TSP.java

1. The Algorithm used for this delivery app project which was to minimize the delay time when delivering pizza and finding shortest path, the BFS (Breadth-first search) is a tree or graph data structure traversal or search algorithm. It begins at the tree root (or an arbitrary node in a graph, known as a 'search key'(1) and examines all the neighbor nodes at the current depth level before moving on to the nodes at the next depth level. It employs the inverse technique of DFS (depth-first search), in which the node branch is explored as far as possible before being forced to backtrack and expand other nodes. (2) Konrad Zuse conceived BFS and its application in finding related components of graphs in his (rejected) Ph.D. thesis on the Plankalkül programming language in 1945, but it was not published until 1972. (3) Edward F. Moore reinvented it in 1959, using it to find the shortest way out of a maze, (4)(5), and C. Y. Lee later transformed it into a wire routing algorithm. We can use the branch-and-bound heuristic to speed up depth first search by not considering paths that can't possibly be better than the best solution found so far. There is no need to continue looking down that road if a partial tour is already longer than the best solution found so far.

• Traverse through one level of children (delivery location in this case) nodes, then traverse through the level of grandchildren (neighboring delivery locations) nodes and so on ….

2. Theirs a single file which contains two classes the first class is Tsp, and the second class is order. The Order class contains all order data members such as the order number, address, and so on. The project's code is included in the Tsp class. The code is structured in an object-oriented manner. For and order, we create order class objects; for example, if there are 40 orders, there will be 40 order class objects. all the objects of the order class are saved in an array list and by using the accordance of these orders we will generate the distance matrix. This is a 2-D array of data type Double, integers and strings etc.

3. My ability to use an understanding of algorithm analysis and data structures to solve problems computationally (Binary trees, Quicksort, Hash tables and so on...). A solid understanding of algorithm analysis and data structures has significantly enhanced my ability to solve problems with effective and efficient programs, such as Big O notations, recursion and linked list which I have used for this project. These concepts

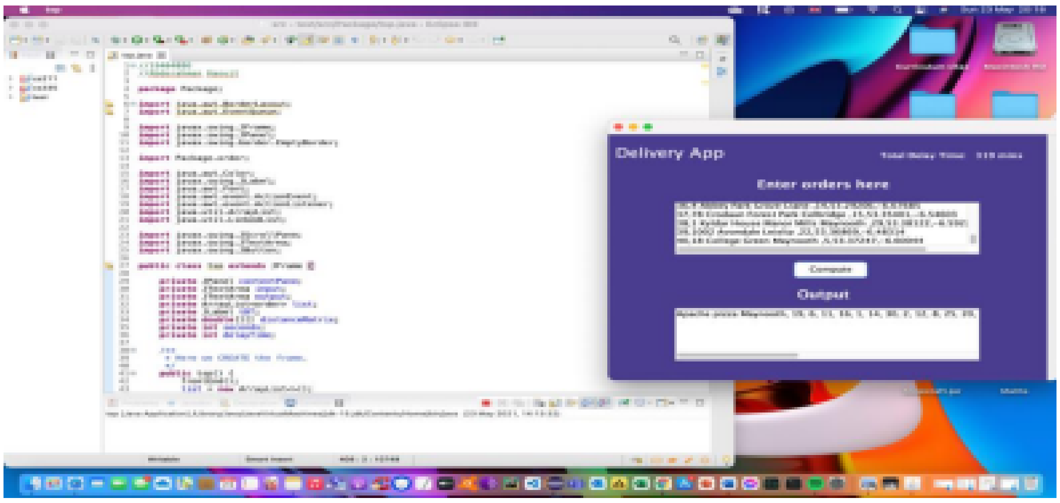
can be difficult to grasp and master at first, but the more practice I got, the better I got. It was a difficult task in this project to compute 100 orders in under 10 seconds while minimizing the delay time, as it involved general knowledge of code structure, Objects, and the GUI, after that the best algorithm for solving this computational problem had to be implemented in order to achieve the goal 'minimizing the delay for delivering pizza'. It is important for an algorithm to use memory as frequently as possible and Knowing algorithms well has benefited me in identifying and optimizing code for this project that I am currently working on. For the GUI side of the project, this project used components from Java's awt and swing libraries in the Java programming language, which gave the app a good decorative finish. Writing graphics applications in Java using Swing can be a challenging task that requires knowledge of many wide libraries as well as some advanced Java concepts. A windowing toolkit is typically responsible for offering a mechanism that allows a graphical user interface (GUI) to render the right bits to the screen at the right time in a graphical environment. A structure like this is provided by both the AWT (abstract windowing toolkit) and Swing. Several tools, including Clear JPanel, JLabel, and others Set the background color as well as the foreground color, font, and other elements.

reference:

1.“Breadth First Search or BFS for a Graph,” *GeeksforGeeks*, 04-Dec-2020. [Online]. Available: https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/. [Accessed: 24-May-2021]

2. “Breadth First Traversal for a Graph | GeeksforGeeks,” *YouTube*, 01-Sep-2016. [Online]. Available: https://www.youtube.com/watch?v=0u78hx 66Xk&ab_channel=GeeksforGeeks. [Accessed: 24-May-2021]

4. Appendix:



“Apache is printed out as an output with the other order numbers as it is a fixed starting point in my code and also with a result of 319 minutes as delay time for 40 orders”.

```java
//19404086

//Abderahman Haouit

package Package;

import java.awt.BorderLayout;

import java.awt.EventQueue;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.border.EmptyBorder;

import Package.order;

import java.awt.Color;

import javax.swing.JLabel;

import java.awt.Font;

import java.awt.event.ActionEvent;

import

java.awt.event.ActionListener;

import java.util.ArrayList;

import java.util.LinkedList;

import

javax.swing.JScrollPane;

import javax.swing.JTextArea;

import javax.swing.JButton;

public class tsp extends JFrame {

private JPanel contentPane;

private JTextArea input;

private JTextArea output;

private ArrayList<order> list;
private JLabel tDT;
private double[][] distanceMatrix;
private int seconds;
private int delayTime;

/**
* Here we CREATE the frame.
*/
public tsp() {
frontEnd();
list = new ArrayList<>();
distanceMatrix = new double[41][41];
seconds=0;
delayTime=0;
}

public void creatingVertices() {

order startingPoint= new order(0, "Apache pizza Maynooth", 0, 53.29235,-6.68577);//The starting point
coordinates.
list.add(startingPoint);
/*
This will retrieve all orders from the text area
in which the user has pasted the orders and
it will store all those orders into a string.
*/
String temp = input.getText();

/*
These five local variables are used
to store a specific attribute of the
order.
*/
String orderNum = "";
String address = "";
String aTime = "";
String longitude = "";
String latitude = "";

/*
i is a local variable used for
loop iteration.
checker is a local variable used
to track ',' in the order.
*/
int i = 0;
int checker = 0;

/*
The loop will keep iterating
until it has reached the
last character of the entries
*/
while (i != temp.length()) {

/*
Here we will read the whole string

if the checker is 0, the program will
store the character into the orderNum

if the checker is 1, the program will
store the character into the
address

if the checker is 2, the program
will store the character into the
aTime

if the checker is 3, the program will
```

```
store the character into the longitude


if the checker is 4, the program will
store the character into the latitude


if ',' occurs, the value of the checker
will be incremented
*/


char character = temp.charAt(i);
if (character == ',')
checker++;
/* \n means the next line.
if this occurs, we will
set value of checker to 0
so it could start storing
data from the oderNum.
\n represents that a specific
order is read by a program
successfully.
*/
else if(character == '\n') {
orderNum = orderNum.replaceAll("\\s", "");
//System.out.println(orderNum+" "+address+" "+aTime+" "+longitude+" "+latitude);
order Order = new order(Integer.parseInt(orderNum), address, Integer.parseInt(aTime),
Double.parseDouble(longitude), Double.parseDouble(latitude));
list.add(Order);
//System.out.println(graph.list.toString());
checker = 0;
orderNum = "";
address = "";
aTime = "";
longitude = "";
latitude = "";
}
```

```
else if (checker == 0) {
orderNum = orderNum + character;
}
else if (checker == 1) {
address = address + character;
}

else if (checker == 2) {
aTime = aTime + character;
}

else if (checker == 3) {
longitude = longitude + character;
}

else if (checker == 4) {
latitude = latitude + character;
}

i++;
}
}

/**
* Here the distance matrix will
* be calculated by keeping in
* view of all the vertices
*/
public void creatingDistanceMatrix()
{

for(int i=0; i<list.size();i++)//This represents the row
{
for(int j=0; j<list.size(); j++)//This represents the column
{
```

```
if(i==j)

{
distanceMatrix[i][j]=0;

}

else

{

/*

This is an implementation of the distance formula on two

specific orders. The distance is calculated by the

longitude and latitude of both orders.

*/

double result=Math.sqrt(Math.pow(list.get(i).logitude-list.get(j).logitude, 2)

+ Math.sqrt(Math.pow(list.get(i).latitude-list.get(j).latitude, 2)));

distanceMatrix[i][j]=(double)list.get(j).mA-result;

}

}

}

}


/**

* I have used BFS algorithm for finding

* the shortest path. The method is also

* calculating the total delay time.

*/

public void findingShortestPath()

{

String output="";

boolean checker=false;/*

A local variable used for

checking whether the order

number is visited or not.

*/

double min=0;//This is used for storing minimum distance.

int index=0;

LinkedList<order> queue= new LinkedList<>();//A temporary queue to perform the BFS
```

```
Algorithm queue.addFirst(list.get(0)); /*

Write the index of the order

through which you want to
start the delivery of Pizzas.

*/

while(!visited())//This will iterate until all orders are visited

{

order current=queue.removeFirst();

if(current.orderID==0)

output=output+current.address;

else

output=output+(", "+Integer.toString(current.orderID));

min=500; /*

A random value given to the min variable,

we can get first minimum distance in the

distance matrix.

*/

index=0;

for(int i=0; i<distanceMatrix.length;i++)/*

This will iterate till the end of the last index

of a specific row of distance matrix.

*/

{

/*

If the specific order is not visited, then it will check

for the minimum distance with other orders; otherwise, the

loop will be terminated.

*/


if(list.get(current.orderID).visited==false)

{

checker=true;

if(min>distanceMatrix[current.orderID][i] && distanceMatrix[current.orderID][i] !=

0 && list.get(i).visited==false)

{

min=distanceMatrix[current.orderID][i];
index=i;
}
}
else
break;
}

list.get(current.orderID).visited=true; //This will make the order number true(boolean)
//true means that the order is visited.
if(checker==true) {
queue.add(list.get(index));
}
/*
If the order was already visited, the program will increment to the
next unvisited order.
*/

else
{
for(int i=0; i<list.size();i++)
{
if(list.get(index).visited==true)
{
index++;
}
else
{
queue.add(list.get(index));
break;
}
}
}
checker=false;

/*
```

```
Here I am incrementing minutes awaiting time
of every unvisited order after the delivery
of every single order.
*/
while(seconds != list.size())
{
if(list.get(seconds).visited==false)
{
list.get(seconds).mA++;
}
seconds++;
}
seconds=0;
}

/*
Once all orders are delivered, here I am calculating
total delay time.
*/
for(int i=0; i<list.size();i++)
{
if(list.get(i).mA>30)
{
int temp= list.get(i).mA-30;
delayTime=delayTime+temp;
}
}

//Displaying the shortest path and total delay
time.
this.output.setText(this.output.getText()+output);
tDT.setText(Integer.toString(delayTime)+" mins");
}

/**
```

```
* if all orders are visited
* it will return true.
* @return
*/
public boolean visited()
{
int temp=0;
while(temp != list.size())
{
if(list.get(temp).visited==false)
return false;
temp++;
}
return true;
}
//GUI display design
public void frontEnd()
{
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
; setBounds(100, 100, 615, 476);
contentPane = new JPanel();
contentPane.setBorder(new EmptyBorder(5, 5, 5,
5)); setContentPane(contentPane);
contentPane.setLayout(null);

JPanel panel = new JPanel();
panel.setBackground(new Color(72, 61, 139));
panel.setBounds(0, 0, 599, 520);
contentPane.add(panel);
panel.setLayout(null);

JLabel title = new JLabel("Delivery App");
title.setBounds(10, 11, 161, 44);
title.setForeground(new Color(255, 255, 255));
title.setFont(new Font("Segoe UI Symbol", Font.PLAIN, 24));
```

```
panel.add(title);

JLabel inputTitle = new JLabel("Enter orders here");
inputTitle.setBounds(203, 68, 189, 44);
inputTitle.setForeground(Color.WHITE);
inputTitle.setFont(new Font("SimSun-ExtB", Font.BOLD,
20)); panel.add(inputTitle);
JScrollPane scrollPane = new JScrollPane();
scrollPane.setBounds(91, 123, 416, 95);
panel.add(scrollPane);

input = new JTextArea();
scrollPane.setViewportView(input);

JLabel outputTitle = new JLabel("Output");
outputTitle.setForeground(Color.WHITE);
outputTitle.setFont(new Font("SimSun-ExtB", Font.BOLD, 20));
outputTitle.setBounds(258, 275, 98, 30);
panel.add(outputTitle);

JScrollPane scrollPane2 = new JScrollPane();
scrollPane2.setBounds(91, 316, 416, 95);
panel.add(scrollPane2);

output = new JTextArea();
scrollPane2.setViewportView(output);

JLabel totalDelayTimeTitle = new JLabel("Total Delay Time:");
totalDelayTimeTitle.setForeground(Color.WHITE);
totalDelayTimeTitle.setFont(new Font("SimSun-ExtB", Font.BOLD, 13));
totalDelayTimeTitle.setBounds(371, 16, 136, 44);
panel.add(totalDelayTimeTitle);

tDT = new JLabel("0"+" mins");
tDT.setForeground(Color.WHITE);
tDT.setFont(new Font("SimSun-ExtB", Font.BOLD, 13));
tDT.setBounds(501, 23, 70, 30);
panel.add(tDT);

JButton compute = new JButton("Compute");
compute.setBounds(251, 229, 105, 35);
panel.add(compute);
setVisible(true);
compute.addActionListener(new ActionListener() {

@Override
public void actionPerformed(ActionEvent e) {
creatingVertices();
creatingDistanceMatrix();
System.out.println();
findingShortestPath();
}
});
}

/**
* Launches the application.
*/
public static void main(String[] args) {
tsp deliveryApp= new tsp();
}
}

//Attributes for the order.
class order {
int orderID;
String address;
int mA;
Double logitude;
Double latitude;
```

```
boolean visited;

public order(int orderID, String address, int mA, Double logitude, Double latitude)

{ this.orderID = orderID;

this.address = address;

this.mA = mA;

this.logitude = logitude;

this.latitude = latitude;
visited=false;

}

            }
```