FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

GROUP NAME: ALLIES

CODE: CSC4202

LECTURER: PROF. MADYA DATIN DR. NORWATI MUSTAPHA

No.	Name	Matric No.
1.	Hazeera Binti Misman	212725
2.	Siti Nur Ainsyah Binti Azhari	212863
3.	Nurul Ainna Syaffa Binti Marzuki	212855

TABLE OF CONTENTS

1.0 PROJECT PLANS

Initial Project Plan (week 10, submission date: 26th May 2023)

Group Name	Allies
Members	Name / Email / Phone number: HAZEERA BINTI MISMAN — 212725@student.upm.edu.my — 0177217547 SITI NUR AINSYAH BINTI AZHARI — 212863@student.upm.edu.my — 0175024557 NURUL AINNA SYAFFA BINTI MARZUKI — 212855@student.upm.edu.my — 0194038580
Problem scenario description	One of the most popular e-commerce companies in a bustling metropolis prides itself on delivering packages within the promised time frame. The logistics of the company's supply chain have become more complex as the company has grown exponentially over the years. Multiple warehouses are operated by the company throughout the city, each holding a large inventory of products. A strategic location of these warehouses allows quick access to multiple parts of the city, minimising delivery times. In spite of this rapid expansion, the company is now facing difficulties in optimising its supply chain logistics.
Why it is important	Since multiple warehouses are located at different locations throughout the city, the logistics company needs to ensure that all the products are received within the given time frame. In order for the logistics company to enhance customer satisfaction, cut costs, enable scalability, and adapt to changing circumstances, they need to figure out the best path which can optimise their supply chain logistics.

Members:

Name	Email	Phone number
HAZEERA BINTI MISMAN	212725@student.upm.edu.my	0177217547
SITI NUR AINSYAH BINTI AZHARI	212863@student.upm.edu.my	0175024557
NURUL AINNA SYAFFA BINTI MARZUKI	212855@student.upm.edu.my	0194038580

	Thus, determining the best possible path to deliver all the packages is one of the solutions that can help to achieve all the goals by taking into account all the factors that may reduce the optimisation of the delivery such as distances and traffic.
Problem specification	1. Route Optimization Finding the best possible route to deliver packages to all warehouses throughout the city within the given time frame. 2. Cost Reduction Minimising travel distance as short as possible and optimising routes to reduce costs associated with transportation, such as fuel consumption.
Potential solutions	1. Greedy algorithm by using Djikstra's algorithm 2. Dynamic programming by using Floyd's algorithm
Sketch (framework, flow, interface)	Steps to Solve Route Optimization Problem (Greedy Algorithm): 1. Identify the input data by compiling warehouse addresses, customer addresses, and transportation information. 2. Build the graph according to the data where nodes represent locations(warehouses and customer addresses) and edges represent transportation routes. 3. Initialize the solution. 4. Iterate through the customer addresses by calculating the distance or travel time from the current location to the customer address. 5. Make a locally optimal choice by choosing the nearest customer address as the next stop in the route. Next, the current location will be updated after visiting the selected customer address. 6. Finalize the route by adding the return trip from the last customer address to the starting point to complete the route. 7. Evaluate the solution by comparing it with other approaches. Steps to Solve Cost Reduction Problem (Dynamic Programming): 1. Identify the input data by analyzing inventory levels, demand forecasts, transportation costs, and other relevant factor costs. 2. Define the subproblems by deciding which key decisions are to be made in order to reduce costs, such as inventory management, transportation allocation, and order fulfillment. 3. Formulate the recurrence relation by defining the relationship between the subproblems and their optimal solutions.

	Afterward, recursively solve the problem by breaking it down into smaller subproblems. 4. Initialize the dynamic programming table by setting up a matrix or table to store intermediate results and initialize it accordingly. 5. By starting small and gradually increasing the size of the subproblems, fill in the table using the recurrence relations. Analyze the previous solutions to calculate the optimal solution for each subproblem. 6. Calculate the optimal solution by tracing back the steps taken to determine each decision made. 7. Analyze and compare the optimal solution's cost reduction with other approaches by calculating the total cost savings.

Project Proposal Refinement (week 11, submission date: 2nd June 2023)

Group Name	Allies
Members	Name / Role: HAZEERA BINTI MISMAN — Leader SITI NUR AINSYAH BINTI AZHARI — Co-leader NURUL AINNA SYAFFA BINTI MARZUKI — Support
Problem statement	The e-commerce business, which is among the most well-known in a busy metropolis, has seen substantial expansion and has a number of warehouses that are thoughtfully placed across the city to reduce

Name	Role
HAZEERA BINTI MISMAN	Leader
SITI NUR AINSYAH BINTI AZHARI	Co-leader
NURUL AINNA SYAFFA BINTI MARZUKI	Support

	delivery times. However, as the business grew, the logistics of its supply chain complicated, making it challenging to optimize package delivery within the specified time period. Due to variables such as distance and traffic, the corporation now confronts difficulty in assuring timely delivery of items from numerous warehouses to clients. The logistics organization must choose the best way to optimize its supply chain logistics in order to improve consumer satisfaction, reduce costs, enable scalability, and react to changing conditions. This entails devising a system that takes into consideration all important aspects and limits in order to efficiently transport items while accounting for variables such as warehouse locations, delivery distances, and traffic conditions. The business hopes to increase performance in delivering items on schedule by tackling these optimization difficulties and streamlining its supply chain processes.
Objectives	Finding an optimal solution for optimizing supply chain logistics
Expected output	Best possible path to deliver all the packages
Problem scenario description	One of the most popular e-commerce companies in a bustling metropolis prides itself on delivering packages within the promised time frame. The logistics of the company's supply chain have become more complex as the company has grown exponentially over the years. Multiple warehouses are operated by the company throughout the city, each holding a large inventory of products. A strategic location of these warehouses allows quick access to multiple parts of the city, minimising delivery times. In spite of this rapid expansion, the company is now facing difficulties in optimising its supply chain logistics.
Why it is important	Since multiple warehouses are located at different locations throughout the city, the logistics company needs to ensure that all the products are received within the given time frame. In order for the logistics company to enhance customer satisfaction, cut costs, enable scalability, and adapt to changing circumstances, they need to figure out the best path which can optimise their supply chain logistics. Thus, determining the best possible path to deliver all the packages is one of the solutions that can help to achieve all the goals by taking into account all the factors that may reduce the optimisation of the delivery such as distances and traffic.
Problem specification	1. Route Optimization Finding the best possible route to deliver packages to all

	warehouses throughout the city within the given time frame. 2. Cost Reduction Minimising travel distance as short as possible and optimising routes to reduce costs associated with transportation, such as fuel consumption.
Potential solutions	1. Greedy algorithm by using Djikstra's algorithm 2. Dynamic programming by using Floyd's algorithm
Sketch (framework, flow, interface)	Steps to Solve Route Optimization Problem (Greedy Algorithm): 1. Identify the input data by compiling warehouse addresses, customer addresses, and transportation information. 2. Build the graph according to the data where nodes represent locations(warehouses and customer addresses) and edges represent transportation routes. 3. Initialize the solution. 4. Iterate through the customer addresses by calculating the distance or travel time from the current location to the customer address. 5. Make a locally optimal choice by choosing the nearest customer address as the next stop in the route. Next, the current location will be updated after visiting the selected customer address. 6. Finalize the route by adding the return trip from the last customer address to the starting point to complete the route. 7. Evaluate the solution by comparing it with other approaches. Steps to Solve Cost Reduction Problem (Dynamic Programming): 1. Identify the input data by analyzing inventory levels, demand forecasts, transportation costs, and other relevant factor costs. 2. Define the subproblems by deciding which key decisions are to be made in order to reduce costs, such as inventory management, transportation allocation, and order fulfillment. 3. Formulate the recurrence relation by defining the relationship between the subproblems and their optimal solutions. Afterward, recursively solve the problem by breaking it down into smaller subproblems. 4. Initialize the dynamic programming table by setting up a matrix or table to store intermediate results and initialize it accordingly. 5. By starting small and gradually increasing the size of the subproblems, fill in the table using the recurrence relations.

Analyze the previous solutions to calculate the optimal solution for each subproblem.

6. Calculate the optimal solution by tracing back the steps taken to determine each decision made.
7. Analyze and compare the optimal solution's cost reduction with other approaches by calculating the total cost savings.

Methodology

Milestone	Time
- scenario refinement - find problem statement - find objective	26/5 (wk10)
- find example solutions and suitable algorithm. - start coding of the chosen problem	2/6 (wk11)
- edit the coding of the chosen problem - complete the coding - debug	9/6 (wk12)
- conduct analysis of correctness - conduct analysis of time complexity	16/6 (wk13)
- Prepare online portfolio - Prepare presentation	23/6 (wk14)

Project Progress (Week 9 – Week 14)

Milestone 1	- Scenario refinement - Find problem statement - Find objective
Date (Wk)	20 - 26 May 2023
Description/ sketch	- Discuss the scenario and make refinement for the scenario. - Create problem statement from the problem happens in the scenario. - Find the objective to achieve from the scenario.
Role	

Member 1	Member 2	Member 3
Hazeera - make scenario refinement	Ainsyah - create problem statement	Ainna - find objective

Milestone 2	- Find example solutions and suitable algorithm. - Start coding of the chosen problem
Date (Wk)	27 May - 2 June 2023

Description/ sketch	- Find example solutions that can solve the problems. - Decide which solutions that best to implement for the scenario. (which is dijkstra algorithm and floyd algorithm). - Start do pseudocode for each solutions.
Role	

Member 1	Member 2	Member 3
Hazeera - find example solutions and decide suitable solution	Ainsyah - do pseudocode for dijkstra algorithm	Ainna - do pseudocode for floyd algorithm

Milestone 3	- Edit the coding of the chosen problem - Complete the coding - Debug
Date (Wk)	3 - 9 June 2023

Description/ sketch	- Start do coding for dijkstra and floyd algorithm. - Edit the coding to get the desired objective. - Complete the coding. - Debugging of the coding.
Role	

Member 1	Member 2	Member 3
Hazeera - debugging both of the algorithm code	Ainsyah - do coding for dijkstra algorithm	Ainna - do coding for floyd algorithm

Milestone 4	- Complete project report
Date (Wk)	10 - 16 June 2023
Description/ sketch	- Conduct analysis of correctness. - Conduct analysis of time complexity - Do model development

Role	<table><tr><th>Member 1</th><th>Member 2</th><th>Member 3</th></tr><tr><td>Hazeera - do model development</td><td>Ainsyah - conduct analysis of correctness</td><td>Ainna - conduct analysis of time complexity</td></tr></table>

Milestone 5	- Prepare online portfolio - Prepare presentation
Date (Wk)	17 - 23 June 2023
Description/ sketch	- Push the coding to the github. - Make online portfolio in the github. - Start to prepare for the presentation.
Role	<table><tr><th>Member 1</th><th>Member 2</th><th>Member 3</th></tr><tr><td>Hazeera - make online portfolio in the github</td><td>Ainsyah - push the coding to the github for dijkstra algorithm</td><td>Ainna - push the coding to the github for floyd algorithm</td></tr></table>

2.0 PROBLEM DEFINITION

Problem scenario:

The e-commerce business, which is among the most well-known in a busy metropolis, has seen substantial expansion and has a number of warehouses that are thoughtfully placed across the city to reduce delivery times. However, as the business grew, the logistics of its supply chain complicated, making it challenging to optimise package delivery within the specified time period. Due to variables such as distance and traffic, the corporation now confronts difficulty in assuring timely delivery of items from numerous warehouses to clients. The logistics organisation must choose the best way to optimise its supply chain logistics in order to improve consumer satisfaction, reduce costs, enable scalability, and react to changing conditions. This entails devising a system that takes into consideration all important aspects and limits in order to efficiently transport items while accounting for variables such as warehouse locations, delivery distances, and traffic conditions. The business hopes to increase performance in delivering items on schedule by tackling these optimization difficulties and streamlining its supply chain processes.

The main goals are to ensure:

1. Route Optimization
   Finding the best possible route to deliver packages to all warehouses throughout the city within the given time frame.

2. Cost Reduction
   Minimising travel distance as short as possible and optimising routes to reduce costs associated with transportation, such as fuel consumption.

The implementation of AAD really comes in hand to help calculate the best route for the logistic company. Even the navigation apps like Google Maps, Waze and Uber, all use the AAD implementation in order to find the best route for their consumer. In this scenario, two approaches of AAD can be used which are greedy algorithms; using Dijkstra's algorithm and dynamic programming; using Floyd's algorithm. The best possible path to deliver all the packages with the shortest distance and route is the expected output from both algorithms. It helps to support the decision making on finding which is the optimal route for the logistic company to take in order to achieve the main goals. However, only one algorithm will give the optimal solution for the scenario.

3.0 MODEL DEVELOPMENT

When developing a model to find the shortest path for a logistic company, we need to know data types used and constraints that need to be considered. From these information, the optimal solution on finding the shortest path can be evaluated using either Dijkstra's algorithm or Floyd's algorithm.

Data type:

I. The main dataset considered is the number of locations of warehouses and customer houses (denoted as Node) together with the distance. It is much easier if we sketch the diagram beforehand.

II. To find the shortest path, the total distance of travel from the source node is being added for every direction it goes.

Objective function:

I. The objective function is defined through finding the shortest path among those locations mapped on the graph, and ensuring that every node is visited at least once so all the items arrive safely at customer houses and warehouses. However, it still needs to consider the shortest path that should be taken by the logistic company.

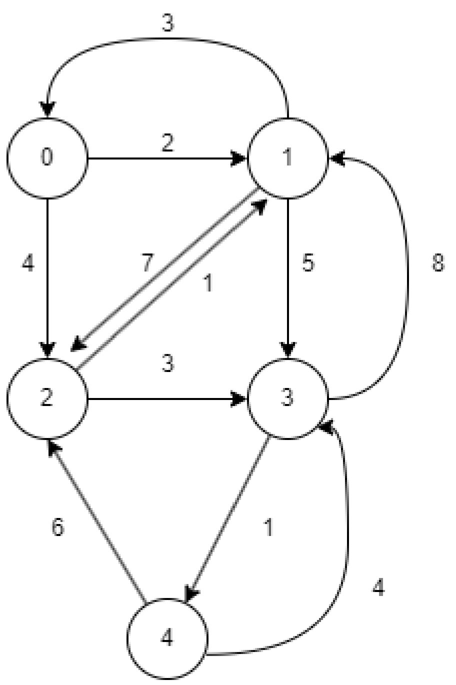


Figure 1 : Graph locating warehouses and customer houses

Distance	: 1 : 1 km
Vertex	: Represents the warehouses and customer houses
Edge	: Road connecting one location to another

As shown on figure 1 above, it is the example of a graph that has been sketched to solve the logistic company supply chain. It includes 5 locations of warehouses and customer houses together with their corresponding distance between the locations.

Constraints:

I. Time constraint: Path taken by the logistic company should be compatible with the time period for them to deliver the items.

II. Space constraint: Logistic companies should know the exact location of warehouses and customer houses. They should know the distance between two different locations and recognize possible paths they might use to go to the location. It can be represented by using directed graphs or undirected graphs. The logistics should only stop at warehouses or customer houses only in a given direction.

III. Value constraint: By using the Dijkstra algorithm, the weight of the edges should only be positive numbers. However, for Floyd's algorithm, it can handle negative values of edge weights.

4.0 SPECIFICATION OF AN ALGORITHM

As to fulfil the goals of finding the optimal solution that ensures the route optimisation and cost reduction for the logistic company, both algorithms; greedy algorithm and dynamic programming have been compared together. Based on the calculation, dynamic programming that implements Floyd's algorithm gives the best optimal solution compared to greedy algorithms that implement Dijkstra's algorithm. This is because Floyd's algorithm finds the shortest path while including all the vertices (visit every vertex at least once). However, Dijkstra's algorithm finds the shortest path from source to destination but may not include all the vertices. It only considers the path that would be the shortest.

For a logistic company, it is important to ensure all the items are delivered safely within the given time period to warehouses or customers houses. The failure to deliver the items will affect the effectiveness and reputation of the company as it does not fulfil consumer needs. Digital business world nowadays has pushed many logistic companies to ensure their services are at the top tier. Thus, it is crucial to ensure all warehouses or customer houses (vertices) are being visited at least once to make sure all items are delivered correctly.

Characteristics	**Dijkstra's Algorithm**	**Floyd's Algorithm**
Source	Single source	All sources
Optimization	Optimised for finding the shortest path **between a single source vertex and all other vertices**	Optimised for finding the shortest path **between all pairs** of vertices
Edge weight	Only works for edge weight with non-negative value	Can works for all non-negative and negative edge weight
Time complexity	O(n^2)	O(n^3)
Types of graph	Works for both directed and undirected graph	

The execution time of Floyd's algorithm is much longer than Dijkstra's algorithm. It also uses more memory usage as the process of Floyd's algorithm uses all vertex information while Dijkstra's algorithm does not necessarily use all vertex information. However, Floyd's algorithm is the most suitable algorithm to determine the best possible routes for the logistic company as it finds the shortest path between all vertices. It gives the optimal solution for the scenario.

5.0 DESIGNING AN ALGORITHM

Pseudocode for Dijkstra’s Algorithm:

```
function Dijkstra(Graph, source):

    for each vertex v in Graph.Vertices:
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        add v to Q
    dist[source] ← 0

    while Q is not empty:
        u ← vertex in Q with min dist[u]
        remove u from Q

        for each neighbour v of u still in Q:
            alt ← dist[u] + Graph.Edges(u, v)
            if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u


    return dist[], prev[]
```

Pseudocode for Floyd’s Algorithm:

```
// let A be a n by n adjacency matrix
for k = 0 to n -1
   for i = 0 to n -1
      for j = 0 to n -1
         if ( A [i , j ] > A[i , k ] + A [k , j ] )
              A [i , j ] = A [i , k ] + A [k , j ];
         end if
      end for
   end for
 end for
```

6.0 CHECKING THE CORRECTNESS OF AN ALGORITHM

In order to ensure the algorithm used gives the right output, the algorithm should provide the correct answer and it should terminate properly. By means, it should not terminate because of looping or error of coding. On the other hand, the algorithm also should not keep on looping without an end.

Initialization:

1. Initialise 2D arrays to store distances from each vertex called graph[][].
2. Declare infinity value, INF to 9999.
3. Create an array that stores the optimal path called optimalPath[].

Maintenance:

1. Find the optimalPath[] from the graph[][].
    - Initialize dist[][] matrix with the values from the graph
    - Compute all-pairs shortest path using Floyd-Warshall algorithm
    - Compute all-pairs shortest path using Floyd-Warshall algorithm

2. Calculate the total distance using the loop function at index of graph[path[i]][path[i+1]] where path is the array of the optimalPath and graph is the array that stores the distances between vertices.

Termination:

1. Print the path taken together with the total distance from the source vertex.

By ensuring these steps run the way it tends to operate, it helps to ensure the correctness of the algorithm and terminates only when the optimal path is found.

7.0 ANALYSIS OF AN ALGORITHM

The Floyd-Warshall algorithm finds the shortest path between all pairs of vertices in a weighted directed graph by calculating the shortest path between any two of them. In this operation, the shortest paths between each pair of vertices are computed as a shortest path matrix. An analysis of Floyd's algorithm examines the best-case, worst-case, and average-case complexity of the algorithm.

Worst-case Analysis:
The worst case scenario occurs when there are many vertices and edges in the graph, and the algorithm must update the distance matrix for every pair of vertices. All pairs of vertices in the graph, as well as possible intermediate vertices, must be iterated through three nested loops. As it iterates, the algorithm determines whether the path between the two intermediate vertices produces a shorter path than the direct path. The worst case has a time complexity of $O(n^3)$, where n is the number of vertices. Every vertex is iterated n times, resulting in a total of n^3 iterations.

Best-case Analysis:
There are two scenarios that are the best: the graph has a very few vertex points or it's already a complete graph with all distances known. Due to the fact that the distance matrix is already known, the algorithm can terminate early without updating it. In order to check for updates, the algorithm must still perform the initialization step and the nested loop, but no actual updates are made. Best case time complexity is $O(n^3)$, where n is the number of vertices. During each iteration, the algorithm goes through all possible pairs of vertices.

Average-case Analysis:
Floyd's algorithm also has an average case time complexity of $O(n^3)$, assuming a random distribution of edge weights. There is no definite pattern or structure in the graph, so it assumes there is none. Through the comparison of paths between intermediate vertices, the algorithm determines the distance matrix and updates it. With the same number of iterations as in the worst case, this results in the same time complexity of $O(n^3)$.

In terms of space complexity, the algorithm requires $O(n^2)$ space to store the distance matrix. The shortest path distances between all pairs of vertices are stored as a matrix of size n by n. In spite of Floyd's algorithm's simplicity and guaranteed shortest paths between pairs of vertices, it has a cubic time complexity. Thus, large graphs with many vertices become less efficient. In such cases, alternative algorithms like Dijkstra's algorithm or the A* algorithm are preferred for finding single-source shortest paths or optimizing for specific search criteria.

8.0 IMPLEMENTATION OF AN ALGORITHM

By Using Dijsktra's Algorithm:

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.PriorityQueue;

class Node implements Comparable<Node> {
   int id;
   int distance;

   public Node(int id, int distance) {
      this.id = id;
      this.distance = distance;
   }

   public int compareTo(Node other) {
      return Integer.compare(this.distance, other.distance);
   }
}

public class DijkstraAlgorithm {
   private int[][] graph;
   private int numNodes;

   public DijkstraAlgorithm(int[][] graph) {
      this.graph = graph;
      this.numNodes = graph.length;
   }

   public void findOptimalPath() {
      boolean[] visited = new boolean[numNodes];
      int[] distance = new int[numNodes];
      int[] previous = new int[numNodes];
      Arrays.fill(distance, Integer.MAX_VALUE);
      Arrays.fill(previous, -1);

      distance[0] = 0;

      PriorityQueue<Node> queue = new PriorityQueue<>();
      queue.add(new Node(0, 0));

      while (!queue.isEmpty()) {
         Node current = queue.poll();
         int currentNode = current.id;

         if (visited[currentNode]) {
            continue;
         }
```

```
            visited[currentNode] = true;

            for (int neighbor = 0; neighbor < numNodes; neighbor++) {
                int edgeWeight = graph[currentNode][neighbor];

                if (edgeWeight > 0 && !visited[neighbor]) {
                    int newDistance = distance[currentNode] + edgeWeight;

                    if (newDistance < distance[neighbor]) {
                        distance[neighbor] = newDistance;
                        previous[neighbor] = currentNode;
                        queue.add(new Node(neighbor, newDistance));
                    }
                }
            }
        }

        // Print the optimal path
        List<Integer> path = new ArrayList<>();
        int currentNode = numNodes - 1;
        while (currentNode != -1) {
            path.add(0, currentNode);
            currentNode = previous[currentNode];
        }

        System.out.println("Optimal Path: " + path);

        // Calculate total distance
        int totalDistance = distance[numNodes - 1];
        System.out.println("Total Distance: " + totalDistance);
    }

    public static void main(String[] args) {
            int[][] graph = {

                {0, 2, 4, 0, 0},
                {3, 0, 7, 5, 0},
                {0, 1, 0, 3, 0},
                {0, 8, 0, 0, 1},
                {0, 0, 6, 4, 0}
            };

        DijkstraAlgorithm dijkstra = new DijkstraAlgorithm(graph);
        dijkstra.findOptimalPath();
    }
}
```

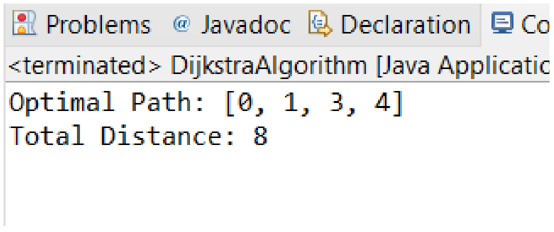
By using Floyd’s Algorithm:

```
import java.util.Arrays;

public class FloydAlgorithm {
   private static final int INF = Integer.MAX_VALUE, nV = 4;

   public static void main(String[] args) {
         int[][] graph = {
            {0, 2, 4, INF, INF},
            {3, 0, 7, 5, INF},
            {INF, 1, 0, 3, INF},
            {INF, 8, INF, 0, 1},
            {INF, INF, 6, 4, 0}
         };

      int[] optimalPath = findOptimalPath(graph);
      int totalDistance = calculateTotalDistance(graph, optimalPath);

      System.out.println("Optimal Path: " + Arrays.toString(optimalPath));
      System.out.println("Total Distance: " + totalDistance);
   }

   private static int[] findOptimalPath(int[][] graph) {
      int n = graph.length;
      int[][] dist = new int[n][n];

      // Initialize dist[][] matrix with the values from the graph
      for (int i = 0; i < n; i++) {
         System.arraycopy(graph[i], 0, dist[i], 0, n);
      }

      // Compute all-pairs shortest path using Floyd-Warshall algorithm
      for (int k = 0; k < n; k++) {
         for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
               if (dist[i][k] != INF && dist[k][j] != INF && dist[i][k] + dist[k][j] < dist[i][j]) {
                  dist[i][j] = dist[i][k] + dist[k][j];
               }
            }
         }
      }
      printMatrix(dist);

      // Find the optimal path starting from node 0
      boolean[] visited = new boolean[n];
      int[] path = new int[n];
      int pathIndex = 0;
      visited[0] = true;
      path[pathIndex++] = 0;
```

```
        for (int i = 1; i < n; i++) {
            int nextNode = -1;
            int minDistance = INF;

            for (int j = 0; j < n; j++) {
                if (!visited[j] && dist[path[pathIndex - 1]][j] < minDistance) {
                    nextNode = j;
                    minDistance = dist[path[pathIndex - 1]][j];
                }
            }

            visited[nextNode] = true;
            path[pathIndex++] = nextNode;
        }

        return path;
    }

    private static int calculateTotalDistance(int[][] graph, int[] path) {
        int totalDistance = 0;
        for (int i = 0; i < path.length - 1; i++) {
            totalDistance += graph[path[i]][path[i + 1]];
        }
        return totalDistance;
    }
    static void printMatrix(int matrix[][]) {
        for (int i = 0; i < nV; ++i) {
          for (int j = 0; j < nV; ++j) {
            if (matrix[i][j] == INF)
              System.out.print("INF ");
            else
              System.out.print(matrix[i][j] + "  ");
          }
          System.out.println();
        }
      }
}
```

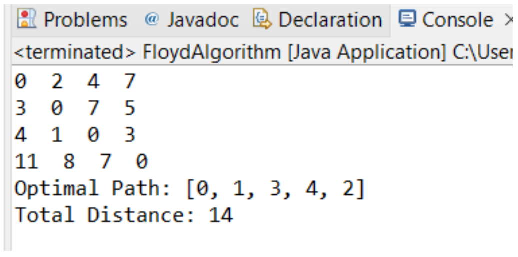
9.0 PROGRAM TESTING

Output of Dijkstra’s Algorithm:



```
<terminated> DijkstraAlgorithm [Java Applicatic
Optimal Path: [0, 1, 3, 4]
Total Distance: 8
```

Output of Floyd’s Algorithm:



```
<terminated> FloydAlgorithm [Java Application] C:\User
0  2  4  7
3  0  7  5
4  1  0  3
11  8  7  0
Optimal Path: [0, 1, 3, 4, 2]
Total Distance: 14
```