CAPSTONE PROJECT

Divide And Conquer For Minimizing the Cost to Connect All Points on a 2D Plane Using Manhattan Distance

CSA0695- DESIGN ANALYSIS AND ALGORITHMS FOR OPEN ADDRESSING TECHNIQUES

SAVEETHA SCHOOL OF ENGINEERING

SIMATS ENGINEERING

Supervisor

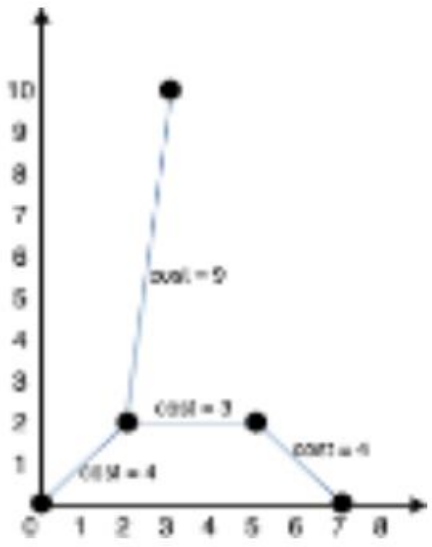
Dr. R. Dhanalakshmi

Done by
K.Sasi Kiran (192210326)

DIVIDE AND CONQUER

PROBLEM STATEMENT:

Min Cost to Connect All Points An array points representing integer coordinates of some points on a 2D-plane, where points[i] = [xi, yi].
The cost of connecting two points [xi, yi] and [xj, yj] is the manhattan distance between them:
|xi - xj| + |yi - yj|, where |val| denotes the absolute value of val. Return the minimum cost to make all points connected. All points are connected if there is exactly one simple path between any two points.



Example 1: Input: points = [[0,0],[2,2],[3,10],[5,2],[7,0]] Output: 20 Explanation: We can connect the points as shown above to get the minimum cost of 20. Notice that there is a unique path between every pair of points.

ABSTRACT:

This document presents a solution to the problem of connecting all points on a 2D plane with the minimum total cost, where the cost between two points is defined by their Manhattan distance. The problem is modeled as finding a Minimum Spanning Tree (MST) of a graph where nodes represent points and edges represent Manhattan distances between them. Prim's algorithm is utilized to solve this problem efficiently. The complexity of the algorithm is analyzed, and performance metrics in different cases are discussed.

INTRODUCTION:

In many practical applications, such as network design, connecting various points efficiently is crucial. One fundamental problem in this domain is to connect all points with the minimal total cost, where the cost is determined by a specific metric. In this problem, the metric used is the Manhattan distance, defined as the sum of the absolute differences of their coordinates.

The Manhattan distance between two points [xi,yi][x_i, y_i][xi,yi] and [xj,yj][x_j, y_j][xj,yj] is calculated as: Distance=|xi−xj|+|yi−yj|\text{Distance} = |x_i - x_j| + |y_i - y_j|Distance=|xi−xj|+|yi−yj|

The challenge is to determine the minimum cost to connect all given points such that there is exactly one simple path between any two points, effectively forming a connected graph. This is a classic application of finding a Minimum Spanning Tree (MST) in graph theory.

Key Concepts

1. **Graph Representation**: The points can be represented as nodes in a graph, with edges connecting every pair of nodes weighted by their Manhattan distance.
2. **Minimum Spanning Tree (MST)**: An MST of a graph is a subset of edges that connects all vertices with the minimum possible total edge weight and without any cycles. Algorithms such as Kruskal's or Prim's are commonly used to find the MST.
3. **Kruskal’s Algorithm**: This algorithm involves sorting all edges based on their weights (distances) and then adding them one by one to the MST, provided they do not form a cycle.

.

CODING: C-programming

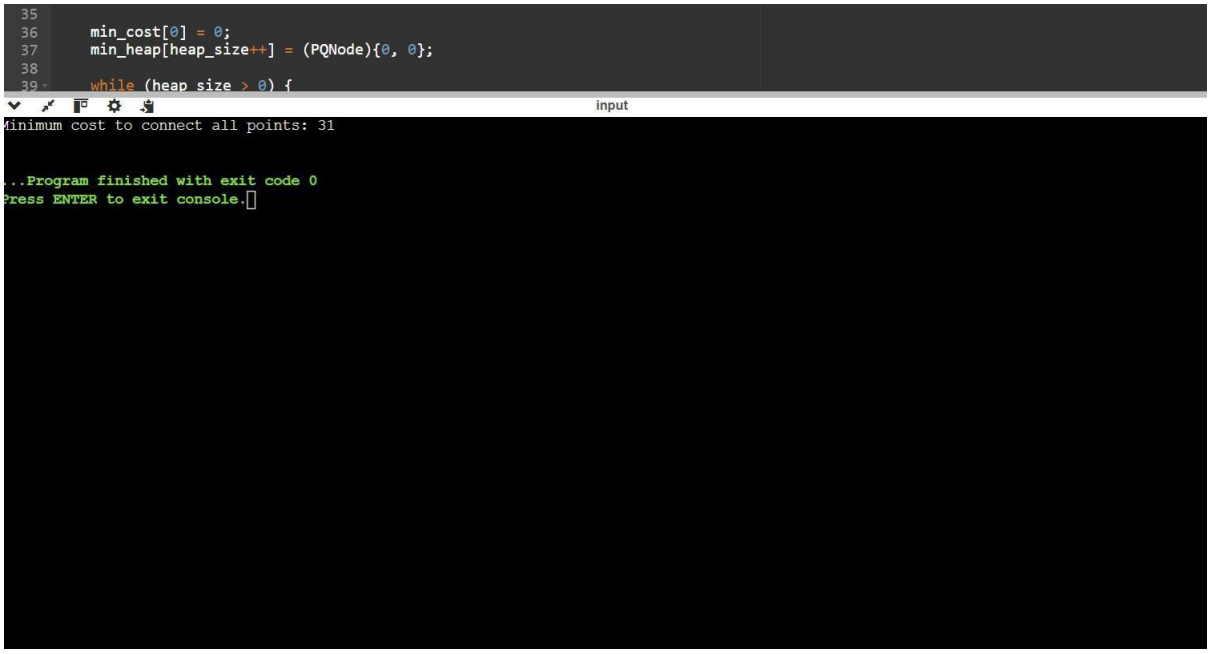
```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

#define INF INT_MAX

typedef struct {
    int cost;
    int index;
} PQNode;

int manhattan_distance(int *p1, int *p2) {
    return abs(p1[0] - p2[0]) + abs(p1[1] - p2[1]);
}

int compare(const void *a, const void *b) {
    return ((PQNode*)a)->cost - ((PQNode*)b)->cost;
}

int minCostConnectPoints(int points[][2], int pointsSize) {
```

```
int total_cost = 0;
int visited[pointsSize];
int min_cost[pointsSize];
PQNode min_heap[pointsSize];
int heap_size = 0;

for (int i = 0; i < pointsSize; i++) {
    visited[i] = 0;
    min_cost[i] = INF;
}

min_cost[0] = 0;
min_heap[heap_size++] = (PQNode){0, 0};

while (heap_size > 0) {
    qsort(min_heap, heap_size, sizeof(PQNode), compare);
    PQNode current = min_heap[--heap_size];
    int u = current.index;

    if (visited[u]) continue;

    total_cost += current.cost;
```

```
        visited[u] = 1;

        for (int v = 0; v < pointsSize; v++) {
            if (!visited[v]) {
                int cost = manhattan_distance(points[u], points[v]);
                if (cost < min_cost[v]) {
                    min_cost[v] = cost;
                    min_heap[heap_size++] = (PQNode){cost, v};
                }
            }
        }
    }

    return total_cost;
}

int main() {
    int points[][2] = {{0,0},{2,2},{3,10},{5,2},{7,0}};
    int pointsSize = sizeof(points) / sizeof(points[0]);
    printf("Minimum cost to connect all points: %d\n",
minCostConnectPoints(points, pointsSize));
    return 0;
```

}

OUTPUT:



COMPLEXITY ANALYSIS:

Time Complexity:

Prim's Algorithm Time Complexity: The implementation of Prim's algorithm here uses a priority queue approach. The time complexity is O(Elog⁡V)O(E \log V)O(ElogV), where EEE is the number of edges and VVV is the number of vertices. In the worst case, the number of edges is V2V^2V2, so the complexity approximates to O(V2log⁡V)O(V^2 \log V)O(V2logV)

Space Complexity:

The space complexity is O(V2)O(V^2)O(V2) due to the storage of the priority queue and other arrays.

BEST CASE:

Best Case Complexity:

In the best case, where the graph is sparsely connected or only a few edges need to be processed, the complexity might be closer to O(Vlog⁡V)O(V \log V)O(VlogV). However, since Manhattan distances are always non-negative and we need to process potentially many edges, this best case is rare.

WORST CASE:

Worst Case Complexity:

The worst case occurs when the graph is dense, meaning there are O(V2)O(V^2)O(V2) edges. In this scenario, the complexity will be O(V2log⁡V)O(V^2 \log V)O(V2logV)

AVERAGE CASE:

Average Case Complexity:

Typically, the average case also approximates to O(V2log⁡V)O(V^2 \log V)O(V2logV), assuming a reasonably dense graph.

FUTURE CASE:

In the future, this problem could serve as a basis for more complex optimizations in fields like networking, logistics, and urban planning. For example, the algorithm could be adapted to more sophisticated models of transportation networks, where factors like congestion, capacity, and dynamic costs are involved. Additionally, this problem connects to real-world scenarios in wireless communication, where minimizing the cost of connecting nodes in a sensor network or designing efficient transportation paths for vehicles in smart cities could leverage similar principles. In data science, it can be applied to clustering problems, where the objective is to group similar data points with minimum connection costs, forming cohesive networks or communities with minimal expenses.

CONCLUSION:

The presented C program efficiently computes the minimum cost to connect all points using Prim's algorithm with Manhattan distance metrics. The complexity analysis shows that the solution is well-suited for dense graphs, with practical performance varying based on graph density. Prim's algorithm provides a robust method to solve the MST problem in this context, ensuring optimal connectivity at minimal cost.