID: 47113751436 **Assignment number: 03**

Name: Doğukan Us **Section: 03**

II) PROBLEM STATEMENT AND CODE DESIGN

Task 1:

My first task is about a electricity distribution company's efficient electric distribution problem. With the given input from user, I was asked to create a undirected edge weighted graph and insert the cities to the vertices and connect them with undirected weighted edges according to their coordinates and find a ***Minimum Spanning Tree***.

Structure chart of Task 1 is shown in *Figure 1*.

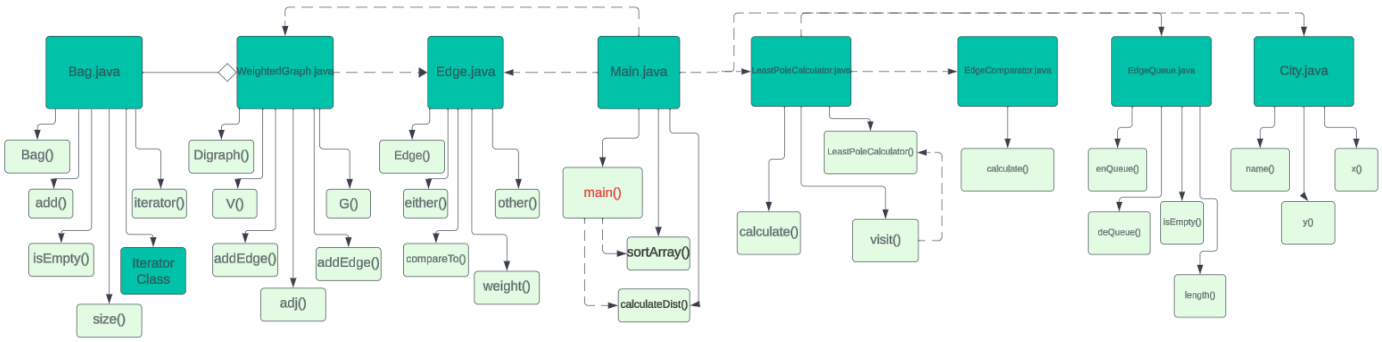

Figure 1 (Structure chart of Task 1)

Task 2:

My second task is about a finding shortest path problem. The cities and their connections are given in an input file, and I was asked to create a undirected edge weighted graph of cities, connecting the cities according to the data in input file by using undirected weighted edges, and find the shortest paths from source city to destination city through the cities user wants to visit.

Structure chart of Task 2 is shown in *Figure 2*.

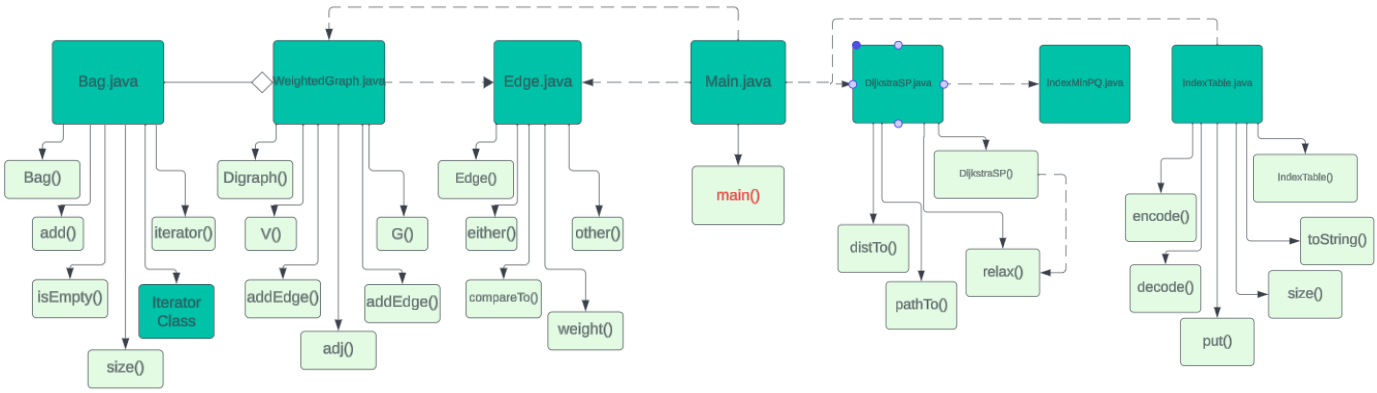

Figure 2 (Structure chart of Task 2)

III) IMPLEMENTATION, FUNCTIONALITY

Task 1:

Weighted graph Class:

```
public class WeightedGraph {
```

I used the Weighted graph class inspiring from classes used in our slides and Sedgewick's book.

Bag Class:

```
public class Bag<Item> implements Iterable<Item> {
```

I used the Bag class inspiring from classes used in our slides and Sedgewick's book.

City Class:

This class is to define the City object, it has name and (x,y) coordinates as attributes, a constructor and getter-setter functions.

```
public City(char name, int x, int y) {
```

- This is the constructor that initializes the name of the city and the x and y coordinates.

Edge Class:

```
public class Edge implements Comparable<Edge> {
```

I used the Weighted edge class inspiring from classes used in our slides and Sedgewick's book.

EdgeComparator Class:

This class imlements Comparator so it overrides its compare() method

```
@Override
public int compare(Edge a, Edge b) {
```

- It compares the edges a and b given as parameter returns -1, 1 or 0 according to which edge has smaller weight.

EdgeQueue Class:

This class provides a linked list-based Edge queue inspired from previous course's slides and Sedgewick's book. It contains typical enQueue() and deQueue() methods.

LeatPoleCalculator Class:

This class is to calculate the minimum spanning tree in a given edge weighted graph by using Prim's algorithm inspired from the Sedgewick's book and slides.

```
public LeastPoleCalculator(WeightedGraph G) {
```

- It's the constructor that initializes the variables and calls the primMST() function to calculate MST.

```
public void primMST() {
```

- It's the function to calculate MST by using Prim's algorithm. It calls visit() function for unvisited vertices and adds their edges to MST queue.

```
private void visit(WeightedGraph G, int v){
```

- This function visits the given vertex v and mark it as visited. For the unvisited adjacent vertices of v, it adds the edge through that vertex to the priority queue.

```
public EdgeQueue calculate(){
```

- This function is the function called by the main function and returns the calculated MST queue.

Main Class:

```
public static void main(String[] args) throws Exception {
```

- This is the main function that runs the program. It reads input from the user and creates a weighted graph according to the input, adds edges to the graph and call the calculate() function of LeastPoleCalculator object created in the main class. At the end, it prints the output in the same form of the example output.

```
private static double calculateDist(City a, City b) {
```

- This method calculates the difference between two given cities in the parameter. It puts their coordinates to the Euclidian Distance formula and returns the calculated double value to be used as the weight of the edge between the cities a and b.

```
private static void sortEdges(Edge [] edges) {
```

- This method sorts the given Edge array as parameter in ascending order according to their weights.

Task 2:

Weighted graph Class:

```
public class WeightedGraph {
```

I used the Weighted graph class inspiring from classes used in our slides and Sedgewick's book.

Bag Class:

```
public class Bag<Item> implements Iterable<Item> {
```

I used the Bag class inspiring from classes used in our slides and Sedgewick's book.

Edge Class:

```
public class Edge implements Comparable<Edge> {
```

I used the Weighted edge class inspiring from classes used in our slides and Sedgewick's book.

IndexTable Class:

```
public IndexTable() {
```

- Constructor that initializes the table.

```
public void put(String s) {
```

- Puts the String s to the table if it's not already in the table.

```
public int encode(String s) {
```

- Returns the index of the string s from the list.

```
public String decode(int i) {
```

- Returns the String at given index

```
public int size() {
```

- Gives the size of the table which is private.

IndexMinPQ Class:

```
public class IndexMinPQ<Key extends Comparable<Key>> implements Iterable<Integer> {
```

- This class is a minimum priority queue inspired from the Sedgewick's book.

DijkstraSP Class:

This class is to calculate the shortest paths through all vertices from a source city in a given edge weighted graph inspired from the Sedgewick's book and slides.

```
public DijkstraSP(WeightedGraph G, int s){
```

- This is the constructor Computes a shortest-paths tree from the source vertex s to every other vertex in the edge-weighted graph. It initializes the priority queue, edgeTo and distTo arrays, for the each vertex in priority queue, calls relax function

```
private void relax(Edge e, int current){
```

- It relaxes edge e and updates edgeTo and distTo arrays if shortest path is changed.

```
public int distTo(int v) {
```

- It returns the length of a shortest path from the source vertex to vertex v.

```
public Edge[] pathTo(int v) {
```

- Returns a shortest path from the source vertex to vertex v.

Main Class:

```
public static void main(String[] args) {
```

- This is the main function that runs the program. It reads input from the user and creates a weighted graph according to the input, adds edges to the graph according to the connections in the input file. It creates DijkstraSP object and call its distTo() and pathTo() functions. At the end, it prints the output of calculated path and path length in the same form of the example output.

IV) FINAL ASSESSMENTS

As conclusion, the assignment has some parts that challenged me. Understanding how the shortest path algorithms work and implementing the weighted graph structure dynamically according to the user input was the two topics I had trouble with. Especially, getting the true outputs by the shortest-path and minimum spanning tree algorithms to find desired paths was very challenging parts for me. The most difficult part was adapting the Dijkstra's shortest path algorithm, which is taught us by using directed weighted edges, to weighted undirected edges. But at the and of the day, this assignment really helped me about gaining a big understanding of shortest-path algorithms and minimum spanning tree algorithms.