اونيۏرسيتي تيكنولوڬي مارا
UNIVERSITI TEKNOLOGI MARA

Cawangan Pahang
Kampus Raub

CSC305: PROGRAMMING PARADIGMS

SEMESTER OCTOBER 2021 - FEBRUARY 2022

GROUP PROJECT (CLO2, CLO3)

Student Details:

No	Student Name	Student ID	Class	Declaration Signature
1	MUHAMMAD AZRI BIN MOKHZANI	2020836218	CS1103E	*Azri*
2	MUHAMMAD AIRUL HAFIQ BIN MUHAMAD AINI @ AHMAD	2020487774	CS1103E	*Airul*
3	LUQMAN SYAKIR BIN YAHYA	2020893248	CS1103E	*Luqman*

Declaration:

I/We confirm that I/We read and shall comply with all the terms and conditions of the UiTM plagiarism policy.

I/We confirm that the online submission had also been sent accordingly with the attachment through the UiTM i-Learn portal and I/We had already checked the submission regularly before submission. Any problem due to the submission will be bare on our responsibilities.

I/We declare that this assignment is free from all forms of plagiarism and for all intents and purposes is my/our own properly derived work.

I/We further confirm that the same work, where appropriate, has been verified by anti-plagiarism software: (*Please insert if applicable*).

Date: 22 January 2022

Table Of Contents

1. Your group task is to construct the development of Djikstra’s algorithm by using the selected programming language and attaching your complete programming code

```
/***************************Djikstra Programming*************************/

import java.util.ArrayList;
class Main{
   public static void main(String[] args){
       Algo a = new Algo();
       ArrayList<Path> p = new ArrayList<>();
       //Problem 1
       System.out.println("Problem 1");
       p.add(new Path("S", new int[] {0,1,0,2,0,0}));
       p.add(new Path("A", new int[] {0,0,6,0,0,0}));
       p.add(new Path("B", new int[] {0,0,0,0,1,2}));
       p.add(new Path("C", new int[] {0,4,0,0,3,0}));
       p.add(new Path("D", new int[] {0,0,0,0,0,1}));
       p.add(new Path("E", new int[] {0,0,0,0,0,0}));

       a.initiateDjikstraProg(p,0);
       System.out.println("\n");
       p.clear();

       //Problem 2
       System.out.println("Problem 2");
       p.add(new Path("A",new int[] {0,0,0,0,7}));
       p.add(new Path("B",new int[] {4,0,0,0,0}));
       p.add(new Path("C",new int[] {0,1,0,0,0}));
       p.add(new Path("D",new int[] {0,2,6,0,0}));
       p.add(new Path("E",new int[] {0,0,0,0,0}));

       a.initiateDjikstraProg(p,3);
       System.out.println("\n");
       p.clear();

       //Problem 3
       System.out.println("Problem 3");
       p.add(new Path("A", new int[] {0,3,5,9,0,0}));
       p.add(new Path("B", new int[] {3,0,3,4,7,0}));
       p.add(new Path("C", new int[] {5,3,0,2,6,8}));
       p.add(new Path("D", new int[] {9,4,2,0,2,2}));
       p.add(new Path("E", new int[] {0,7,6,2,0,5}));
       p.add(new Path("F", new int[] {0,0,8,2,5,0}));

       a.initiateDjikstraProg(p,0);
       System.out.println("\n");
       p.clear();
```

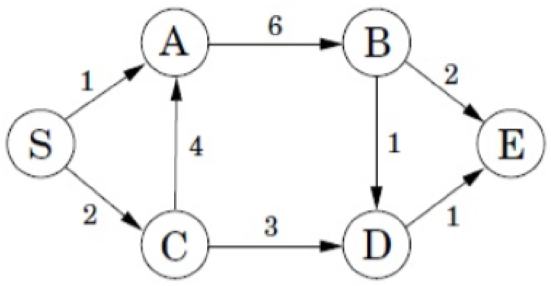
```
        //Problem 4
        System.out.println("Problem 4");
        p.add(new Path("A", new int[] {0,3,5,0,0}));
        p.add(new Path("B", new int[] {0,0,2,6,0}));
        p.add(new Path("C", new int[] {0,1,0,4,6}));
        p.add(new Path("D", new int[] {0,0,0,0,2}));
        p.add(new Path("F", new int[] {3,0,0,7,0}));

        a.initiateDjikstraProg(p,0);
        System.out.println("\n");
        p.clear();

        //Problem 5
        System.out.println("Problem 5");
        p.add(new Path("A",new int[] {0,4,2,0,0,0}));
        p.add(new Path("B",new int[] {0,0,5,10,0,0}));
        p.add(new Path("C",new int[] {0,0,0,0,3,0}));
        p.add(new Path("D",new int[] {0,0,0,0,0,11}));
        p.add(new Path("E",new int[] {0,0,0,4,8,0}));
        p.add(new Path("F",new int[] {0,0,0,0,0,0}));

        a.initiateDjikstraProg(p,0);
        System.out.println("\n");
        p.clear();

/** How to initiate Djikstra
* 1. initialize the class for Algo and Path.
*     Algo algo = new Algo();
* ArrayList<Path> path = new ArrayList()<>;
*
* 2. Enter the value to path as follows :-
*     path.add(new Path([String VertexName],[int array of distance]));
*
* 3. Enter the value to algo as follows :-
* algo.initiateDjikstraProg(path,source);
*
*           !!!Important Notes!!!
* To Enter the value of Path you need to imagine it as a table.
* _______________________________
* |Vertex|    A  |   B  | C  |  D  |
* -------------------------------
* |  A   |    0  |   3  | 4  |  0  |
* -------------------------------
* |  B   |    0  |   0  | 2  |  0  |
* -------------------------------
* |  C   |    0  |   0  | 0  |  0  |
* -------------------------------
* |  D   |    5  |   0  | 0  |  0  |
* ------------------------------
*
* The graph is illustrated as below
*
*           5           3
*     D --------> A --------> B
*                 |           |
*               4 |           | 2
*                 |           |
*                 +---> C <---+
*
* We can conclude that D is a starting point while C is the ending point
* based on the picture above.
*
* So, enter the value as written :-
* path.add(new Path("A",new int[] {0,3,4,0}));
* path.add(new Path("B",new int[] {0,0,2,0}));
* path.add(new Path("C",new int[] {0,0,0,0}));
* path.add(new Path("D",new int[] {5,0,0,0}));
*
* a.initiateDjikstraProg(path,3); *Source is 3 as we picked "D" as our
*                     starting point(using path[index of 3)

**/
   }
}

class Algo {
   private int numOfPoints;
   private Path temp;

   public void initiateDjikstraProg(ArrayList path,int source){
       numOfPoints = path.size();
       int[] distance = new int[numOfPoints];
       String[] status = new String[numOfPoints];

/********Part 1 (Initialization of each possible path and distance)**********/
       for (int i = 0; i < numOfPoints; i++) { /** Assume that all distance
are infinity and NOT VISITED **/
           distance[i] = Integer.MAX_VALUE;
           status[i] = "Not Visited";
       }

       distance[source] = 0; /** Source Vertex will always be 0 as it is
pointed to itself **/
       for (int i = 0; i < numOfPoints - 1; i++) {
           int index = 0;
           int min = Integer.MAX_VALUE;
```

```
/*************Part 2 (Finding the minimum value of the path)***************/
           for (int j = 0; j < numOfPoints; j++) { /** Find the minimum
distance of a vertex and its index position will be used as a marker**/
               if(distance[j] <= min && status[j].equalsIgnoreCase("Not
Visited")){
                   min = distance[j];
                   index = j;
               }
           }
           status[index] = "Visited"; /** The index position of minimum
distance will be marked as VISITED**/
           temp = (Path) path.get(index); /** Get the value of minimum
distance input**/

/************Part 3 (Comparing the distance for each vertex)**************/
           for (int v = 0; v < numOfPoints; v++) {
               /** Compare the distance that are (NOT VISITED) && (HAS A VALUE
> 0 ) && (IS NOT AN INFINITY VALUE) && (The total minimum distance added by
current value must be < current distance)**/
               /** When the current value of distance are lower than (minimum
distance + current value of input), the current value of distance will be
updated.**/
               /** For Example :
                *
                * [0]   2    [2]   3  [Infinity]
                * (A) -----> (B) -----> (C)
                *
                * From distance A to B, we know that the value is 2 as B is
the closest possible distance from A.
                * But the total distance of C is INFINITY as it was currently
assumed that the distance from A to C has been set to INFINITY.
                *
                * But, using the formula = (minimum distance + current value
of input) < current value of distance, we can change the total value of C.
                * (Current Value of B + Distance From B to C) COMPARED TO
(Total Value of C)
                * [ 2 + 3 < INFINITY]
                * [5 < INFINITY]
                *
                * (5) is lower than infinity thus it is true.
                * When the statement is true we can convert the total Distance
of C to (minimum distance + current value of input)
                *
                * [0]   2    [2]   3    [5]
                * (A) -----> (B) -----> (C)
                * **/
               if (status[v].equalsIgnoreCase("Not Visited") &&
temp.getValue()[v] != 0 && distance[index] != Integer.MAX_VALUE &&
distance[index] + temp.getValue()[v] < distance[v]) {
```

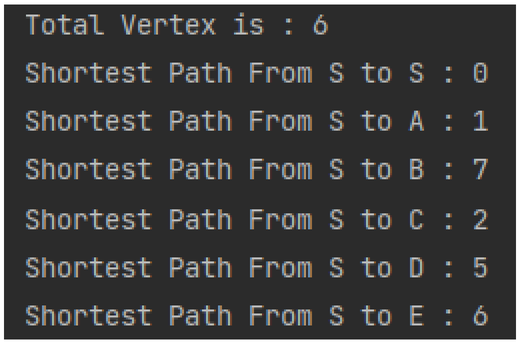
```
                        distance[v] = distance[index] + temp.getValue()[v];
                    }
                }
            }

            /** Print the Shortest Path from source and total number of Vertices**/
            System.out.println("Total Vertex is : " + path.size());
            for (int i = 0; i < numOfPoints; i++) {
                temp = (Path) path.get(i);
                Path startingPoint = (Path) path.get(source);
                System.out.println("Shortest Path From " +
startingPoint.getPoints() + " to " + temp.getPoints() + " : " + distance[i]);
            }
        }
}

class Path { /** General Class of Path which contain Vertices Points and Edges
Value**/
private String points;
   private int[] value;

   public Path(){
       points = null;
       value = null;
   }
   public Path(String points, int[] value){
       this.points = points;
       this.value = value;
   }
   public void setValue(int[] value) {this.value = value;}
   public void setPoints(String points) {this.points = points;}
   public int[] getValue() {return value;}
   public String getPoints() {return points;}

}
/******************************End of Programs******************************/
```

2. From the development, find the answer to the shortest path problems provided.

Problem 1



```
Algo a = new Algo();
ArrayList<Path> p = new ArrayList<>();
                                //{A,B,C,D,E}
p.add(new Path("S", new int[] {0,1,0,2,0,0}));
p.add(new Path("A", new int[] {0,0,6,0,0,0}));
p.add(new Path("B", new int[] {0,0,0,0,1,2}));
p.add(new Path("C", new int[] {0,4,0,0,3,0}));
p.add(new Path("D", new int[] {0,0,0,0,0,1}));
p.add(new Path("E", new int[] {0,0,0,0,0,0}));

a.initiateDjikstraProg(p,0);
```



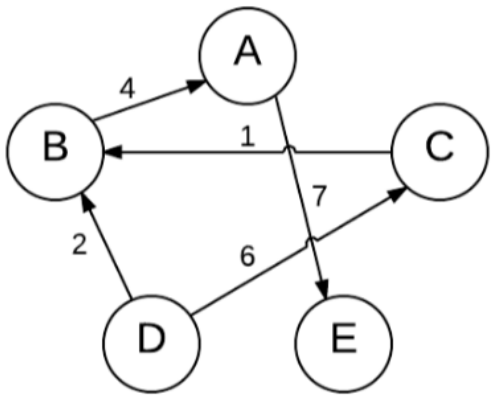
```
Total Vertex is : 6
Shortest Path From S to S : 0
Shortest Path From S to A : 1
Shortest Path From S to B : 7
Shortest Path From S to C : 2
Shortest Path From S to D : 5
Shortest Path From S to E : 6
```

By looking at the output we can conclude that the shortest path from S to E is **:**

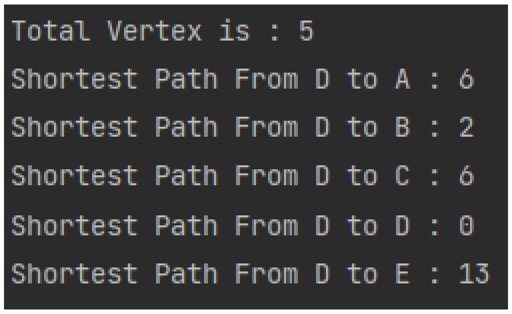
S - C - D - E
0 + 2 + 3 + 1

S to E = 6

Problem 2



```
Algo a = new Algo();
ArrayList<Path> p = new ArrayList<>();
                                 //{A,B,C,D,E}
p.add(new Path("A",new int[] {0,0,0,0,7}));
p.add(new Path("B",new int[] {4,0,0,0,0}));
p.add(new Path("C",new int[] {0,1,0,0,0}));
p.add(new Path("D",new int[] {0,2,6,0,0}));
p.add(new Path("E",new int[] {0,0,0,0,0}));

a.initiateDjikstraProg(p,3);
```



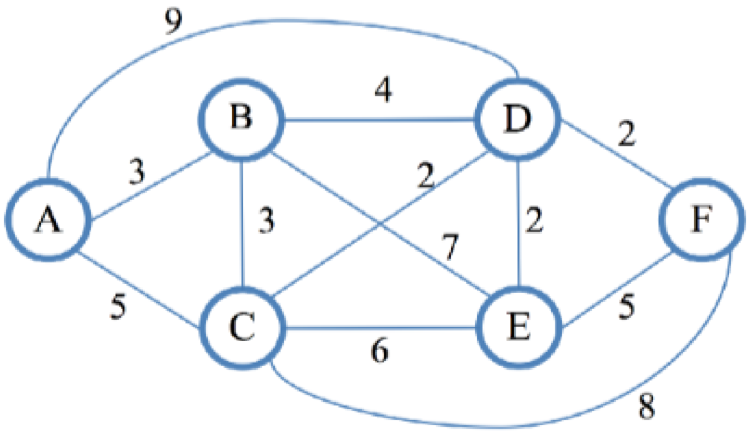
```
Total Vertex is : 5
Shortest Path From D to A : 6
Shortest Path From D to B : 2
Shortest Path From D to C : 6
Shortest Path From D to D : 0
Shortest Path From D to E : 13
```

By looking at the output we can conclude that the shortest path from D to E is **:**

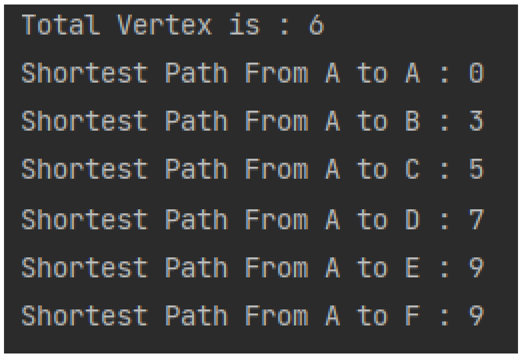
D - B - A - E
0 + 2 + 4 + 7

D to E = 13

Problem 3



```
Algo a = new Algo();
ArrayList<Path> p = new ArrayList<>();
                                 //{A,B,C,D,E}
p.add(new Path("A", new int[] {0,3,5,9,0,0}));
p.add(new Path("B", new int[] {3,0,3,4,7,0}));
p.add(new Path("C", new int[] {5,3,0,2,6,8}));
p.add(new Path("D", new int[] {9,4,2,0,2,2}));
p.add(new Path("E", new int[] {0,7,6,2,0,5}));
p.add(new Path("F", new int[] {0,0,8,2,5,0}));

a.initiateDjikstraProg(p,0);
```



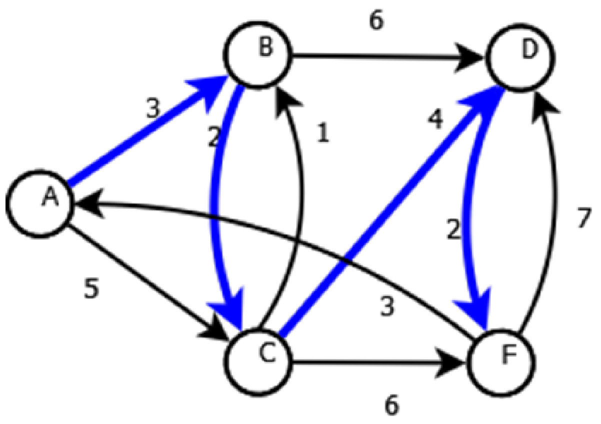
```
Total Vertex is : 6
Shortest Path From A to A : 0
Shortest Path From A to B : 3
Shortest Path From A to C : 5
Shortest Path From A to D : 7
Shortest Path From A to E : 9
Shortest Path From A to F : 9
```

By looking at the output we can conclude that the shortest path from A to F is :

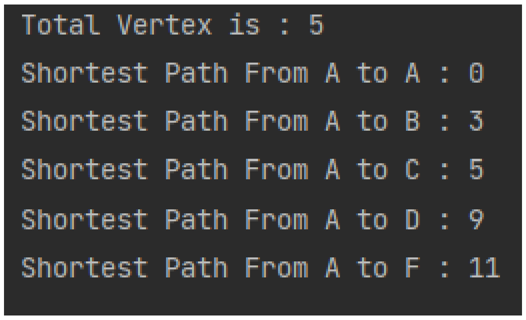
A - B - D - F
0 + 3 + 4 + 2

A to F = 9

Problem 4



```
Algo a = new Algo();
ArrayList<Path> p = new ArrayList<>();
//{A,B,C,D,E,F}
p.add(new Path("A", new int[] {0,3,5,0,0}));
p.add(new Path("B", new int[] {0,0,2,6,0}));
p.add(new Path("C", new int[] {0,1,0,4,6}));
p.add(new Path("D", new int[] {0,0,0,0,2}));
p.add(new Path("F", new int[] {3,0,0,7,0}));

a.initiateDjikstraProg(p,0);
```

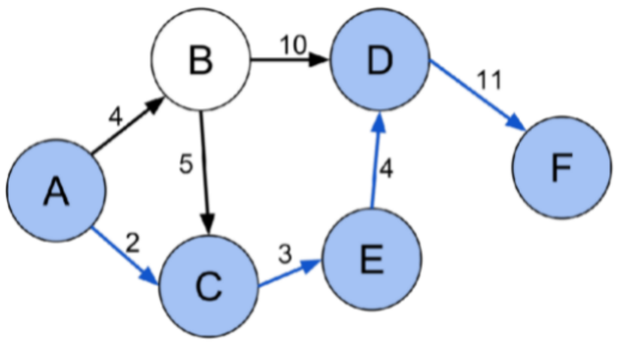


```
Total Vertex is : 5
Shortest Path From A to A : 0
Shortest Path From A to B : 3
Shortest Path From A to C : 5
Shortest Path From A to D : 9
Shortest Path From A to F : 11
```

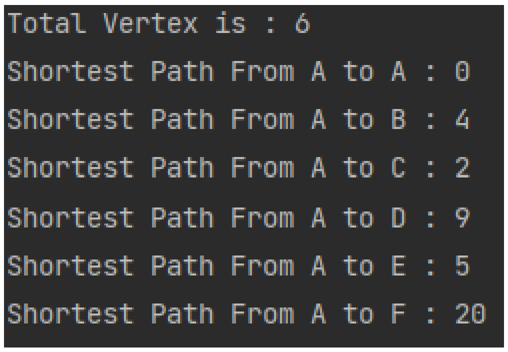
By looking at the output we can conclude that the shortest path from A to F is **:**

A - B - C - D - F
0 + 3 + 2 + 4 + 2
A to F = 11

Problem 5



```
Algo a = new Algo();
ArrayList<Path> p = new ArrayList<>();
                                 //{A,B,C,D,E}
p.add(new Path("A",new int[] {0,4,2,0,0,0}));
p.add(new Path("B",new int[] {0,0,5,10,0,0}));
p.add(new Path("C",new int[] {0,0,0,0,3,0}));
p.add(new Path("D",new int[] {0,0,0,0,0,11}));
p.add(new Path("E",new int[] {0,0,0,4,8,0}));
p.add(new Path("F",new int[] {0,0,0,0,0,0}));

a.initiateDjikstraProg(p,0);
```



```
Total Vertex is : 6
Shortest Path From A to A : 0
Shortest Path From A to B : 4
Shortest Path From A to C : 2
Shortest Path From A to D : 9
Shortest Path From A to E : 5
Shortest Path From A to F : 20
```

By looking at the output we can conclude that the shortest path from A to F is **:**

A - C - E - D - F

0 + 2 + 3 + 4 + 11

A to F = 20

3. Explain your experiences in developing the case study from the view of programming languages, characteristics, and benefits of the programming language in solving the case study

3.1 Developing of Case Study from Programming Language Perspectives

Day 1 (21 December 2021 - 22 December 2021)

On the first day, we decided to view the given case study video and identify its problem as well as its concept so that we could build the algorithm with ease later on. We spent hours on digesting about the video regarding of the development of Djikstra Programming. After that, we decided to view the explanation of the Djikstra algorithm concept from other sources to deepen our understanding such as:-

- 3.6 Dijkstra Algorithm - Single Source Shortest Path - Greedy Method, Abdul Bari (10 February 2018), **https://www.youtube.com/watch?v=XB4MIexjvY0&t=244s**

- How to use Dijkstra's Algorithm with Code,
  Gaurav Sen (1 August 2017),
  https://www.youtube.com/watch?v=d6ZFqjH63vo&t=12s

- Dijkstra's Algorithm with Example,
  Beena Ballal (12 April 2020),
  https://www.youtube.com/watch?v=Rv2RzVu9S9k

Day 2 (24 December 2021 - 26 December 2021)

After analyzing the concept of Djikstra and its problem we started to divide our work into parts so that we could improve the working process as well as lessen the burden of each group member. The divided parts and their workflow are listed as follows:-

1) Initialization: In this part, the selected group member needs to declare the variable that will be used as well as initialize its value so that it could be used in

further program segments. This includes the initialization of the maximum value in each vertex and setting the value of each vertex as “NOT VISITED” at the beginning of each loop.

2) Comparison: In this part, the selected group member must compare the value of the input distance that has been entered by the user. If the value entered is 0, then it will not be compared to. Otherwise, the value will be stored in a temporary variable and compared to other values in the same vertex. The location of the least value is stored and will be used as a marker.

3) Calculation: In this part, the selected group member will need to perform the calculation of Djikstra. The value will be calculated if and only if it meets certain requirements such as :

    a) The current value is not set as the maximum value.
    b) The current value is not set as “VISITED”.
    c) The input value is not 0.
    d) The total of current value and the input value is less than the least amount of value in a vertex.

Day 3 (28 December 2021 - 29 December 2021)

There is some task that has been completed such as the stage of initialization. Some of our group members encountered some difficulty in completing some tasks but we managed to complete them by looking at some sources as references. With the guide of these references, we could have not completed this case study on time.

- Beginners Book by Chaitanya Singh (2021),
  Java – Finding minimum and maximum values in an array,
  https://beginnersbook.com/2014/07/java-finding-minimum-and-maximum-values-in-an-array/

- Stack Overflow by Justin (15 January 2012),
  Dijkstra's Algorithm Termination,
  https://stackoverflow.com/questions/8870474/dijkstras-algorithm-termination

After all of the task has been completed, all group members are requested to do documentation in each task so that users can view our explanation whenever they want to review the calculation for our algorithm. The documentation will also provide some examples so that the user will know how to input the value as well as the detailed explanation behind the concept of Djikstra.

Day 4 (30 December 2021 - 4 January 2022)

We started to finalize our algorithm by checking any possible errors and making sure to display the output properly so that it can increase the readability of the programs. In the end, our algorithm has reached 140+ lines of codes including the documentation. After the algorithm has been completed we decided to answer every question asked in this case study and started the completion of paperwork which took about another 4 days.

3.2 Characteristics of the Programs

a) Easy to understand and use

The coding is divided into 3 parts. Every part has at least one comment to explain the function of the coding. The comments also explain what and how much value to input. If there is any confusion for the user, they can always refer back to the comment.

b) Implementation of Abstraction

In this project, we have used the abstraction implementation to make our codes shorter. It helps to hide unnecessary details from users. For example, we used a user-defined function named initiateDjikstraProg(), and whenever we want to solve a djikstra problem, we only need to call the function and insert the value. Djikstra operation has been separated from the main method so the user only needs to call the function in the main class only. It helps as we can reuse the same code and avoid code duplication. With this abstraction implementation, the readability for this case study increased.

c) Reliable and Flexible

The code is divided into 3 parts :

- Part 1: Initialization of each possible path and distance.
- Part 2: Finding the minimum value of each vertex.
- Part 3: Comparing the distance for each vertex.

With the separation of the algorithm, modification of the codes can easily to done as there is clear documentation for each part. The user is able to learn and know how the algorithm works through the documentation.

3.3 Benefits of the Programming Languages in solving the case study

a) The Familiarity of the current Programming Language is high

The first reason why we chose this programming language is that we have learned this programming language which is Java for almost two years. It makes this Djikstra shortest path algorithm easy and time-efficient to solve with Java programming language. Another reason why Java programming language is our main programming language in solving this Djikstra algorithm is that we are familiar with this programming language. If there is any error or logic mistake, we can easily figure it out.

b) Have many useful Libraries that have been studied.

In the java Case study, we have learned how to use ArrayList and Object-oriented programming. So, we can implement what we have learned in Fundamentals of data structure to make this Djikstra algorithm with ease. When we use what we have learned for two years, it becomes easy for us to solve this task and we know all the functions in the Integrated Development Environment (IDE) very well.

c) The IDE of the current Programming Language has many useful features

We use IntelliJ as our integrated development environment (IDE), IntelliJ provides many features in their IDE to ease programmers. The main reason why we use IntelliJ is because of the two features called " Code-with-me" and "Auto-correct".

- Code with me is the feature that allows us to build programs together without any hesitation due to pandemic Covid-19 as it allows user to share their codes with others in real-time.
- The auto-correct feature will scan all the errors in the whole program, if there is any error, it will recommend the solution and make our program efficient

In the presence of Intellij, we manage to complete case study even faster than normal while any problems that occur can be fixed in real-time. This is the reason why we choose Intellij as our main IDE.

d) Many examples and guidance can be found online

Java is also a very famous language that has many references on the Internet. We can simply search for guidance on websites such as Quora, Coursesity, W3school, and Geekforgeeks. If we are still stuck on any code, we will search for a solution there or ask in the community section. They will guide on how to solve the current problem about the case study. We make their explanation as a reference to learn a new thing and concept to complete the case study.

4. Project Assessment Rubric

Criteria	**Outstanding (40 -50 marks)**	**Very Good (31 - 40 marks)**	**Good (21 - 30 marks)**	**Average (11 - 20 marks)**	**Poor (0 -10 marks)**
Development	The development is correct and outstanding relevant to the case study with more than 80% correct. Readable and easy to understand.	The development is correct and relevant to the case study with more than 70% correct. Readable and easy to understand.	The development is relevant to the case study with more than 50% correct. Readable and easy to understand.	The development is relevant to the case study with less than 50% correct. Readable and easy to understand.	The development is not relevant to the case study with less than 30% correct. Readable and easy to understand.
Criteria	**Outstanding (20-25 marks)**	**Very Good (16 - 19 marks)**	**Good (11 - 15 marks)**	**Average (6 - 10 marks)**	**Poor (0 - 5 marks)**
Coding Originality	The originality is outstanding in explanation and relevant to the case study with 100% originality. Readable and easy to understand.	The originality is very good in explanation and relevant to the case study with 80-90% originality. Readable and easy to understand.	The originality is good in explanation and relevant to the case study with 60-70% originality. Readable and easy to understand.	The originality is average in explanation and somehow relevant to the case study with 50% originality. Readable and easy to understand.	The originality is poor in explanation and not relevant to the case study with less than 50% originality. Readable and easy to understand.
Coding	The	The	The	The	The

Formatting	formatting is outstanding with no formatting errors, readable, and easy to understand.	formatting is very good 10-20% formatting errors, readable, and easy to understand.	formatting is good with 30% formatting errors, readable, and easy to understand.	formatting is average with 40% formatting errors, readable, and easy to understand.	formatting is poor with more than 50% formatting errors, readable, and easy to understand.