项目说明文档

离散数学课程项目

——最小生成树

作 者 姓 名： 陈奕名

学 号： 2351883

指 导 教 师： 李冰

学院、 专业： 软件学院 软件工程

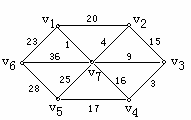
同济大学

Tongji University

# 1 项目分析

## 1.1 项目要求

如下图所示的赋权图表示某七个城市，预先计算出它们之间的一些直接通信道路造价（单位：万元），试给出一个设计方案，使得各城市之间既能够保持通信，又使得总造价最小，并计算其最小值。



# 2 项目设计

## 2.1 数据结构设计

1.Edge 结构体：表示图中的每一条边，包含两个端点 u, v 和权值 weight，并提供了按权值升序排列的比较函数。

2.并查集（Union-Find）：

parent[] 数组：存储每个节点的父节点，维护节点的集合关系。

rank[] 数组：用于按秩合并优化，保持树的平衡，减少查询时间。

3.边的存储：使用 vector<Edge> 存储图中的所有边，便于对边进行排序，并依次选择权值最小的边。

4.最小生成树（MST）：存储被选中的边，表示最小生成树，最终输出这组边以及最小生成树的总权值。

## 2.2 算法设计

1.Kruskal 算法 (Kruskal) 实现细节：

使用并查集来维护顶点的连接信息。

将所有边按权重排序。

依次遍历排序后的边数组，对于每条边，使用find检查判断是否会形成环。

如果没有环，选中该边，并使用unionSets合并集合，累加当前边的权值到总耗费。

继续处理直到所有边都被检查。

2.并查集操作： -

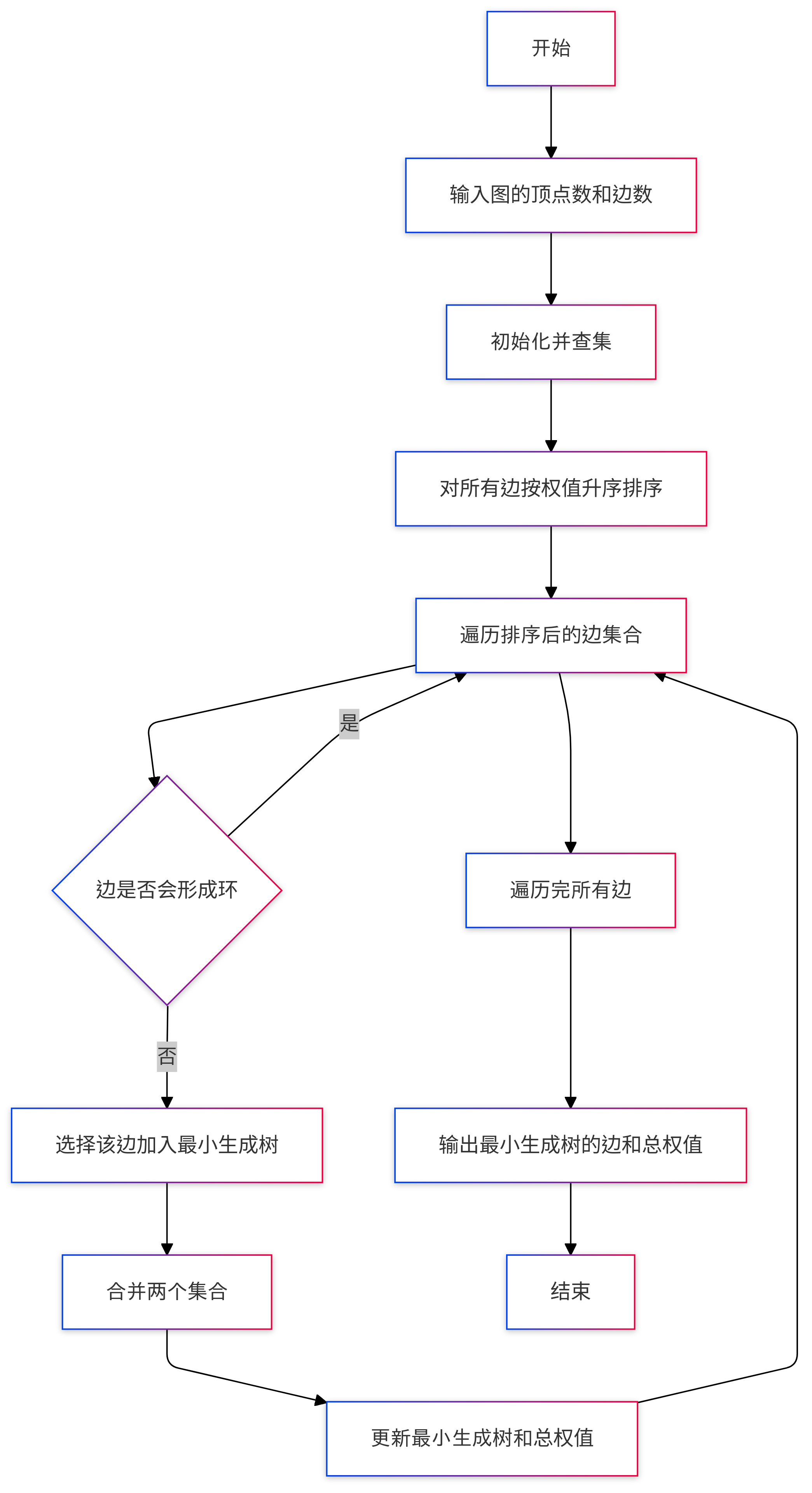
find: 查找集合的根节点，使用路径压缩优化。

unionSets: 合并两个集合，按秩合并优化。

# 3 主要功能实现

## 3.1 程序总流程

### 3.1.1 流程图



### 3.1.2 程序总流程

**1. 程序启动和图的初始化**

-程序启动：当程序启动时，执行 main 函数，开始整个流程。

-输入顶点数和边数：程序首先要求用户输入图的顶点数和边数。用户输入的数值用于确定图的大小以及需要处理的边的数量。

-初始化边数组：接下来，程序创建一个 vector<Edge> 类型的边数组，用于存储图中的所有边。每个边包含三个信息：起始顶点、结束顶点和权重。

-读取边的信息：程序要求用户为每条边输入起始顶点、结束顶点和权重，然后将这些边的信息存入 edges 数组中。

**2. 执行最小生成树算法**

-调用 Kruskal 方法：在图初始化完成后，程序进入最小生成树的计算部分，调用 kruskal 方法来计算图的最小生成树。

-初始化并查集：kruskal 方法首先初始化并查集（Union-Find）。并查集用于判断图中的节点是否属于同一集合，用于防止生成环。

-边的排序：然后，程序对所有边按照权重进行升序排序，准备按权重选择边加入最小生成树。

-遍历边集合：程序接下来遍历排序后的边集合，逐一检查每条边，判断它的两个端点是否已经在同一个集合中。如果不在同一个集合中，说明这条边不会形成环，程序将这条边加入最小生成树。

-输出最小生成树的边和总耗费：最后，程序遍历 mst 数组，输出所有构成最小生成树的边。同时输出最小生成树的总权重。

## 3.2 具体代码实现

### 3.2.1 Kruskal最小生成树

void kruskal(int n, int m, vector<Edge> &edges) {

vector<Edge> mst; // 存储最小生成树的边

int totalCost = 0; // 存储最小生成树的总耗费

// 初始化并查集

int parent[n + 1], rank[n + 1];

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

parent[i] = i;

rank[i] = 0;

}

// 按边的权值升序排序

sort(edges.begin(), edges.end());

// Kruskal 算法主体

for (const Edge &e : edges) {

int u = e.u, v = e.v;

// 判断是否会形成环

if (find(parent, u) != find(parent, v)) {

// 如果没有环，选中该边，并合并集合

mst.push\_back(e);

totalCost += e.weight; // 累加当前边的权值到总耗费

unionSets(parent, rank, u, v);

}

}

// 输出最小生成树的边

for (const Edge &e : mst) {

cout <<"最小耗费是: "<< e.u << "和" << e.v << endl;

}

// 输出最小生成树的总耗费

cout << "最小生成树的总耗费为: " << totalCost << endl;

}

### 3.2.2 并查集操作

// 查找集合的根节点，使用路径压缩优化

int find(int parent[], int i) {

if (parent[i] == i)

return i;

return parent[i] = find(parent, parent[i]);

}

// 合并两个集合，按秩合并优化

void unionSets(int parent[], int rank[], int u, int v) {

int rootU = find(parent, u);

int rootV = find(parent, v);

if (rootU != rootV) {

if (rank[rootU] > rank[rootV]) {

parent[rootV] = rootU;

} else if (rank[rootU] < rank[rootV]) {

parent[rootU] = rootV;

} else {

parent[rootV] = rootU;

rank[rootU]++;

}

}

}

### 3.2.3 主函数

int main() {

int n, m; // 顶点数和边数

cout << "请输入所求图的顶点数目和边的数目(以空格分隔): ";

cin >> n >> m;

vector<Edge> edges; // 存储图的边

cout << "请输入两条边的节点信号以及它们的权值(以空格分隔各个数): "<<endl;

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

int u, v, w;

cin >> u >> v >> w;

edges.push\_back({u, v, w}); // 添加边

}

// 调用 Kruskal 算法求最小生成树

kruskal(n, m, edges);

return 0;

}

# 4 测试

**测试用例**：

7 12

1 2 20

1 6 23

1 7 1

2 7 4

2 3 15

6 7 36

3 7 9

5 6 28

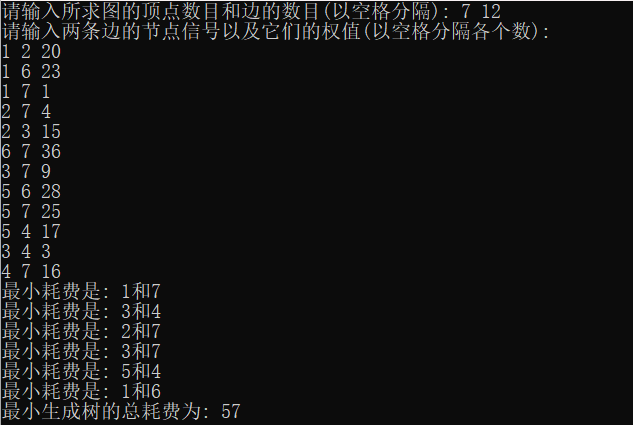
5 7 25

5 4 17

3 4 3

4 7 16

**实验结果:**



# 5 心得体会

通过这次实现 Kruskal 算法的过程，我对最小生成树算法和并查集有了更深入的理解。首先，Kruskal 算法的核心思想是贪心策略，它通过选择权值最小的边，逐步构建最小生成树，这让我对贪心算法的应用场景和优缺点有了更清晰的认识。实现过程中，我深入理解了如何使用并查集来高效地解决集合合并和环路检测问题，这不仅提升了我的算法设计能力，也让我感受到并查集在解决图的连通性问题中的高效性。

在具体实现时，我特别关注了并查集的优化技巧，如路径压缩和按秩合并，这使得查找和合并操作更加高效，能够在更大的数据集上运行得更快。此外，Kruskal 算法的边排序和遍历步骤让我更好地理解了如何处理带权图中的边，并学会了如何利用排序来简化问题。

通过这次实践，我不仅增强了对图论和算法的理论知识，还提高了自己处理复杂数据结构和算法的能力。编写代码的过程中，我更加注重了算法的时间复杂度和空间复杂度，力求在保证正确性的前提下，优化代码的执行效率。

总的来说，这次实现不仅锻炼了我的编程技巧，尤其是数据结构和算法的实现能力，也让我在解决实际问题时更加注重算法的选择和优化，进一步提升了我的问题分析和解决能力。