DOKUZ EYLUL UNIVERSITY
ENGINEERING FACULTY
DEPARTMENT OF COMPUTER ENGINEERING

CME2201 DATA STRUCTURES ASSIGNMENT REPORT

JOURNEY PLANNER

by
Rıdvan Özdemir
2017510086

Lecturers
Dr. Zerrin Işık
Ali Cüvitoğlu
Feriştah Dalkılıç
Altuğ Yiğit

IZMIR

15.11.2019

CHAPTER ONE

PROGRESS DESCRIPTION

The aim of the project is to develop a journey planner program implementation with graph and Dijkstra Algorithm. This program read some txt files and take the stop and transportation line informations from them. After the spliting the informations and eliminating, informations enter the Graph as a vertex or edge. And then, find the shortest path and lines by using Dijkstra algorithm

CHAPTER TWO

TASK SUMMARY

2.1. Completed Tasks

All necessary txt files were read.

All stops and its informations were put into Graph as a Vertex.

All edges were created.

All lines were read and put into a data structure to reach them when it is necessary.

Trip txt were read and missing edges were added as distance is 625m.

Dijkstra Algorithm were implemented with necessary changes.

According to test_stop, all alternative lines were found. And each alternative were calculated. The 5-shortest path were wrote on screen.

2.1. Incomplete Tasks

There is no incompleted task.

2.1. Additional Improvements

A keyboard command were added to make easier and clear to see all alternative lines.

CHAPTER THREE

EXPLANATION OF ALGORITHMS

3.1. Algorith and Solution Strategies

First of all I read Stop.txt and added the vertices into graph, and keep the all neighbour stops in a list. Becasue when I tried to put them with Stop.txt, there was a "NullPointerException" error because neighbour stops have not been added yet. After the read Stop.txt, I added neigbour stops and then read the distance.txt and added the edges. After this, I read Line.txt and added all lines into HashMap according to their lineID and kept the necessary informations for trip in a data structure. After all this, firstly I tried to implemented Dijkstra Algorithm, but it was not run correctly. That's why first, I found the all alternative path for test_stops.txt line. To solve this issue, I implemented findShortestPath() method. This method according to source and target lines, found all alternative paths between them. Then calculate the distance according to criteria and put the 5-shortest alternative path in a array. Then thesee 5-shortest path is wrote on screen as lines.

CHAPTER FOUR

PROBLEMS ENCOUNTERED

I encountered "NullPointerException" error when I tried to add neighbour stops before add all vertices. To solve this, I kept the all neighbour stops in a list and added them after add all vertices.

The second problem I encountered about Dijkstra algorithm. I encountered same shortest path results and distances after first time using Dijkstra because I did not re-initiate distance,previous and stopCounts of vertices.

CHAPTER FIVE

SCREENSHOTS

```
public void findShortestPath(Graph g1, HashMap<Integer,Line> lineList, List<Integer> tripHelper, int source, int desti
    int transferCount=0;
    String[][] transferStops = new String[2000][3];
    String[] tempSourceLines = new String[50];
    String[] tempTargetLines = new String[50];
    int sourceID=source;
    int targetID=destination;
    for (int i = 0; i < g1.getVertex(sourceID).getEdges().size(); i++) {
        if(g1.getVertex(sourceID).getEdges().get(i).isNeighbourEdge()) {
            String[] tempNeighbourLines = new String[30];
            g1.getVertex(sourceID).getEdges().get(i).getDestination().getLines().keySet().toArray(tempNeighbourLines);
            for (int j = 0; j < tempNeighbourLines.length; j++) {
                if(tempNeighbourLines[j]==null) {
                    break;
                }
                if(!g1.getVertex(sourceID).getLines().containsKey(tempNeighbourLines[j])) {
                    g1.getVertex(sourceID).getLines().put(tempNeighbourLines[j]+";"+g1.getVertex(sourceID).getEdges().
                }
            }
        }
    }
    for (int i = 0; i < g1.getVertex(targetID).getEdges().size(); i++) {
        if(g1.getVertex(targetID).getEdges().get(i).isNeighbourEdge()) {
            String[] tempNeighbourLines = new String[30];
            g1.getVertex(targetID).getEdges().get(i).getDestination().getLines().keySet().toArray(tempNeighbourLines);
            for (int j = 0; j < tempNeighbourLines.length; j++) {
                if(tempNeighbourLines[j]==null) {
                    break;
                }
                if(!g1.getVertex(targetID).getLines().containsKey(tempNeighbourLines[j])) {
                    g1.getVertex(targetID).getLines().put(tempNeighbourLines[j]+";"+g1.getVertex(targetID).getEdges().
                }
            }
        }
```

Figure.1 parts of findShortestPath method

```
public void readStop(Graph g1) throws NumberFormatException, IOException {
    FileReader fileReader = new FileReader("Stop.txt");
    String line;

    BufferedReader br = new BufferedReader(fileReader);
    int stopID=0;
    String stopName=" ";
    double coordinateX=0.0;
    double coordinateY=0.0;
    int vehicleType=0;
    int k=0;
    ArrayList<String> tempEdges = new ArrayList<String>();
    ArrayList<Integer> tempVertices = new ArrayList<Integer>();
    while ((line = br.readLine()) != null) {
        if(k==0) {
            k++;
            continue;
        }
       String[] splitted = line.split(";");


        stopID=Integer.parseInt(splitted[0]);

        stopName=splitted[1].toString();

        coordinateX=(double)Float.parseFloat(splitted[2].replace(",", "."));

        coordinateY=(double)Float.parseFloat(splitted[3].replace(",", "."));

        vehicleType=Integer.parseInt(splitted[4]);
```

Figure.2 reading Stop.txt and add vertices

```java
public void readDistance(Graph g1) throws NumberFormatException, IOException {
    FileReader fileReader = new FileReader("Distance.txt");
    BufferedReader br = new BufferedReader(fileReader);
      String line=null;
      int k=0;
       int sourceID;
       int destinationID;
       int distance;
       while ((line = br.readLine()) != null) {
            if(k==0) {
                k++;
                continue;
            }

        String[] splitted1 = line.split(";");
        sourceID = Integer.parseInt(splitted1[0]);
        destinationID = Integer.parseInt(splitted1[1]);
        distance = Integer.parseInt(splitted1[2]);
        if(g1.getVertex(sourceID)==null) {
            Vertex v = new Vertex(sourceID);
            g1.addVertex(v);
        }
        if(g1.getVertex(destinationID)==null) {
            Vertex v = new Vertex(destinationID);
            g1.addVertex(v);
        }
        if(g1.getEdge(sourceID, destinationID)==null) {
            g1.addEgde(g1.getVertex(sourceID), g1.getVertex(destinationID), distance,false
        }
```

Figure.3 *reading Stop.txt and add vertices*

```java
public void clearGraphDistances(Graph g1) {
    for(Vertex v: g1.vertices()) {
        v.setDistance(Integer.MAX_VALUE);
        v.setPrevious(null);
        v.setVisited(false);
        v.setStopCountForReach(0);
    }
}
```

Figure.4 *re-initiate of necessary variables*

```java
public int calculateDist(Graph g1,int sourceID, boolean isWalkFromSource, int walkSourceID, String line1, int lineTransferID, bo
    clearGraphDistances(g1);
    g1.getVertex(target).setVisited(false);
    g1.getVertex(sourceID).setDistance(0);
    g1.getVertex(sourceID).setStopCountForReach(0);
    PriorityQueue<Vertex> pq = new PriorityQueue<>();
    pq.add(g1.getVertex(sourceID));
    g1.getVertex(sourceID).setVisited(true);
    int tempOrder=0;
    int stopCount=0;
    boolean isLine1=true;
    boolean isLine2=false;
    while(!pq.isEmpty()) {
        Vertex currVertex=pq.poll();
        if(currVertex.getStopID()==target &&!isWalkToTarget) {
            break;
        }
            for(Edge edge: currVertex.getEdges()) {
                Vertex v=edge.getDestination();
                if(currVertex.getStopID()==sourceID && isWalkFromSource && !v.isVisited() && v.getStopID()==walkSourceID) {
                    int newDistance =  ((int)currVertex.getDistance()) + edge.getDistance();
                    if(newDistance < v.getDistance()) {
                        pq.remove(v);
                        v.setDistance(newDistance);
                        v.setPrevious(currVertex);
                        pq.add(v);
                        stopCount++;
                        v.setStopCountForReach(stopCount);
                        isWalkFromSource=false;
                    }
```

Figure.5 *some part of Dijkstra Algorithm*

CONCLUSİON

I completed all expected task of this assignment. I've learned implementing Graph and Dijkstra Algorithm. This is the first time, I worked on a real daily problem. That's why it was kindly hard for me. After this project I realized, Shortest path problems is not too easy and working on real problem is much more attractive.

REFERENCES

HTTPS://WWW.GEEKSFORGEEKS.ORG/DİJKSTRAS-SHORTEST-PATH-ALGORİTHM-GREEDY-ALGO-7/

HTTPS://WWW.GEEKSFORGEEKS.ORG/GRAPH-DATA-STRUCTURE-AND-ALGORİTHMS/