Name:-Devansh Koyani
Erno:-22162171007
Batch:-54

Institute of Computer Technology
B. Tech Computer Science and Engineering

Sub: Algorithm Analysis and Design

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('prac12.html', output=output)

    return render_template('prac12.html', output=None)

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

html

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>TSP Solver</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f4f4;
            color: #333;
            margin: 0;
            padding: 20px;
        }

        h1,
        h2,
        h3 {
            color: #2c3e50;
        }

        .container {
            display: flex;
            flex-wrap: wrap;
            justify-content: space-between;
        }

        .form-container,
        .result-container {
            background: #ffffff;
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
            margin: 0 10px;
            flex: 1;
            min-width: 300px;
            max-width: 48%;
        }
```

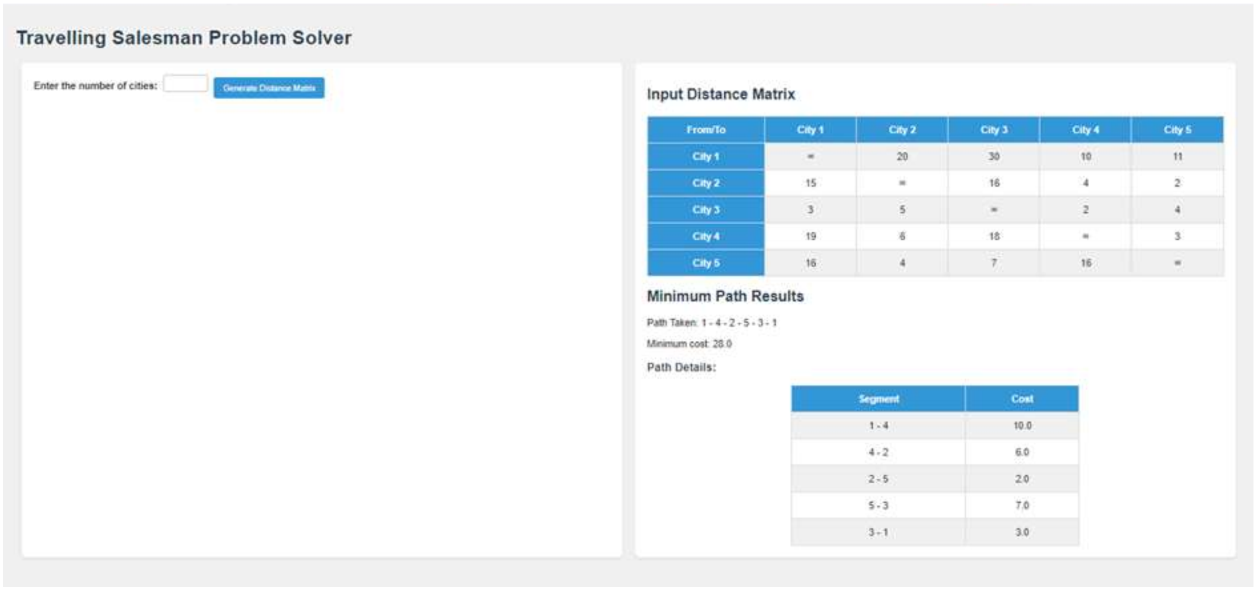
```css
label {
    display: block;
    margin-bottom: 8px;
    font-weight: bold;
}

input[type="number"],
input[type="text"] {
    width: 60px;
    padding: 5px;
    margin-right: 10px;
    margin-bottom: 15px;
    border: 1px solid #ccc;
    border-radius: 4px;
    transition: border-color 0.3s;
}

input[type="number"]:focus,
input[type="text"]:focus {
    border-color: #3498db;
    outline: none;
}

button {
    background-color: #3498db;
    color: white;
    padding: 10px 15px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    transition: background-color 0.3s;
}

button:hover {
    background-color: #2980b9;
}

.submit-button {
    background-color: #e67e22;
```

```css
        color: white;
        padding: 10px 20px;
        border: none;
        border-radius: 4px;
        cursor: pointer;
        transition: background-color 0.3s;
        font-weight: bold;
    }

    .submit-button:hover {
        background-color: #d35400;
    }

    table {
        border-collapse: collapse;
        width: 100%;
        margin: 20px 0;
        background-color: #ffffff;
    }

    table,
    th,
    td {
        border: 1px solid #ddd;
    }

    th,
    td {
        padding: 12px;
        text-align: center;
    }

    th {
        background-color: #3498db;
        color: white;
    }

    tr:nth-child(even) {
        background-color: #f2f2f2;
    }
```

```
        tr:hover {
            background-color: #d1e7fd;
        }

        .input-row {
            display: flex;
            align-items: center;
            margin-bottom: 15px;
        }

        .city-names,
        .distance-inputs {
            display: flex;
            justify-content: flex-start;
            flex-wrap: wrap;
            margin-bottom: 15px;
        }

        .distance-input-row {
            display: flex;
            align-items: center;
            margin-bottom: 10px;
        }

        .distance-input-row label {
            margin-right: 10px;
        }
    </style>
</head>

<body>
    <h1>Travelling Salesman Problem Solver</h1>
    <div class="container">
        <div class="form-container">
            <form id="nodes-form" method="post">
                  <div style="display: flex; align-items: center;"> <label
for="nodes" style="margin-right: 10px;">Enter
                        the number of cities:</label> <input type="number"
id="nodes" name="nodes" min="2" required>
```

```
                                                   <button   type="button"
onclick="generateMatrixInputs()">Generate Distance Matrix</button>
                </div>
            </form>
            <form id="matrix-form" method="post" style="display: none;">
                     <div id="graphContainer"></div> <input type="hidden"
id="nodes-hidden" name="nodes" value=""> <input
                    type="submit" value="Submit" class="submit-button">
            </form>
        </div>
         <div class="result-container"> {% if output %} <h2>Input Distance
Matrix</h2>
            <table>
                <tr>
                                         <th>From/To</th>  {%  for  j  in
range(output.input_matrix|length) %} <th>City {{ j + 1 }}</th> {%
                    endfor %}
                   </tr> {% for i in range(output.input_matrix|length) %}
<tr>
                                <th>City  {{  i  +  1  }}</th>  {%  for  j  in
range(output.input_matrix[i]|length) %} <td>{{
                        output.input_matrix[i][j] }}</td> {% endfor %}
                </tr> {% endfor %}
            </table>
            <h2>Minimum Path Results</h2>
            <p>Path Taken: {{ output.path }}</p>
            <p>Minimum cost: {{ output.cost }}</p>
            <h3>Path Details:</h3>
            <table style="width: 50%; margin: auto;">
                <tr>
                    <th>Segment</th>
                    <th>Cost</th>
                   </tr> {% for segment in output.segments.split(', ') %}
<tr>
                    <td>{{ segment.split(' = ')[0] }}</td>
                    <td>{{ segment.split(' = ')[1] }}</td>
                </tr> {% endfor %}
            </table> {% endif %}
        </div>
    </div>
    <script> function generateMatrixInputs() {
                 const  nodes  =  document.getElementById('nodes').value;
document.getElementById('nodes-hidden').value     =     nodes;     const
graphContainer          =         document.getElementById('graphContainer');
graphContainer.innerHTML = ''; // Clear previous inputs
             for (let i = 0; i < nodes; i++) { graphContainer.innerHTML +=
`<h4>Distances   from   City   ${i   +   1}</h4>`;   const   rowDiv   =
document.createElement('div'); rowDiv.classList.add('distance-input-row');
for   (let   j   =   0;   j   <   nodes;   j++)   {   const   label   =
document.createElement('label');                label.setAttribute('for',
`weight_${i}_${j}`); label.textContent = `To City ${j + 1}:`; const input
=  document.createElement('input');  input.type  =  'text';  input.id  =
`weight_${i}_${j}`; input.name = `weight_${i}_${j}`; input.defaultValue =
'∞';     rowDiv.appendChild(label);     rowDiv.appendChild(input);     }
graphContainer.appendChild(rowDiv);                                    }
document.getElementById('matrix-form').style.display = 'block';
        }
    </script>
</body>

</html>
```

Output:-



Travelling Salesman Problem Solver

Enter the number of cities:

Input Distance Matrix

From/To	City 1	City 2	City 3	City 4	City 5
City 1	∞	20	30	10	11
City 2	15	∞	16	4	2
City 3	3	5	∞	2	4
City 4	19	6	18	∞	3
City 5	16	4	7	16	∞

Minimum Path Results

Path Taken: 1 - 4 - 2 - 5 - 3 - 1

Minimum cost: 28.0

Path Details:

Segment	Cost
1 - 4	10.0
4 - 2	6.0
2 - 5	2.0
5 - 3	7.0
3 - 1	3.0