NETWORKING ALGORITHM SIMULATION

2022-2023
A PROJECT REPORT

Submitted by

NAGASHREE	**201231522197**
SHRIDEVI JOGI	**201231522233**
SURAKSHA R	**201231522296**

In partial fulfillment for the award of the degree of

BACHELOR OF COMPUTER APPLICATIONS

Under the guidance of
Mrs. Amratha. C
Lecturer
Department of Computer Science

BHANDARKARS' ARTS AND SCIENCE COLLEGE

KUNDAPURA

MANGALORE UNIVERSITY

BHANDARKARS' ARTS AND SCIENCE COLLEGE

KUNDAPURA -576201

Department of Computer Science

Certificate

Certified that the project work entitled

……..NETWORKING ALGORITHM SIMULATION……..……
is a bonafied work carried out by

…….. NAGASHREE, SHRIDEVI JOGI, SURAKSHA R……..……
in partial fulfilment for the award of degree of Bachelor of Computer Applications of the Mangalore University during the year 2022-23. The project report has been approved as it satisfies the academic requirements of project work prescribed for the Bachelor of Computer Applications.

Project Guide

Head of the Department
Department of Computer Science

Name of the Student:	NAGASHREE, SHRIDEVI JOGI, SURAKSHA R
Roll Number:	200598,200606,200625
University Register Number:	201231522197,201231522233,201231522296

Examiners: 1. **2.**

BHANDARKARS' ARTS AND SCIENCE COLLEGE

KUNDAPURA-576201

ATTENDANCE CERTIFICATE

This is to certify that the following students of sixth semester BCA has got adequate attendance in the project work as stipulated by Mangalore University in BCA regulations.

NAGASHREE	:200598
SHRIDEVI JOGI	:200606
SURAKSHA	:200625

PRINCIPAL

DECLARATION

We hereby declare that this project work entitled "NETWORKING ALGORITHM SIMULAION" has been prepared by us during the year 2022–23 under the guidance of Mrs. AMRATHA C, Department of Computer Science, Bhandarkars' Arts and Science College, Kundapura in the partial fulfillment of BCA degree prescribed by the Mangalore University.

We also declare that this project is the outcome of our own effort, that it has not been submitted to any other university for the award of any degree.

Date:

NAGASHREE :200598
SHRIDEVI JOGI:200606
SURAKSHA R :200625

ACKNOWLEDGEMENT

It gives us immense pleasure to present report on "**NETWORKING ALGORITHM SIMULATION".**

Our sincere thanks to **Mrs. Amratha C** our project guide who helped us in developing this project.

We would like to express our gratitude to Mrs. Vijayalakshmi N Shetty, **Head of the Department, Computer Science** for her kind concern and encouragement during the completion of our project.

We are sincerely thankful to Dr.NP. Narayan Shetty, Principal of Bhandarkar's Arts and Science College Kundapura, for granting an opportunity to work our project.

Our sincere thanks for all faculty members of Computer Science Department. We are thankful to our parents for their encouragement towards the project. Last but not the least, we whole heartly appreciates the co-operation of our friends.

Thank You,

Project Team,

Nagashree

Shridevi Jogi

Suraksha R

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

1. Introduction

1.1 Introduction of the System

1.1.1 Project title:

Networking Algorithm Simulation

1.1.2 Category:

Standalone Application

1.1.3 Overview:

Network algorithm simulation is a technique used to evaluate performance of networking algorithms and protocols under various conditions and scenarios. The simulation involves creating a model of the network and simulating the behaviour of network and algorithms being tested.

1.2 Background

1.2.1 Introduction of the Company:

Not applicable.

1.2.2 Brief note on existing system:

The Network Algorithm Simulation is compared with manual system. As it overcomes the method of explaining it on the board by the lectures. It makes us to clearly understand working of algorithm through simulation and visualization, as it enhances the time management and users can encatch the topics easily and gain knowledge about it.

1.3 Objective of the System

- Objective are the pre-determined goals or the outcome of the system process.
- The main objective of networking algorithms simulation is to evaluate the performance of different network protocols and algorithms under various conditions and scenarios.
- Network Algorithms are designed to optimize network performance such as reducing latency, increasing throughput, improving readability, and ensuring fairness.
- Network Algorithm simulation is a widely used technique to study networking algorithms and protocols.
- Network Algorithm provides a controlled environment to evaluate their performance without disrupting real-world networks.
- Network Algorithm simulation can be used to evaluate how different routing algorithms perform under different traffic loads, topologies and fairness scenarios.
- Network Algorithm simulation can help network designer and researchers understand strength and weakness of different network protocol and algorithms.
- Network Algorithm simulation helps the researches in identifying potential issues and make improvements to network design before implementing it into real-world.
- Network Algorithm simulation help researchers understand impact of different network characters and conditions on performance of networking algorithms, leading to robust and efficient network designs.

1.4 Scope of the System

Scope is the limitation that processes faces from beginning to end.

- We faced exceptions while drawing the graphs using turtle.
- We faced many challenges while visualizing or simulating the graph.

1.5 Structure of the System

1.5.1 Parity Check

1.5.1.1 Simple Parity Check

Simple parity check is a method of error detection used in data transmission and storage. Here the parity bit is calculated based on the binary string entered by the user. Output generated is in the form of graphical representation of binary string with blocks.

1.5.1.2 Two-Dimensional Parity Check

Two-Dimensional parity check is a method used to detect errors in data transmission and storage. It involves adding parity bit to Two-Dimensional data bits to detect errors. Output generated is in the form of graphical representation of matrix.

1.5.2 Cyclic Redundancy Check

Cyclic Redundancy Check is an error detecting algorithm based on polynomial division. It takes data as a binary bit value and performs division operation using polynomial. The remainder of the division which is appended to the binary bit value is the CRC value.

1.5.3 Checksum

Checksum is a value calculated by summing data bits using XOR operation and then taking 1's complement of the sum.

1.5.4 A Star Algorithm

A Star is an informed search algorithm that efficiently determines the shortest path between two nodes in a graph and it uses heuristic function to estimate distance from start node to goal node.

1.5.5 Hamming Code

Hamming code is an error correcting code used in digital communication and data storage. Process starts by detecting error in the parity bits by check-one

skip-one method for P1 parity and so on. If number of ones in the parity bit is odd then error is detected in the received code, otherwise no error. Then to find the error position in the received code using XOR operation.

1.5.6 Path Finding Algorithm(Greedy Kruskal)

Path finding algorithm based on Greedy Kruskal algorithm is used to find path between source node and destination node.

1.5.7 Shortest Path First (Dijkstra's)

Shortest Path First is a routing algorithm used in computer networks to find the shortest path between two nodes in the network.

1.5.8 Open Shortest Path First

Open Shortest Path First is a link state routing protocol commonly used in IP networks to determine the shortest path and exchange routing information.

1.5.9 Minimum Spanning Tree(Greedy Kruskal)

Minimum Spanning Tree iterates through all edges in increasing order of their weight and displays the edges without creating circle.

1.5.10 Flooding Routing Algorithm

Flooding Routing Algorithm is a routing algorithm used in the computer network. When the node receives the data, it forwards the data to all its connected neighbor except the neighbors from which it received the data.

1.5.11 Distance Vector Routing Algorithm

Distance Vector Routing algorithm is a type of routing algorithm used in computer networks to determine the best path to transmit the data between nodes and calculates the distance.

1.6 System Architecture

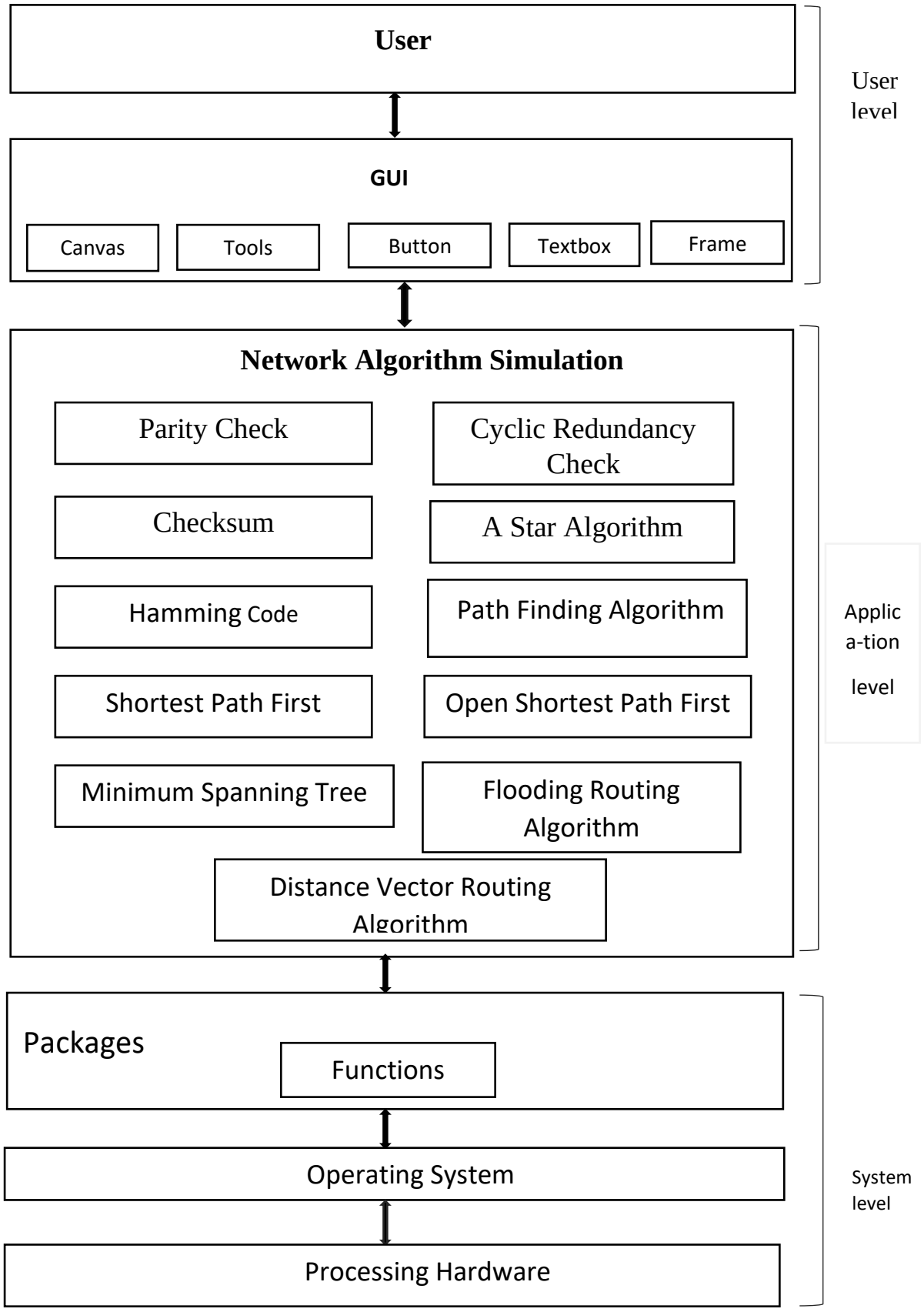


Figure 1.1 System Architecture

1.7 End User

- In this application the end users are all Network designers, students, staff.

1.8 Software/Hardware need for the development

- **OS:** Windows 7 or higher Recommended: Windows 10.
- **CPU:** Intel or AMD processor with 64 bit support; Recommended: 2.8 GHz or faster processor.
- **Disk Storage:** 4 GB of free disk space or higher.
- **Language use:** Python
- **Application:** Python

1.9 Software/Hardware need for the Implementation

- Computer or Laptop
- **OS:** Windows 7 or higher. Recommended: Windows 10.
- **CPU:** Intel or AMD processor with 64 bit support; Recommended: 2.8 GHz or faster processor.
- **Disk Storage:** 4 GB free of disk space or higher.

SOFTWARE REQUIREMENT SPECIFICATION

2. Software Requirement Specification

2.1 Introduction

- SRS describe the completely an external behaviour of proposed software. The basic goal of the requirement specification document enlists enough and necessary requirement that are required for the project development.
- The purpose of the software requirement specification document to provide a detail overview of our software product its parameter and goal. SRS bridge the communication gap between client and developer. It is basic agreement between client and developer.
- Software requirement specification provides a reference for validation for the final product. The software development using software specification provided the better graphical representation.

2.2 Overall Description

2.2.1 Product Perspective

2.2.1.1 System Interface

This application runs in the latest version of Python 3.11.1.

2.2.1.2 User Interface

This application provide canvas, frame, button, textbox, allowing easy control by a keyboard and mouse.

2.2.1.3 Hardware Interface

Not Applicable.

2.2.1.4 Software Interface

This application allows Python 3.11.1 and Windows 7 or higher. Recommended: Windows 10.

2.2.1.5 Communication Interface

Not applicable.

2.2.1.6 Interface with Server

Not applicable.

2.2.2 Product Function

Specify the general function of this application.

- Simple Parity Check: Simple parity check is a method of error detection used in data transmission and storage. Here the parity bit is calculated based on the binary string entered by the user. Output generated is in the form of graphical representation of binary string with blocks.
- Two-Dimensional Parity Check: Two-Dimensional parity check is a method used to detect errors in data transmission and storage. It involves adding parity bit to Two-Dimensional data bits to detect errors. Output generated is in the form of graphical representation of matrix.
- Cyclic Redundancy Check: Cyclic Redundancy Check is an error detecting algorithm based on polynomial division. It takes data as a binary bit value and performs division operation using polynomial. The remainder of the division which is appended to the binary bit value is the CRC value.
- Checksum: Checksum is a value calculated by summing data bits using XOR operation and then taking 1's complement of the sum.
- A Star Algorithm: A Star is an informed search algorithm that efficiently determines the shortest path between two nodes in a graph and it uses heuristic function to estimate distance from start node to goal node.
- Hamming Code: Hamming code is an error correcting code used in digital communication and data storage. Process starts by detecting error in the parity bits by check-one skip-one method for P1 parity and so on. If number of ones in the parity bit is odd then error is detected in the received code, otherwise no error. Then to find the error position in the received code using XOR operation.

- Path Finding Algorithm (Greedy Kruskal): Path finding algorithm based on greedy algorithm is used to find path between source node and destination node.
- Shortest Path First (Dijkstra's): Shortest Path First is a routing algorithm used in computer networks to find the shortest path between two nodes in the network.
- Open Shortest Path First: Open Shortest Path First is a link state routing protocol commonly used in IP networks to determine the shortest path and exchange routing information.
- Minimum Spanning Tree (Greedy Kruskal): Minimum Spanning Tree iterates through all edges in increasing order of their weight and displays the edges without creating circle.
- Flooding Routing Algorithm: Flooding Routing Algorithm is a routing algorithm used in the computer network. When the node receives the data, it forwards the data to all its connected neighbor except the neighbors from which it received the data.
- Distance Vector Routing Algorithm: Distance Vector Routing algorithm is a type of routing algorithm used in computer networks to determine the best path to transmit the data between nodes and calculates the distance.

2.2.3 User Characteristics

End users are network designers, students and staff who use this product should have the knowledge of computer that how to use keyboard and mouse.

2.2 4 General Constraints

The above applications required any python software.

- General constraint describes how the product operates inside various circumstances and limit the options designers have if building the product.

- For colour identification set the limited range of colour.
- Resolution: 160*900

2.2.5 Assumption and Dependencies

These factors are not design constraint on the software but any changes to these factors can affect the requirement in the SRS.

2.3 Special Requirement

Not applicable.

2.4 Functional Requirement

2.4.1.1 Simple Parity Check

a) Input: Sequence of binary string.

b) Process: Process calculates the parity bit by counting number of 1's and draws coloured blocks representing bit value, green for 1 and red for 0. The parity bit draws blue block if the parity bit value is 1, otherwise white block is drawn.

c) Output: Graphical representation of binary string with blocks.

2.4.1.2 Two-Dimensional Parity Bit

a) **Input:** Number of rows, columns and data bits.

b) **Process:** Process calculates the row parity and column parity. If number of 1's in the row and column is odd, the row parity and column parity value is 1, otherwise 0.Overall parity is generated based on the sum of the row parity and column parity.

c) **Output:** New 2d list representing data matrix with parity bits.

2.4.2 Cyclic Redundancy Check

a) **Input:** Sequence of binary string and polynomial.

b) **Process:** Process starts with appending 0's to binary bit message equal to the degree of polynomial. Performs bitwise XOR of binary bit message with polynomial, calculates the remainder (CRC) of final message. Remainder is appended to the original message which gives us the CRC value.

c) **Output:** Calculated CRC (remainder) and message with the CRC value added.

2.4.3 Checksum

a) **Input:** Sequence of binary string and blocks.

b) **Process:** Checksum value is calculated by making the one's complement of the results obtained by XOR operation of the given input message.

c) **Output:** It returns the Checksum i.e, one's complement of result.

2.4.4 A Star Algorithm

a) **Input:** Nodes, Edges.

b) **Process:** The heuristic function takes a node as input and returns a value representing the estimated cost (heuristic) from that node to the goal node and determines the valid neighbours of a given node.

c) **Output:** Optimal path from the start node to goal node.

2.4.5 Hamming Code

a) **Input:** 7-bit hamming code.

b) **Process:** Process starts by detecting error in the parity bits by check-one skip-one method for P1 parity and so on. If number of ones in the parity bit is

odd then error is detected in thereceived code, otherwise no error. Then to find the error position in the received code using XOR operation.

c) **Output:** Displays the position of the error in the received code if error is detected.

2.4.6 Path Finding Algorithm

a) **Input:** Nodes, edges.

b) **Process:** Finds path between source node and destination node.

c) **Output:** Path between the source node and destination node.

2.4.7 Shortest Path First

a) **Input:** Nodes, edges, source and destination nodes.

b) **Process:** Process starts to find the best path which is the shortest distance between source node to destination node based on the sum of the weights.

c) **Output:** Shortest path between the source node and destination node.

2.4.8 Open Shortest Path First

a) **Input:** Nodes, edges, source and destination nodes.

b) **Process:** Process starts to find the best path which is the shortest distance between source node to destination node based on the sum of the weights.

c) **Output: S**hortest path between the source node and destination node.

2.4.9 Minimum Spanning Tree

a) **Input:** Nodes, edges with weight.

b) **Process:** Spans(covers) the tree by drawing edges with minimum weight in increasing order without creating a circle.

c) **Output:** Displays minimum spanning tree.

2.5.10 Flooding Routing Algorithm

a) **Input:** Nodes, source node and data.

b) **Process:** Process starts by creating network topology then the source node transmits the data to all its connected neighbours.

c) **Output:** Highlighted flooded paths.

2.5.11 Distance Vector Routing Algorithm

a) **Input:** Nodes, edges, source and destination nodes.

b) **Process:** Process starts to find the best path which is the shortest distance between source node to destination node based on the sum of weights and gives the distance.

c) **Output:** Shortest distance between source and destination node.

2.5 Design Constraints

2.5.1 Hardware Constraints

- **OS:** Windows 7 or higher. Recommended: Windows 10.
- **CPU:** Intel or AMD Processor with 64 bit support. Recommended: 2.8 GHz or faster processor.
- **Disk Storage:** 4 GB of free disk space or higher.
- **Storage:** 64 GB or higher.

2.5.2 Software Constraints

OS : Windows 7 or higher. Recommended: Windows 10.

2.5.3 Fault Tolerance

Fault tolerance can place a major constraint on how the system is to be designed. Fault tolerance requirements often make the system more complex and expensive so they should be minimized.

2.5.4 Security

Currently Security requirements have become essential and major for all types of systems. Security requirements place restriction on the use of certain commands control access to database, provide different kind of access, requirement for different people, require the use of password and cryptography techniques, and maintain a log of activities in the system.

2.5.5 Standard Compliance

It specifies the requirements for the standard the system must follow. The standards may include the report format, type of navigation, naming conventions for buttons, access keys, shortcut keys.

2.6 System Attributes

- Availability: Availability refers to the percentage of time and that the infrastructure, system or solution remains operational under normal circumstances in order to serve its intended purpose.
- Portability: Portability, in relation to software, is a measure of how easily an application can be transferred from one computer environment to another. A computer application software is considered portable to a new environment if the effort required to adapt it to the new environment is within resource limits.
- Reliability: Reliability refers to the portability list that system will meet certain performance standards in yielding correct output for desired time duration.
- Maintainability: Maintainability refers to the ease with which you can repair, improve and understand software code. software maintenance is a place in the software development cycle that starts after the customer has received the product.
- Scalability: Software scalability is an attribute of a tool or a system to increase its capacity and functionalities based on user demands. Scalable software can

remain stable while adapting to changes, upgrades, overhauls, and resource reduction.

2.7 Other Requirements

Not applicable.

SYSTEM DESIGN

3. System Design

3.1 Introduction

- System Design is the process of defining the architecture, module interfaces and data for a system to satisfy specified requirements.
- The purpose of the design phase is to plan the solution of the problem specified by the requirement documents.
- This is the first step that moving from problem domain to the solution domain.
- The design of the system is essentially a blueprint or a plan for a solution for the system.

3.2 Assumption and Constraints

An assumption is a condition you think to be true and a constraint is fixed limitation of project development.

- All the functional requirement collected from client are sufficient for the project lifecycle.
- All the Non-functional and Specific requirement specified in SRS well enough for the development of the system.
- Time constraint.

3.3 Functional Decomposition

Functional decomposition is the process of taking a complex process and breaking down into its smaller, simpler parts. Using functional decomposition large or complex functionalities are more easily understood. It is mainly used during project analysis phase, and each phase can be viewed as software. So this has module with some sub modules.

3.3.1 System Software Architecture

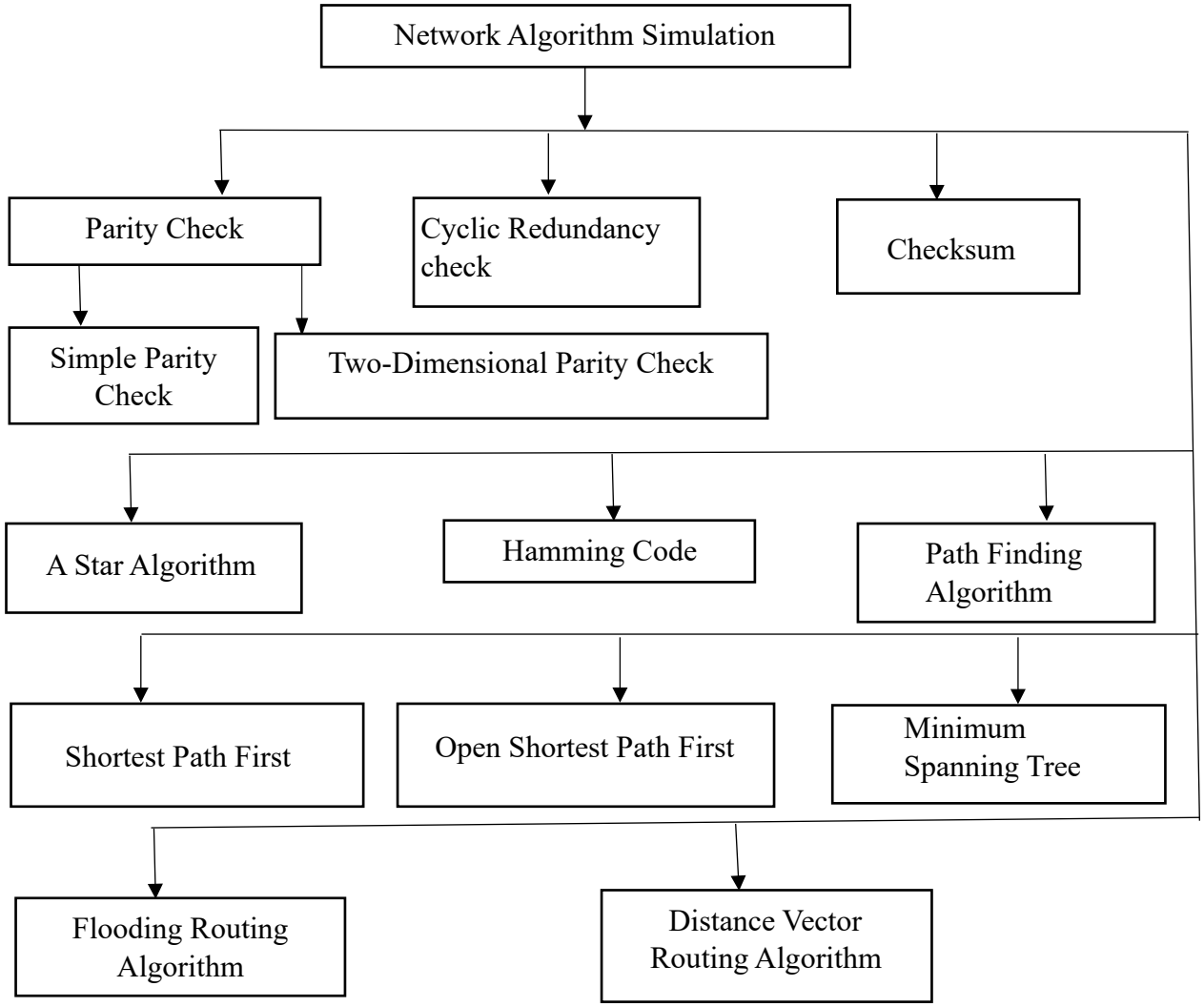


Figure 3.1 System Software Architecture

3.3.2 System Technical Architecture

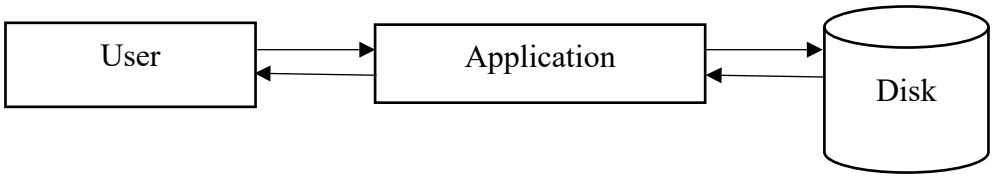


Figure 3.2 System Technical Architecture

3.3.3 System Hardware Architecture

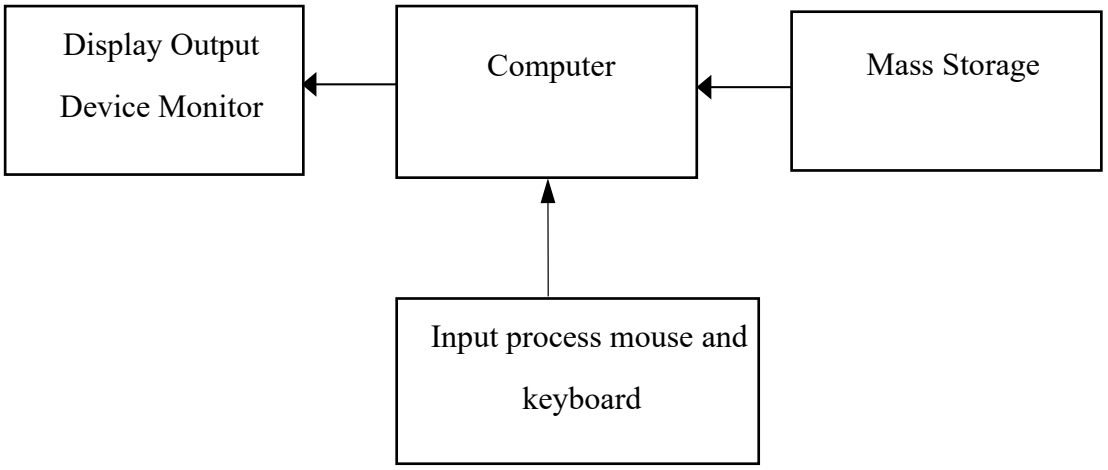


Figure 3.3 System Hardware Architecture

3.3.4 External Interface

Not applicable

3.4 Description of programs

3.4.1 Context Flow Diagram

In CFD entire system is considered as a single process. Context flow diagram shows input and output of the system. It shows all the external entities that interact with the system and how the data flows between those external entities and system.

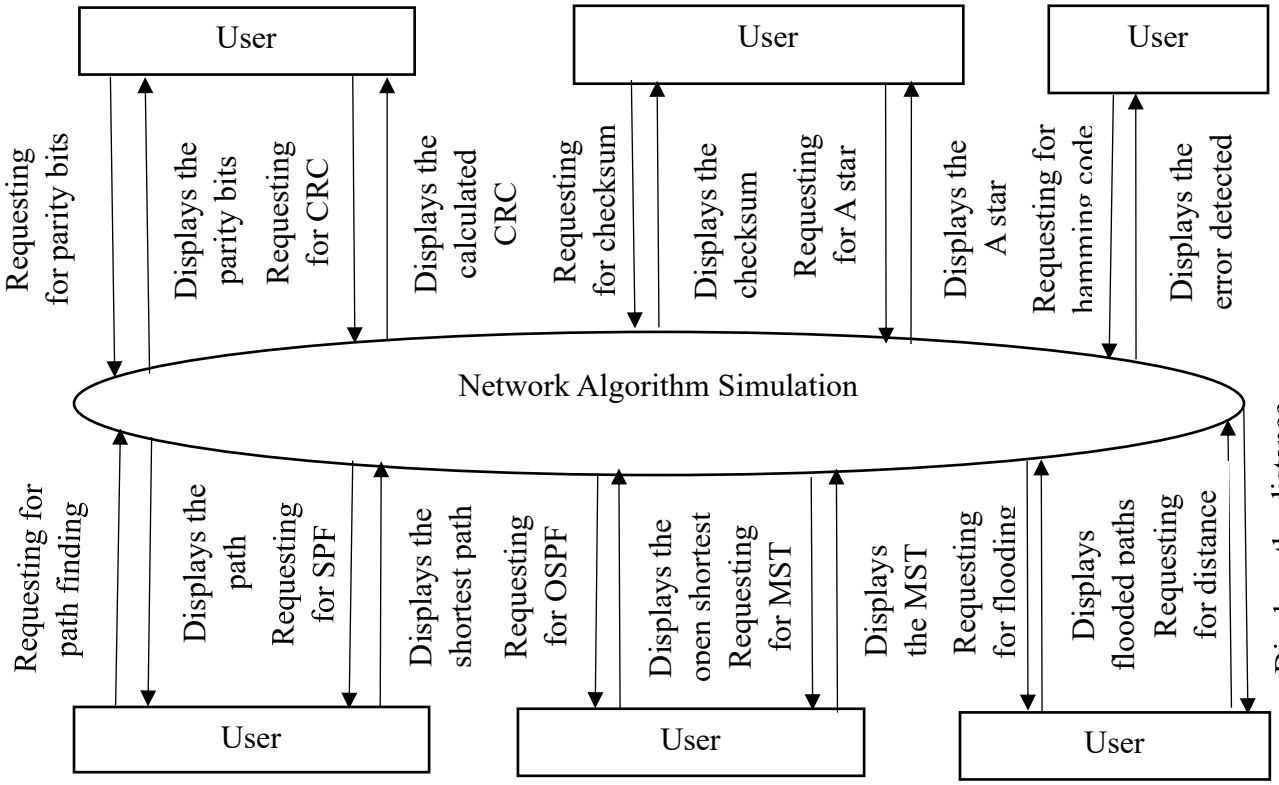


Figure 3.4 Context Flow Diagram

3.4.2 Data Flow Diagram

Data flow diagram shows the flow of data through system. Data flow diagram also called the data flow graphs. It views a system as a function that transforms the input into desired outputs. It aims to capture the transformation that taken place within a system to the input data so that eventually the output data is produced.

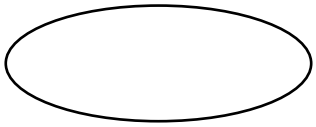
Symbols	Name	Description
	Process	It performs transformation of data from one state to another.
	Source or Sink	It represents the external entity that may be either source or sink.
	Flow of data	It represents flow of data from source to destination.
	Data Source or Data Storage	It is the place where data is stored.

Table 3.1 Data Flow Diagram

3.4.2.1 DFD

LEVEL _ 0

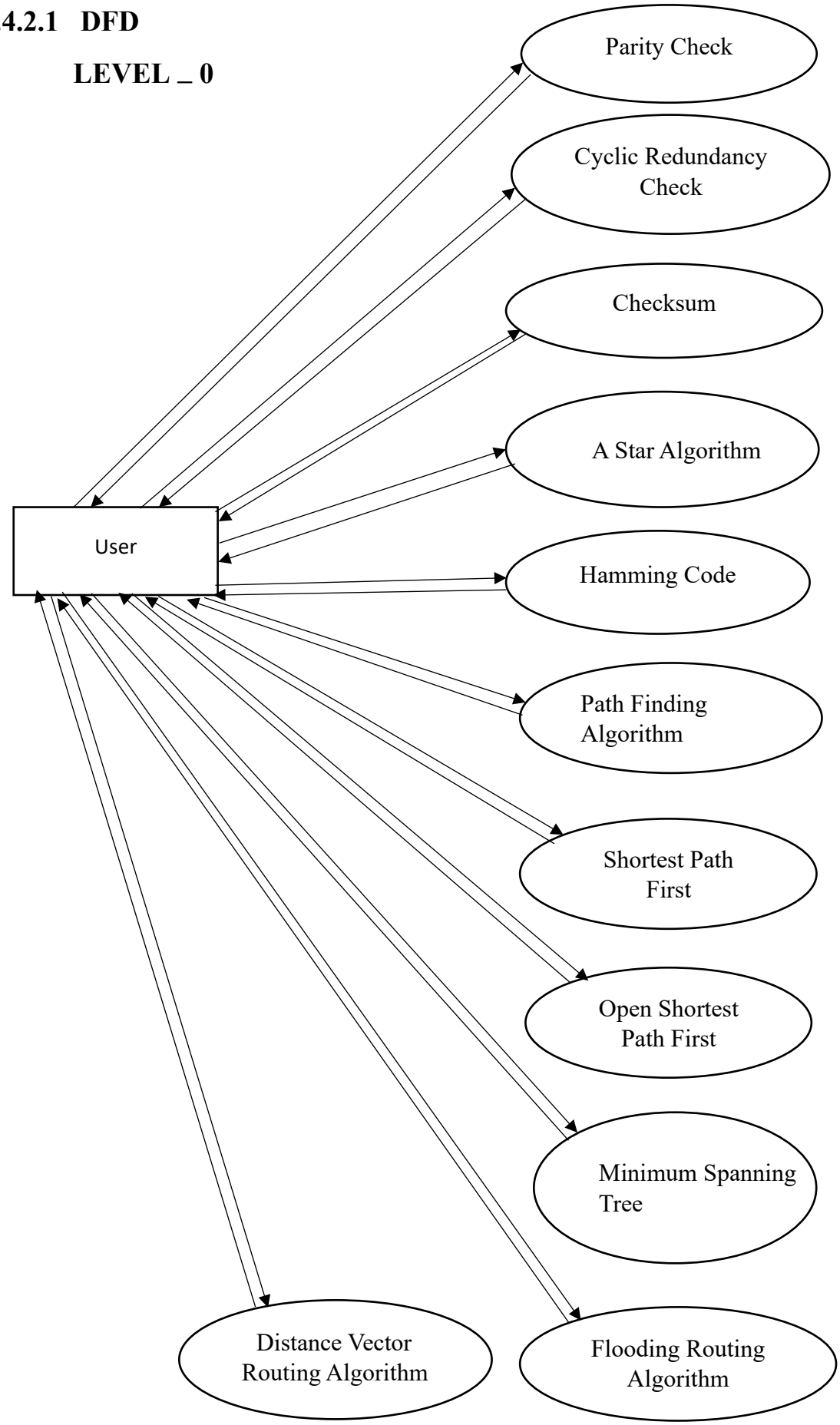


Figure 3.5 DFD

3.5 Description of Component

3.5.1.1 Simple Parity Check

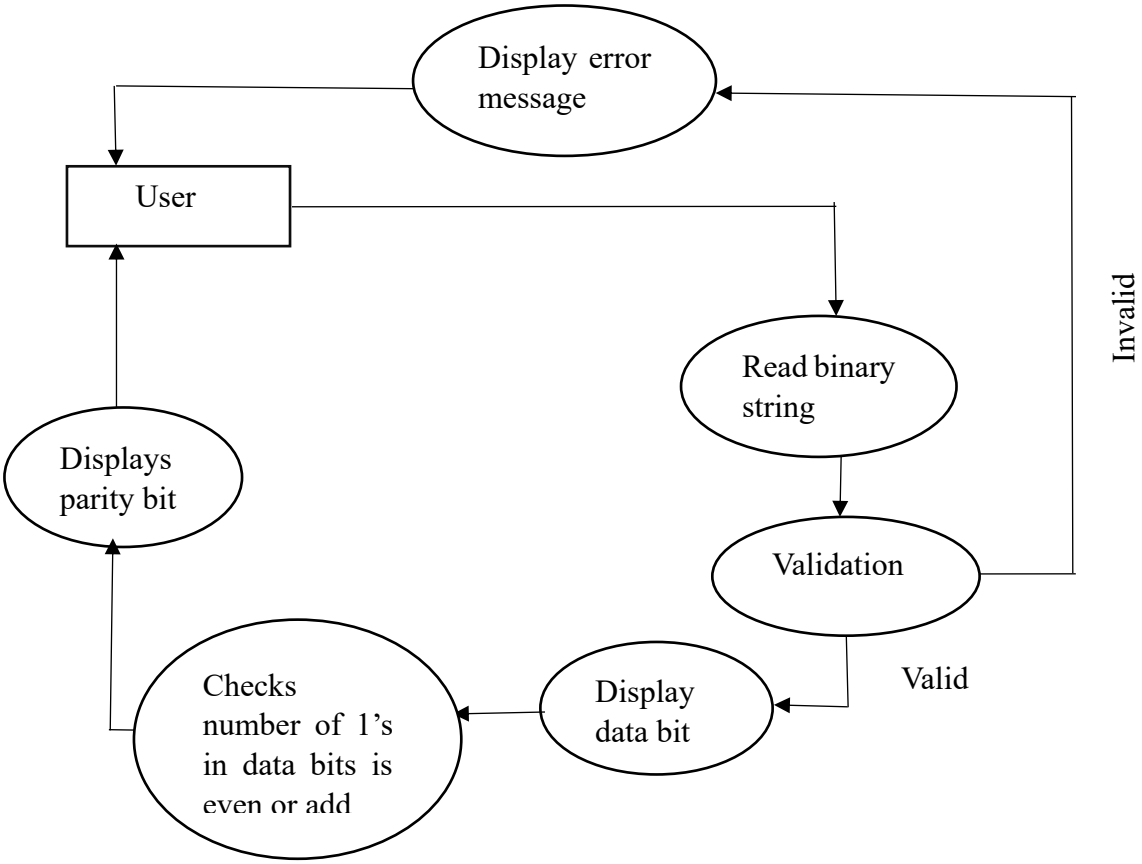


Figure 3.6 Simple Parity Check

3.5.1.1.1 Input: Sequence of binary string.

3.5.1.1.2 Process: Process calculates the parity bit by counting number of 1's and draws coloured blocks representing bit value, green for 1 and red for 0. The parity bit draws blue block if the parity bit value is 1, otherwise white block is drawn.

3.5.1.1.3 Output: Graphical representation of binary string with blocks.

3.5.1.1.4 Interface with other functional component: Not applicable.

3.5.1.1.5 Resource Allocation: Not applicable.

3.5.1.1.6 User Interface: Canvas, Textbox, Frame, Tools, Button.

3.5.1.2 Two-Dimensional Parity Check

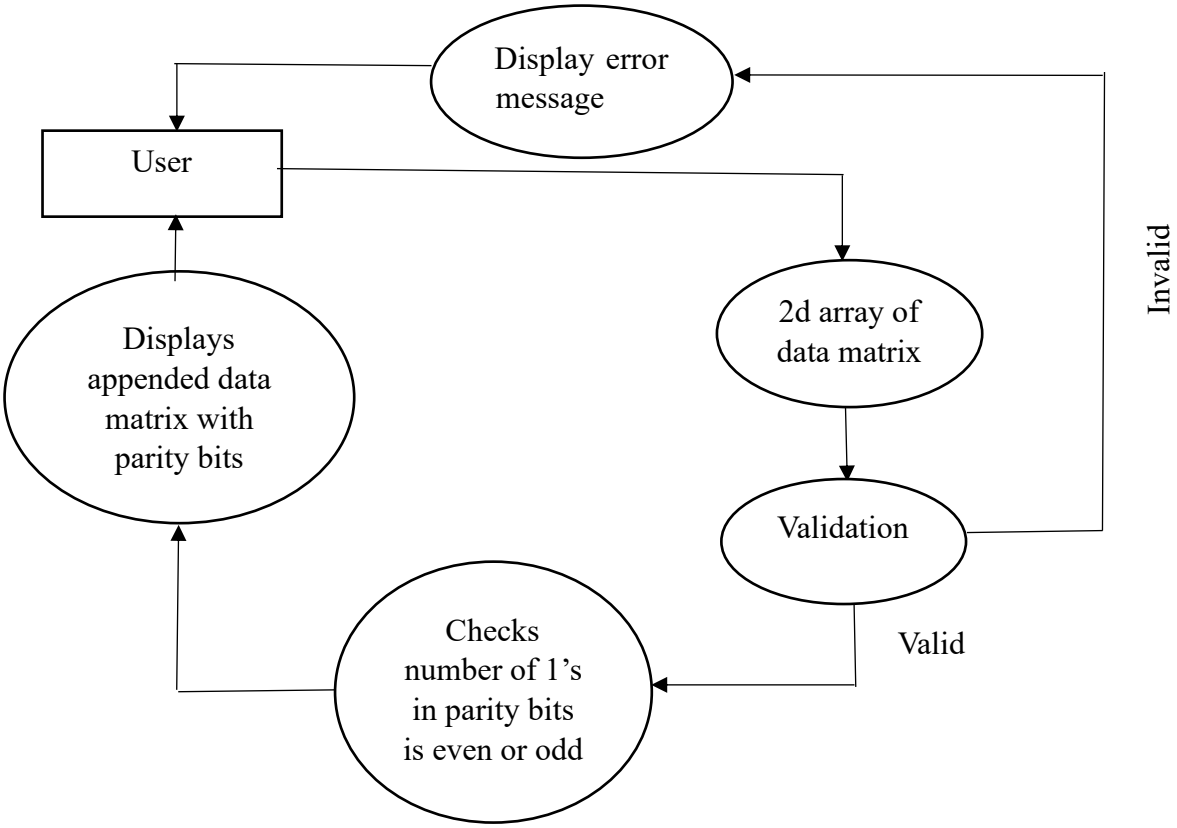


Figure 3.7 Two-Dimensional Parity Check

3.5.1.2.1 Input: Number of rows, columns and data bits.

3.5.1.2.2 Process: Process calculates the row parity and column parity. If number of 1's in the row and column is odd, the row parity and column parity value is 1, otherwise 0.Overall parity is generated based on the sum of the row parity and column parity.

3.5.1.2.3 Output: New 2d list representing data matrix with parity bits.

3.5.1.2.4 Interface with other functional component: Not applicable.

3.5.1.2.5 Resource Allocation: Not applicable.

3.5.1.2.6 User Interface: Canvas, Textbox, Frame, Tools, Button.

3.5.2 Cyclic Redundancy Check

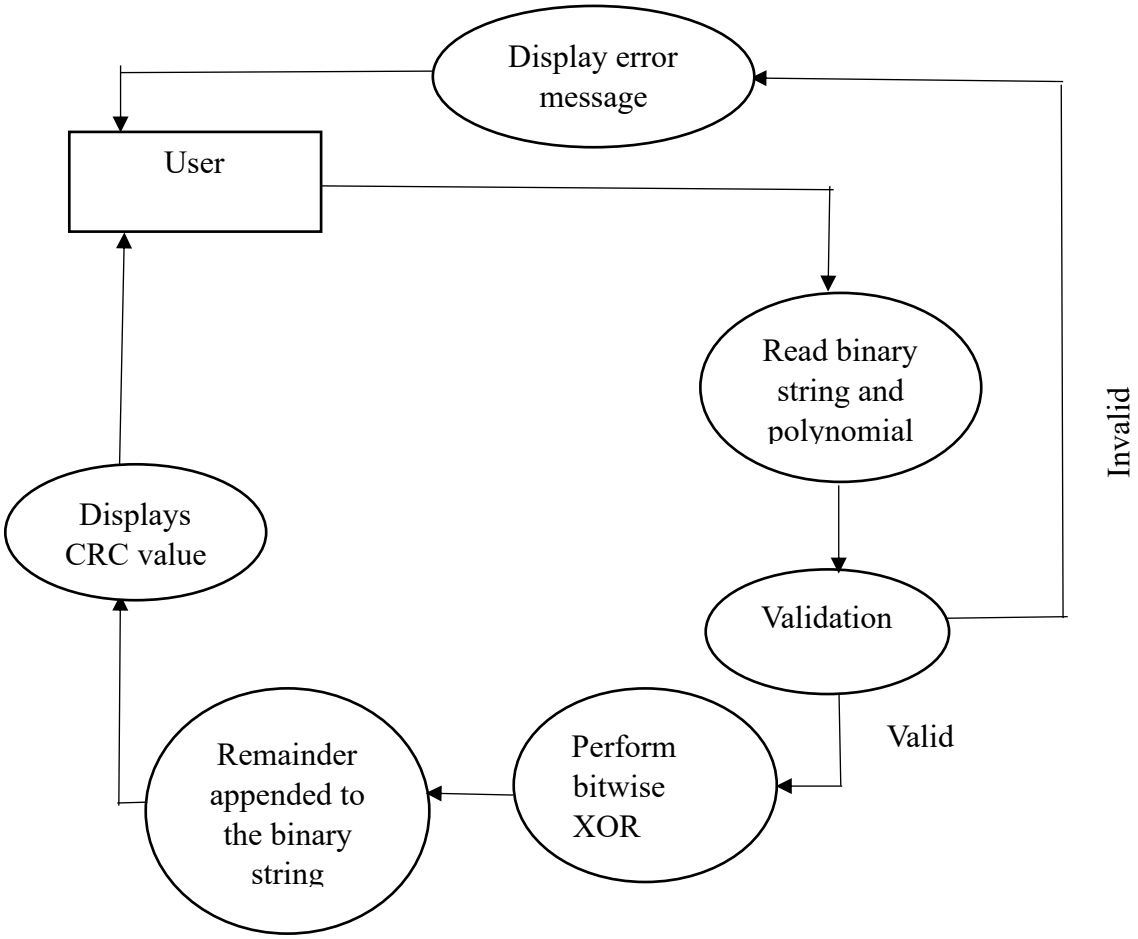


Figure 3.8 Cyclic Redundancy Check

3.5.2.1 Input: Binary bit message value and polynomial.

3.5.2.2 Process: Process starts with appending 0's to binary bit message equal to the degree of polynomial. Performs bitwise XOR of binary bit message with polynomial, calculates the remainder (CRC) of final message. Remainder is appended to the original message which gives us the CRC value.

3.5.2.3 Output: Calculated CRC (remainder) and message with the CRC value added.

3.5.2.4 Interface with other functional component: Not applicable.

3.5.2.5 Resource Allocation: Not applicable.

3.5.2.6 User Interface: Canvas, Textbox, Frame, Tools, Button.

3.5.3 Checksum

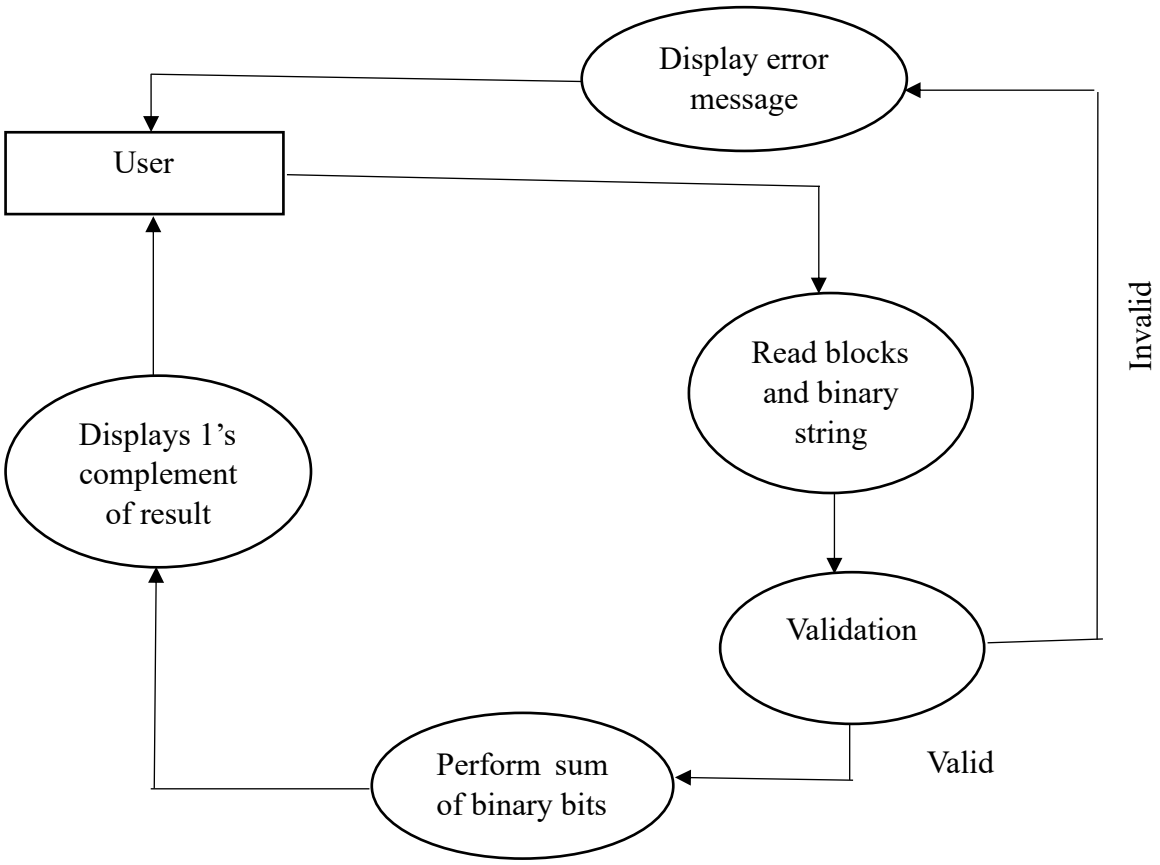


Figure 3.9 Checksum

3.5.3.1 Input: Sequence of binary string and blocks.

3.5.3.2 Process: Checksum value is calculated by making the one's complement of the results obtained by XOR operation of the given input message.

3.5.3.3 Output: It returns the Checksum i.e, one's complement of result.

3.5.3.4 Interface with other functional component: Not applicable.

3.5.3.5 Resource Allocation: Not applicable.

3.5.3.6 User Interface: Canvas, Textbox, Frame, Tools, Button.

3.5.4 A Star Algorithm

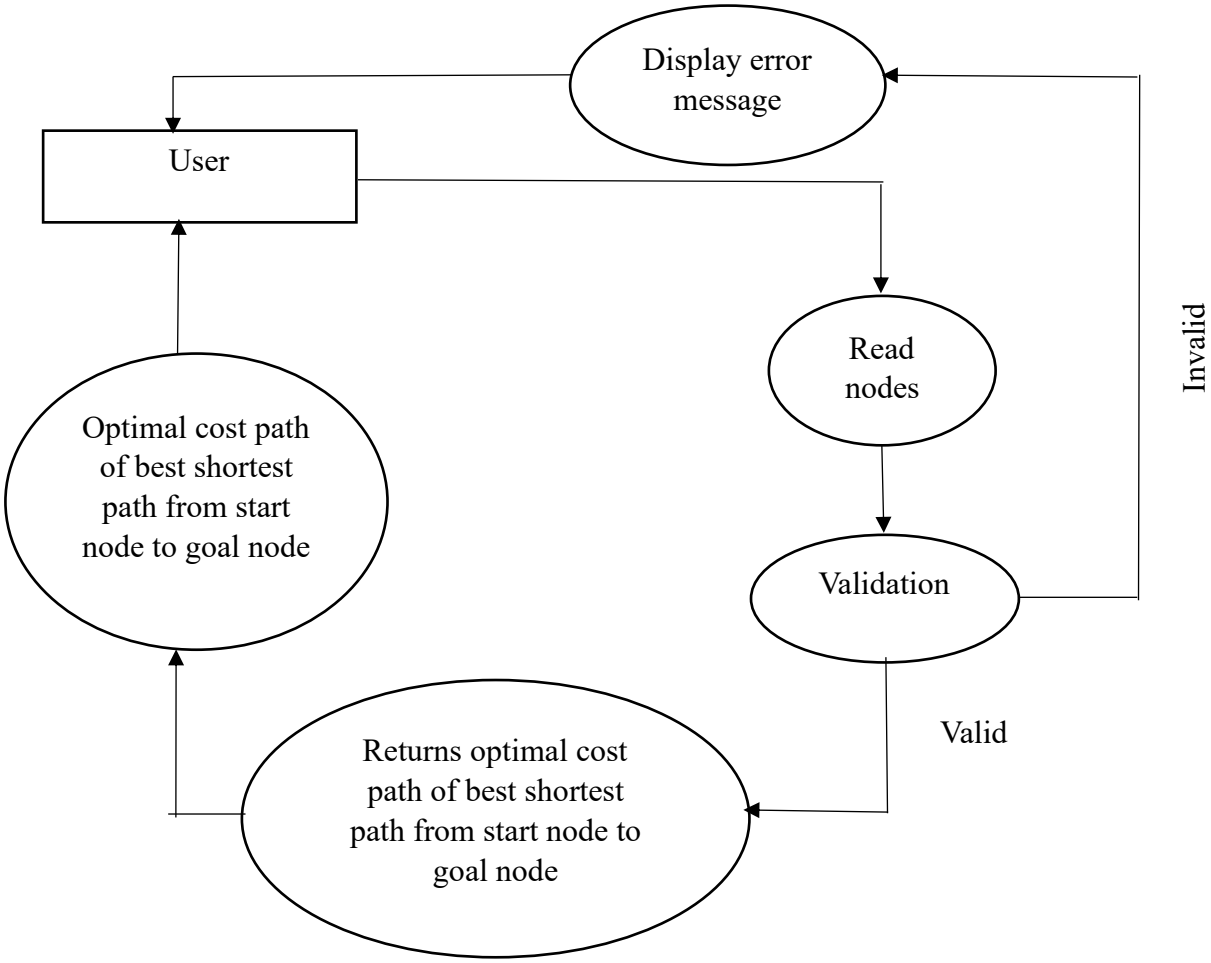


Figure 3.10 A Star Algorithm

3.5.4.1 Input: Nodes, Edges.

3.5.4.2 Process: The heuristic function takes a node as input and returns a value representing the estimated cost (heuristic) from that node to the goal node and determines the valid neighbours of a given node.

3.5.4.3 Output: Optimal path from the start node to goal node.

3.5.4.4 Interface with other functional component: Not applicable.

3.5.4.5 Resource Allocation: Not applicable.

3.5.4.6 User Interface: Canvas, Textbox, Frame, Tools, Button.

.

3.5.5 Hamming Code

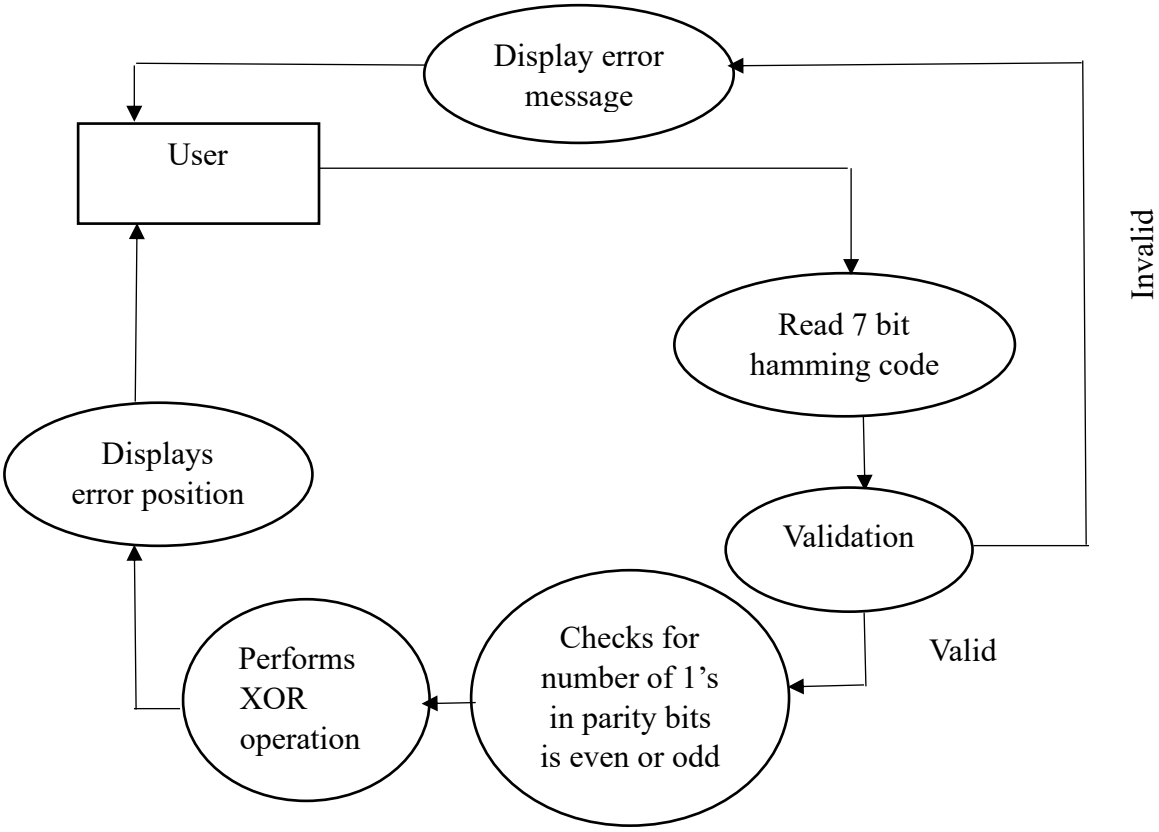


Figure 3.11 Hamming Code

3.5.5.1 Input: 7-bit hamming code.

3.5.5.2 Process: Process starts by detecting error in the parity bits by check-one skip- one method for P1 parity and so on. If number of ones in the parity bit is odd then error is detected in the received code, otherwise no error. Then to find the error position in the received code using XOR operation.

3.5.5.3 Output: Displays the position of the error in the received code if error is detected.

3.5.5.4 Interface with other functional component: Not applicable.

3.5.5.5 Resource Allocation: Not applicable.

3.5.5.6 User Interface: Canvas, Textbox, Frame, Tools, Button.

3.5.6 Path Finding Algorithm(Greedy Kruskal)

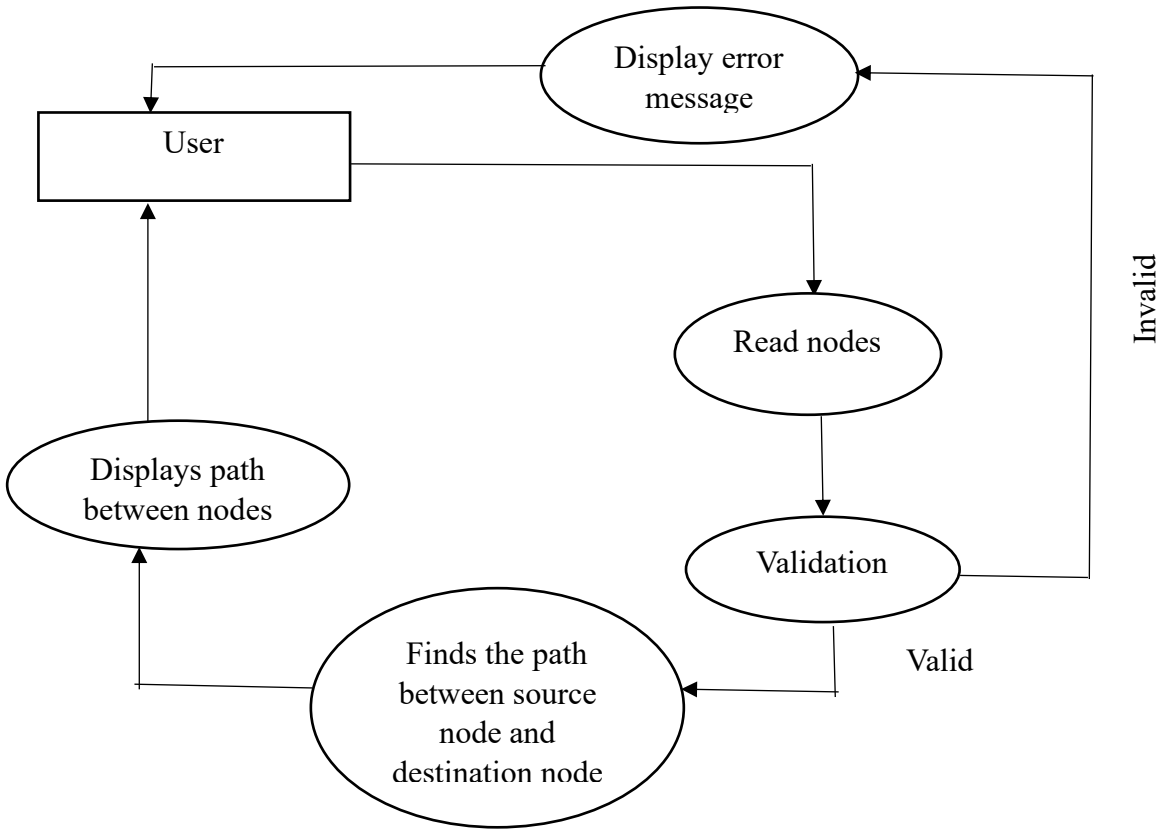


Figure 3.12 Path Finding Algorithm

3.5.6.1 Input: Nodes, edges.

3.5.6.2 Process: Finds path between source node and destination node.

3.5.6.3 Output: Path between the source node and destination node.

3.5.6.4 Interface with other functional component: Not applicable.

3.5.6.5 Resource Allocation: Not applicable.

3.5.6.6 User Interface: Canvas, Textbox, Frame, Tools, Button.

3.5.7 Shortest Path First(Dijkstra's)

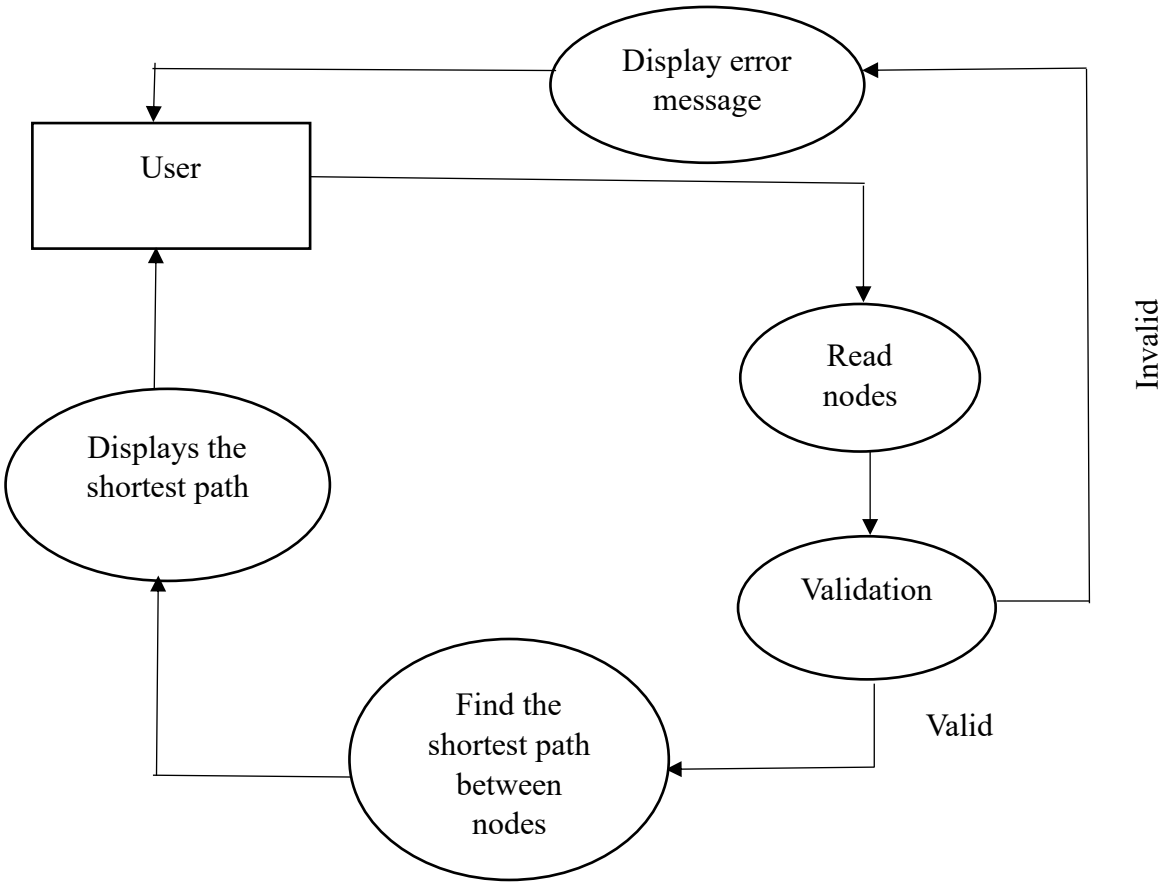


Figure 3.13 Shortest Path First

3.5.7.1 Input: Nodes, edges, source and destination nodes..

3.5.7.2 Process: Process starts to find the best path which is the shortest distance between source node to destination node based on the sum of the weights.

3.5.7.3 Output: Shortest path between the source node and destination node.

3.5.7.4 Interface with other functional component: Not applicable.

3.5.7.5 Resource Allocation: Not applicable.

3.5.7.6 User Interface: Canvas, Textbox, Frame, Tools, Button.

.

3.5.8 Open Shortest Path First

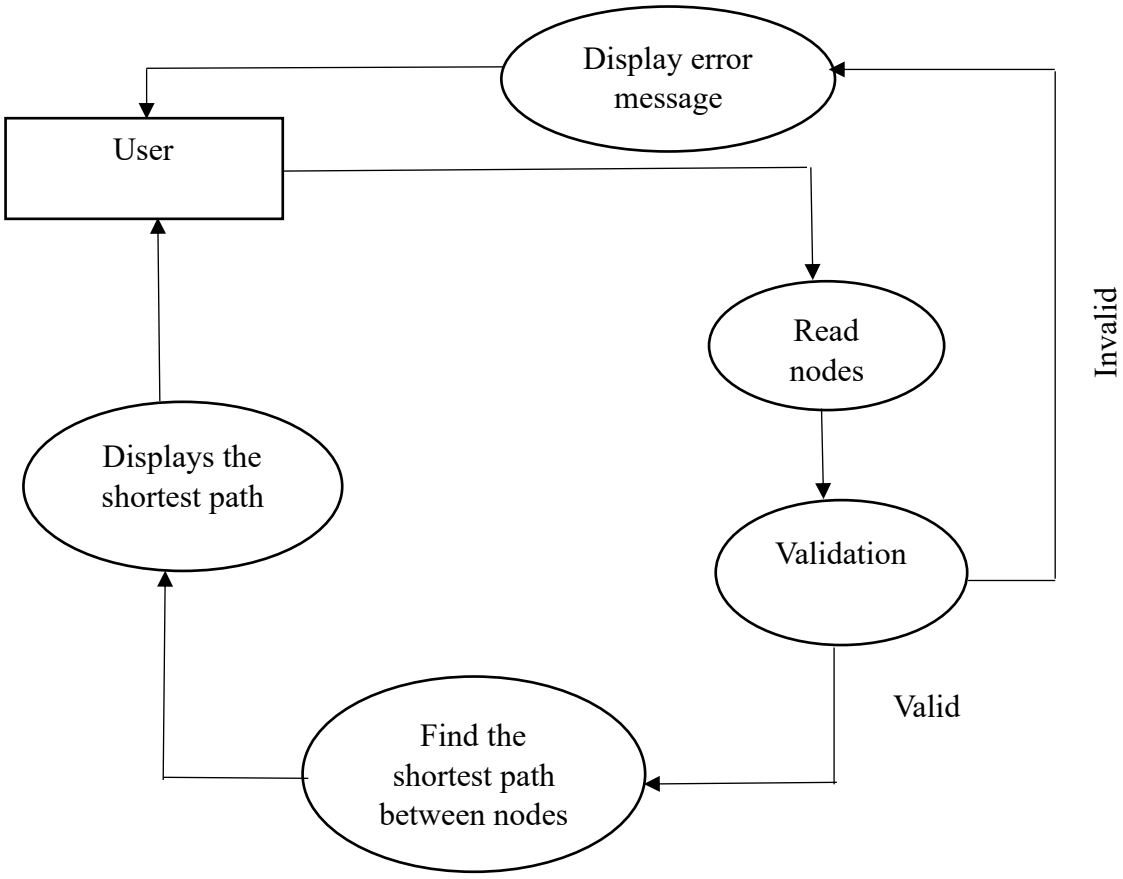


Figure 3.14 Open Shortest Path First

3.5.8.1 Input: Nodes, edges, source and destination nodes.

3.5.8.2 Process: Process starts to find the best path which is the shortest distance between source node to destination node based on the sum of the weights.

3.5.8.3 Output: Shortest path between the source node and destination node.

3.5.8.4 Interface with other functional component: Not applicable.

3.5.8.5 Resource Allocation: Not applicable.

3.5.8.6 User Interface: Canvas, Textbox, Frame, Tools, Button.

.

3.5.9 Minimum Spanning Tree(Greedy Kruskal)

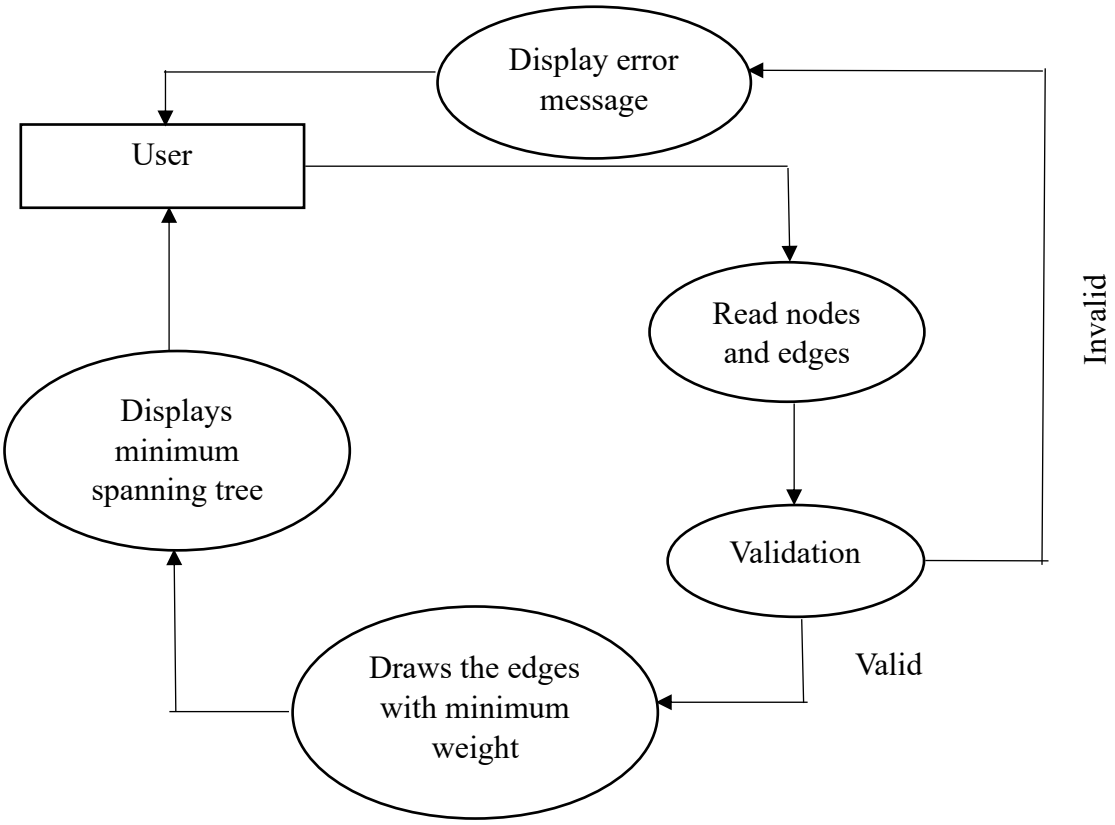


Figure 3.15 Minimum Spanning Tree

3.5.9.1 Input: Nodes, edges with weight.

3.5.9.2 Process: Spans(covers) the tree by drawing edges with minimum weight in increasing order without creating a circle

3.5.9.3 Output: Displays minimum spanning tree.

3.5.9.4 Interface with other functional component: Not applicable.

3.5.9.5 Resource Allocation: Not applicable.

3.5.9.6 User Interface: Canvas, Textbox, Frame, Tools, Button.

3.5.10 Flooding Routing Algorithm

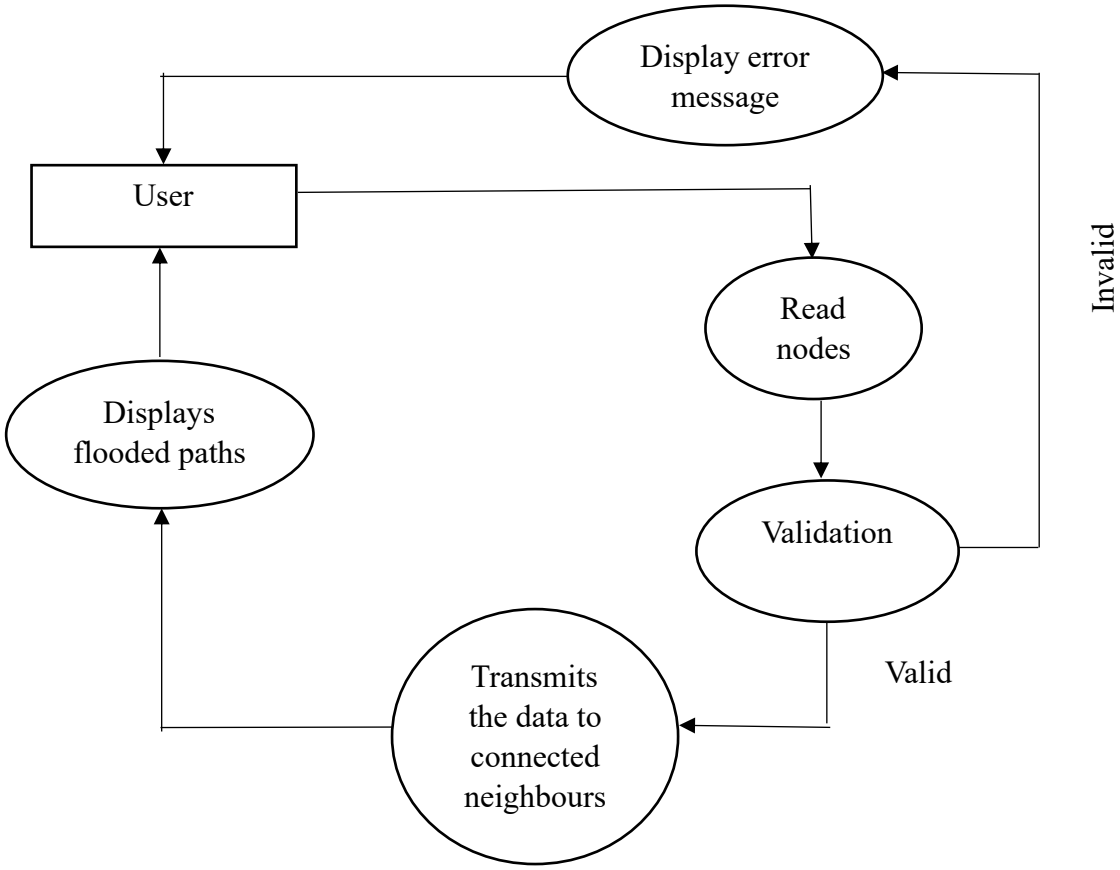


Figure 3.16 Flooded Routing Algorithm

3.5.10.1 Input: Nodes, source node and data.

3.5.10.2 Process: Process starts by creating network topology then the source node transmits the data to all its connected neighbours.

3.5.10.3 Output: Highlighted flooded paths.

3.5.10.4 Interface with other functional component: Not applicable.

3.5.10.5 Resource Allocation: Not applicable.

3.5.10.6 User Interface: Canvas, Textbox, Frame, Tools, Button.

3.5.11 Distance Vector Routing Algorithm

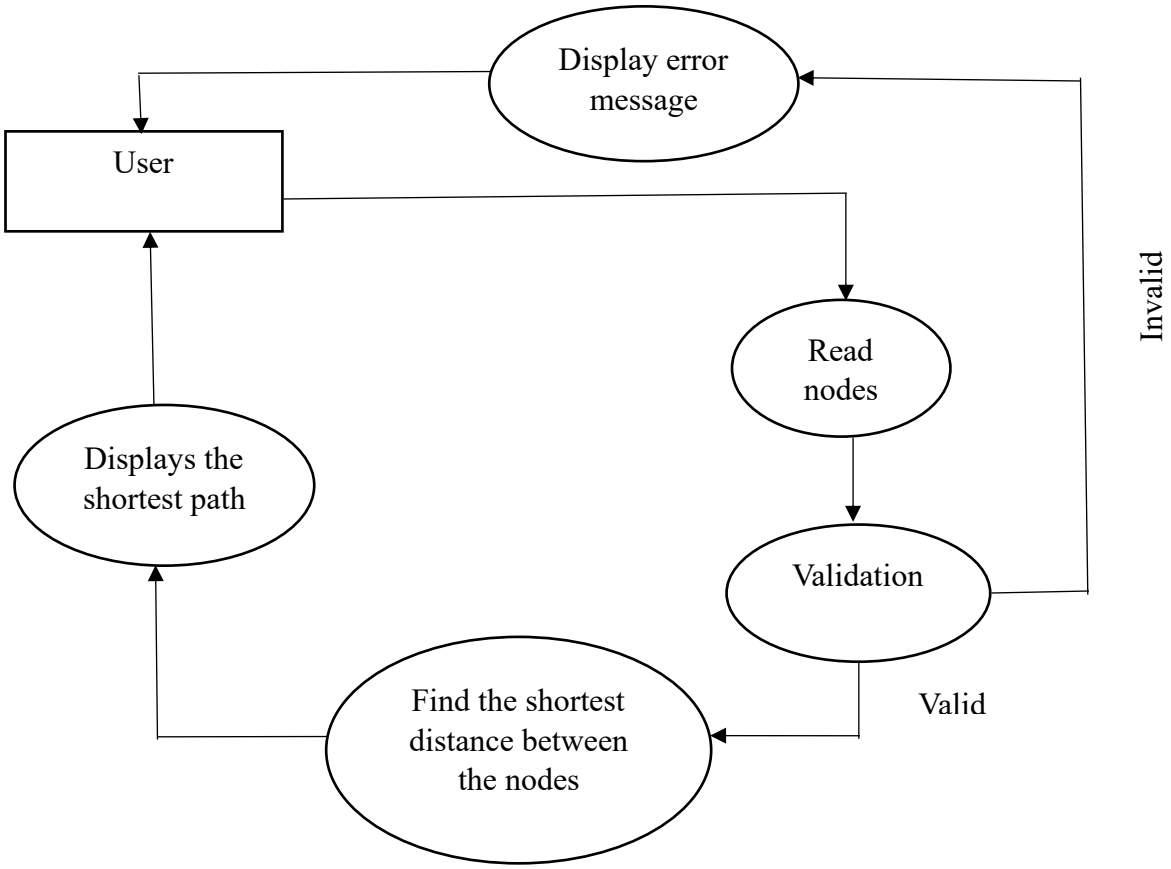


Figure 3.17 Distance Vector Routing Algorithm

3.5.11.1 Input: Nodes, edges, source and destination nodes.

3.5.11.2 Process: Process starts to find the best path which is the shortest distance between source node to destination node based on the sum of weights and gives the distance.

3.5.11.3 **Output:** Shortest distance between the source node an destination node.

. **3.5.11.4** **Interface with other functional component:** Not applicable.

3.5.11.5 **Resource Allocation:** Not applicable.

3.5.11.6 **User Interface:** Canvas, Textbox, Frame, Tools, Button.

DETAIL DESIGN

4. DETAILED DESIGN

4.1 Introduction

During detailed design, the internal logic of each module specified in system design is decided. During this phase further details of the modules are decided. Design of each of the modules usually specified in a high-level description language which is independent of the language in which software eventually be implemented.

4.2 Structure of software package

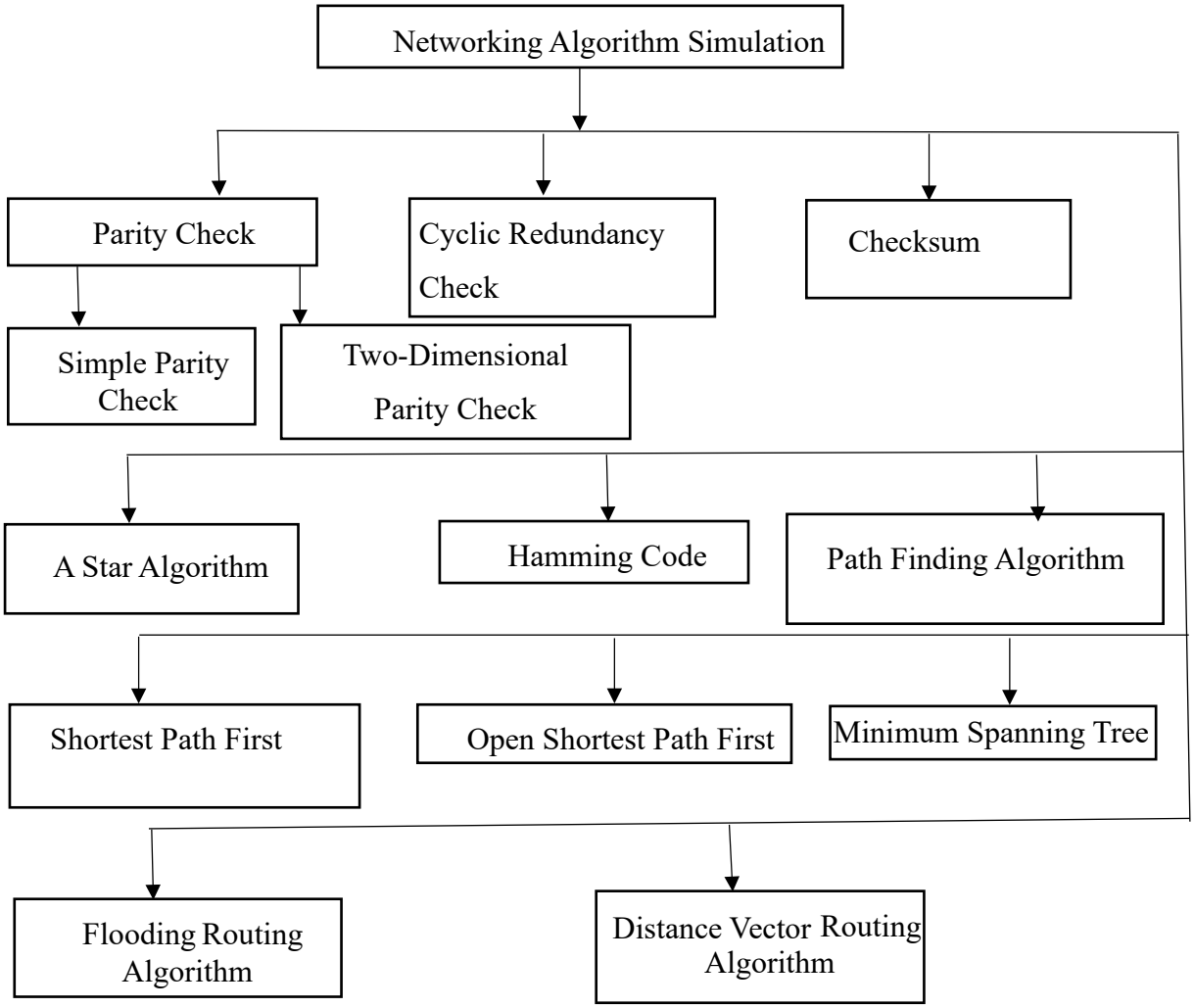


Figure 4.1 Structure of software package

4.3 Module decomposition of software

Structure Chart:

Structure chart is a top-down modular design, consist of squares representing different models in a system and lines. Structure chart shows how program has been partitioned into manageable modules hierarchy and organization of those modules and communicational interface.

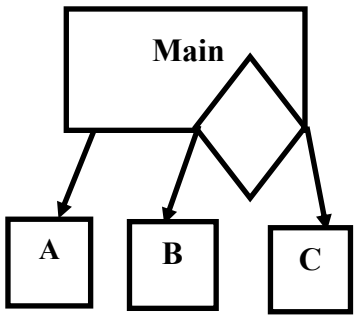
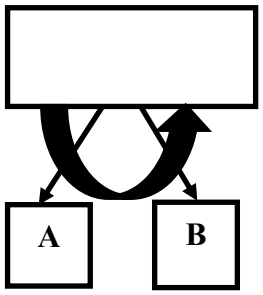
Symbol	Name	Process
	**Data flow**	**Show the direction flow of data.**
	**Control flow**	**Shows the direction of flow control.**
	**Processing**	**Shows manipulation, calculation and processing.**
	Module Invocation	**It represents subordinate module being invoked by superior ordinate module.**
	**Condition invocation**	**It indicates that the invocation of subordinates** **Module depends on the evaluation of a condition**
	**Iteration**	**It represents the iteration**

Table 4.1 Structure chart

Flow Chart:

Flow chart is a graphical representation of solution to the given problems. A Flowchart is a pictorial representation of an algorithm, workflow or process. The diagrammatic representation illustrates a solution model to given problem. It uses the following symbol.

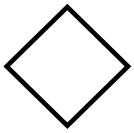
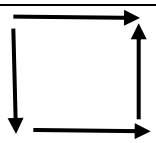
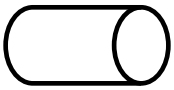
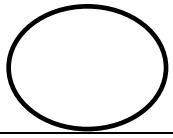
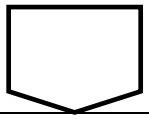
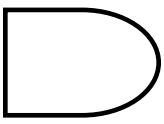
Symbol	Name	Purpose
	**Terminator**	**It indicates the start and end process.**
	**Input/output**	**Input / output data**
	**Decision**	**It represents a comparison or question that determines an alternative path to be followed.**
	**Flow direction**	**Shows the direction of data flow.**
	**Processing**	**It represents manipulation, calculation or information processing.**
	**Direction access storage**	**File storage**
	**Preparation (Looping)**	**An instruction or Group of instruction**
	**In-Page**	
	**Off-Page**	
	**Delay**	

Table 4.2 Flowchart

4.3.1 Parity Check

4.3.1.1 Simple Parity Check

4.3.1.1.1 **Input:** Sequence of binary string.

4.3.1.1.2 **Procedural details**

Algorithm:

Step 1: Start

Step 2: Input binary string

Step 3: Calculate parity bit by counting number of 1's in binary string

Step 4: Draws the blocks

for bit in binary string:

if bit == '1':

color = 'green'

else:

color = 'red'

draw_block(x, y, block_size, color, bit)

x += block_size

Step 5: Draw parity bit

Call parity_label = "p = 1" if parity_bit == 1 else "p =0"

draw_block(x, y, block_size,'blue'

if parity_bit == 1 else 'white' parity_label)

Step 6: if number of 1's odd then

displays blue colored block

else:

displays white colored block

Step 7: End

Algorithm 4.1 Simple Parity Check

4.3.1.1.3 File I/O Interface: Not applicable.

4.3.1.1.4 Output: Graphical representation of binary string with blocks.

4.3.1.1.5 Implementation aspects: Canvas, Button, Frame, Tools, Textbox

4.3.1.2 Two-Dimensional Parity Check

4.3.1.2.1 Input: Number of rows, columns and data bits.

4.3.1.2.2 Procedural details

Flow Chart:

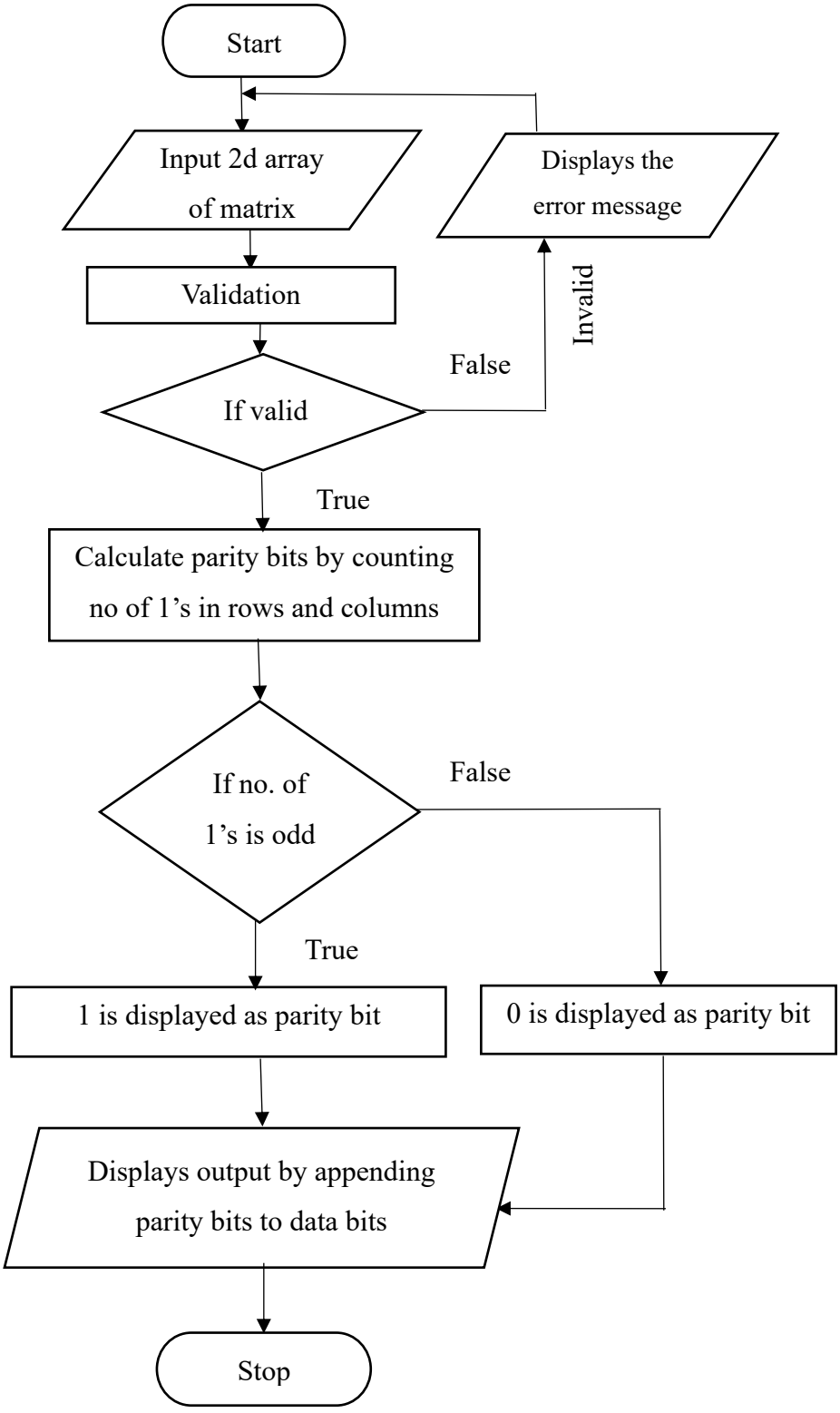


Figure 4.2 Two-Dimensional Parity Check

4.3.1.2.3 File I/O Interface: Not Applicable

4.3.1.2.4 Output: New 2d list representing data matrix with parity bits.

4.3.1.2.5 Implementation aspects: Canvas, Button, Frame, Tools, Textbox

4.3.2 Cyclic Redundancy Check

4.3.2.1 Input: Sequence of binary string and polynomial.

4.3.2.2 Procedural details

Flow Chart:

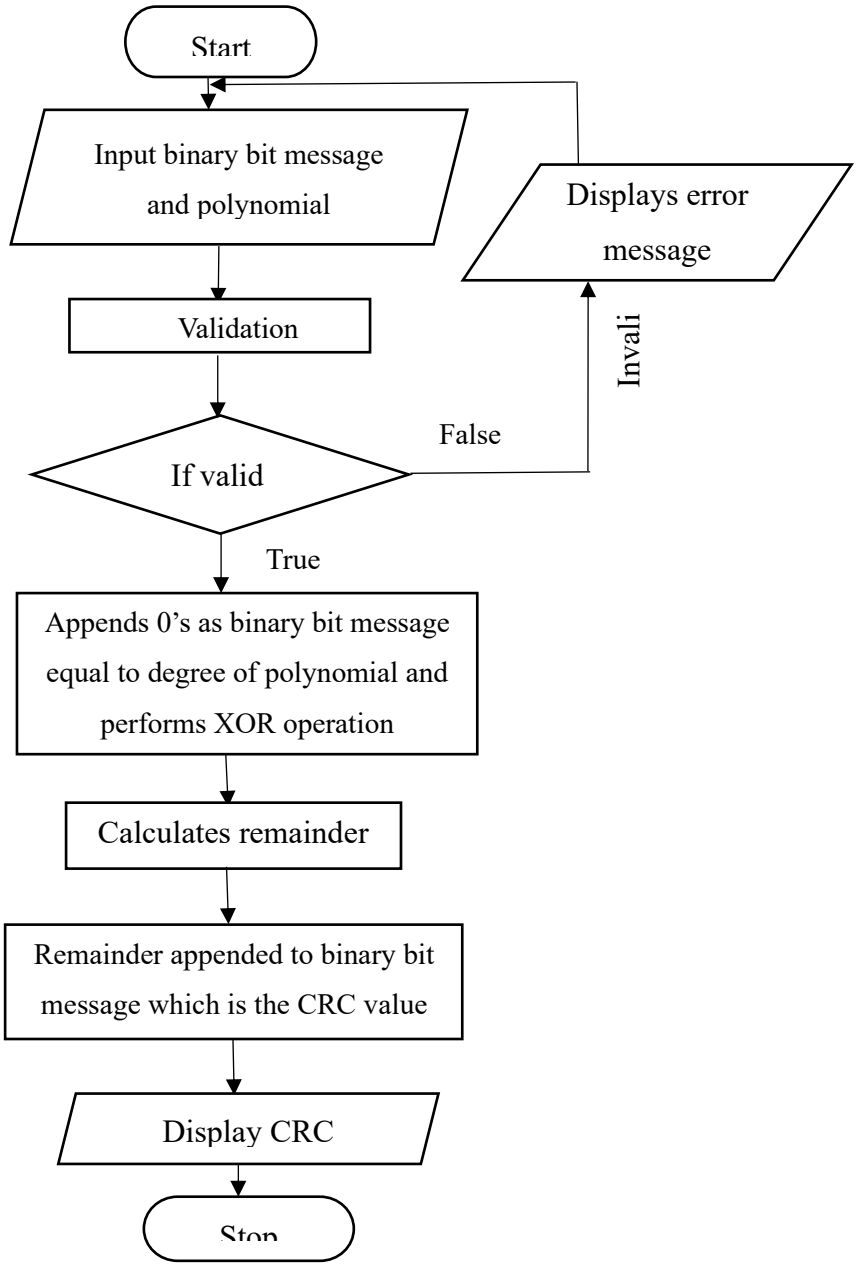


Figure 4.3 Cyclic Redundancy Check

4.3.2.3 File I/O Interface: Not applicable.

4.3.2.4 Output: Calculated CRC (remainder) and message with the CRC value added.

4.3.2.5 Implementation aspects: Canvas, Button, Frame, Tools, Textbox

4.3.3 Checksum

4.3.3.1 Input: Sequence of binary string and blocks.

4.3.3.2 Procedural details

Flow Chart:

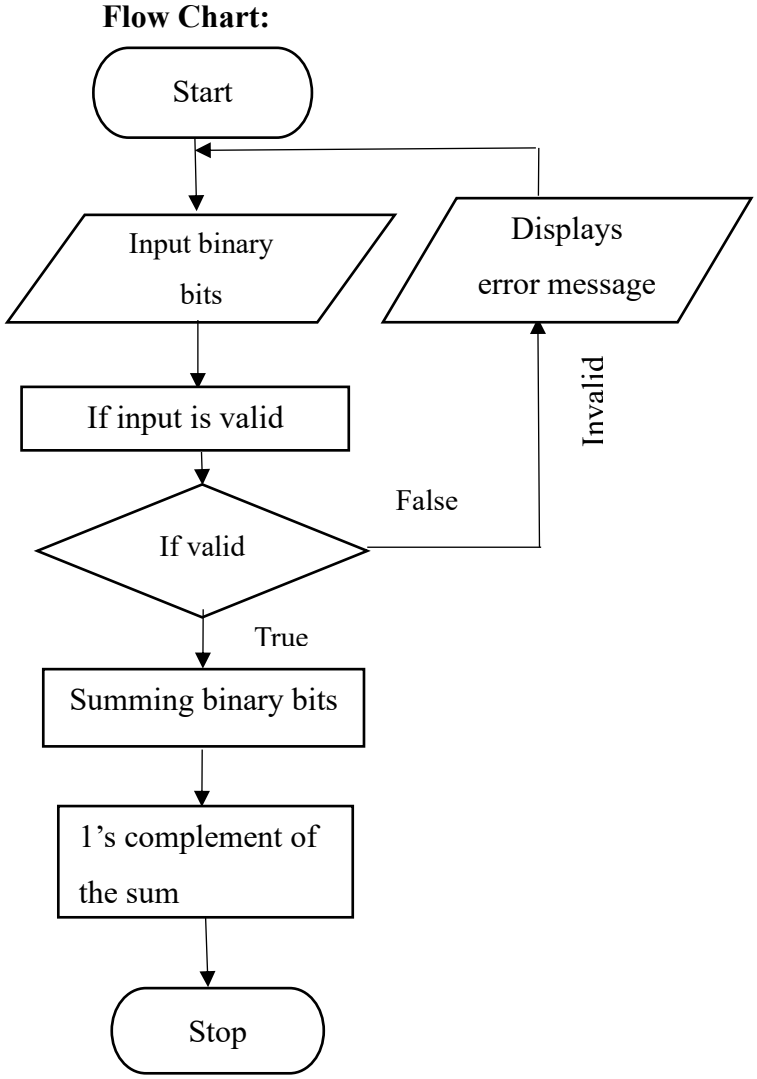


Figure 4.4 Checksum

4.3.3.3 File I/O Interface: Not applicable.

4.3.3.4 Output: It returns the Checksum i.e, one's complement of result.

4.3.3.5 Implementation aspects: Canvas, Button, Frame, Tools, Textbox

4.3.4 A Star Algorithm

4.3.4.1 Input: Nodes, Edges.

4.3.4.2 Procedural details

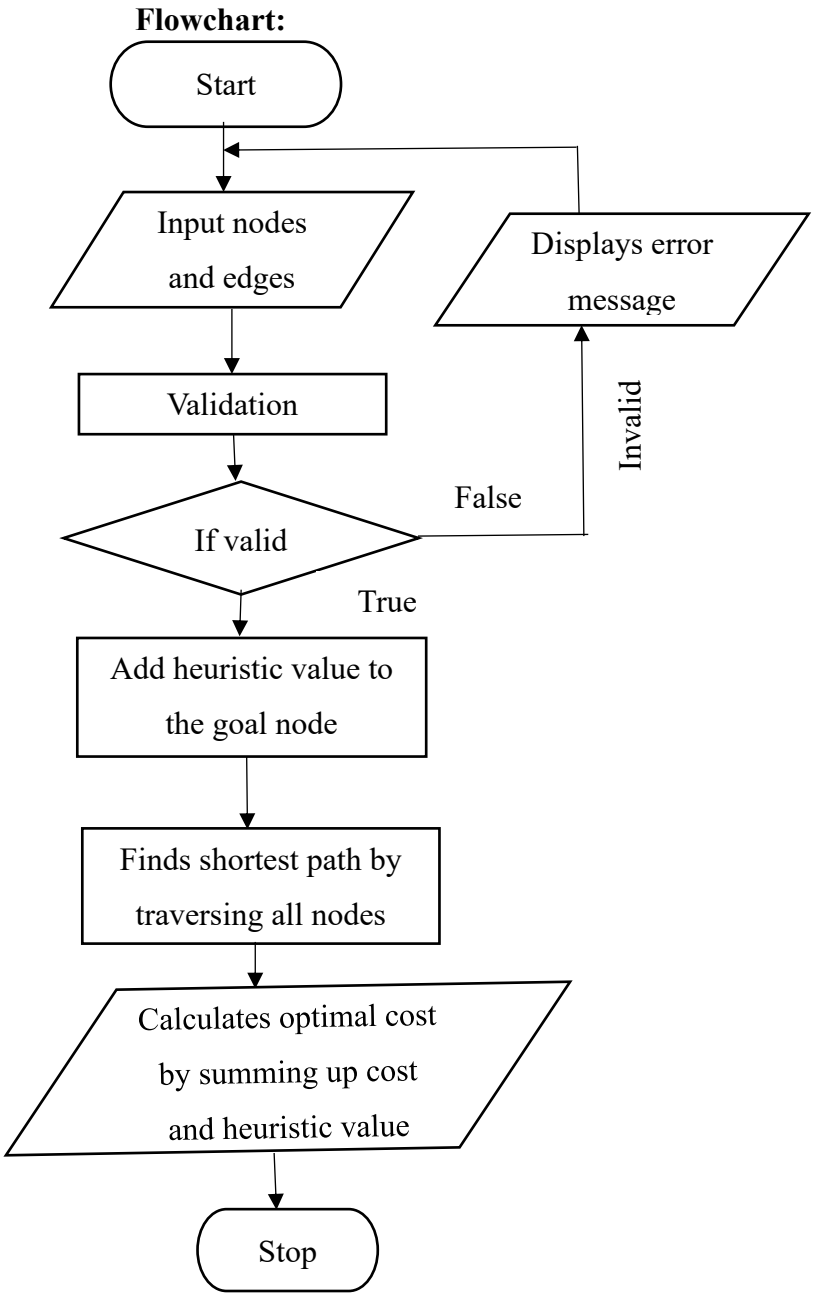


Figure 4.5 A Star Algorithm

4.3.4.3 File I/O Interface: Not applicable

4.3.4.4 Output: Optimal path from the start node to the goal node.

4.3.4.5 Implementation aspects: Canvas, Button, Frame, Tools, Textbox

4.3.5 Hamming Code

4.3.5.1 Input: 7-bit hamming code.

4.3.5.2 Procedural details

Structure Chart:

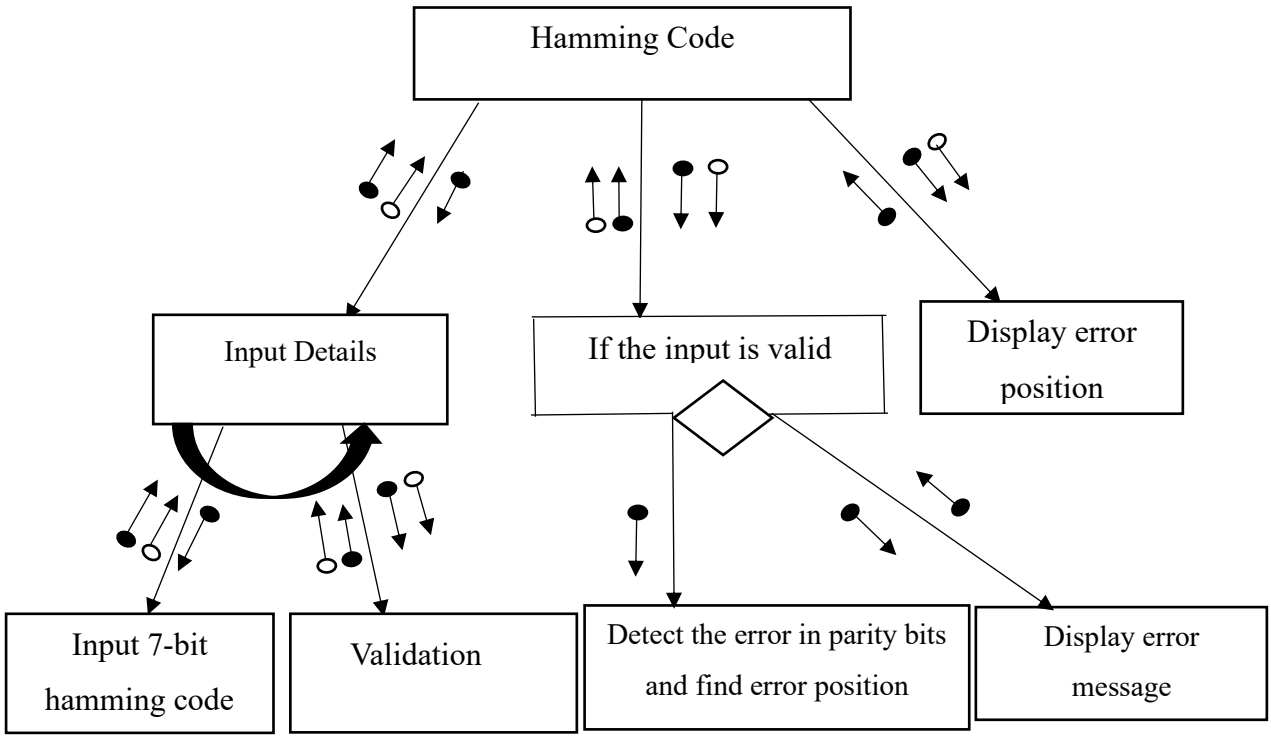


Figure 4.6 Hamming Code

4.3.5.3 File I/O Interface: Not applicable

4.3.5.4 Output: Displays the position of the error in the received code if error is detected.

4.3.5.5 Implementation aspects: Canvas, Button, Frame, Tools, Textbox

4.3.6 Path Finding Algorithm (Greedy Kruskal)

4.3.6.1 Input: Nodes, Edges

4.3.6.2 Procedural details

Structure Chart:

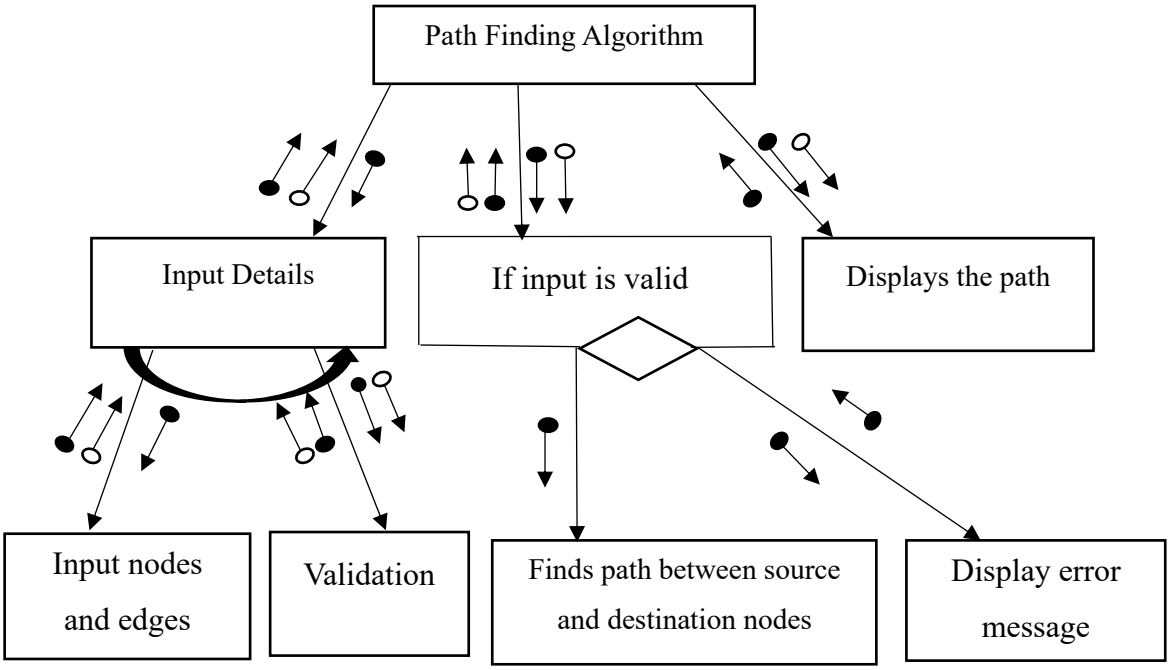


Figure 4.7 Path Finding Algorithm

4.3.6.3 File I/O Interface: Not applicable

4.3.6.4 Output: Path between the source and destination nodes.

4.3.6.5 Implementation aspects: Canvas, Button, Frame, Tools, Textbox

4.3.7 Shortest Path First (Dijkstra's)

4.3.7.1 Input: Nodes, edges, source and destination nodes.

4.3.7.2 Procedural details

Structure Chart:

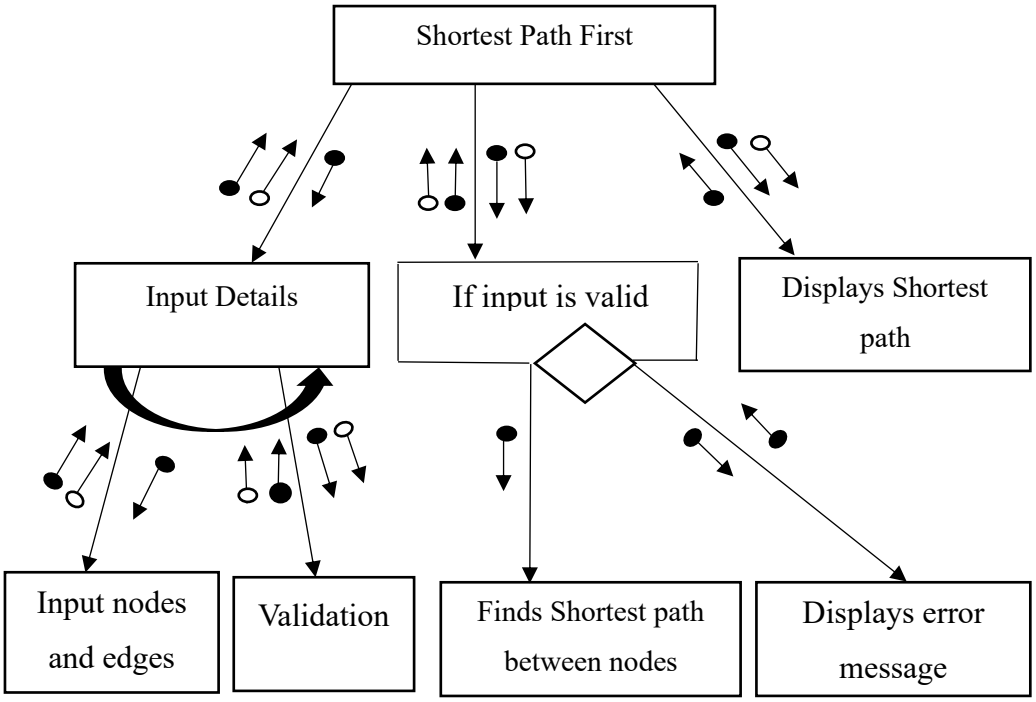


Figure 4.8 Shortest Path First

4.3.7.3 File I/O Interface: Not applicable

4.3.7.4 Output: Shortest path between the source node and destination node.

4.3.7.5 Implementation aspects: Canvas, Button, Frame, Tools, Textbox

4.3.8 Open Shortest Path First

4.3.8.1 Input: Nodes, edges, source and destination nodes.

4.3.8.2 Procedural details

Algorithm:

Step 1: Start

Step 2: Input the nodes and edges with weight.

Step 3: Arrange nodes in a linear order.

Step 4: Generate a fully connected topology based on the weights of the edges

Step 5: Displays the network topology.

Step 6: Input the start vertex and end vertex.

Step 7: if start vertex and end vertex exists

Displays the shortest path

else:

No path found

Step 8: Display the shortest path between the start and end vertex highlighted in red color.

Step 9: End

Algorithm 4.2 Open Shortest Path First

4.3.8.3 File I/O Interface: Not applicable

4.3.8.4 Output: Shortest path between the source node and destination node.

4.3.8.5 Implementation aspects: Canvas, Button, Frame, Tools, Textbox

4.3.9 Minimum Spanning Tree(Greedy Kruskal)

4.3.9.1 Input: Nodes, Edges with weight

4.3.9.2 Procedural details

Structure Chart:

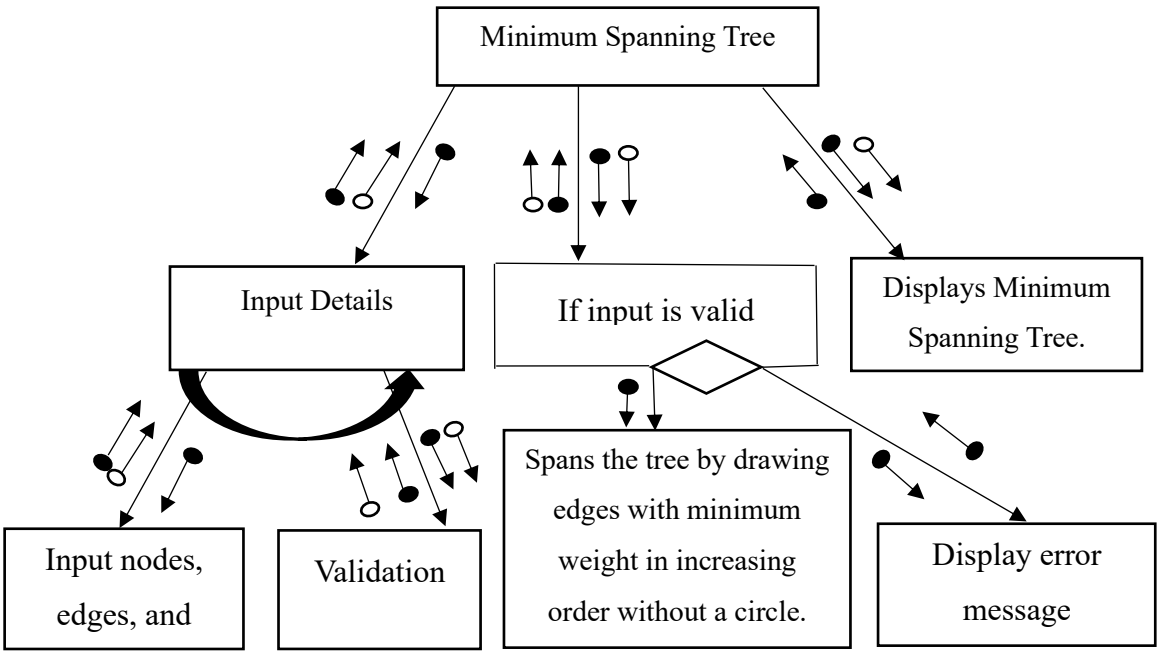


Figure 4.9 Minimum Spanning Tree

4.3.9.3 File I/O Interface: Not applicable

4.3.9.4 Output: Displays minimum spanning tree.

4.3.9.5 Implementation aspects: Canvas, Button, Frame, Tools, Textbox

4.3.10 Flooding Routing Algorithm

4.3.10.1 Input: Nodes, source node and data.

4.3.10.2 Procedural details

Structure Chart:

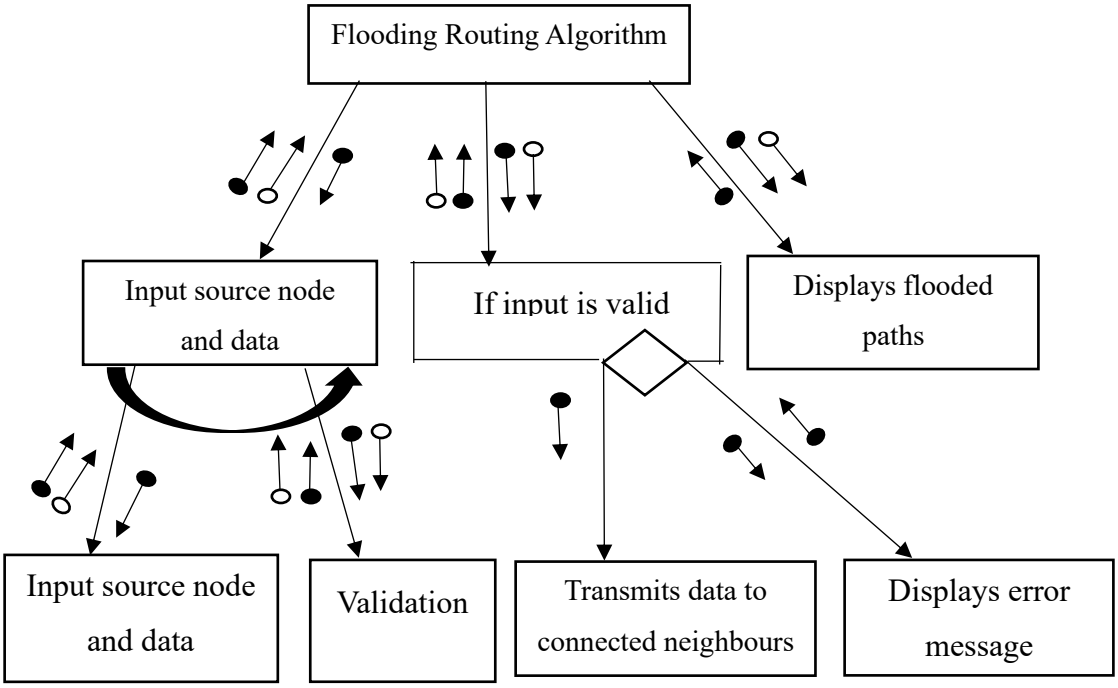


Figure 4.10 Flooding Routing Algorithm

4.3.10.3 File I/O Interface: Not applicable

4.3.10.4 Output: Highlighted flooded paths.

4.3.10.5 Implementation aspects: Canvas, Button, Frame, Tools, Textbox

4.3.11 Distance Vector Routing Algorithm

4.3.11.1 Input: Nodes, edges, source and destination nodes.

4.3.11.2 Procedural details

Algorithm:

Step 1: Start

Step 2: Input the nodes, edges with weight.

Step 3: Draw the network graph.

Step 4: Highlight the shortest path step by step.

Step 5: Shows Shortest distance between the source and destination nodes.

Step 6: Calculates the distance between the source and the destination node.

Step 7: Displays the shortest path highlighting in red and the distance calculated.

Step 8: End

Algorithm 4.3 Distance Vector Routing Algorithm

4.3.11.3 File I/O Interface: Not applicable

4.3.11.4 Output: Shortest distance between source and destination node.

4.3.11.5 Implementation aspects: Canvas, Button, Frame, Tools, Textbox

PROGRAM CODE LISTING

5. CODING

Simple Parity Check:

```
import turtle
import tkinter as tk
from tkinter import messagebox
import pyttsx3


def calculate_parity():
    binary_string = entry.get()


    if not binary_string:
        show_error_message("Please enter a binary string.")
        return


    if not all(bit in ('0', '1') for bit in binary_string):
        show_error_message("Binary string can only contain 0's and 1's.")
        return
    parity_bit = binary_string.count('1') % 2


    # Clear the screen
    turtle_screen.clear()


    # Calculate starting position for blocks
    num_bits = len(binary_string)
```

```
    block_size = 80
    total_width = num_bits * block_size
    x_start = -total_width / 2
    y = 0

    # Draw blocks
    for bit in binary_string:
        if bit == '1':
            color = 'green'
        else:
            color = 'red'
        draw_block(x_start, y, block_size, color, bit)
        x_start += block_size

    # Calculate position for parity bit
    x_parity = x_start + (block_size / 2)

    # Draw parity bit
    parity_label = "P=1" if parity_bit == 1 else "P=0"
    draw_block(x_parity, y, block_size, 'blue' if parity_bit == 1 else 'white', parity_label)

    # Add note to the turtle graphics
    note = "Note: If the number of 1's is odd, the parity bit is represented by a blue block labeled 'P=1'. Else, it is represented by a white block labeled 'P=0'."
    turtle.penup()
```

```
    turtle.goto(0, 200)
    turtle.pendown()
    turtle.color("teal")
    turtle.write(note, align='center', font=('Arial', 11, 'italic'))

    # Generate audio for the note
    engine = pyttsx3.init()
    engine.say(note)
    engine.runAndWait()

def draw_block(x, y, size, color, label):
    turtle.penup()
    turtle.goto(x, y)
    turtle.pendown()
    turtle.fillcolor(color)
    turtle.begin_fill()
    for _ in range(4):
        turtle.forward(size)
        turtle.right(90)
    turtle.end_fill()

    turtle.penup()
    turtle.goto(x + size / 2, y + size + 10)
    turtle.pendown()
```

```
    turtle.write(label, align='center', font=('Arial', 12, 'normal'))


def show_error_message(message):
    messagebox.showerror("Error", message)
    entry.delete(0, tk.END)  # Clear the entry widget


# Create GUI window
window = tk.Tk()
window.title("Parity Bit Simulation")
window.geometry("800x400")
window.configure(bg="teal")


# Create frame for user input
input_frame = tk.Frame(window)
input_frame.pack(side=tk.LEFT, padx=10)


# Create label for binary string
label = tk.Label(input_frame, text="Enter binary string:", font=("Times New Roman",
14))
label.pack()


# Create entry for binary string
entry = tk.Entry(input_frame, font=("Times New Roman", 14))
entry.pack(pady=20)  # Add vertical padding of 10 pixels
entry.focus()
```

```
# Create button to calculate parity
button = tk.Button(input_frame, text="Calculate", command=calculate_parity, bg="teal", font=("Times New Roman", 14))
button.pack()

# Create turtle graphics window
turtle_frame = tk.Frame(window)
turtle_frame.pack(side=tk.RIGHT, padx=10)

# Initialize turtle graphics
turtle_canvas = tk.Canvas(turtle_frame, width=1000, height=800)
turtle_canvas.pack()

turtle_screen = turtle.TurtleScreen(turtle_canvas)
turtle_screen.bgcolor("white")

turtle = turtle.RawTurtle(turtle_screen)
turtle.speed(3)

# Run GUI
window.mainloop()
```

Cyclic Redundancy Check:

```
import turtle
import tkinter as tk


def calculate_crc():
    """
    Calculates the CRC of a message using bitwise XOR operations.

    :return: binary CRC as a list of bits
    """
    bit = [int(x) for x in bit_entry.get()]
    poly = [int(x) for x in poly_entry.get()]

    x = 0  # Update initial value of x to 0
    y = 100
    x1 = 0  # Update initial value of x1 to 0
    y1 = 70
    y2 = 0
    x2 = 20
    # Append zeros to the message equal to the degree of the polynomial
    message = bit + [0] * (len(poly) - 1)
    for i in range(len(message)):
        turtle.penup()
```

```
    turtle.goto(x + i * 20, y)
    turtle.pendown()
    turtle.write(message[i], font=("Arial", 16, "normal"))
for i in range(len(poly)):
    turtle.penup()
    turtle.goto(x1 + i * 20, y1)
    turtle.pendown()
    turtle.write(poly[i], font=("Arial", 16, "normal"))

# Perform bitwise XOR of the message with the polynomial
for i in range(len(message) - len(poly) + 1):
    y1 += -50
    y2 += -50

    if message[i]:
        for j in range(1, len(poly)):  # Ignore the first bit of the polynomial
            message[i + j] ^= poly[j]
            print(message[i+j])
            turtle.penup()
            turtle.goto(x1 + i * 20, y1)
            turtle.pendown()
            turtle.write(message[i + j], font=("Arial", 16, "normal"))

            x1 += 20
```

```
            turtle.goto(x2 + i * 20, y2)
            turtle.pendown()
            turtle.write(poly[j], font=("Arial", 16, "normal"))
            x2 += 20

    # Return the CRC as the remainder of the final message
    crc_result = message[-len(poly) + 1:]
    crc_output.config(state='normal')
    crc_output.delete(1.0, tk.END)
    crc_output.insert(tk.END, "".join([str(x) for x in crc_result]))
    crc_output.config(state='disabled')

    # Append the CRC to the original message
    message += crc_result

    # Display the updated message in the message_output text box
    message_output.config(state='normal')
    message_output.delete(1.0, tk.END)
    message_output.insert(tk.END, "".join([str(x) for x in message]))
    message_output.config(state='disabled')

# Create a GUI window
root = tk.Tk()
root.title("CRC Calculator")
```

```
# Add bit input textbox and label
bit_label = tk.Label(root, text="Enter binary bit value:")
bit_label.grid(row=0, column=0)
bit_entry = tk.Entry(root)
bit_entry.grid(row=0, column=1)

# Add polynomial input textbox and label
poly_label = tk.Label(root, text="Enter polynomial:")
poly_label.grid(row=1, column=0)
poly_entry = tk.Entry(root)
poly_entry.grid(row=1, column=1)

# Add button to start CRC algorithm
calculate_button = tk.Button(root, text="Calculate CRC", command=calculate_crc)
calculate_button.grid(row=2, column=0, columnspan=2)

#Add output textbox and label for CRC value

crc_label = tk.Label(root, text="CRC:")
crc_label.grid(row=3, column=0)
crc_output = tk.Text(root, width=30, height=1, state='disabled')
crc_output.grid(row=3, column=1)
```

```
#Add output textbox and label for message with CRC value added


note_label = tk.Label(root, text="Message with CRC value added:")
note_label.grid(row=4, column=0)
message_output = tk.Text(root, width=30, height=1, state='disabled')
message_output.grid(row=4, column=1)


root.mainloop()
```

Path Finding Algorithm (Greedy Kruskal):

```
import networkx as nx
import matplotlib.pyplot as plt
from tkinter import messagebox
import re
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from tkinter import *


# Initialize the text-to-speech engine
import pyttsx3
engine = pyttsx3.init()


root = Tk()
root.geometry("1000x600")
root.configure(bg='peach puff')
```

```
G = nx.Graph()

nodes = []

edges = []


main_frame = Frame(root)

main_frame.pack(fill=BOTH, expand=True)

main_frame.configure(background='peach puff')  # Set the desired background color



graph_frame = Frame(main_frame, width=600)

graph_frame.pack(side=LEFT, padx=10, pady=10, fill=BOTH, expand=True)

graph_frame.configure(background='peach puff')  # Set the desired background color


calc_frame = Frame(main_frame, width=300)

calc_frame.pack(side=LEFT, padx=10, pady=10)


fig = plt.figure(figsize=(12, 14), dpi=120)

ax = fig.add_subplot(111)

canvas = None


root.after(1, lambda: root.focus_force())

root.after(2, lambda: textbox2.focus_set())
```

```
def addnode():
    node = textbox2.get()
    if not node:
        messagebox.showerror("Error", "Please enter a node.")
        return

    if not re.match("^[a-zA-Z]+$", node):
        messagebox.showerror("Error", "Node can only contain alphabets.")
        return
    nodes.append(node)
    G.add_node(node)
    textbox2.delete(0, END)
    listbox.insert(END, node)
    root.after(1, lambda: root.focus_force())
    root.after(2, lambda: textbox2.focus_set())
    draw_graph()


def addpath():
    selected_node = snode.get()
    if not selected_node:
        messagebox.showerror("Error", "Please enter a node.")
        return
```

```
if not re.match("^[a-zA-Z]+$", selected_node):
    messagebox.showerror("Error", "Node can only contain alphabets.")
    return

selected_indices = listbox.curselection()
if not selected_indices:
    messagebox.showerror("Error", "Please select a destination node.")
    return

selected_index = selected_indices[0]  # Assuming only single selection is allowed
selected_node1 = listbox.get(selected_index)
snode.delete(0, END)  # Clear the select node entry widget
# Set the focus to the select node entry widget after a small delay
root.after(10, lambda: snode.focus_set())

weight = wt.get().strip()
if not weight:
    messagebox.showerror("Error", "Please enter a weight.")
    return

if not re.match("^[0-9]+$", weight):
    messagebox.showerror("Error", "Weight can only contain numerical values.")
    return
```

```
    weight = int(weight)

    wt.delete(0, END)  # Clear the weight entry widget
    root.after(1, lambda: root.focus_force())
    root.after(2, lambda: wt.focus_set())

    G.add_edge(selected_node, selected_node1, weight=weight)
    edges.append((selected_node, selected_node1, weight))

    draw_graph()

    # Set the focus on the select node entry widget when the program starts
    root.after(10, lambda: snode.focus_set())



def draw_graph():
    global G, nodes, edges, canvas, fig, ax

    # Clear the canvas if it exists
    if canvas:
        canvas.get_tk_widget().destroy()
```

```
    # Create a new canvas
    canvas = FigureCanvasTkAgg(fig, master=graph_frame)
    canvas.get_tk_widget().pack()

    # Clear the previous graph
    ax.clear()

    # Draw the network graph
    pos = nx.spring_layout(G, seed=42)
    nx.draw(G, pos, with_labels=True, ax=ax)
    nx.draw_networkx_edge_labels(
        G,
        pos,
        edge_labels={(u, v): d["weight"] for u, v, d in G.edges(data=True)},
        font_size=12,
        font_color="Blue",
        ax=ax,
    )

    # Redraw the canvas
    canvas.draw()


def find_path(source, destination):
```

```
    mst = nx.minimum_spanning_tree(G, algorithm="kruskal", weight="weight")
    try:
        path = nx.shortest_path(mst, source=source, target=destination)
        return path
    except nx.NetworkXNoPath:
        return None



def highlight_path(path, pos):
    if path:
        for i in range(len(path) - 1):
            edge = (path[i], path[i + 1])
            nx.draw_networkx_edges(G, pos, edgelist=[edge], edge_color="r", width=5,
ax=ax)
            canvas.draw()
            root.update()
            engine.say(f"Going from {edge[0]} to {edge[1]} with weight
{G[edge[0]][edge[1]]['weight']}")
            engine.runAndWait()
            root.after(1000)
    else:
        engine.say("No path exists between the source and destination.")
        engine.runAndWait()
```

```
def find_and_highlight_path():
    source = wt1.get()
    destination = wt2.get()
    if not source or not destination:
        messagebox.showerror("Error", "Please enter source and destination nodes.")
        return

    if not re.match("^[a-zA-Z]+$", source) or not re.match("^[a-zA-Z]+$", destination):
        messagebox.showerror("Error", "Source and destination nodes can only contain alphabets.")
        return
    path = find_path(source, destination)
    pos = nx.spring_layout(G, seed=42)
    highlight_path(path, pos)

    # Display the note in the same window
    note_text = "The path between {} and {} is highlighted in red using the greedy Kruskal algorithm,\n".format(source, destination)
    note_text += "in which we find the minimum spanning tree (MST) and traverse it using Depth First Search (DFS) to find the shortest path."

    note_label = Label(graph_frame, text=note_text, font=("Arial", 17, "italic"), wraplength=500)
    note_label.place(x=110, y=8)  # Adjust the position of the label
```

```
    root.after(1000, read_note, note_text)  # Delay execution and call read_note after 1000 milliseconds


def read_note(note_text):
    engine.say(note_text)
    engine.runAndWait()



label1 = Label(calc_frame, text="Enter Node:", font=("Times New Roman", 18))
label1.pack(side=TOP, pady=(10, 10))

textbox2 = Entry(calc_frame, font=("Times New Roman", 16))
textbox2.pack(side=TOP, pady=(10, 10))

but1 = Button(calc_frame, text="Add Node", command=addnode, font=("Times New Roman", 16))
but1.pack(side=TOP, pady=(10, 10))

listbox = Listbox(calc_frame, width=25, height=10, font=("Times New Roman", 16))
listbox.pack(side=TOP, pady=(10, 10))



label2 = Label(calc_frame, text="Select node from listbox:", font=("Times New Roman", 16))
label2.pack(side=TOP, pady=(10, 10))
```

```
snode = Entry(calc_frame, font=("Times New Roman", 16))
snode.pack(side=TOP, pady=(10, 10))

label3 = Label(calc_frame, text="Weight:", font=("Times New Roman", 16))
label3.pack(side=TOP, pady=(10, 10))

wt = Entry(calc_frame, font=("Times New Roman", 16))
wt.pack(side=TOP, pady=(10, 10))

but2 = Button(calc_frame, text="Add Path", command=addpath, font=("Times New
Roman", 16))
but2.pack(side=TOP, pady=(10, 10))

label4 = Label(calc_frame, text="Source Node:", font=("Times New Roman", 16))
label4.pack(side=TOP, pady=(10, 10))

wt1 = Entry(calc_frame, font=("Times New Roman", 16))
wt1.pack(side=TOP, pady=(10, 10))

label5 = Label(calc_frame, text="Destination Node:", font=("Times New Roman", 16))
label5.pack(side=TOP, pady=(10, 10))

wt2 = Entry(calc_frame, font=("Times New Roman", 16))
wt2.pack(side=TOP, pady=(10, 10))
```

```
but3 = Button(calc_frame, text="Find", command=find_and_highlight_path,
font=("Times New Roman", 16))

but3.pack(side=TOP, pady=(10, 10))


root.mainloop()
```

Shortest Path First (Dijkstra's):

```
import networkx as nx

import matplotlib.pyplot as plt

from tkinter import messagebox

import re

from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

import pyttsx3

from tkinter import *


# Initialize the text-to-speech engine

engine = pyttsx3.init()


root = Tk()

root.geometry("1000x600")


G = nx.Graph()

nodes = []

edges = []
```

```
selected_items = ""

selected_items1 = ""

l = []


main_frame = Frame(root)

main_frame.pack(fill=BOTH, expand=True)

main_frame.configure(background='teal')  # Set the desired background color




# Create the frame for graph visualization

graph_frame = Frame(main_frame, width=600)

graph_frame.pack(side=RIGHT, padx=10,pady=10, fill=BOTH, expand=True)




# Create the right frame for user input

left_frame = Frame(main_frame, width=300)

left_frame.pack(side=LEFT, padx=10,pady=10)


# Create the figure and canvas

fig = plt.figure(figsize=(10, 10), dpi=100)

ax = fig.add_subplot(111)

canvas = None
```

```
root.after(1, lambda: root.focus_force())
root.after(2, lambda: textbox2.focus_set())



def addnode():
    node = textbox2.get()
    if not node:
        messagebox.showerror("Error", "Node can't be empty.")
        return
    if not re.match(r"^[A-Za-z]+$", node):
        messagebox.showerror("Error", "Node must contain only alphabetic characters.")
        return
    node = textbox2.get()
    nodes.append(node)
    G.add_node(node)
    textbox2.delete(0, END)
    listbox.insert(END, node)
    root.after(1, lambda: root.focus_force())
    root.after(2, lambda: textbox2.focus_set())
    draw_graph()

def addpath():
```

```
selected_node = snode.get()
if not selected_node:
    messagebox.showerror("Error", "Please select a node.")
    return
if not re.match(r"^[A-Za-z]+$", selected_node):
    messagebox.showerror("Error", "Node must contain only alphabetic characters.")
    return
selected_node1 = listbox.get(listbox.curselection())
snode.delete(0, END)  # Clear the select node entry widget
# Set the focus to the select node entry widget after a small delay
root.after(10, lambda: snode.focus_set())
weight = wt.get()
if not weight or not weight.isdigit():
    messagebox.showerror("Error", "Please enter a valid weight (integer).")
    return
weight = int(weight)
G.add_edge(selected_node, selected_node1, weight=weight)
edges.append((selected_node, selected_node1, weight))
wt.delete(0, END)
root.after(1, lambda: root.focus_force())
root.after(2, lambda: wt.focus_set())
draw_graph()
# Set the focus on the select node entry widget when the program starts
root.after(10, lambda: snode.focus_set())
```

```
def draw_graph():

    global G, nodes, edges, canvas, fig, ax

    # Clear the canvas if it exists
    if canvas:
        canvas.get_tk_widget().destroy()

   # Create a new canvas
    canvas = FigureCanvasTkAgg(fig, master=graph_frame)
    canvas.get_tk_widget().pack()

    # Clear the previous graph
    ax.clear()

    # Draw the network graph
    pos = nx.spring_layout(G, seed=42)
    nx.draw(G, pos, with_labels=True, ax=ax)
    nx.draw_networkx_edge_labels(G, pos, edge_labels={(u, v): d['weight'] for u, v, d in
G.edges(data=True)},
                                 font_size=12, font_color='Blue', ax=ax)

    # Redraw the canvas
    canvas.draw()
```

```
def shortestpath():
    sn = wt1.get()
    dn = wt2.get()
    if not sn or not dn:
        messagebox.showerror("Error", "Please enter valid source and destination nodes.")
        return
    if not re.match(r"^[A-Za-z]+$", sn) or not re.match(r"^[A-Za-z]+$", dn):
        messagebox.showerror("Error", "Node must contain only alphabetic characters.")
        return
    if sn and dn:
        pos = nx.spring_layout(G, seed=42)
        shortest_path = nx.dijkstra_path(G, sn, dn, weight='weight')
        shortest_path_edges = list(zip(shortest_path, shortest_path[1:]))
        path_length = nx.dijkstra_path_length(G, sn, dn, weight='weight')

        # Highlight the shortest path step by step
        for i in range(len(shortest_path_edges)):
            edge = shortest_path_edges[i]
            node1, node2 = edge
            nx.draw_networkx_edges(G, pos, edgelist=[edge], edge_color='r', width=5,
ax=ax)

            for u, v, d in G.edges(data=True):
                if (u, v) in shortest_path_edges[:i + 1] or (v, u) in shortest_path_edges[:i + 1]:
```

```
            x = (pos[u][0] + pos[v][0]) / 2
            y = (pos[u][1] + pos[v][1]) / 2
            ax.text(x, y, f"{d['weight']}", color='r')

        canvas.draw()
        root.update()
        engine.say(f"Going from {node1} to {node2} with weight {G[node1][node2]['weight']}")
        engine.runAndWait()

        # Pause for a moment to visualize the step
        root.after(1000)

    ax.text(0.05, 0.97, f"Shortest path between {sn} and {dn} is calculated using Dijkstra's algorithm and is highlighted in red ", fontfamily="Times New Roman", fontsize=17,
            color='teal', weight='bold', ha='left', va='top', transform=fig.transFigure, fontstyle='italic')

    ax.text(0.05, 0.92, f"Here it calculates the Shortest Best Path by summing the edge weights ", fontfamily="Times New Roman", fontsize=18,
            color='brown', weight='bold', ha='left', va='top', transform=fig.transFigure, fontstyle='italic')
```

```
        ax.text(0.05, 0.88, f"and highlights the path whose sum is less compared to the other paths ", fontfamily="Times New Roman", fontsize=18,
                color='brown', weight='bold', ha='left', va='top', transform=fig.transFigure, fontstyle='italic')

        # Display the total distance at the top left corner
        ax.text(0.05, 0.83, f"Total Distance here is  {path_length} which is considered as the best path", fontfamily="Times New Roman", fontsize=18,
                color='teal', weight='bold', ha='left', va='top', transform=fig.transFigure, fontstyle='italic')


        # Read the total distance aloud
        engine.say("Shortest path between the {} and {} is calculted using djkstraa's slgorithm and is highlighted in red".format(sn, dn, path_length))
        engine.say("Here it calculates the Shortest Best Path by summing the edge weights and highlights the path whose sum is less compared to the other paths ")
        engine.say(" Total Distance here is  {} which is considered as the best path".format(path_length))
        engine.runAndWait()

        # Redraw the canvas
        canvas.draw()
    else:
        engine.say("Please enter valid source and destination nodes.")
        engine.runAndWait()
```

```
# Create labels and input widgets in the right frame
label2 = Label(left_frame, font=("Times New Roman", 17), text="NODE NAME")
label2.grid(row=1, column=0)

textbox2 = Entry(left_frame, width=30)
textbox2.grid(row=1, column=1)

but = Button(left_frame, text="Add",font=("Times New Roman", 17),
command=addnode)
but.grid(row=2, column=1)

label11 = Label(left_frame, text="From-node", font=("Times New Roman", 17))
label11.grid(row=3, column=0)

snode = Entry(left_frame, width=30)
snode.grid(row=4, column=0)

labelto = Label(left_frame, text="To-node", font=("Times New Roman", 17))
labelto.grid(row=3, column=1)

listbox = Listbox(left_frame, width=25, height=10, font=("Times New Roman", 16))
listbox.grid(row=4, column=1)
```

```
labelw = Label(left_frame, text="Weight", font=("Times New Roman", 17))
labelw.grid(row=3, column=2)

wt = Entry(left_frame, width=30)
wt.grid(row=4, column=2)

labelw1 = Label(left_frame, text="SOURCE", font=("Times New Roman", 17))
labelw1.grid(row=5, column=0)

wt1 = Entry(left_frame, width=30)
wt1.grid(row=6, column=0)

labelw2 = Label(left_frame, text="DESTINATION", font=("Times New Roman", 17))
labelw2.grid(row=5, column=2)

wt2 = Entry(left_frame, width=30)
wt2.grid(row=6, column=2)

but1 = Button(left_frame, text="Add_Path", font=("Times New Roman", 17),
command=addpath)
but1.grid(row=7, column=1)

but2 = Button(left_frame, text="createGraph", font=("Times New Roman", 17),
command=shortestpath)
```

```
but2.grid(row=10, column=1)


root.mainloop()
```

Minimum Spanning Tree(Greedy Kruskal):

```
import networkx as nx
import matplotlib.pyplot as plt
import tkinter as tk
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from tkinter import messagebox
from tkinter import ttk
import re



def calculate_spanning_tree(network_graph):
    # Perform MST calculations
    spanning_tree = nx.minimum_spanning_tree(network_graph)
    return spanning_tree



def create_network_topology(nodes, edges_with_weights):
    network_graph = nx.Graph()
    network_graph.add_nodes_from(nodes)
    for edge, weight in edges_with_weights:
```

```
        node1, node2 = edge
        network_graph.add_edge(node1, node2, weight=float(weight))
    return network_graph



def display_network_topology(network_graph, pos, spanning_tree=None):
    if spanning_tree:
        # Create a list of edges to be drawn as a circle
        circle_edges = []

        # Create a list of edges to be drawn as non-circle edges
        non_circle_edges = []

        # Iterate over edges and classify them based on the presence in the spanning tree
        for edge in network_graph.edges():
            if edge in spanning_tree.edges():
                circle_edges.append(edge)
            else:
                non_circle_edges.append(edge)

        # Draw the circle edges
        nx.draw_networkx_edges(network_graph, pos, edgelist=circle_edges,
edge_color='gray', alpha=0.3)

        # Draw the non-circle edges
```

```
        nx.draw_networkx_edges(network_graph, pos, edgelist=non_circle_edges,
edge_color='gray')

    else:
        # Draw all edges if no spanning tree is provided
        nx.draw_networkx_edges(network_graph, pos, edge_color='gray')

    nx.draw_networkx_nodes(network_graph, pos, node_color='lightblue', node_size=500)
    nx.draw_networkx_labels(network_graph, pos)

    labels = nx.get_edge_attributes(network_graph, 'weight')
    nx.draw_networkx_edge_labels(network_graph, pos, edge_labels=labels)


def create_output_screen(node_positions):
    global figure1, figure2, combined_figure, output_window, figure1_canvas,
figure2_canvas

    # Create the output window
    output_window = tk.Toplevel(root)
    output_window.title("Output")
    output_window.configure(background = 'light green')

    # Set the dimensions and position of the output window to center it on the screen
    screen_width = output_window.winfo_screenwidth()
```

```
screen_height = output_window.winfo_screenheight()
window_width = 800  # Adjust the width as needed
window_height = 600  # Adjust the height as needed
x = (screen_width - window_width) // 2
y = (screen_height - window_height) // 2
output_window.geometry(f"{window_width}x{window_height}+{x}+{y}")

# Create a combined figure to hold both network topology and minimum spanning tree
combined_figure = plt.figure(figsize=(10, 5))

# Plot the network topology on the left side
network_topology_ax = combined_figure.add_subplot(121)
display_network_topology(network_topology, node_positions)
network_topology_ax.set_title('Network Topology')

# Plot the minimum spanning tree on the right side
spanning_tree_ax = combined_figure.add_subplot(122)
display_network_topology(spanning_tree, node_positions)
spanning_tree_ax.set_title('Minimum Spanning Tree')

# Create a canvas for the combined figure
figure_canvas = FigureCanvasTkAgg(combined_figure, master=output_window)
figure_canvas.get_tk_widget().pack()
```

```
    # Add a note to the minimum spanning tree plot
    spanning_tree_ax.text(
        -0.1,
        -0.1,
        "Note: Minimum Spanning Tree calculated by placing the edges in the increasing order of weights,\nbut the path should not form a closed structure",
        horizontalalignment='center',
        verticalalignment='center',
        transform=spanning_tree_ax.transAxes,
        fontsize=10,
        color='teal',
        style='italic'
    )

    # Update the output window
    output_window.update()


def display_output_screen(spanning_tree, node_positions):
    # Clear the combined figure
    combined_figure.clear()

    # Plot the network topology on the left side
    network_topology_ax = combined_figure.add_subplot(121)
    display_network_topology(network_topology, node_positions)
```

```
    network_topology_ax.set_title('Network Topology')

    # Plot the minimum spanning tree on the right side
    spanning_tree_ax = combined_figure.add_subplot(122)
    display_network_topology(spanning_tree, node_positions)
    spanning_tree_ax.set_title('Minimum Spanning Tree')

    # Refresh the canvas
    figure_canvas.draw()


def main():
    global root, output_window, nodes, edges_with_weights, num_edges, edges_entry, weights_entry, network_topology, spanning_tree, figure_canvas, node_positions

    # Create the GUI
    root = tk.Tk()
    root.title("Minimum Spanning Tree")
    root.configure(background='light blue')

    nodes = []
    edges_with_weights = []
    num_edges = 0
    edges_entry = []
    weights_entry = []
```

```
network_topology = None
spanning_tree = None
figure_canvas = None
node_positions = None


def add_node():
    node = node_entry.get()
    if not node:
        messagebox.showerror("Error", "Please enter a node!")
        return
    if not re.match("^[a-zA-Z]$", node):
        messagebox.showerror("Error", "Enter a valid node name")
        return


    nodes.append(node)
    node_entry.delete(0, tk.END)
    display_network()


def add_edge():
    global num_edges


    num_edges += 1


    # Create edge entry
```

```
        edge_label = tk.Label(root, text=f"Enter edge {num_edges} in the format 'node1 node2':", font=("Times New Roman", 12, "italic"), fg="teal",)
        edge_label.pack()
        edge_entry = tk.Entry(root, font=("Times New Roman", 12, "italic"), fg="black",)
        edge_entry.pack()
        edges_entry.append(edge_entry)
        edge_entry.focus_set()  # Set focus on the edge entry box

        # Create weight entry
        weight_label = tk.Label(root, text=f"Enter the weight for edge {num_edges}:", font=("Times New Roman", 12, "italic"), fg="teal",)
        weight_label.pack()
        weight_entry = tk.Entry(root, font=("Times New Roman", 12, "italic"), fg="black",)
        weight_entry.pack()
        weights_entry.append(weight_entry)

    def validate_edges():
        for i in range(num_edges):
            edge = edges_entry[i].get().split()
            if len(edge) != 2 or not re.match("^[a-zA-Z]+$", edge[0]) or not re.match("^[a-zA-Z]+$", edge[1]):
                messagebox.showerror("Error", "Enter valid values for edges (format: 'node1 node2')")
                return False
        return True
```

```
    def validate_weights(event):
        weight_entry = event.widget
        weight = weight_entry.get()
        if not re.match("^[0-9]*\.?[0-9]+$", weight):
            messagebox.showerror("Error", "Enter a valid weight value (numeric)")
            weight_entry.focus_set()

    def create_topology_and_calculate():
        global network_topology, node_positions, spanning_tree

        # Validate if any entry label is empty
        for i in range(num_edges):
            if not edges_entry[i].get() or not weights_entry[i].get():
                messagebox.showerror("Error", "Please enter a value for all edges and
weights!")
                return

        # Validate the edge format
        if not validate_edges():
            return

        # Validate the weight format
        for weight_entry in weights_entry:
            weight = weight_entry.get()
```

```
            if not re.match("^[0-9]*\.?[0-9]+$", weight):
                messagebox.showerror("Error", "Enter a valid weight value (numeric)")
                return

        # Create the network topology
        edges_with_weights.clear()
        for i in range(num_edges):
            edge = edges_entry[i].get().split()
            weight = weights_entry[i].get()
            edges_with_weights.append((edge, weight))
        network_topology = create_network_topology(nodes, edges_with_weights)
        node_positions = nx.spring_layout(network_topology, seed=42)  # Use a fixed seed for consistent layout

        # Calculate the spanning tree
        spanning_tree = calculate_spanning_tree(network_topology)

        # Create and display the output screen
        create_output_screen(node_positions)

    def display_network():
        # Clear previous display
        for widget in root.winfo_children():
            if (
                widget != add_node_frame
```

```
            and widget != add_edge_frame
            and widget != calculate_frame
        ):
            widget.pack_forget()

    # Display nodes
    nodes_label = tk.Label(root, text="Nodes:", font=("Times New Roman", 12, "italic"), fg="teal",)
    nodes_label.pack()
    for node in nodes:
        node_text = tk.Label(root, text=node, font=("Times New Roman", 12, "italic"), fg="black")
        node_text.pack()

    # Display edges
    edges_label = tk.Label(root, text="Edges:", font=("Times New Roman", 12, "italic"), fg="teal",)
    edges_label.pack()
    for i, edge_entry in enumerate(edges_entry):
        edge_text = tk.Label(root, text=f"{edge_entry.get()} (Weight: {weights_entry[i].get()})", font=("Times New Roman", 12, "italic"), fg="teal",)
        edge_text.pack()

# Add Node Frame
add_node_frame = tk.Frame(root)
```

```
    add_node_frame.pack()


    node_label = tk.Label(add_node_frame, text="Add a node:", font=("Times New
Roman", 12, "italic"), fg="teal",)
    node_label.pack(side=tk.LEFT)
    node_entry = tk.Entry(add_node_frame, font=("Times New Roman", 12, "italic"),
fg="black",)
    node_entry.pack(side=tk.LEFT)
    node_entry.focus_set()
    add_node_button = tk.Button(add_node_frame, text="Add", font=("Times New
Roman", 12, "italic"), fg="teal", command=add_node)
    add_node_button.pack(side=tk.LEFT)


    # Add Edge Frame
    add_edge_frame = tk.Frame(root)
    add_edge_frame.pack()


    add_edge_button = tk.Button(add_edge_frame, text="Add an edge", font=("Times
New Roman", 12, "italic"), fg="teal", command=add_edge)
    add_edge_button.pack(side=tk.LEFT)


    # Calculate Frame
    calculate_frame = tk.Frame(root)
    calculate_frame.pack()
```

```
    calculate_button = tk.Button(
        calculate_frame,
        text="Create Network Topology and Calculate Minimum Spanning
Tree",font=("Times New Roman", 12, "italic"), fg="teal",
        command=create_topology_and_calculate
    )
    calculate_button.pack()

    # Run the main Tkinter event loop
    root.mainloop()

if __name__ == "__main__":
    main()
```

Flooding Routing Algorithm:

```
import random
import math
import tkinter as tk
import turtle
from tkinter import messagebox

class StdoutRedirector:
    def __init__(self, text_widget):
        self.text_widget = text_widget
```

```
    def write(self, text):
        self.text_widget.insert(tk.END, text)
        self.text_widget.see(tk.END)

# Define the network topology as a dictionary
network = {}

# Store the node positions globally
node_positions = {}

# Store the edge IDs globally
edge_ids = {}

# Flag to check if the topology is created
topology_created = False

# Set a random seed for consistent network structures
random.seed(42)

def create_network_topology(num_nodes):
    global network, node_positions, edge_ids, topology_created
    # Store the network topology, node positions, and edge IDs globally
    network = {}
```

```
    # Generate node labels
    nodes = [chr(ord('A') + i) for i in range(num_nodes)]

    # Create random connections between nodes
    for node in nodes:
        # Ensure each node is connected to at least one other node
        neighbors = set(nodes) - {node}
        num_neighbors = random.randint(1, num_nodes - 1)  # Randomly select the number of neighbors
        network[node] = random.sample(list(neighbors), num_neighbors)

    # Fixed positions for the nodes
    node_positions = {}
    angle = 360 / num_nodes
    radius = 200
    for i, node in enumerate(nodes):
        x = radius * math.cos(math.radians(i * angle))
        y = radius * math.sin(math.radians(i * angle))
        node_positions[node] = (x, y)

    turtle.clear()

    edge_ids = {}
    for node in network:
```

```
turtle.penup()
turtle.goto(*get_center(node_positions[node]))
turtle.pendown()
turtle.fillcolor('blue')
turtle.begin_fill()
for _ in range(4):
    turtle.forward(20)
    turtle.right(90)
turtle.end_fill()

# Label the nodes with letters
turtle.penup()
turtle.goto(*get_center(node_positions[node]))
turtle.pendown()
turtle.write(node, align='center', font=('Arial', 12, 'normal'))

for neighbor in network[node]:
    edge_ids[(node, neighbor)] = (node_positions[node], node_positions[neighbor])
    edge_ids[(neighbor, node)] = (node_positions[neighbor], node_positions[node])

    # Draw lines between nodes
    turtle.penup()
    turtle.goto(*get_center(node_positions[node]))
    turtle.pendown()
```

```
            turtle.color('black')
            turtle.speed(4)
            turtle.goto(*get_center(node_positions[neighbor]))
            turtle.speed(0)
            turtle.penup()  # Add this line to lift the pen after drawing the line
            turtle_screen.update()

    topology_created = True
    show_simulation_controls()

def get_center(position):
    if position:
        return position[0], position[1]
    else:
        return 0, 0
def create_topology_button_click():
    num_nodes = num_nodes_entry.get()
    if not num_nodes:
        messagebox.showerror("Error", "Number of nodes can't be empty.")
        return

    try:
        num_nodes = int(num_nodes)
    except ValueError:
```

```
            messagebox.showerror("Error", "Number of nodes must be a valid integer.")
            return

        create_network_topology(num_nodes)

    def show_simulation_controls():
        # Clear the existing controls
        for widget in simulation_controls_frame.winfo_children():
            widget.pack_forget()

        num_nodes_label.pack(in_=simulation_controls_frame, pady=5)
        num_nodes_entry.pack(in_=simulation_controls_frame, pady=5)
        create_topology_button.pack(in_=simulation_controls_frame, pady=5)

        if topology_created:  # Show the controls only if the topology is created
            source_label.pack(in_=simulation_controls_frame, pady=5)
            source_entry.pack(in_=simulation_controls_frame, pady=5)
            data_label.pack(in_=simulation_controls_frame, pady=5)
            data_entry.pack(in_=simulation_controls_frame, pady=5)
            run_button.pack(in_=simulation_controls_frame, pady=5)

            # Set focus on the source entry box
            source_entry.focus_set()
        else:
```

```
        # Set focus on the num_nodes_entry box
        num_nodes_entry.focus_set()

    turtle_screen.update()

def highlight_path(src, dest):
    if (src, dest) in edge_ids:
        turtle.penup()
        turtle.color('red')
        turtle.speed(1)
        turtle.goto(*edge_ids[(src, dest)][0])
        turtle.pendown()
        turtle.goto(*edge_ids[(src, dest)][1])
        turtle.penup()  # Add this line to lift the pen after drawing the path
        turtle_screen.update()
def flooding(src, data):
    # Perform flooding simulation
    flooded_nodes = set()
    queue = [src]
    while queue:
        node = queue.pop(0)
        if node not in flooded_nodes:
            flooded_nodes.add(node)
            queue.extend(network[node])  # Access the globally stored network
```

```
    return flooded_nodes


# Define a function to handle the button click event
def run_simulation():
    global topology_created
    # Check if both the source node and data fields are filled in
    if not source_var.get():
        messagebox.showerror("Error", "Source node can't be empty.")
        return
    if not data_entry.get():
        messagebox.showerror("Error", "Data can't be empty.")
        return

    # Check if the source node is a capital letter
    source_node = source_var.get()
    if not source_node.isalpha() or not source_node.isupper():
        messagebox.showerror("Error", "Source node must be a capital letter.")
        return

    # Check if the data is a valid integer
    data = data_entry.get()
    try:
        int(data)
    except ValueError:
```

```
        messagebox.showerror("Error", "Data must be a valid integer.")
        return

    # Get the selected source node and data from the GUI
    src = source_var.get()
    data = data_entry.get()

    # Clear the canvas and create the network topology visualization if it hasn't been created already
    if not topology_created:
        turtle.clear()
        num_nodes = int(num_nodes_entry.get())
        create_network_topology(num_nodes)

    # Simulate the transmission of the data from the source node
    flooded_nodes = flooding(src, data)

    # Highlight the paths in red
    for node in flooded_nodes:
        if node != src:
            if (src, node) in edge_ids:
                highlight_path(src, node)

    # Set focus on the data entry box
    data_entry.focus_set()
```

```
    turtle_screen.update()


root = tk.Tk()
root.title("Flooding Routing Simulation")
root.geometry("900x800")  # Set the initial window size

# Allow both horizontal and vertical resizing
root.resizable(True, True)

# Custom styles
label_style = {'font': ('Times New Roman', 13, 'italic')}
entry_style = {'font': ('Times New Roman', 13, 'italic')}
button_style = {'font': ('Times New Roman', 13, 'italic')}

root.grid_rowconfigure(0, weight=1)
root.grid_columnconfigure(0, weight=1)

canvas_frame = tk.Frame(root)
canvas_frame.grid(row=0, column=0, sticky='nsew', padx=10, pady=8)

simulation_controls_frame = tk.Frame(root)
simulation_controls_frame.grid(row=0, column=1, sticky='nsew', padx=10, pady=10)
```

```
simulation_controls_frame.configure(background="mint cream")

canvas = tk.Canvas(canvas_frame, width=1200, height=1000)
canvas.pack(side=tk.LEFT)

turtle_screen = turtle.TurtleScreen(canvas)
turtle_screen.bgcolor("light green")

window_width = 1200
window_height = 1000
turtle_screen.screensize(window_width, window_height)
turtle_screen.setworldcoordinates(-window_width / 2, -window_height / 2,
window_width / 2, window_height / 2)

turtle = turtle.RawTurtle(turtle_screen)
turtle.speed(0)
turtle.hideturtle()

num_nodes_label = tk.Label(simulation_controls_frame, text="Enter number of nodes:",
fg="teal", **label_style)

num_nodes_entry = tk.Entry(simulation_controls_frame, **entry_style)

create_topology_button = tk.Button(simulation_controls_frame, text="Create Network
Topology", fg="teal",
```

```
                    command=create_topology_button_click, **button_style)

source_var = tk.StringVar()
source_label = tk.Label(simulation_controls_frame, text="Source Node:", fg="teal",
**label_style)

data_var = tk.StringVar()
data_label = tk.Label(simulation_controls_frame, text="Data:", fg="teal", **label_style)

source_entry = tk.Entry(simulation_controls_frame, textvariable=source_var,
**entry_style)
data_entry = tk.Entry(simulation_controls_frame, textvariable=data_var, **entry_style)
run_button = tk.Button(simulation_controls_frame, text="Run Simulation", fg="teal",
command=run_simulation,
                **button_style)
show_simulation_controls()
root.mainloop()
```

Distance Vector Routing Algorithm:

```
import networkx as nx
import matplotlib.pyplot as plt
from tkinter import messagebox
import re
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import pyttsx3
```

```
from tkinter import *

# Initialize the text-to-speech engine
engine = pyttsx3.init()

root = Tk()
root.geometry("1000x600")

G = nx.Graph()
nodes = []
edges = []

main_frame = Frame(root)
main_frame.pack(fill=BOTH, expand=True)
main_frame.configure(background='peach puff')  # Set the desired background color


graph_frame = Frame(main_frame, width=600)
graph_frame.pack(side=LEFT, padx=10, pady=10, fill=BOTH, expand=True)

calc_frame = Frame(main_frame, width=300)
calc_frame.pack(side=LEFT, padx=10, pady=10)

fig = plt.figure(figsize=(12, 14), dpi=120)
```

```
ax = fig.add_subplot(111)
canvas = None

root.after(1, lambda: root.focus_force())
root.after(2, lambda: textbox2.focus_set())



def addnode():
    node = textbox2.get().strip()
    if not node:
        messagebox.showerror("Error", "Please enter a node.")
        return

    if not re.match("^[a-zA-Z]+$", node):
        messagebox.showerror("Error", "Node can only contain alphabets.")
        return
    nodes.append(node)
    G.add_node(node)
    textbox2.delete(0, END)
    listbox.insert(END, node)
    root.after(1, lambda: root.focus_force())
    root.after(2, lambda: textbox2.focus_set())
    draw_graph()
```

```
def addpath():
    selected_node = snode.get()
    if not selected_node:
        messagebox.showerror("Error", "Please enter a node.")
        return

    if not re.match("^[a-zA-Z]+$", selected_node):
        messagebox.showerror("Error", "Node can only contain alphabets.")
        return

    selected_node1 = listbox.get(listbox.curselection())
    snode.delete(0, END)  # Clear the select node entry widget
    # Set the focus to the select node entry widget after a small delay
    root.after(10, lambda: snode.focus_set())

    weight = wt.get().strip()
    if not weight:
        messagebox.showerror("Error", "Please enter a weight.")
        return

    if not re.match("^[0-9]+$", weight):
        messagebox.showerror("Error", "Weight can only contain numerical values.")
        return
```

```
    weight = int(weight)

    wt.delete(0, END)  # Clear the weight entry widget
    root.after(1, lambda: root.focus_force())
    root.after(2, lambda: wt.focus_set())

    G.add_edge(selected_node, selected_node1, weight=weight)
    edges.append((selected_node, selected_node1, weight))

    draw_graph()

    # Set the focus on the select node entry widget when the program starts
    root.after(10, lambda: snode.focus_set())



def draw_graph():
    global G, nodes, edges, canvas, fig, ax

    # Clear the canvas if it exists
    if canvas:
        canvas.get_tk_widget().destroy()

   # Create a new canvas
    canvas = FigureCanvasTkAgg(fig, master=graph_frame)
```

```
    canvas.get_tk_widget().pack()

    # Clear the previous graph
    ax.clear()

    # Draw the network graph
    pos = nx.spring_layout(G, seed=42)
    nx.draw(G, pos, with_labels=True, ax=ax)
    nx.draw_networkx_edge_labels(G, pos, edge_labels={(u, v): d['weight'] for u, v, d in
G.edges(data=True)},
                                 font_size=12, font_color='Blue', ax=ax)

    # Redraw the canvas
    canvas.draw()

def shortestpath():
    sn = wt1.get()
    dn = wt2.get()
    if not sn or not dn:
        messagebox.showerror("Error", "Please enter source and destination nodes.")
        return

    if not re.match("^[a-zA-Z]+$", sn) or not re.match("^[a-zA-Z]+$", dn):
        messagebox.showerror("Error", "Source and destination nodes can only contain
alphabets.")
```

```
        return
    if sn and dn:
        pos = nx.spring_layout(G, seed=42)
        shortest_path = nx.dijkstra_path(G, sn, dn, weight='weight')
        shortest_path_edges = list(zip(shortest_path, shortest_path[1:]))
        path_length = nx.dijkstra_path_length(G, sn, dn, weight='weight')

        # Highlight the shortest path step by step
        for i in range(len(shortest_path_edges)):
            edge = shortest_path_edges[i]
            node1, node2 = edge
            nx.draw_networkx_edges(G, pos, edgelist=[edge], edge_color='r', width=5,
ax=ax)

            for u, v, d in G.edges(data=True):
                if (u, v) in shortest_path_edges[:i + 1] or (v, u) in shortest_path_edges[:i + 1]:
                    x = (pos[u][0] + pos[v][0]) / 2
                    y = (pos[u][1] + pos[v][1]) / 2
                    ax.text(x, y, f"{d['weight']}", color='r')

            canvas.draw()
            root.update()
            engine.say(f"Distance between {node1} to {node2} is
{G[node1][node2]['weight']}")
            engine.runAndWait()
```

```
            # Pause for a moment to visualize the step
            root.after(1000)

        # Display the total distance at the top left corner
        ax.text(0.05, 0.95, f"Total Distance: {path_length} which is determined by summing the edges assigned, ", fontfamily="Times New Roman", fontsize=15,
                color='black', weight='bold', ha='left', va='top', transform=fig.transFigure)

        # Read the total distance aloud
        engine.say("Total distance between {} and {} is {} which is determined by summing the edges assigned".format(sn, dn, path_length))
        engine.runAndWait()

        # Redraw the canvas
        canvas.draw()
    else:
        engine.say("Please enter valid source and destination nodes.")
        engine.runAndWait()

label1 = Label(calc_frame, text="Enter Node:", font=("Times New Roman", 18))
label1.pack(side=TOP, pady=(10, 10))

textbox2 = Entry(calc_frame, font=("Times New Roman", 16))
textbox2.pack(side=TOP, pady=(10, 10))
```

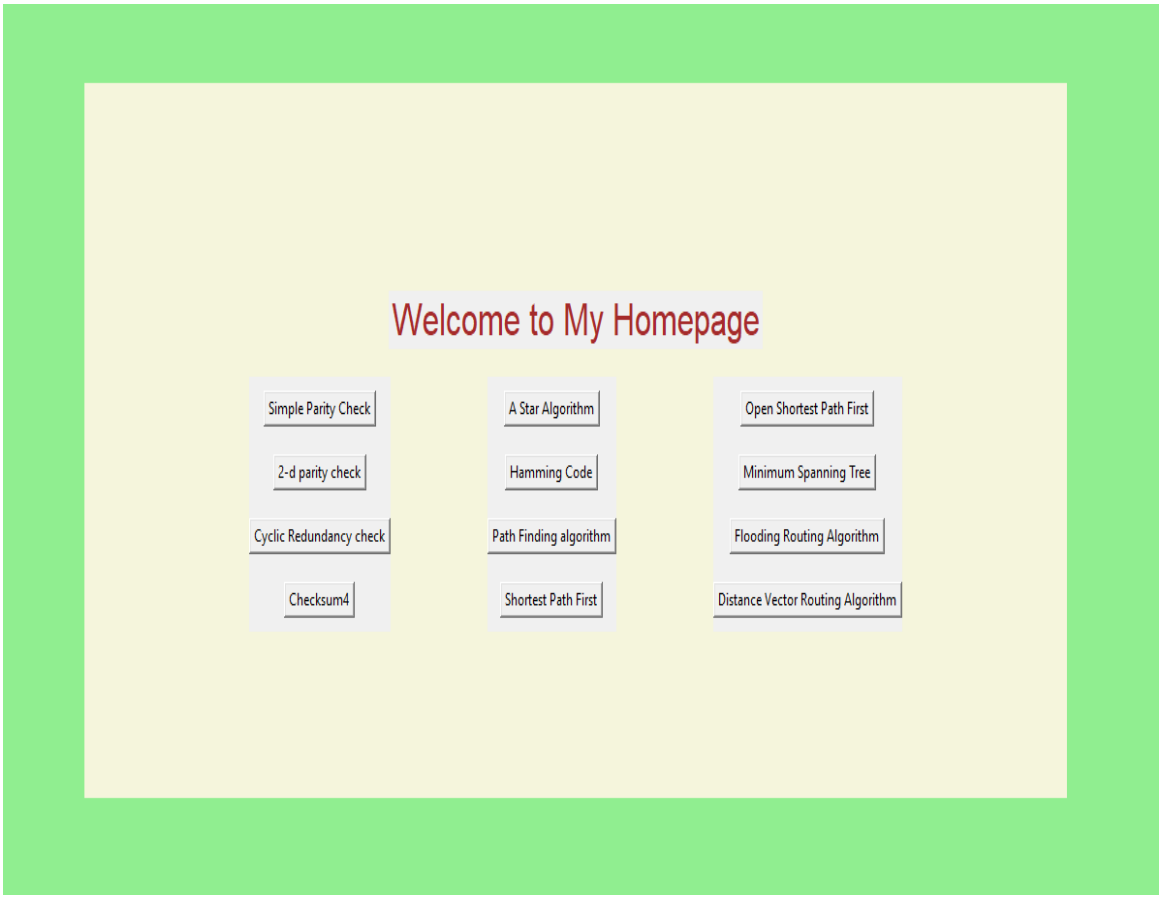
```
but1 = Button(calc_frame, text="Add Node", command=addnode, font=("Times New Roman", 16))
but1.pack(side=TOP, pady=(10, 10))

listbox = Listbox(calc_frame, width=25, height=10, font=("Times New Roman", 16))
listbox.pack(side=TOP, pady=(10, 10))


label2 = Label(calc_frame, text="Select node from listbox:", font=("Times New Roman", 16))
label2.pack(side=TOP, pady=(10, 10))

snode = Entry(calc_frame, font=("Times New Roman", 16))
snode.pack(side=TOP, pady=(10, 10))

label3 = Label(calc_frame, text="Weight:", font=("Times New Roman", 16))
label3.pack(side=TOP, pady=(10, 10))

wt = Entry(calc_frame, font=("Times New Roman", 16))
wt.pack(side=TOP, pady=(10, 10))

but2 = Button(calc_frame, text="Add Path", command=addpath, font=("Times New Roman", 16))
but2.pack(side=TOP, pady=(10, 10))
```

```
label4 = Label(calc_frame, text="Source Node:", font=("Times New Roman", 16))
label4.pack(side=TOP, pady=(10, 10))

wt1 = Entry(calc_frame, font=("Times New Roman", 16))
wt1.pack(side=TOP, pady=(10, 10))

label5 = Label(calc_frame, text="Destination Node:", font=("Times New Roman", 16))
label5.pack(side=TOP, pady=(10, 10))

wt2 = Entry(calc_frame, font=("Times New Roman", 16))
wt2.pack(side=TOP, pady=(10, 10))

but3 = Button(calc_frame, text="Find", command=shortestpath, font=("Times New
Roman", 16))
but3.pack(side=TOP, pady=(10, 10))

root.mainloop()
```

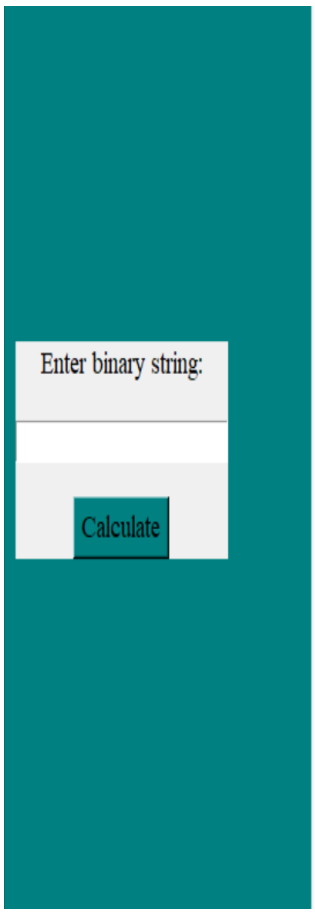
USER INTERFACE

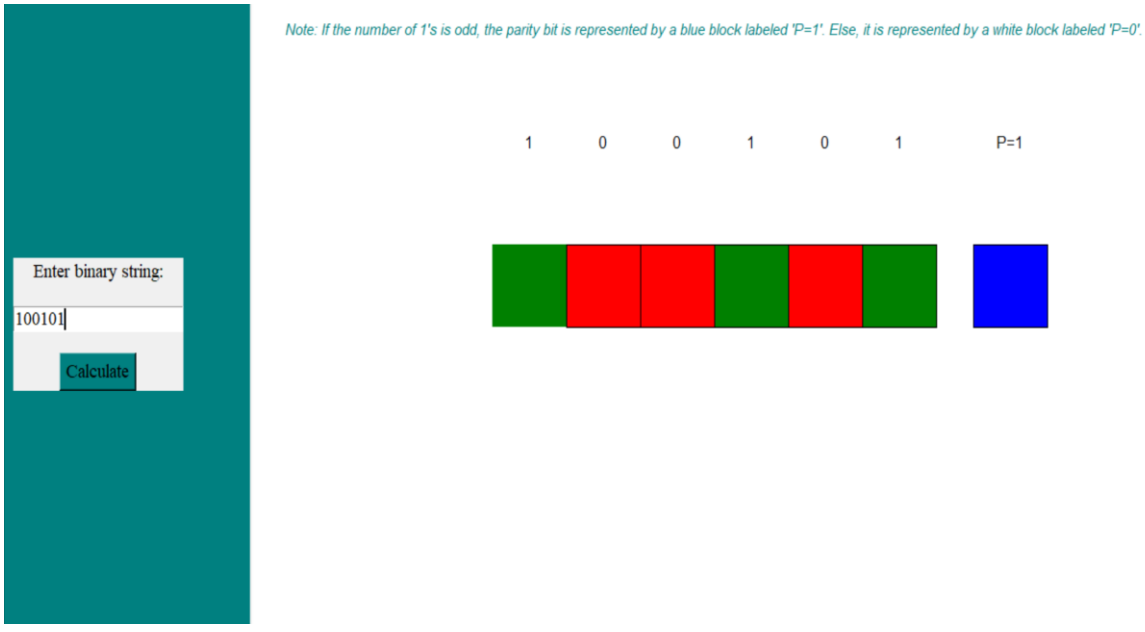
6. USER INTERFACE

Home Page

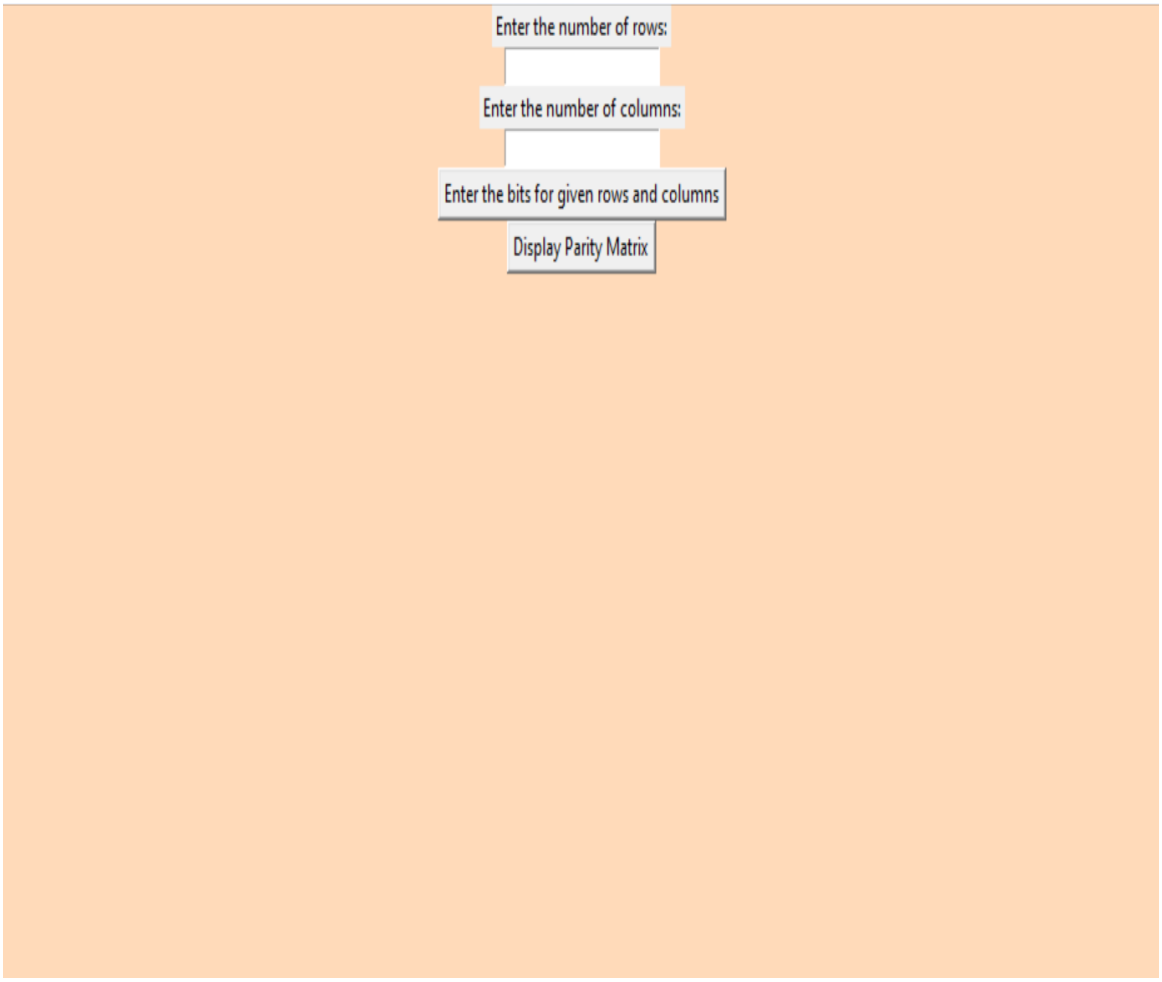


Simple Parity Check:





Two-Dimensional Parity Check:



User Data:
111 101 110

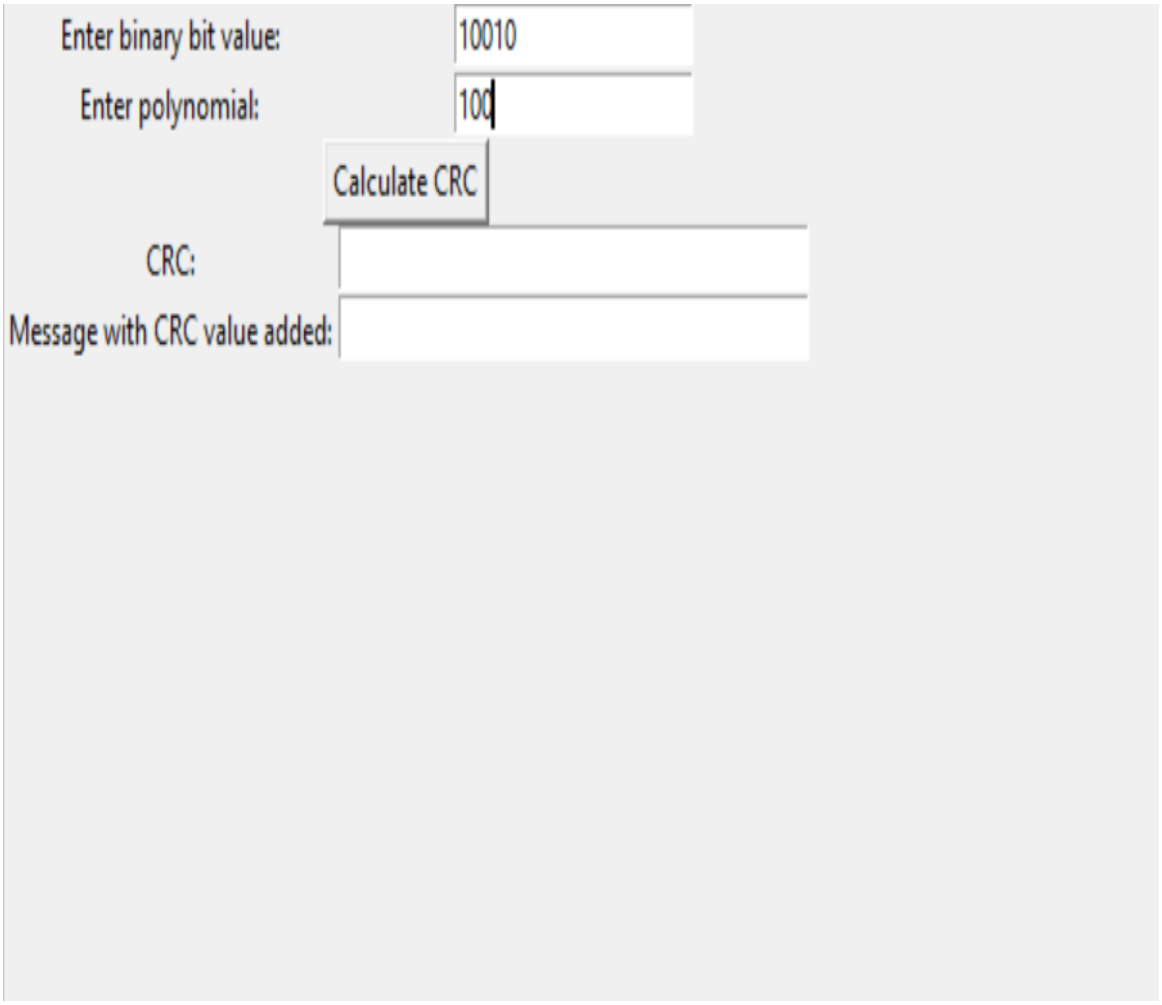
Note1 : Here the matrix is generated based on the number of 1's
If the number of one's is odd,the sum of each row and column is 1,otherwise 0
and the final bit which is LCR is generated by considering the sum of
1's in final row and column and it is ignored.

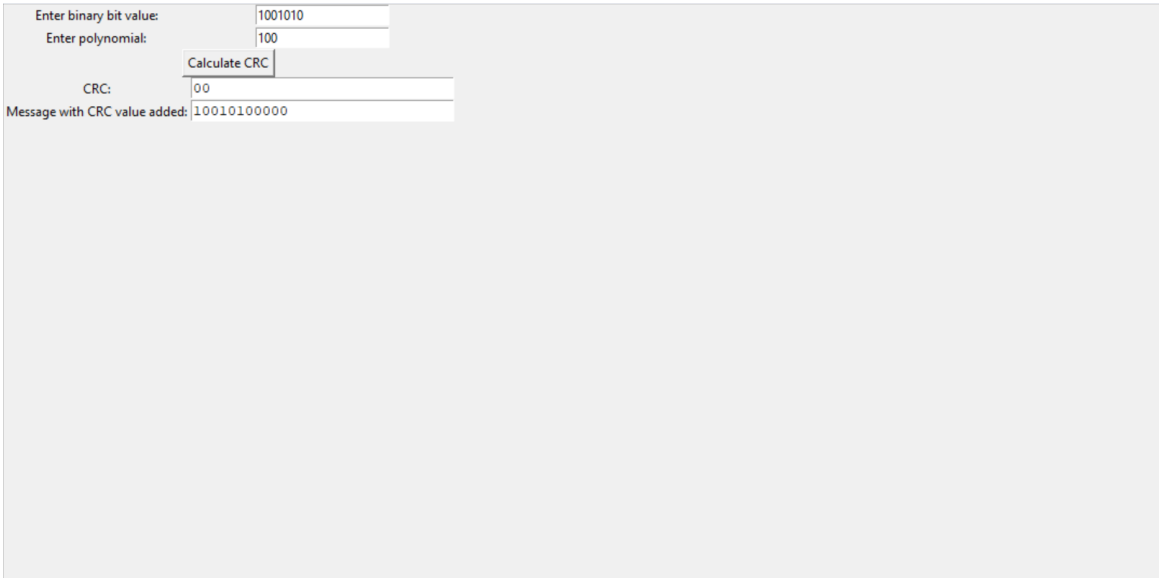
1	1	1	1
1	0	1	0
1	1	0	0
1	0	0	0

Note2 : Here in output data, the user data is appended with row parities and the sum of the last row last column is appended alongside with it.

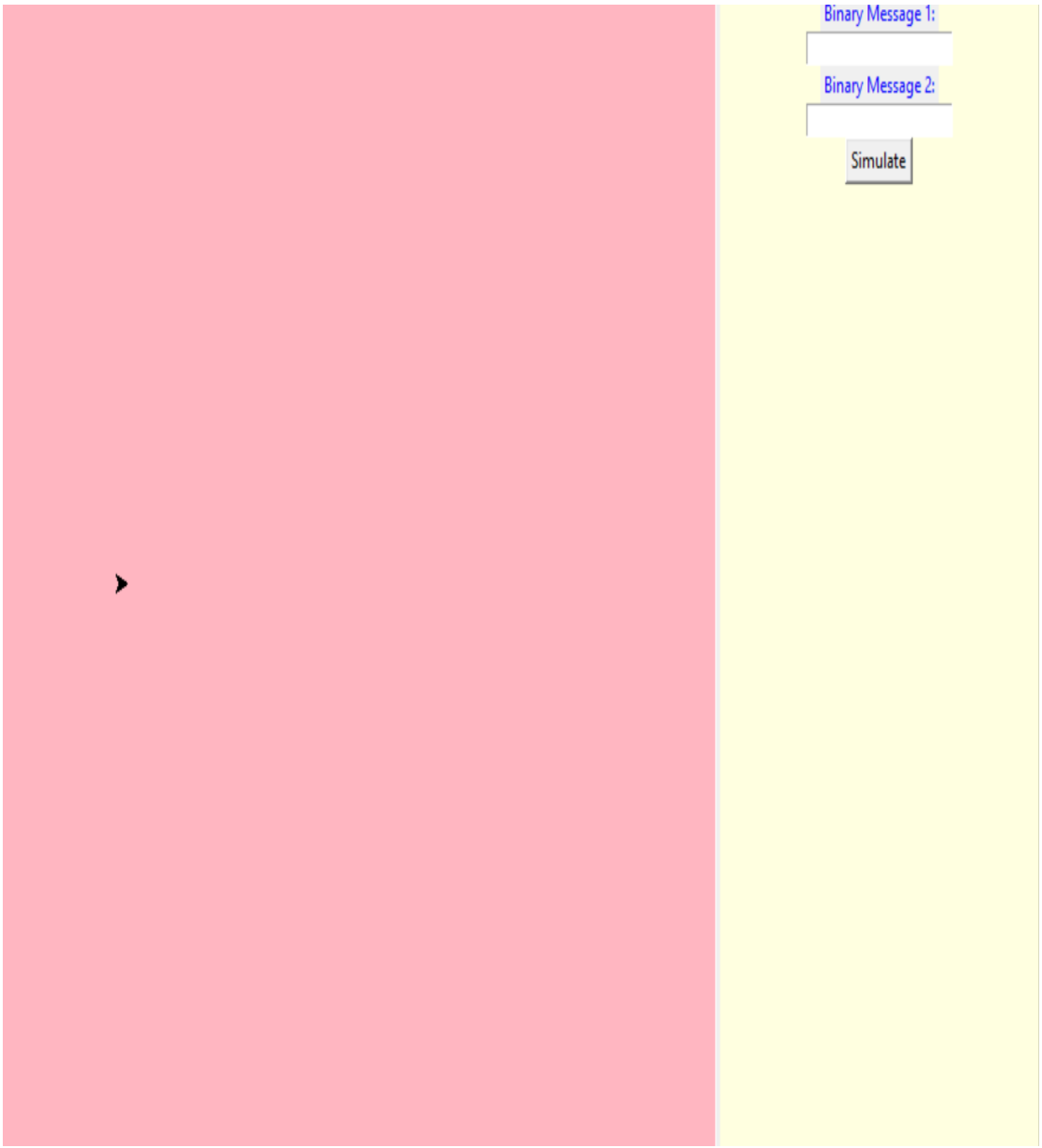
Output Data:
1111 1010 1100 1000

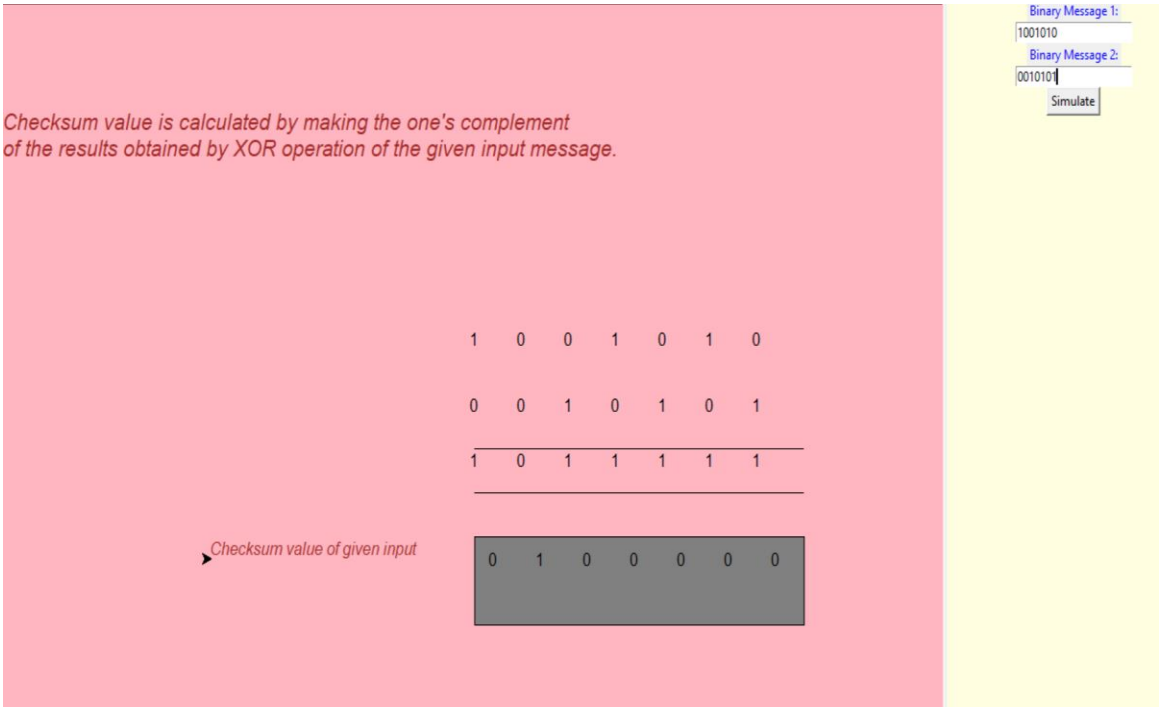
Cyclic Redundancy Check:



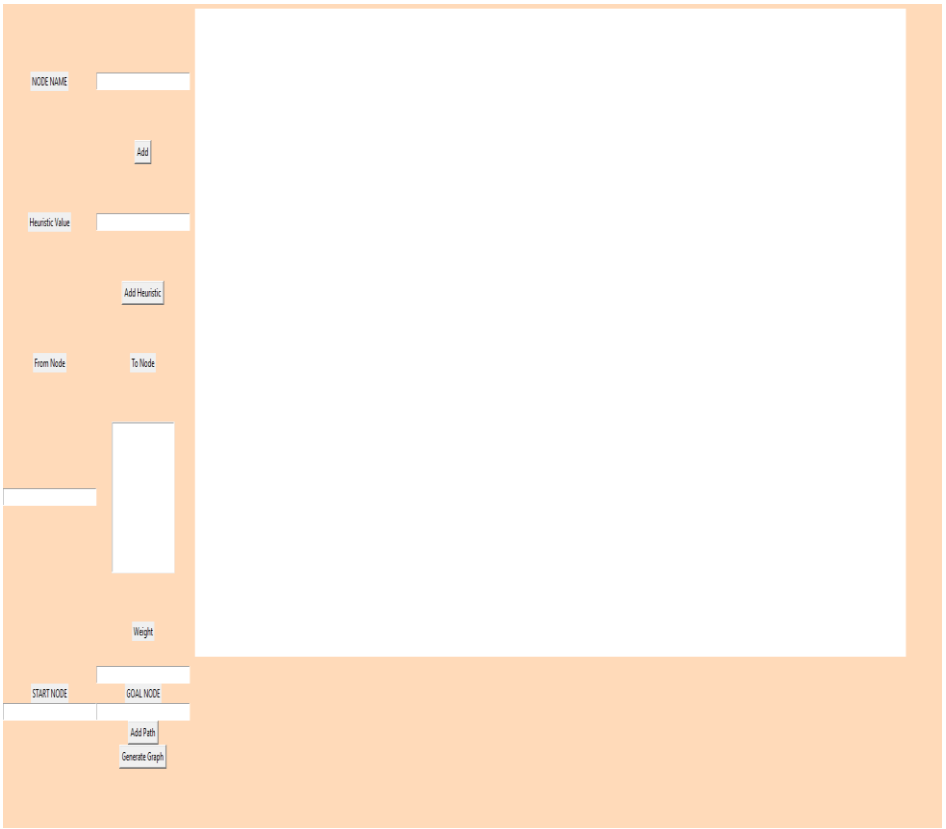


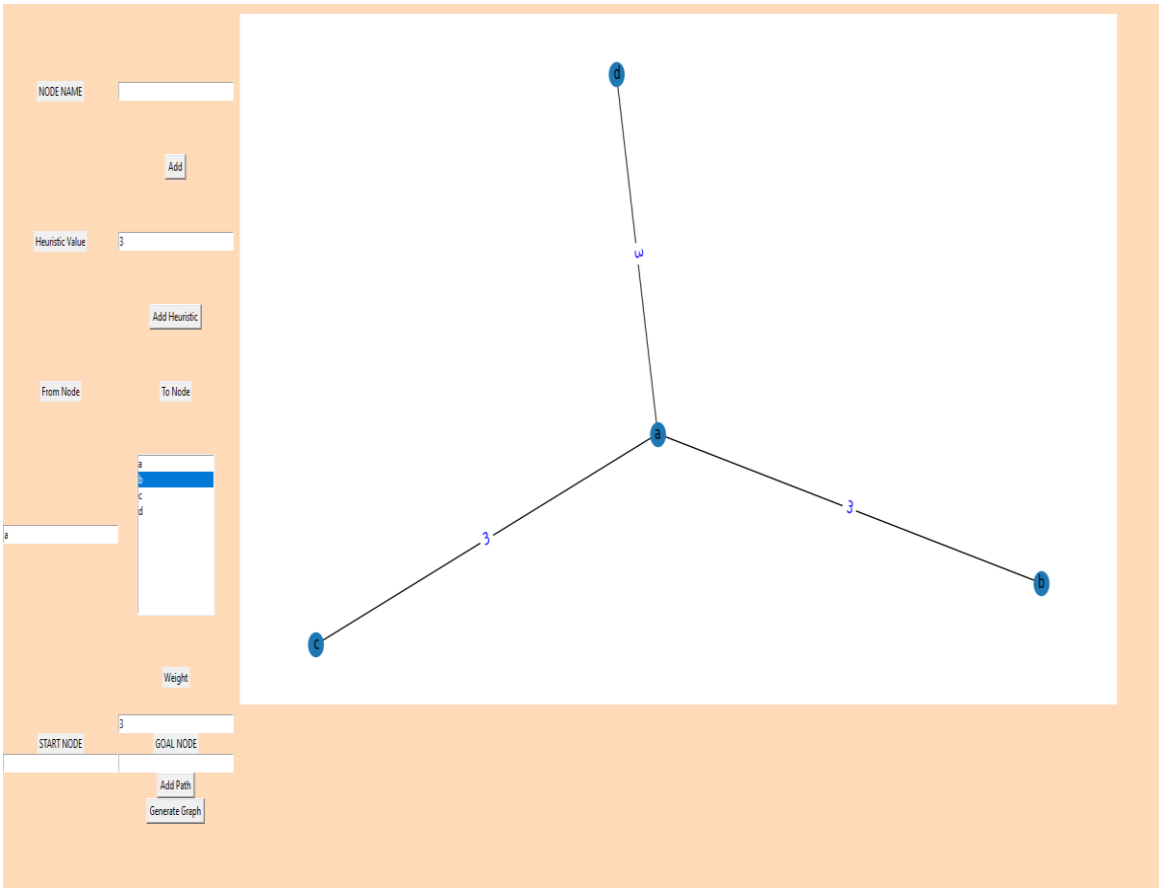
Checksum:

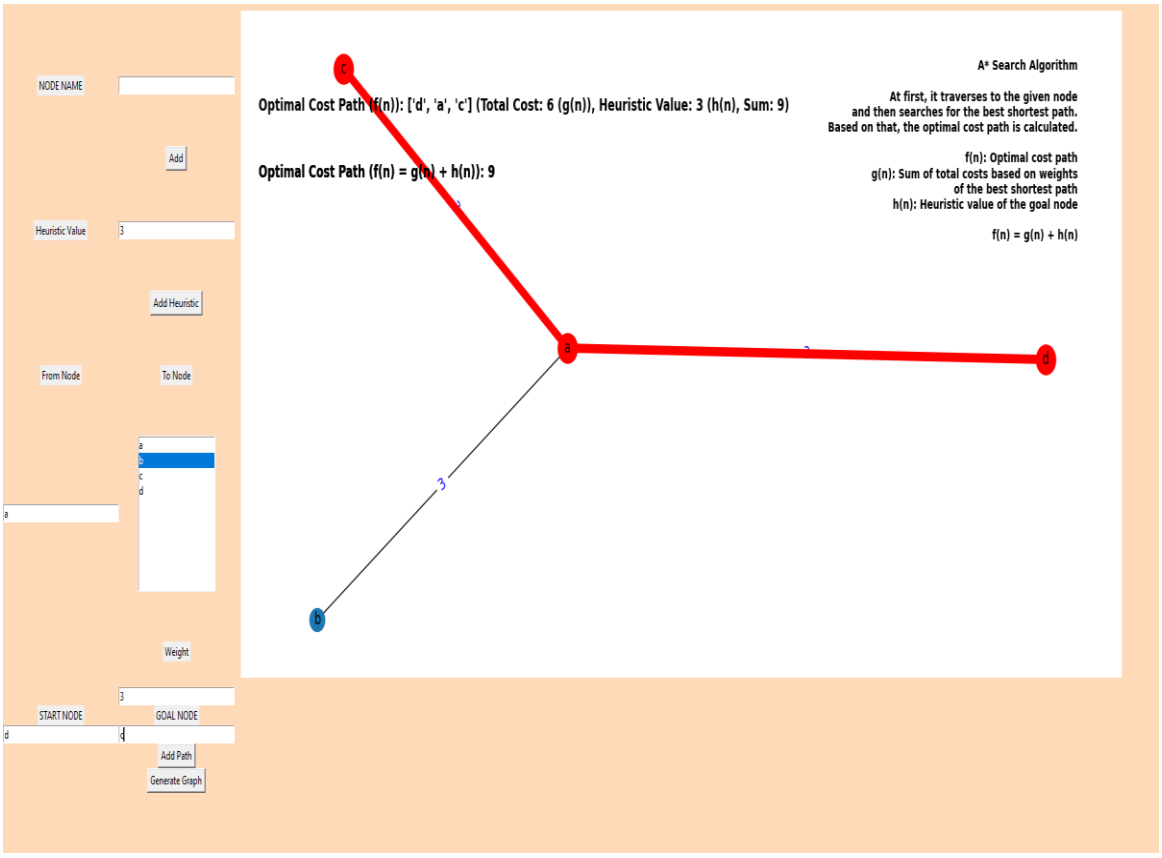




A Star Algorithm:




NODE NAME
Add
Heuristic Value
3
Add Heuristic
From Node
To Node
a
b
c
d
a
Weight
3
START NODE
GOAL NODE
Add Path
Generate Graph
d
3
a
3
3
c
b


NODE NAME
Add
Heuristic Value
3
Add Heuristic
From Node
To Node
a
b
c
d
a
Weight
3
START NODE
GOAL NODE
d
Add Path
Generate Graph
Optimal Cost Path (f(n)): ['d', 'a', 'c'] (Total Cost: 6 (g(n)), Heuristic Value: 3 (h(n), Sum: 9)
Optimal Cost Path (f(n) = g(n) + h(n)): 9
A* Search Algorithm
At first, it traverses to the given node
and then searches for the best shortest path.
Based on that, the optimal cost path is calculated.
f(n): Optimal cost path
g(n): Sum of total costs based on weights
of the best shortest path
h(n): Heuristic value of the goal node
f(n) = g(n) + h(n)
c
a
d
3
b

Hamming Code:

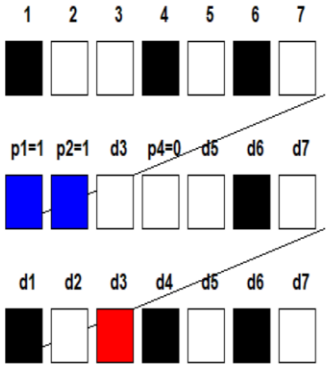
Enter the received 7-bit Hamming code:

Simulate

Enter the received 7-bit Hamming code: 1 - Black
0 - White

1001010

Simulate



Error bit index: 3

Note: Detecting error in parity bits (1 means error exists, 0 means no error)

Received Code: 1001010

P1 = 1 ⊕ 0 ⊕ 0 ⊕ 0 = 1

P2 = 0 ⊕ 0 ⊕ 1 ⊕ 0 = 1

P4 = 1 ⊕ 0 ⊕ 1 ⊕ 0 = 0

Final Calculation of Error in Received Code:

Error bit index = 1 * 2 ** 0 + 1 * 2 ** 1 + 0 * 2 ** 2 = 3

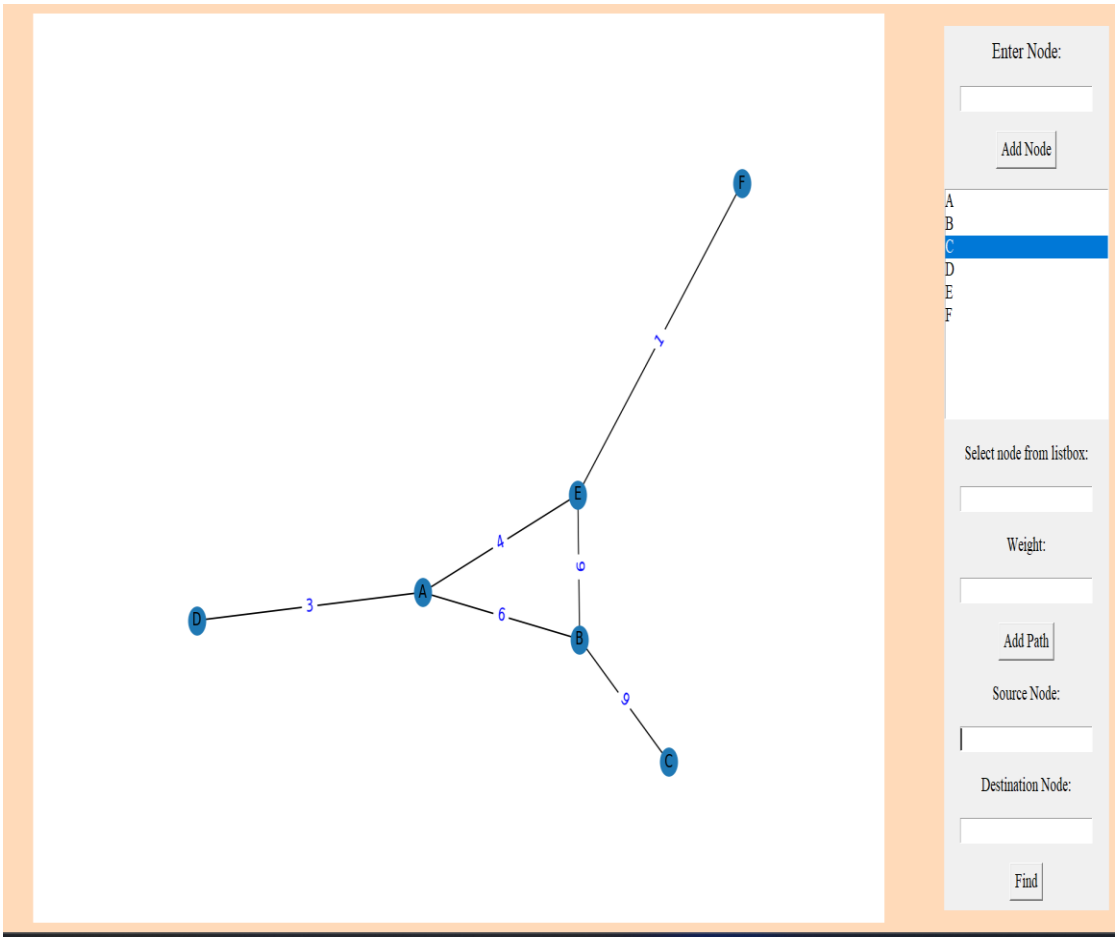
Corrected Code: 1011010

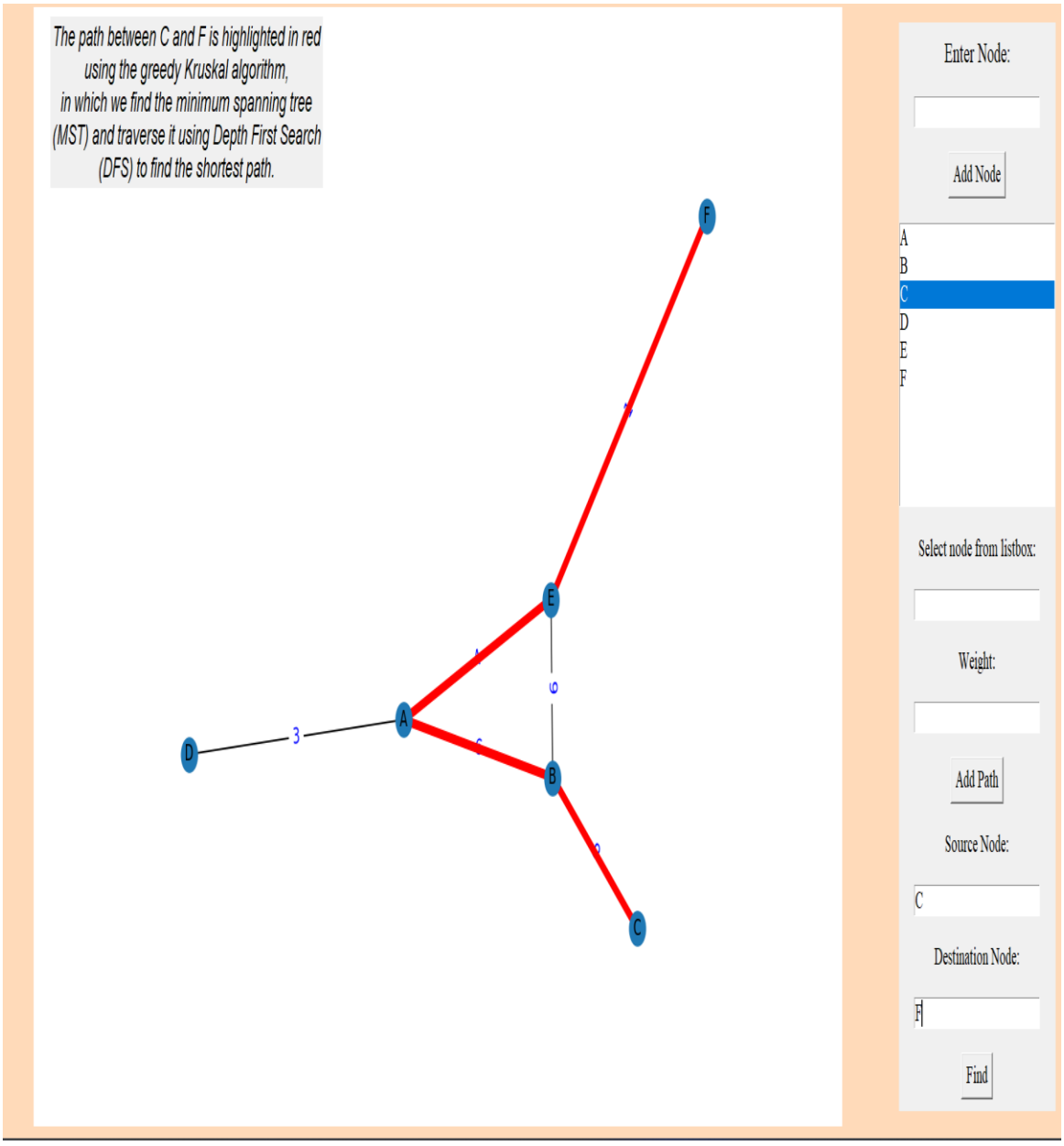
Error Corrected!

Note: The blue colored blocks represent errors in parity bits

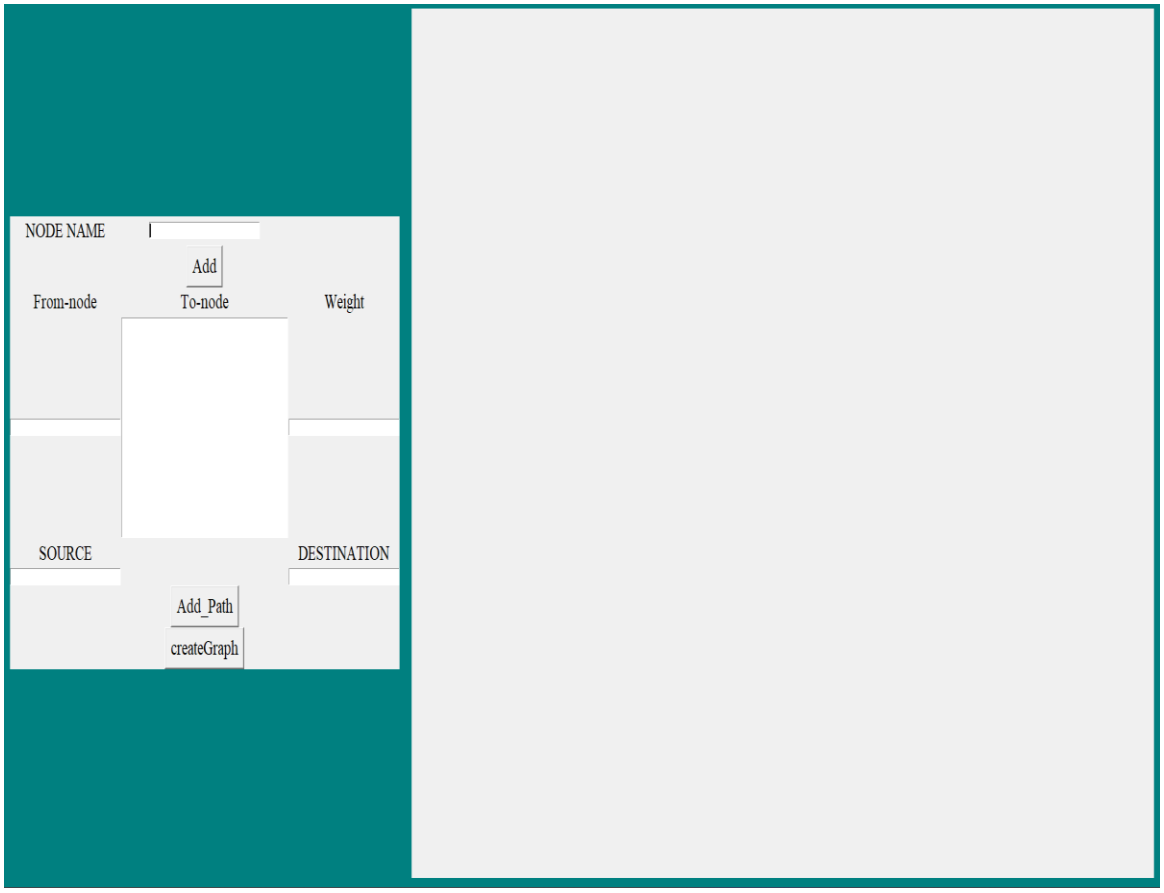
Note: The red colored block represents an error in the received code

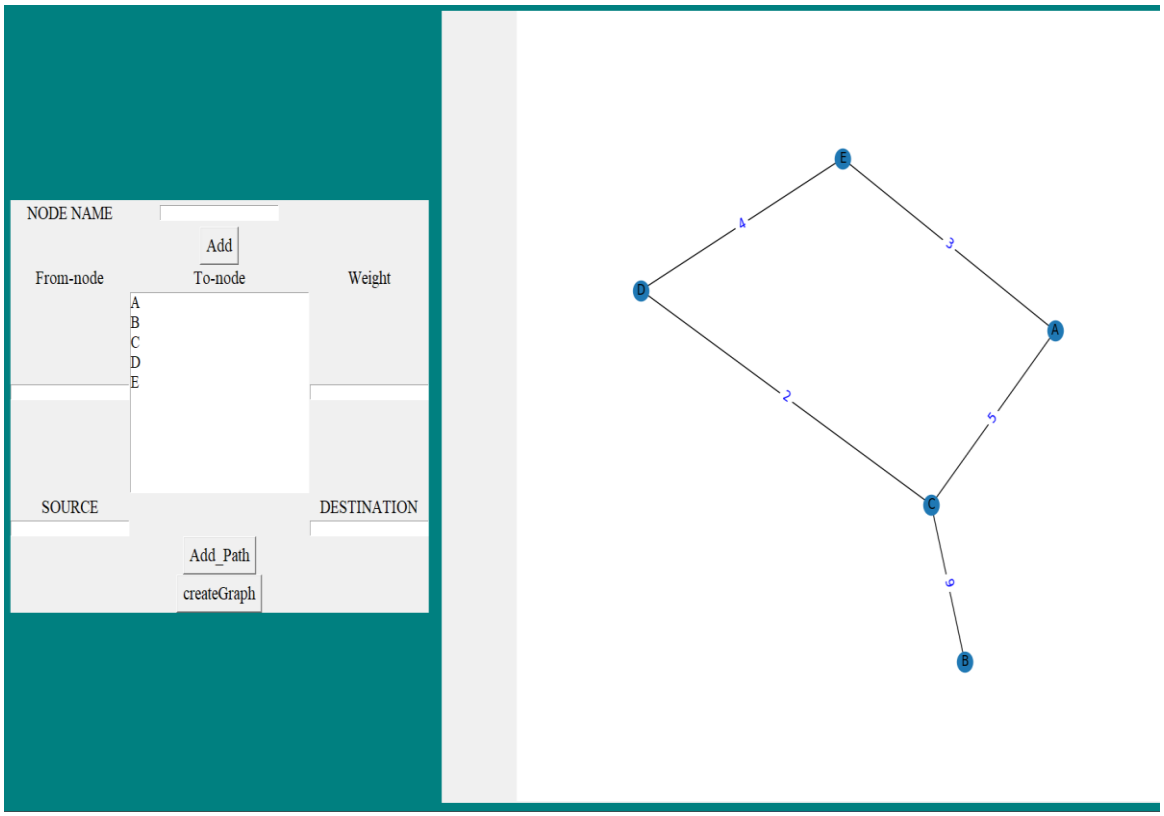
Path Finding Algorithm (Greedy Kruskal):

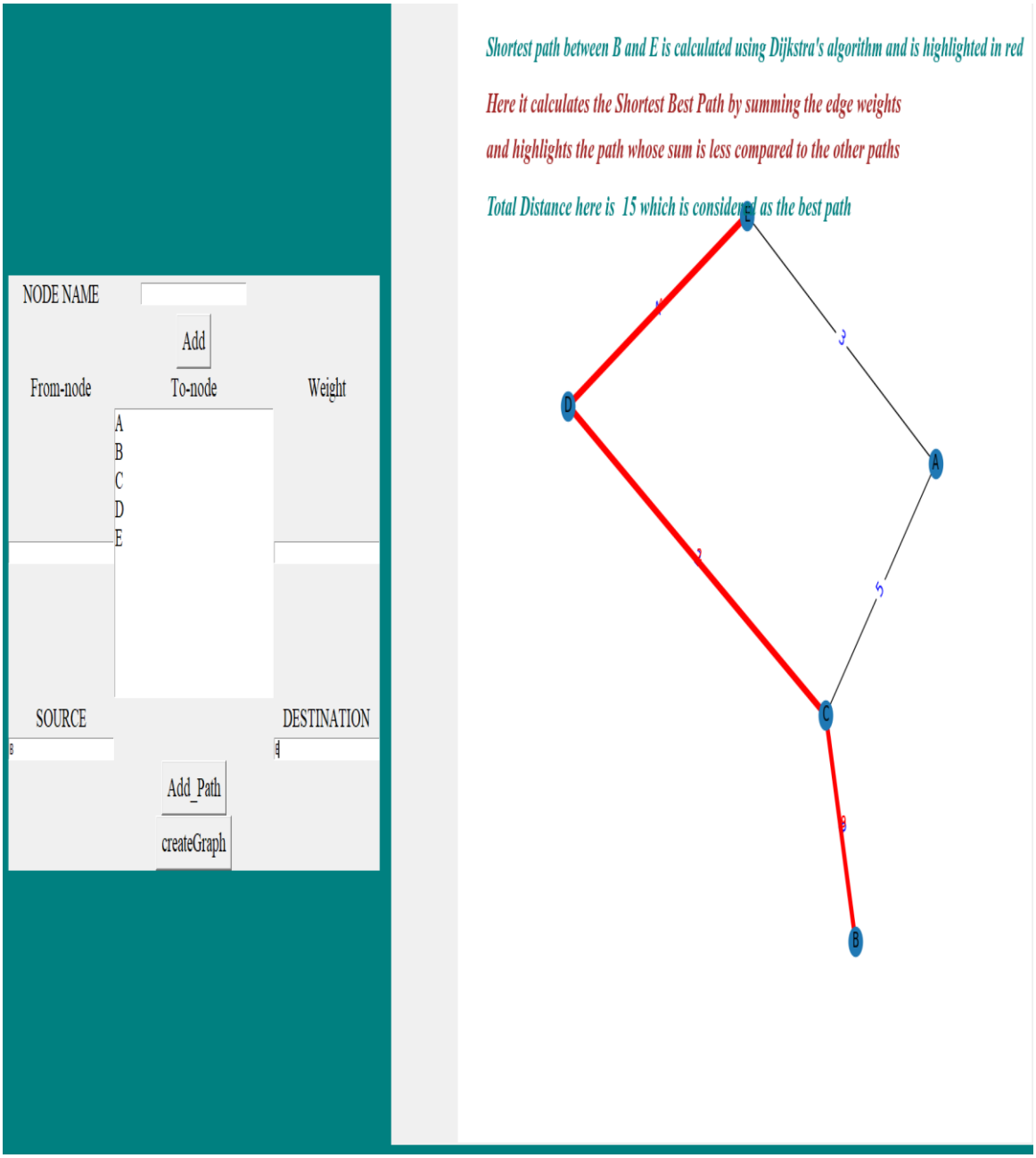



The path between C and F is highlighted in red using the greedy Kruskal algorithm, in which we find the minimum spanning tree (MST) and traverse it using Depth First Search (DFS) to find the shortest path.
Enter Node:
Add Node
A
B
C
D
E
F
Select node from listbox:
Weight:
Add Path
Source Node:
C
Destination Node:
F
Find
F
E
A
D
B
C
3
5

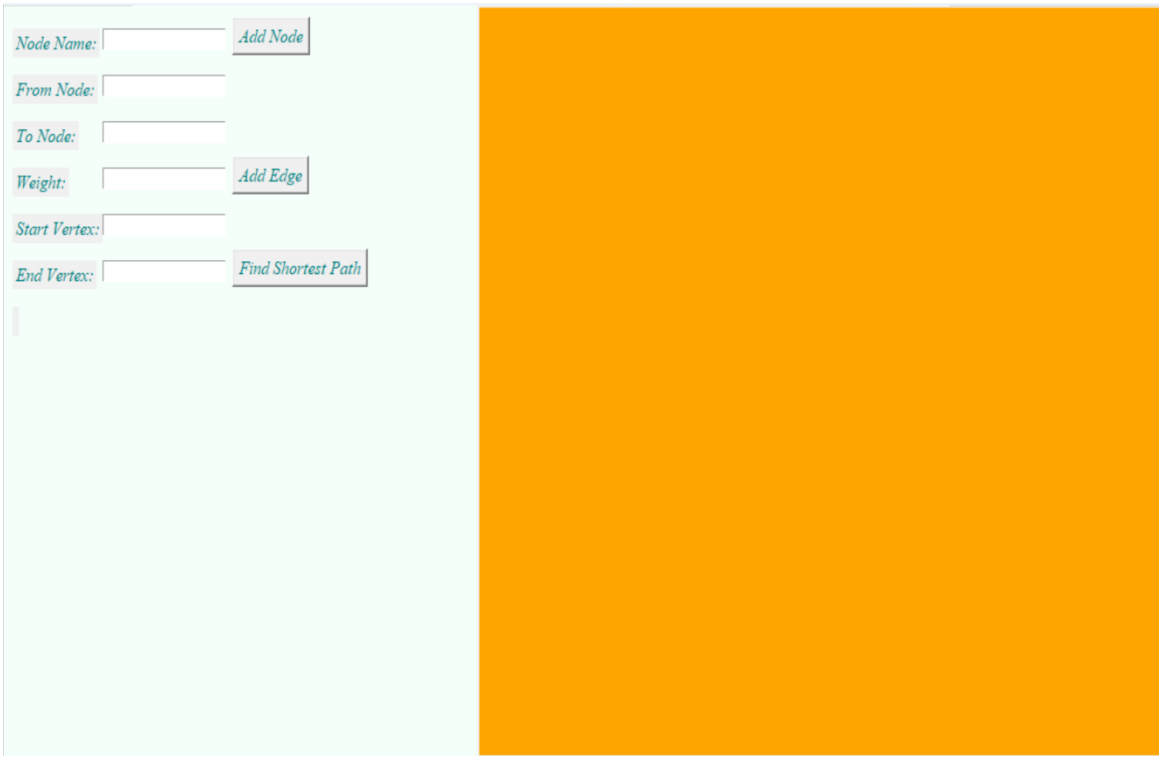
Shortest Path First (Dijkstra's):

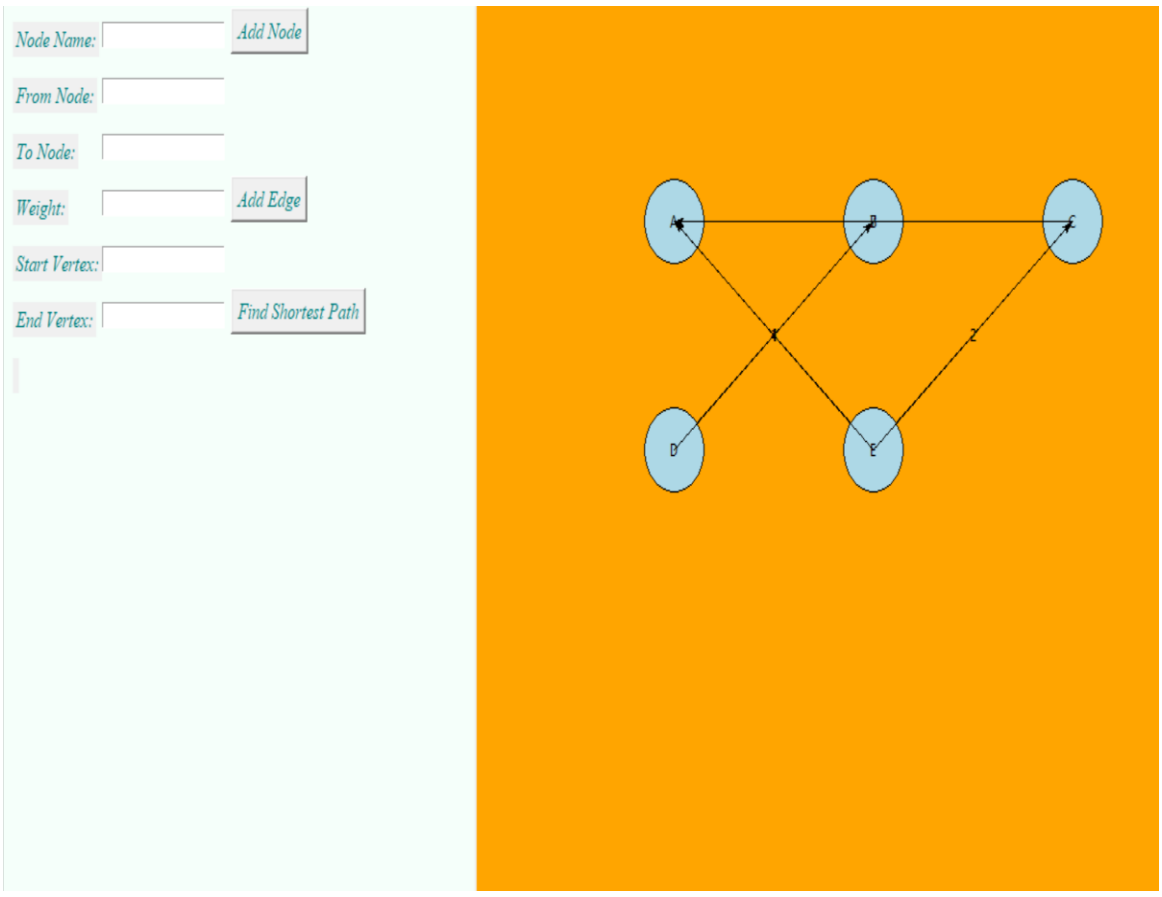





NODE NAME
Add
From-node
To-node
Weight
A
B
C
D
E
SOURCE
DESTINATION
Add_Path
createGraph
Shortest path between B and E is calculated using Dijkstra's algorithm and is highlighted in red
Here it calculates the Shortest Best Path by summing the edge weights
and highlights the path whose sum is less compared to the other paths
Total Distance here is 15 which is considered as the best path
E
D
A
C
B
4
3
2
5
3

Open Shortest Path First:





Node Name: Add Node

From Node:

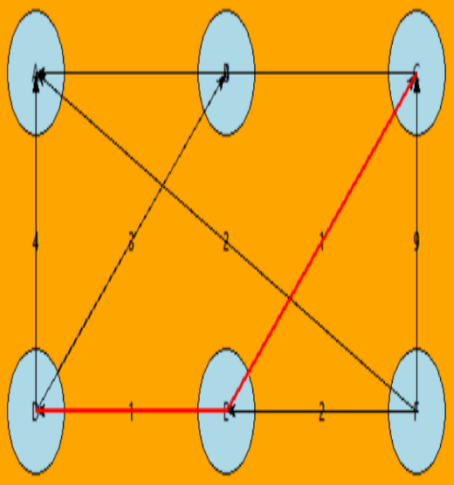
To Node:

Weight: Add Edge

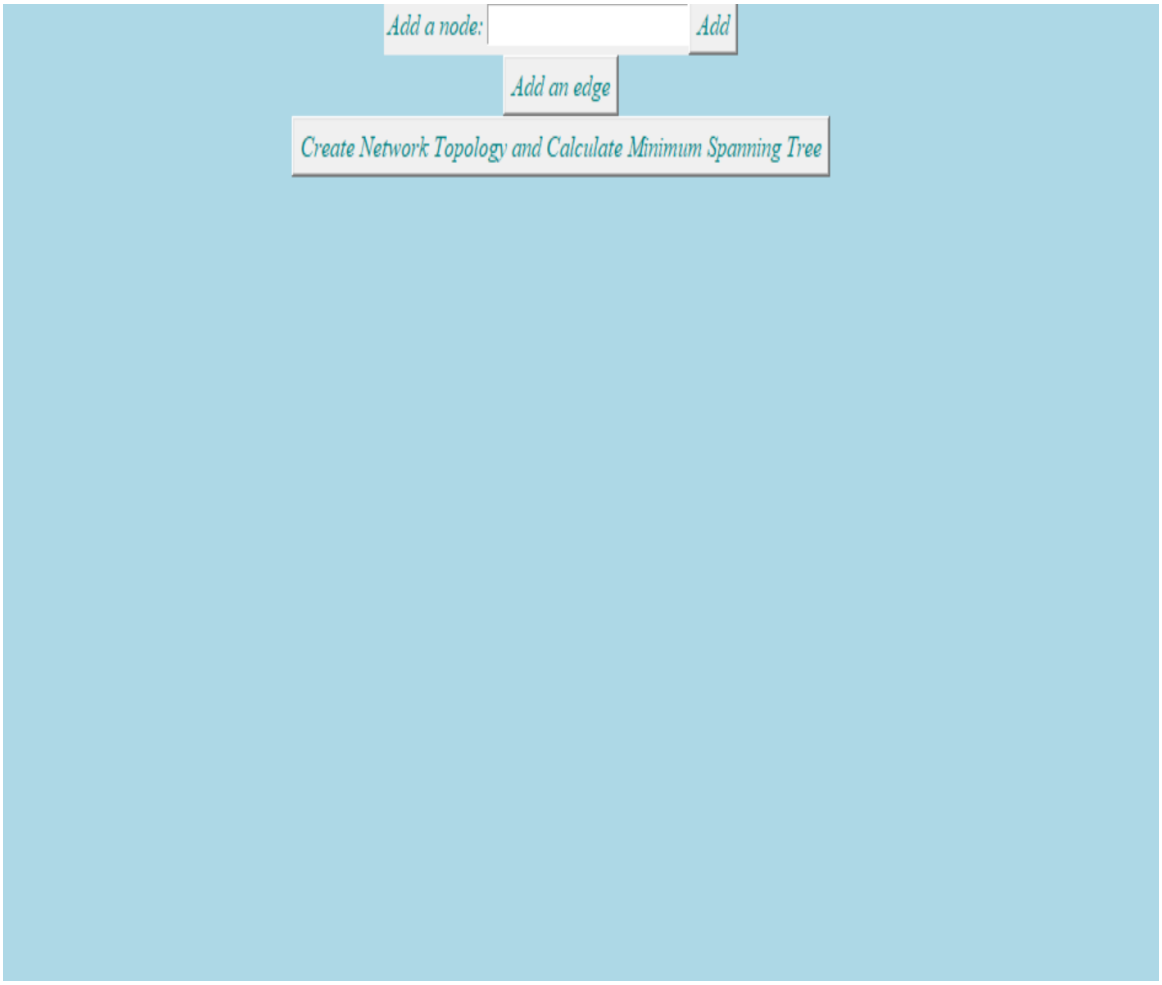
Start Vertex: C

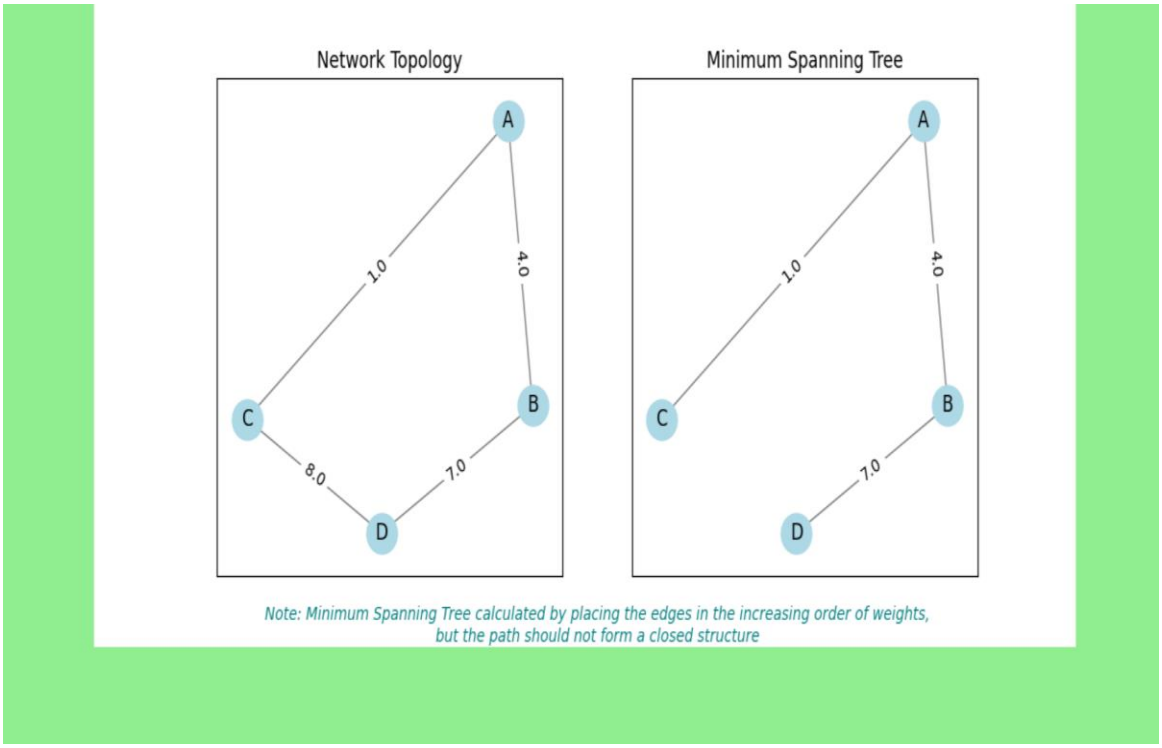
End Vertex: D Find Shortest Path

Shortest path from C to D: C -> C -> E -> D

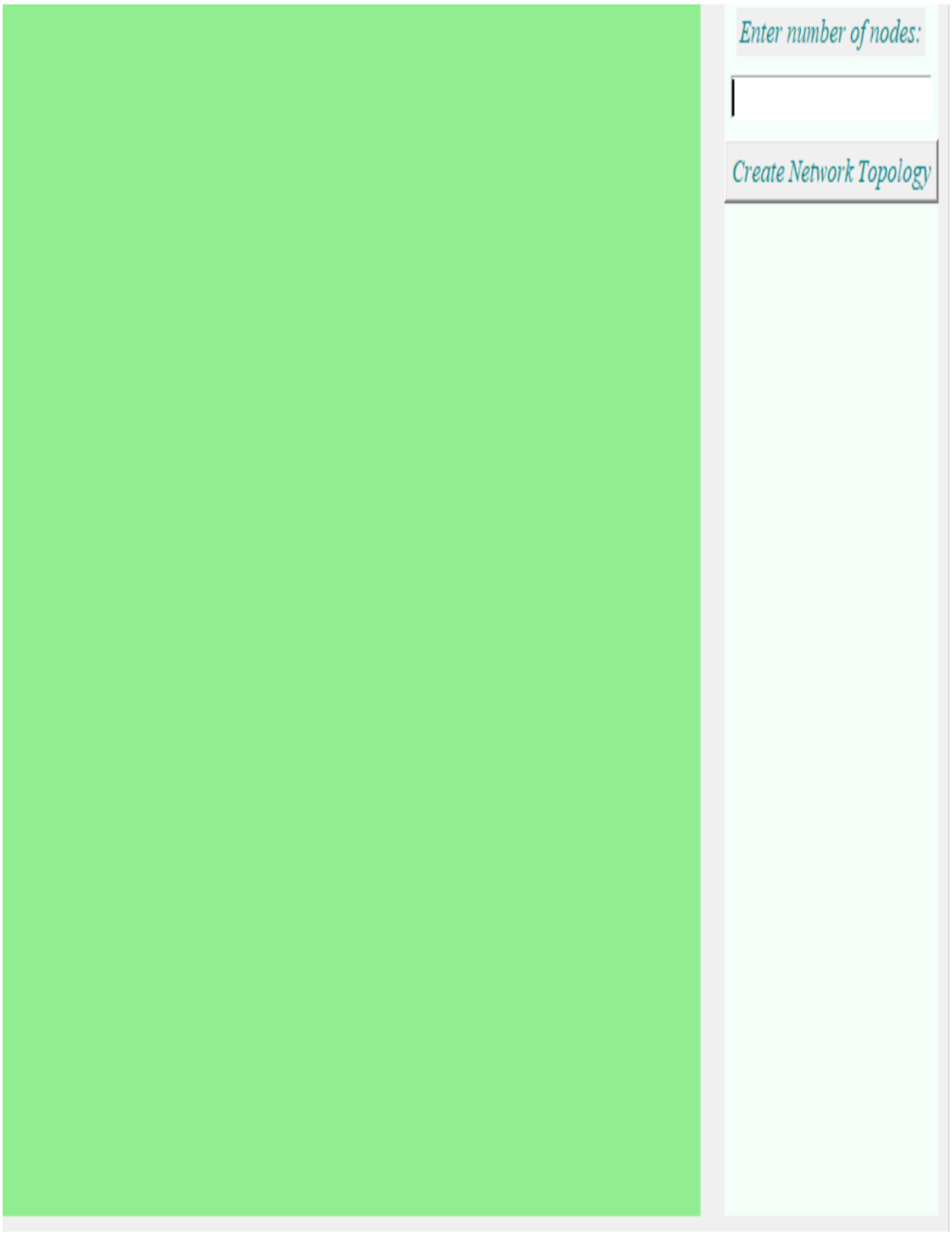


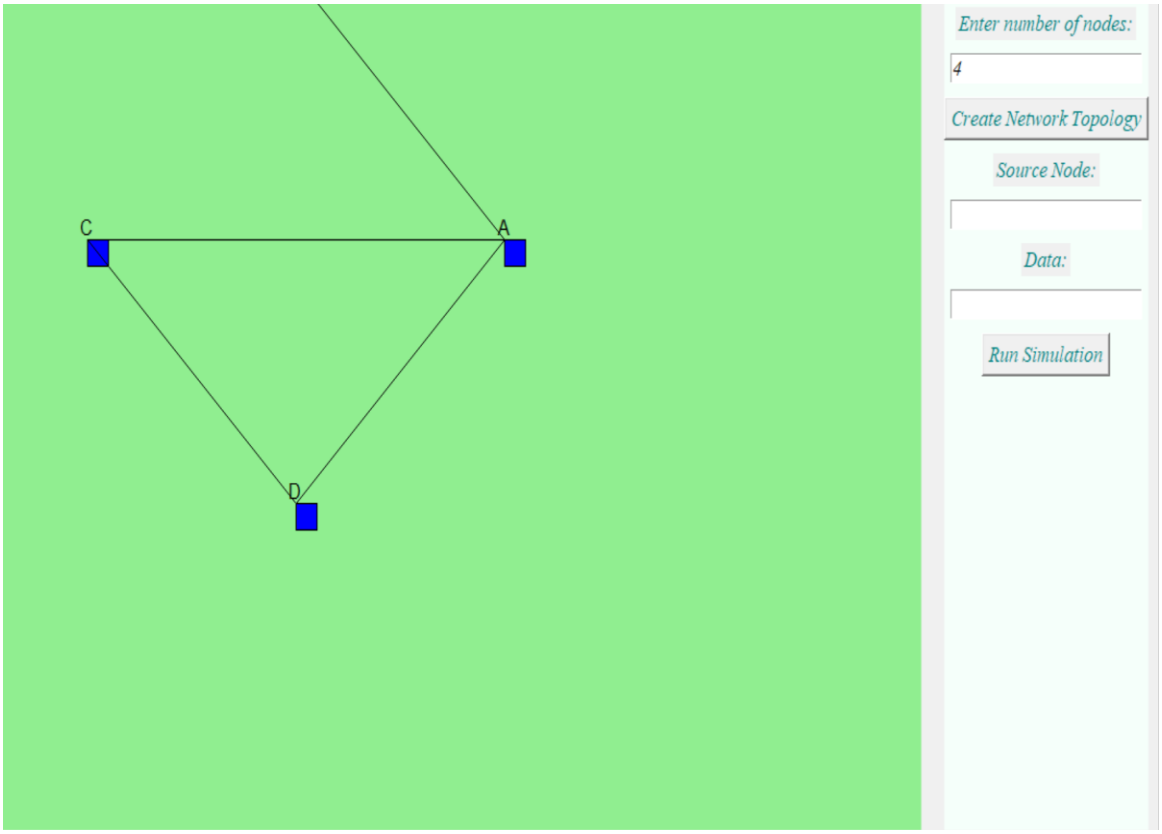
Minimum Spanning Tree(Greedy Kruskal):

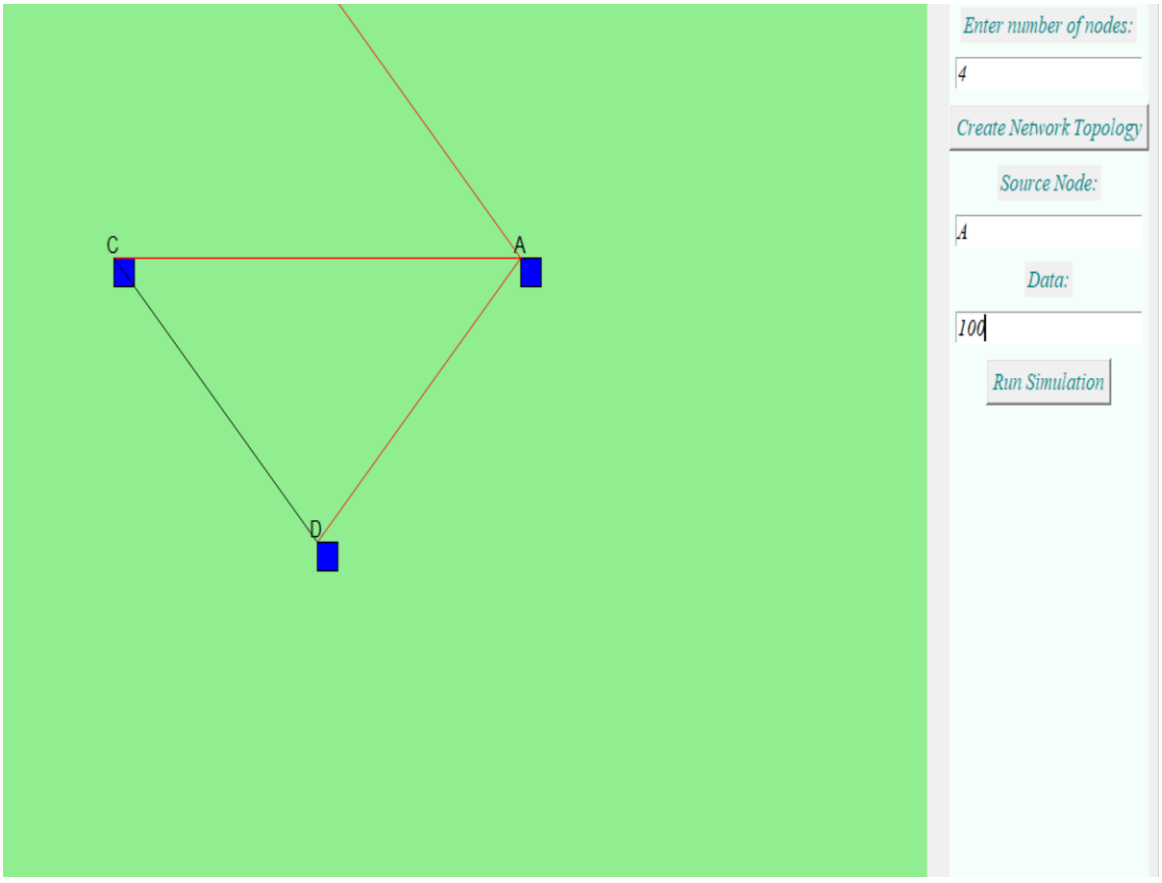




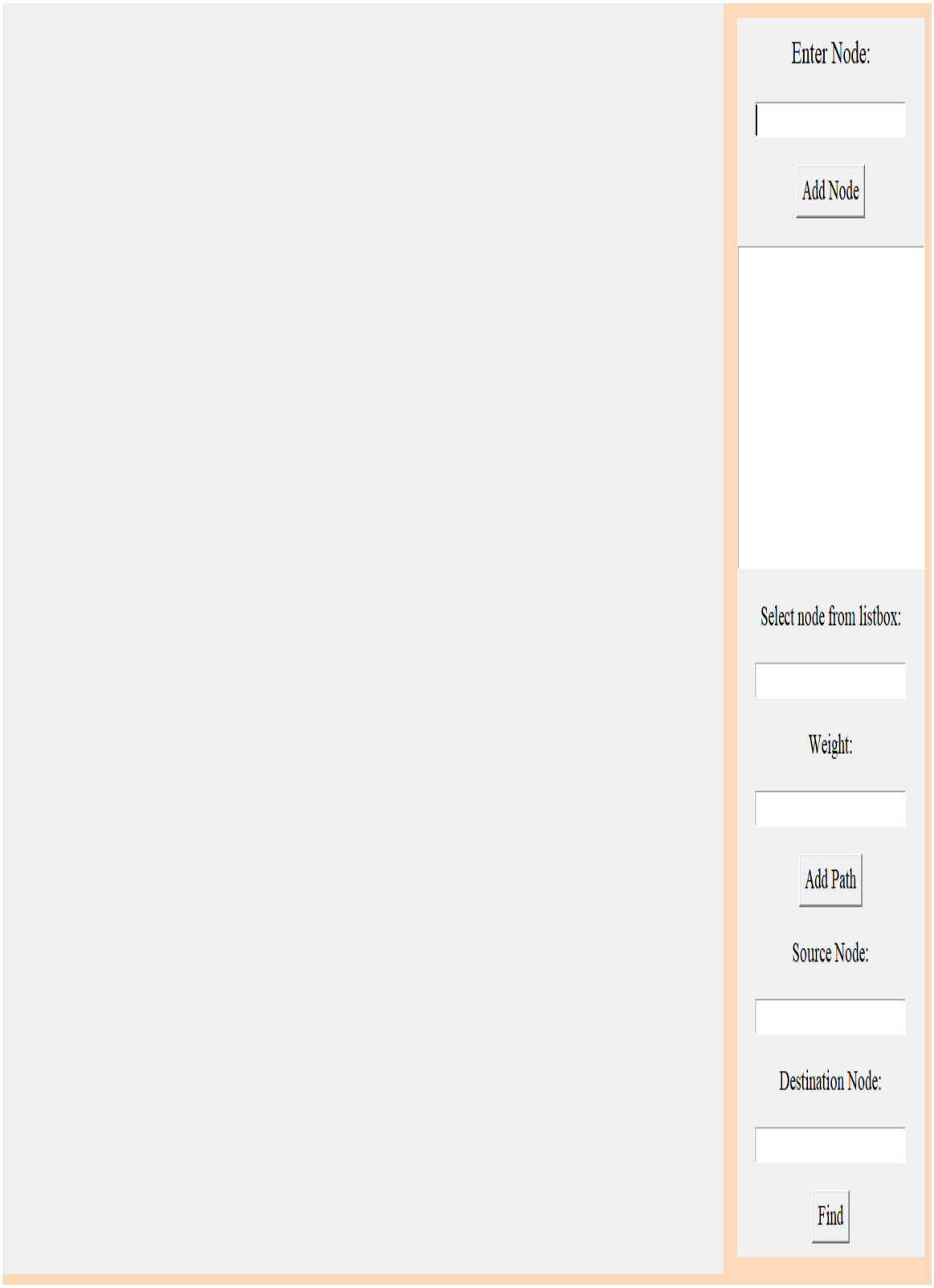
Flooding Routing Algorithm:

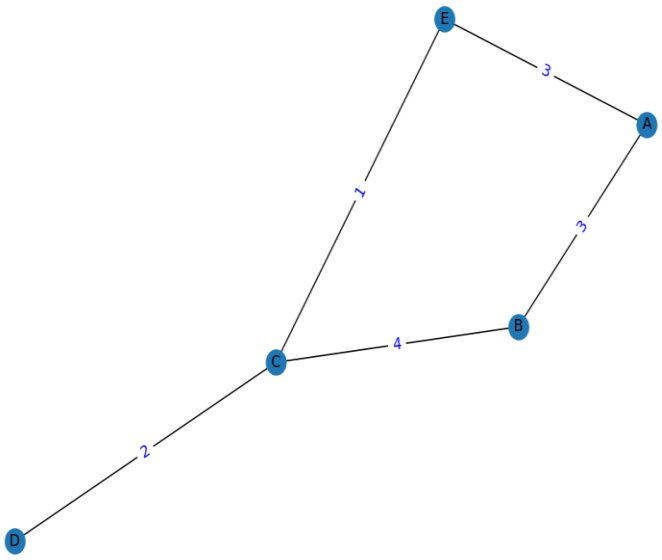



C
A
D
Enter number of nodes:
4
Create Network Topology
Source Node:
Data:
Run Simulation

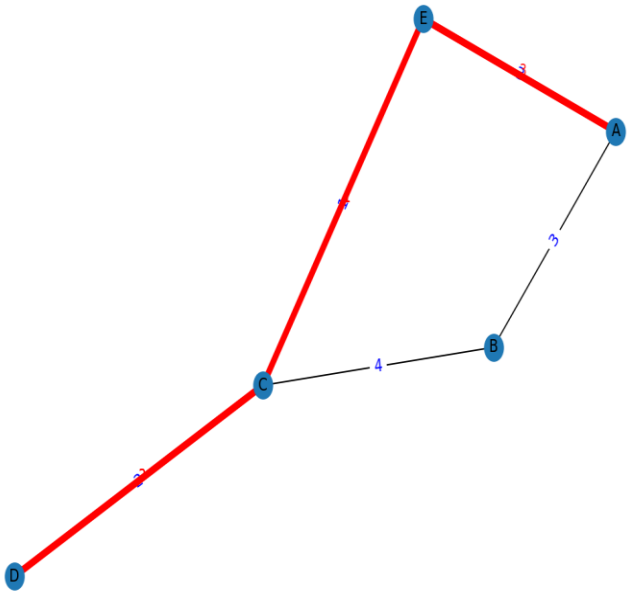

C
A
D
Enter number of nodes:
4
Create Network Topology
Source Node:
A
Data:
100
Run Simulation

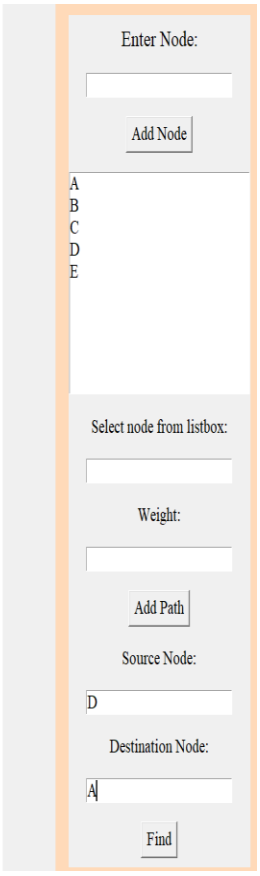
Distance Vector Routing Algorithm:





Total Distance: 6 which is determined by summing the edges assigned,





TESTING

7. Testing

7.1 Introduction

Testing is the major quality control measures and during the software development it is used to detect errors that could have occurred during any of the phase like requirement analysis, design, coding. The goal of testing is to uncover errors in the program.

7.2 Levels of Testing

Testing is done in different levels which includes the following.

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

- **Unit Testing**
  In Unit Testing each module gets tested during the coding phase itself. The purpose is to exercise the different parts of the module code to detect the coding errors.

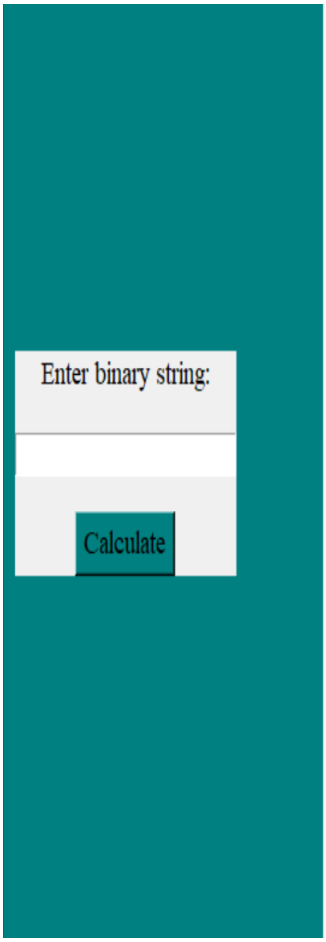
- **Integration Testing**
  After new testing the modules are gradually integrated into sub systems. It is performed to detect design by errors by focusing on testing the Interconnection between modules.

- **System Testing**
  System is tested against the system requirement if all requirements are met and if the system performs as specified by the requirement.

- **Acceptance Testing**
  It is performed to demonstrate to the client on real life data of the client, the operation of the system.

7.3 Test Case

It is the input that tests genuineness of the program and successful execution of the test case levels, that there are no errors in the program that are under testing. It is a set of conditions or variables under which tester will determine whether an application or software works currently.

Test Case Title	Simple Parity Check
Purpose of Testing	Purpose is to calculate the parity bits.
Test Data	Click on the calculate Parity bit Button
Steps	1.Enter the binary string. 2.Click on the parity bit button. 3.Calculates the parity bit by counting number of 1's and draws coloured blocks. 4.Coloured block represents bit value, green for 1 and red for 0. 5. The parity bit draws blue block if parity value is 1, otherwise white block is drawn.
Expected Output	Valid Output: 

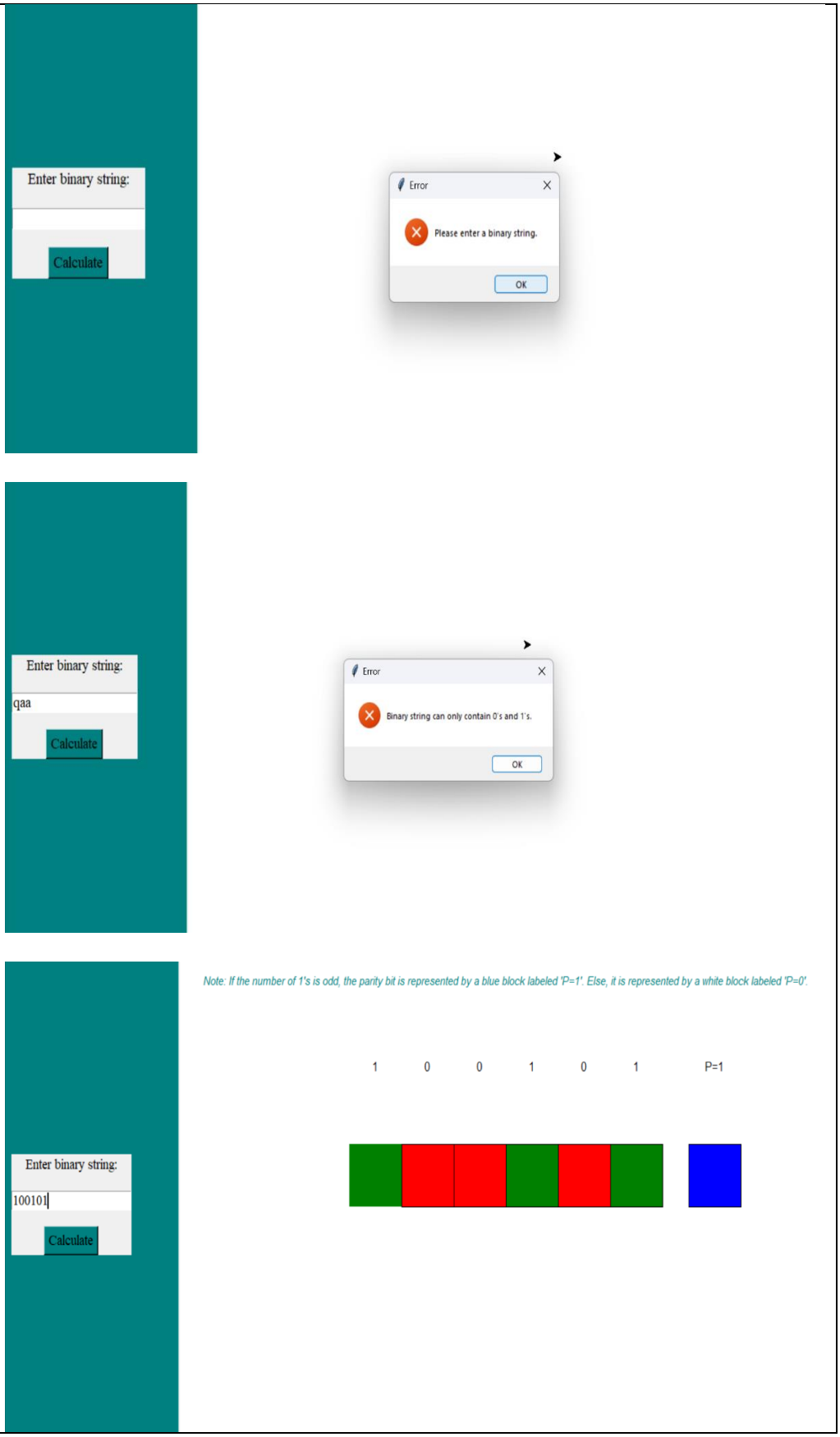
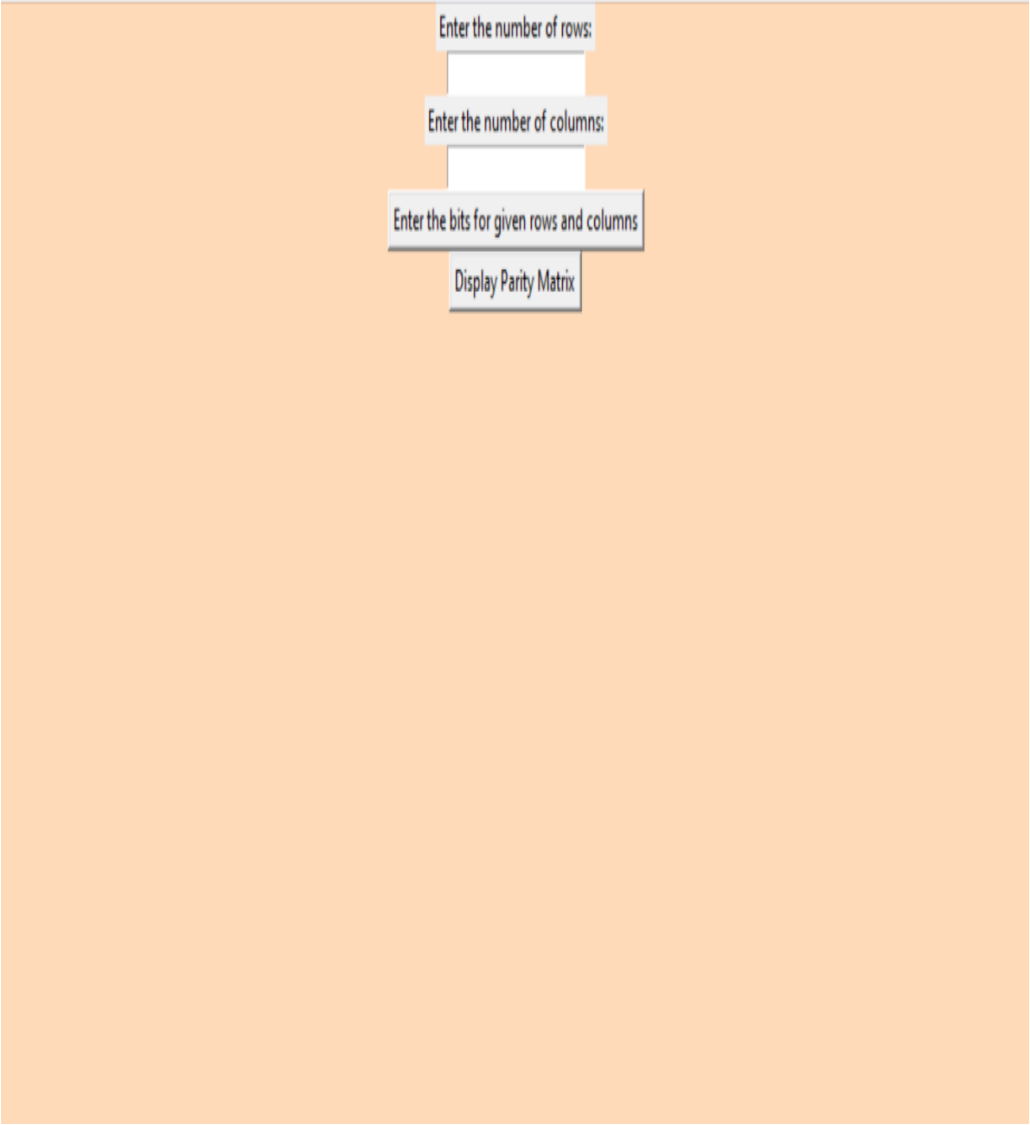
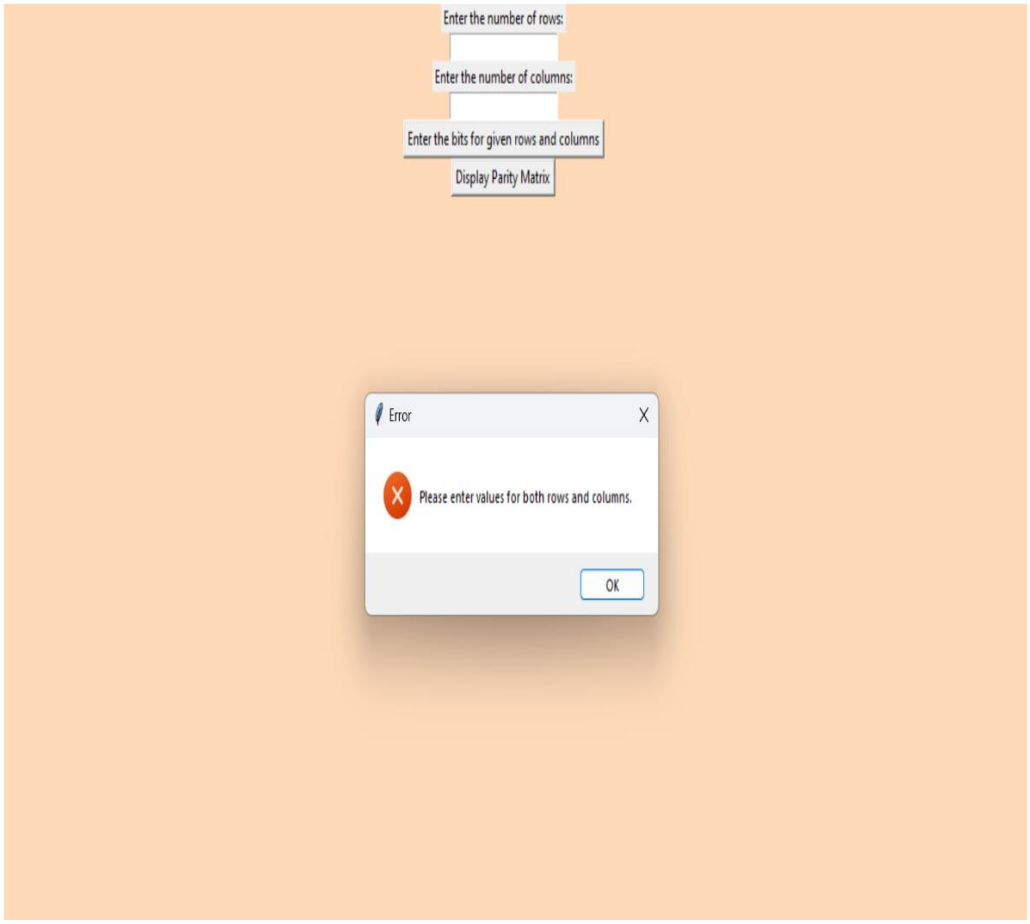
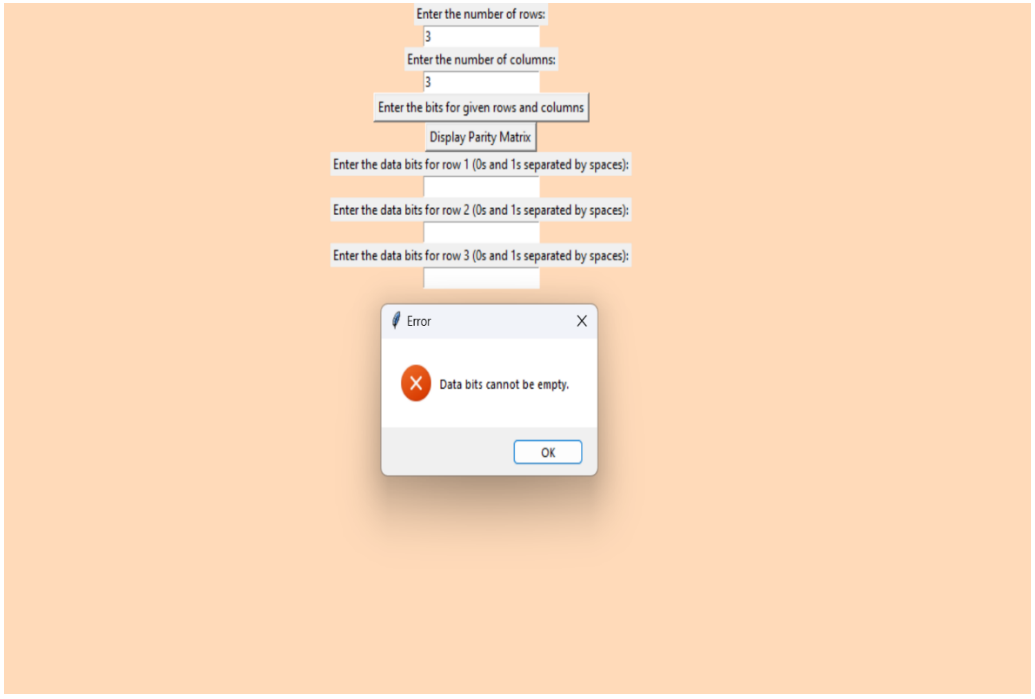
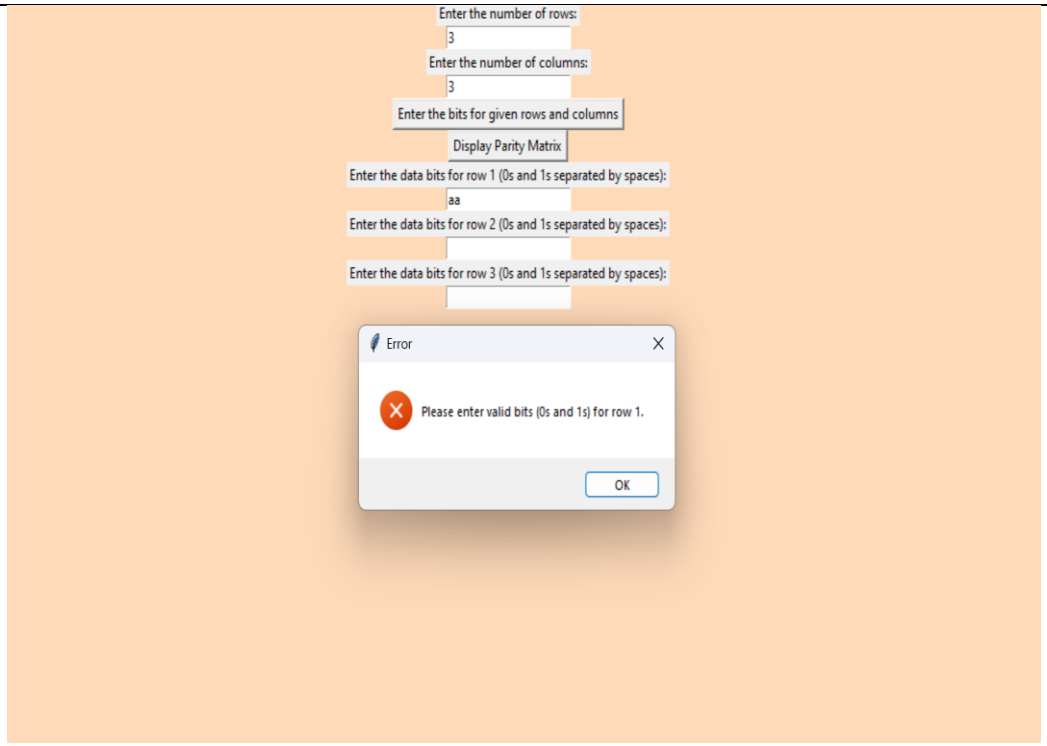


Table 7.1.1 Simple Parity Check

Test Case Title	Two Dimensional Parity Check
Purpose of Testing	Purpose is to calculate the parity bits.
Test Data	Click on the calculate Parity bit Button
Steps	1.Enter the number of rows and columns. 2.Enter the bits for given rows and columns. 3.Click on the Display Parity Matrix button. 4.Calculates row and column parities. 5. Displays new 2d list representing data matrix with parity bits.
Expected Output	Valid Output: 


Enter the number of rows:
Enter the number of columns:
Enter the bits for given rows and columns
Display Parity Matrix
Error
Please enter values for both rows and columns.
OK


Enter the number of rows:
3
Enter the number of columns:
3
Enter the bits for given rows and columns
Display Parity Matrix
Enter the data bits for row 1 (0s and 1s separated by spaces):
Enter the data bits for row 2 (0s and 1s separated by spaces):
Enter the data bits for row 3 (0s and 1s separated by spaces):
Error
Data bits cannot be empty.
OK



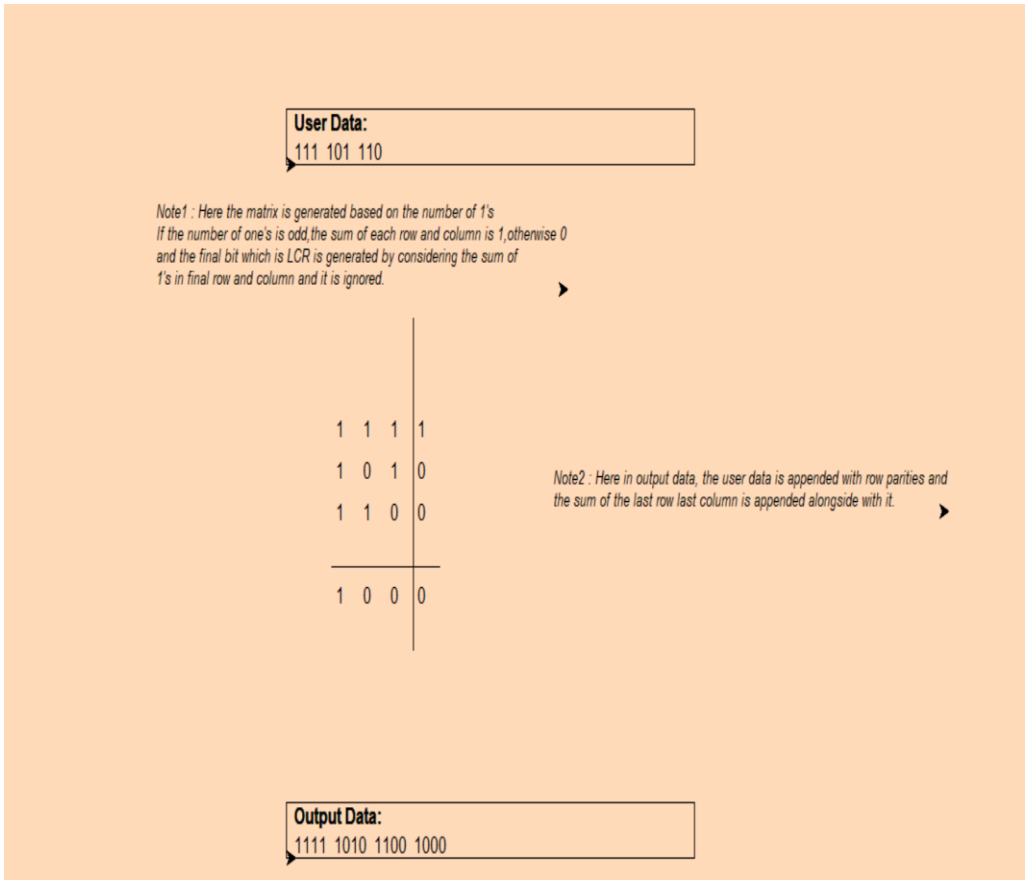


Table 7.1.2 Two-Dimensional Parity Check

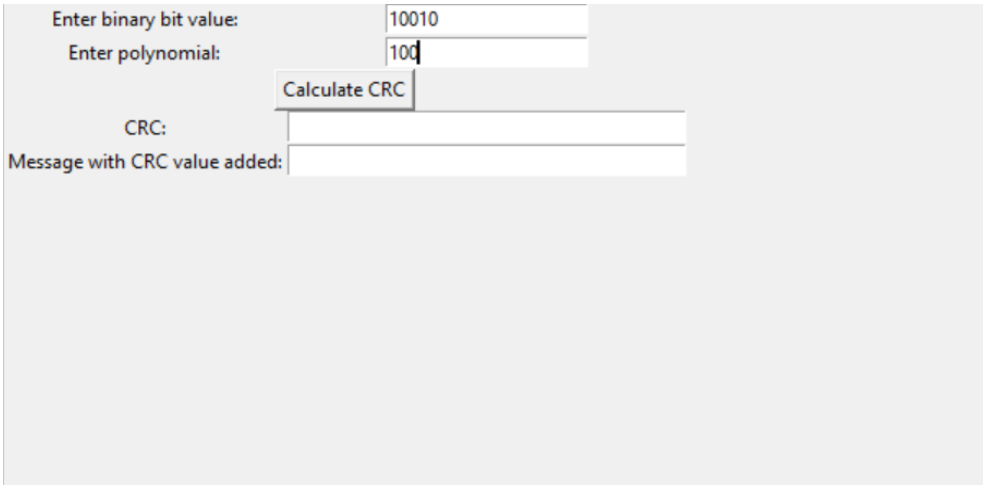
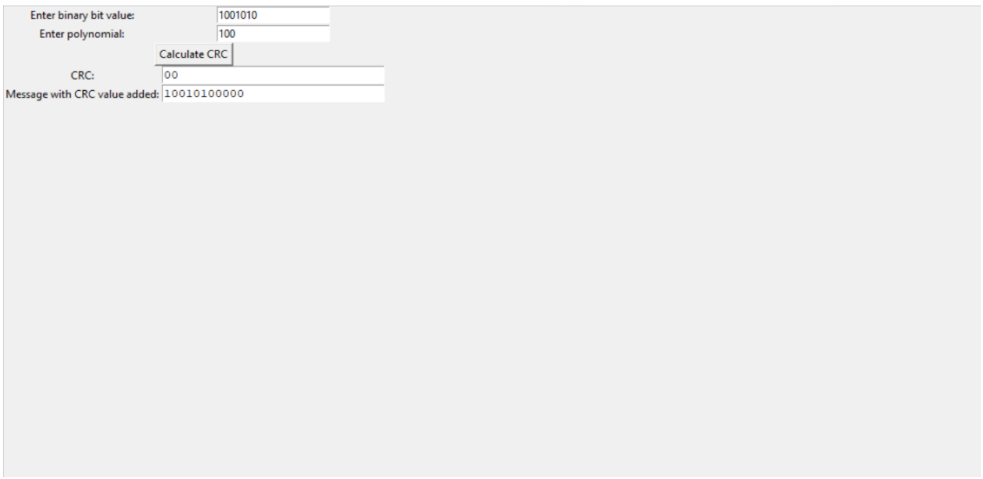
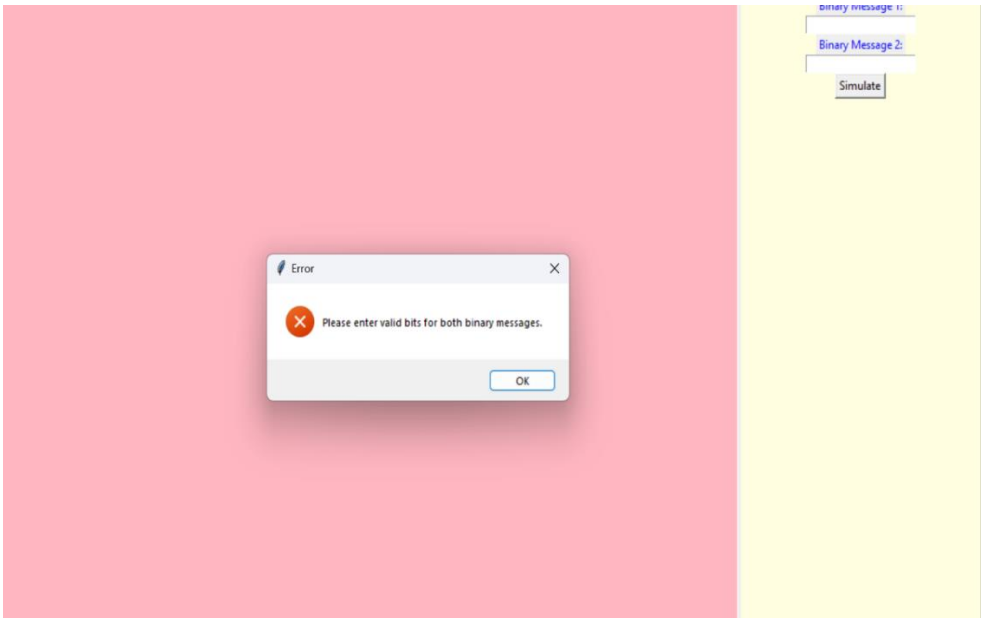
Test Case Title	Cyclic Redundancy Check
Purpose of Testing	Purpose is to calculate the CRC value
Test Data	Click on the calculate Parity bit Button
Steps	1.Enter the binary bit value and polynomial. 2.Click on calculate CRC button. 3.Appends 0's to binary bit message equal to the degree of polynomial. 4.Performs bitwise XOR operation. 5.Displays calculated CRC and message with CRC value added.
Expected Output	Valid Output:  

Table 7.2 Cyclic Redundancy Check

Test Case Title	Checksum
Purpose of Testing	Purpose is to calculate checksum of the input message.
Test Data	Click on the Simulate button
Steps	1.Input binary bits 2.Check if the entered binary bits are valid or not. 3.If the binary bits entered are valid then sum the binary bits. 4.Output is displayed by taking the one's complement of the sum
Expected Output	Valid Output:  

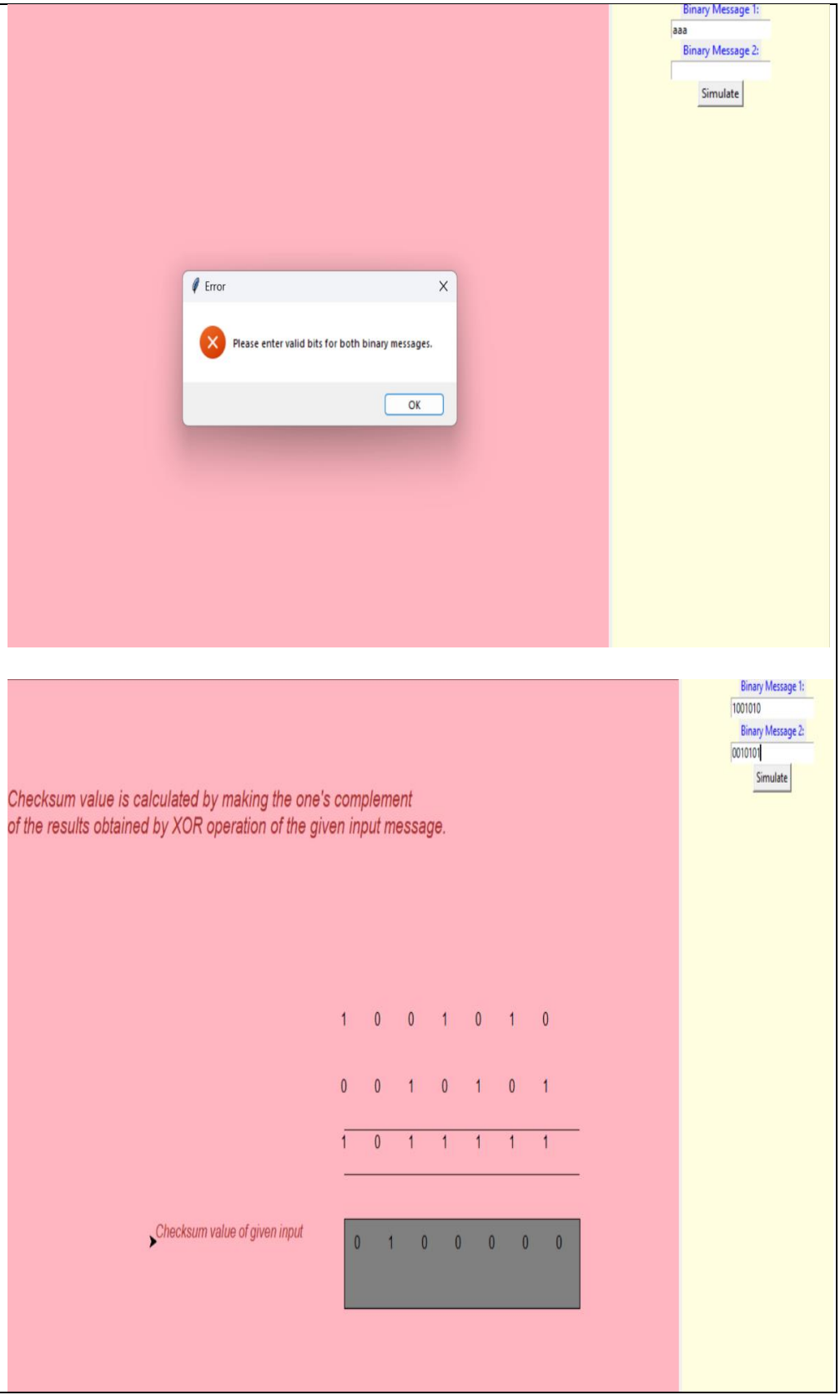
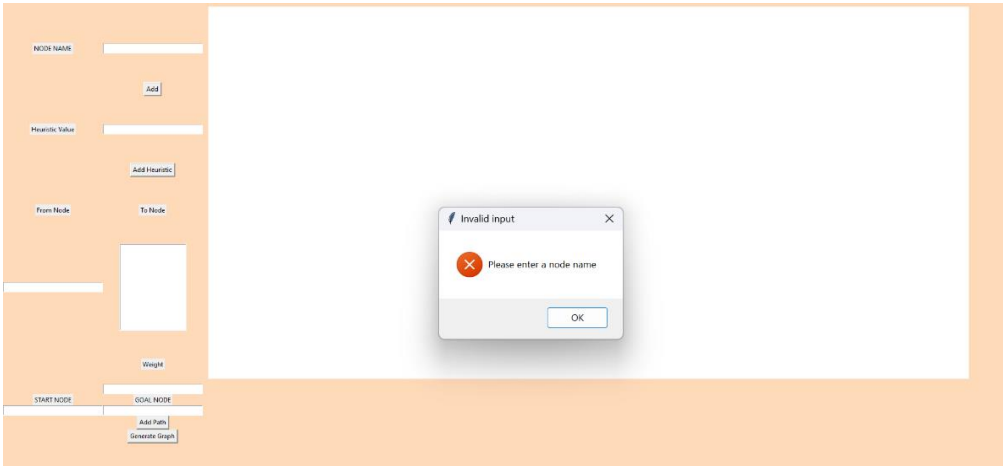
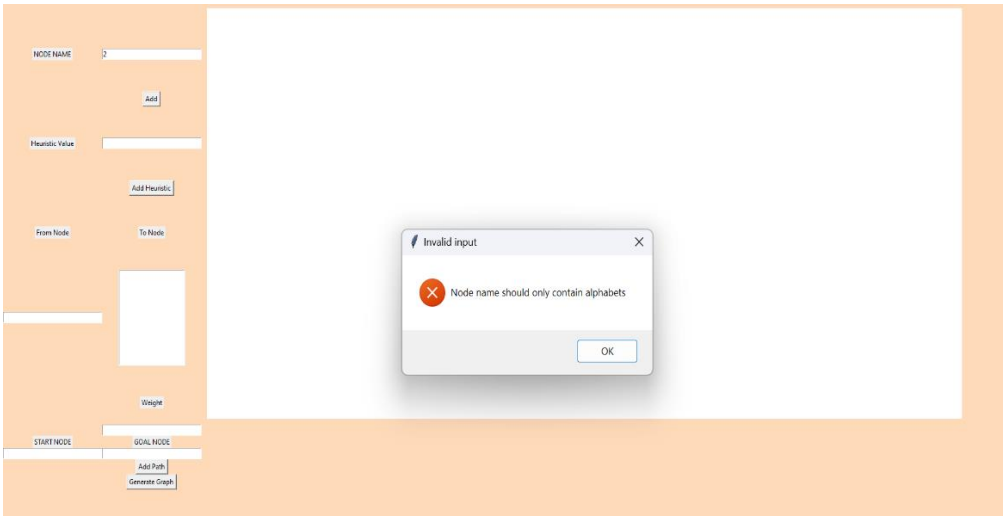
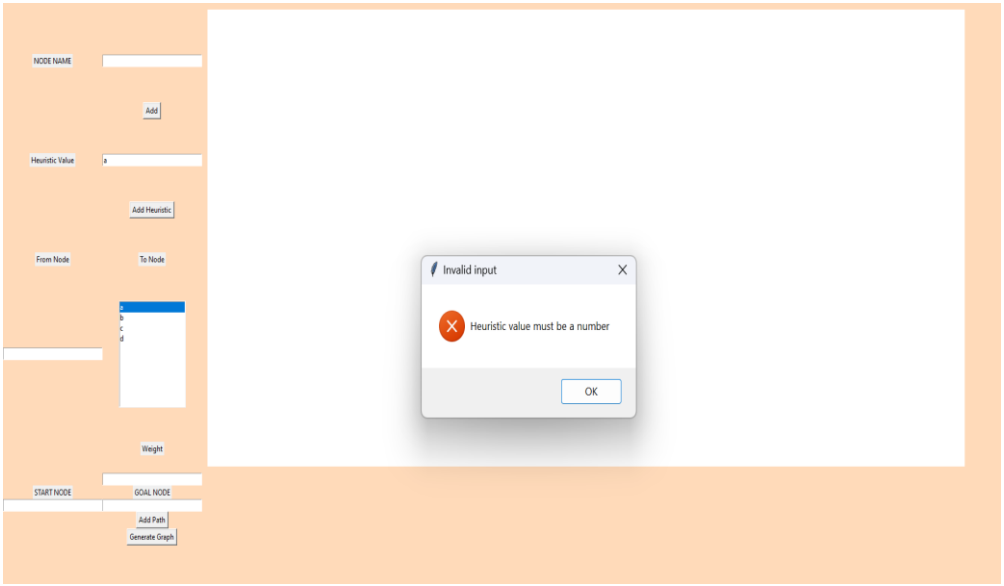
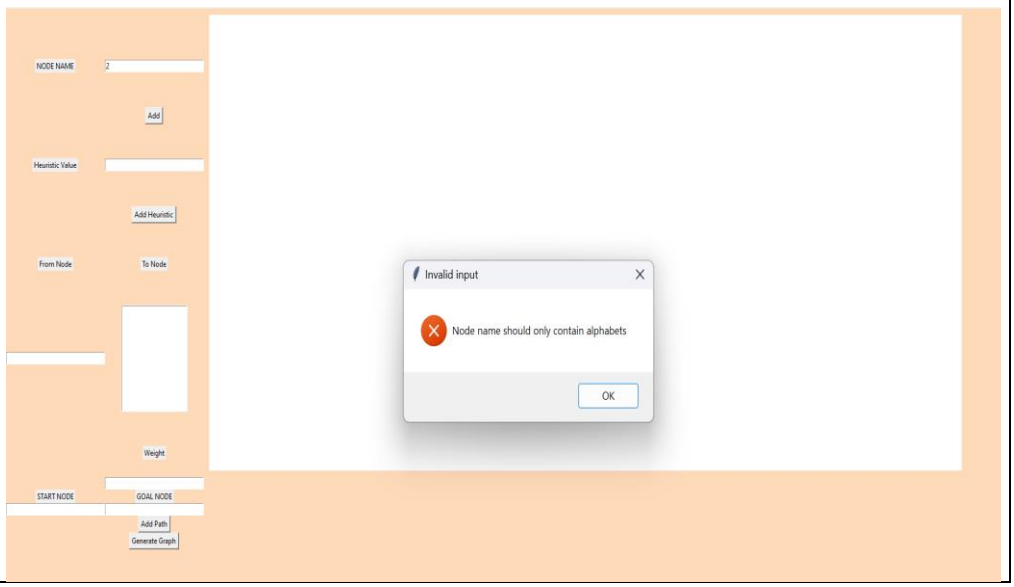


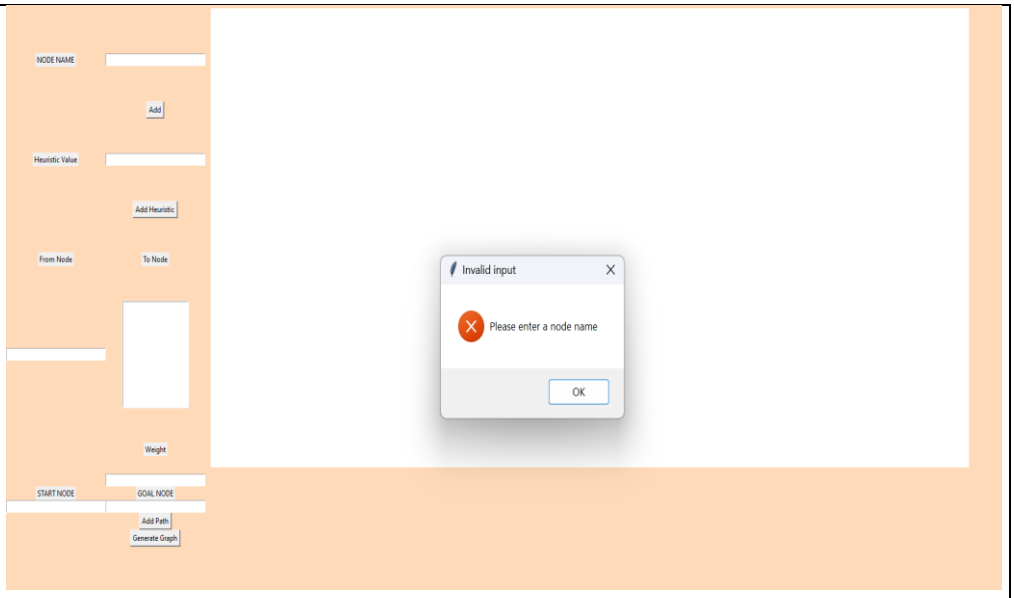
Table 7.3 Checksum

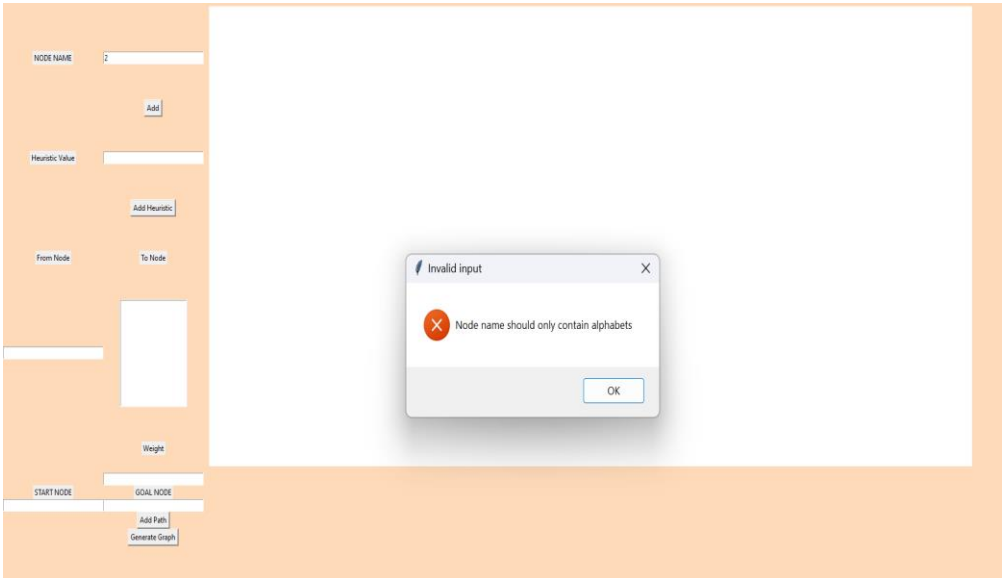
Test Case Title	A Star Algorithm
Purpose of Testing	Purpose is to calculate optimal cost path from start node to goal node.
Test Data	Click on the generate graph button.
Steps	1.Add the nodes. 2.Add heuristic values. 3.Add path between each node with weight. 4.Enter the start node and goal node and click on the generate graph button. 5.Displays the optimal cost path.
Expected Output	Valid Output:  

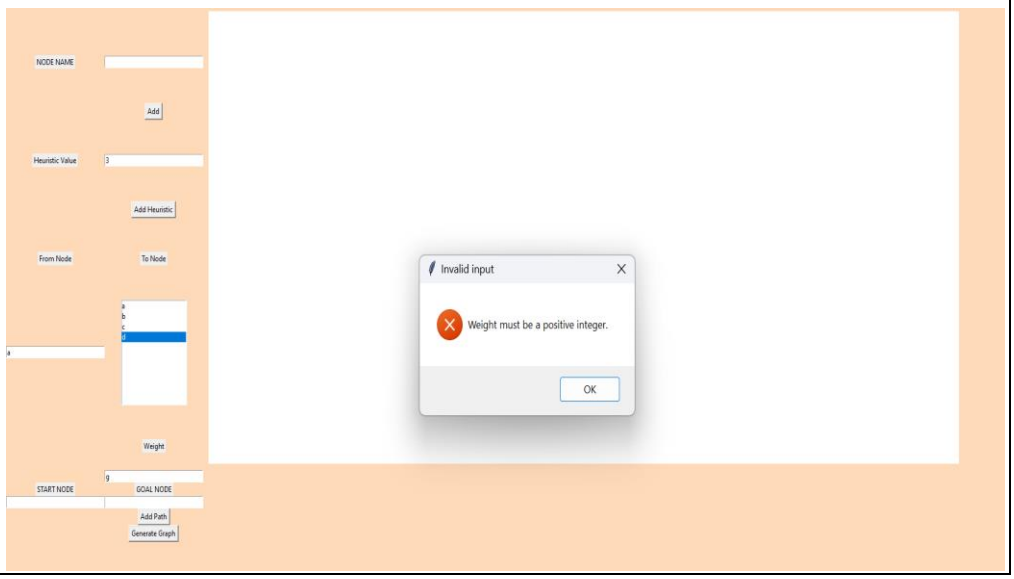

NODE NAME
Add
Heuristic Value
Add Heuristic
From Node
To Node
Weight
START NODE
GOAL NODE
Add Path
Generate Graph
Invalid input
Node name should only contain alphabets
OK

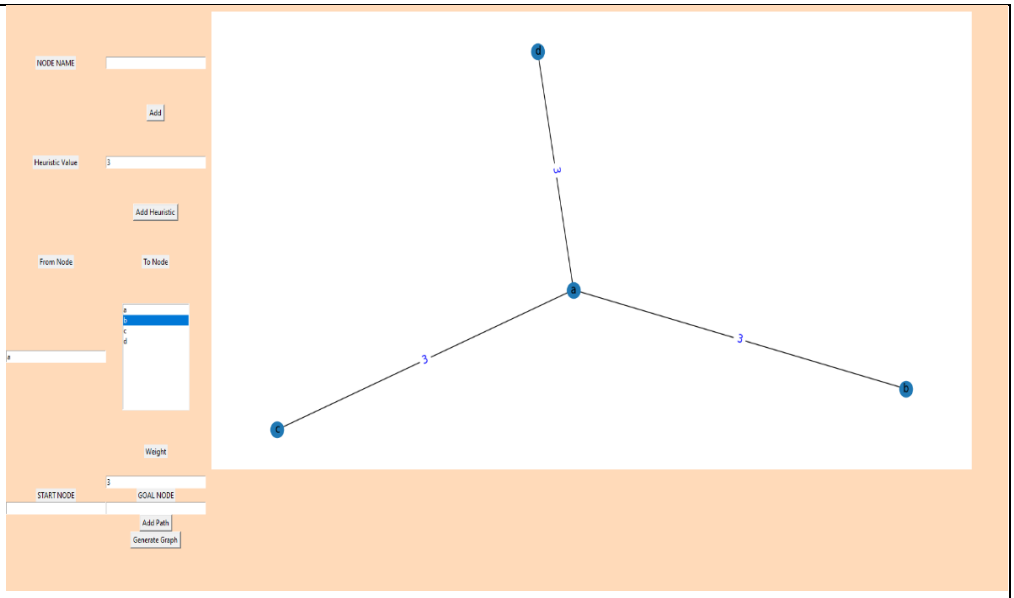

NODE NAME
Add
Heuristic Value
Add Heuristic
From Node
To Node
Weight
START NODE
GOAL NODE
Add Path
Generate Graph
Invalid input
Heuristic value must be a number
OK

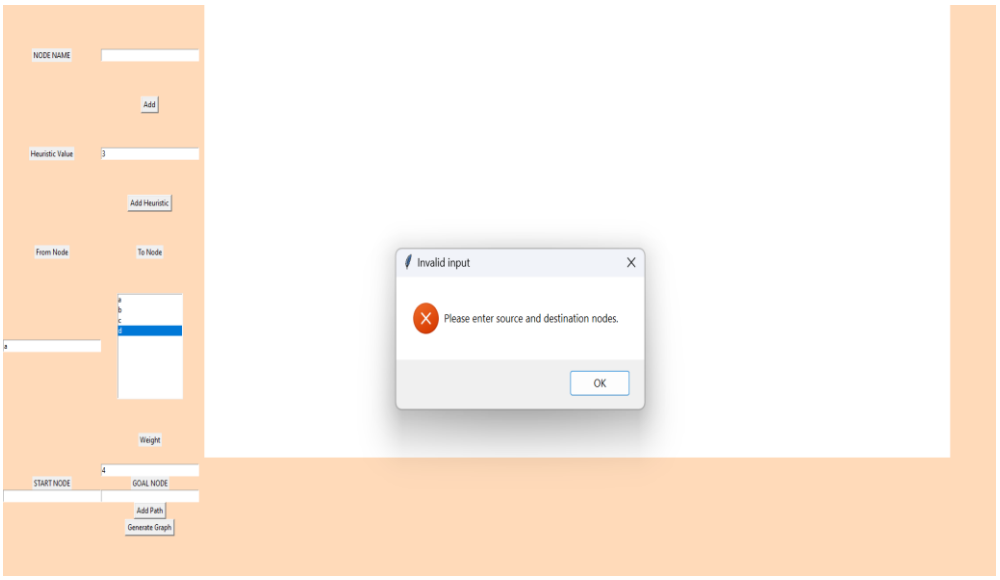

NODE NAME
Add
Heuristic Value
Add Heuristic
From Node
To Node
Weight
START NODE
GOAL NODE
Add Path
Generate Graph
Invalid input
Node name should only contain alphabets
OK


NODE NAME
Add
Heuristic Value
Add Heuristic
From Node
To Node
Weight
START NODE
GOAL NODE
Add Path
Generate Graph
Invalid input
Please enter a node name
OK


NODE NAME
2
Add
Heuristic Value
Add Heuristic
From Node
To Node
Weight
START NODE
GOAL NODE
Add Path
Generate Graph
Invalid input
Node name should only contain alphabets
OK


NODE NAME
Add
Heuristic Value
3
Add Heuristic
From Node
To Node
a
b
c
Weight
g
START NODE
GOAL NODE
Add Path
Generate Graph
Invalid input
Weight must be a positive integer.
OK


NODE NAME
Add
Heuristic Value
3
Add Heuristic
From Node
To Node
Weight
3
START NODE
GOAL NODE
Add Path
Generate Graph
d
3
a
3
3
c
b


NODE NAME
Add
Heuristic Value
3
Add Heuristic
From Node
To Node
Weight
4
START NODE
GOAL NODE
Add Path
Generate Graph
Invalid input
Please enter source and destination nodes.
OK

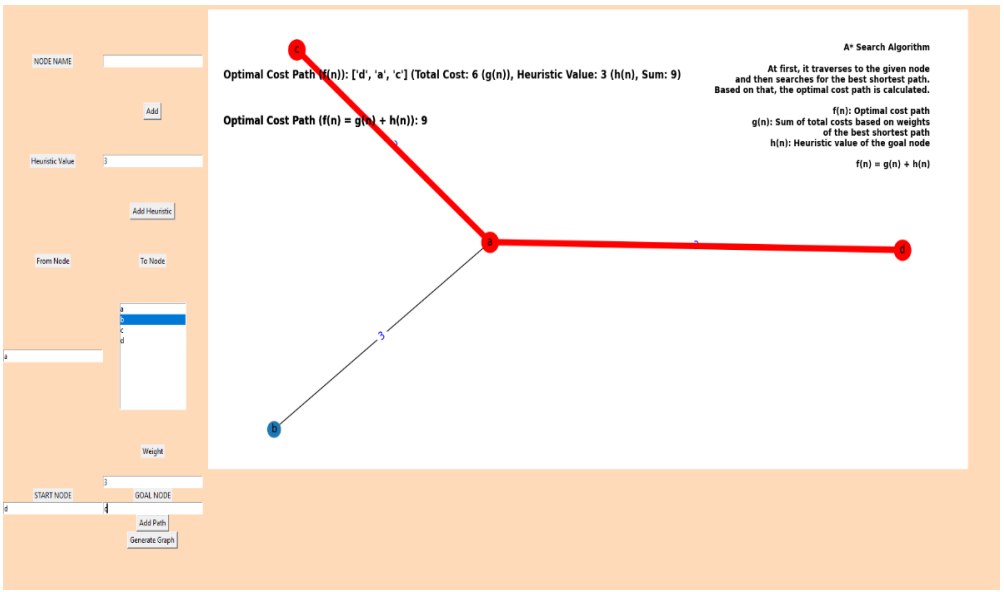
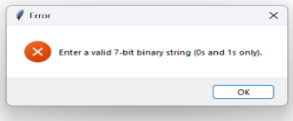
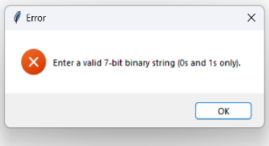

NODE NAME
Add
Heuristic Value
3
Add Heuristic
From Node
To Node
Weight
3
START NODE
GOAL NODE
Add Path
Generate Graph
Optimal Cost Path (f(n)): ['d', 'a', 'c'] (Total Cost: 6 (g(n)), Heuristic Value: 3 (h(n), Sum: 9)
Optimal Cost Path (f(n) = g(n) + h(n)): 9
A* Search Algorithm
At first, it traverses to the given node
and then searches for the best shortest path.
Based on that, the optimal cost path is calculated.
f(n): Optimal cost path
g(n): Sum of total costs based on weights
of the best shortest path
h(n): Heuristic value of the goal node
f(n) = g(n) + h(n)
c
a
d
3
b

Table 7.4 A Star Algorithm

Test Case Title	Hamming Code
Purpose of Testing	Purpose is to detect the error in the parity bits and error position in the received code.
Test Data	Click on the calculate Simulate Button
Steps	1.Enter the 7-bit hamming code. 2.If it is valid then it detects error in the parity bits by check-one skip-one method for P1 and so on. 3.If number 1's in the parity bit is odd, then error is detected in the received code, otherwise no error. 4.Then the error position is detected using XOR operation. 5. Displays the position in the error if the received code using XOR operation.
Expected Output	Valid Output:   





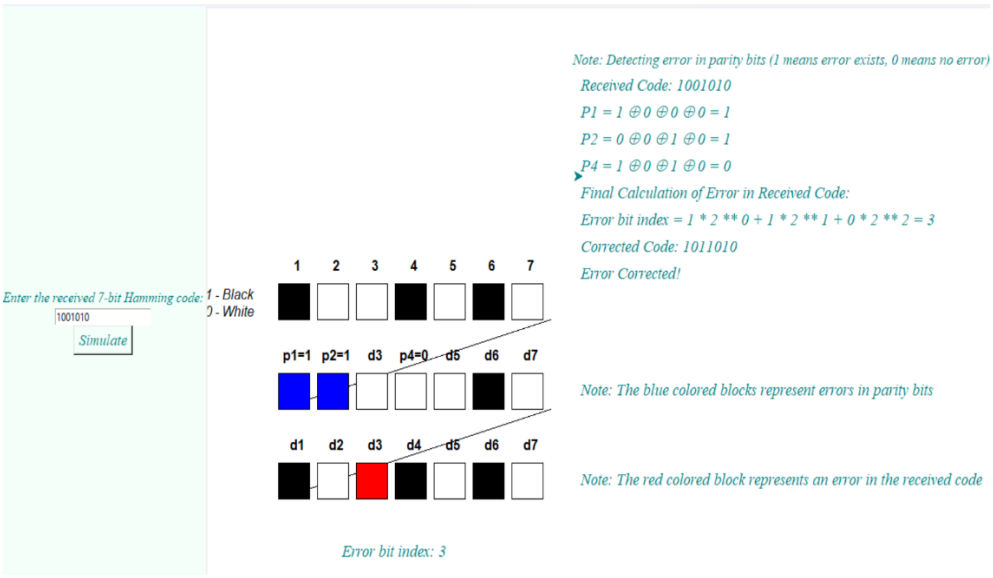
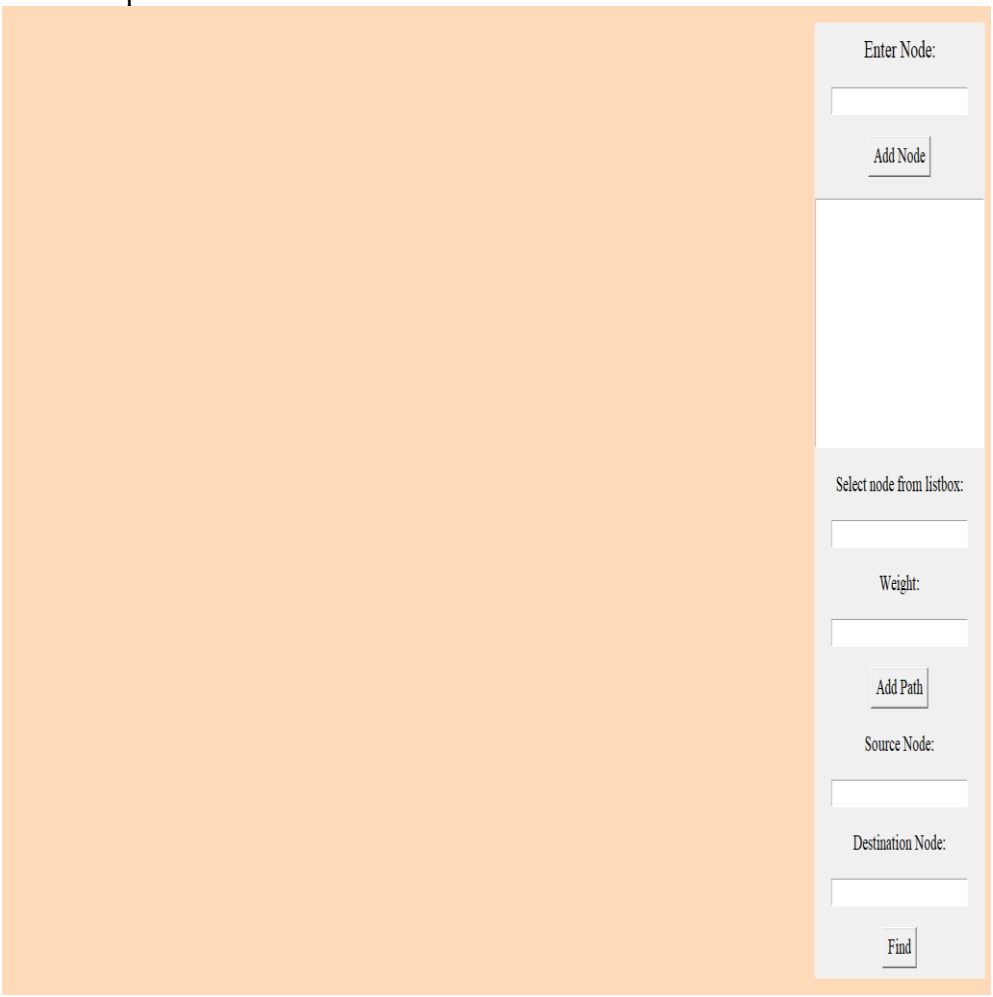
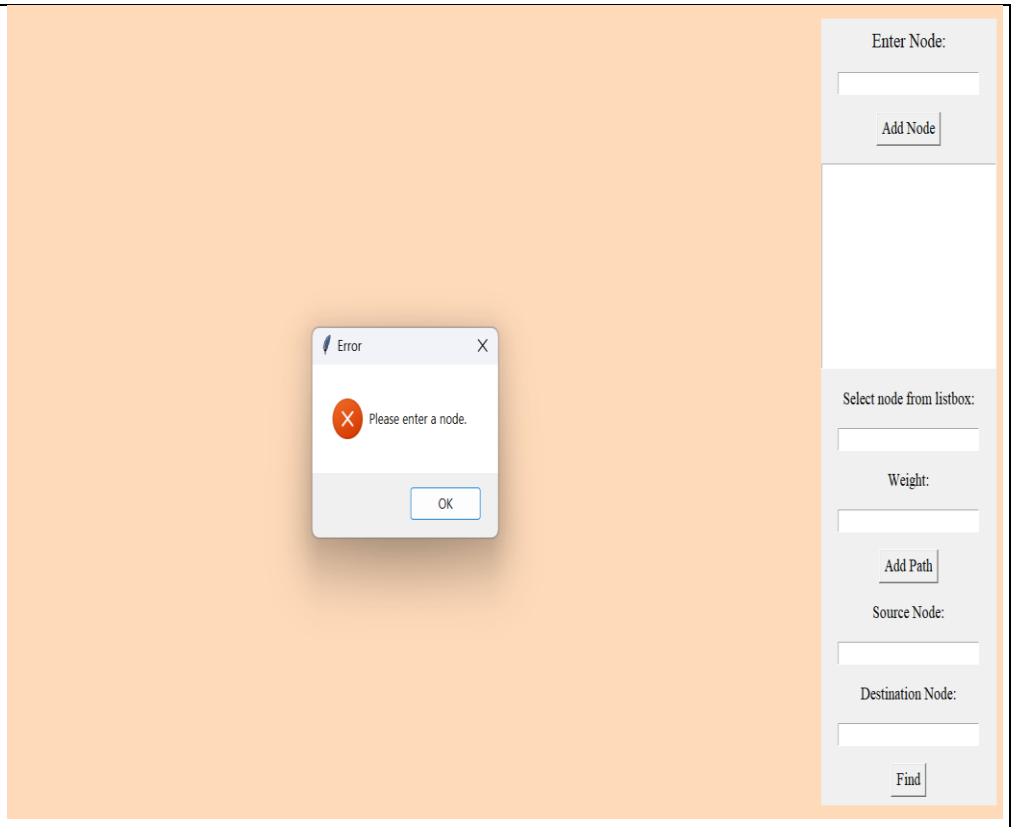
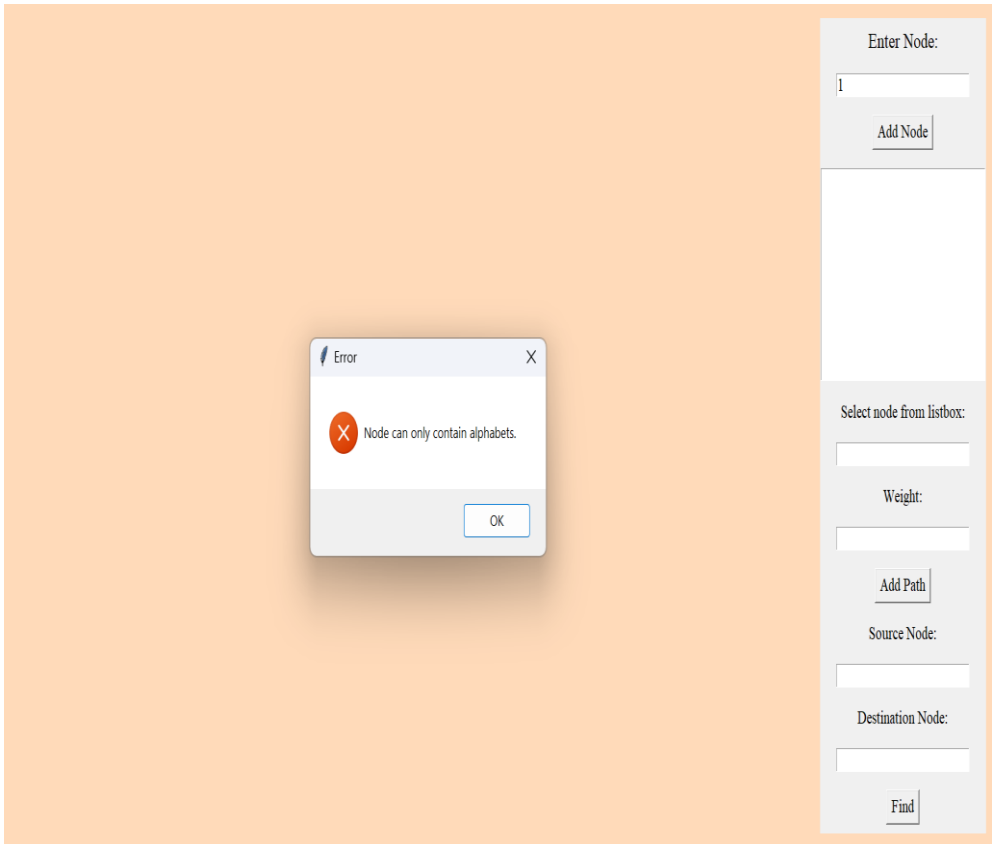
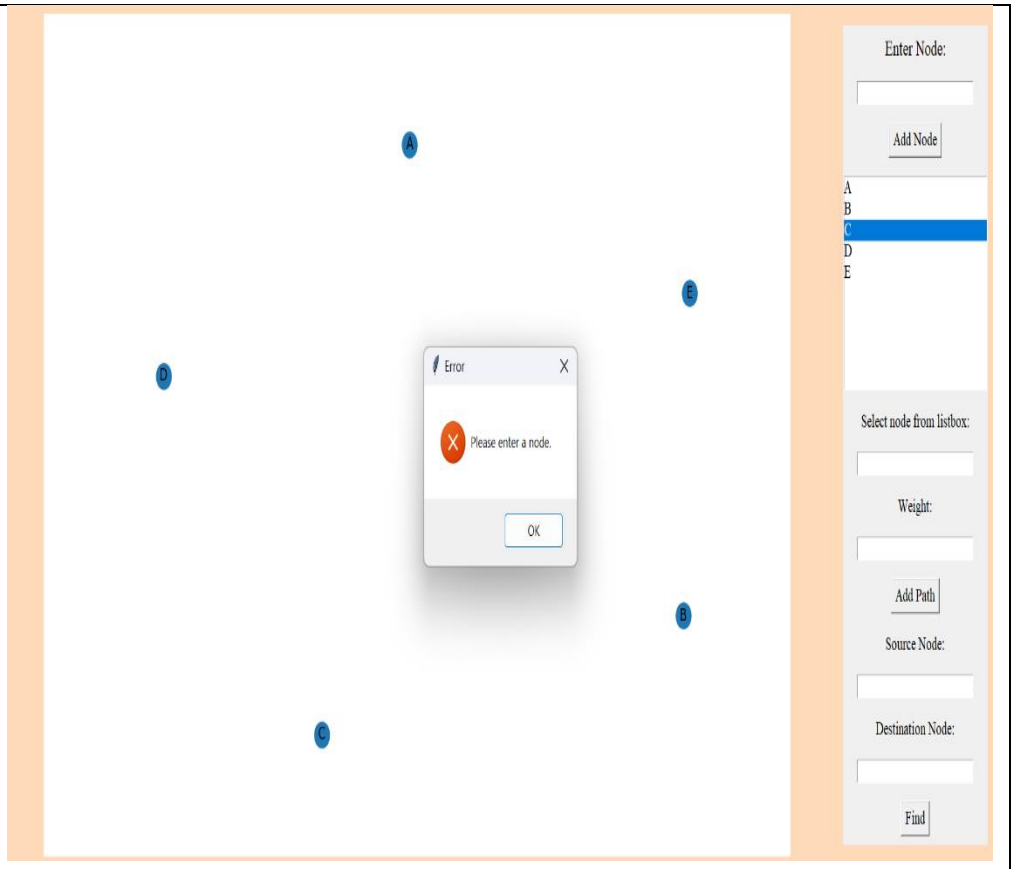


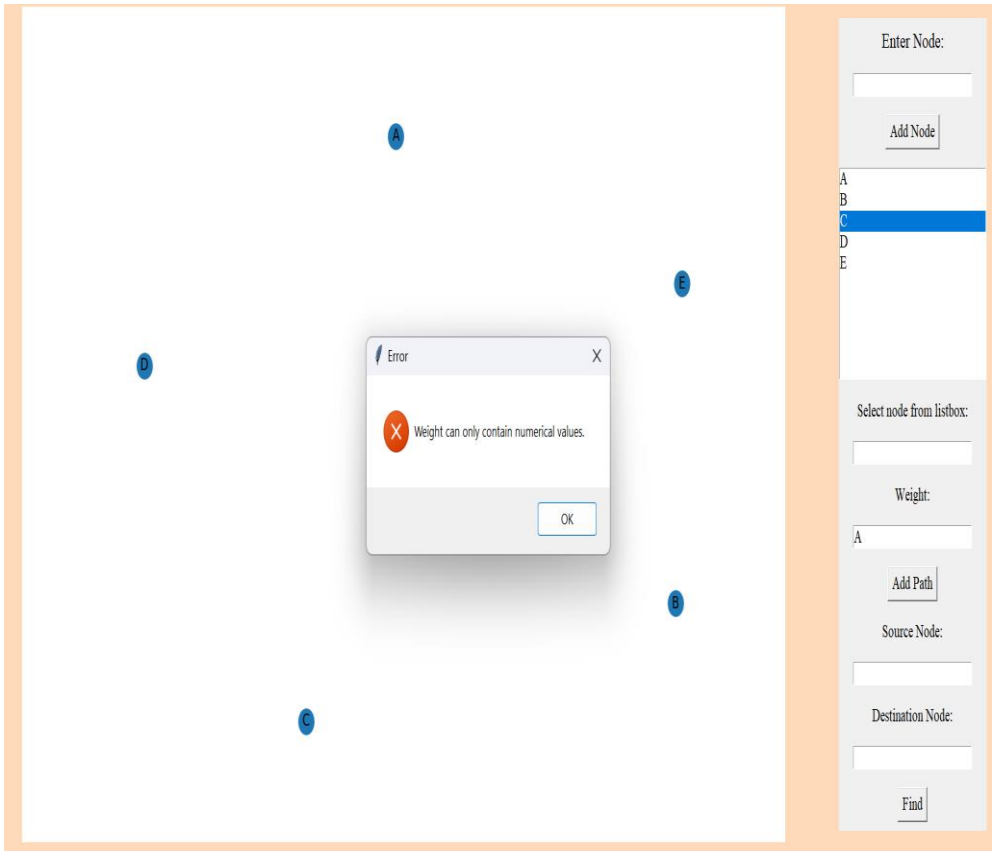
Table 7.5 Hamming Code

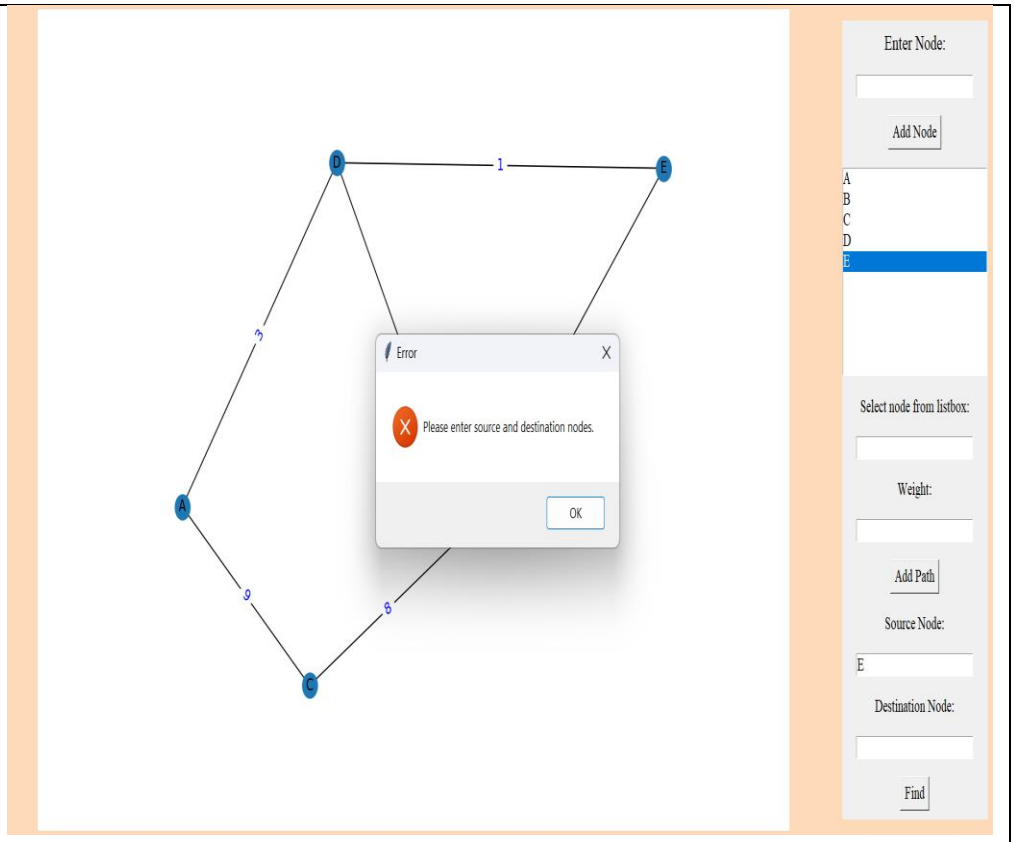
Test Case Title	Path Finding Algorithm (Greedy Kruskal)
Purpose of Testing	Purpose is to find the best path between the source and destination nodes.
Test Data	Click on the calculate Find Button
Steps	1.Input nodes and edges. 2.Checks if the nodes are valid, if valid then add path between the nodes with weight (edges). 3.Input source and destination nodes. 4.Highlights the best path using the greedy Kruskal algorithm in which it finds the Minimum Spanning Tree and traverse it using Depth First Search (DFS). 5.Displays the best path between the source and destination nodes.
Expected Output	Valid Output: 


Enter Node:
Add Node
Error
Please enter a node.
OK
Select node from listbox:
Weight:
Add Path
Source Node:
Destination Node:
Find


Enter Node:
1
Add Node
Error
Node can only contain alphabets.
OK
Select node from listbox:
Weight:
Add Path
Source Node:
Destination Node:
Find


Enter Node:
Add Node
A
B
C
D
E
Error
Please enter a node.
OK
Select node from listbox:
Weight:
Add Path
Source Node:
Destination Node:
Find


Enter Node:
Add Node
A
B
C
D
E
Error
Weight can only contain numerical values.
OK
Select node from listbox:
Weight:
A
Add Path
Source Node:
Destination Node:
Find



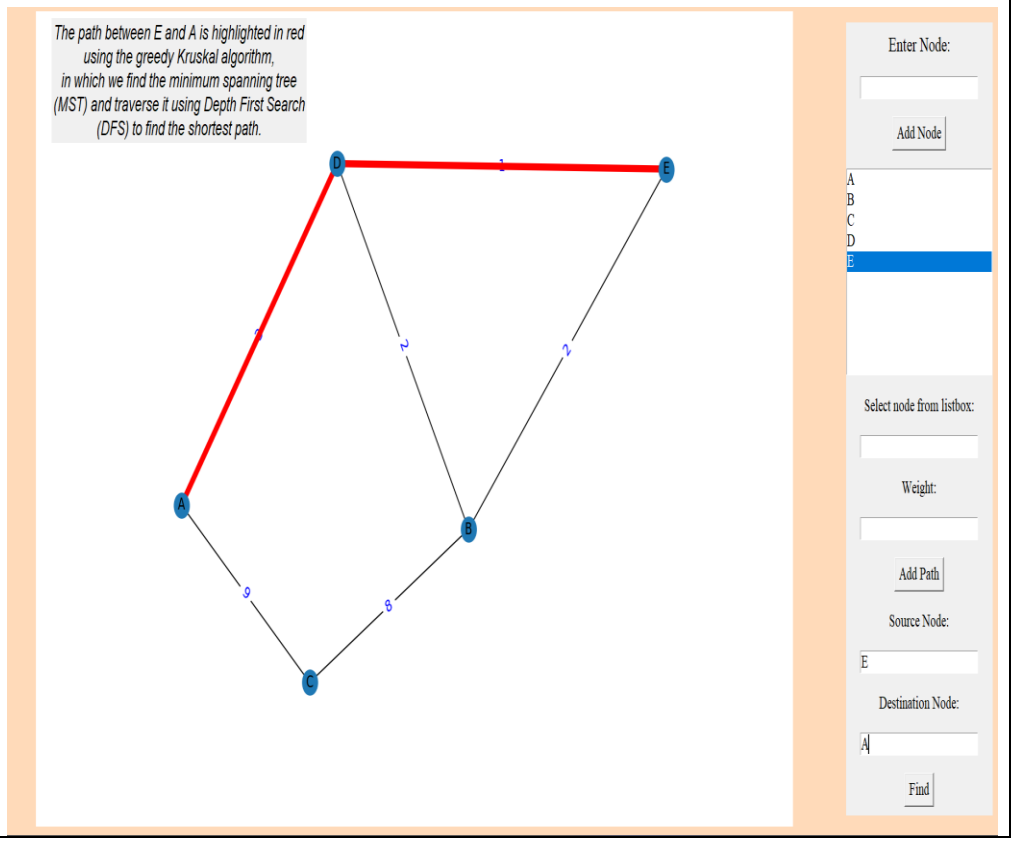
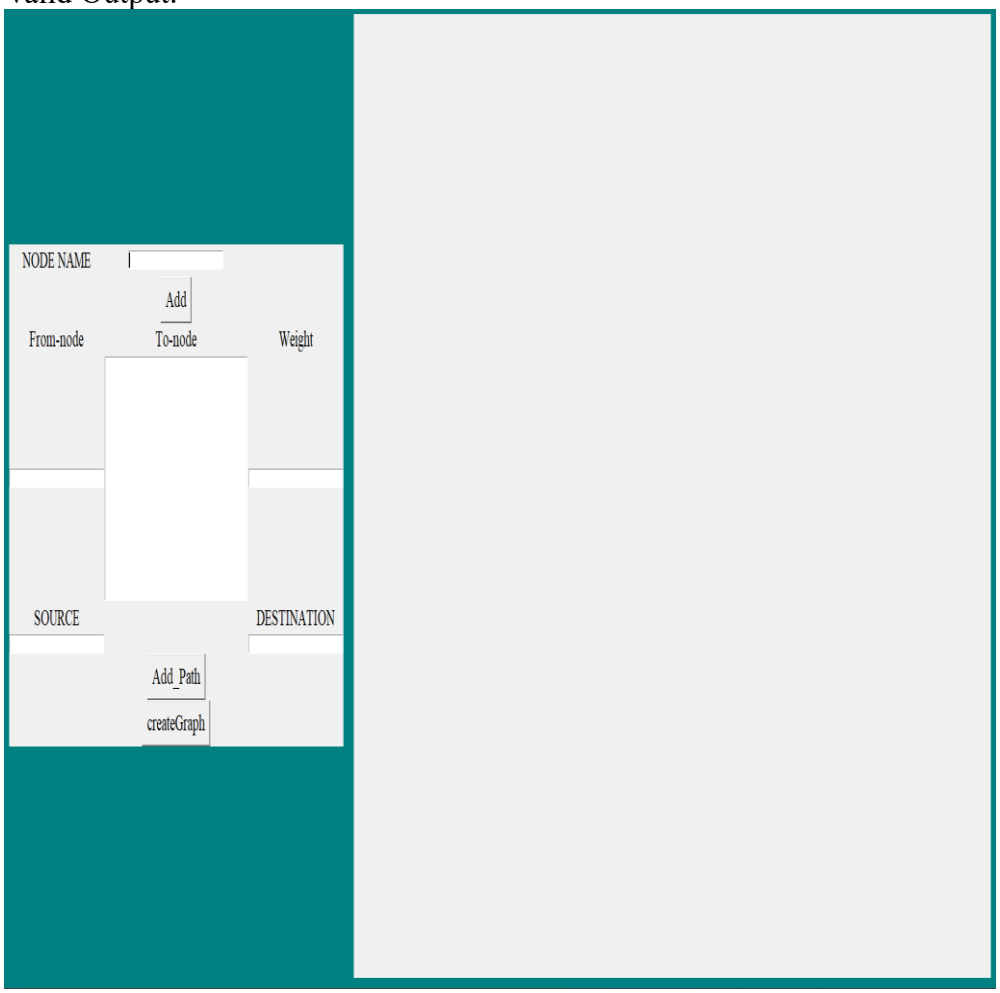
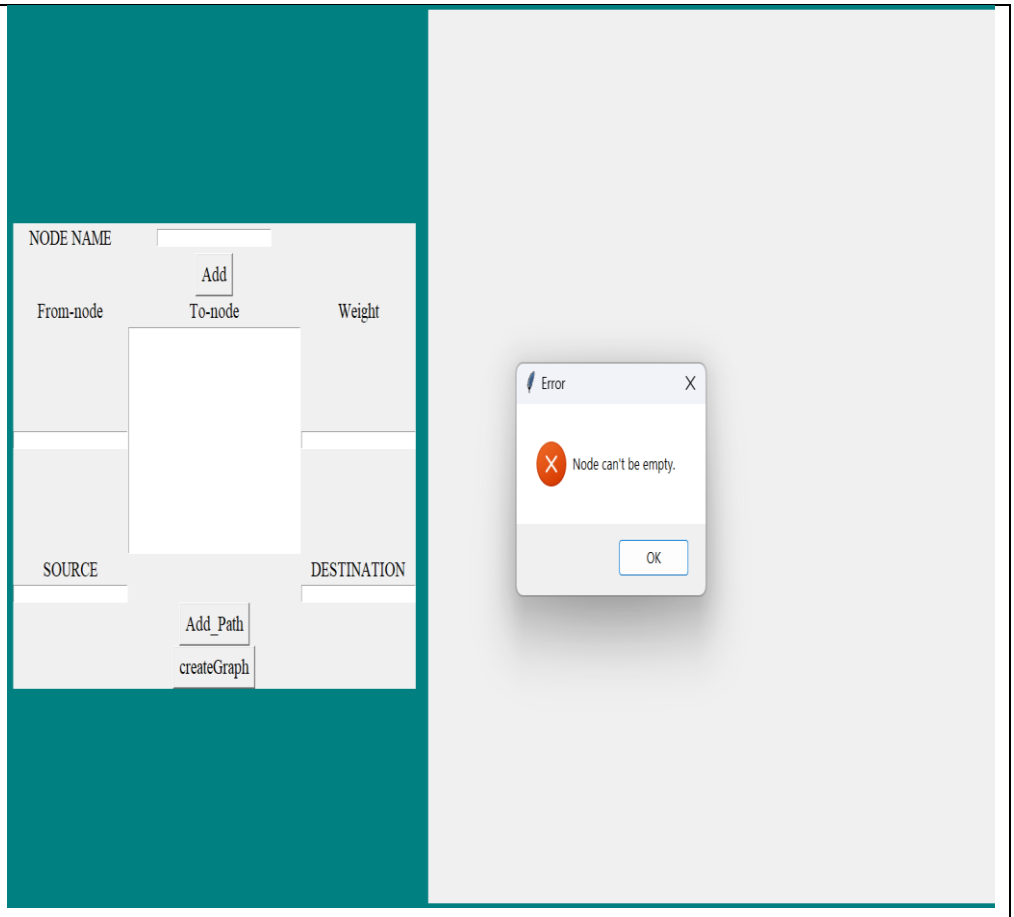
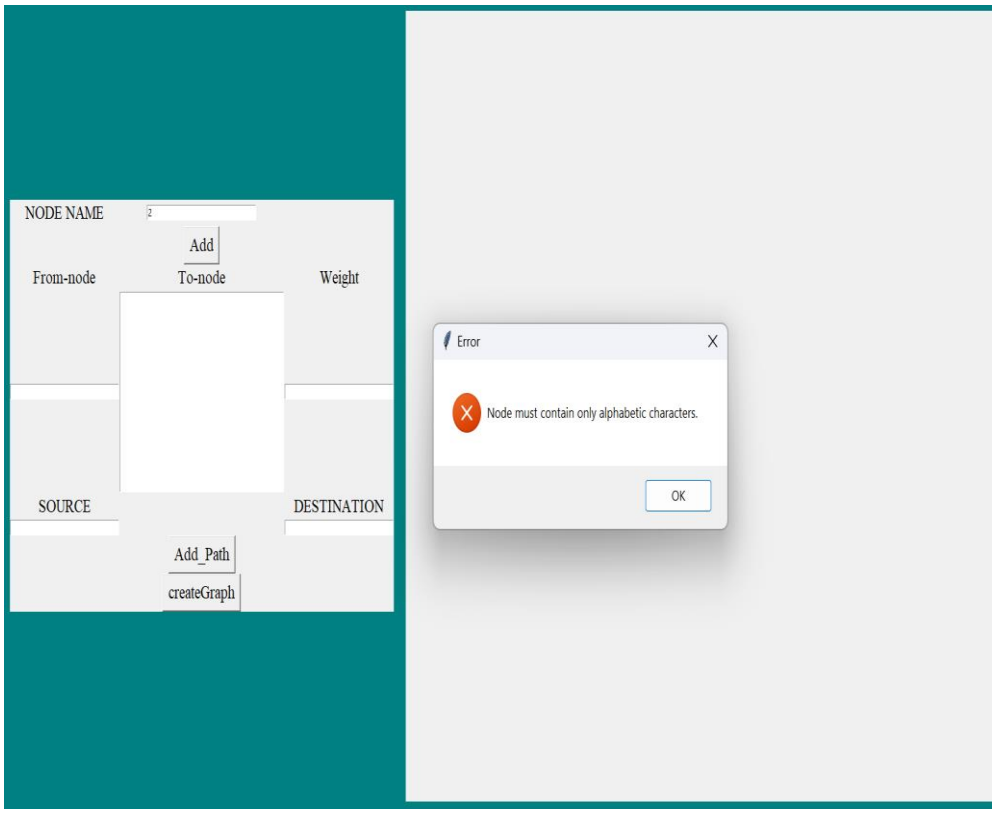
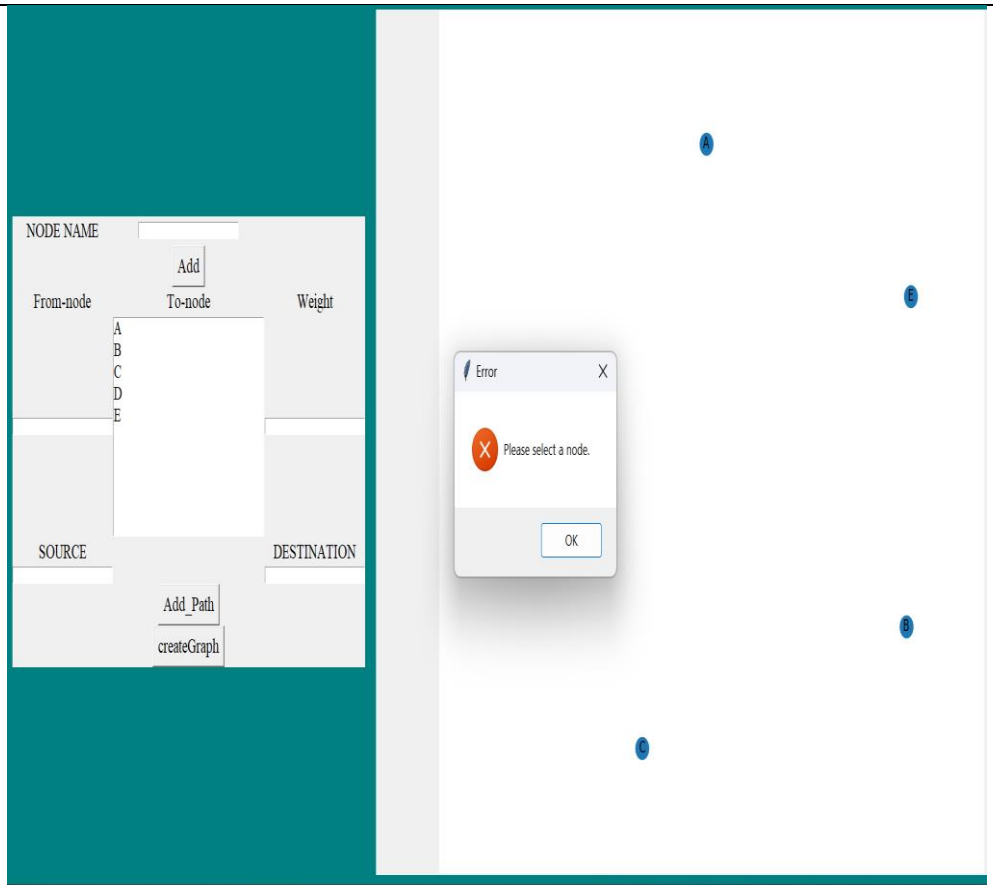


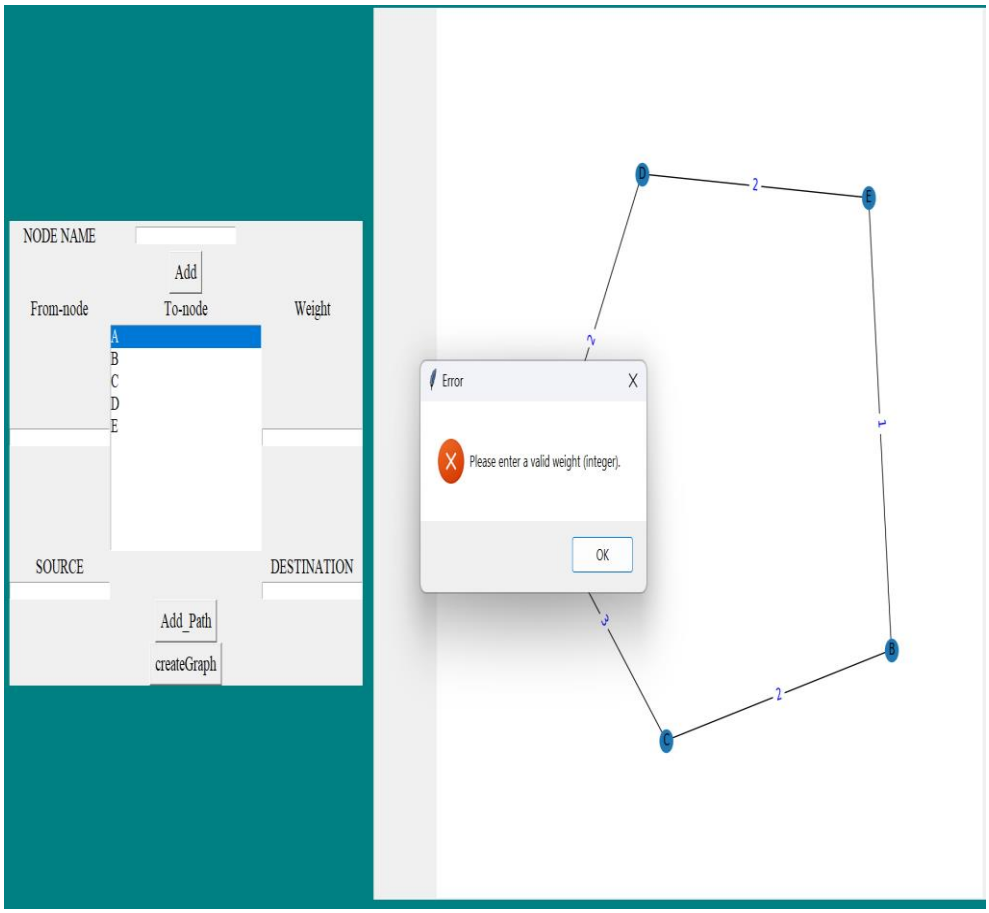
Table 7.6 Path Finding Algorithm

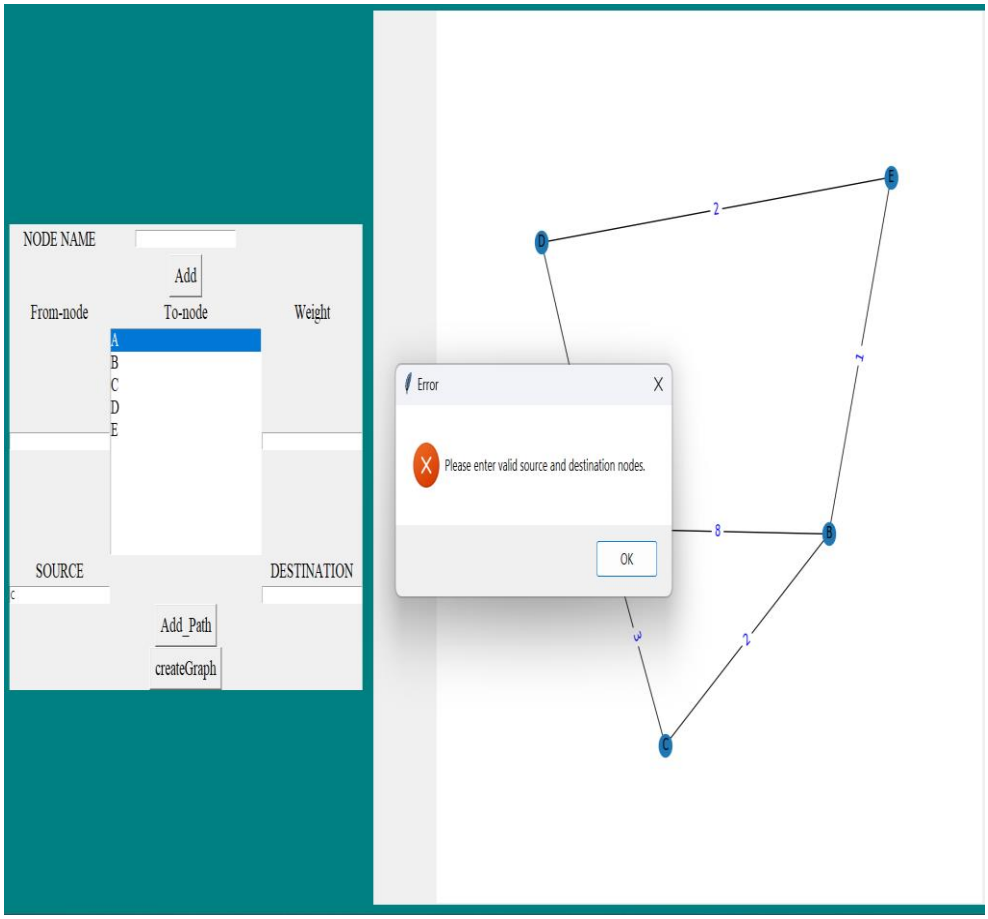
Test Case Title	Shortest Path First (Dijkstra's)
Purpose of Testing	Purpose is to find the shortest path between the source and destination nodes.
Test Data	Click on the calculate Create_Graph Button
Steps	1.Input nodes and edges. 2.Checks if the nodes are valid, if valid then add path between the nodes with weight (edges). 3.Input source and destination nodes. 4.Shortest path is calculated using the Dijkstra's Algorithm by summing the edge weights and highlights the path whose sum is less compared to others. 5.Displays the shortest Best Path.
Expected Output	Valid Output: 


NODE NAME
Add
From-node
To-node
Weight
SOURCE
DESTINATION
Add_Path
createGraph
Error
Node can't be empty.
OK


NODE NAME
Add
From-node
To-node
Weight
SOURCE
DESTINATION
Add_Path
createGraph
Error
Node must contain only alphabetic characters.
OK


NODE NAME
Add
From-node
To-node
Weight
A
B
C
D
E
SOURCE
DESTINATION
Add_Path
createGraph
Error
Please select a node.
OK
A
E
B
C


NODE NAME
Add
From-node
To-node
Weight
A
B
C
D
E
SOURCE
DESTINATION
Add_Path
createGraph
Error
Please enter a valid weight (integer).
OK
D
2
E
2
1
3
B
2
C



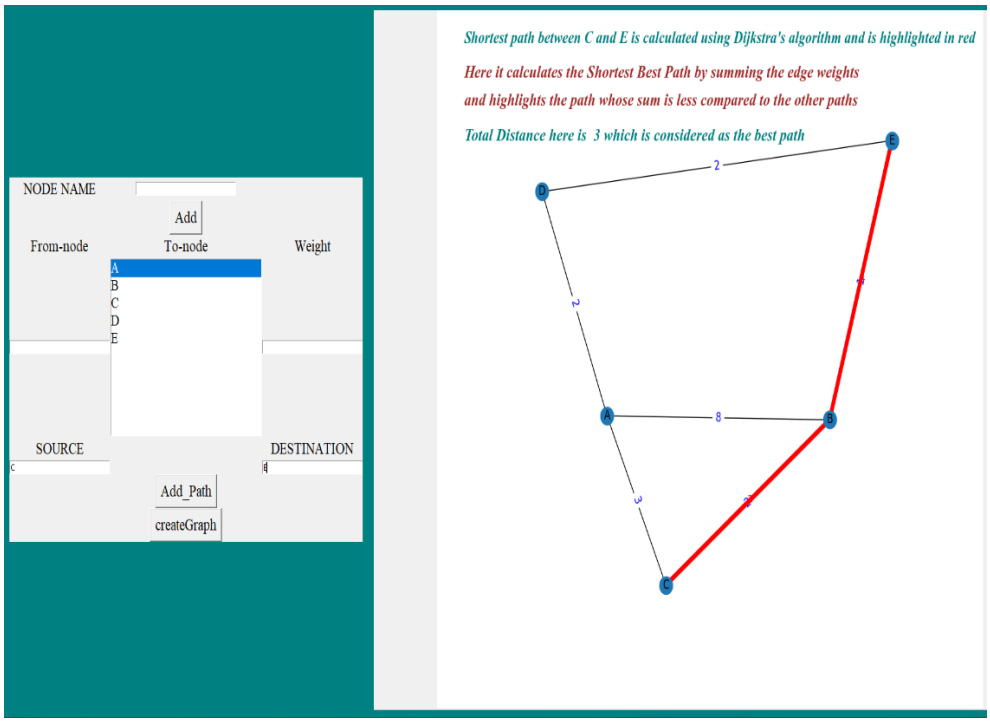
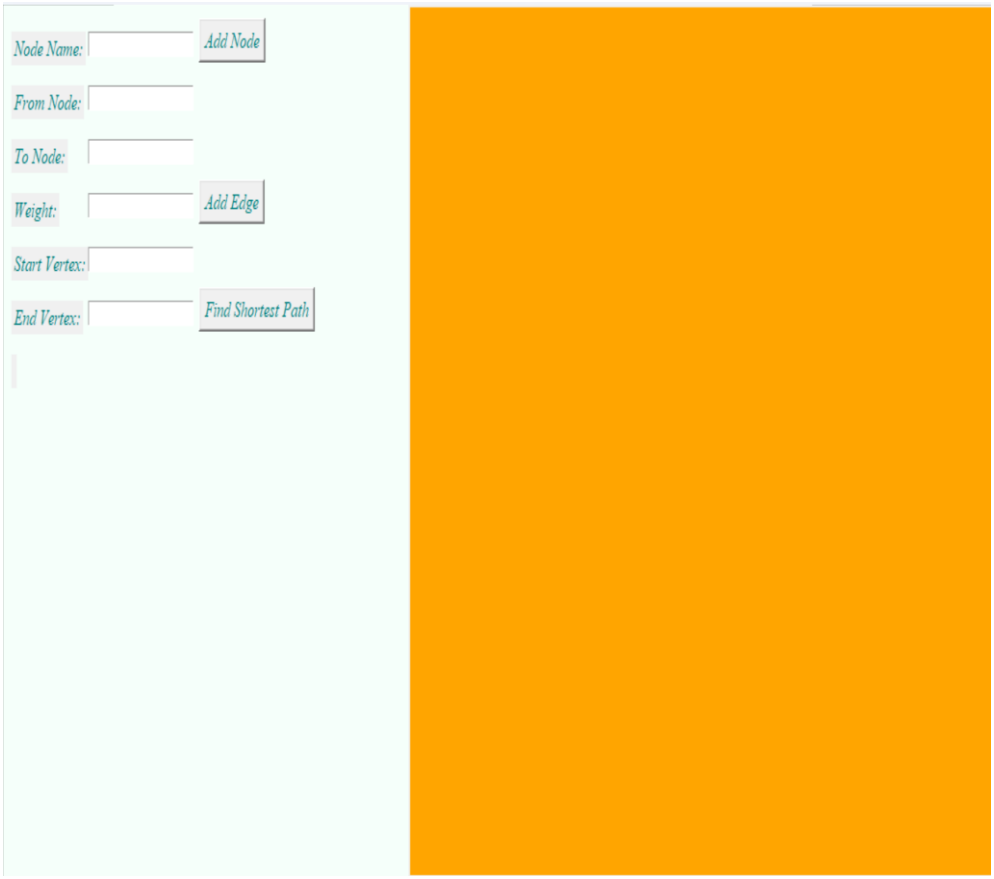
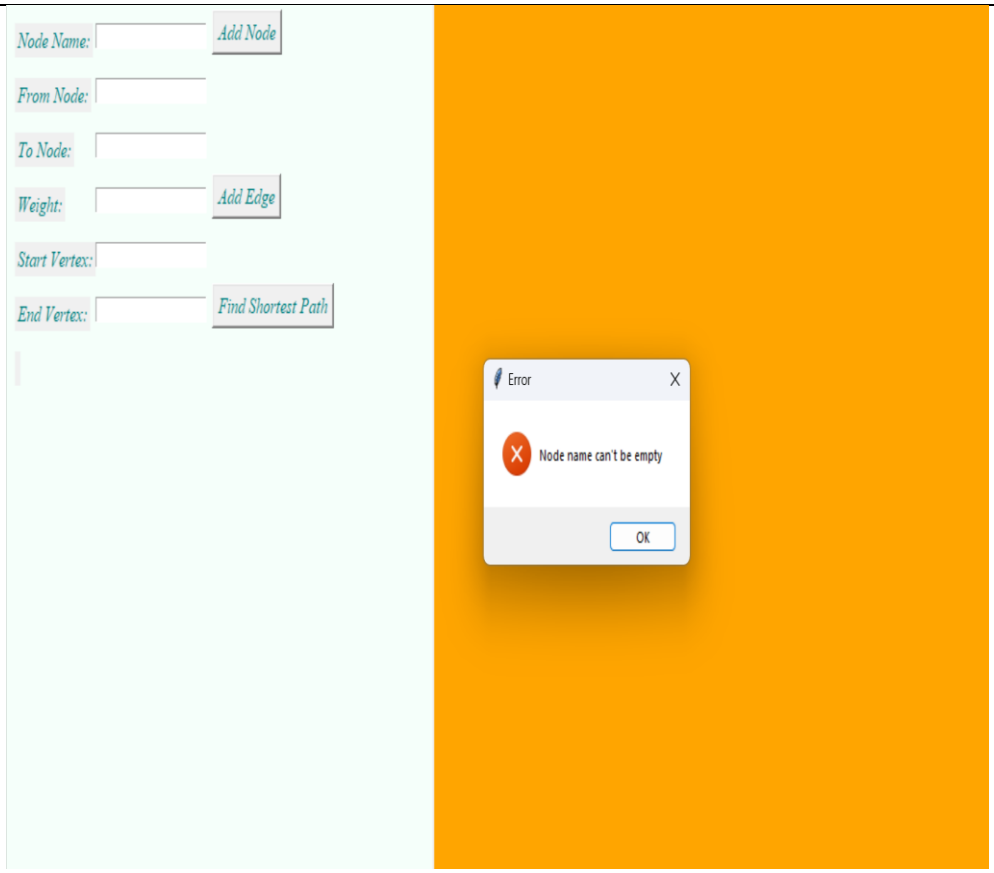
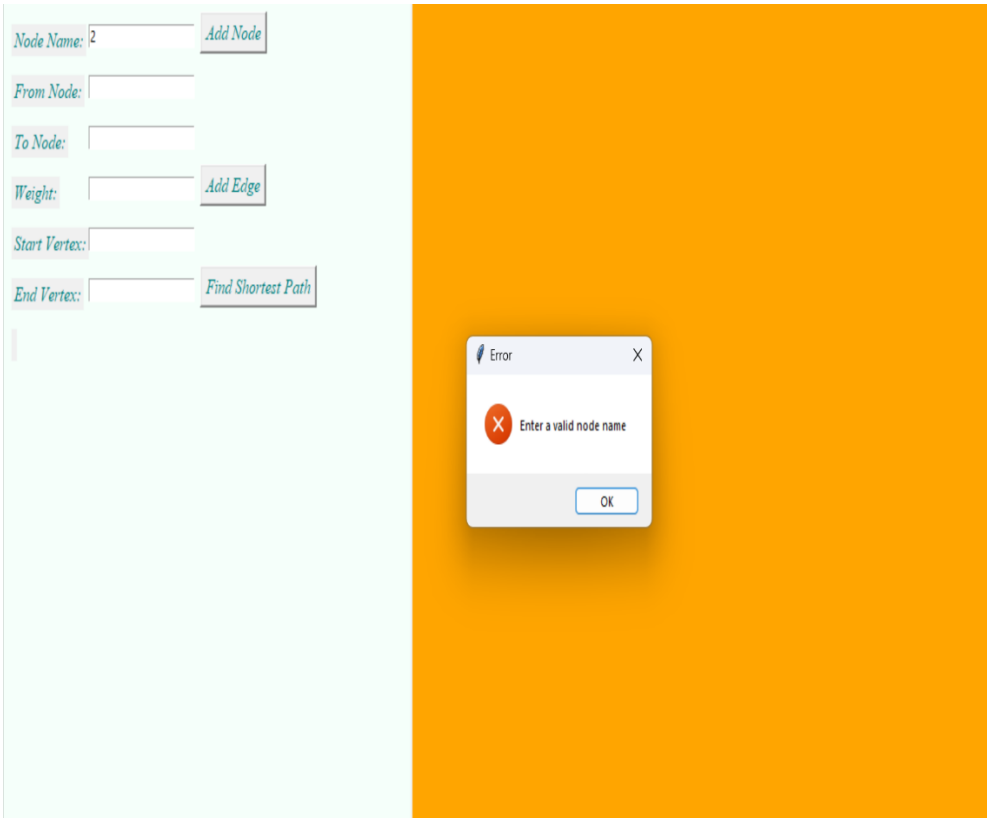
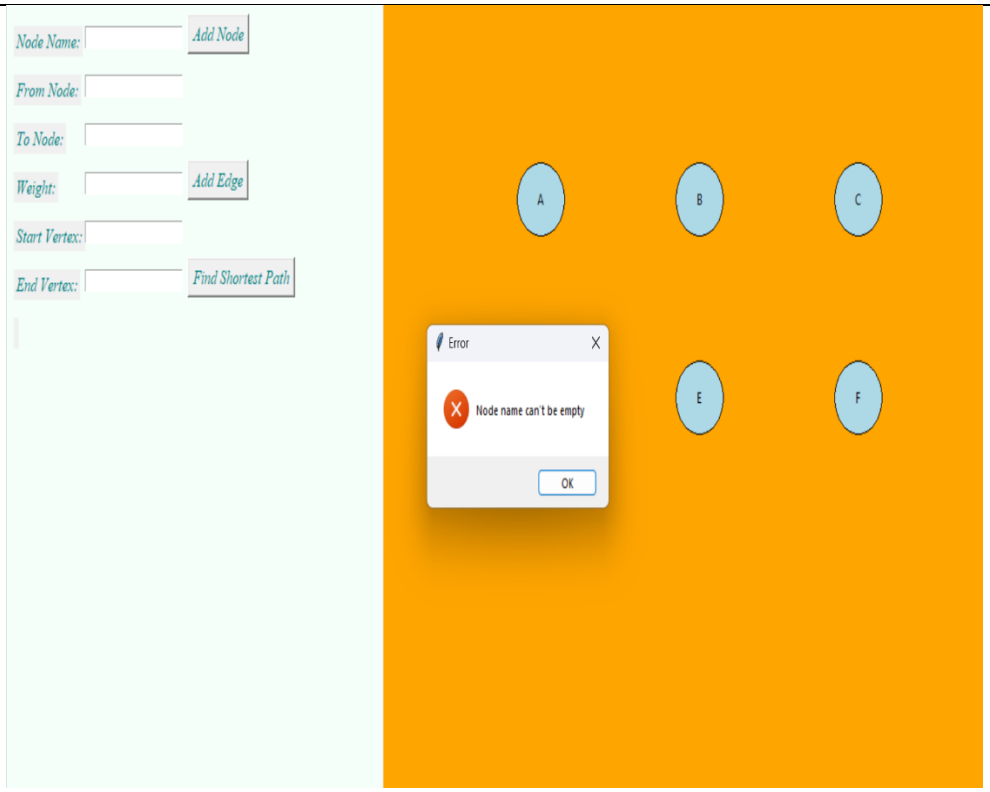


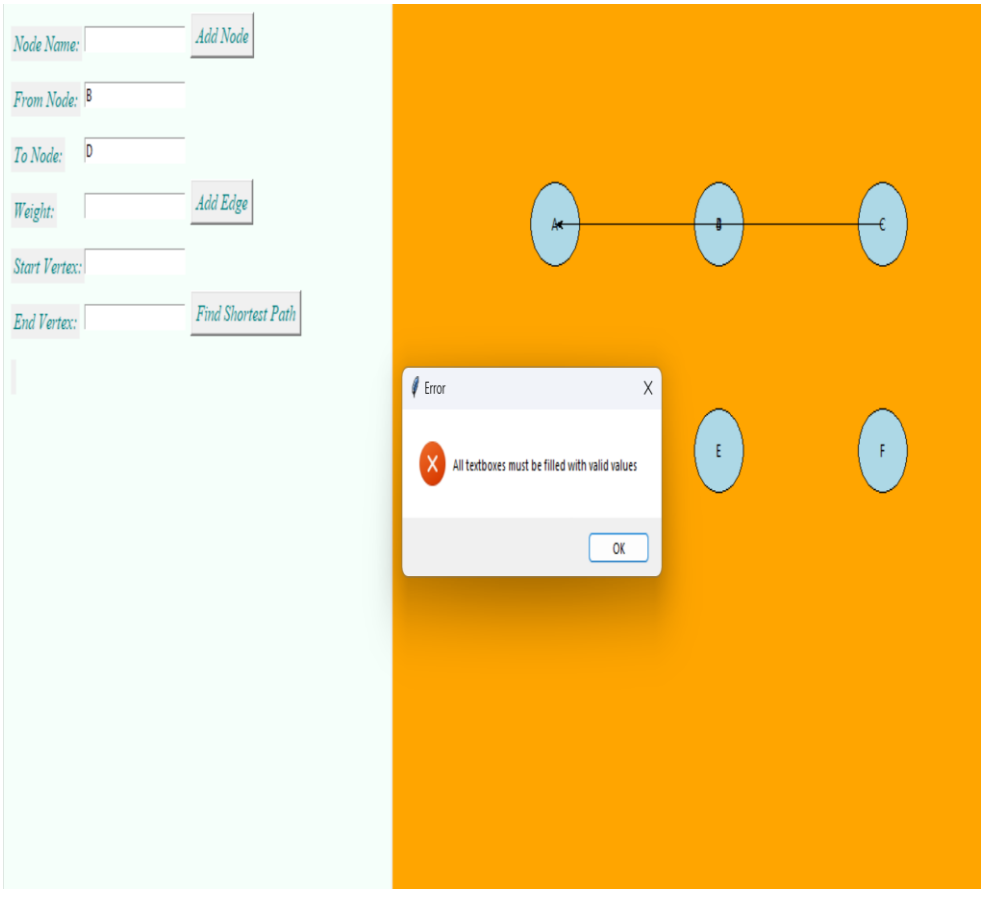
Table 7.7 Shortest Path First

Test Case Title	Open Shortest Path First
Purpose of Testing	Purpose is to find the shortest distance between the source and destination nodes.
Test Data	Click on the Find Shortest Path Button
Steps	1.Input nodes and edges. 2.Checks if the nodes are valid, if valid then add path between the nodes with weight (edges). 3.Input source and destination nodes. 4.Shortest path is calculated using the Dijkstra's Algorithm by summing the edge weights and highlights the path whose sum is less compared to others. 5.Displays the shortest Best Path.
Expected Output	Valid Output: 


Node Name:
Add Node
From Node:
To Node:
Weight:
Add Edge
Start Vertex:
End Vertex:
Find Shortest Path
Error
X
Node name can't be empty
OK


Node Name:
2
Add Node
From Node:
To Node:
Weight:
Add Edge
Start Vertex:
End Vertex:
Find Shortest Path
Error
X
Enter a valid node name
OK


Node Name:
Add Node
From Node:
To Node:
Weight:
Add Edge
Start Vertex:
End Vertex:
Find Shortest Path
A
B
C
E
F
Error
Node name can't be empty
OK


Node Name:
Add Node
From Node:
B
To Node:
D
Weight:
Add Edge
Start Vertex:
End Vertex:
Find Shortest Path
A
B
C
E
F
Error
All textboxes must be filled with valid values
OK

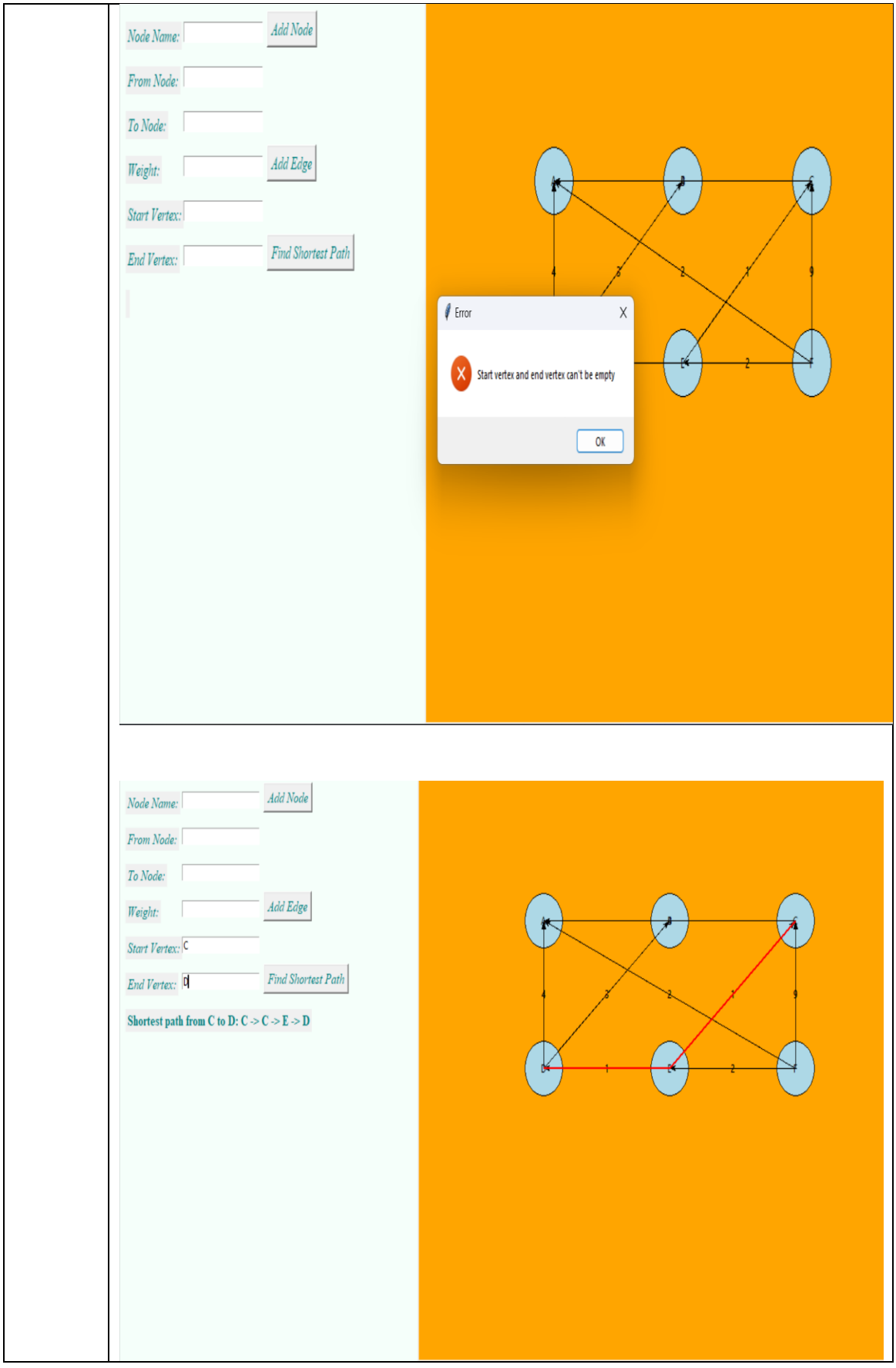
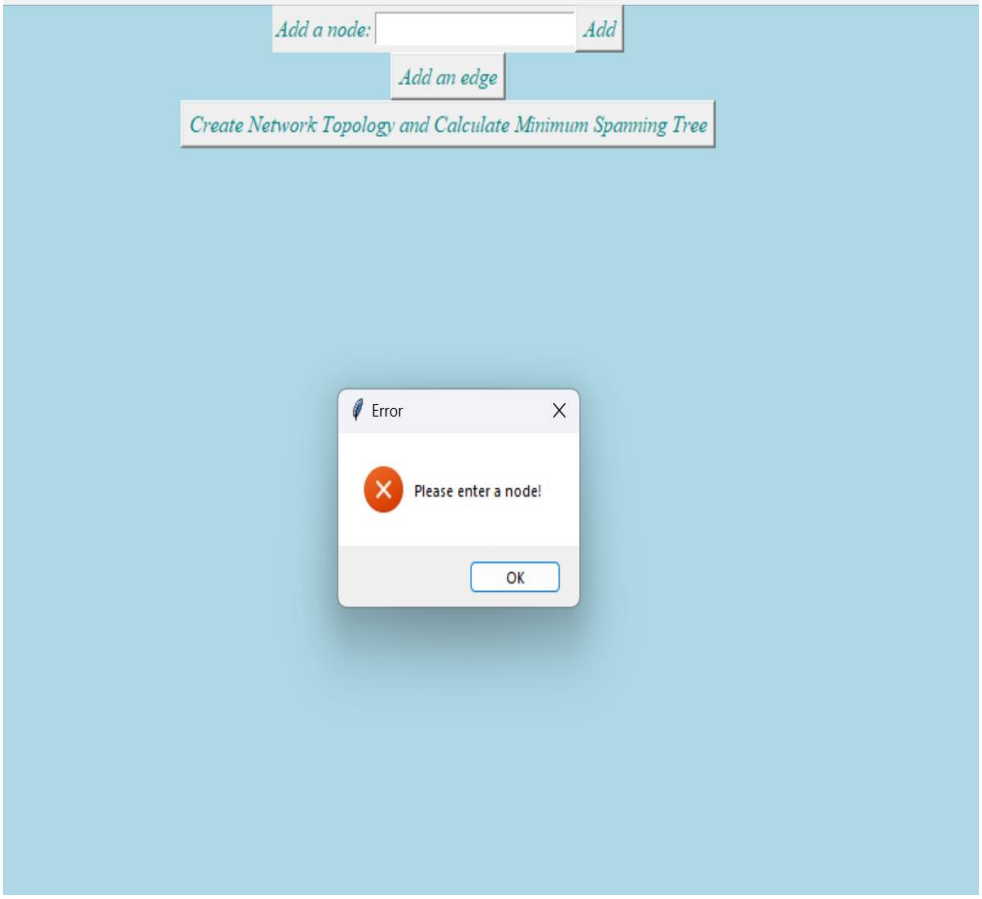
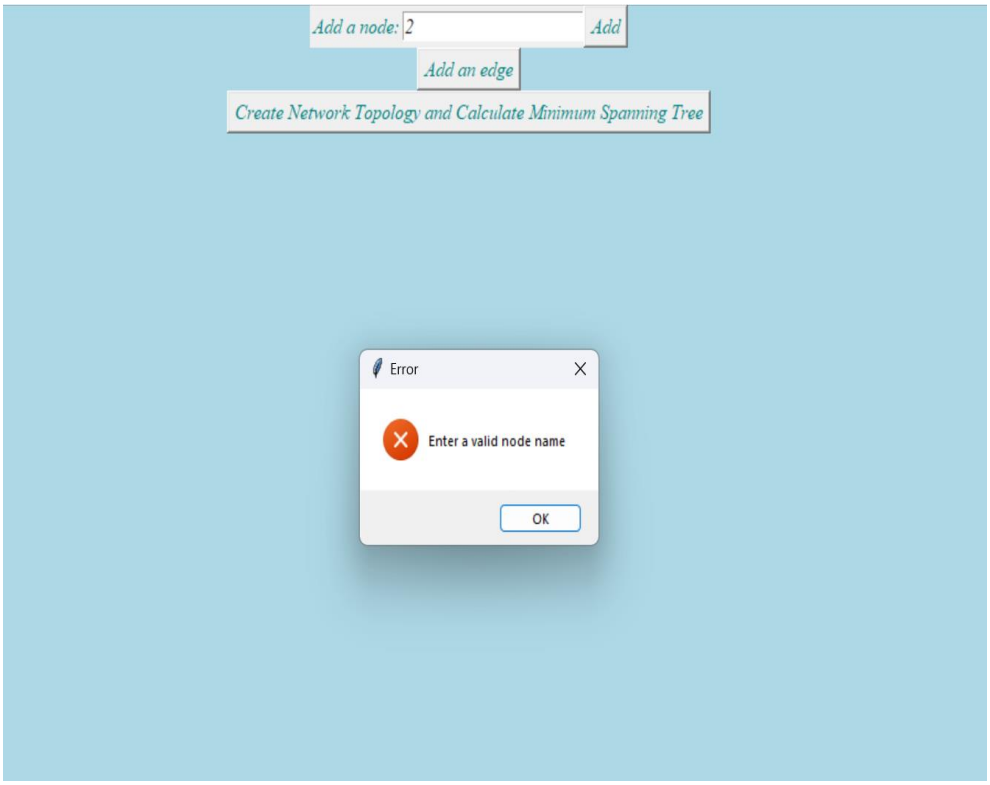
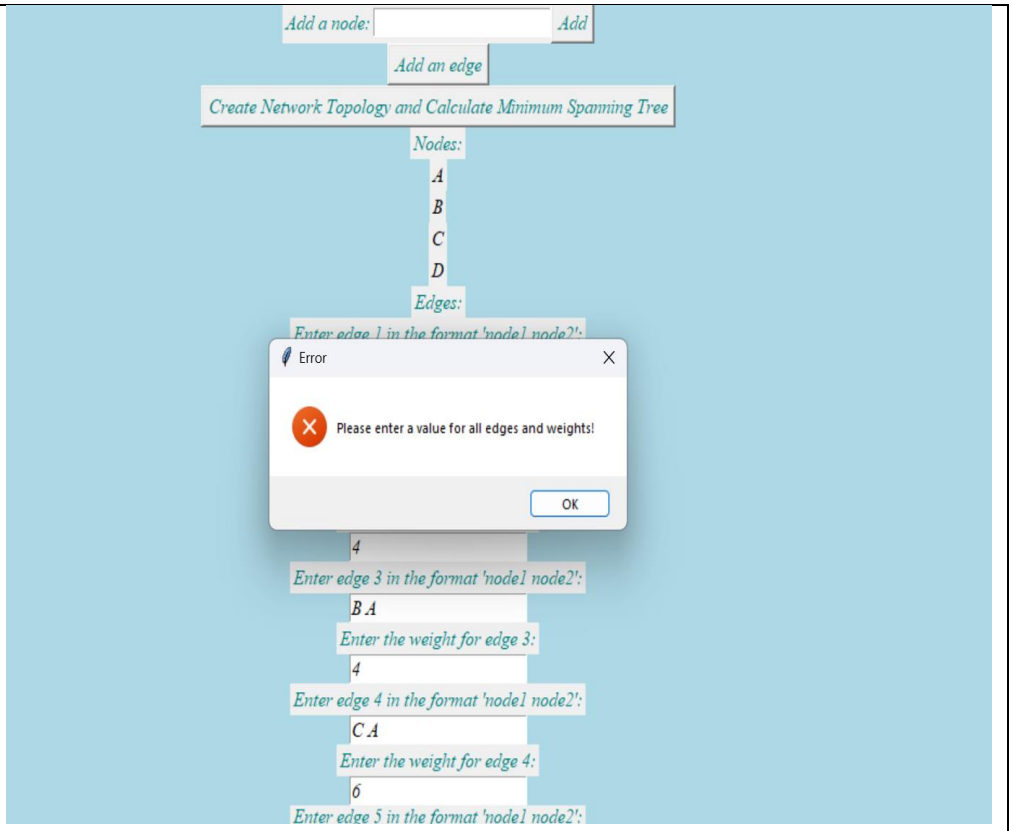


Table 7.8 Open Shortest Path First

Test Case Title	Minimum Spanning Tree (Greedy Kruskal)
Purpose of Testing	Spans the tree by drawing edges with minimum weight in increasing order without creating a circle.
Test Data	Click on the Create Network Topology and Minimum Spanning Tree Button
Steps	1.Input nodes and edges. 2.Checks if the nodes are valid, if valid then add path between the nodes with weight (edges). 3.Input source and destination nodes. 4.Shortest path is calculated based on the sum of the weights. 5.Displays the Open shortest Path.
Expected Output	Valid Output: 


Add a node:
Add
Add an edge
Create Network Topology and Calculate Minimum Spanning Tree
Error
Please enter a node!
OK


Add a node: 2
Add
Add an edge
Create Network Topology and Calculate Minimum Spanning Tree
Error
Enter a valid node name
OK



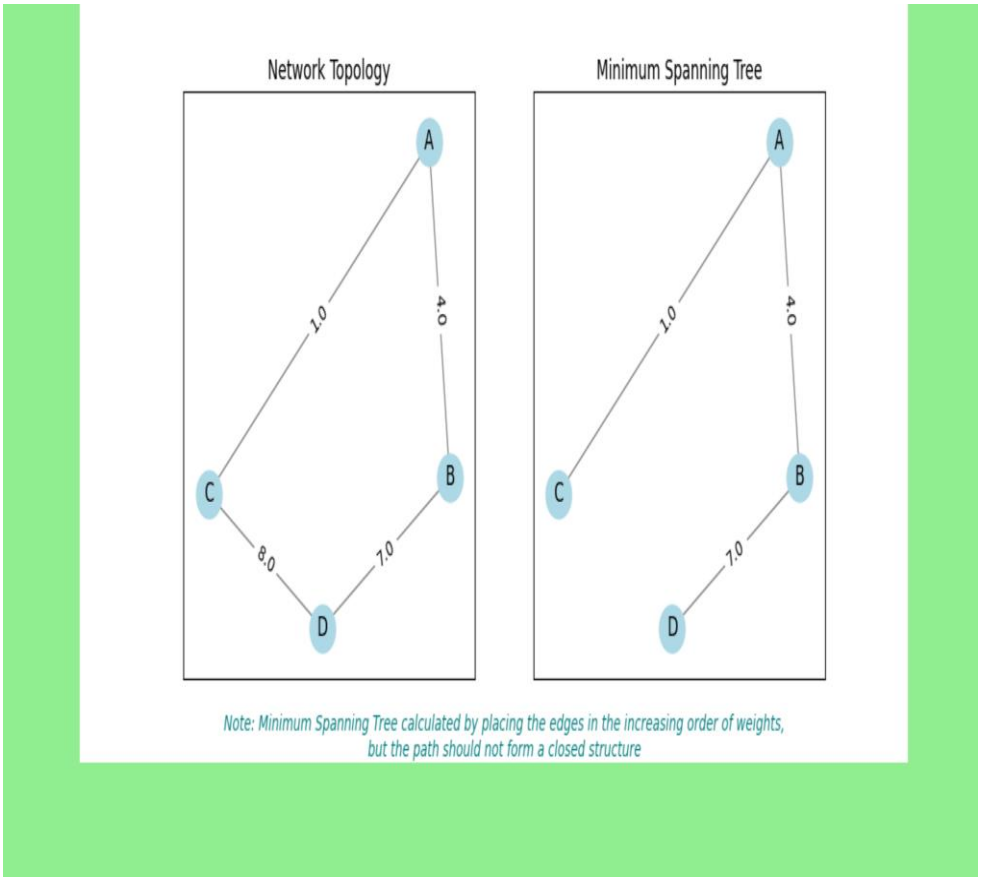
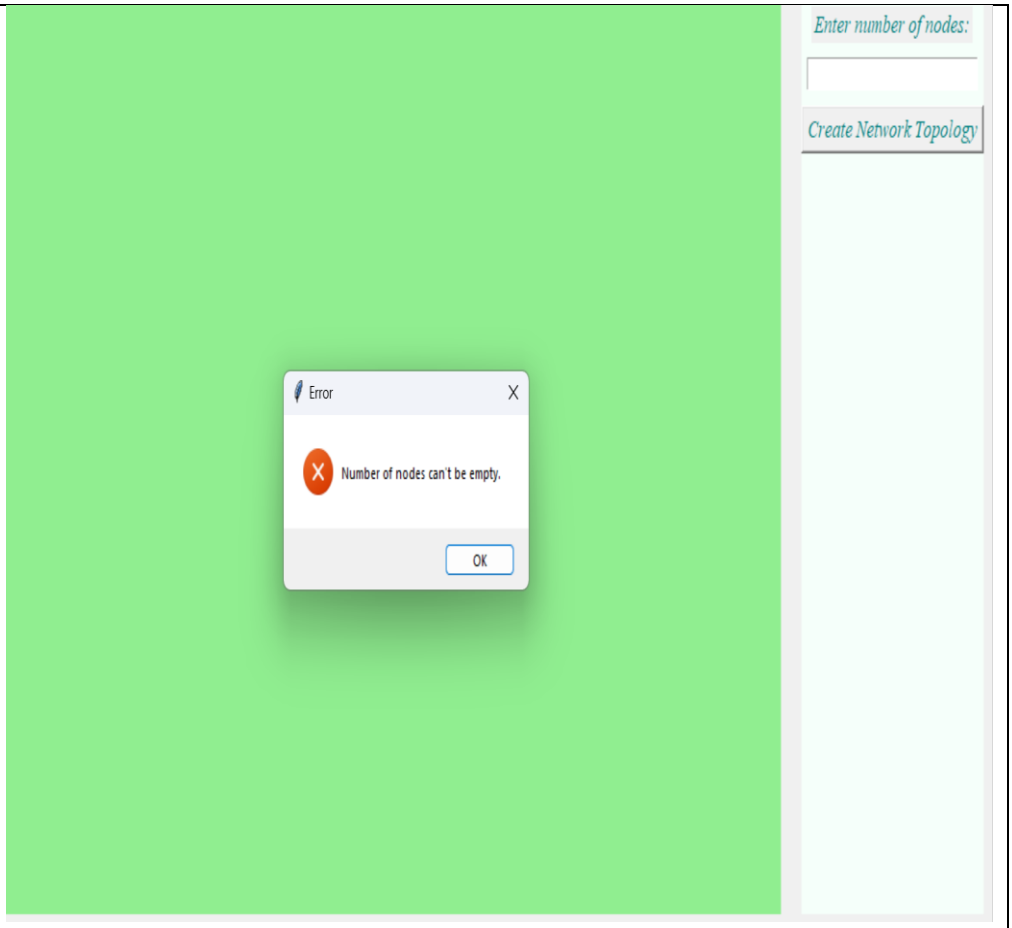
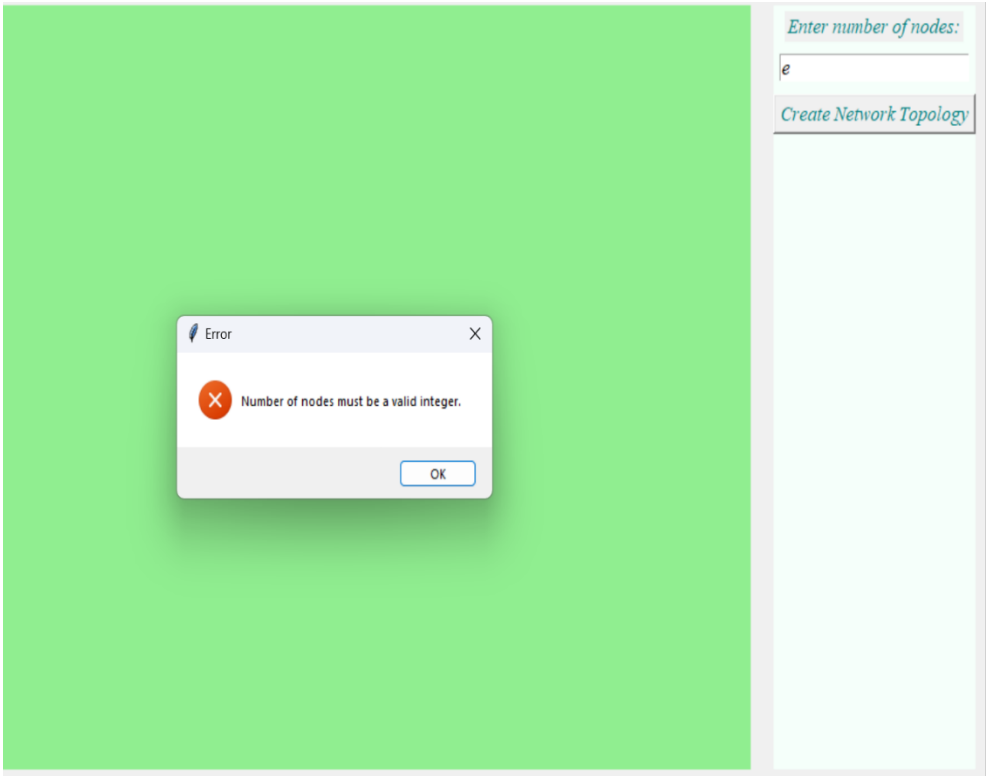
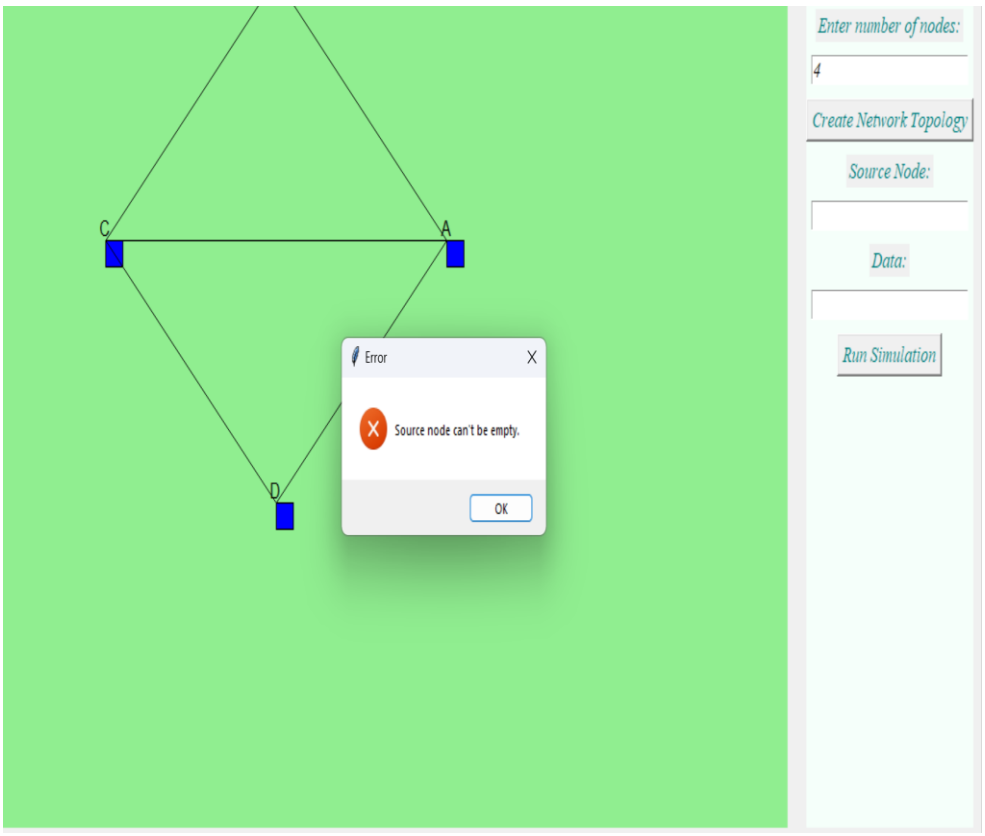


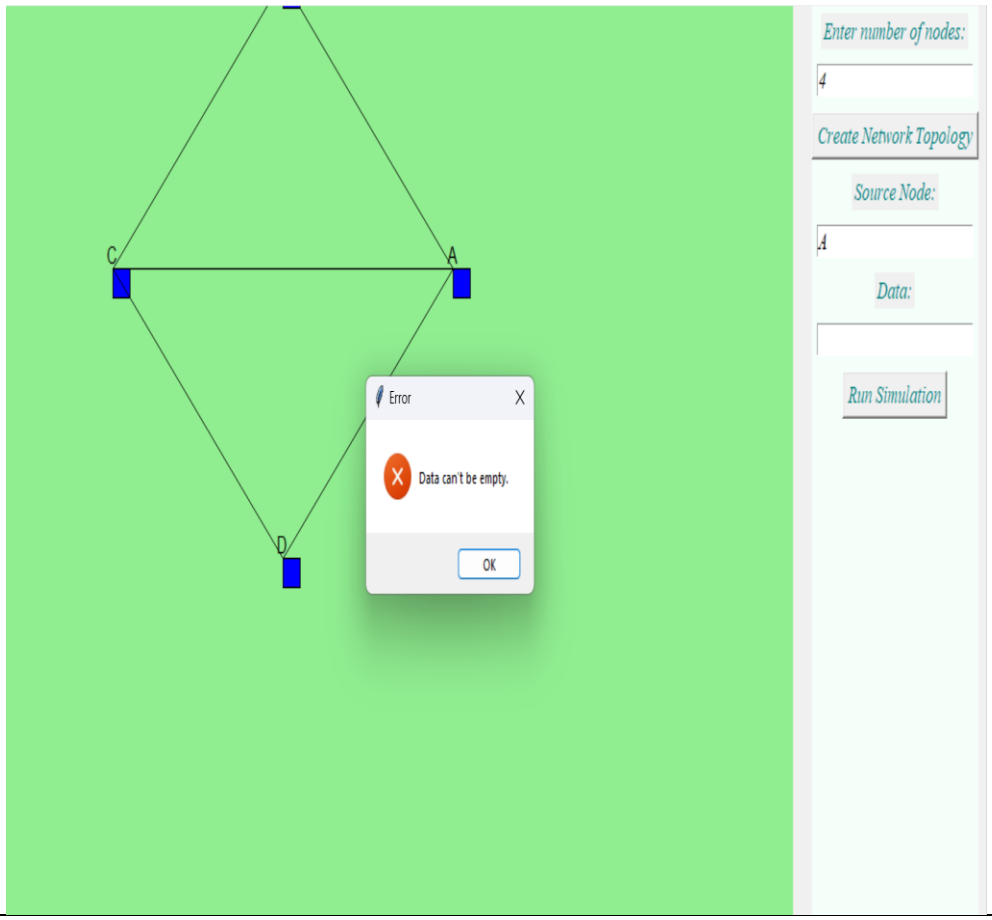
Table 7.9 Minimum Spanning Tree

Test Case Title	Flooding Routing Algorithm
Purpose of Testing	Purpose is to transmit the data from source node to all its neighbours.
Test Data	Click on the Button
Steps	1.Input nodes and edges. 2.Checks if the nodes are valid, if valid then add path between the nodes with weight (edges). 3.Input source and destination nodes. 4.Shortest path is calculated based on the sum of the weights. 5.Displays the Open shortest Path.
Expected Output	Valid Output: 


Enter number of nodes:
Create Network Topology
Error
Number of nodes can't be empty.
OK


Enter number of nodes:
e
Create Network Topology
Error
Number of nodes must be a valid integer.
OK


Enter number of nodes:
4
Create Network Topology
Source Node:
Data:
Run Simulation
C
A
D
Error
X
Source node can't be empty.
OK


Enter number of nodes:
4
Create Network Topology
Source Node:
A
Data:
Run Simulation
C
A
D
Error
X
Data can't be empty.
OK

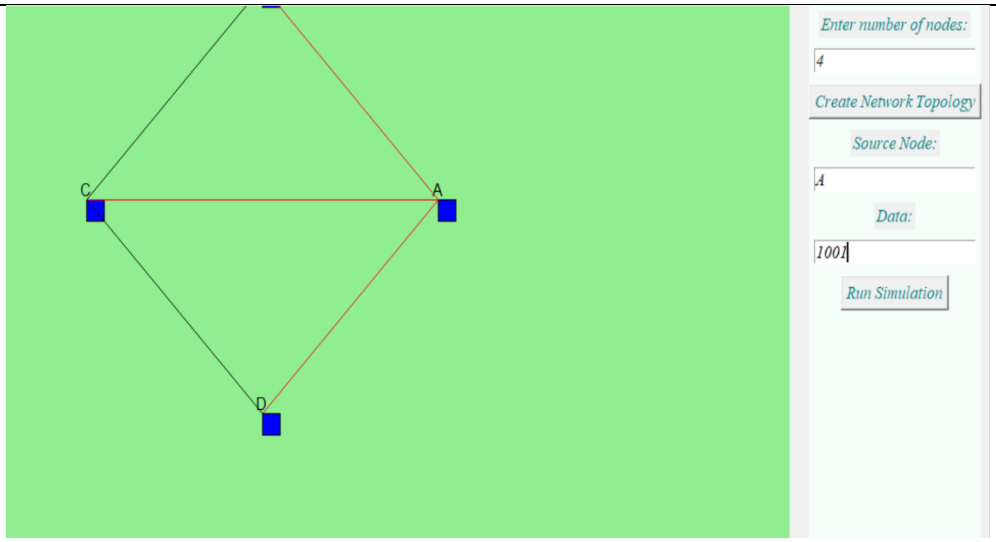
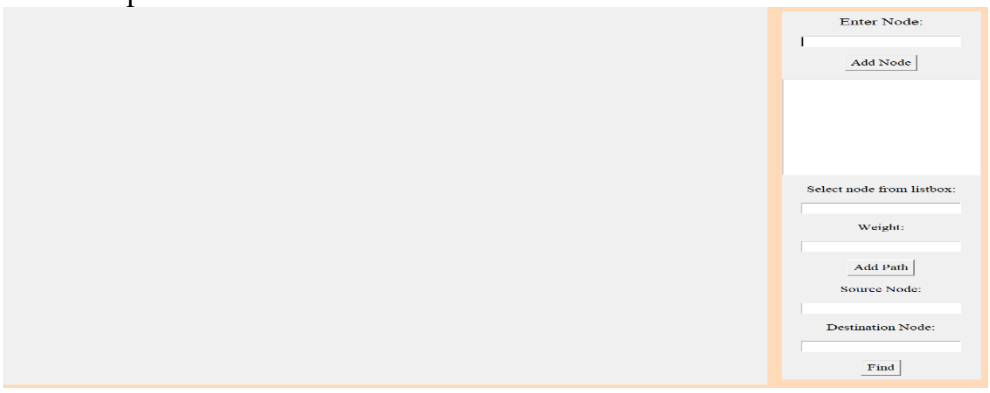
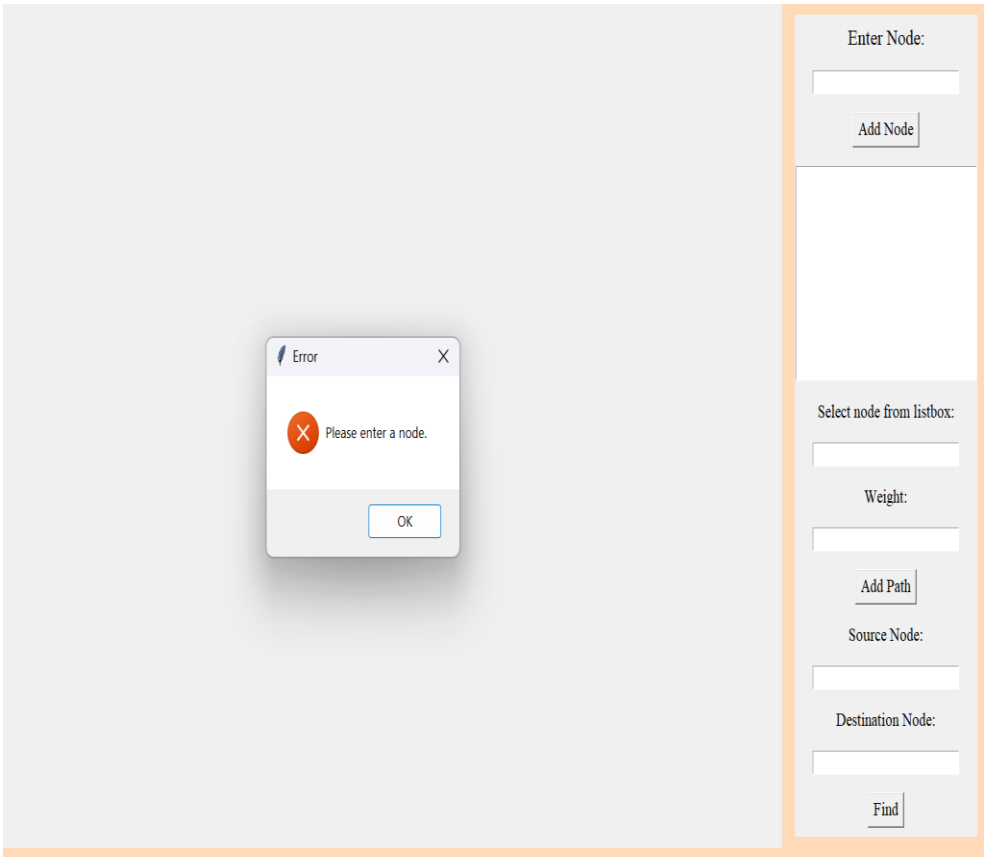
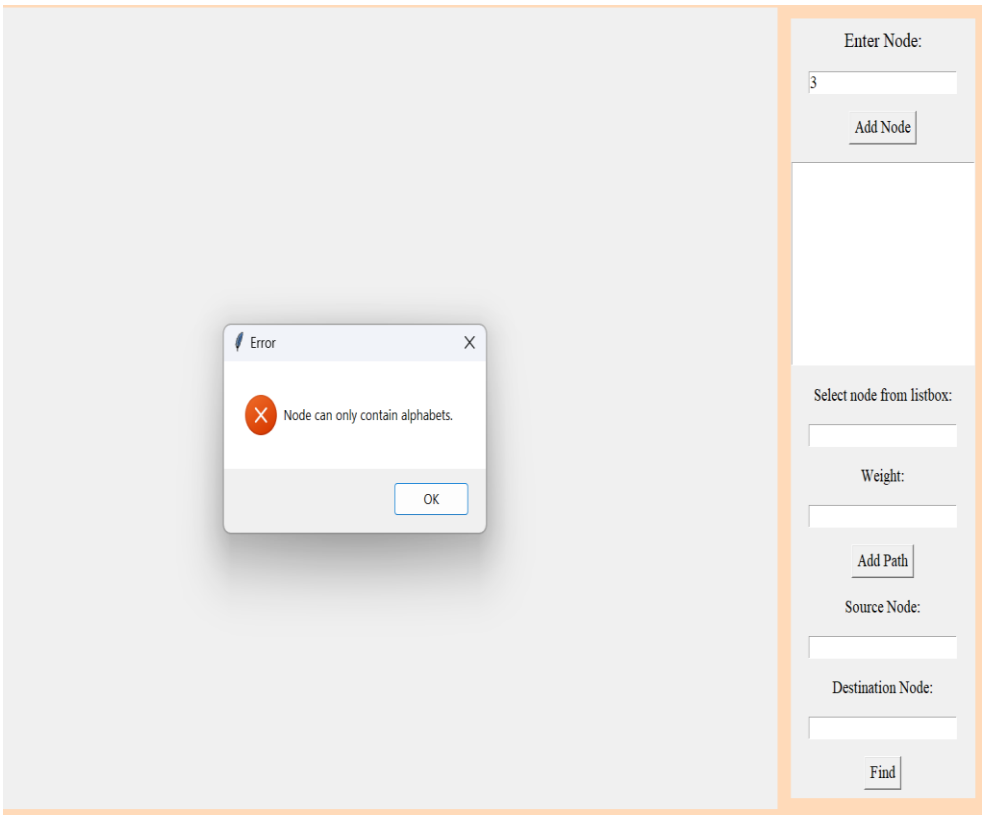
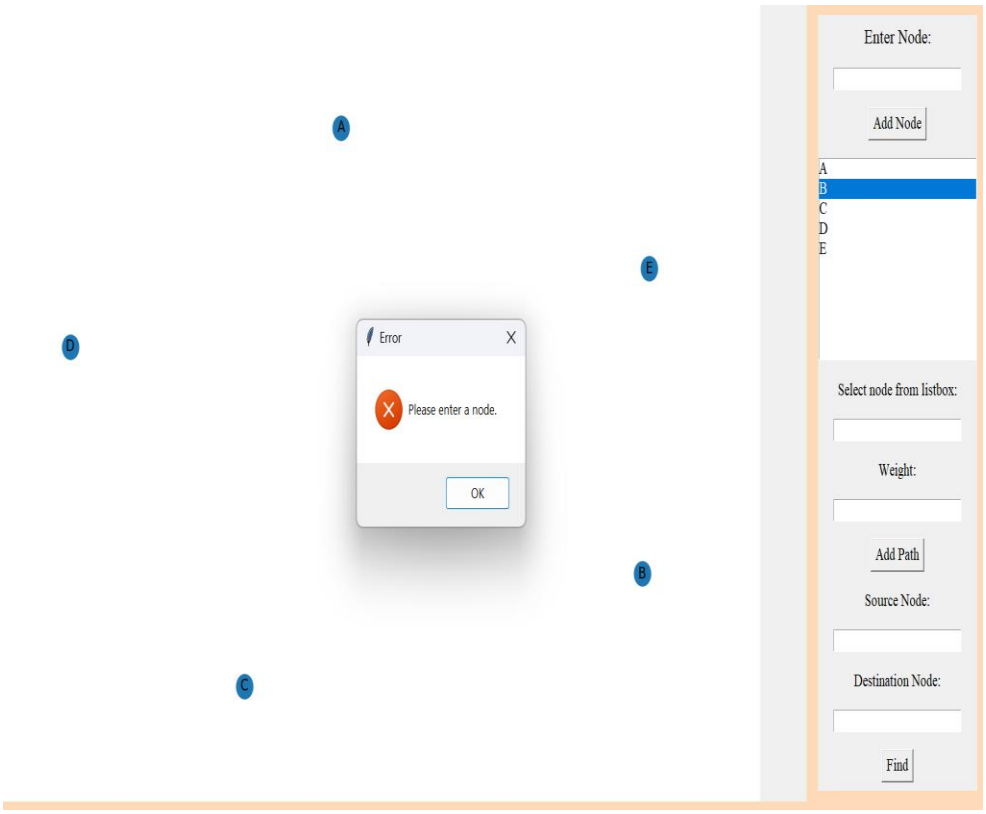
	

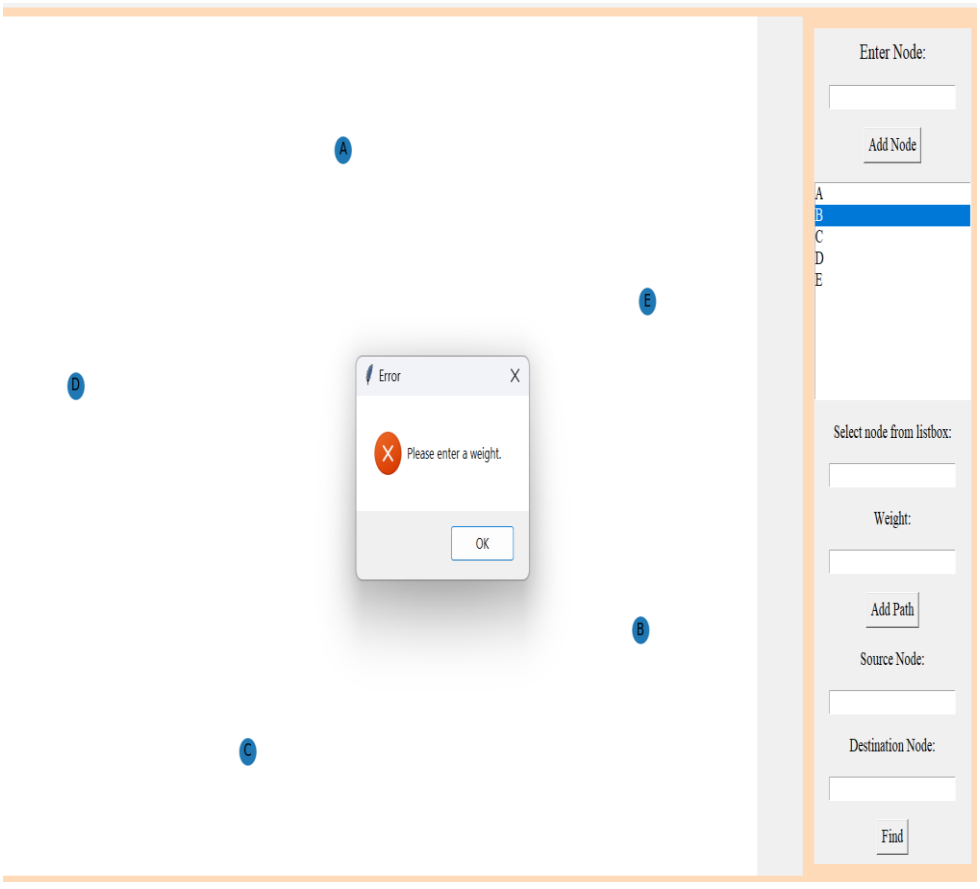
Table 7.10 Flooding Routing Algorithm

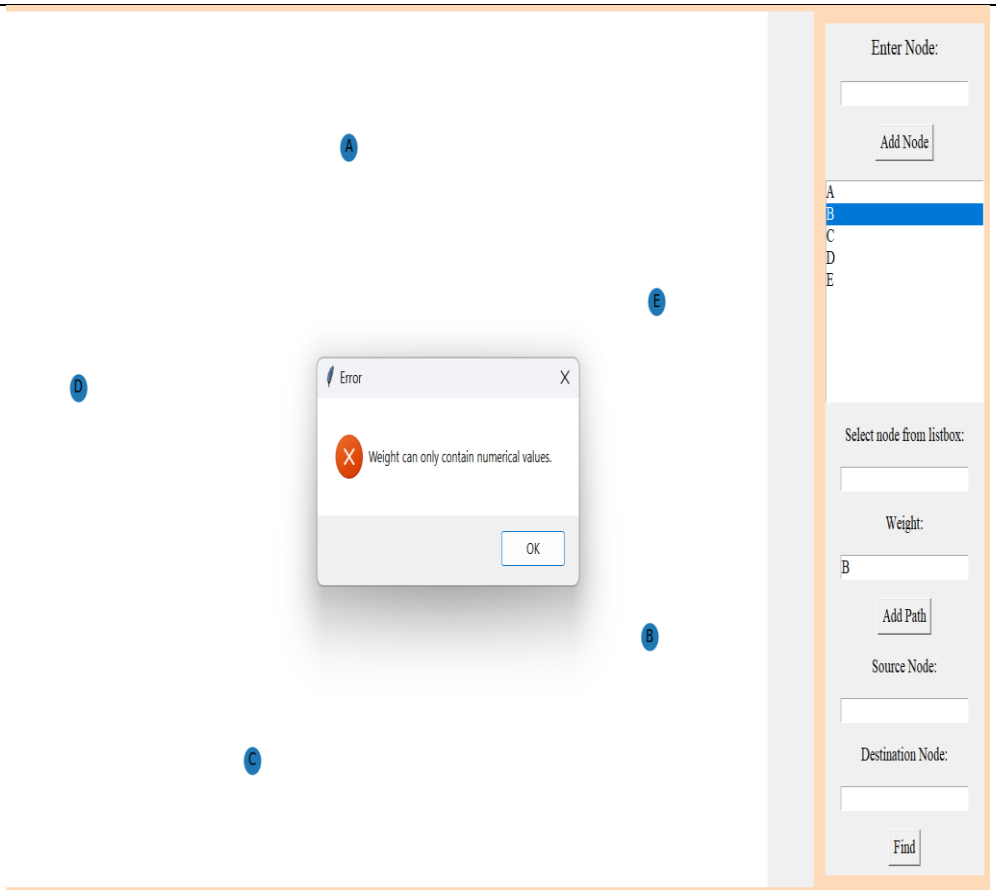
Test Case Title	Distance Vector Routing Algorithm
Purpose of Testing	Purpose is to find the best path which is the shortest distance between source node to destination node based on the sum of weights and gives the distance.
Test Data	Click on the Find Button
Steps	1.Input nodes and edges. 2.Checks if the nodes are valid, if valid then add path between the nodes with weight (edges). 3.Input source and destination nodes. 4. Process starts to find the best path which is the shortest distance between source node to destination node based on the sum of weights and gives the distance 5.Displays Shortest Distance between source and destination nodes.
Expected Output	Valid Output: 

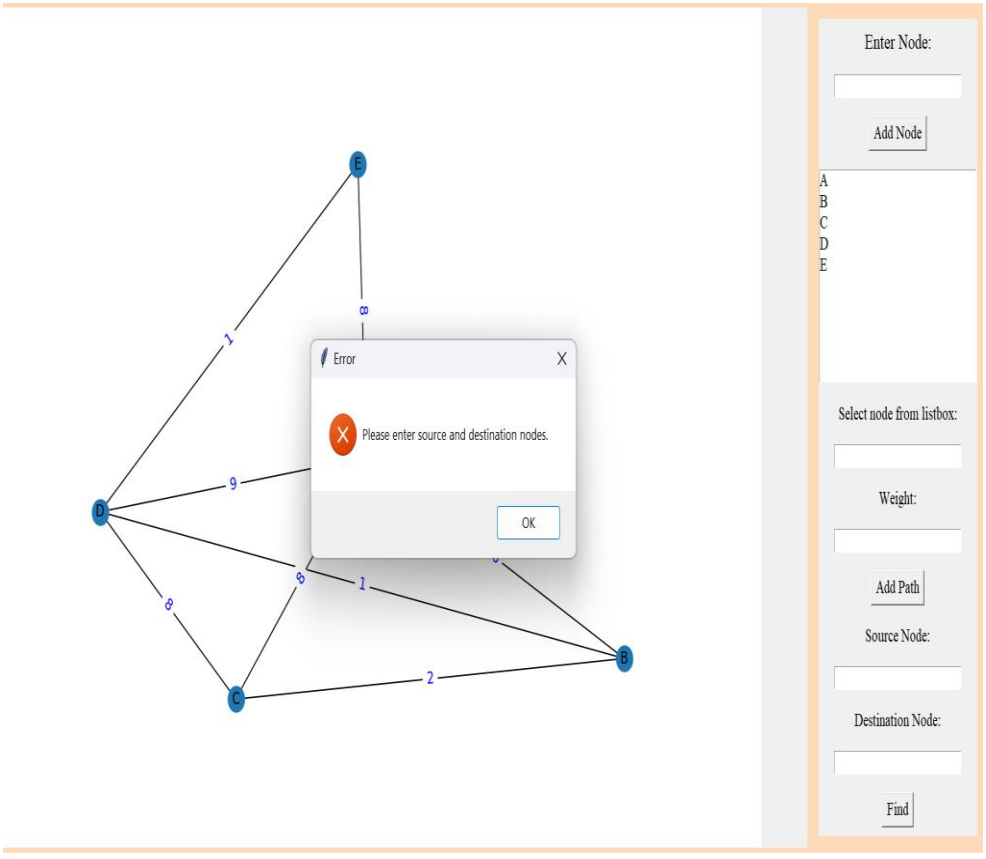

Enter Node:
Add Node
Error
Please enter a node.
OK
Select node from listbox:
Weight:
Add Path
Source Node:
Destination Node:
Find


Enter Node:
3
Add Node
Error
Node can only contain alphabets.
OK
Select node from listbox:
Weight:
Add Path
Source Node:
Destination Node:
Find


A
E
D
B
C
Error
Please enter a node.
OK
Enter Node:
Add Node
A
B
C
D
E
Select node from listbox:
Weight:
Add Path
Source Node:
Destination Node:
Find


A
E
D
B
C
Error
Please enter a weight.
OK
Enter Node:
Add Node
A
B
C
D
E
Select node from listbox:
Weight:
Add Path
Source Node:
Destination Node:
Find


A
E
D
B
C
Error
Weight can only contain numerical values.
OK
Enter Node:
Add Node
A
B
C
D
E
Select node from listbox:
Weight:
B
Add Path
Source Node:
Destination Node:
Find


E
8
1
Error
Please enter source and destination nodes.
OK
9
D
8
8
1
2
B
C
Enter Node:
Add Node
A
B
C
D
E
Select node from listbox:
Weight:
Add Path
Source Node:
Destination Node:
Find

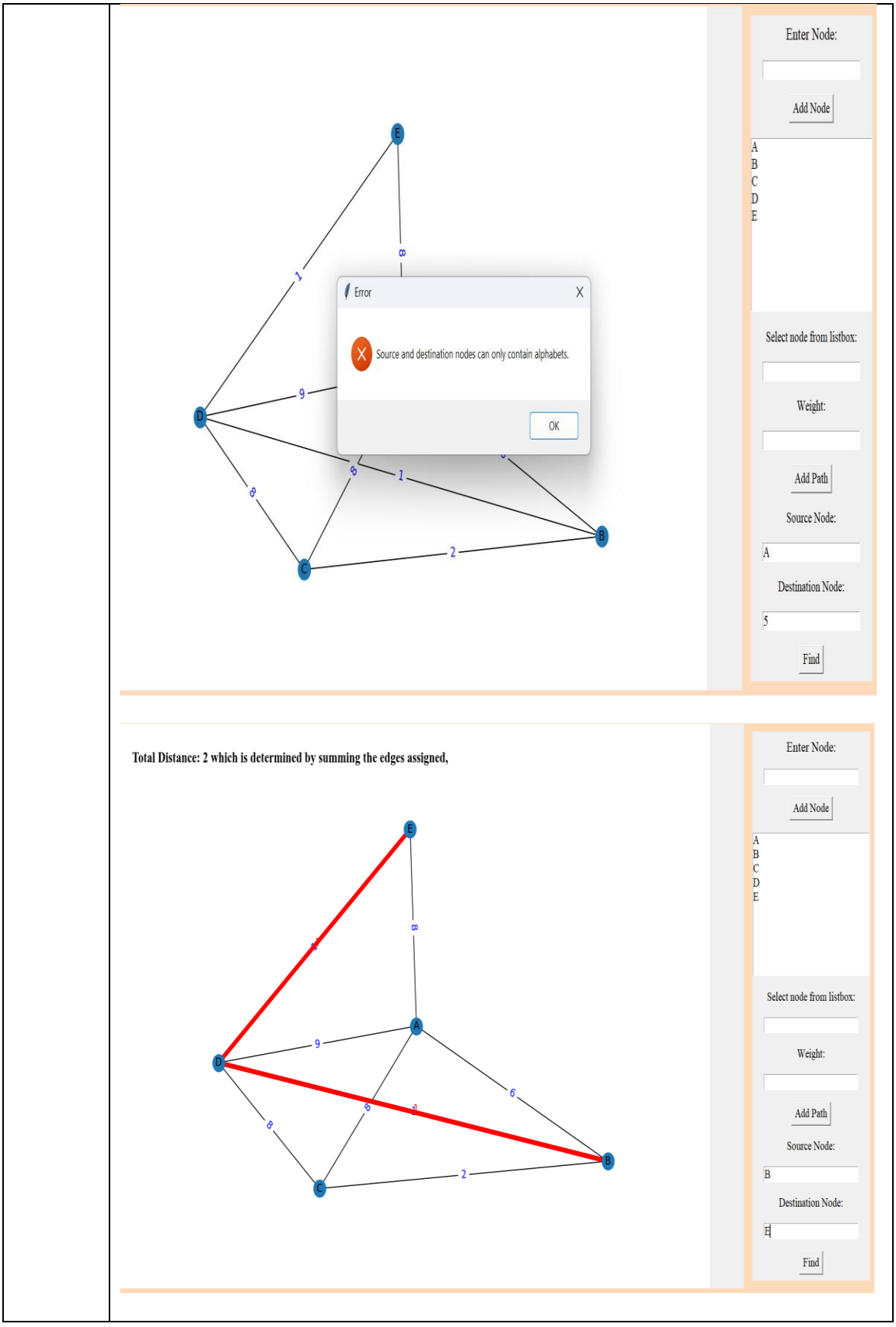


Table 7.11 Distance Vector Routing Algorithm

CONCLUSION

CONCLUSION

In conclusion, this project was successfully implemented using python 3.11.0. This project carried out various technique such as drawing using figure canvas, simulation or visualization in turtle graphics, adding voice over to the notes etc.

During the implementation we have faced many challenges in making of algorithm simulation using Turtle Graphics, and handled all the challenges successfully. And handle various exceptions during development of the project work.

We have learnt about drawing using figure canvas, simulation or visualization using turtle graphics and adding interface through Tkinter. In addition to that, we learnt about python coding.

Moreover, this project helped us to understand Software Development Life Cycle (SDLC), Time bound work, team spirit, and preparing project document, project testing, designing and presentation.

Finally concluded that we tried to fulfill the objectives of project work and goal of our project 'NETWORKING ALGORITHM SIMULATION'.

Nagashree 200598

Shridevi Jogi 200606

Suraksha R 200625

LIMITATIONS

LIMITATIONS

It explains the boundaries of project establishment, boundaries for each team members and set of procedure for how completed work is to be verified.

- This project is just a Stand-alone Application, so it’s not a Client-Server Application where the error gets detected.
- We used only few networking algorithms, there are still many algorithms where we can use it and develop the application.

FUTURE SCOPE

FUTURE SCOPE

- In Future, can Enhance this project by making it useful to the students and staff for studies purpose.
- We can enhance this project by adding more visual representation and graphical implementation of algorithm.

ABBREVIATIONS AND ACRONYMS

ABBREVATIONS AND ACRONYMS

GUI- Graphical User Interface

CPU- Central Processing Unit

RAM- Random Access Memory

OS- Operating Systems

DFD- Data Flow Diagram

CFD- Control Flow Diagram

I/O- Input and Output

BIBLIOGRAPHY

BIBLIOGRAPHY

Referred Books:

- An Integrated approached to software to software engineering- Pankaj Jalote
- The Computer reference PYTHON
- Networking routing Algorithms, protocols and architectures - D.Medhi and K.Ramaswamy

Referred Sites:

- www.stackoverflow.com
- www.github.com
- www.tutorialspoint.com
- www.geeksforgeeks