VIETNAM GENERAL CONFEDERATION OF LABOUR

TON DUC THANG UNIVERSITY

INFORMATION TECHNOLOGY FACULTY

ESSAY DATA STRUCTURE AND ALGORITHMS II

FIND THE MINIMUM PATH

Instructing Lecturer: **Mr. NGUYEN QUOC BINH**

Student's name: **HUỲNH LÊ THIÊN Ý – 518H0320**

Class **:** **18H50204**

Course **:** **22**

HO CHI MINH CITY, YEAR 2020

VIETNAM GENERAL CONFEDERATION OF LABOUR
TON DUC THANG UNIVERSITY
INFORMATION TECHNOLOGY FACULTY

COMBINATORICS AND GRAPHS

FIND THE MINIMUM PATH

Instructing Lecturer: **Mr. NGUYEN QUOC BINH**
Student's name: **HUỲNH LÊ THIÊN Ý – 518H0320**
Class **:** **18H50204**
Course **:** **22**

HO CHI MINH CITY, YEAR 2020

ACKNOWLEDGEMENT

In order to make this report complete and achieve good results, we have received the support and assistance of many teachers and classmates. With deep affection, sincerity, we express deep gratitude to all individuals and agencies who have helped us in our study and research. First of all, I would like to express a special appreciation to Ton Duc Thang.

University's teachers for their conscientious guidance and advices throughout the last semester by gave me their modern outlook and meticulous supervision to carry out the job perfectlyWe would like to express our sincere gratitude to the leadership of Ton Duc Thang University for supporting, helping and facilitating us to complete the report well during the study period. With limited time and experience, this report cannot avoid mistakes. We are looking forward to receiving advice and comments from teachers so that we can improve our awareness, better serve the practical work later. We sincerely thank you!

THE PROJECT WAS COMPLETED
AT TON DUC THANG UNVERSITY

The content of research, results in this subject is honest and not published in any form before. The data in the tables used for the analysis, comment, and evaluation were collected by the authors themselves from various sources indicated in the reference section. In addition, many comments and assessments as well as data from other authors and organizations have been used in the project, with references and annotations. If any fraud is found, I am fully responsible for the content of my project. Ton Duc Thang University is not involved in any copyright infringement or copyright infringement in the course of implementation (if any).

Ho Chi Minh City, April 26th 2020

Author

Signed

Huỳnh Lê Thiên Ý

EVALUATION OF INSTRUCTING LECTURER

Confirmation of the instructor

_____

_____

_____

_____

_____

_____

_____

_____

_____

Ho Chi Minh City, month day 2020

(Sign and write full name)

The assessment of the teacher marked

_____

_____

_____

_____

_____

_____

Ho Chi Minh City, month day 2020

(Sign and write full name)

TABLE OF CONTENTS

LIST OF TABLES, DRAWINGS, GRAPHICS

INTRODUCTION

Research issues: Using Java language to write a GUI application that allows users to find the minimum path from a location to another.

Approaches:

- Gather information to create data
- Learn about Java GUI to create map interface
- Learn about the shortest path finding algorithm

Resolving research issues:

- Gathering information creates data
- Show the map to the user
- Displays dialogs for users to enter source and destination
- Displays a window showing location suggestions
- Use the algorithm to find the shortest path
- Show a window showing the shortest path in words

DATA

I use my map which is supported by google map to collect information to create data.

The information collected includes: *longitude*, *latitude*, *place name*.

The information I gathered included 103 locations and I manually generated 169 paths from those locations.

This is part of the information I gathered.

```
0 10.77309 106.68994 Ngân_hàng_TMCP_Kỹ_thương_Việt_Nam_(Techcombank)_95_Cách_Mạng_Tháng_Tám_Phường_Phạm_Ngũ_Lão_Quận_1
1 10.77289 106.69049 ABC_Bakery_91-91B_Cách_Mạng_Tháng_Tám_Phường_Phạm_Ngũ_Lão_Quận_1
2 10.77262 106.69082 Trung_Nguyên_Legend_Café_02_Bùi_Thị_Xuân_Phường_Phạm_Ngũ_Lão_Quận_1
3 10.77214 106.69152 Khách_Sạn_Rạng_Đông_81_Cách_Mạng_Tháng_Tám_Phường_Phạm_Ngũ_Lão_Quận_1
4 10.77198 106.69205 DON_CHICKEN_Bến_Thành67-69_Cách_Mạng_Tháng_Tám_Phường_Phạm_Ngũ_Lão_Quận_1
5 10.77241 106.69302 Homestead_Parkview_149_Nguyễn_Du_Phường_Bến_Thành_Quận_1
6 10.77299 106.69377 Trường_THCS_Nguyễn_Du139_Nguyễn_Du_Phường_Bến_Thành_Quận_1
7 10.77333 106.69443 Tạp_chí_Thế_Giới_Vi_Tính_-_PC_World_VN_79_Trương_Định_Phường_Bến_Thành_Quận_1
8 10.77265 106.69165 Liên_Đoàn_Lao_Động_Tp._Hồ_Chí_Minh_14_Cách_Mạng_Tháng_Tám_Phường_Bến_Thành_Quận_1
9 10.77394 106.6893 Phong_Vũ_264_Nguyễn_Thị_Minh_Khai_Phường_6_Quận_3
10 10.77187 106.69001 EdenStar_Saigon_Hotel_38_Bùi_Thị_Xuân_Phường_Phạm_Ngũ_Lão_Quận_1
11 10.77119 106.68953 Ambassador_Saigon_Hotel_84A_Bùi_Thị_Xuân_Phường_Phạm_Ngũ_Lão_Quận_1
12 10.77152 106.69239 Bánh_mì_Huỳnh_Hoa_26_Lê_Thị_Riêng_P.BT_Quận_1
13 10.77202 106.69274 Cửa_Hàng_Điện_Thoại_Di_Động_Hoàng_Phát_19_Cách_Mạng_Tháng_Tám_Phường_Bến_Thành_Quận_1
14 10.77263 106.69287 Cello_Coffee_116_Nguyễn_Du_Phường_Bến_Thành_Quận_1
15 10.77331 106.69364 Clb_Văn_Hóa_Tdtt_Nguyễn_Du_116_Nguyễn_Du_Phường_Bến_Thành_Quận_1
16 10.77244 106.69419 Quy_Thanh_Hote_110_Đặng_Trần_Côn_Phường_Bến_Thành_Quận_1
17 10.77205 106.69427 Shimmer_Silver_238_Lý_Tự_Trọng_Phường_Bến_Thành_Quận_1
18 10.77147 106.69342 Italiani's_Pizza_290_Ly_Tu_Trong_Phuong_Ben_Thanh_Quan_1
19 10.7712 106.69046 NHÀ_HÀNG_COM_XÍU_106A_Lê_Thị_Riêng_Phường_Phạm_Ngũ_Lão_Quận_1
20 10.77155 106.69091 Quán_nướng_Giấy_Bạc_96/1_Lê_Thị_Riêng_Phường_Phạm_Ngũ_Lão_Quận_1
21 10.77265 106.69544 Trung_Nguyên_Legend_Café_219_Lý_Tự_Trọng_Phường_Bến_Thành_Quận_1
22 10.77245 106.69478 Nhà_Hàng_Phương_Cua_228_Lý_Tự_Trọng,_Phường_Bến_Thành,_Quận_1
23 10.77289 106.69544 Manmaru_2_Japanese_Restaurant_192A_Lý_Tự_Trọng_Phường_Bến_Thành_Quận_1
```

Figure 1. *Example for location information display in **location.txt** file*

```
1 10
1 2
1 3
1 4
1 5
100 102
101 96
11 12
13 38
15 16
17 18
17 23
18 23
18 35
19 37
19 38
19 93
20 26
20 27
20 97
21 20
21 97
```

Figure 2. *Example for location information display in **path.txt** file*

The file contains a location named: **location.txt**

The file contains the path named: **path.txt**

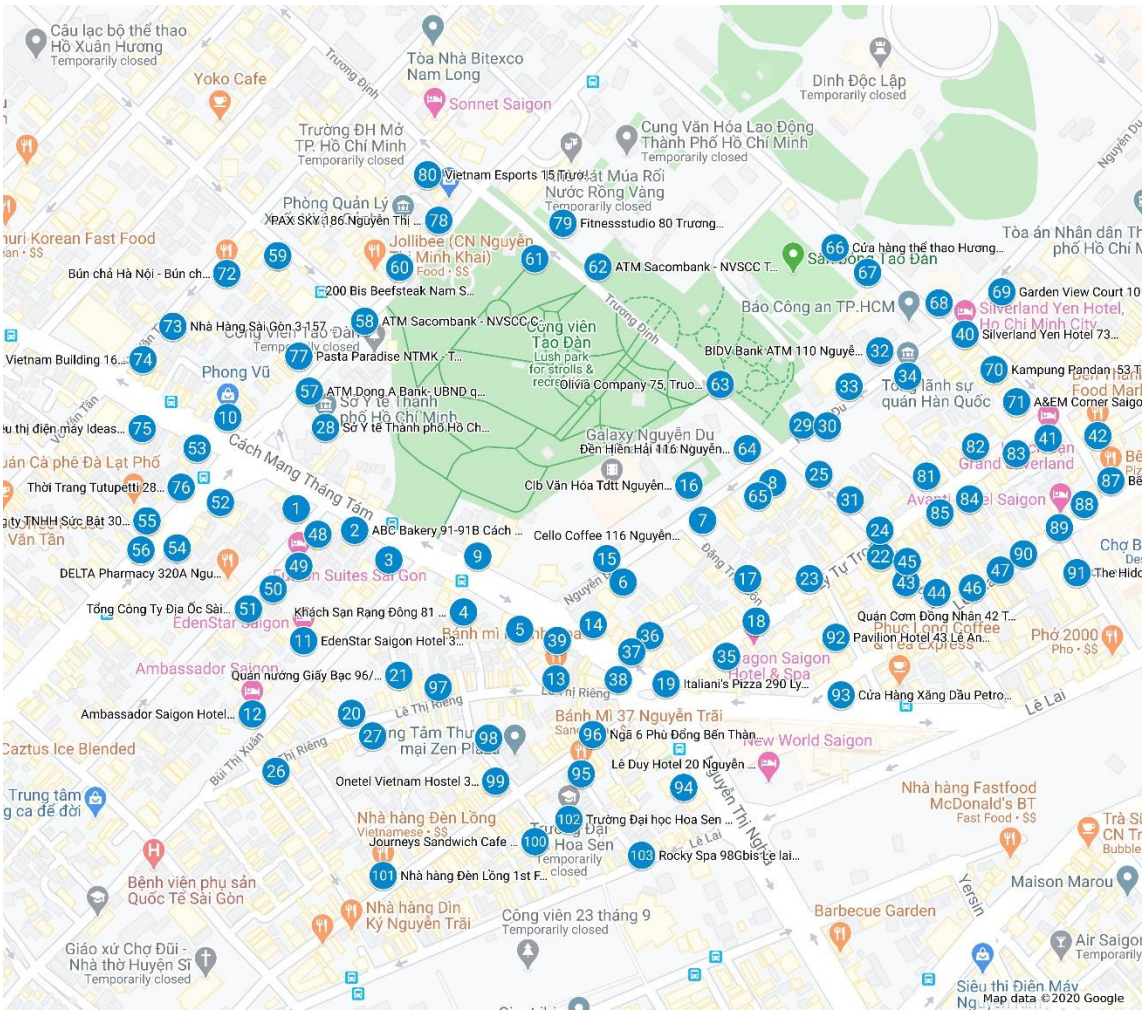


Figure 3. *Map display location and some of their name*

After manually collecting locations and paths, I used the GenerateAL.java program to convert:

- 103 places into data stored in variables named datas and list_name.
- 169 paths into 338 paths stored in a variable named edges (because I used a undirected graph).

This is the GenerateAL.java program I used.

```java
import java.util.*;
import java.io.*;
import javax.swing.*;

class Data
{
    int id;
    double lat, lon;
    public Data(int id, double lat, double lon)
    {
        this.id = id;
        this.lat = lat;
        this.lon = lon;
    }
}

class Reachable
{
    public int source;
    public int dest;

    public Reachable(int source, int dest)
    {
        this.source = source;
        this.dest = dest;
    }
}
```

Figure 4. *Code of* ***GenerateAL.java***

```java
class GenerateAL {

    public static Map<String,Integer> reference_name = new HashMap<>();
    public static ArrayList<String> list_name = new ArrayList<String>();

    // Calculate distance between 2 locations
    public static int measure (double lat1, double lon1,double lat2, double lon2)
    {
        lon1 = Math.toRadians(lon1);
        lon2 = Math.toRadians(lon2);
        lat1 = Math.toRadians(lat1);
        lat2 = Math.toRadians(lat2);

        // Haversine formula
        double dlon = lon2 - lon1;
        double dlat = lat2 - lat1;
        double a = Math.pow(Math.sin(dlat / 2),2)
                 + Math.cos(lat1) * Math.cos(lat2)
                 * Math.pow(Math.sin(dlon / 2),2);

        double c = 2 * Math.asin(Math.sqrt(a));

        double r = 6371;
        int result = (int) Math.round ((c * r)*100);
        // calculate the result
        return result;
    }
```

Figure 5. *Code of* ***GenerateAL.java*** *measure the distance between to location using their latitude and longitude*

```
public static void make_AL_to_file (List<Data> datas,List<Reachable> edges)
{
    try
    {
        FileWriter myWriter = new FileWriter("AL.txt");
        int size = edges.size();
        int s=0,d=0,i=1;
        myWriter.write(size + "\r\n");
        while (i<size){
            s = edges.get(i).source - 1;
            d = edges.get(i).dest - 1;
            myWriter.write(s + " " + d + " " + measure(datas.get(s).lat,datas.get(s).lon,
                                                       datas.get(d).lat,datas.get(d).lon) + "\r\n");
            i++;
            }

        myWriter.close();
    } catch (IOException e)
    {
        JOptionPane.showMessageDialog(null,"An error occurred.");
        e.printStackTrace();
    }
}
```

Figure 6. *Code of **GenerateAL.java** make a list includes source, destination, distance using path.txt and stored in **AL.txt***

```
public static void make_list_name (List<String> names)
{
    try
    {
        FileWriter myWriter = new FileWriter("names.txt");
        int n =0;
        for (int i = 0;i<names.size();i++)
        {
            n=i+1;
            String s = names.get(i);
            myWriter.write(n + s +"\r\n");
            reference_name.put(s,i);
        }
        myWriter.close();
    } catch (IOException e)
    {
        System.out.println();
        JOptionPane.showMessageDialog(null,"An error occurred.");
        e.printStackTrace();
    }
}
```

Figure 7.*Code of **GenerateAL.java** make list of names from information, stored in **names.txt***

```
private static void err(int code)
{
    System.out.println("Error code:" +code);
    System.exit(code);
}
public static void Generate_Data ()
{
    int id = 0,s=0,d=0;
    double lat = 0 ,lon = 0;
    ArrayList<Data> datas = new ArrayList<Data>();
    ArrayList<Reachable> edges = new ArrayList<Reachable>();
    try{
        Scanner sc = new Scanner(new File("location.txt"), "UTF-8");

        while (sc.hasNext()) {
            if (sc.hasNextInt())
                id = sc.nextInt();
            else {System.out.println(id);err(2);}
            if (sc.hasNextDouble())
                lat = sc.nextDouble();
            else {System.out.println(lat);err(3);}
            if (sc.hasNextDouble())
                lon = sc.nextDouble();
            else {System.out.println(lon);err(4);}
            if (sc.hasNext())
                list_name.add(new String(sc.nextLine()));
            else {System.out.println(lon);err(5);}


            datas.add(new Data(id,lat,lon));
        }
```

Figure 8. *Read file* ***location.txt***

```
        Scanner sc1 = new Scanner(new File("path.txt"));

        while (sc1.hasNext()) {
            if (sc1.hasNextInt())
                s = sc1.nextInt();
            else {System.out.println(s);err(6);}
            if (sc1.hasNextInt())
                d = sc1.nextInt();
            else {System.out.println(d);err(7);}

            edges.add(new Reachable(s,d));
            edges.add(new Reachable(d,s));
        }

    }catch(Exception e){System.out.println(e);}

    make_AL_to_file (datas, edges);
    make_list_name (list_name);
}
```

Figure 9. *Read file* ***path.txt***

The data contained in the variable named **edges** and **datas** is used to create data that contains the path and distance of the locations, they are stored in a file called **AL.txt** because I use the adjacency listed graph structure.

The data contained in the variable name **list_name** printed from this program consists of ordinal numbers and location names stored in the **names.txt** file.

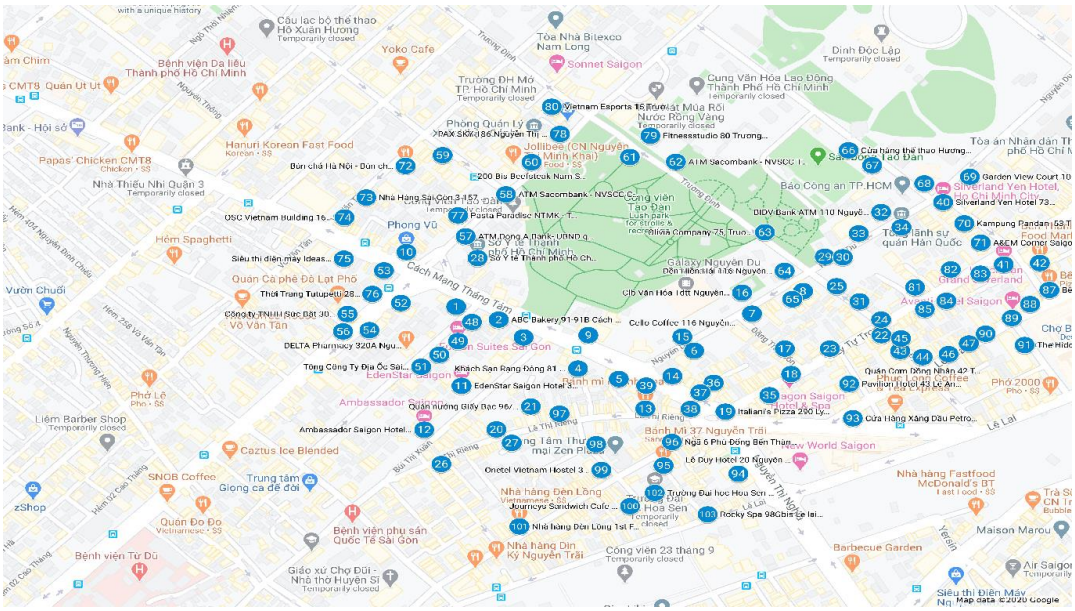
INTERFACE

To create the user interface, I use the Swing library built at the top of the Abstract Window Toolkit API. First, I created the map using the program ***MyCanvas.java***.

```
import java.awt.*;
import javax.swing.*;

public class MyCanvas extends JComponent{

    Image img;

    public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D) g;

        img = new ImageIcon(getClass().getResource("mapimage.jpg")).getImage();
        g2.drawImage(img,0,0, getWidth(), getHeight(), null);
    }
}
```

Figure 10. *Code of **MyCanvas.java***

Due to my lack of understanding of how to implement the interface, I could only display the map with images, not the zoom and scroll function. This is an image showing what the map will look like when running the program.



After showing the map, the program will run a dialog box to ask users about where they started moving and where to go.

When you first enter the route information, the ***FindandSuggest.java*** program will run to check whether the data exists or not.

- If exists:
    - The case is completely correct: the program will switch to a dialog asking about the destination.
    - the case is correct but not enough: the program will suggest the exact location in a Suggest window and then display a dialog box for the user to re-enter the location with the request "Enter the correct address with a space in front of the address: "
- If none exists or false information
    - In case of wrong entry or location not found in the collected data, the program will display a dialog asking to re-enter with the message "Can't find this address. Enter another address:"

After the program confirms the starting point is correct and exists that data, it will display a dialog asking to enter the destination will come. When you enter the above errors, the program will repeat the above steps until the information entered is correct.

This is the ***FindandSuggest.java*** program I used.

```
import java.util.*;
import javax.swing.*;
import java.awt.*;

class FindandSuggest {

    GenerateAL a = new GenerateAL();
    public static ArrayList<String> suggest = new ArrayList<String>();
    public static JFrame fl = new JFrame("Suggest");

    public static int Find (String input_name, ArrayList<String> list_name){
        int flag_F = 1;
        for (int i=0; i<list_name.size();i++){
            if ((input_name.equals(list_name.get(i)) == true))
                flag_F = 0;
        }
        return flag_F;
    }

    public static int Suggest(String input_name, ArrayList<String> list_name){
        int flag_S = 1;
        System.out.println();
        JOptionPane.showMessageDialog(null, "Did you mean one of these addresses: ");

        for (int i=0; i<list_name.size();i++){
            if ((list_name.get(i).contains(input_name)) == true){
                suggest.add(list_name.get(i));
                flag_S = 0;
            }
        }
        display_Suggest();

        return flag_S;
    }

    private static void display_Suggest() {
        fl.add(new JList(suggest.toArray()));
        fl.setSize(400, 400);
        fl.setBounds(100, 30, 600, 300);
        fl.pack();
        fl.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //fl.setLocationRelativeTo(null);
        fl.setVisible(true);

    }
```

Figure 11. *Code of* ***FindandSuggest.java***

This is the image to show the explanation above:

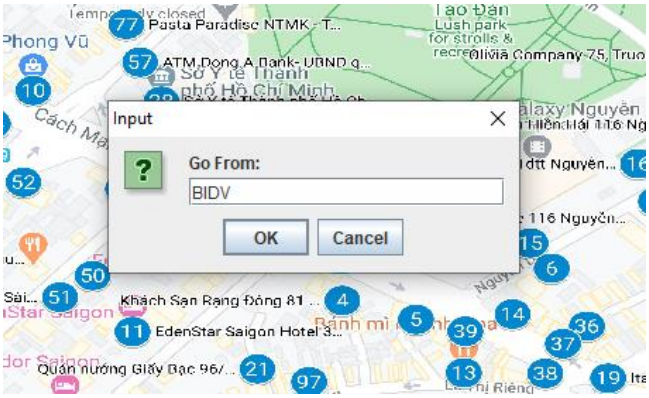


Figure 12. *Dialog ask for the start location appear*

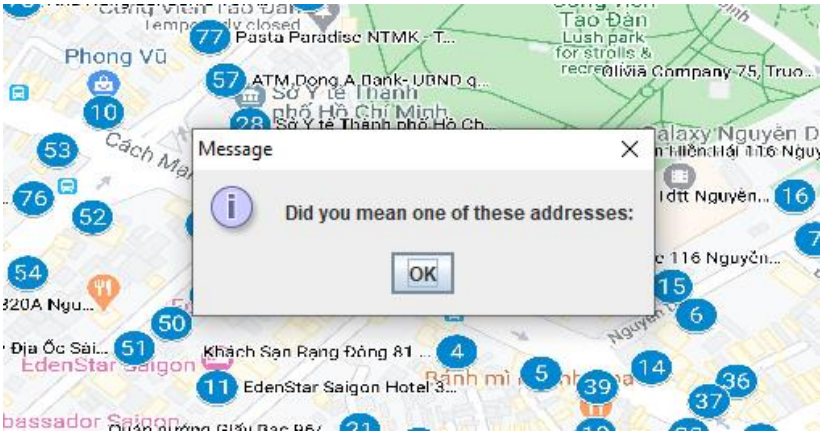


Figure 13. *Dialog announce before give suggest addresses*

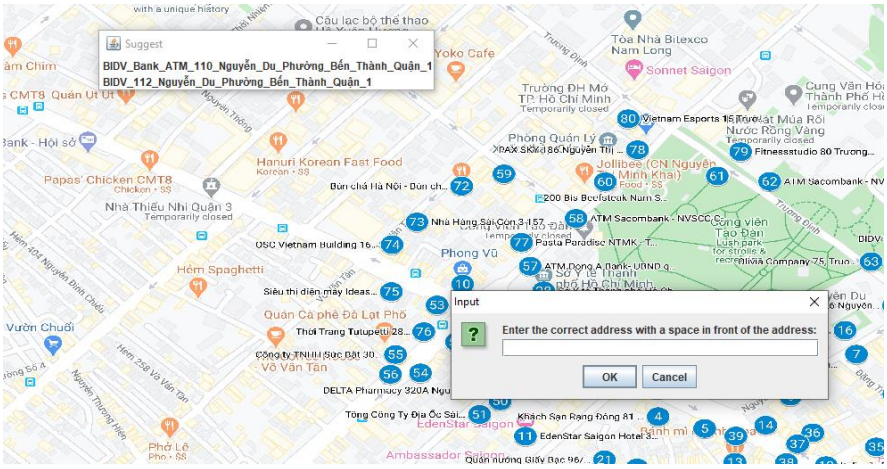


Figure 14. *Suggest window appear with some correct addreses*

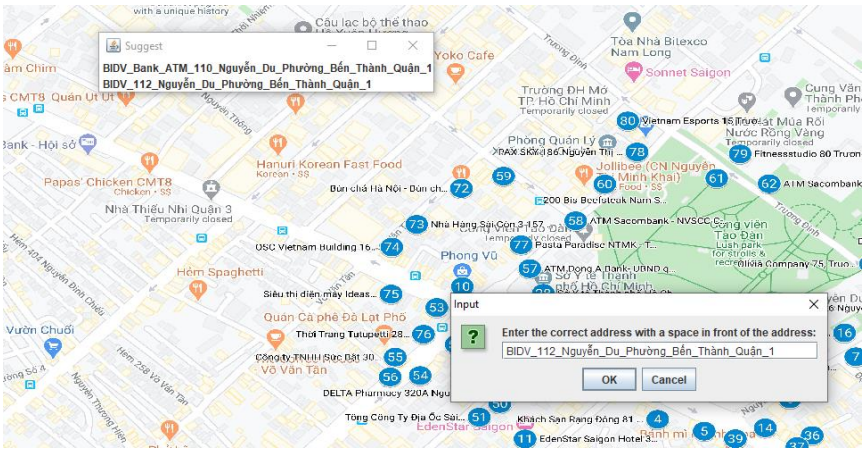


Figure 15. *Enter correct address again*

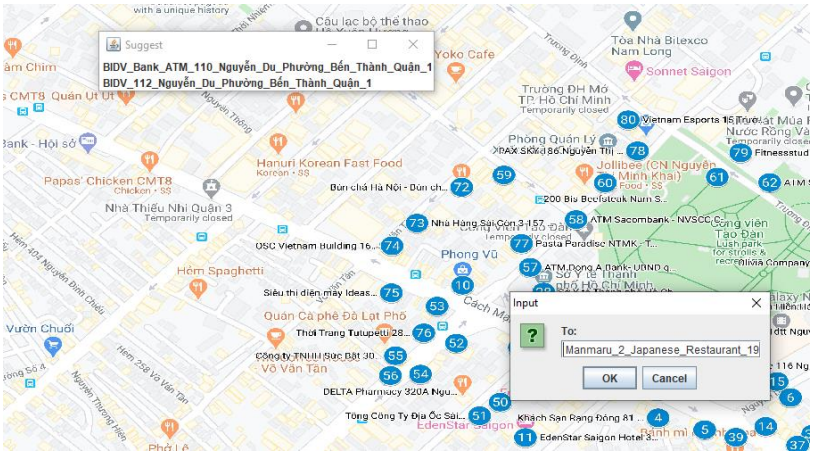


Figure 16. *Another dialog ask for destination location*

After having enough data, the program will print the shortest path. Pictures for this result will be shown in the following sections.

FIND THE SHORTEST PATH

I use the Dijkstra's algorithm to find the shortest path.

Here is the pseudo code for this algorithm from Wikipedia

```
 1  function Dijkstra(Graph, source):
 2
 3      create vertex set Q
 4
 5      for each vertex v in Graph:
 6          dist[v] ← INFINITY
 7          prev[v] ← UNDEFINED
 8          add v to Q
 9      dist[source] ← 0
10
11      while Q is not empty:
12          u ← vertex in Q with min dist[u]
13
14          remove u from Q
15
16          for each neighbor v of u:           // only v that are still in Q
17              alt ← dist[u] + length(u, v)
18              if alt < dist[v]:
19                  dist[v] ← alt
20                  prev[v] ← u
21
22      return dist[], prev[]
23
```

```
1  S ← empty sequence
2  u ← target
3  if prev[u] is defined or u = source:
// Do something only if the vertex is reachable
4      while u is defined:
// Construct the shortest path with a stack S
5          insert u at the beginning of S
// Push the vertex onto the stack
6          u ← prev[u]
// Traverse from target to source
```

Figure 17. *Dijkstra's algorithm pseudo code*

This is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks.

This algorithm is predicated the principle of relaxtion, during which an approximation to the proper distance is gradually replaced by more accurate values until shortest distance is reached. The approximate distance to every vertex is usually an overestimate of the verify distance, and is replaced by the minimum of its old value with the length of a newly found path. It uses a priority queue to greedily select the closest vertex that has not yet been processed and performs this relaxation process on all of its outgoing edges.

This is the ***Dijkstra.java*** program I used.

```
import java.util.*;
import java.io.*;
import javax.swing.*;

public class Dijkstra
{
    public static ArrayList<Integer> arr = new ArrayList<Integer>();
    public static ArrayList<String> path = new ArrayList<String>();

    public static void printRoute()
    {
        GenerateAL a = new GenerateAL();
        for (int i = 0; i< arr.size();i++){
            for ( Map.Entry<String,Integer> entry : a.reference_name.entrySet()) {
                if (entry.getValue() == arr.get(i)){
                    path.add("Go to:");
                    path.add(entry.getKey());
                }
            }
        }
    }

    private static void Route(int prev[], int i)
    {
        if (i <= 0)
            return;

        Route(prev, prev[i]);
        arr.add(i);
    }
```

Figure 18. Code of ***Dijkstra.java*** *print route from start point to end point*

In this code I use the number that represents the address the user enters, so when printing a route, we need to convert it from the number to the address.

```
// Run Dijkstra's algorithm on given graph
public static void shortestPath(Graph graph, int source, int N, int dest)
{
    // create min heap and push source node having distance 0
    PriorityQueue<Node> minHeap = new PriorityQueue<>((lhs, rhs) -> lhs.weight - rhs.weight);

    minHeap.add(new Node(source, 0));

    // set infinite distance from source to v initially
    List<Integer> dist = new ArrayList<>(Collections.nCopies(N, Integer.MAX_VALUE));

    // distance from source to itself is zero
    dist.set(source, 0);

    // boolean array to track vertices for which minimum
    // cost is already found
    boolean[] done = new boolean[N];
    done[0] = true;

    // stores predecessor of a vertex (to print path)
    int prev[] = new int[N];
    prev[0] = -1;
```

Figure 19. *Code of* ***Dijkstra.java*** *represents the Dijkstra's algorithm*

```
// run till minHeap is not empty
while (!minHeap.isEmpty())
{
    // Remove and return best vertex
    Node node = minHeap.poll();

    // get vertex number
    int u = node.vertex;

    // do for each neighbor v of u
    for (Edge edge: graph.adjList.get(u))
    {
        int v = edge.dest;
        int weight = edge.weight;

        // Relaxation step
        if (!done[v] && (dist.get(u) + weight) < dist.get(v))
        {
            dist.set(v, dist.get(u) + weight);
            prev[v] = u;
            minHeap.add(new Node(v, dist.get(v)));
        }
    }

    // marked vertex u as done so it will not get picked up again
    done[u] = true;
}

for (int i = 1; i < N; ++i)
{
    if (i == dest){
        Route(prev, dest);
        printRoute();
    }
```

Figure 20. *Code of **Dijkstra.java** represents the Dijkstra's algorithm and invoked printRoute function*

```
    private static void err(int code) {
      System.exit(code);
    }

    public static void Graph_from_File(int source, int dest)
    {
        int V = 0;
        List<Edge> edges = new ArrayList<Edge>();
        Scanner sc = null; // compiler wants
        int num1 = 0, num2 = 0, len = 0;
        try {
            sc = new Scanner(new File("AL.txt")); // file
        } catch (Exception exception ) { err(1); }
        if (sc.hasNextInt())
            V = sc.nextInt();
        else err(2);
        // get vertices = num of vertices
        while (sc.hasNextInt()) {
            num1 = sc.nextInt();
            if (sc.hasNextInt())
                num2 = sc.nextInt();
            else err(2);
            if (sc.hasNextInt())
                len = sc.nextInt();
            else err(3);

            edges.add(new Edge(num1,num2,len));
            edges.add(new Edge(num2,num1,len));
        }
        Graph graph = new Graph(edges, V);
        sc.close();

        shortestPath(graph, source, V, dest);
    }
```

Figure 21. *Code of **Dijkstra.java** represents the undirected graph creation*

The above program uses the following program named ***Graph.java*** to create a undirected graph using **AL.txt** file which I made before running the algorithm to find the shortest path.

```java
import java.util.*;

// class to represent a graph object
class Node {
    int vertex, weight;

    public Node(int vertex, int weight) {
        this.vertex = vertex;
        this.weight = weight;
    }
}

// Data structure to store graph edges
class Edge {

    int source, dest, weight;

    public Edge(int source, int dest, int weight) {
        this.source = source;
        this.dest = dest;
        this.weight = weight;
    }
}
```

Figure 22. *Code of **Graph.java** support for the **Dijkstra.java** program*

```java
class Graph {
    // A List of Lists to represent an adjacency list
    List<List<Edge>> adjList = null;

    // Constructor
    public Graph(List<Edge> edges, int N) {

        adjList = new ArrayList<>(N);

        for (int i = 0; i < N; i++) {
            adjList.add(i, new ArrayList<>());
        }

        // add edges to the undirected graph
        for (Edge edge: edges) {
            adjList.get(edge.source).add(edge);
        }
    }
}
```

Figure 23. *Code of **Graph.java** support for the **Dijkstra.java** program*

RUNNING RESULT

To run all of the above, all I need to do is use a program called ***Main.java***

This program will perform each of the following steps:

- Run the program ***MyCanvas.java*** to represent the map
- Run the ***GenerateAL.java*** program to create the data file
- Run the ***Dijkstra.java*** program to find the shortest path.
- Inside the program there are also functions to call the ***FindandSuggest.java*** program to check and suggest the path for the user.

I use cmd to run the program. Here is the command to run:

javac Main.java & java Main

The result when you enter full information and the program receives the correct data is shown as the image below:

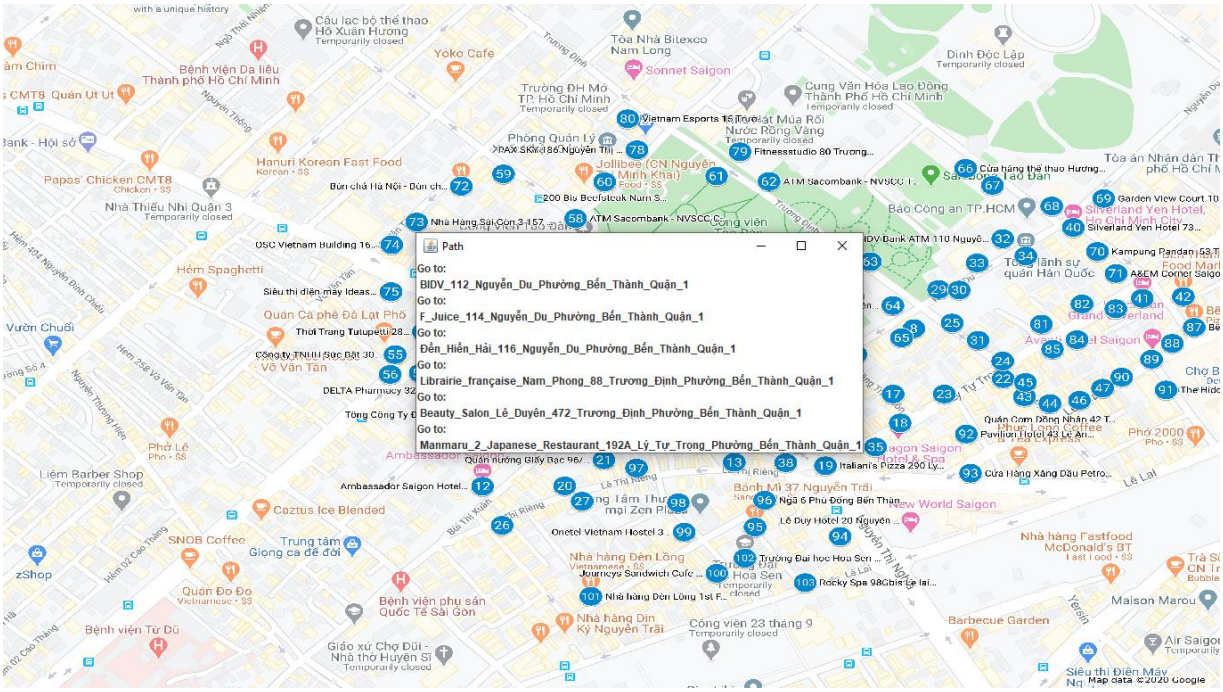


Figure 24. *Result after running* ***Main.java*** *: The shortest path*

DISCUSSIONS

Achievement

After completing this research, I gained more experience:

- In information gathering and data conversion.
- In using Class and Object and using GUI

This will help my work in the future.

Unfinished

- Have not used the GUI well so users can interact more.
- Haven't collected enough information as required.
- The program has redundancy that has not been carefully calculated.

Upgrade in the future

- Create a GUI that shows an interactive map.
- Show the way based on reality when there is a one-way street, roundabout, traffic stalled, ...

REFERENCES

[1] https://www.techiedelight.com/single-source-shortest-paths-dijkstras-algorithm/

[2] http://zetcode.com/java/hashmap/?fbclid=IwAR1bjAP14Mh8JSmoBKHani2IXPk81auyIPYJULQuxLeWfD2o0xdhqOKlVf8

[3] https://www.tutorialspoint.com/java/java_map_interface.htm

[4] https://drive.google.com/open?id=1jGnT4PxEB2AfFISVgYFaYXyD14vbfMZe&usp=sharing

[5] https://freetuts.net/thao-tac-tren-doi-tuong-va-pham-vi-truy-cap-trong-java-1122.html

[6] http://www.cs.utsa.edu/~wagner/CS3343/newgraph/graphrep.html

[7] https://algorithms.tutorialhorizon.com/weighted-graph-implementation-java/

[8] https://www.geeksforgeeks.org/program-distance-two-points-earth/

[9] https://v1study.com/java-swing-cach-tao-hop-thoai-dialog.html

[10] https://congdongjava.com/forum/threads/hi%E1%BB%83n-th%E1%BB%8B-%E1%BA%A3nh-trong-java-swing.16125/

SELF-EVALUATION

Requirement	Score /14	Level 1	Level 2	Level 3	Self-evaluated	Reason
		0 mark	½ mark	Full mark		
1/ Data	**2.0**					
Collect the map data	1.0	V < 100 and E < 500	V < 100 or E < 500	100 ≤ V ≤ 1000 and 500 ≤ E ≤ 10000	0.5	
Data structures	1.0	Not store in a file, and use graph DS.	Not store in a file, or not use graph DS.	Store in a file, use graph DS.	1.0	
2/ Interface	**2.0**					
Show the map.	0.5	Not show.		Show.	0.5	
Highlight.	0.5	Not highlight.		Highlight.	0	
Scroll, zoom.	1.0	Not scroll nor zoom.	Scroll or zoom.	Scroll and zoom.	0	
3/ Min path	**8.0**					
Input source and destination. (1.5 điểm)	2.0	Not input.	From the map or from the combo boxes.	From the map and from the combo boxes.	2.0	
Find and suggest	2.0	Not implement.	Find but cannot suggest.	Find and suggest.	2.0	

incorrect inputs.						
Find min path.	2.0	Not implement.	Find but cannot show.	Find and show.	2.0	
Show min path.	2.0	Not show.	Show incorrectly.	Show correctly	1.0	There are some path show incorrectly
4/ Report	**2.0**					
Report file.	2.0	Not report	Incorrect form.	Complete the report in the correct form.	2.0	
Total	**14.0**			**Result:**	**11**	