**项目说明文档**

**离散数学课程实验**

**——最小生成树**

作 者 姓 名 刘淑仪

学 号 2251730

指 导 教 师 唐剑锋

学 院 专 业 软件学院 软件工程



二〇二三 年 十 月 二十五 日

目录

1 项目分析 1

1.1 实验目的分析 1

1.2 实验内容概述 1

2 项目设计与实现 2

2.1 实验原理 2

2.1.1 题目涉及知识 2

2.1.2 具体实验原理 2

2.2 实验过程设计 3

2.2.1 实验思路简述 3

2.2.2 实验设计流程图 4

2.3 项目具体实现 5

3 项目实现与测试 5

4 源代码展示 7

5 集成开发环境与编译运行环境 11

# 项目分析

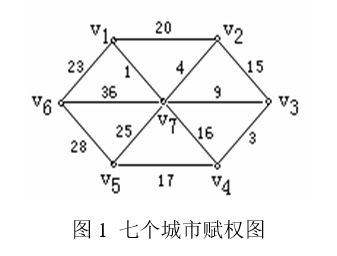
## 实验目的分析

该实验课程的目标是培养学生在命题逻辑领域的技能，包括联接词的运用、真值表的构建以及主范式的应用，以便进一步将这些技能应用于解决实际问题。通过实验课程的学习，学生将提升其编写实验报告和总结实验结果的能力，同时培养他们的程序设计思维，使他们能够独立完成简单算法的设计和分析。

## 实验内容概述

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。

用C++实现，将v1-v7七个结点连通，并且各连通线的相加之和最小。即将七个城市作为图的结点，他们之间的距离作为两点之间的权值所形成的图，求七个结点所形成的最小生成树。



# 项目设计与实现

## 实验原理

### 题目涉及知识

1.最小生成树

给定一个有 n 个结点的连通图 G=(V,E)，其中 V 是结点集合，E 是边集合，每条边 (u,v) 都有一个权重 w(u,v)。图 G 的生成树是一个极小连通子图 T=(V,F)，其中 F 是 E 的子集，且包含了 n−1 条边，使得图 T 保持连通。图 G 的最小生成树是权重之和 w(T)=∑(u,v)∈F​w(u,v) 最小的生成树。最小生成树可以用 Kruskal 算法或 Prim 算法求出。最小生成树也称为最小权重生成树。

2.并查集

并查集是一种数据结构，用于管理一组不相交的集合。每个集合中的元素都有一个唯一的标识，称为代表元。并查集支持两种操作：查找和合并。查找操作用于查询一个元素所属的集合，即返回该元素的代表元。合并操作用于将两个集合合并为一个集合，即将一个集合的代表元修改为另一个集合的代表元。

3.最小堆

堆是一种特殊的完全二叉树，满足堆序性质：对于任意的非叶子结点，其数据值不大于（或不小于）其左右子结点的数据值。最大堆和最小堆是堆的两种常见形式。最小堆是一种满足最小堆序性质的堆，即根结点的数据值是所有堆结点数据值中最小的。最小堆常用于实现优先队列等数据结构。

### 具体实验原理

Ⅰ、从键盘读取无向图的顶点数 n 和边数 m，如果 n 和 m 均为 0，则结束程序；否则，继续执行下一步。

Ⅱ、从键盘读取每条边的起点、终点和权重，将无向图的信息存储在邻接表中。

Ⅲ、执行 Kruskal 算法：

① 将邻接表中的边按照权重从小到大排序，构建最小堆。

② 从最小堆中依次取出最小的边，如果该边与已经加入最小生成树的边不构成回路，则将该边加入最小生成树和并查集。

③ 当最小生成树中的边数达到 n−1 时，即已经将 n 个顶点全部连通时，停止算法，输出最小生成树，并返回第Ⅰ步。

## 实验过程设计

### 2.2.1 实验思路简述

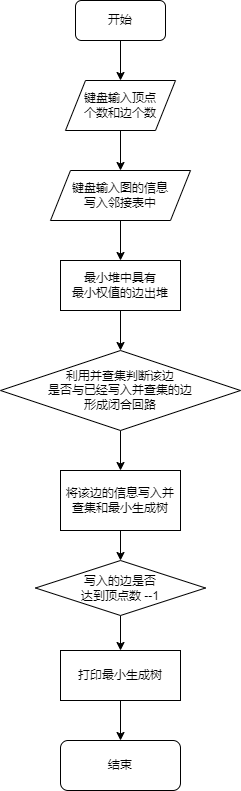
实现Kruskal 算法用于求解一个无向图的最小生成树1。最小生成树是一个包含了图中所有顶点的连通子图，且其边的权重之和最小2。Kruskal 算法的基本思想是从小到大选择权重最小的边，如果这条边不会和已经选择的边构成回路，就将它加入最小生成树中，直到最小生成树包含了所有的顶点3。

代码中定义了一个结构体 Edge，用于存储边的起点、终点和权重。代码中还使用了一个向量 parent，用于实现并查集4的功能。并查集是一种数据结构，用于管理一组不相交的集合，支持查找和合并两个集合的操作5。代码中的 findSet 函数用于查找一个顶点所属的集合的代表元，即根节点。代码中还使用了路径压缩的技巧，即在查找的过程中，将沿途的顶点的父节点都指向根节点，以提高查找的效率。

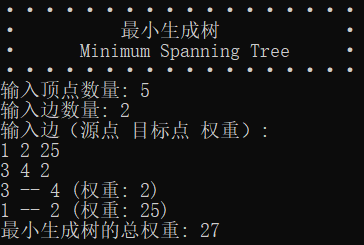
代码中的 KruskalMST 函数是 Kruskal 算法的主要实现部分。它接受一个边的向量、顶点数和边数作为参数。首先，它将每个顶点的父节点初始化为自己，表示每个顶点都是一个单独的集合。然后，它将边的向量按照权重从小到大排序，使用了 C++ 的 sort 函数和一个匿名比较函数。接下来，它定义了两个变量，totalWeight 和 edgeCount，分别用于记录最小生成树的总权重和边数。然后，它遍历排序后的边的向量，对于每一条边，它先用 findSet 函数查找它的两个顶点所属的集合的根节点，如果根节点不同，说明这条边不会和已经选择的边构成回路，就将它加入最小生成树中，并输出它的信息。同时，更新最小生成树的总权重和边数，并用 union 函数将两个集合合并为一个。如果最小生成树的边数达到了顶点数减一，说明已经找到了最小生成树，就停止遍历，并输出最小生成树的总权重。

代码中的 menu 函数用于输出一些菜单信息，提示用户输入顶点数和边数，以及每条边的信息。代码中的 main 函数是程序的入口，它首先调用 menu 函数，然后从标准输入读取顶点数和边数，以及每条边的信息，并存储在一个边的向量中。最后，它调用 KruskalMST 函数，传入边的向量、顶点数和边数，求解最小生成树。

### 2.2.2 实验设计流程图

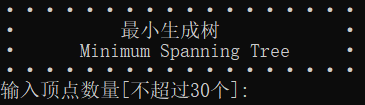


## 项目具体实现



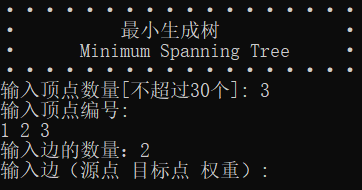
# 项目实现与测试

1.进入界面



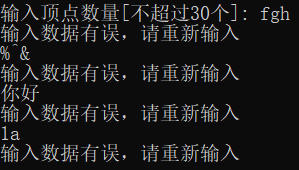
2.输入边和顶点

2.1.正常输入



2.2.错误输入

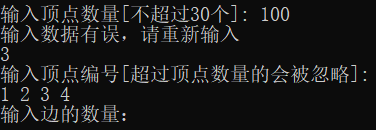
2.2.1.非法输入



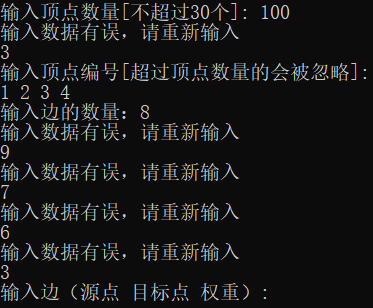
2.2.2.顶点超过最大个数



2.2.3.顶点编号个数大于总顶点数

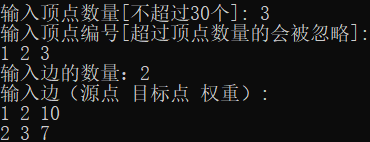


2.2.4.边超过最大边数量

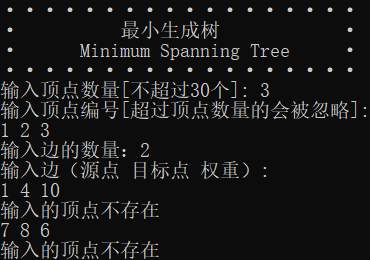


3.输入边及其权重

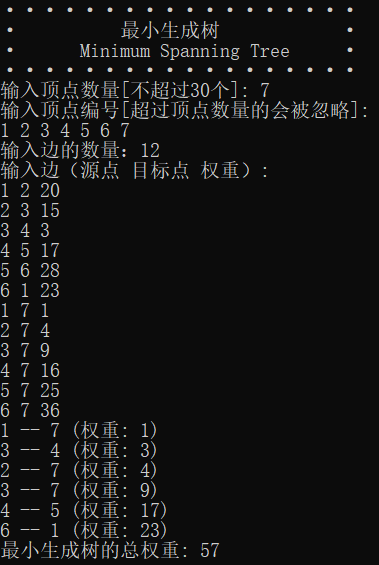
3.1.正常输入



3.2.输入的顶点不存在



4.输出边及其权重



# 源代码展示

#include <iostream>

#include <vector>

#include <algorithm>

#include <set>

const int MAX\_VERTICES = 5005;  // 定义最大顶点数

const int MAX\_EDGES = 200005;   // 定义最大边数

struct Edge {

    int src, dest, weight;       // 边结构，包括源点、目标点和权重

};

// 并查集类用于辅助Kruskal算法

class UnionFind {

public:

    UnionFind(int size) : parent(size + 1) {

        for (int i = 1; i <= size; ++i) parent[i] = i; // 初始化父节点

    }

    int find(int vertex) {

        // 查找根节点，同时进行路径压缩

        return parent[vertex] == vertex ? vertex : (parent[vertex] = find(parent[vertex]));

    }

    void unite(int x, int y) {

        // 合并两个集合

        parent[find(x)] = find(y);

    }

private:

    std::vector<int> parent; // 存储父节点信息

};

// Kruskal最小生成树算法类

class KruskalMST {

public:

    KruskalMST(int vertices) : numVertices(vertices), uf(vertices) {}

    void addEdge(int src, int dest, int weight) {

        // 添加边

        edgeList.push\_back({ src, dest, weight });

    }

    void constructMST() {

        // 构建最小生成树

        sortEdges(); // 首先对边进行排序

        int totalWeight = 0; // 记录总权重

        for (const auto& edge : edgeList) {

            // 遍历每条边，检查是否形成环

            if (uf.find(edge.src) != uf.find(edge.dest)) {

                // 如果不形成环，则添加这条边

                std::cout << edge.src << " -- " << edge.dest << " (权重: " << edge.weight << ")\n";

                totalWeight += edge.weight;

                uf.unite(edge.src, edge.dest);

            }

        }

        std::cout << "最小生成树的总权重: " << totalWeight << std::endl;

    }

private:

    int numVertices;

    std::vector<Edge> edgeList;

    UnionFind uf; // 并查集实例

    void sortEdges() {

        // 边按权重排序的函数

        std::sort(edgeList.begin(), edgeList.end(), [](const Edge& a, const Edge& b) {

            return a.weight < b.weight;

            });

    }

};

class InputHandler {

public:

    static int inputNumber(bool isVertices, int numVertices = 0) {

        // 输入处理函数，确保输入有效

        int maxSize = isVertices ? 30 : numVertices \* (numVertices - 1) / 2;

        int inputNum;

        while (true) {

            std::cin >> inputNum;

            if (std::cin.fail() || inputNum < 0 || inputNum > maxSize) {

                // 输入验证

                std::cin.clear();

                std::cin.ignore(65536, '\n');

                std::cout << "输入数据有误，请重新输入\n";

            }

            else {

                if (std::cin.get() == '\n')

                    break;

                else {

                    std::cin.clear();

                    std::cin.ignore(65536, '\n');

                    std::cout << "输入数据有误，请重新输入\n";

                }

            }

        }

        return inputNum;

    }

    static void validateVertices(int vertex, const std::set<int>& validVertices) {

        // 验证顶点是否有效

        if (validVertices.find(vertex) == validVertices.end()) {

            throw std::invalid\_argument("输入的顶点不存在");

        }

    }

};

void menu() {

    // 程序菜单

    std::cout << "··················\n";

    std::cout << "·          最小生成树            ·\n";

    std::cout << "·      Minimum Spanning Tree     ·\n";

    std::cout << "··················\n";

}

int main() {

    menu();

    std::cout << "输入顶点数量[不超过30个]: ";

    int numVertices = InputHandler::inputNumber(true);

    std::set<int> validVertices;

    std::cout << "输入顶点编号[超过顶点数量的会被忽略]:\n";

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

        int vertex;

        std::cin >> vertex;

        validVertices.insert(vertex);

    }

    std::cin.ignore(65536, '\n');

    std::cout << "输入边的数量：";

    int numEdges = InputHandler::inputNumber(false, numVertices);

    KruskalMST mst(numVertices);

    std::cout << "输入边（源点 目标点 权重）:\n";

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

        int src, dest, weight;

        std::cin >> src >> dest >> weight;

        try {

            InputHandler::validateVertices(src, validVertices);

            InputHandler::validateVertices(dest, validVertices);

            mst.addEdge(src, dest, weight);

        }

        catch (const std::invalid\_argument& e) {

            std::cout << e.what() << std::endl;

            --i; // 如果输入无效，重新输入这条边

        }

    }

    mst.constructMST(); // 构建最小生成树

    return 0;

}

# 集成开发环境与编译运行环境

Windows系统：Windows 11 x64

Windows集成开发环境：Microsoft Visual Studio 2022 (Debug模式)

Windows编译运行环境：本项目适用于x86架构和x64架构