Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Spring, Year: 2023), B.Sc. in CSE (Day)

Delivery Rider

Course Title: Algorithms Lab
Course Code: CSE 206
Section: PC CSE 213 - DA

Students Details

Name	**ID**
Md. Kawsar Ahamed	213902013

significant Submission Date: 20 June 2023
Course Teacher's Name: Md. Sultanul Islam Ovi

[For teachers use only: Don't write anything inside this box]

Lab Project Status

Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

1.1 Overview

It is an application that will guide the delivery rider in a way that minimizes cost, time, and distance.

1.2 Motivation

In the context of a delivery system, the motivation behind using the Dijkstra algorithm is to determine the most efficient path for delivering goods or services from one location to another. The algorithm considers factors such as distance, time, or cost to identify the optimal route.

1.3 Problem Definition

1.3.1 Logistics and Routing

The primary objective of the delivery system is to optimize the routes between the source and destination locations. The Dijkstra algorithm is utilized to find the shortest or most efficient path(s) in terms of time, distance, or cost. The system needs to consider various constraints and preferences, such as traffic congestion, delivery priorities, delivery windows, or specific customer requirements.

1.3.2 Customer Experience

In a Dijkstra algorithm-based delivery system, customer experience is paramount. The system optimizes routes and manages resources to ensure timely deliveries, provides transparency and communication through real-time tracking and updates, offers flexibility and customization options, maintains accuracy and reliability in order fulfillment, offers responsive customer support, delivers a seamless user interface, and gathers postdelivery feedback. By prioritizing these aspects, the system aims to provide a positive and satisfactory customer experience, fostering loyalty and a strong reputation.

1.3.3 Cost Handling

In a Dijkstra algorithm-based delivery system, cost handling is a critical aspect of the problem domain. The system focuses on optimizing routes and resource allocation to minimize transportation costs and improve operational efficiency. It incorporates cost analysis and decision-making processes, promotes cost transparency in customer communication, monitors and optimizes expenses, and considers supplier/vendor cost management. By effectively managing costs, the system aims to enhance profitability, sustainability, and customer value.

1.4 Design Goals/Objectives

A Dijkstra algorithm-based delivery system has two primary objectives: route optimization and timeliness. It aims to find the shortest or most efficient routes between source and destination locations using the Dijkstra algorithm, minimizing distances and travel times. By optimizing routes, the system reduces transportation costs and improves operational efficiency. Additionally, the system focuses on meeting delivery deadlines and time windows, ensuring timely deliveries to enhance customer satisfaction and build trust in the delivery service. It strives to provide accurate delivery estimates, minimize delays, and maintain clear communication with customers throughout the delivery process.

Another set of objectives includes resource optimization, scalability, customer experience enhancement, and cost optimization. The system aims to optimize the allocation and utilization of delivery resources, such as vehicles and drivers, to maximize efficiency and minimize costs. It is designed to scale efficiently, handling increasing volumes of deliveries and locations while maintaining high performance. The system focuses on providing a positive customer experience by offering transparency, real-time tracking, and personalized service. It also seeks to optimize costs associated with fuel consumption, vehicle maintenance, and operational expenses to improve profitability and financial sustainability. By achieving these objectives, the Dijkstra algorithm-based delivery system strives to provide efficient, reliable, and cost-effective delivery services to meet customer expectations and drive business success.

1.5 Application

A Dijkstra algorithm-based delivery system finds application in various industries and scenarios where efficient route optimization and timely delivery are critical.

1.5.1 E-commerce and Retail

Delivery systems used by e-commerce platforms and retail companies can leverage the Dijkstra algorithm to optimize routes for last-mile deliveries. By calculating the shortest paths between warehouses, distribution centers, and customer locations, the system ensures the timely delivery of orders while minimizing transportation costs.

1.5.2 Food Delivery Services

Delivery platforms for food and meal delivery services can benefit from the Dijkstra algorithm to optimize routes for delivering orders to customers. By finding the shortest or fastest paths between restaurants and delivery locations, the system can enhance delivery speed and ensure fresh and hot food reaches customers promptly.

1.5.3 Courier and Package Delivery

Courier companies and package delivery services can employ the Dijkstra algorithm to optimize their delivery networks. The algorithm helps in determining the most efficient routes for package pickup and drop-off, considering factors like distance, traffic conditions, and delivery priorities.

Overall, the application of a Dijkstra algorithm-based delivery system is diverse and spans industries such as e-commerce, retail, logistics, healthcare, food delivery, courier services, waste management, and public transportation. Optimizing routes enhances delivery efficiency, reduces costs, and improves overall customer satisfaction.

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

A Dijkstra algorithm-based delivery system focuses on finding the shortest or most efficient paths between source and destination locations. By considering factors like distance, time, or cost, the system calculates optimal routes that minimize transportation expenses, reduce delivery times, and enhance operational efficiency.

This system finds applications in e-commerce, retail, logistics, food delivery, health-care, and other sectors where timely and cost-effective deliveries are paramount. It helps companies streamline their delivery operations, meet customer expectations, and ultimately gain a competitive edge in the market.

2.2 Project Details

The Dijkstra algorithm-based delivery system aims to develop a robust and efficient solution for optimizing delivery routes and ensuring timely and cost-effective deliveries. The system will utilize the Dijkstra algorithm, a well-established graph algorithm, to calculate the shortest or most efficient paths between source and destination locations. The project will focus on designing and implementing a comprehensive system that encompasses route optimization.

2.2.1 User Interface

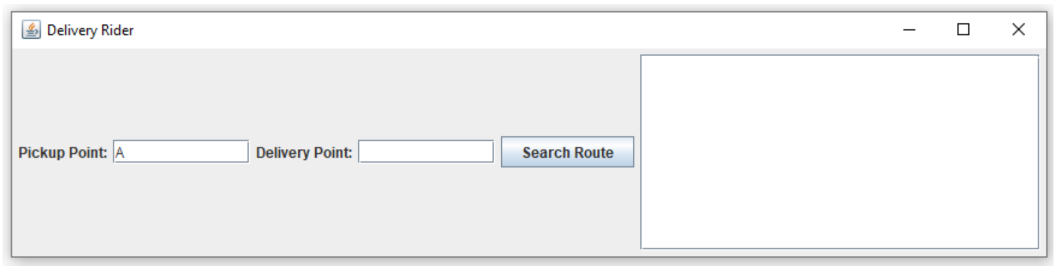


Figure 2.1: User Interface

2.3 Implementation

2.3.1 Flow Chart

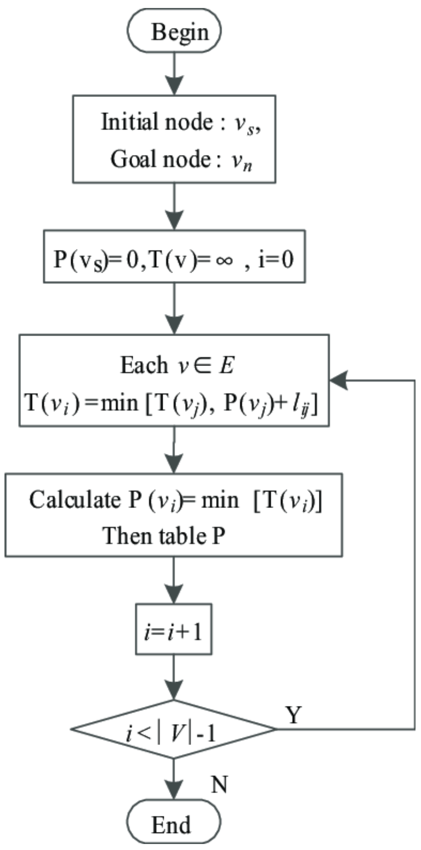

Begin
Initial node : v_s,
Goal node : v_n
$P(v_S)=0, T(v)=\infty$, i=0
Each $v \in E$
$T(v_i) = \min [T(v_j), P(v_j)+l_{ij}]$
Calculate $P(v_i) = \min [T(v_i)]$
Then table P
$i=i+1$
$i<|V|-1$
Y
N
End

Figure 2.2: Flow Chart

2.3.2 Tools and libraries

Development Environment

Java Development Kit (JDK): Provides the necessary tools and libraries for Java development.
Integrated Development Environment (IDE): Popular IDEs for Java development include Eclipse, IntelliJ IDEA, and NetBeans. These IDEs offer features like code completion, debugging, and project management.

Build and Dependency Management

Apache Maven: A build automation tool that manages project dependencies, compiles code, and generates the project's final artifact (e.g., JAR file). It can also help with managing external libraries and their versions.

2.4 Algorithms

- Create the Dijkstra method: Initialize distances and visited arrays.
  Set the distance of the source node to 0.
  Iterate NUMNODES - 1 time.
  Find the minimum distance vertex (minDistance).
  Mark the vertex as visited.
  Update the distances of neighboring vertices if a shorter path is found.

- Create the minDistance method
  Initialize min and minIndex variables.
  Iterate through all vertices.
  Find the vertex with the minimum distance that has not been visited.

- Create the reconstructPath method:
  Create a stack to store the path.
  Start from the destination node and traverse back to the source node.
  Push the nodes onto the stack.
  Pop the nodes from the stack to get the correct order of the path.
  Build a string representation of the path.

- Create the findPreviousNode method:
  Iterate through all nodes.
  Find the node that contributes to the current node's shortest path.

- Create the getNodeIndex method:
  Iterate through all nodes.
  Find the index of the given node name.

- Create the main method:
  Create an instance of DeliveryRider.
  Make the frame visible.

Chapter 3

Performance Evaluation

3.1 Simulation Environment/ Simulation Procedure

The experimental setup for simulating a Dijkstra algorithm-based delivery project in Java involves installing the Java Development Kit (JDK) and an Integrated Development Environment (IDE). The project setup includes creating a new Java project and managing external libraries. If using the JGraphT library, it can be installed by downloading the JAR file and adding it to the project's classpath. Testing and verification ensure that the user interface is displayed correctly and the Dijkstra algorithm functions as expected. Following these steps provides the necessary environment to simulate the delivery project and evaluate its performance.

3.2 Results Analysis/Testing

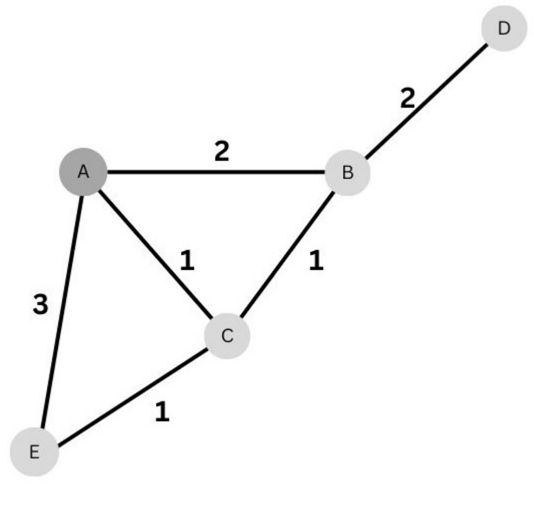
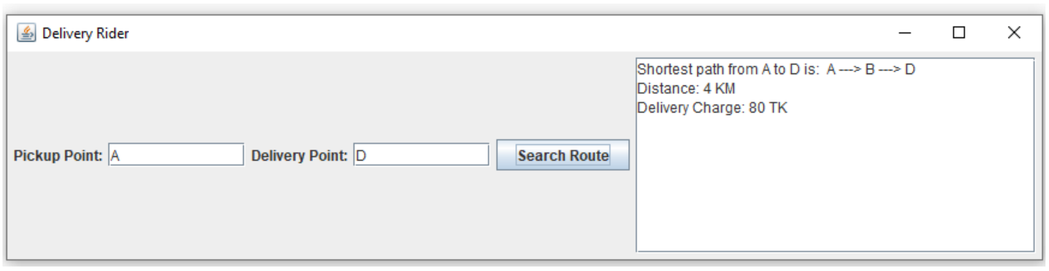


Figure 3.1: The Delivery Map

3.2.1 Result_portion_1

When the pickup point is A and the delivery point is D.

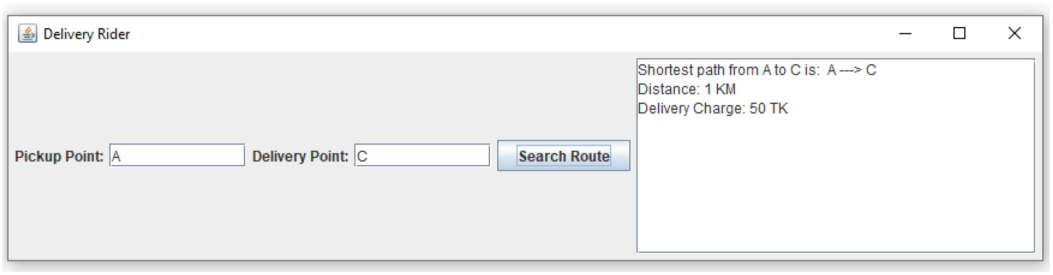


```
Shortest path from A to D is:  A ---> B ---> D
Distance: 4 KM
Delivery Charge: 80 TK
```

Figure 3.2: Destination A to D

3.2.2 Result_portion_2

When the pickup point is A and the delivery point is C. Here delivery charge is the minimum cause if the delivery charge is less than 50 TK then it will charge 50 TK.

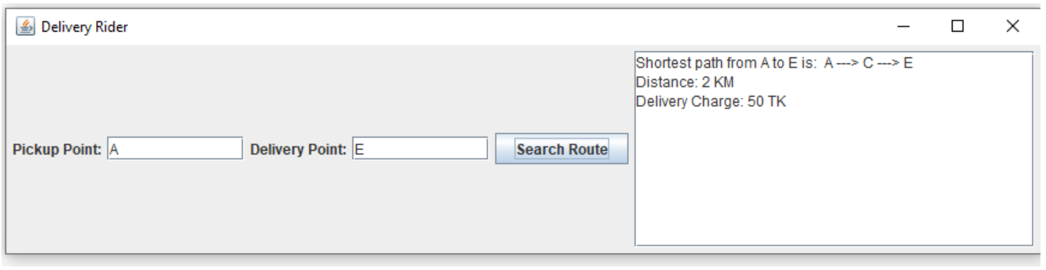


```
Shortest path from A to C is:  A ---> C
Distance: 1 KM
Delivery Charge: 50 TK
```

Figure 3.3: Destination A to C

3.2.3 Result_portion_3

When the pickup point is A and the delivery point is E.



```
Shortest path from A to E is:  A ---> C ---> E
Distance: 2 KM
Delivery Charge: 50 TK
```

Figure 3.4: Destination A to E

3.3 Results Overall Discussion

The output of the provided code is a graphical user interface (GUI) application for a Delivery Rider system. The application allows the user to input a pickup point and a delivery point. Upon clicking the "Search Route" button, the Dijkstra algorithm is used to find the shortest path between the two points based on a predefined graph.

The output displays the shortest path, the distance in kilometers, and the delivery charge in TK. If there is no route found between the specified points, an appropriate message is displayed. The GUI provides a user-friendly way to interact with the application and obtain the necessary information for delivery routing.

Chapter 4

Conclusion

4.1 Discussion

The provided code implements a Delivery Rider application using Java and Swing, which enables users to find the shortest route and calculate the delivery charge between two points in a predefined delivery network.

The code begins by defining constants, such as the number of nodes in the network and their corresponding names, along with the adjacency matrix representation of the graph. The code then sets up the GUI components, including text fields for inputting the pickup and delivery points, a button to initiate the search for the shortest route, and a text area to display the result.

When the "Search Route" button is clicked, the code retrieves the inputted pickup and delivery points, validates them, and proceeds with the Dijkstra algorithm to find the shortest path between the two points. It updates the result area with the shortest path, distance in kilometers, and the calculated delivery charge based on the distance.

The Dijkstra algorithm implementation utilizes an array to store distances, a boolean array to track visited nodes, and nested loops to iterate over the nodes and update the distances based on the edges in the graph. The algorithm repeats until all nodes are visited, ensuring the shortest path is found.

The code also includes utility methods to reconstruct the shortest path, find the previous node in the path, and retrieve the index of a given node name.

Overall, this code provides a functional GUI-based Delivery Rider application using the Dijkstra algorithm to calculate the shortest route and delivery charge. It showcases the use of Java, Swing for the GUI, and algorithmic logic to solve the delivery routing problem efficiently.

4.2 Limitations

The Delivery Rider application has several limitations. It utilizes a fixed adjacency matrix for graph representation, which may not be suitable for larger networks due to its space complexity. The code lacks comprehensive input validation, potentially leading to errors if invalid or nonexistent locations are entered. Error handling is minimal, providing basic error messages without detailed information. The application does not support dynamic graph modifications, limiting its adaptability to changing delivery networks. Lastly, there is no visual representation of the optimal path, which could impede user understanding and visualization of the route. Addressing these limitations would enhance the application's scalability, robustness, user-friendliness, and adaptability.

4.3 Scope of Future Work

In the future work for the Delivery Rider application includes dynamic graph updates to accommodate changes in the delivery network, exploring alternative routing algorithms, integrating real-time data sources for more accurate route recommendations, improving the user interface with visual representations of the network and optimal path, incorporating user-profiles and preferences for personalized delivery routes, integrating external services for enhanced location validation and geolocation-based services, optimizing the code and algorithm for scalability, and implementing advanced error handling mechanisms. These improvements would enhance the application's functionality, scalability, user experience, and adaptability to real-world delivery scenarios.