Name : Harpalsinh Bhati

Enr. No.: 22162171010

Sub: Algorithm Analysis & Design

Branch: CS

Batch: 54

Sem: 5-B

Practical 11

AIM:

A government official needs to visit several cities within a state. To minimize travel costs, they want to find the shortest path between their starting city and each destination city.

Task:

Given a graph representing the cities and their connecting roads, determine the minimum cost path from a given starting city to all other cities.

Input:

Enter total number of nodes: 5

Enter the node from where you want to calculate the distance: A

Enter Data (Weight):

	A	*B*	*C*	*D*	*E*
A	0	20	30	∞	∞
B	∞	0	∞	15	∞
C	∞	∞	0	∞	25
D	∞	∞	∞	0	10
E	∞	∞	∞	∞	0

Output:

	A	*B*	*C*	*D*	*E*
A	0	20	30	35	45
B	∞	0	∞	15	25
C	∞	∞	0	∞	25
D	∞	∞	∞	0	10
E	∞	∞	∞	∞	0

OR

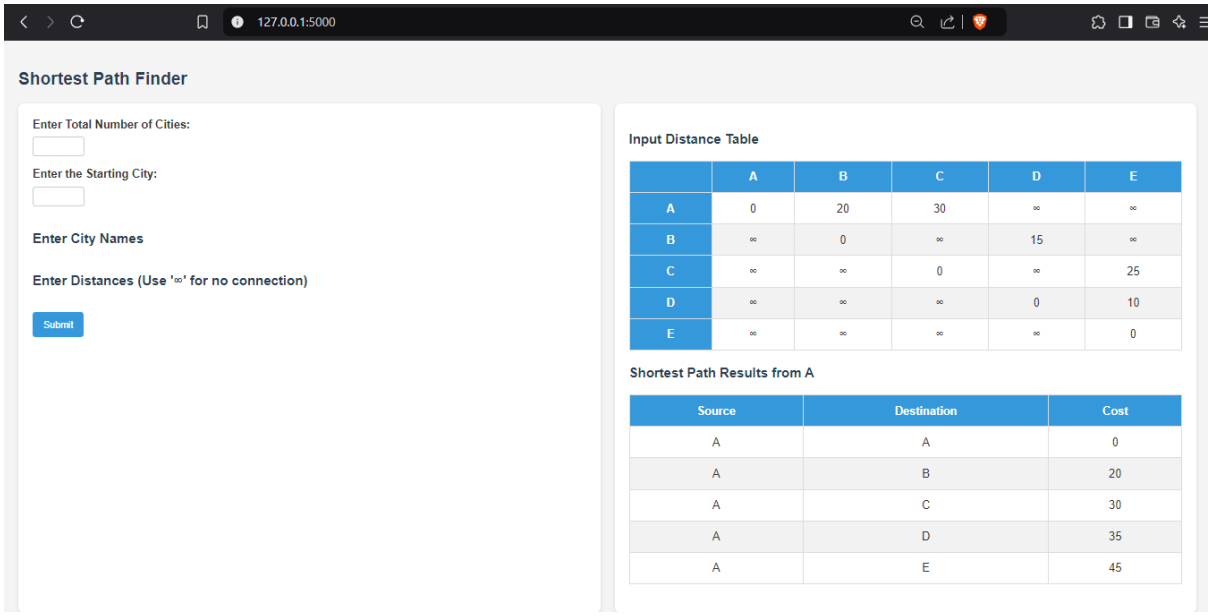
Source	Destination	Cost
A	A	0
	B	20
	C	30
	D	35
	E	45

Code:

```
from flask import Flask, render_template, request
import heapq

app = Flask(__name__)

def dijkstra(graph, start):
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    priority_queue = [(0, start)]

    while priority_queue:
        current_distance, current_node = heapq.heappop(priority_queue)

        if current_distance > distances[current_node]:
            continue

        for neighbor, weight in graph[current_node].items():
            if weight == float('inf'):
                continue  # Skip unconnected nodes
            distance = current_distance + weight

            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(priority_queue, (distance, neighbor))

    return distances
```

```
@app.route("/", methods=["GET", "POST"])
def index():
    result = None
    nodes = 0
    start_city = None
    graph = {}

    if request.method == "POST":
        nodes = int(request.form.get("nodes"))
        start_city = request.form.get("start_city").upper()
        cities = [request.form.get(f"city_{i}").upper() for i in range(nodes)]
        graph = {city: {} for city in cities}

        for i in range(nodes):
            for j in range(nodes):
                neighbor = cities[j]
                weight = request.form.get(f"weight_{i}_{j}")
                weight_value = float('inf') if weight == '∞' else int(weight)
                graph[cities[i]][neighbor] = weight_value

        result = dijkstra(graph, start_city)

        for city in result:
            if result[city] == float('inf'):
                result[city] = '∞'
            else:
                result[city] = str(result[city])

        renderable_graph = {
            city: {neighbor: ('∞' if weight == float('inf') else str(weight))
                   for neighbor, weight in neighbors.items()}
            for city, neighbors in graph.items()
        }

        return render_template("Prac_11.html", result=result, nodes=nodes,
start_city=start_city, graph=renderable_graph)

    return render_template("Prac_11.html", result=result, nodes=nodes,
start_city=start_city, graph=graph)

if __name__ == "__main__":
    app.run(debug=True)
```

Output:


127.0.0.1:5000
Shortest Path Finder
Enter Total Number of Cities:
Enter the Starting City:
Enter City Names
Enter Distances (Use '∞' for no connection)
Submit
Input Distance Table
A B C D E
A 0 20 30 ∞ ∞
B ∞ 0 ∞ 15 ∞
C ∞ ∞ 0 ∞ 25
D ∞ ∞ ∞ 0 10
E ∞ ∞ ∞ ∞ 0
Shortest Path Results from A
Source Destination Cost
A A 0
A B 20
A C 30
A D 35
A E 45