Project Report

On

"Metro Route Finder"

Submitted by

Tanu Priya

25MCI10123

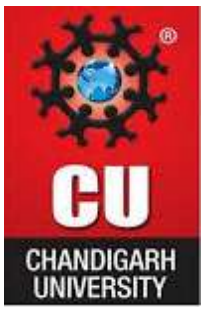
Under the guidance of

Prof. Sarthak Kushwaha

In partial fulfilment for the award of the degree of

MASTER OF COMPUTER APPLICATIONS

ARTIFICIAL INTELLGENCE & MACHINE LEARNING



Chandigarh University

November 2025

Certificate

This is to certify that **Tanu Priya**, a student of **Master of Computer Applications(MCA) – Artificial Intelligence & Machine Learning**, has successfully completed the **Mini Project** titled "**Metro Route Finder**" under the esteemed guidance of **Mr. Sarthak Kushwaha, University Institute of Computing (UIC), Chandigarh University.**

This project was undertaken as a part of the academic curriculum and is submitted in **partial fulfilment of the requirements** for the MCA program. The work presented in this project is a result of **independent research, diligent effort, and dedication,** demonstrating the student's ability to apply theoretical knowledge to practical problem-solving.

The project, **"Metro Route Finder – 20 Station Network"**, is a desktop-based application designed to find the **shortest route between metro stations** using **Dijkstra's Algorithm**. It visually represents a 20-station metro network through a **Tkinter-based GUI**, allowing users to select start and destination stations and visualize the optimal route with highlighted paths. The project showcases the integration of **Python programming**, **graph theory algorithms**, and **graphical user interface design**.

I hereby confirm that this project is an **original work** carried out by the student and has **not been submitted elsewhere** for the award of any other degree, diploma, or certification.

Project Guide:

Mr. Sarthak Kushwaha

University Institute of Computing

Chandigarh University

Acknowledgement

I would like to express my heartfelt gratitude to **Chandigarh University** and the **University Institute of Computing (UIC)** for providing me with the opportunity to undertake this project, **"Metro Route Finder – 20 Station Network."**

I extend my sincere appreciation to my respected mentor, **Mr. Sarthak Kushwaha**, for his invaluable guidance, continuous support, and insightful feedback throughout the duration of this project. His expertise, patience, and encouragement have been instrumental in shaping my understanding and successfully completing this work.

I am also thankful to my classmates and peers for their cooperation, constructive discussions, and motivation during the project's development. Their valuable suggestions and teamwork made this journey both enriching and enjoyable.

Finally, I express my deepest gratitude to my family for their unwavering support, understanding, and inspiration during the entire process of developing this project.

This project has been a remarkable learning experience, allowing me to enhance my knowledge of **Python programming, Dijkstra's algorithm, and GUI development using Tkinter**. I hope it serves as a stepping stone for further exploration into software development and intelligent route optimization systems.

Tanu Priya

MCA – Artificial Intelligence & Machine Learning

Chandigarh University

Index

Introduction

Efficient route planning and navigation are essential in today's rapidly expanding urban environments, where metro networks play a vital role in public transportation. The **"Metro Route Finder – 20 Station Network"** project aims to design a **desktop-based application** that calculates and visualizes the **shortest path between two metro stations**, providing commuters with a fast and optimized route suggestion.

This project applies the concept of **graph theory** and **Dijkstra's Algorithm** to compute the minimum distance between stations in a weighted metro network. Each station is represented as a **node**, and the connecting paths between them as **edges** with associated distances (weights). The algorithm ensures the selection of the most efficient route, minimizing travel time and distance between two user-selected stations.

The graphical user interface (GUI) is developed using **Python's Tkinter library**, offering an intuitive, user-friendly environment where users can choose the **source** and **destination** stations from dropdown menus and instantly view the **highlighted route** on a visual metro map. The interface also displays the **total distance** and the **sequence of stations** included in the optimal route.

The project demonstrates a successful combination of **algorithmic problem-solving**, **data visualization**, and **interactive interface design**. It emphasizes how **Python programming** can be leveraged to simulate real-world systems such as metro route navigation. Beyond educational value, the project also provides a foundational approach that can be extended for **real-time metro applications**, including **dynamic route updates**, **fare calculations**, and **GPS integration** in future enhancements.

Objectives

The primary objective of this project is to design and implement a **Python-based desktop application** that enables users to find the **shortest route between two metro stations** in a given network using **Dijkstra's Algorithm**. The project aims to demonstrate the practical application of **graph theory** and **pathfinding algorithms** through an interactive and visually appealing **Tkinter GUI**.

Specific Objectives:

1. **To implement Dijkstra's Algorithm for route optimization:** Develop an efficient algorithm to calculate the shortest path between two selected metro stations in a weighted network, minimizing distance and ensuring computational accuracy.
2. **To create an interactive graphical interface using Tkinter:** Design a user-friendly GUI that allows users to select starting and destination stations, display routes, and visualize paths through highlighted connections on a metro map.
3. **To represent metro data using graph structures:** Model the metro network as a graph, where each station represents a node and each connection between stations represents an edge with a corresponding distance weight.
4. **To enhance user understanding through visualization:** Display both the **calculated route** and **total distance** clearly, with color-coded highlights on the metro map to make path tracking intuitive and easy to interpret.
5. **To ensure reliability and user input validation:** Incorporate input checks and meaningful feedback messages to handle invalid selections, missing inputs, or same-station queries effectively.
6. **To demonstrate real-world problem-solving using Python:** Showcase how Python's algorithmic capabilities, combined with GUI development, can be utilized to solve practical navigation and optimization problems in transportation systems.

Tools & Libraries Used

This project utilizes a combination of software tools, Python libraries, and development environments to build and execute the **Metro Route Finder – 20 Station Network** application. Each tool plays a vital role in implementing the algorithm, designing the graphical interface, and visualizing the metro map effectively.

---

1. Python

Python serves as the **core programming language** for this project due to its simplicity, versatility, and strong library support. It is used for implementing **Dijkstra's shortest path algorithm**, handling user input/output, and managing graphical operations in the Tkinter-based interface.

---

2. Tkinter Library

Tkinter is the standard GUI library for Python used to create the **graphical user interface** of the Metro Route Finder. It enables the creation of windows, buttons, dropdown menus, text boxes, and the visual metro map on a canvas. Tkinter provides an intuitive and interactive way for users to visualize station connections and routes.

---

3. Heapq Module

The **heapq** module from Python's standard library is used to efficiently implement the **priority queue** required by **Dijkstra's Algorithm**. It ensures that nodes with the smallest tentative distance are processed first, optimizing the performance of the pathfinding algorithm.

---

4. Canvas Widget

The **Tkinter Canvas widget** is used to draw the **metro map**, including stations (nodes), connecting paths (edges), and route highlights. Each station is represented as a circle, and each path is a line with labeled distances. The Canvas also supports real-time updates for highlighting routes dynamically.

---

5. Messagebox and Combobox Widgets

The **messagebox** module is used to display informative pop-up alerts such as missing inputs, invalid selections, or confirmation messages.
The **ttk.Combobox** widget provides dropdown menus that allow users to select the **start** and **destination** stations conveniently.

---

6. IDE / Code Editors

Development was carried out using **Visual Studio Code** and **PyCharm**, which provided a robust environment for coding, debugging, and testing the project efficiently.

---

7. Operating System

The project is platform-independent and can run on any system that supports Python (Windows, macOS, or Linux). It was primarily developed and tested on **Windows 11**.

Implementation Steps

```
import tkinter as tk
from tkinter import ttk, messagebox

# ---------------- Graph (Metro Map – 20 Stations) ----------------
graph = {
    "Station A": {"Station B": 4, "Station D": 7},
    "Station B": {"Station A": 4, "Station C": 3, "Station E": 6},
    "Station C": {"Station B": 3, "Station F": 5, "Station G": 4},
    "Station D": {"Station A": 7, "Station E": 2, "Station H": 5},
    "Station E": {"Station B": 6, "Station D": 2, "Station F": 4, "Station I": 6},
    "Station F": {"Station C": 5, "Station E": 4, "Station J": 7},
    "Station G": {"Station C": 4, "Station K": 6},
    "Station H": {"Station D": 5, "Station I": 3, "Station L": 4},
    "Station I": {"Station E": 6, "Station H": 3, "Station J": 5, "Station M": 6},
    "Station J": {"Station F": 7, "Station I": 5, "Station N": 4},

    # Newly added stations
    "Station K": {"Station G": 6, "Station O": 5},
    "Station L": {"Station H": 4, "Station P": 6},
    "Station M": {"Station I": 6, "Station Q": 5},
    "Station N": {"Station J": 4, "Station R": 3},
    "Station O": {"Station K": 5, "Station P": 4},
    "Station P": {"Station L": 6, "Station O": 4, "Station Q": 5},
    "Station Q": {"Station M": 5, "Station P": 5, "Station R": 6},
    "Station R": {"Station N": 3, "Station Q": 6, "Station S": 4},
    "Station S": {"Station R": 4, "Station T": 5},
    "Station T": {"Station S": 5}
```

```
}

# ---------------- Dijkstra Algorithm ----------------
def dijkstra(start, end):
    import heapq
    pq = [(0, start, [])]
    visited = set()

    while pq:
        cost, node, path = heapq.heappop(pq)
        if node in visited:
            continue
        visited.add(node)

        path = path + [node]
        if node == end:
            return cost, path

        for neighbor, weight in graph[node].items():
            if neighbor not in visited:
                heapq.heappush(pq, (cost + weight, neighbor, path))
    return None

# ---------------- GUI ----------------
root = tk.Tk()
root.title("Metro Route Finder – 20 Station Network")
root.geometry("1300x800")
root.configure(bg="#000000")

# ---------------- Left Side (Map Canvas) ----------------
canvas = tk.Canvas(root, width=950, height=800, bg="#111111", highlightthickness=0)
canvas.pack(side="left", fill="both")
```

```
# --- Coordinates for 20 stations ---
pos = {
    "Station A": (100, 100),  "Station B": (230, 100),  "Station C": (360, 100),
    "Station D": (100, 230),  "Station E": (230, 230),  "Station F": (360, 230),
    "Station G": (500, 100),  "Station H": (100, 360),  "Station I": (230, 360),
    "Station J": (360, 360),  "Station K": (500, 230),  "Station L": (100, 500),
    "Station M": (230, 500),  "Station N": (360, 500),  "Station O": (500, 360),
    "Station P": (100, 630),  "Station Q": (230, 630),  "Station R": (360, 630),
    "Station S": (500, 500),  "Station T": (650, 500)
}

# Draw edges + distance labels
edge_ids = {}
for s in graph:
    for n, w in graph[s].items():
        if (n, s) in edge_ids:
            continue

        x1, y1 = pos[s]
        x2, y2 = pos[n]

        # Draw route line
        line = canvas.create_line(x1, y1, x2, y2, fill="#22ffff", width=3)
        edge_ids[(s, n)] = line

        # Midpoint for distance label
        mx = (x1 + x2) / 2
        my = (y1 + y2) / 2

        canvas.create_text(mx, my, text=str(w), fill="white", font=("Arial", 10, "bold"))
```

```
# Draw nodes
node_ids = {}
def draw_stations():
    for name, (x, y) in pos.items():
        oval = canvas.create_oval(x-18, y-18, x+18, y+18,
                                  fill="#ff00ff", outline="white", width=2)
        canvas.create_text(x, y+32, text=name, fill="white", font=("Arial", 10))
        node_ids[name] = oval

draw_stations()

# ---------------- Right Panel ----------------
panel = tk.Frame(root, bg="#000000")
panel.pack(side="right", fill="y", padx=15, pady=15)

label = tk.Label(panel, text="Metro Route Finder", fg="white", bg="black",
                 font=("Consolas", 16))
label.pack(pady=10)

stations = list(graph.keys())
start_var = tk.StringVar()
end_var = tk.StringVar()

tk.Label(panel, text="From:", fg="white", bg="black", font=("Arial", 12)).pack(anchor="w")
start_menu = ttk.Combobox(panel, textvariable=start_var, values=stations, state="readonly")
start_menu.pack(fill="x", pady=5)

tk.Label(panel, text="To:", fg="white", bg="black", font=("Arial", 12)).pack(anchor="w")
end_menu = ttk.Combobox(panel, textvariable=end_var, values=stations, state="readonly")
end_menu.pack(fill="x", pady=5)

result_box = tk.Text(panel, height=20, width=32, bg="#111111", fg="white",
```

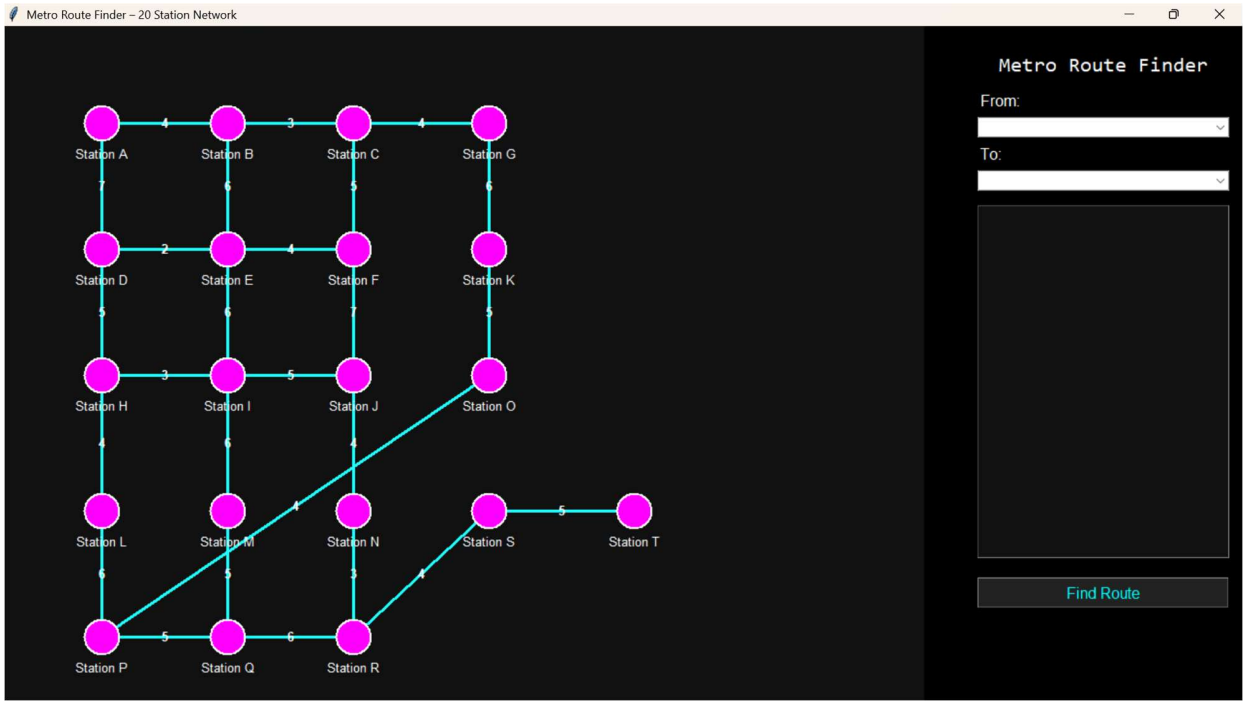
```
                font=("Consolas", 11))
result_box.pack(pady=10)

# ---------------- Highlighter ----------------
def highlight_path(path):
    # Reset everything
    for e in edge_ids:
        canvas.itemconfig(edge_ids[e], fill="#22ffff", width=3)
    for n in node_ids:
        canvas.itemconfig(node_ids[n], fill="#ff00ff")

    # Highlight nodes
    for station in path:
        canvas.itemconfig(node_ids[station], fill="red")

    # Highlight edges
    for i in range(len(path)-1):
        a, b = path[i], path[i+1]

        if (a, b) in edge_ids:
            canvas.itemconfig(edge_ids[(a, b)], fill="yellow", width=5)
        elif (b, a) in edge_ids:
            canvas.itemconfig(edge_ids[(b, a)], fill="yellow", width=5)

# ---------------- Find Route Button ----------------
def find_route():
    start = start_var.get()
    end = end_var.get()

    if not start or not end:
        messagebox.showwarning("Missing Input", "Choose both start and end stations.")
        return
```

```
    if start == end:
        messagebox.showinfo("Same Station", "Both stations cannot be same.")
        return

    result_box.delete("1.0", tk.END)

    res = dijkstra(start, end)
    if res is None:
        result_box.insert(tk.END, "No route found.")
    else:
        dist, path = res
        result_box.insert(tk.END, f"Shortest Distance: {dist}\n\nRoute:\n")
        for p in path:
            result_box.insert(tk.END, f"→ {p}\n")
        highlight_path(path)

tk.Button(panel, text="Find Route", command=find_route, bg="#222222",
          fg="cyan", font=("Arial", 12)).pack(fill="x", pady=10)

root.mainloop()
```

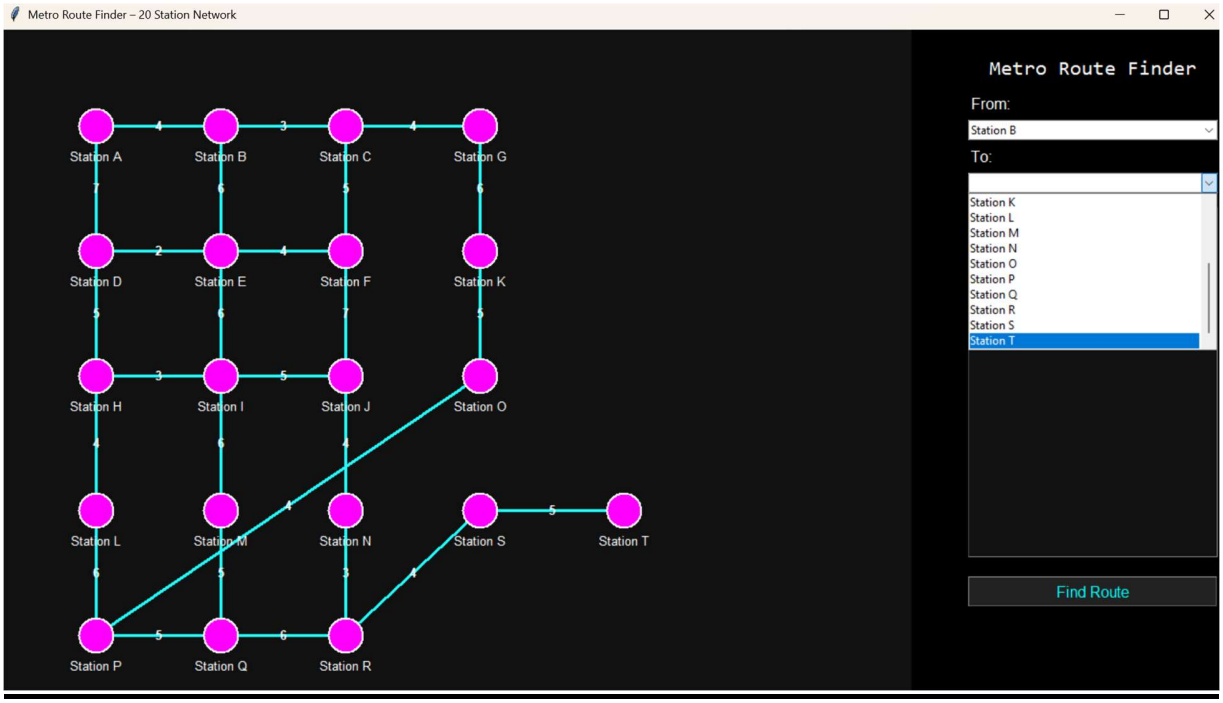
Screenshots/Results

Output:-

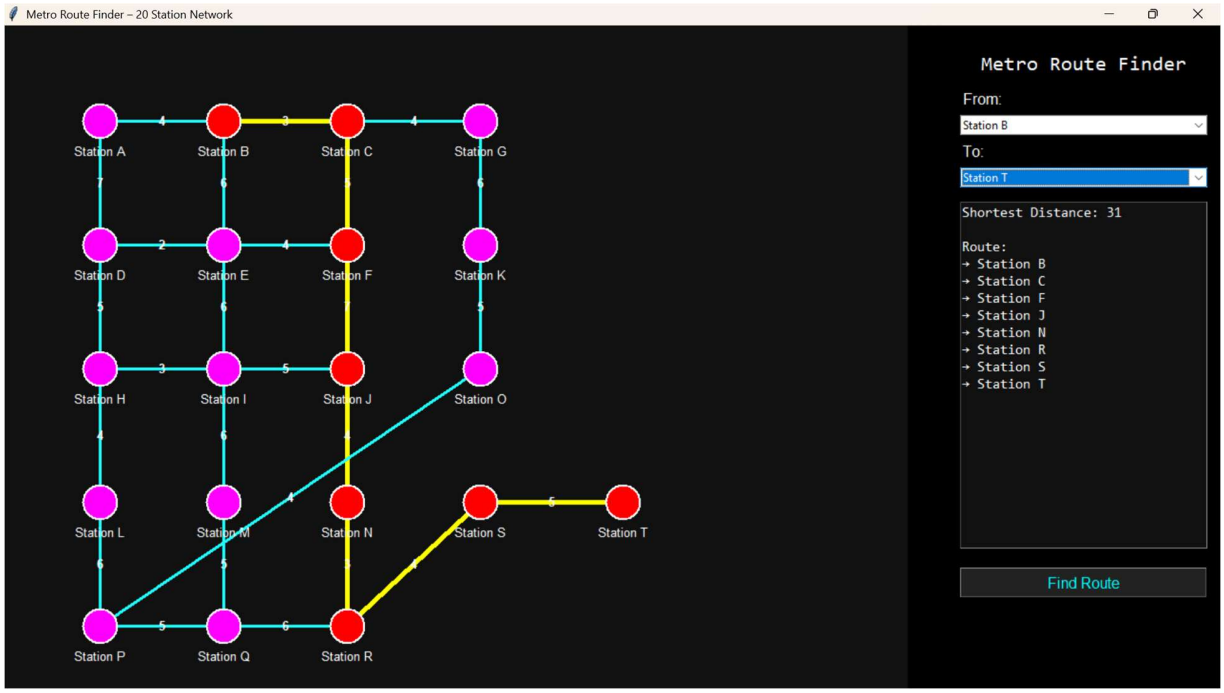
Main Window – Metro Map View:



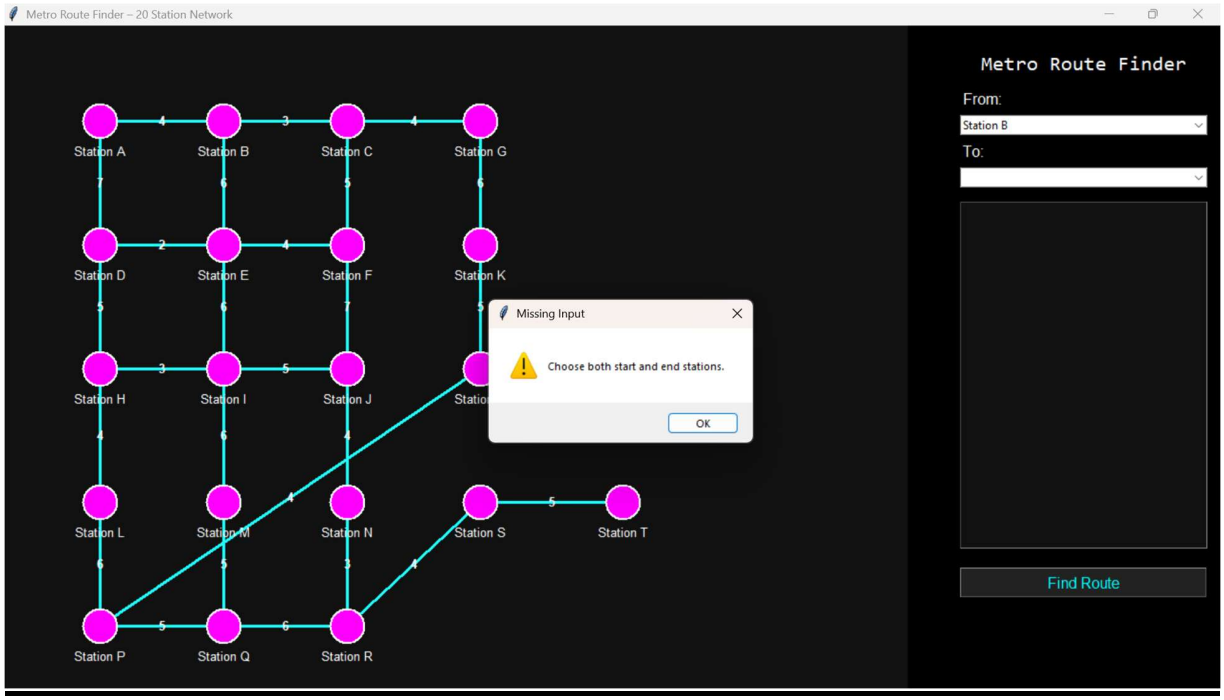
Selecting Start and Destination Stations:



Displaying the Shortest Route and Distance:



Invalid Input Handling:



Conclusion

The **Metro Route Finder – 20 Station Network** project successfully demonstrates how **Python programming**, **graph theory**, and **graphical user interface design** can be integrated to solve real-world problems such as **shortest path navigation** in metro networks.

By applying **Dijkstra's Algorithm**, the system effectively calculates the **minimum distance and optimal route** between any two metro stations. The visual representation using **Tkinter Canvas** enhances user interaction and understanding, allowing users to not only see the shortest route but also observe how stations are connected throughout the network.

The project meets its primary objectives by:

- Accurately finding the shortest route between two stations.
- Providing an interactive and user-friendly graphical interface.
- Visually highlighting the computed path with clear color distinctions.
- Implementing robust error handling and input validation mechanisms.

Beyond being an academic exercise, this project showcases the practical application of **algorithmic problem-solving** in transportation systems. It serves as a foundation for further enhancements such as:

- Incorporating **real-time metro data** for live route updates.
- Adding **fare calculation and time estimation** features.
- Integrating **GPS-based route tracking** and **multi-line metro systems**.

Overall, this project successfully bridges the gap between theoretical concepts and real-world implementation, illustrating how **Python and data structures** can be used to design intelligent, efficient, and interactive software solutions for everyday challenges in public transportation.

Reference

- **Python Official Documentation — Tkinter Library**
  https://docs.python.org/3/library/tkinter.html
- **Python Official Documentation — heapq Module**
  https://docs.python.org/3/library/heapq.html
- **GeeksforGeeks — Dijkstra's Shortest Path Algorithm in Python**
  https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-in-python/
- **W3Schools — Python Tkinter GUI Programming**
  https://www.w3schools.com/python/python_gui_tkinter.asp
- **Real Python — Introduction to Tkinter**
  https://realpython.com/python-gui-tkinter/
- **Python Official Documentation — messagebox and ttk Widgets**
  https://docs.python.org/3/library/tkinter.messagebox.html
  https://docs.python.org/3/library/tkinter.ttk.html
- **Stack Overflow — Tkinter Canvas Examples and Discussions**
  https://stackoverflow.com/questions/tagged/tkinter

GithubLink:-