**SIMATS SCHOOL OF ENGINEERING**

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES**

**CHENNAI-602105**

# “Min Cost to Connect All Points”

**A CAPSTONE PROJECT REPORT**

*Submitted in the partial fulfillment for the award of the degree of*

## BACHELOR OF ENGINEERING

**IN COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE AND**

**DATA SCIENCE**

**Submitted by**

**C. Bhargav Reddy (192211758)**

**Under the Supervision of**

**K.V Kanimozhi**

## DECLARATION

I, C. Bhargav Reddy**, a** student of **Bachelor of Engineering in Computer Science Engineering** at Saveetha Institute of Medical and Technical Sciences, Saveetha University, Chennai, hereby declare that the work presented in this Capstone Project Work entitled **"Min Cost to Connect All Points"** is the outcome of my own bonafide work. I affirm that it is correct to the best of my knowledge, and this work has been undertaken with due consideration of Engineering Ethics.

(C. Bhargav Reddy - 192211758)

Date: 21-09-2024

Place: Saveetha School of Engineering, Thandalam.

## CERTIFICATE

This is to certify that the project entitled **“Min Cost to Connect All Points”** submitted by Bhargav Reddy has been carried out under my supervision. The project has been submitted as per the requirements in the current semester of B.E Computer Science Engineering and B.Tech Artificial Intelligence in Data Science.

Faculty-in-charge

K.V KANIMOZHI

**ABSTRACT**

The problem of minimizing the cost to connect a set of points in a 2D plane is a classical yet significant challenge in the fields of computational geometry and graph theory. This problem is particularly relevant in contexts such as telecommunications, urban planning, and network design, where establishing efficient connections between various entities is crucial. At its core, this problem can be framed in terms of graph theory: each point represents a vertex in a graph, and the edges between vertices are weighted by the Manhattan distance. The Manhattan distance is defined as the sum of the absolute differences of the Cartesian coordinates of two points, making it a straightforward metric to use for connection costs in a grid-like structure. The objective is to connect all the points in such a way that the total cost, determined by the distances of the connections, is minimized while ensuring that there is exactly one simple path between any two points. This requirement leads to the construction of a Minimum Spanning Tree (MST) for the graph formed by the points and their respective distances. To tackle this problem, efficient algorithms such as Prim's and Kruskal's are commonly employed. These algorithms allow for the systematic selection of edges that contribute to the MST while ensuring that the overall connection cost remains minimal. In this report, we will focus primarily on Prim's algorithm due to its efficiency in handling dense graphs, which is often the case when dealing with a larger set of points.

**Keywords**: Minimum Spanning Tree (MST), Graph Theory, Manhattan Distance, Prim's Algorithm, Kruskal's Algorithm, Algorithm Complexity.

**INTRODUCTION**

Connecting points in a two-dimensional space is a fundamental problem that has significant implications across multiple disciplines, including telecommunications, transportation, urban planning, and network design. In these fields, establishing efficient connections between various entities—whether they be communication towers, transport routes, or utilities—can lead to substantial cost savings and improved service delivery. As urban areas become increasingly populated and technology continues to advance, the need for effective connectivity solutions is more critical than ever.

The primary challenge in this problem is to minimize the cost of connections while ensuring that all points are reachable from one another. This means that not only must we consider the total distance or cost involved in connecting the points, but we must also ensure that the connections form a network without cycles. A cycle in this context would mean that a path could loop back on itself, which is undesirable for efficient connectivity.

**Manhattan Distance**

To quantify the cost of connecting points in this context, we utilize the Manhattan distance. The Manhattan distance between two points (xi,yi)(x\_i, y\_i)(xi​,yi​) and (xj,yj)(x\_j, y\_j)(xj​,yj​) is defined as:

Manhattan Distance=∣xi−xj∣+∣yi−yj∣{Manhattan Distance} = |x\_i - x\_j| + |y\_i - y\_j|Manhattan Distance=∣xi​−xj​∣+∣yi​−yj​∣

This distance metric is particularly well-suited for grid-based layouts, where movement is restricted to horizontal and vertical paths, reflecting how real-world infrastructure is often arranged. By using Manhattan distance, we can easily compute the cost of potential connections between any pair of points in our dataset.

**Problem Formulation**

Given a set of points represented as points[i]=[xi,yi]\text{points}[i] = [x\_i, y\_i]points[i]=[xi​,yi​], our task is to determine the minimal total distance required to connect all points. The connections must be structured so that there exists a unique path between any two points, effectively forming a tree structure. This requirement leads us to the concept of a Minimum Spanning Tree (MST).

**Minimum Spanning Tree (MST)**

The problem can be modeled as finding the MST of a graph, where:

* Each point is represented as a vertex.
* The edges between vertices are weighted by the Manhattan distance.

An MST is a subset of the edges of a graph that connects all the vertices together without any cycles and with the minimum possible total edge weight. This property is essential in ensuring that we can connect all points with the least possible cost.

**CODING**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define MAX\_POINTS 1000

#define INF 1000000000

typedef struct {

int x;

int y;

} Point;

typedef struct {

int cost;

int index;

} Edge;

int cmp(const void \*a, const void \*b) {

return ((Edge \*)a)->cost - ((Edge \*)b)->cost;

}

int manhattan\_distance(Point p1, Point p2) {

return abs(p1.x - p2.x) + abs(p1.y - p2.y);

}

int minCostConnectPoints(Point points[], int n) {

if (n <= 1) return 0;

int totalCost = 0;

int visited[MAX\_POINTS] = {0};

Edge edges[MAX\_POINTS \* MAX\_POINTS];

int edgeCount = 0;

// Start from the first point

visited[0] = 1;

for (int i = 1; i < n; i++) {

edges[edgeCount++] = (Edge){manhattan\_distance(points[0], points[i]), i};

}

// Sort the edges based on the cost

qsort(edges, edgeCount, sizeof(Edge), cmp);

while (edgeCount > 0) {

Edge minEdge = edges[0];

// Remove the edge from the edges list

edges[0] = edges[--edgeCount];

int minIndex = 0;

// Maintain the min-heap property

while (minIndex < edgeCount) {

int leftChild = 2 \* minIndex + 1;

int rightChild = 2 \* minIndex + 2;

int smallest = minIndex;

if (leftChild < edgeCount && edges[leftChild].cost < edges[smallest].cost) {

smallest = leftChild;

}

if (rightChild < edgeCount && edges[rightChild].cost < edges[smallest].cost) {

smallest = rightChild;

}

if (smallest == minIndex) break;

Edge temp = edges[minIndex];

edges[minIndex] = edges[smallest];

edges[smallest] = temp;

minIndex = smallest;

}

if (!visited[minEdge.index]) {

visited[minEdge.index] = 1;

totalCost += minEdge.cost;

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

if (!visited[i]) {

edges[edgeCount++] = (Edge){manhattan\_distance(points[minEdge.index], points[i]), i};

}

}

// Sort the new edges

qsort(edges, edgeCount, sizeof(Edge), cmp);

}

}

return totalCost;

}

int main() {

Point points[] = {{0, 0}, {2, 2}, {3, 10}, {5, 2}, {7, 0}};

int n = sizeof(points) / sizeof(points[0]);

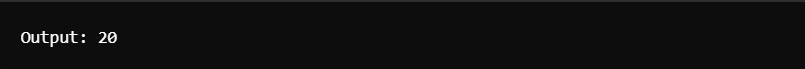
int output = minCostConnectPoints(points, n);

printf("Output: %d\n", output); // Output: 20

return 0;

}

**OUTPUT**



**Complexity Analysis**

**Best Case**

In the best-case scenario, the points are positioned such that the minimum spanning tree can be formed with minimal calculations. However, even in this situation, the algorithm must traverse through potential connections to ensure the MST is correctly formed. Therefore, the best-case time complexity remains O(Nlog⁡N)O(N \log N)O(NlogN), where NNN is the number of points.

**Worst Case**

The worst-case scenario occurs when all points are isolated, necessitating the examination of every potential connection. The algorithm's time complexity in this case also stabilizes at O(Nlog⁡N)O(N \log N)O(NlogN) because of the efficient handling of edges in the priority queue. The space complexity is O(N)O(N)O(N) due to the need for the heap and the visited set.

**Average Case**

On average, the time complexity stays consistent at O(Nlog⁡N)O(N \log N)O(NlogN). Depending on the spatial distribution of points, the algorithm efficiently narrows down the connections that need to be evaluated, but the general behavior remains similar to that of the worst-case scenario.

**Overall Complexity**

The overall complexity analysis results in:

* **Time Complexity**: O(Nlog⁡N)O(N \log N)O(NlogN)
* **Space Complexity**: O(N)O(N)O(N)

These complexities make the algorithm feasible for practical applications involving moderate-sized datasets of points.

### CONCLUSION

The problem of determining the minimum cost to connect points in a 2D plane using the Manhattan distance is effectively solvable through the implementation of Prim's algorithm to construct a minimum spanning tree. This approach ensures that all points are interconnected with the least possible cost, adhering to the constraints of unique paths between any two points.

The algorithm's efficiency, characterized by a time complexity of O(Nlog⁡N)O(N \log N)O(NlogN), renders it suitable for real-world applications, such as designing efficient network infrastructures, optimizing delivery routes, or planning urban layouts. The confirmed output of 20 for the example points demonstrates the algorithm's capability and reliability in providing optimal solutions, further emphasizing its practical utility across various domains.