Name : Harpalsinh Bhati

Enr. No.: 22162171010

Sub: Algorithm Analysis & Design

Branch: CS

Batch: 54

Sem: 5-B

Practical 12

"Rocket Singh: Salesman of the Year" is a travelling salesman, who sales good in various cities. One day in the morning, he decided to visit all the cities to sales good and come back to the starting city (from where he has started). Travelling Salesman Problem (TSP) is a touring problem in which n cities and distance between each pair is given. We have to help him to find a shortest route to visit each city exactly once and come back to the starting point.

Sample Input:

[[∞, 20, 30, 10, 11],
[15, ∞, 16, 4, 2],
[3, 5, ∞, 2, 4],
[19, 6, 18, ∞, 3],
[16, 4, 7, 16, ∞]]

Sample Output:

Minimum Path
1 – 4 = 10
4 – 2 = 6
2 – 5 = 2
5 – 3 = 7
3 – 1 = 3

Minimum cost: 28
Path Taken: 1 - 4 - 2 - 5 - 3 – 1

Code:

```
from flask import Flask, render_template, request
from itertools import permutations
import numpy as np

app = Flask(__name__)

def tsp(distance_matrix):
    n = len(distance_matrix)
    min_path_cost = float('inf')
```

```
    best_path = []
    best_segments = []

    # Generate all permutations of node indices to find the shortest path
    for perm in permutations(range(n)):
        current_cost = 0
        current_segments = []

        for i in range(n):
            current_cost += distance_matrix[perm[i]][perm[(i + 1) % n]]
            current_segments.append((perm[i] + 1, perm[(i + 1) % n] + 1,
distance_matrix[perm[i]][perm[(i + 1) % n]]))

        # Update best path if the current path has a lower cost
        if current_cost < min_path_cost:
            min_path_cost = current_cost
            best_path = perm
            best_segments = current_segments

    return best_path, min_path_cost, best_segments

@app.route('/', methods=['GET', 'POST'])
def index():
    input_matrix = []
    if request.method == 'POST':
        nodes = int(request.form['nodes'])
        distance_matrix = np.full((nodes, nodes), np.inf)

        # Fill the distance matrix with user-provided weights
        for i in range(nodes):
            row = []
            for j in range(nodes):
                weight_key = f'weight_{i}_{j}'
                weight_value = request.form[weight_key]
                if weight_value == '∞':
                    distance_matrix[i][j] = float('inf')  # Use infinity for
unreachable paths
                else:
                    distance_matrix[i][j] = int(weight_value)
                row.append(weight_value)
            input_matrix.append(row)

        # Get the shortest path and segments
        path, min_cost, segments = tsp(distance_matrix)
        path_display = ' - '.join(str(i + 1) for i in path) + ' - 1'
        segments_display = ', '.join([f"{start} - {end} = {cost}" for start,
end, cost in segments])
```

```
        output = {
            'path': path_display,
            'cost': min_cost,
            'input_matrix': input_matrix,
            'segments': segments_display
        }
        return render_template('Prac_12.html', output=output)

    return render_template('Prac_12.html', output=None)

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

Output: