# Optimizing Courier Delivery Routes using graphs

-using Kruskal's algorithm

MAY 31, 2023

Application of minimum spanning tree

BY KABILESHWARAN S D - 2021115046

## Problem statement:

Consider you are the manager of the courier delivery service in State. You need to cover all the delivery center cities in shortest path, so that you can save lot of amount in transportation. Find the path.

## Aim:

Our aim is to design a courier service application to optimize the delivery process for a courier service operating in a city.

## Goal :

The goal of the application is to find the most efficient route for delivering couriers to various homes within the city along with the map.

## Solution:

This is achieved by using Kruskal's algorithm to construct a minimum spanning tree of the city's road network.

Here's how the application works:

1. **User Input:**

   The user provides input for the city's road network. This includes the number of homes (vertices), the names of the homes (vertex names), and the distances between the homes (weights of the edges). The input represents the connectivity and distance information of the road network.

2. **Graph Visualization:**

   The application visualizes the input graph, representing the city's road network. It uses the Graphviz library to generate a graphical representation of the input graph, showing the homes as vertices and the roads as edges. This visualization helps to understand the connectivity and distances between the homes.

3. **Minimum Spanning Tree:**

   The application applies Kruskal's algorithm to the input graph to construct a minimum spanning tree. Kruskal's algorithm finds the minimum total weight (distance) tree that connects all the homes without forming cycles. This ensures that every home is reachable while minimizing the total distance traveled.

4. **Minimum Spanning Tree Visualization:**

   The application generates a graphical representation of the minimum spanning tree using Graphviz. The minimum spanning tree visualization shows the optimal route for delivering the couriers. It highlights the edges (roads) that are part of the minimum spanning tree, providing a clear visual representation of the optimized delivery route.

5. **Output:**

   The application displays the path (edges) of the minimum spanning tree, showing the connectivity between the homes and the distances of the roads. It also calculates and displays the total distance of the minimum spanning tree, which represents the optimal distance for delivering the couriers.

By utilizing Kruskal's algorithm and visualizing the input graph and minimum spanning tree, the courier service application provides an efficient and optimized route for delivering couriers to all the homes in the city. This helps to minimize travel distances, reduce delivery time, and improve the overall efficiency of the courier service.

# Application Code :

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <algorithm>
#include <fstream>

using namespace std;

// Structure to represent an edge in the graph
struct Edge {
    string src;
    string dest;
    int weight;

    Edge(string src, string dest, int weight) : src(src), dest(dest), weight(weight)
{}
};

// Function to compare edges based on their weights (for sorting)
bool compareEdges(const Edge& edge1, const Edge& edge2) {
    return edge1.weight < edge2.weight;
}

int findParent(const vector<int>& parent, int vertex) {
    if (parent[vertex] == vertex)
        return vertex;
    return findParent(parent, parent[vertex]);
```

```
}

// Function to perform union of two sets (used in the disjoint set)
void unionSets(vector<int>& parent, vector<int>& rank, int vertex1, int
vertex2) {
    int root1 = findParent(parent, vertex1);
    int root2 = findParent(parent, vertex2);
    if (rank[root1] < rank[root2])
        parent[root1] = root2;
    else if (rank[root1] > rank[root2])
        parent[root2] = root1;
    else {
        parent[root2] = root1;
        rank[root1]++;
    }
}


// Function to construct and print the minimum spanning tree using Kruskal's
algorithm
void constructMinimumSpanningTree(const vector<Edge>& edges, const
unordered_map<string, int>& vertexMap) {
    int numVertices = vertexMap.size();
    vector<int> parent(numVertices);
    vector<int> rank(numVertices, 0);

    // Initialize parent array for the disjoint set
    for (const auto& vertex : vertexMap) {
        parent[vertex.second] = vertex.second;
```

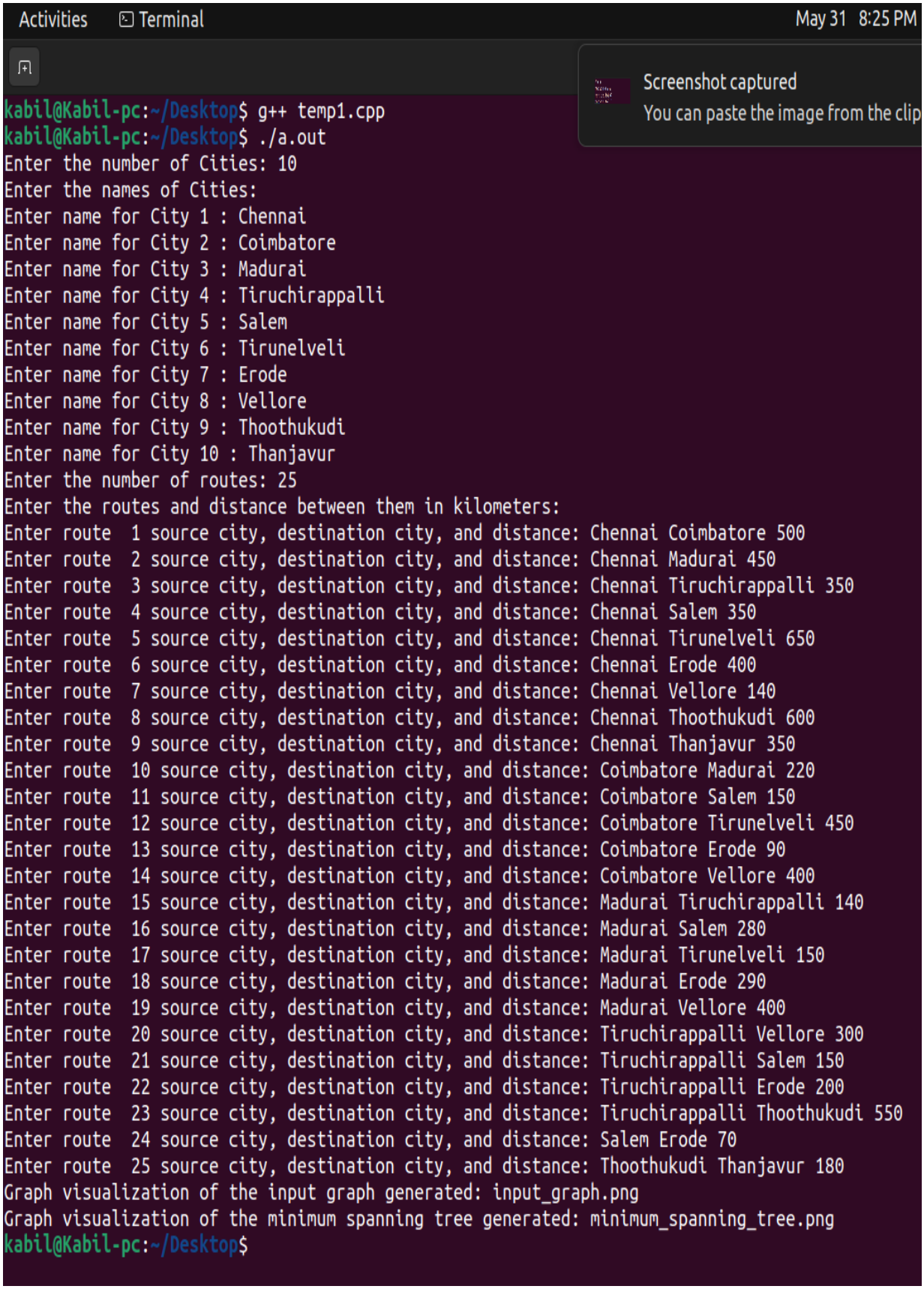
```
}

vector<Edge> minimumSpanningTree;
for (const auto& edge : edges) {
    int srcIndex = vertexMap.at(edge.src);
    int destIndex = vertexMap.at(edge.dest);

    int parent1 = findParent(parent, srcIndex);
    int parent2 = findParent(parent, destIndex);

    // Include the edge in the minimum spanning tree if it doesn't form a cycle
    if (parent1 != parent2) {
        minimumSpanningTree.push_back(edge);
        unionSets(parent, rank, parent1, parent2);
    }
}

// Generate graph visualization using Graphviz
ofstream dotFile("minimum_spanning_tree.dot");
if (dotFile.is_open()) {
    dotFile << "graph MinimumSpanningTree {\n";
    dotFile << "node [shape=circle]\n";

    // Write vertices
    for (const auto& vertex : vertexMap) {
        dotFile << "  " << vertex.first << "\n";
    }

    // Write input edges
```

```
        dotFile << "\n  // Input Edges\n";
        for (const auto& edge : edges) {
            dotFile << "  " << edge.src << " -- " << edge.dest << " [label=\"" <<
edge.weight << "\"]\n";
        }

        // Write minimum spanning tree edges
        dotFile << "\n  // Minimum Spanning Tree Edges\n";
        for (const auto& edge : minimumSpanningTree) {
            dotFile << "  " << edge.src << " -- " << edge.dest << " [label=\"" <<
edge.weight << "\", color=\"red\"]\n";
        }

        dotFile << "}";
        dotFile.close();

        // Generate the visualization using Graphviz
        system("dot -Tpng minimum_spanning_tree.dot -o
minimum_spanning_tree.png");
        cout << "Graph visualization of the minimum spanning tree generated:
minimum_spanning_tree.png" << endl;
    }
}

int main() {
    int numVertices, numEdges;
    cout << "Enter the number of Cities: ";
    cin >> numVertices;
```

```
    vector<Edge> edges;
    unordered_map<string, int> vertexMap; // Map to store vertex names and
their corresponding indices

    cout << "Enter the names of Cities:" << endl;
    for (int i = 0; i < numVertices; i++) {
        string vertexName;
        cout << "Enter name for City " << i + 1 << " : ";
        cin >> vertexName;

        vertexMap[vertexName] = i;
    }
    cout << "Enter the number of routes: ";
    cin >> numEdges;

    cout << "Enter the routes and distance between them in kilometers:" << endl;
    for (int i = 0; i < numEdges; i++) {
        string src, dest;
        int weight;
        cout << "Enter route  " << i + 1 << " source city, destination city, and
distance: ";
        cin >> src >> dest >> weight;

        edges.emplace_back(src, dest, weight);
    }

    // Generate graph visualization using Graphviz for input graph
    ofstream dotFile("input_graph.dot");
    if (dotFile.is_open()) {
```

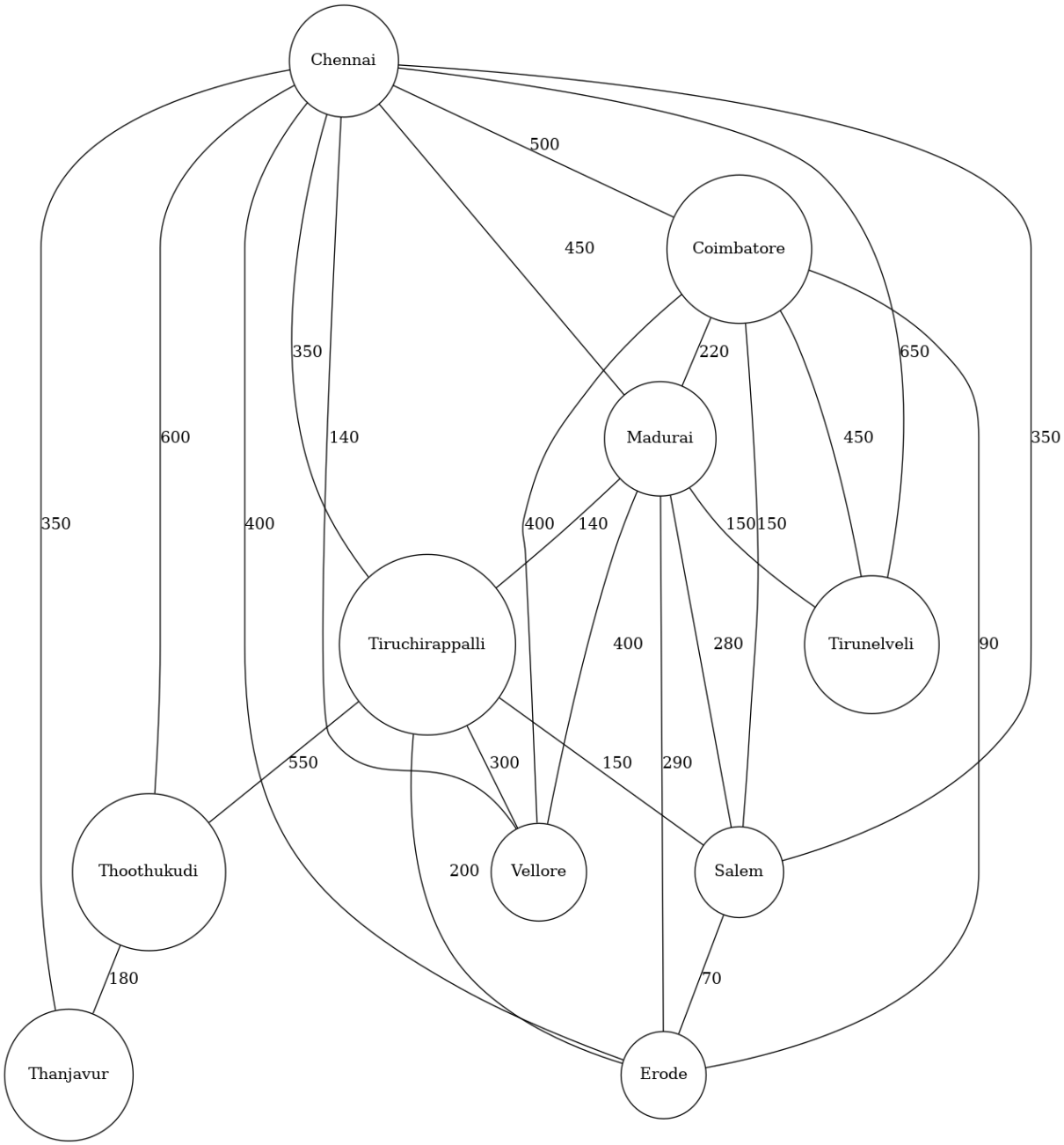
```
        dotFile << "graph InputGraph {\n";
        dotFile << "node [shape=circle]\n";

        // Write vertices
        for (const auto& vertex : vertexMap) {
            dotFile << "  " << vertex.first << "\n";
        }

        // Write edges
        for (const auto& edge : edges) {
            dotFile << "  " << edge.src << " -- " << edge.dest << " [label=\"" <<
edge.weight << "\"]\n";
        }

        dotFile << "}";
        dotFile.close();

        // Generate the visualization using Graphviz for input graph
        system("dot -Tpng input_graph.dot -o input_graph.png");
        cout << "Graph visualization of the input graph generated:
input_graph.png" << endl;
    }

    // Sort the edges based on their weights
    sort(edges.begin(), edges.end(), compareEdges);

    constructMinimumSpanningTree(edges, vertexMap);
    return 0;
}
```

## Input:


Activities
Terminal
May 31 8:25 PM
Screenshot captured
You can paste the image from the clip
kabil@Kabil-pc:~/Desktop$ g++ temp1.cpp
kabil@Kabil-pc:~/Desktop$ ./a.out
Enter the number of Cities: 10
Enter the names of Cities:
Enter name for City 1 : Chennai
Enter name for City 2 : Coimbatore
Enter name for City 3 : Madurai
Enter name for City 4 : Tiruchirappalli
Enter name for City 5 : Salem
Enter name for City 6 : Tirunelveli
Enter name for City 7 : Erode
Enter name for City 8 : Vellore
Enter name for City 9 : Thoothukudi
Enter name for City 10 : Thanjavur
Enter the number of routes: 25
Enter the routes and distance between them in kilometers:
Enter route 1 source city, destination city, and distance: Chennai Coimbatore 500
Enter route 2 source city, destination city, and distance: Chennai Madurai 450
Enter route 3 source city, destination city, and distance: Chennai Tiruchirappalli 350
Enter route 4 source city, destination city, and distance: Chennai Salem 350
Enter route 5 source city, destination city, and distance: Chennai Tirunelveli 650
Enter route 6 source city, destination city, and distance: Chennai Erode 400
Enter route 7 source city, destination city, and distance: Chennai Vellore 140
Enter route 8 source city, destination city, and distance: Chennai Thoothukudi 600
Enter route 9 source city, destination city, and distance: Chennai Thanjavur 350
Enter route 10 source city, destination city, and distance: Coimbatore Madurai 220
Enter route 11 source city, destination city, and distance: Coimbatore Salem 150
Enter route 12 source city, destination city, and distance: Coimbatore Tirunelveli 450
Enter route 13 source city, destination city, and distance: Coimbatore Erode 90
Enter route 14 source city, destination city, and distance: Coimbatore Vellore 400
Enter route 15 source city, destination city, and distance: Madurai Tiruchirappalli 140
Enter route 16 source city, destination city, and distance: Madurai Salem 280
Enter route 17 source city, destination city, and distance: Madurai Tirunelveli 150
Enter route 18 source city, destination city, and distance: Madurai Erode 290
Enter route 19 source city, destination city, and distance: Madurai Vellore 400
Enter route 20 source city, destination city, and distance: Tiruchirappalli Vellore 300
Enter route 21 source city, destination city, and distance: Tiruchirappalli Salem 150
Enter route 22 source city, destination city, and distance: Tiruchirappalli Erode 200
Enter route 23 source city, destination city, and distance: Tiruchirappalli Thoothukudi 550
Enter route 24 source city, destination city, and distance: Salem Erode 70
Enter route 25 source city, destination city, and distance: Thoothukudi Thanjavur 180
Graph visualization of the input graph generated: input_graph.png
Graph visualization of the minimum spanning tree generated: minimum_spanning_tree.png
kabil@Kabil-pc:~/Desktop$

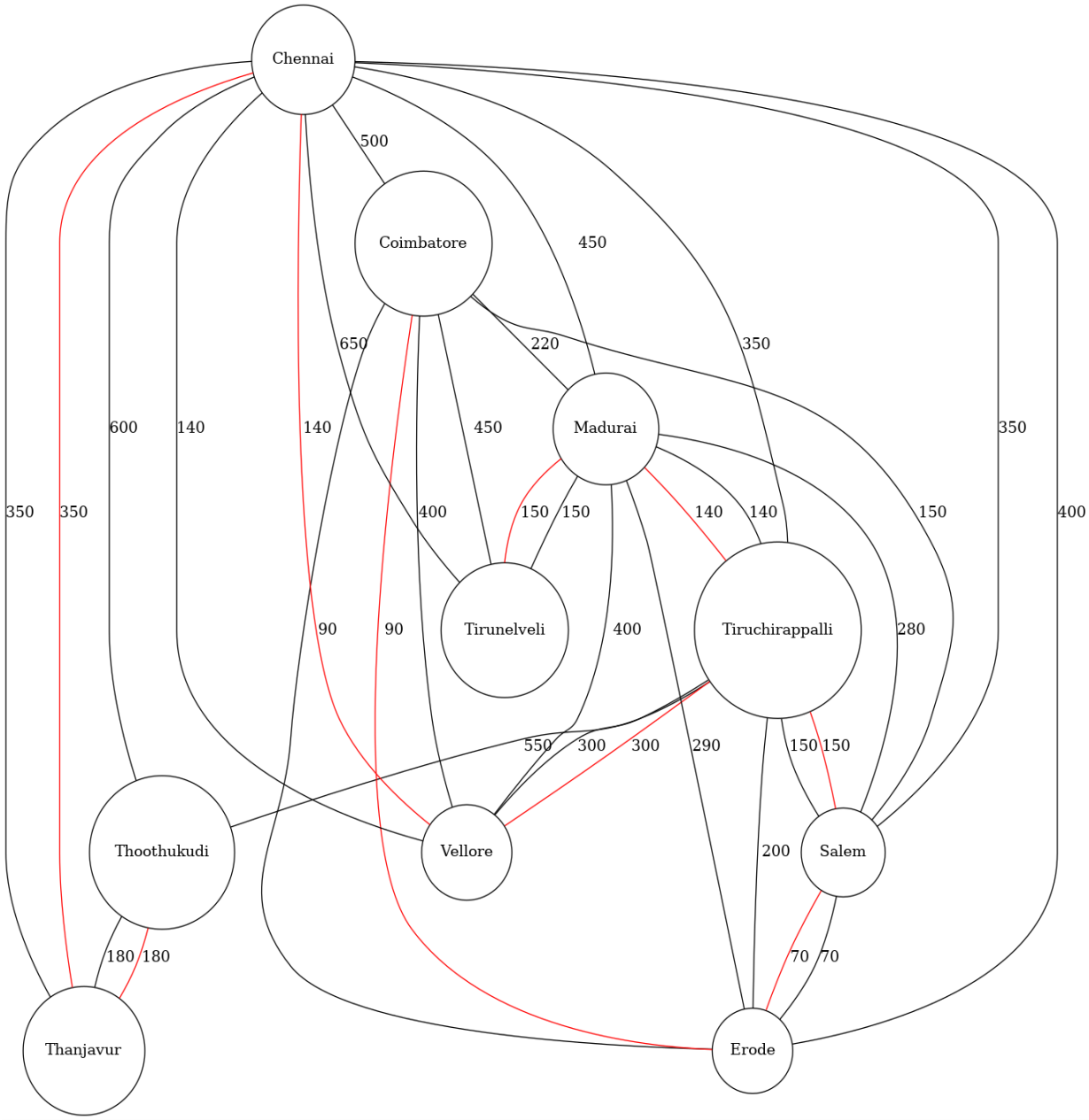
# OUTPUT:

My application code will create a graph using graphix package from the user input as input_graph.png and it will show the final output in minimum_spanning_tree.png as red line




Input_graph.png

Here red line shows the actual shortest path, since we need to go along the red path to cover all cities in possible  shortest distance




Minimum_spanning_tree.png

Here red line shows the shortest path between the cities