Project Report :

Romania Map GUI with Uniform Cost Search

Aim-

The aim of the project is to create GUI, where Romania map will be taken as an input. Apply Uniform cost search algorithm on Romania map and show the solution in each step. Give a provision to select any source and destination by the user. On clicking “SUBMIT” button, algorithm execution will be visualized on the screen. You can use different colors to represent each step. Final state will be represented by “BLUE” color.

Group Members-

04 Etisha Shastri A1

16 Vedanti Dalvi A1

09 Pragati Pawar A1

18 Yukta Agrawal A1

8 Nirmiti Umekar A1

1. **Introduction:**
   The project aims to create a Graphical User Interface (GUI) where the user can select a source and destination on the Romania map. It applies the Uniform Cost Search algorithm to find the optimal path between the selected source and destination. The solution is visualized step by step on the GUI, with different colors representing each step. The final optimal path is highlighted in blue.

2. **Code Description:**

tkinter: Tkinter is used for creating the GUI components.
networkx: Networkx library is used for representing the Romania map as a graph and for graph traversal operations.
Math: Math library is used for mathematical calculations.

The code can be divided into several parts:

Initialization: The GUI is initialized with the necessary components such as canvas, buttons, and labels. The Romania map is represented as a graph, with nodes representing cities and edges representing connections between cities with corresponding weights (distances).

City Selection: Users can select a source and destination city by clicking on the respective nodes on the map. Once selected, the source city turns blue and the destination city turns red.

Uniform Cost Search: Upon clicking the "Submit" button, the Uniform Cost Search algorithm is applied to find the optimal path between the selected source and destination cities. The algorithm explores paths based on their costs and updates the queue accordingly.

Visualization: The search process is visualized step by step on the GUI. Each step highlights the current city being explored and the path being traversed. Different colors are used to represent different stages of the search process.

Final Path Highlighting: Once the optimal path is found, it is highlighted in red on the map. Additionally, the optimal path and its cost are displayed as a result.

Code-

```
import tkinter as tk
from tkinter import messagebox
```

```
import networkx as nx
import math


romania_map = nx.Graph()
romania_map.add_nodes_from([
    ("Arad"), ("Zerind"), ("Oradea"), ("Sibiu"),
    ("Timisoara"), ("Lugoj"), ("Mehadia"), ("Drobeta"),
    ("Craiova"), ("Rimnicu Vilcea"), ("Fagaras"),
    ("Pitesti"), ("Bucharest"), ("Giurgiu"), ("Urziceni"),
    ("Hirsova"), ("Eforie"), ("Vaslui"), ("Iasi"), ("Neamt")
])
romania_map.add_weighted_edges_from([
    ("Arad", "Zerind", 75), ("Arad", "Timisoara", 118),("Arad", "Sibiu", 140),
    ("Zerind", "Oradea", 71), ("Oradea", "Sibiu", 151),
    ("Timisoara", "Lugoj", 111), ("Lugoj", "Mehadia", 70),
    ("Mehadia", "Drobeta", 75), ("Drobeta", "Craiova", 120),
    ("Craiova", "Rimnicu Vilcea", 146), ("Craiova", "Pitesti", 138),
    ("Rimnicu Vilcea", "Sibiu", 80), ("Rimnicu Vilcea", "Pitesti", 97),
    ("Sibiu", "Fagaras", 99), ("Fagaras", "Bucharest", 211),
    ("Pitesti", "Bucharest", 101),
    ("Bucharest", "Giurgiu", 90), ("Bucharest", "Urziceni", 85),
    ("Urziceni", "Hirsova", 98), ("Hirsova", "Eforie", 86),
    ("Urziceni", "Vaslui", 142), ("Vaslui", "Iasi", 92),
    ("Iasi", "Neamt", 87)
])



class RomaniaMapGUI:
    def __init__(self, master):
        self.master = master
        self.master.title("Romania Map GUI")


        self.canvas = tk.Canvas(self.master, width=800, height=600)
        self.canvas.pack()


        self.node_positions = {
            "Arad": (50, 150), "Zerind": (80, 80), "Oradea": (120, 20),
            "Sibiu": (150, 250), "Timisoara": (50, 310), "Lugoj": (150, 340),
            "Mehadia": (210, 400), "Drobeta": (220, 480), "Craiova": (330,
480),
            "Rimnicu Vilcea": (320, 270), "Fagaras": (300, 150), "Pitesti":
(400, 230),
            "Bucharest": (500, 150), "Giurgiu": (550, 250), "Urziceni": (600,
150),
            "Hirsova": (650, 250), "Eforie": (700, 250), "Vaslui": (650, 50),
```

```
            "Iasi": (700, 50), "Neamt": (750, 50)
        }
        self.city_objects = {}
        for city, (x, y) in self.node_positions.items():
            city_object = self.canvas.create_oval(x - 10, y - 10, x + 10, y +
10, fill="green", width=2, outline="black")
            self.city_objects[city] = city_object
            self.canvas.create_text(x, y + 20, text=city, fill="black",
font=("Arial", 8, "bold"), anchor="center")
            self.canvas.tag_bind(city_object, "<Button-1>", lambda event,
city=city: self.select_city(city))


        for edge in romania_map.edges:
            start_node, end_node = edge
            start_x, start_y = self.node_positions[start_node]
            end_x, end_y = self.node_positions[end_node]
            start_radius = 10
            end_radius = 10
            angle = math.atan2(end_y - start_y, end_x - start_x)
            start_x += start_radius * math.cos(angle)
            start_y += start_radius * math.sin(angle)
            end_x -= end_radius * math.cos(angle)
            end_y -= end_radius * math.sin(angle)
            weight = romania_map[start_node][end_node]['weight']
            self.canvas.create_line(start_x, start_y, end_x, end_y,
fill="gray", width=2)
            x_text = (start_x + end_x) / 2
            y_text = (start_y + end_y) / 2
            self.canvas.create_text(x_text, y_text, text=str(weight),
fill="black", font=("Arial", 8, "bold"))


        self.paths_to_goal = []

    def select_city(self, city):
        if not hasattr(self, 'source_selected'):
            self.source_selected = city
            self.canvas.itemconfig(self.city_objects[city], fill="blue")
        else:
            if not hasattr(self, 'destination_selected'):
                self.destination_selected = city
                self.canvas.itemconfig(self.city_objects[city], fill="red")
                self.submit_button = tk.Button(self.master, text="Submit",
command=self.submit, bg="#0066cc", fg="white", padx=10, relief=tk.RAISED,
borderwidth=2)
                self.submit_button.pack(pady=10)
```

```
    def submit(self):
        self.path_generator = uniform_cost_search(romania_map,
self.source_selected, self.destination_selected)
        self.visualize_next_step()

    def visualize_next_step(self):
        try:
            node, path, cost = next(self.path_generator)
            if node == self.destination_selected:
                self.paths_to_goal.append((node, path, cost))
            self.visualize_search(node, path, cost)
            self.master.after(2000, self.visualize_next_step)
        except StopIteration:
            self.highlight_final_path()

    def visualize_search(self, node, path, path_cost):
        self.canvas.itemconfig(self.city_objects[node], fill="yellow")

        if len(path) > 1:
            for idx in range(len(path) - 1):
                start_city = path[idx]
                end_city = path[idx + 1]
                self.canvas.itemconfig(self.city_objects[start_city],
fill="blue")
                self.canvas.itemconfig(self.city_objects[end_city],
fill="blue")
                self.canvas.create_line(self.node_positions[start_city][0],
self.node_positions[start_city][1],
                                        self.node_positions[end_city][0],
self.node_positions[end_city][1],
                                        fill="blue", width=2)
                weight = romania_map[start_city][end_city]['weight']
                x_text = (self.node_positions[start_city][0] +
self.node_positions[end_city][0]) / 2
                y_text = (self.node_positions[start_city][1] +
self.node_positions[end_city][1]) / 2
                self.canvas.create_text(x_text, y_text, text=str(weight),
fill="black", font=("Arial", 8, "bold"))

    def highlight_final_path(self):
        if self.paths_to_goal:
            final_path = min(self.paths_to_goal, key=lambda x: x[2])
            _, path, cost = final_path
            for idx in range(len(path) - 1):
                start_city = path[idx]
                end_city = path[idx + 1]
                self.canvas.itemconfig(self.city_objects[start_city],
fill="red")
```

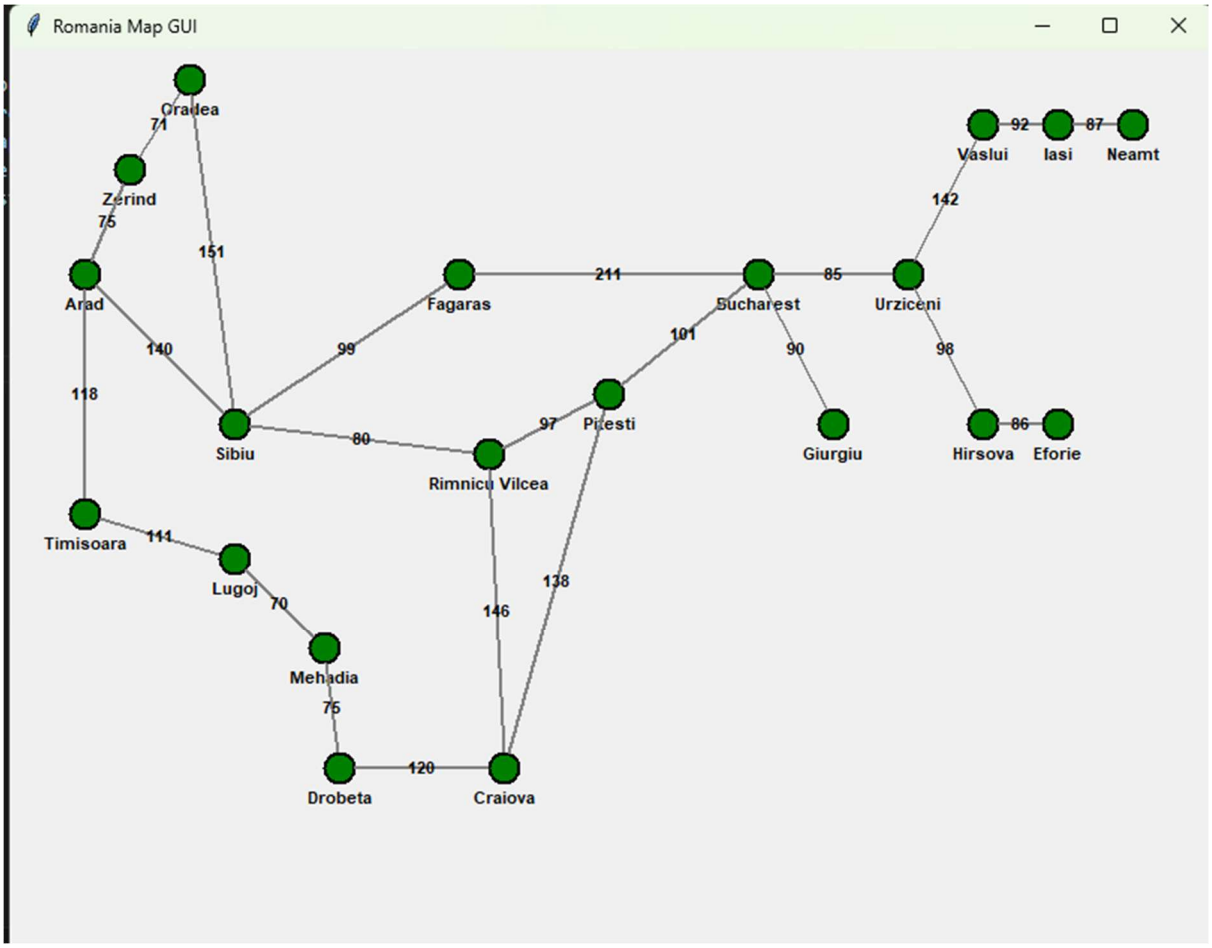
```
                self.canvas.itemconfig(self.city_objects[end_city],
fill="red")
                self.canvas.create_line(self.node_positions[start_city][0],
self.node_positions[start_city][1],
                                        self.node_positions[end_city][0],
self.node_positions[end_city][1],
                                        fill="red", width=2)
                weight = romania_map[start_city][end_city]['weight']
                x_text = (self.node_positions[start_city][0] +
self.node_positions[end_city][0]) / 2
                y_text = (self.node_positions[start_city][1] +
self.node_positions[end_city][1]) / 2
                self.canvas.create_text(x_text, y_text, text=str(weight),
fill="black", font=("Arial", 8, "bold"))

            result_str = f"Optimal Path: {' -> '.join(path)}\nOptimal Path
Cost: {cost}"
            self.result_label = tk.Label(self.master, text=result_str,
font=("Arial", 10))
            self.result_label.pack(pady=10)


def uniform_cost_search(graph, start, goal):
    explored = set()
    queue = [(0, [start])]

    while queue:
        cost, path = queue.pop(0)
        node = path[-1]
        if node == goal:
            yield node, path, cost
            break
        if node not in explored:
            explored.add(node)
            for neighbor in graph.neighbors(node):
                if neighbor not in explored:
                    new_cost = cost + graph[node][neighbor]['weight']
                    new_path = path + [neighbor]
                    queue.append((new_cost, new_path))
                    queue.sort()
        yield node, path, cost

if __name__ == "__main__":
    root = tk.Tk()
    app = RomaniaMapGUI(root)
    root.mainloop()
```

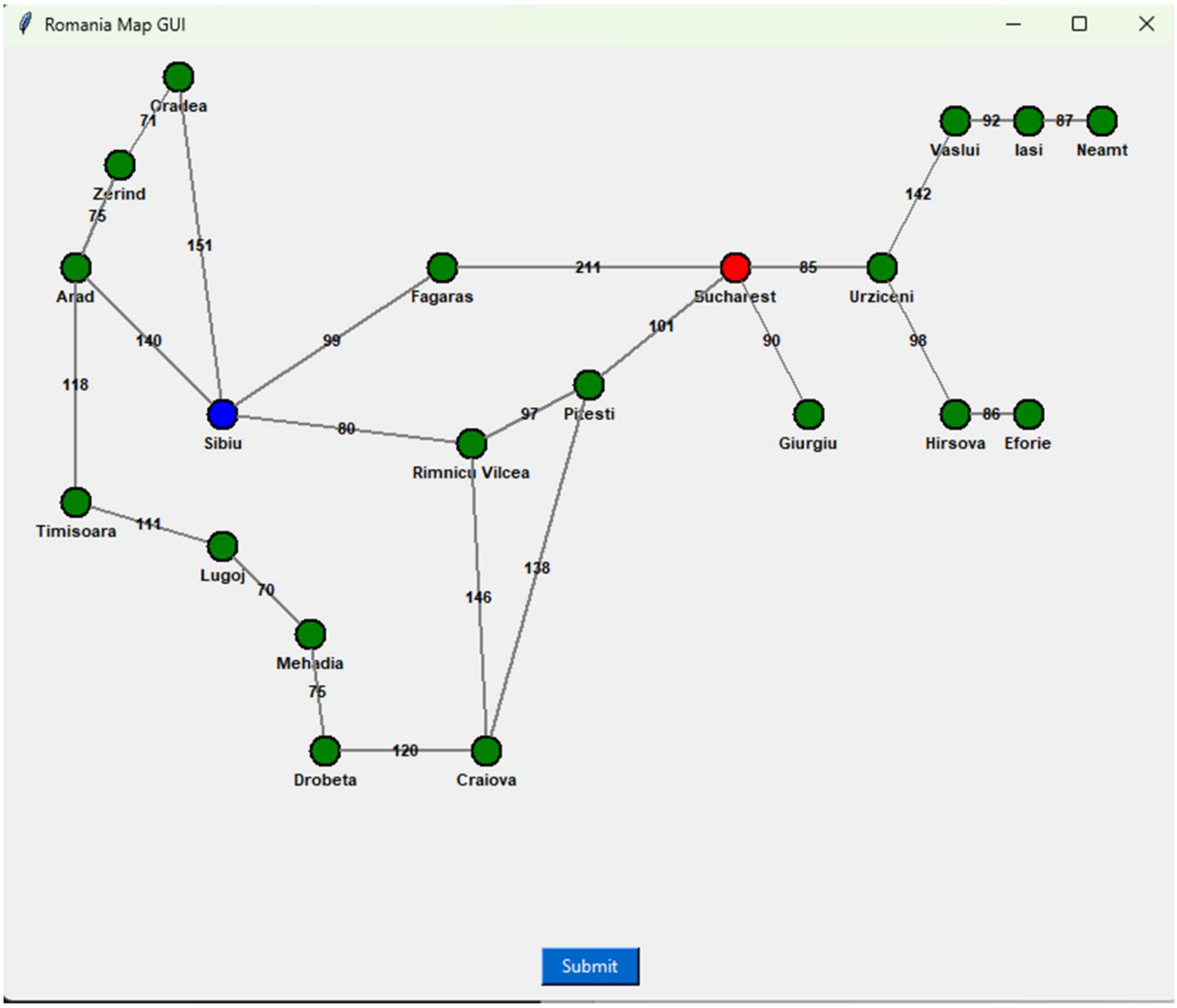
3. Project Demonstration:

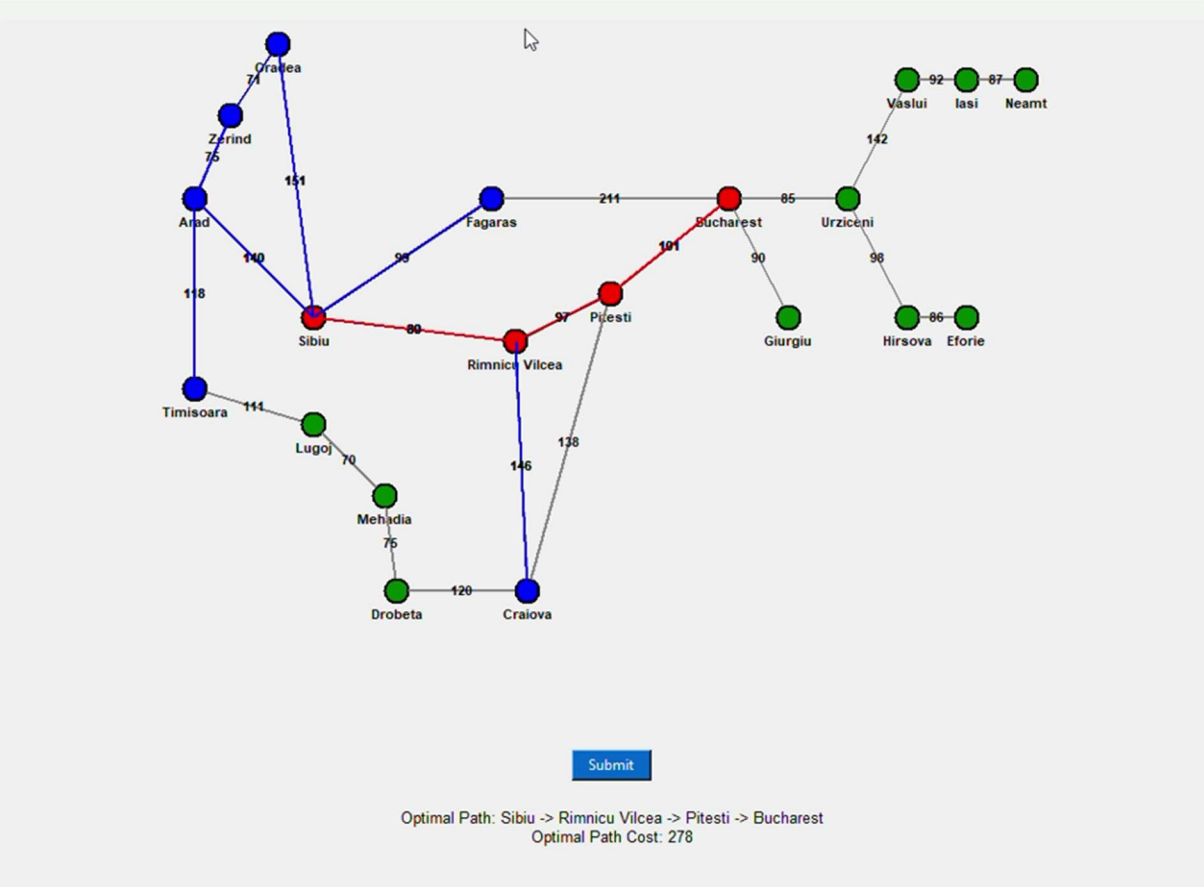
A video demonstrating the working of the GUI along with screenshots of different stages of the search process will be provided as part of the project submission.

20240406-1404-10.1
725664.mp4

Output-




Romania Map GUI
Oradea
71
Zerind
75
151
Arad
140
118
Timisoara
111
Lugoj
70
Mehadia
75
Drobeta
120
Craiova
146
Sibiu
99
80
Rimnicu Vilcea
97
Pitesti
138
Fagaras
211
Bucharest
101
90
Giurgiu
85
Urziceni
142
98
Vaslui
92
Iasi
87
Neamt
Hirsova
86
Eforie
Submit



Conclusion:

In conclusion, the Romania Map GUI project successfully demonstrates the implementation of the Uniform Cost Search algorithm for finding the optimal path between two cities on the Romania map. The GUI provides an intuitive interface for users to select their desired source and destination cities, and the algorithm execution is visualized step by step, enhancing understanding and transparency.

The project combines various programming concepts such as graph representation, algorithm implementation, and GUI development using libraries like tkinter and networkx. It provides a practical application of graph theory and search algorithms in solving real-world problems like route planning.

Furthermore, the project can be extended by incorporating additional features such as heuristic search algorithms (e.g., A*), allowing users to customize the map and add new cities or connections, and integrating real-time data for

dynamic route planning.

Overall, the Romania Map GUI project serves as an educational tool for learning about graph traversal algorithms and their application in solving pathfinding problems, while also offering a user-friendly interface for interactive exploration of the Romania map.