项目说明文档

数据结构课程设计

——电网建设造价模拟系统



作 者 姓 名： 刘相成

学 号： 2452207

指 导 教 师： 张颖

学院、 专业： 计算机科学与技术学院 软件工程

同济大学

Tongji University

# 1 项目分析

## 背景分析

## 在城市基础设施建设中，电网铺设是一项关键工程，其核心需求是在多个小区（或节点）之间构建连通的电网系统，同时使总工程造价最低。这一问题本质上是加权连通图的最小生成树（MST）问题 —— 在包含 n 个顶点的连通图中，选择 n-1 条边连接所有顶点，且使边的总权重（对应工程造价）最小。最小生成树算法在网络设计、交通规划、通信布线等领域具有广泛应用，本项目通过实现 Prim 和 Kruskal 两种经典算法，模拟电网建设的造价优化过程，为实际工程决策提供参考。

## 1.2 功能分析

1.2.1功能要求

支持创建电网顶点（小区），顶点数量范围为 2-100 个，顶点名称唯一不重复。

支持添加电网边（线路），输入顶点对及对应权重（造价），支持边的更新和重复检测，输入 “? ? 0” 结束边的添加。

支持两种最小生成树构造算法：Prim 算法和 Kruskal 算法，生成过程中计算算法耗时。

支持显示最小生成树的边信息及总造价。

支持从文件读取顶点和边数据，提高输入效率。

完善的输入验证机制，处理非法输入（非数字、超出范围、重复顶点、不存在的顶点等）。

1.2.2输入要求

顶点输入：输入顶点个数（2-100 的整数），随后依次输入每个顶点的唯一名称（字符串）。

边输入：输入格式为 “顶点 1 顶点 2 权重”，权重为正整数；输入 “? ? 0” 终止边的添加。

文件输入：文件需按指定格式存储（首行顶点数，随后是顶点名称，再之后是若干行边数据），文件名为 “input.txt”。

算法选择输入：输入对应操作符（A-F）选择功能，支持大小写兼容。

1.2.3输出要求

操作反馈：顶点创建完成、边添加完成、文件读取成功等提示信息。

算法运行结果：显示所选算法（Prim/Kruskal）的运行耗时。

最小生成树信息：显示每条边的起点、权重、终点，按 3 条边为一行排版，最后输出总造价。

错误提示：输入无效、顶点不存在、边数不足、图不连通等错误信息。

1.2.4项目实例



# 2 项目设计

## 2.1 数据结构设计

**2.1.1 边（Edge）**

边是构成图的基本元素之一，在代码中通过edge结构体实现，包含起点（start）、终点（end）和权重（weight）三个属性。其中，权重表示该边的成本或长度。

顶点连接表示：在电网造价模拟中，边用于表示两个顶点之间的连接关系，start和end字段分别记录边的起点和终点名称，以明确顶点间的关联。

权重存储：weight字段用于存储边的成本或长度信息，为最小生成树算法（Prim 和 Kruskal）提供计算依据。

排序需求：Kruskal 算法中需按权重对边进行升序排序，代码中通过grid\_sys类的sort\_e方法（冒泡排序）实现边的排序，确保算法能按权重顺序选择边。

**2.1.2 点（Vertex）**

点代表图中的顶点，在代码中通过v\_name数组存储顶点名称，并在find\_set类中通过parent数组维护顶点在并查集中的祖先关系。

顶点标识：v\_name数组存储每个顶点的唯一名称，通过find\_index方法可根据名称查找顶点的索引，实现顶点的唯一标识与快速定位。

并查集支持：Kruskal 算法中使用find\_set类管理顶点的祖先关系，parent数组记录每个顶点的父节点，通过find方法查找顶点的根节点，通过unite方法合并两个顶点所在的集合，从而检测环并维护连通分量。

**2.1.3 图（Graph）**

图由顶点和边构成，在代码中通过grid\_sys类实现，包含顶点数（num\_v）、边数（num\_e）、顶点数组（v\_name）、边数组（store\_e）、邻接矩阵（matrix），以及用于存储最小生成树结果的边数组（mst\_final）。

顶点和边的存储：v\_name数组存储所有顶点名称，store\_e数组存储所有边的信息；邻接矩阵matrix用于 Prim 算法，快速访问顶点间的边权重。

最小生成树：mst\_final数组存储最小生成树的边，mst\_e\_cnt记录边的数量，is\_mst标识最小生成树是否已生成，支持通过show\_mst方法展示结果。

排序与算法支持：图中通过sort\_e方法对边进行排序（支持 Kruskal 算法）；同时实现了 Prim 算法（基于邻接矩阵）和 Kruskal 算法（基于边数组和并查集），分别通过mst\_prim和mst\_kruskal方法执行，满足不同场景下最小生成树的求解需求。

## 2.2 类结构设计

**2.2.1 find\_set 类**

提供并查集的核心操作，支持 Kruskal 算法中的环检测和集合合并，具体方法如下：

| 方法名称 | 功能描述 |
| --- | --- |
| find\_set | 构造函数，初始化 n 个独立元素，每个元素的父节点为自身 |
| find | 查找元素 i 的根节点，路径压缩优化（使后续查找更快） |
| unite | 合并 i 和 j 所在的集合，若两者根节点不同，则将一个根节点指向另一个 |

**2.2.2 grid\_sys 类**

封装电网系统的所有数据和操作，是项目的核心类，具体方法如下：

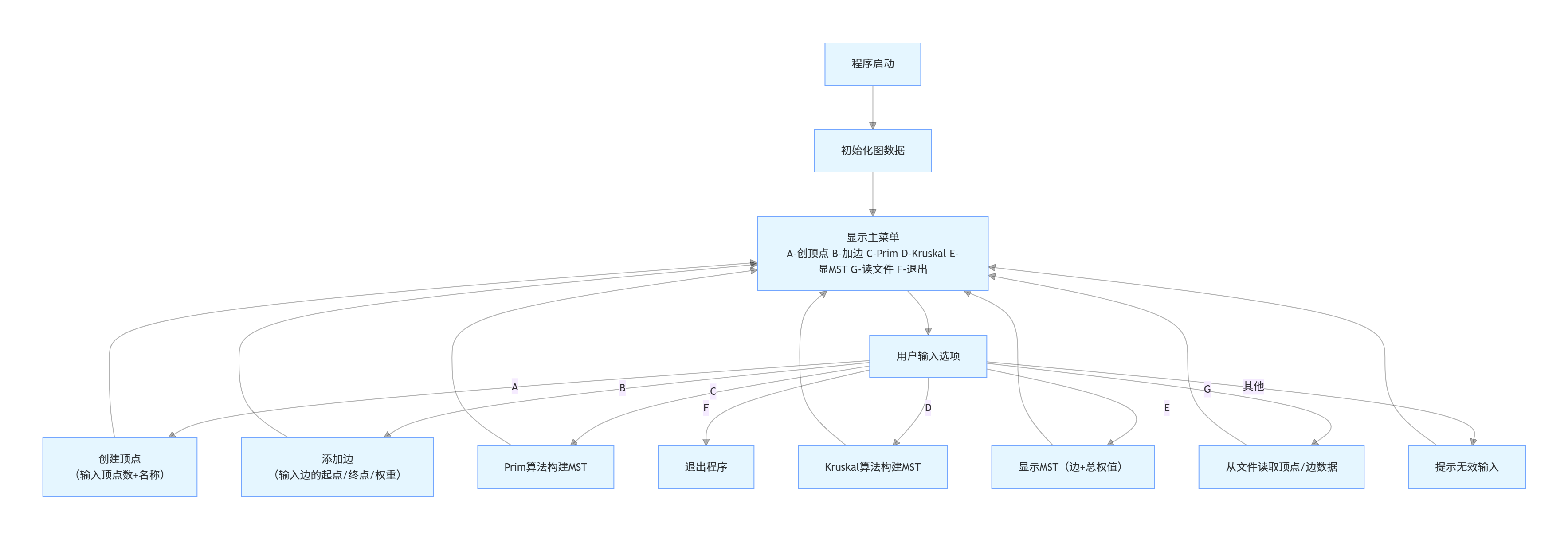
| 方法名称 | 功能描述 |
| --- | --- |
| grid\_sys | 构造函数，初始化顶点数、边数为 0，邻接矩阵填充为无穷大（inf），MST 标记为未生成 |
| find\_index | 查找名称为 name 的顶点索引，找到返回索引，未找到返回 - 1 |
| sort\_e | 对 store\_e 数组中的边按权重升序排序（冒泡排序） |
| reset\_graph\_data | 重置顶点数、边数、MST 边数为 0，邻接矩阵填充为 inf，MST 标记为未生成 |
| create\_v | 接收用户输入，创建指定数量的顶点，验证输入合法性（数量范围、名称唯一） |
| add\_e | 接收用户输入，添加边数据，验证顶点存在性、权重合法性，支持边更新和终止输入 |
| mst\_prim | 用 Prim 算法构造最小生成树，需指定起始顶点，计算算法耗时，处理图不连通情况 |
| mst\_kruskal | 用 Kruskal 算法构造最小生成树，先排序边，再通过并查集选边，计算算法耗时 |
| show\_mst | 显示最小生成树的边信息和总造价，若未生成 MST 则提示 |
| read\_from\_file | 从 input.txt 文件读取顶点和边数据，自动初始化电网，处理文件打开失败情况 |
| menu | 显示系统功能菜单，提示用户操作选择 |

## 2.3 成员与函数设计

/ 并查集类Kruskal  
class find\_set {  
private:  
    int parent[max\_v]; //存储父节点的数组  
    int count; //元素数量  
public:  
    find\_set(int n);// 构造函数  
    int find(int i);// 查找元素的根节点  
    void unite(int i, int j);// 合并两个集合  
};  
​  
// grid\_sys 类  
class grid\_sys {  
private:  
    char v\_name[max\_v][50]; // 存储顶点名称  
    int matrix[max\_v][max\_v]; // 邻接矩阵  
    int num\_v; // 当前顶点数  
    edge store\_e[max\_e]; // 存储所有边的数组  
    int num\_e; // 当前边数  
    edge mst\_final[max\_v]; // 存储MST结果的数组  
    int mst\_e\_cnt; // MST边的数量  
    bool is\_mst; // 标记MST是否生成  
    int find\_index(const char\* name);  // 查找顶点下标  
    void sort\_e();// 边排序，冒泡排序  
    void reset\_graph\_data(); // 重置图数据（内部辅助函数）  
​  
public:  
    grid\_sys();// 构造函数  
    void create\_v(); // 创建顶点  
    void add\_e();// 添加边  
    void mst\_prim(); // 构造最小生成树Prim  
    void mst\_kruskal(); // 构造最小生成树Kruskal  
    void show\_mst(); // 显示最小生成树  
    void read\_from\_file(); // 从文件读取数据  
    void menu();// 显示菜单  
};

## 2.4 代码总设计（流程图）

简化版



# 3 项目实现

## 3.1 初始化顶点功能的实现

### 3.1.1 初始化顶点功能原理与流程图

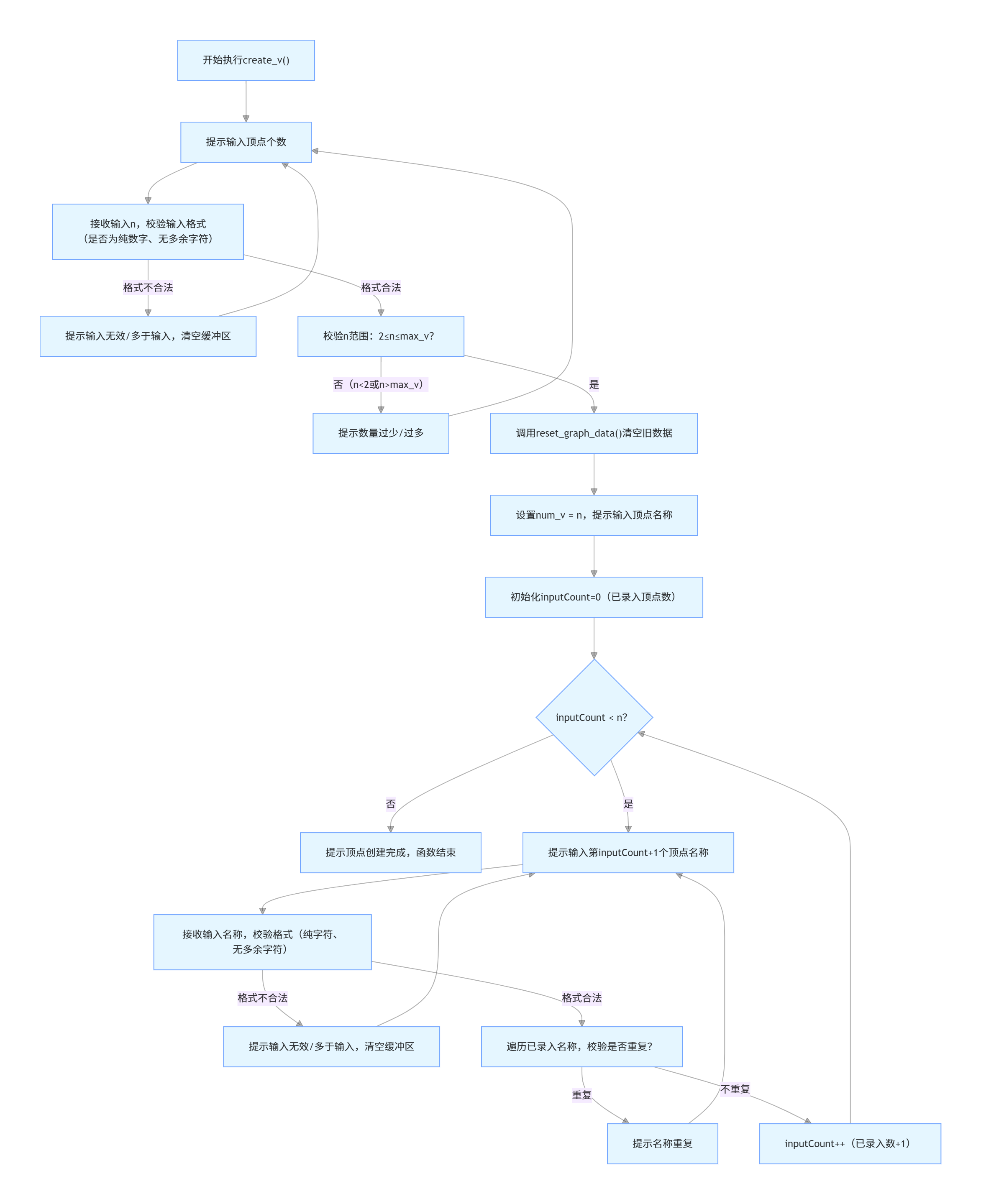
create\_v() 是 grid\_sys 类中用于创建图顶点的核心函数，核心目标是安全、合法地录入顶点数量和名称，确保顶点数据满足程序运行的基础规则，具体逻辑拆解如下：

1. 顶点数量校验（核心规则：2 ≤ 数量 ≤ 最大支持数max\_v）。首先循环接收用户输入的顶点数 n，先做输入格式校验：再做数值范围校验：所有校验通过后，跳出循环，确定最终顶点数num\_v = n。

2. 重置图数据。调用 reset\_graph\_data() 清空之前的顶点、边、邻接矩阵等数据，避免旧数据干扰新顶点的创建。

3. 顶点名称录入（名称唯一、输入格式合法）。循环 n 次接收每个顶点的名称，每次输入后先做格式校验（非合法输入 / 多余字符均重新输入）；

4. 最终反馈。所有顶点录入完成后，输出 “顶点创建完成”，结束函数。



### 3.1.2 初始化顶点功能核心代码

//创建顶点  
void grid\_sys::create\_v() {  
    cout << "请输入顶点的个数: ";  
    int n;  
    while (1) {  
        if (!(cin >> n)) {  
            cin.clear();  
            cin.ignore(INT\_MAX, '\n');  
            cout << "输入无效，请重新输入" << endl;  
            continue;  
       }  
        char next = cin.peek();  
        while (next == ' ') {  
            cin.get();  
            next = cin.peek();  
       }  
        if (next != '\n' && next != EOF) {  
            cout << "多于输入，重新输入" << endl;  
            cin.clear();  
            cin.ignore(INT\_MAX, '\n');  
            continue;  
       }  
        if (n < 2) {  
            cout << "顶点数量过少，至少为2" << endl;  
            continue;  
       }  
        if (n > max\_v) {  
            cout << "顶点数量过多，最大支持 " << max\_v << " 个" << endl;  
            continue;  
       }  
        break;  
   }  
    reset\_graph\_data();  
    num\_v = n;  
​  
    cout << "请依次输入各顶点的名称：" << endl;  
    int inputCount = 0;  
    while (inputCount < n) {  
        cout << "顶点 " << (inputCount + 1) << ": ";  
        if (!(cin >> v\_name[inputCount])) {  
            cin.clear();  
            cin.ignore(INT\_MAX, '\n');  
            cout << "输入无效，请重新输入" << endl;  
            continue;  
       }  
        char next = cin.peek();  
        while (next == ' ') {  
            cin.get();  
            next = cin.peek();  
       }  
        if (next != '\n' && next != EOF) {  
            cout << "多于输入，重新输入" << endl;  
            cin.clear();  
            cin.ignore(INT\_MAX, '\n');  
            continue;  
       }  
        bool duplicate = false;  
        for (int i = 0; i < inputCount; i++) {  
            if (strcmp(v\_name[i], v\_name[inputCount]) == 0) {  
                cout << "顶点名称重复，请重新输入" << endl;  
                duplicate = true;  
                break;  
           }  
       }  
        if (!duplicate) {  
            inputCount++;  
       }  
   }  
    cout << "顶点创建完成" << endl;  
}

## 3.2 prim算法功能的实现

### 3.2.1 prim算法功能原理与流程图

mst\_prim() 是 grid\_sys 类中基于 Prim 算法构建图的最小生成树（MST）的核心函数，Prim 算法核心思想是从指定起始顶点出发，逐步贪心选择连接 “已加入 MST 的顶点集合” 和 “未加入集合” 的最小权值边，最终覆盖所有顶点。具体逻辑拆解如下：

一. 前置合法性校验（略）

二. 初始化

lowcost[]：记录 “未加入 MST 的顶点” 到 “已加入 MST 集合” 的最小权值；

closest[]：记录 “未加入 MST 的顶点” 在 MST 集合中对应的最近顶点；

in\_mst[]：标记顶点是否已加入 MST（初始全为 false）。

以起始顶点为基准初始化：lowcost[i] 赋值为起始顶点到顶点i的邻接矩阵权值，closest[i] 赋值为起始顶点索引，同时将起始顶点标记为 “已加入 MST”（true），并重置 MST 边计数 mst\_e\_cnt=0。

三. Prim 算法核心

循环num\_v-1次（MST 需顶点数-1条边）：

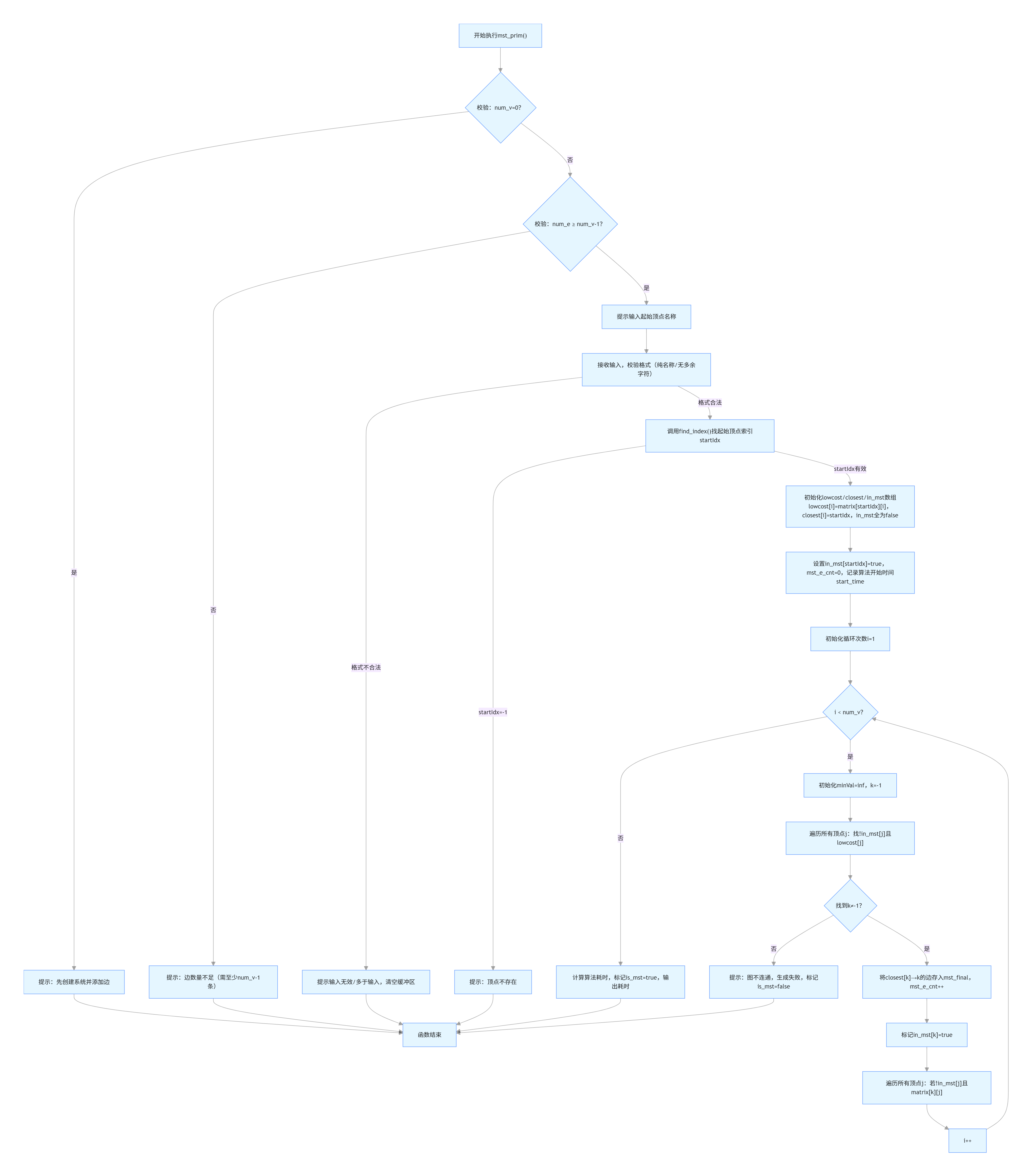
1.找最小权值边：遍历所有未加入 MST 的顶点，找到 lowcost[j] 最小的顶点k（即连接已 / 未加入集合的最小边）；若找不到（k=-1），说明图不连通，标记is\_mst=false并提示 “生成失败”。

2.记录 MST 边：将 “closest[k]（已入集合顶点）→k（新入顶点）” 这条边的起点、终点、权值存入 MST 结果数组 mst\_final，并增加 MST 边计数。

3.更新核心数组：将顶点k标记为 “已加入 MST”，并遍历所有未加入顶点j，若顶点k到j的权值小于当前lowcost[j]，则更新lowcost[j]和closest[j]

四. 收尾处理

统计算法耗时（通过clock()记录开始 / 结束时间，计算并输出耗时）；标记is\_mst=true（MS生成成功），结束函数。



### 3.2.2 prim算法功能核心代码

//构造最小生成树Prim算法  
void grid\_sys::mst\_prim() {  
    if (num\_v == 0) {  
        cout << "请先创建系统并添加边" << endl;  
        return;  
   }  
    if (num\_e < num\_v - 1) {  
        cout << "错误：边数量不足。顶点数(" << num\_v << ")，至少需要 " << (num\_v - 1) << " 条边。" << endl;  
        return;  
   }  
    cout << "请输入起始顶点: ";  
    char startName[50];  
    if (!(cin >> startName)) {  
        cin.clear();  
        cin.ignore(INT\_MAX, '\n');  
        cout << "输入无效" << endl;  
        return;  
   }  
    char next = cin.peek();  
    while (next == ' ') {  
        cin.get();  
        next = cin.peek();  
   }  
    if (next != '\n' && next != EOF) {  
        cout << "多于输入，重新输入" << endl;  
        cin.clear();  
        cin.ignore(INT\_MAX, '\n');  
        return;  
   }  
    int startIdx = find\_index(startName);  
    if (startIdx == -1) {  
        cout << "顶点不存在" << endl;  
        return;  
   }  
    clock\_t start\_time = clock();  
    int lowcost[max\_v];  
    int closest[max\_v];  
    bool in\_mst[max\_v];  
    for (int i = 0; i < num\_v; i++) {  
        lowcost[i] = matrix[startIdx][i];  
        closest[i] = startIdx;  
        in\_mst[i] = false;  
   }  
    lowcost[startIdx] = 0;  
    in\_mst[startIdx] = true;  
    mst\_e\_cnt = 0;  
    cout << "正在生成Prim最小生成树" << endl;  
    for (int i = 1; i < num\_v; i++) {  
        int minVal = inf;  
        int k = -1;  
        for (int j = 0; j < num\_v; j++) {  
            if (!in\_mst[j] && lowcost[j] < minVal) {  
                minVal = lowcost[j];  
                k = j;  
           }  
       }  
        if (k == -1) {  
            cout << "生成失败：图不连通" << endl;  
            is\_mst = false;  
            return;  
       }  
        strcpy(mst\_final[mst\_e\_cnt].start, v\_name[closest[k]]);  
        strcpy(mst\_final[mst\_e\_cnt].end, v\_name[k]);  
        mst\_final[mst\_e\_cnt].weight = minVal;  
        mst\_e\_cnt++;  
        in\_mst[k] = true;  
        for (int j = 0; j < num\_v; j++) {  
            if (!in\_mst[j] && matrix[k][j] < lowcost[j]) {  
                lowcost[j] = matrix[k][j];  
                closest[j] = k;  
           }  
       }  
   }  
    clock\_t end\_time = clock();  
    double elapsed\_time = (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC;  
    is\_mst = true;  
    //show\_mst();  
    cout << "Prim 算法耗费时间: " << fixed << setprecision(6) << elapsed\_time << " 秒" << endl;  
}

## 3.3 kruskal功能的实现

### 3.3.1 kruskal功能原理与流程图

mst\_kruskal()是grid\_sys类中基于 Kruskal 算法构建最小生成树（MST）的核心函数，Kruskal 算法核心思想是按边的权值从小到大排序，依次选择边，若选择的边不会使已选边形成环，则加入 MST，直到选够顶点数-1条边（MST 的边数固定为顶点数 - 1）。具体拆解如下：

一. 前置合法性校验（略）

二. 前置准备

统计算法开始时间（clock()），用于后续计算耗时；

调用sort\_e()对存储所有边的数组 store\_e 按权值从小到大排序（Kruskal 算法的核心前提：贪心选最小权值边）；

初始化并查集对象 uf（大小为顶点数）：并查集是 Kruskal 算法检测 “选边是否形成环” 的核心工具，通过 “查找顶点所属集合”“合并集合” 实现环检测；

重置 MST 边计数 mst\_e\_cnt=0（记录已选入 MST 的边数）。

三. Kruskal 算法核心迭代（选边构建 MST）

循环遍历排序后的边数组 store\_e（终止条件：遍历完所有边 或 已选够num\_v-1条 MST 边）：

步骤 1：通过 find\_index() 找到当前边起点、终点对应的顶点索引 idx1、idx2；

步骤 2：环检测：调用并查集的 find() 方法，判断 idx1 和 idx2 是否属于同一集合：

若不属于同一集合：说明选这条边不会形成环，调用 unite() 合并两个顶点的集合，将这条边存入 MST 结果数组 mst\_final，并将 MST 边计数 mst\_e\_cnt 加 1；

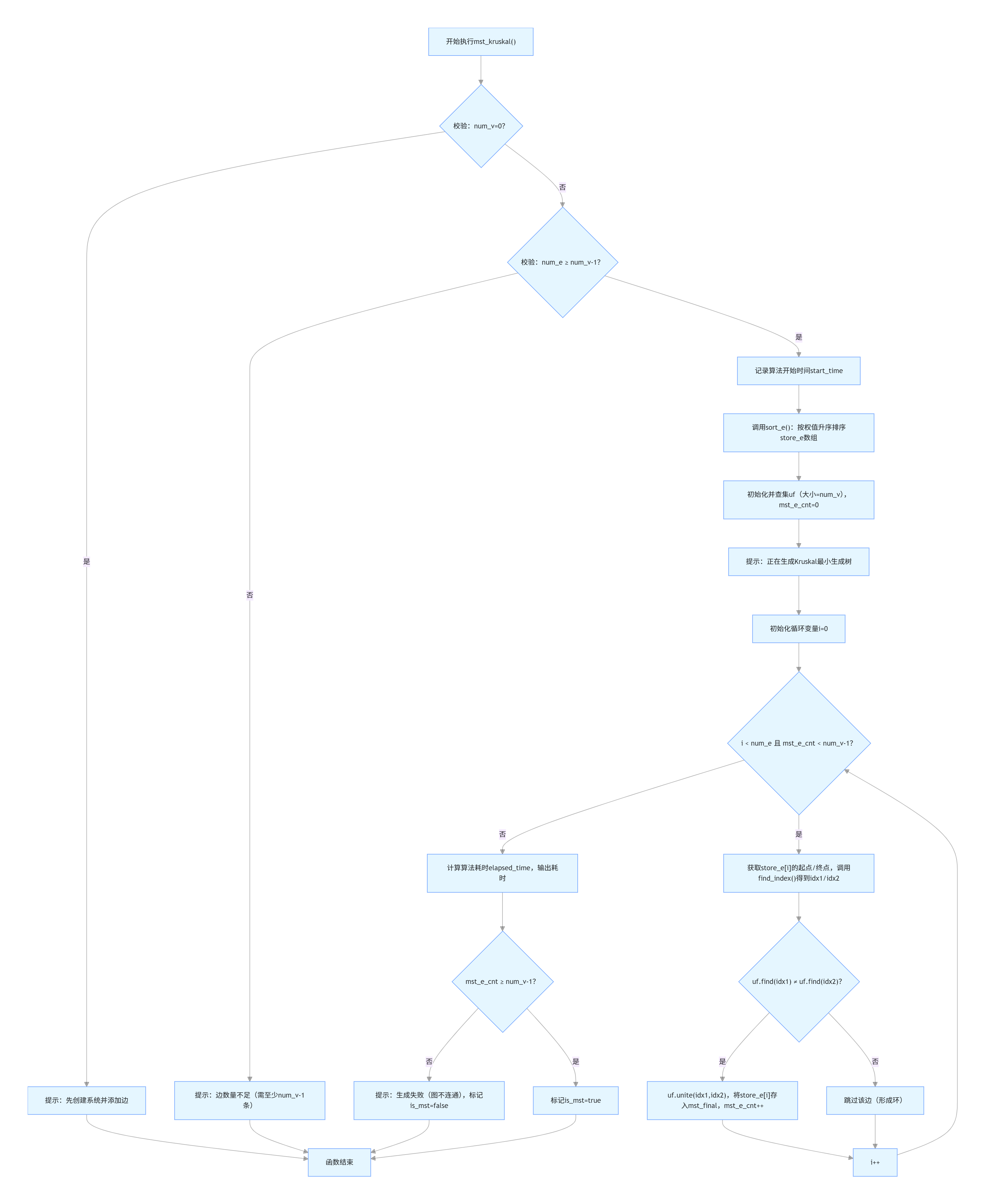
若属于同一集合：说明选这条边会形成环，直接跳过该边。

四. 收尾处理与结果校验

统计算法结束时间，计算并输出耗时（单位：秒，保留 6 位小数）；

校验 MST 边数：若 mst\_e\_cnt < num\_v-1，说明遍历完所有边仍未选够 MST 所需边数，即图不连通，提示 “生成失败：图不连通”，标记 is\_mst=false 并退出；

若边数达标：标记 is\_mst=true（MST 生成成功），输出算法耗时，结束函数（show\_mst() 被注释，仅输出耗时）。



### 3.3.2 kruskal功能核心代码

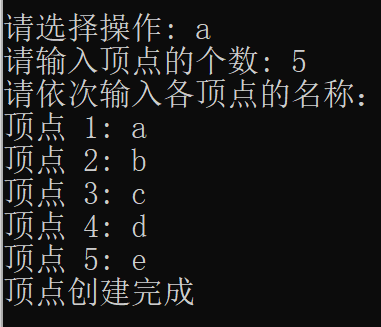
// 构造最小生成树Kruskal算法  
void grid\_sys::mst\_kruskal() {  
    // Kruskal 算法构造最小生成树  
    if (num\_v == 0) {  
        cout << "请先创建系统并添加边" << endl;  
        return;  
   }  
    if (num\_e < num\_v - 1) {  
        cout << "错误：边数量不足。顶点数(" << num\_v << ")，至少需要 " << (num\_v - 1) << " 条边。" << endl;  
        return;  
   }  
    clock\_t start\_time = clock();  
    sort\_e();  
    find\_set uf(num\_v);  
    mst\_e\_cnt = 0;  
    cout << "正在生成Kruskal最小生成树" << endl;  
    for (int i = 0; i < num\_e && mst\_e\_cnt < num\_v - 1; i++) {  
        int idx1 = find\_index(store\_e[i].start);  
        int idx2 = find\_index(store\_e[i].end);  
        if (uf.find(idx1) != uf.find(idx2)) {  
            uf.unite(idx1, idx2);  
            mst\_final[mst\_e\_cnt] = store\_e[i];  
            mst\_e\_cnt++;  
       }  
   }  
    clock\_t end\_time = clock();  
    double elapsed\_time = (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC;  
    if (mst\_e\_cnt < num\_v - 1) {  
        cout << "生成失败：图不连通" << endl;  
        is\_mst = false;  
        return;  
   }  
    is\_mst = true;  
    //show\_mst();  
    cout << "Kruskal 算法耗费时间: " << fixed << setprecision(6) << elapsed\_time << " 秒" << endl;  
}

# 4 项目测试

## 4.1 功能测试

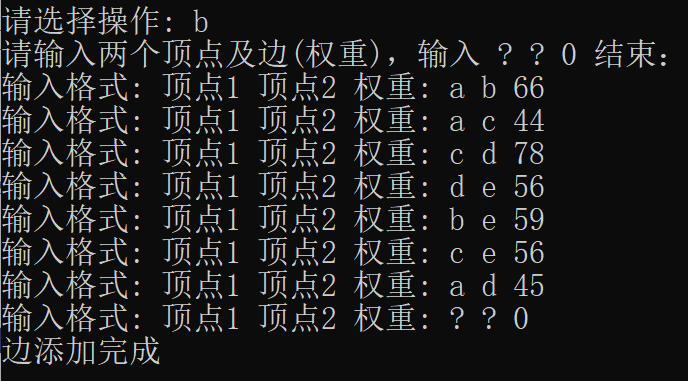
### 4.1.1 创建顶点测试

**实验结果**



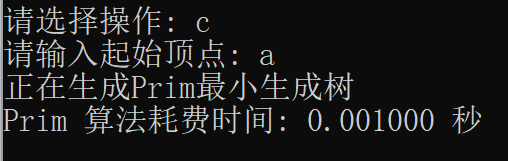
### 4.1.2 添加边测试

**实验结果：**



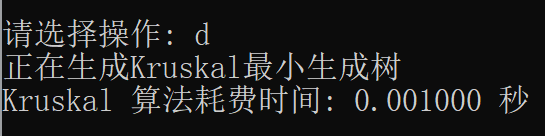
### 4.1.3 prim算法测试

**实验结果：**



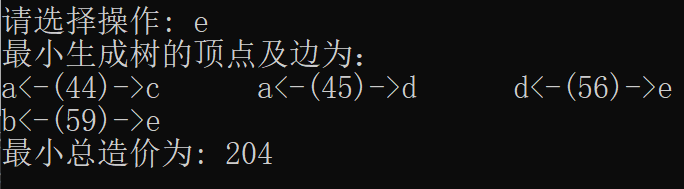
### 4.1.4 kruskal算法测试

**实验结果：**

****

### 4.1.5 显示路径测试

**实验结果：**

****

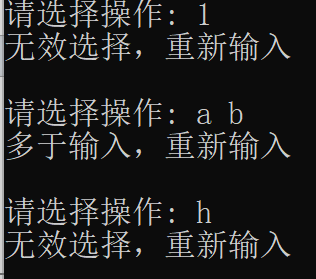
## 4.2 错误测试

### 4.2.1 选择操作不合要求

**测试用例：1|a b|h**

**预期结果：**错误提示，运行正常不崩溃。

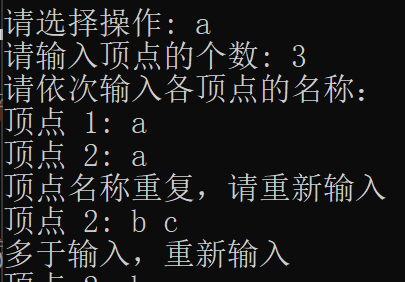
**实验结果：**



### 4.2.2 顶点重复与多于输入

**测试用例：a|a|b c**

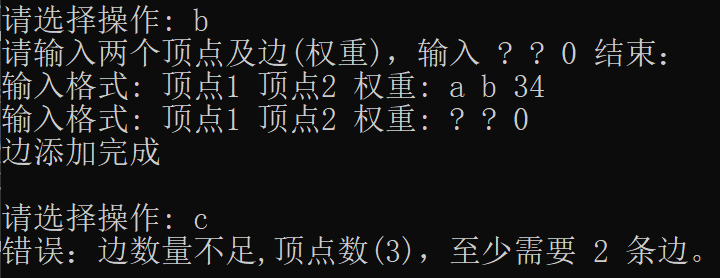
**预期结果：**错误提示，程序运行正常不崩溃。**实验结果：**

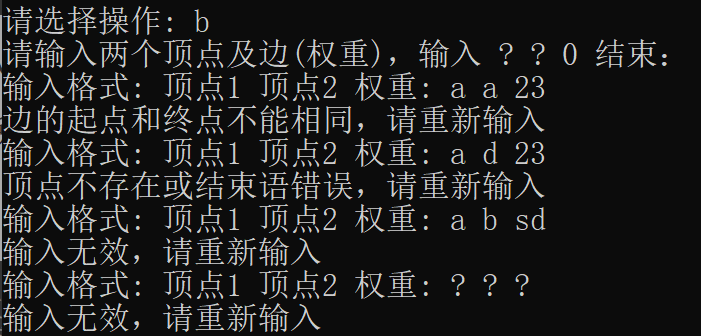


### 4.2.2 边过少，起点终点相同，输入格式不对，结束与错误

**测试用例：如截图所示**

**预期结果：**错误提示，程序运行正常不崩溃。**实验结果：**





# 5 算法对比测试

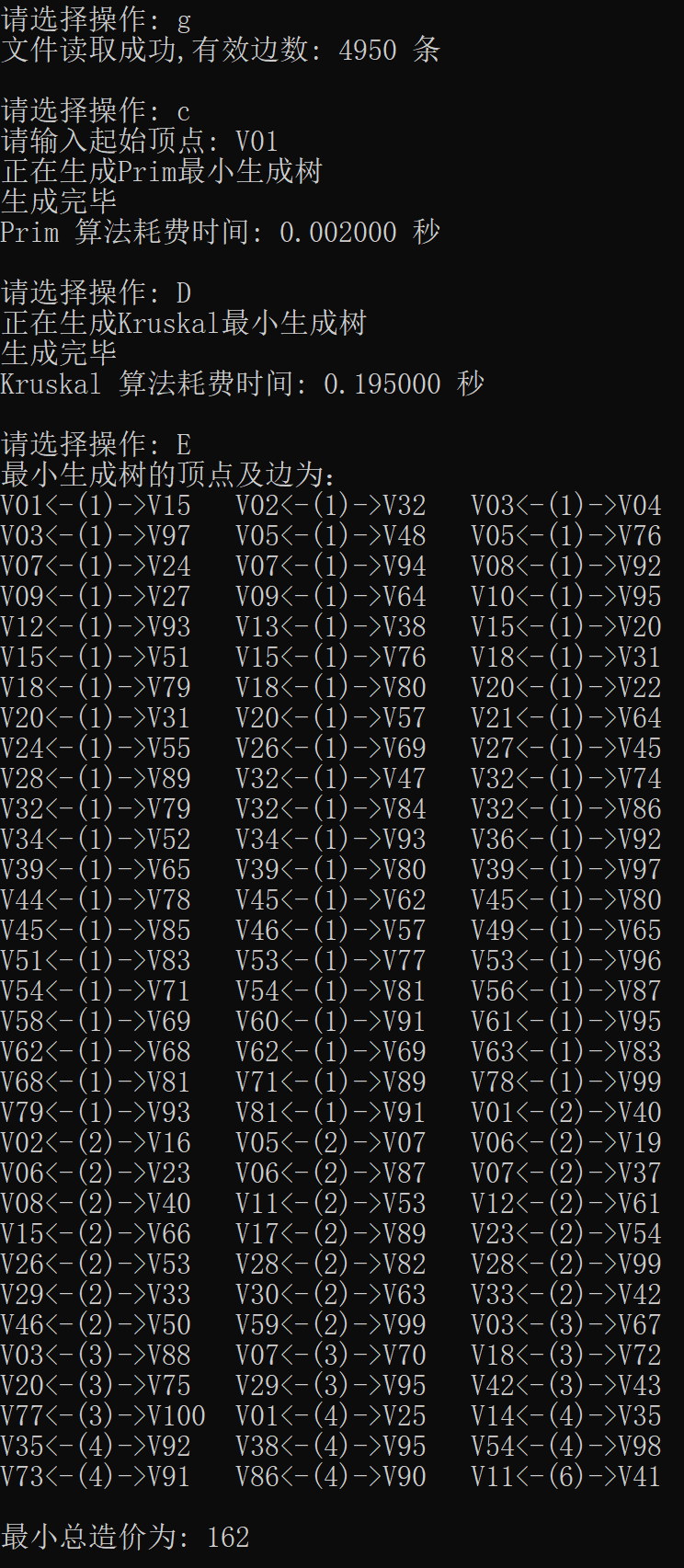
**一、核心思想与实现差异**

| 维度 | Prim 算法 | Kruskal 算法 |
| --- | --- | --- |
| 核心思想 | 从顶点出发，贪心选择 “已加入 MST 的顶点集合” 与 “未加入集合” 的最小权边，逐步扩展 MST。 | 从边出发，先按权值排序，贪心选择不形成环的最小权边，通过并查集合并连通分量。 |
| 依赖数据结构 | 邻接矩阵 / 邻接表 + 数组（lowcost/closest），优化版用优先队列。 | 边列表 + 并查集（Union-Find）。 |
| 实现复杂度 | 较简单（无需排序 / 并查集）。 | 略复杂（需排序边 + 实现并查集）。 |

**二、时间 / 空间复杂度对比**

| 维度 | Prim 算法 | Kruskal 算法 |
| --- | --- | --- |
| 时间复杂度 | - 邻接矩阵版：O(V2)（V 是顶点数）- 邻接表 + 优先队列版：O(ElogV)。 | O(ElogE)（E 是边数，核心开销是边排序）。 |
| 空间复杂度 | 主要依赖顶点数：O(V2)（邻接矩阵）或O(V+E)（邻接表）。 | 主要依赖边数：O(E)（存储所有边）。 |

**三、使用文件读取大数据（稠密图，边多于点）**



**三、优劣与适用场景**

**1. Prim 算法的优劣**

**优势**：1.对稠密图（边数E≈V2）更高效；2.内存访问更连续（顶点操作），缓存命中率高；3.无需排序边，避免稠密图下的排序开销。

**劣势**：1.对稀疏图效率低（V2复杂度远大于ElogE）；2.依赖邻接矩阵时空间开销大（V2）。

**2. Kruskal 算法的优劣**

**优势**：1.对稀疏图（边数E≪V2）更高效（排序开销小）；2.空间开销小（仅存边列表）；3.适合动态边场景（新增边只需插入排序）。

**劣势**：1.对稠密图效率低；2.并查集操作有额外开销。

**四、总结**

案例中边数 4950 属于稠密图（接近顶点数的平方），因此 Prim 算法表现远优于 Kruskal—— 这也印证了两者的核心适用场景差异：

选Prim：稠密图、顶点数较少但边数多的场景（如电网、数据中心网络）；

选Kruskal：稀疏图、边数少的场景（如社交网络、分布式节点连接）。

# 6 项目心得

完成电网建设造价模拟系统的开发，是一次将数据结构理论与实际应用结合的扎实实践，没有复杂的感悟，更多的是实实在在的收获和对编程的深入理解。

一开始拿到最小生成树这个需求，只知道是找连接所有顶点且总权重最小的边集，但对 Prim 和 Kruskal 两种算法的实际实现还是很模糊。翻了课本和资料后，才慢慢理清思路：Prim 算法适合稠密图，基于邻接矩阵找最小边；Kruskal 算法适合稀疏图，需要排序边并通过并查集避免环。这个从理解问题到选择算法的过程，让我明白数据结构不是死记硬背的知识点，而是解决实际问题的工具，选对工具才能高效解题。

在编码过程中，细节问题最磨人。比如实现 Kruskal 算法时，一开始忘了对边排序，导致生成的 MST 总造价错误；实现并查集时，没加路径压缩，虽然功能能实现，但总觉得效率不够高。还有输入验证部分，一开始只考虑了非数字输入，测试时才发现用户可能输入多余字符、重复顶点名称，不得不一次次补全逻辑。这些经历让我深刻体会到，编程不仅要实现功能，还要考虑周全，多测试、多排查，才能让程序稳定可靠。

另外，这次项目让我对封装和模块化有了更具体的认识。把电网系统的功能封装到 grid\_sys 类中，把并查集的操作封装到 find\_set 类中，每个类各司其职，后续修改或调试时不用到处找代码，逻辑更清晰。比如后续想优化排序算法，只需要修改 sort\_e 函数，不影响其他功能，这让我明白好的代码结构能大大提高开发效率。

还有文件读取功能的实现，一开始没想到批量输入的需求，后来添加了从文件读取数据的功能，不仅简化了测试流程，也让系统更实用。这让我意识到，编程不能只满足于完成基本要求，还要多站在用户角度考虑，增加一些实用的功能，让程序更具价值。

总的来说，这个项目不算复杂，但从头到尾做下来，让我把最小生成树算法、并查集、邻接矩阵、输入验证等知识点都过了一遍。没有什么特别深刻的感悟，只是实实在在地知道了：学编程就是要多动手、多试错，遇到问题拆解问题，不懂的知识点翻书查资料，一步一个脚印就能把看似难的问题解决。这次实践也让我更踏实，知道只有把基础打牢，才能应对更复杂的编程任务。