**CAPSTONE PROJECT**

**Efficient Solutions for Minimum Cost 2D Plane Point Connection: A Mathematical, Graph Theoretic, and Algorithmic Approach**

**CSA0637 - Design and Analysis of Algorithms for Dynamic Programming**

**SAVEETHA SCHOOL OF ENGINEERING**

**SIMATS ENGINEERING**



**Supervisor**

**M.Joy Priyanka**

**DONE BY**

**Y.Shiva Prasad (192210273)**

**K.Chaithanya prakash (192210276)**

**M.Kushwanth Varma(192210202)**

**APRIL 2024**

**Problem Statement**

You are given a set of points on a 2D plane, where each point is represented by its integer coordinates. The task is to find the minimum cost required to connect all these points such that there is exactly one simple path between any two points. The cost of connecting two points is the Manhattan distance between them, which is the sum of the absolute differences of their x and y coordinates.

**Proposed Design Work**

**1.Identifying the Key Components**

* **Creating Edges:**

For each pair of points, calculate the Manhattan distance (sum of absolute differences in coordinates--given in question).

Store the distance with the indices of the two points in the edges vector. like pair<distance<pair< i , j>>edge

* **Sorting Edges:**

Sort the edges in ascending order based on their distances.

* **Initializing Parent Array:**

Create a parent array where initially each point to its own parent.

* **Iterating Through Edges:**

For each edge in the sorted list of edges:

Get the distance and indices of the two points (u and v) connected by the edge.

Find the parents of u and v using the find Parent helper function.

* **Counting Edges:**

Increment the Number of Edges counter to keep track of the number of edges added to the MST.

* **Exiting the Loop:**

If we have added enough edges to form an MST (equal to n - 1 edges for n points), exit the loop.

* **Returning Minimum Cost:**

Return the total minimum cost of connecting all points.

**2.Functionality**

* **Find Parent Helper Function:**

This is a recursive function to find the parent of a set. It uses path compression for efficiency.

This algorithm follows the steps of Kruskal's algorithm, a greedy approach for finding the Minimum Spanning Tree (MST) of a graph.

* **Minimum Cost Calculation Function:**

Finds the minimum cost to connect all points using Kruskal's algorithm. Initializes points, calls the minimum cost calculation function, and prints the result.

**3.Architectural Design:**

**Input Data and module**

**Kruskal’s Algorithm**

**Edge Representation**

**Input handling**

**Validation Module**

Output presentation

Point representation

**UI Design**

**1.Layout Design**

* **Flexible Layout:**

The UI should be responsive and able to adapt to different screen sizes and orientations.

* **User-Friendly:**

Intuitive arrangement of elements for easy understanding and interaction by the user.

* **Colour Selection:**

Choose colours that are visually appealing and provide good contrast for readability

**2.Feasible Elements Used**

* **Elements Positioning:**

Place elements such as buttons, input fields, and labels logically to guide the user through the process.

* **Accessibility:**

Ensure that the UI is accessible to users with disabilities by using appropriate contrast, font sizes, and labelling.

* **Array Input:**

Provide an interface for users to input the array of points representing integer coordinates on a 2D plane

* **Calculate Button:**

Button to trigger the calculation of the minimum cost to connect all points based on the provided input.

* **Result Display:**

Show the minimum cost calculated to make all points connected.

* **Error Handling:**

or if there is any issue during the calculation Display error messages if the input is invalid process.

**3. Elements Function**

* **Input Elements:**

Provide a way for the user to input the coordinates of points on the 2D plane.

* **Calculate Distance**

Implement a function to calculate the Manhattan distance between two given points.

* **Build Function**

Create a graph representation where each point is a node, and the edges represent connections between points. This could involve creating an adjacency matrix or list.

* **Minimum Spanning Tree (MST)**

Implement an algorithm to find the minimum spanning tree of the graph. Kruskal's algorithm is commonly used for this purpose.

* **Cost Calculation:**

Sum up the weights (Manhattan distances) of the edges in the minimum spanning tree to determine the minimum cost to connect all points.

* **Output Result:**

Display the minimum cost to the user.

**CODE:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Minimum Cost to Connect All Points</title>

<style>

canvas {

border: 1px solid black;

}

</style>

</head>

<body>

<h1>Minimum Cost to Connect All Points</h1>

<p>Enter points in the format [[x1,y1],[x2,y2],...]</p>

<form id="inputForm">

<label for="pointsInput">Points:</label><br>

<input type="text" id="pointsInput" name="pointsInput" required><br><br>

<button type="submit">Calculate Minimum Cost</button>

</form>

<div id="output"></div>

<canvas id="canvas"></canvas>

<script>

document.getElementById('inputForm').addEventListener('submit', function(event) {

event.preventDefault();

const pointsInput = document.getElementById('pointsInput').value;

const points = JSON.parse(pointsInput);

const { cost, connections } = minCostConnectPoints(points);

document.getElementById('output').innerText = Minimum cost: ${cost};

drawGraph(points, connections);

});

function minCostConnectPoints(points) {

let cost = 0;

const n = points.length;

const connections = [];

const visited = new Array(n).fill(false);

const distances = new Array(n).fill(Infinity);

distances[0] = 0;

for (let i = 0; i < n; i++) {

let minDistance = Infinity;

let u = -1;

for (let j = 0; j < n; j++) {

if (!visited[j] && distances[j] < minDistance) {

minDistance = distances[j];

u = j;

}

}

visited[u] = true;

cost += minDistance;

for (let v = 0; v < n; v++) {

const dist = Math.abs(points[u][0] - points[v][0]) + Math.abs(points[u][1] - points[v][1]);

if (!visited[v] && dist < distances[v]) {

distances[v] = dist;

connections[v] = u;

}

}

}

return { cost, connections };

}

function drawGraph(points, connections) {

const canvas = document.getElementById('canvas');

const ctx = canvas.getContext('2d');

const canvasWidth = canvas.width;

const canvasHeight = canvas.height;

ctx.clearRect(0, 0, canvasWidth, canvasHeight);

// Find the range of x and y coordinates

let minX = Infinity, maxX = -Infinity, minY = Infinity, maxY = -Infinity;

for (const [x, y] of points) {

minX = Math.min(minX, x);

maxX = Math.max(maxX, x);

minY = Math.min(minY, y);

maxY = Math.max(maxY, y);

}

// Scale the points to fit within the canvas

const scaleX = canvasWidth / (maxX - minX + 1);

const scaleY = canvasHeight / (maxY - minY + 1);

// Draw lines connecting points

ctx.strokeStyle = "blue";

for (let i = 0; i < points.length; i++) {

if (connections[i] !== undefined) {

const startX = (points[i][0] - minX) \* scaleX;

const startY = (points[i][1] - minY) \* scaleY;

const endX = (points[connections[i]][0] - minX) \* scaleX;

const endY = (points[connections[i]][1] - minY) \* scaleY;

ctx.beginPath();

ctx.moveTo(startX, startY);

ctx.lineTo(endX, endY);

ctx.stroke();

}

}

// Draw points

ctx.fillStyle = "black";

for (const [x, y] of points) {

const scaledX = (x - minX) \* scaleX;

const scaledY = (y - minY) \* scaleY;

ctx.beginPath();

ctx.arc(scaledX, scaledY, 5, 0, 2 \* Math.PI);

ctx.fill();

}

}

</script>

</body>

</html>

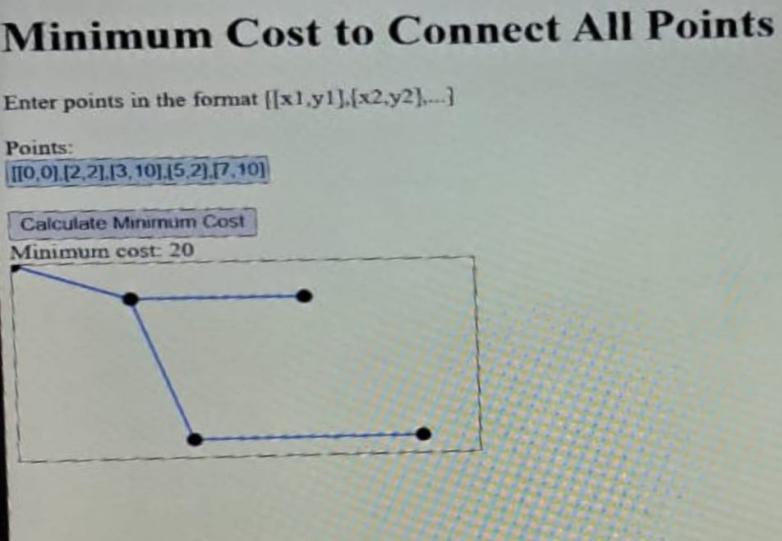
**Output**

Enter Points in the Format[[x], y], [x2, y2] …]

Points: [[0,0], [2,2], [3,10], [5,2], [7,10]]

Calculate Minimum Cost

Minimum cost to connect all points: 20



**CONCLUSION:**

Solving the problem involves a combination of mathematical calculations, graph theory concepts, and algorithmic techniques. By properly implementing these components and functions, we can efficiently determine the minimum cost to connect all points on the 2D plane while meeting the given conditions.