河南理工大学

《数据结构》

课程设计报告

（2022—2023 学年第一学期）

题　　目　电网造价设计

学生姓名　刘晨阳

专业班级　计算机 2106

学生学号　312105010207

教师姓名　贾盼盼

成　　绩：

评　　语：

教师签名：

日期：2022 年 12 月 23 日

目　录

1. 需求分析

1.1 问题描述

假设一个城市有 n 个小区，要实现 n 个小区之间的电网都能够相互接通，构造这个城 市 n 个小区之间的电网，使总工程造价最低。请分别用克鲁斯卡尔和普里姆两种算法分别 设计一个能满足要求的造价方案。

1.2 问题要求

要求设计并编写一程序，选择适当的数据结构，解决上述问题，要求（1）输入输出界面友好；（2）程序可读性强；（3）程序具有较强的健壮性；（4）程序能够根据用户输入的电网信息建立邻接矩阵以及边集数组；（5）程序能够分别根据 Prim 算法以及 Kruskal 算法计算电网的最低造价；（6）输出不同算法情况下计算的电网的最低造价图

1.3 问题分析

该程序需求是为一个城市所有的小区搭建一个电网，而这个电网要实现能让该城市的所有小区都能够实现相互连通，那么这里我们可以把它理解成为一个连通图。

并且需求还提出了，搭建的电网在能够使所有小区都连通的情况下，电网造价最低，也就说使总工程造价最低，这里我们可以理解成让抽象出来的连通图的所有权重之和达到一个最小值，那么总的来说就是求一个图的最小生成树问题。

总结以下就是，首先选择一个合理的存储结构去存储我们根据城市小区线路信息抽象出来的电网图，然后利用 Prim 算法以及 Kruskal 算法去求出这张图的最小生成树。

2 总体设计

2.1 设计思路

- 本游戏的数学建模如下：

在每个小区之间都可以设置一条电网线路，相应的都要付出一点经济代价。n 个小区 之间最多可以有 n(n-1)/2 条线路，选择其中的 n-1 条使总的耗费最少。可以用连通网来表 示 n 个城市之间以及 n 个城市之间可能设置的电网线路，其中网的顶点表示小区，边表示 两个小区之间的线路，赋予边的权值表示相应的代价。对于 n 个顶点的连通网可以建立许 多不同的生成树，每一颗生成树都可以是一个电路网。现在，我们要选择总耗费最少的生 成树，就是构造连通网的最小代价生成树的问题，一颗生成树的代价就是树上各边的代价 之和。

- 本游戏的要求用户输入的内容包括：

（1） 小区的个数，也就是 n 的值；

（2）所有小区之间的线路数，也就是图的边数值；

（3）每个小区之间线路路径的具体情况，也就是图中每条路径的起点，终点，以及权重值，按要求输入即可；

（4）小区之间的电网造价设计图抽象为一组数据存储在邻接矩阵中

- 本游戏要求输出的内容是包括

（1）根据小区电网造价设计图抽象出来的邻接矩阵的值；

（2）根据不同算法对电网设计出来最低造价图的路径；

所以，根据上面的模型分析及输入输出参数分析，可以定义一种数据结构后进行算法实现。

2.2 总体设计功能图

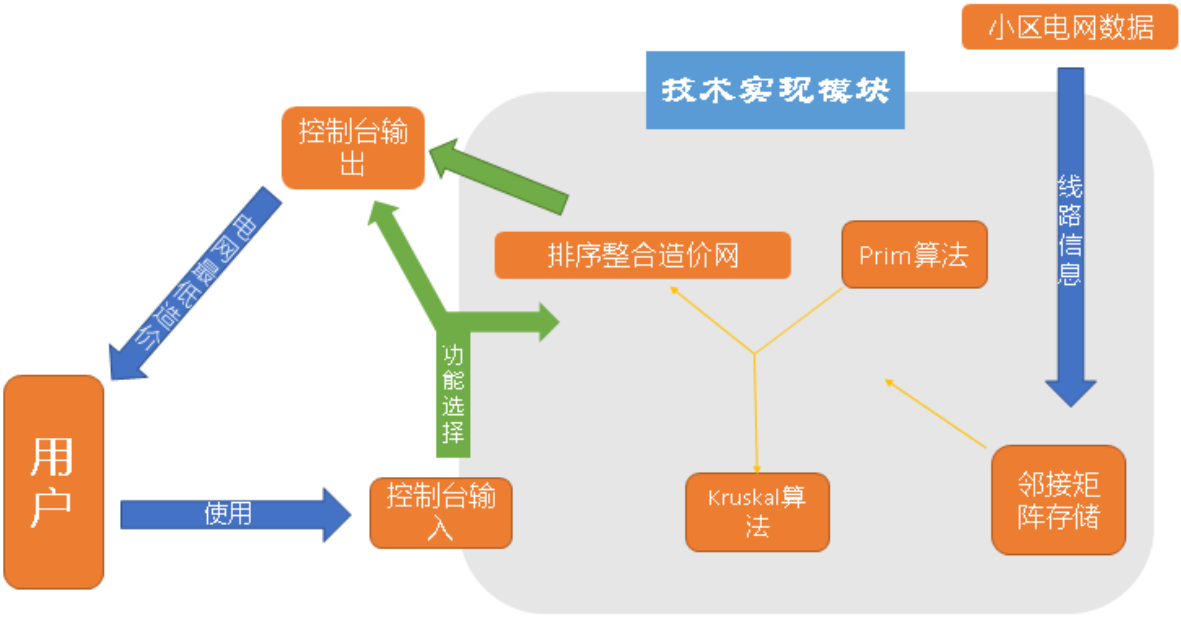


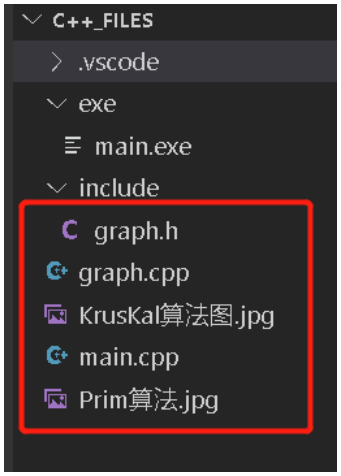
图 2-1 总体功能图（5 号字,按节编号,表示第 2 节第 1 个图)

2.3 程序设计文件部署与说明

1. 程序环境部署

本程序主要采用 vscode 编辑器配置 g++编译器进行编写，选用 C++语言，C++标准采用 C++14 标准。

2. 文件部署



程序主要由三个文件构成，分别为 graph.h(头文件)，graph.cpp(源文件)，main.cpp(程序入口源文件)

头文件放在 include 文件夹中，源文件全部放在项目根目录下，生成的可执行程序放在 exe 文件夹中。

- graph.h 头文件中主要写数据存储结构以及函数声明
- graph.cpp 源文件中为各函数具体定义
- main.cpp 文件为程序入口以及 UI 界面操作

3.详细设计

3.1 抽象数据类型定义

程序采用 C++编程语言进行编写，这里主要根据程序需求设计了一个抽象的图类。该抽象图类中主要分为两大部分，将他分为两大部分的属性。

第一部分属性主要包括图的边数，以及其对应的邻接矩阵。第二部分属性主要包括该抽象图类在接下来的程序中会用到的主要操作，由于函数具体实现在源

文件中，故在头文件的图类里面注明了各函数的声明，如下为图类定义

```
1.//设置一个图类定义
2.class Solution
3.{
4.    public://公有属性
5.    int edge_num;//边数
6.    int adj_matrix[MaxV][MaxV];//邻接矩阵
7.    Edge adj_pn[num];
8.    int vex=MaxV;//顶点数
9.
10.
11.     public://公有属性
12.     //1.初始化邻接矩阵(边集合数组，顶点数，边数)
13.     int initial_adj_matrix(Edge*pn,int node,int edge);
14.
15.     //2.创建一个检查函数：(起点，终点，权重，顶点数)
16.     void check(int i,int j,int w,int node);
17.
18.     //3.创建邻接矩阵(边集数组结构体指针，边数，顶点数)
19.     void create_adj_matrix(Edge*pn,int edge,int node);
20.
21.     //4.将边集数组从小到大排序(边集数组，边的个数)
22.     void sort_edge(Edge*pn,int edge);
23.
24.     //5.prim 算法实现图的最小生成树(访问数组，顶点数，邻接矩阵，边集数组)
25.     void adj_prim(int vis[MaxV],int n,int G[MaxV][MaxV],Edge edge_prim[MaxV]);
26.
27.     //6.邻接矩阵测试图数据函数(邻接矩阵)
28.     void G(int G[MaxV][MaxV],Edge*pn);
29.
30.     //7.KrusKal 算法(边集数组，邻接矩阵，边数)
```

```
    void adj_KrusKal(Edge* pn,int adj_matrix[MaxV][MaxV],int edge);

    //8.判断是否围成圈(标记访问数组，遍历索引)
    int get_end(int vends[], int i);

    //9.创建边集数组（Kruskal）(邻接矩阵，边数，顶点数)
    void create_adj_pn(int G[MaxV][MaxV],int edge_num,int node);
};
```

3.2 存储结构设计

1. 存储结构设计原理说明：

程序的存储结构主要采用**邻接矩阵**和**边集数组**两种形式，具体代码形式如下，我们将电网图抽象为一张图。

图上的每一个结点代表每一个小区，而小区之间的线路我们用图的边来表示，图上具体的距离信息我们用图的权重来表示。

求最低电网造价设计，要保证在每个小区都连通的基础上，保证我们的权重之和最低。首先我们采用图的结构来进行抽象，其次用邻接矩阵来表示小区与小区之间的所有可能的线路关系，然后经过我们的 Prim 算法和 Kruskal 算法计算出的路径图，将满足最低造价条件的图中的线路关系重新用一个边集数组去保存。

2. 具体存储结构实现：

- **图存储结构采用邻接矩阵**

```
1.int adj_matrix[MaxV][MaxV];//邻接矩阵
```

- **边存储结构**

```
typedef struct
{
    int start_vex,stop_vex;//边的起点和终点
    int weight;//边的权
}Edge;
```

3. 存储设计预定义值：

```
//设置最大顶点数
const int MaxV=7;
//设置最大权重
//默认最大权值即无法到到达
const int MaxValue=99;
```

3.3 算法设计（自然语言+流程图）

1） 问题分析

本课程设计主要将电网图抽象为图结构，采用邻接矩阵结构进行电网线路信息的存储，分别通过 Prim 算法和 Kruskal 算法遍历图结点和边信息，生成最小代价生成树，一棵生成树的代价就是树上各边的代价之和。

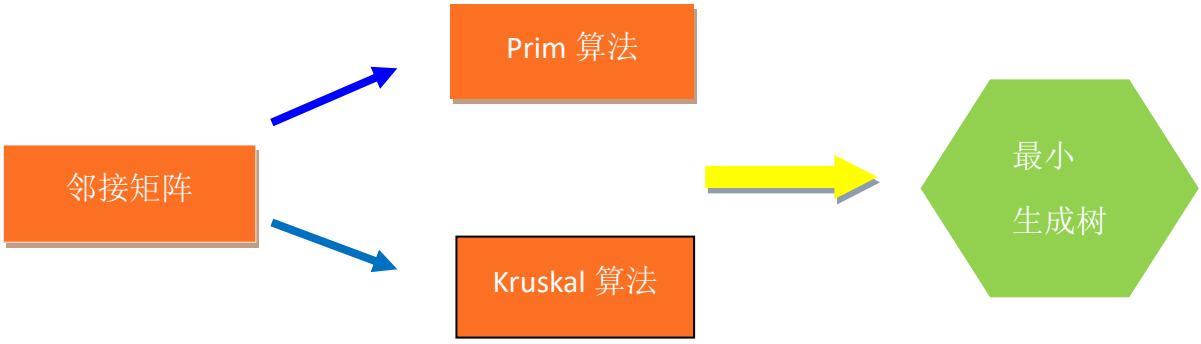


图 3-1

2) Prim 算法

设 G=（V，E）是具有 n 个顶点的网络，T=（U，TE）为 G 的最小生成树，是 T 的顶点集合，TE 是 T 的边集合。

Prim 算法的基本思想是：首先从顶点集合 V 中取出任一顶点在集合 U 中另一个顶点在集合 V-U 里的边，使权（u，v）（u∈U，v∈V-U）最小，将该边放入 TE，并将顶点加入集合 U。重置上述操作知道 U=V 为止。这时 TE 中有 n-1 条边，T=（U，TE）就是 G 的一颗最小生成树。

这里我们的 Prim 算法主要采用暴力循环遍历的方式去依次添加未访问的过点，使得在已访问过的点中和添加的未访问过的点组成的边的权最小。

用两个集合分别表示未访问的点以及访问过的点。

图 3-2

设置一个访问数组 vis[MaxV]，访问过的结点值设为 1。而未访问过的结点值就设为 0.初始化的时候，该访问数组所有结点的值均设为 0。从邻接矩阵的每一行元素依次对应邻接矩阵的每一列元素，

这里设置两个循环，表示一趟筛选权值最小边的过程，且每新加入一个合适的结点，就将其访问数组中对应值设为 1，表示已经访问过了。

由于 Prim 算法结束条件是直到 U=V 后就结束，故再设置 一次嵌套循环，表示所有趟的筛选过程。总结就是设置三层嵌套循环，依次完成筛选最小权值边的过程。

3) Kruskal 算法

Kruskal 算法基本思想：

克鲁斯卡尔算法，用来求加权连通图的最小生成树，给定一张图，设定 n 个顶点，按照权值大小从小到大顺序依次选择 n-1 条边，且这 n-1 条边不能构成回路。具体做法是，首先构造一个只含 n 个顶点的森林，然后后依据权值从小到大从联通网中选择权重最小的边加到森林中，并且该森林中不产生回路，直到森林变成一棵树为止。

这里 Kruskal 算法的具体实现的关键在于依次选择权重最小边的同时，如何去判断森林中是否会形成回路。那么这里用到的是并查集的思想，简单来说就是每次加入的边的两端顶点不能都指向同一个终点，如果都指向同一个终点，那必然是回形成回路的，因为代表某一个顶点是可以从自己到达自己的，所以此处在 Kruskal 算法的基础上额外定义了一个回路判断函数用于辅助我们的 Kruskal 算法。

程序定义的 Kruskal 算法的实现主要是通过对边集合数组的一个遍历，在遍历之前，首先将边集数组进行一个按照权值从小到大的排序。以便之后对边的依次选择。每遍历一条边的时候，都要对该森林是否形成回路进行一次判断，如果形成回路，那么这条边就不能被添加，如果不会形成回路，我们就该边的两个顶点划分到一个阵营中，为下次的回路判断提供新的数据更新，然后依次将符合条件的边保存下来，直到该森林变成一棵树。

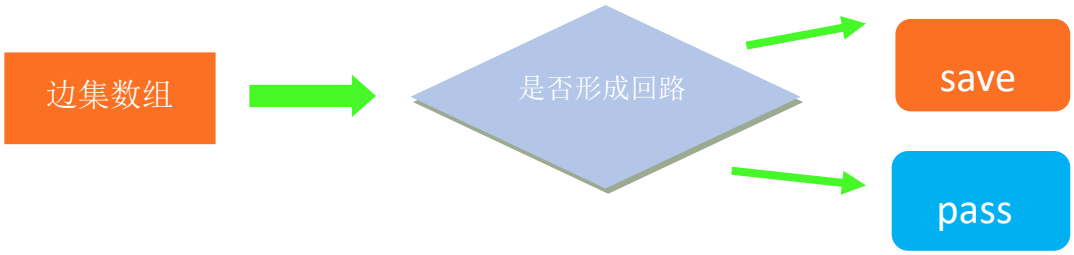


图 3-3

4.函数说明

4.1 函数清单

程序设计主要由 9 个功能函数组成，函数声明如下所示，括号内代表形参的具体代表意义。

1. 初始化邻接矩阵函数(边集合数组，顶点数，边数)

int initial_adj_matrix(Edge*pn,int node,int edge);

该函数主要用于对存储结构邻接矩阵进行一个初始化操作，数组数据类型为整形，自身到自身的权重设为 0，表示不可到达，其他的权重全部初始化为 99，同代表不可到达。

2. 创建一个检查函数：(起点，终点，权重，顶点数)

void check(int i,int j,int w,int node);

该函数主要用于对创建邻接矩阵时，用户输入的值进行合理性判断，如果输入数据不合理或没有按照要求输入，将提醒用户重新输入。

3. 创建邻接矩阵(边集数组结构体指针，边数，顶点数)

void create_adj_matrix(Edge*pn,int edge,int node);

该函数主要用于创建邻接矩阵，将用户输入的小区线路信息进行录入，同时生成邻接矩阵，以存储图的信息。当输入的边的权重为 0 时，输入结束。

4. 将边集数组从小到大排序(边集数组，边的个数)

void sort_edge(Edge*pn,int edge);

该函数主要用于对边集数组内的边按权重从小到大进行一个排序，以便 Kruskal 算法的边的选择

5. prim 算法实现图的最小生成树(访问数组，顶点数，邻接矩阵，边集数组)

void adj_prim(int vis[MaxV],int n,int G[MaxV][MaxV],Edge edge_prim[MaxV]);

该函数主要用于使用 Prim 算法对电网线路图实现最小生成树，实现小区最低造价网的生成。

6. 邻接矩阵测试图数据函数(邻接矩阵)

void G(int G[MaxV][MaxV],Edge*pn);

该函数主要是一个测试函数，为了方便用户对程序正确性以及可行性进行一个验证而准备的验证数据，避免用户频繁输入线路信息进行检验。

7. KrusKal 算法(边集数组，邻接矩阵，边数)

void adj_KrusKal(Edge* pn,int adj_matrix[MaxV][MaxV],int edge);

该函数主要用于使用 Kruskal 算法对电网线路图实现最小生成树，实现小区最低造价网的生成。

8. 判断是否围成圈(标记访问数组，遍历索引)

int get_end(int vends[], int i);

该函数主要采用并查集思想，用于对 Kruskal 算法依次进行最小权重边的选择时，对正在构造图的森林进行判断是否会形成环。

9. 创建边集数组（Kruskal）(邻接矩阵，边数，顶点数)

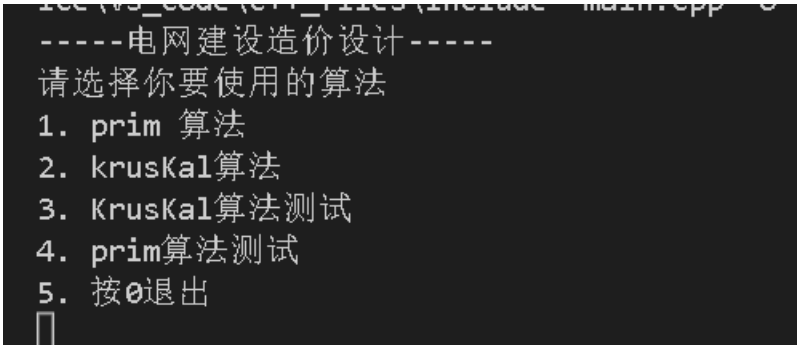
void create_adj_pn(int G[MaxV][MaxV],int edge_num,int node);

该函数主要用于创建生成边集数组，Kruskal 算法会用到。

4.2 主函数

主函数主要是提供一个程序入口，通过对 graph.h 中的编写的功能函数进行调用，分模块以完成需求实现。

采用 switch 选择结构，主要分五个选择，其中按 0 退出，反之则可以重复选择，以便用户使用。



主函数程序如下

```
#include"graph.cpp"
int n=MaxV;int G[MaxV][MaxV];   //n 为顶点数，MaxV 为最大顶点数  G 为邻接矩阵
int vis[MaxV] = {0};    //标记数组，vis[i]==true 表示已访问。初始值均为 false
Edge edge_prim[num]; //边集数组
int edge_num;//边数

int main()
{
    int choice=99;
     while(choice!=0)//按 0 退出
     {
     std::cout<<"-----电网建设造价设计-----"<<endl;
     Solution A;//创建一个对象 A
     std::cout<<"请选择你要使用的算法"<<endl;
     std::cout<<"1. prim 算法"<<endl;
     std::cout<<"2. krusKal 算法"<<endl;
     std::cout<<"3. KrusKal 算法测试"<<endl;
     std::cout<<"4. prim 算法测试"<<endl;
```

```
19.     std::cout<<"5. 按 0 退出"<<endl;
20.     cin>>choice;
21.     switch (choice)
22.     {
23.     case 1:
24.         std::cout<<"请输入小区个数(顶点数)"<<endl;cin>>n;A.vex=n;
25.         std::cout<<"请输入小区之间的路径数(边数)"<<endl;cin>>edge_num;
26.         //初始化邻接矩阵和边集数组
27.         A.initial_adj_matrix(edge_prim,n,edge_num);
28.         //创建邻接矩阵和边集数组
29.         A.create_adj_matrix(edge_prim,edge_num,n);
30.         A.adj_prim(vis,n,A.adj_matrix,edge_prim);
31.         break;
32.     case 2:
33.         std::cout<<"请输入小区个数(顶点数)"<<endl;cin>>n;A.vex=n;
34.         std::cout<<"请输入小区之间的路径数(边数)"<<endl;cin>>edge_num;
35.         //初始化邻接矩阵和边集数组
36.         A.initial_adj_matrix(edge_prim,n,edge_num);
37.         //创建边集数组
38.         A.create_adj_pn(A.adj_matrix,edge_num,n);
39.         A.adj_KrusKal(A.adj_pn,A.adj_matrix,edge_num);
40.         break;
41.     case 3://测试 KrusKal 算法
42.         A.G(A.adj_matrix,edge_prim);//测试数据函数
43.         A.adj_KrusKal(edge_prim,A.adj_matrix,10);//调用 Kruskal 算法函数
44.         break;
45.     case 4://测试 prim 算法
46.         A.G(A.adj_matrix,edge_prim);//测试数据函数
47.         A.adj_prim(vis,n,A.adj_matrix,edge_prim);//调用 Prim 算法函数
48.         break;
49.     default:
50.         break;
51.     }
52.     }
53. }
```

4.3 主要函数

1. **Prim 算法：**

prim 算法函数的形参为 (访问数组，顶点数，邻接矩阵，边集数组)

其中访问数组用来记录已经访问过的顶点，邻接矩阵用来反映电网图的信息。该算法主要通过三

个 for 循环遍历邻接矩阵，边集数组用于存储 Prim 算法依次选择顶点所构成的边，最后通过边集数组将 Prim 算法筛选出的边输出。

```
2.  void Solution::adj_prim(int vis[MaxV],int n,int G[MaxV][MaxV],Edge edge_prim[MaxV])
3.  {
4.      vis[0]=1; int flag=0; int weight=99;int t=0;int weight_sum=0;//(t 用于记录 j)
5.      for(int m=0;m<MaxV;m++)
6.      {
7.          weight=99;
8.          for(int i=0;i<n&&flag<MaxV;i++)//每一行的
9.          {
10.             if(vis[i]==1)//代表其已经访问
11.             {
12.                 for(int j=0;j<n&&flag<=MaxV;j++)//每一列的
13.                 {
14.                     if (vis[j]==0&&G[i][j]<weight)//代表其未访问
15.                     {
16.                         edge_prim[flag].weight=G[i][j];
17.                         edge_prim[flag].start_vex=i;
18.                         edge_prim[flag].stop_vex=j;
19.                         t=j;
20.                         weight=G[i][j];
21.                     }
22.                 }
23.             }
24.             else continue;
25.
26.         }
27.     if(weight==99)
28.         continue;//说明它无法构成最小生成树
29.         else
30.         {
31.             vis[t]=1;
32.             flag++;
33.         }
34.     }
35.     // std::cout<<"Prim 算法的最小生成树为（未排序）"<<endl;
36.     // for(int k=0;k<flag;k++)//flag 初始化为 1，所以从 1 开始输出
37.     // {
38.     //  std::cout<<"("<<edge_prim[k].start_vex<<","<<edge_prim[k].stop_vex<<","<<edge_prim[k].weight<
<")"<<" ";
39.     //  weight_sum=weight_sum+edge_prim[k].weight;
40.     // }
```

```
    // std::cout<<"prim 算法所得最小权值之和为"<<weight_sum<<endl;
    sort_edge(edge_prim,flag);
    std::cout<<"Prim 算法的最低造价电网图为（按“权”排序之后）"<<endl;
    weight_sum=0;
    for(int k=0;k<flag;k++)
    {
        std::cout<<"("<<edge_prim[k].start_vex<<","<<edge_prim[k].stop_vex<<","<<edge_prim[k].weight<<")"<<" ";
        weight_sum=weight_sum+edge_prim[k].weight;
    }
    std::cout<<"prim 算法的最低造价电网和为："<<weight_sum<<endl;
}
```

2. Kruskal 算法

Kruskal 算法的函数参数为（边集数组，邻接矩阵，边数），首先将边集数组按边的权重从小到大排序，Kruskal 算法主要通过依次选择权重最小的边，在确保森林不成环的前提下，依次将选择的边存储起来并输出。

```
//8.KrusKal 算法
void Solution::adj_KrusKal(Edge* pn,int adj_matrix[MaxV][MaxV],int edge)
{
    //std::cout<<"开始执行 KrusKal 算法"<<endl;
    int i,m,n,p1,p2;
    int length;
    int index = 0;          // rets 数组的索引
    int vends[MaxV];     // 用于保存"已有最小生成树"中每个顶点在该最小树中的终点。
    std::fill(vends,vends+MaxV,99);
    Edge rets[MaxV];        // 结果数组，保存 kruskal 最小生成树的边
    // 将边按照"权"的大小进行排序(从小到大)
    sort_edge(pn, edge);


    for (i=0; i<edge; i++)
    {
        //std::cout<<"("<<pn[i].start_vex<<","<<pn[i].stop_vex<<","<<pn[i].weight<<")"<<" ";// 输出所有路径
        p1 = pn[i].start_vex;   // 获取第 i 条边的"起点"的序号
        p2 = pn[i].stop_vex;     // 获取第 i 条边的"终点"的序号

        m = get_end(vends, p1);                 // 获取 p1 在"已有的最小生成树"中的终点
        n = get_end(vends, p2);                 // 获取 p2 在"已有的最小生成树"中的终点
        // 如果 m!=n，意味着"边 i"与"已经添加到最小生成树中的顶点"没有形成环路
        if (m != n)
        {
            //std::cout<<"("<<m<<","<<n<<")"<<endl;
```

```
28.             vends[m] = n;                           // 设置 m 在"已有的最小生成树"中的终点为 n
29.             rets[index++] = pn[i];              // 保存结果
30.         }
31.     }
32.
33.     // 统计并打印"kruskal 最小生成树"的信息
34.     length = 0;
35.     for (i = 0; i < index; i++)
36.         length += rets[i].weight;
37.     std::cout<<"Kruskal 算法的最低造价为"<<length<<endl;
38.     std::cout<<"KrusKal 算法的最低造价电网图为（按“权”从小到大排序）"<<endl;
39.     for (i = 0; i < index; i++)
40.         std::printf("(%d,%d,%d) ", rets[i].start_vex, rets[i].stop_vex,rets[i].weight);
41.     std::printf("\n");
42. }
```

5.运行与测试

5.1 正常测试数据

5.1 Prim 算法测试

输入：小区的个数、线路信息（起点 终点 权重）

输出：原始输入信息的邻接矩阵、最低造价电网图路线，依次以（起点，终点，权重）形式呈现。

样例测试 1：

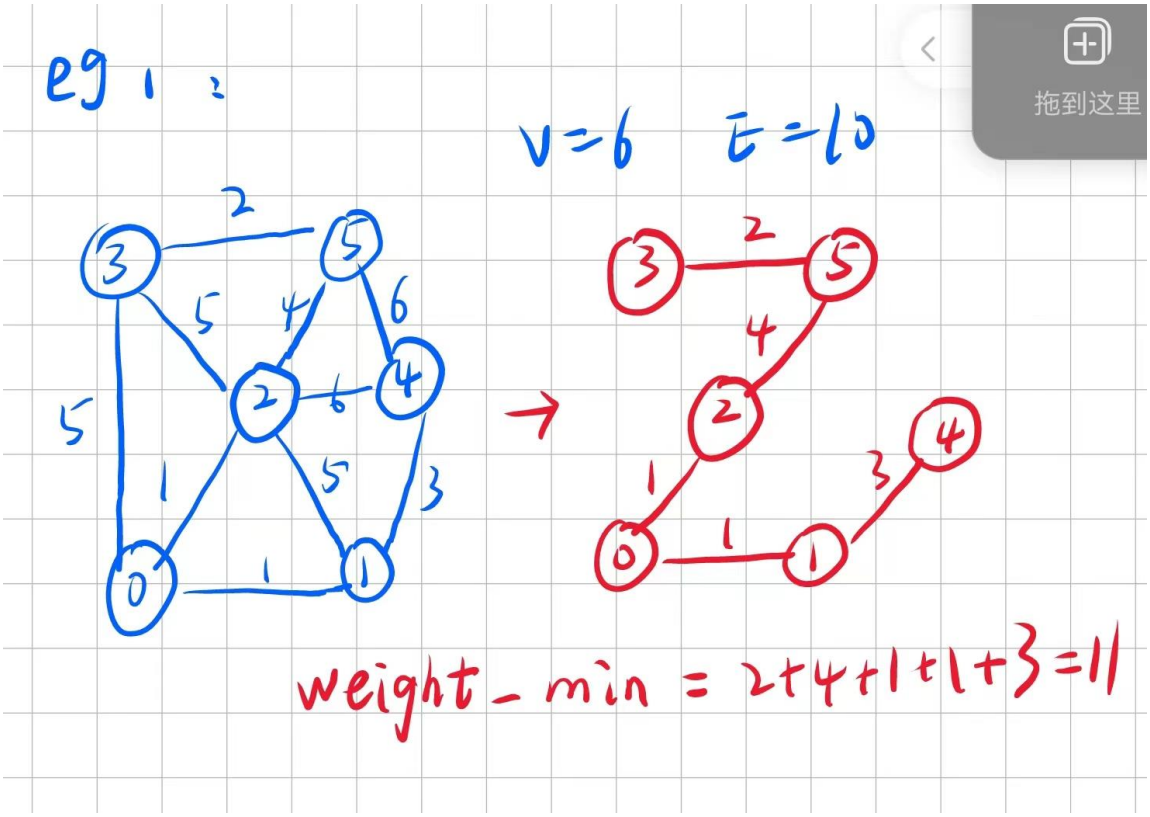
输入：

```
-----电网建设造价设计-----
请选择你要使用的算法
1. prim 算法
2. krusKal算法
3. KrusKal算法测试
4. prim算法测试
5. 按0退出
1
请输入小区个数(顶点数)
6
请输入小区之间的路径数(边数)
10
请依次输入权重不为0的边的起点，终点，权重：
直到输入一条权重为0的边后退出输入
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
0 1 1
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
0 2 1
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
0 3 5
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
1 2 5
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
1 4 3
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
2 3 5
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
2 5 4
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
2 4 6
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
3 5 2
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
4 5 6
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
5 5 0
```

输出： 邻接矩阵、最低造价电网图路线、造价电网和

```
创建的邻接矩阵为:
0 1 1 5 99 99
1 0 5 99 3 99
1 5 0 5 6 4
5 99 5 0 99 2
99 3 6 99 0 6
99 99 4 2 6 0
Prim算法的最低造价电网图为（按“权”排序之后）
(0,1,1) (0,2,1) (5,3,2) (1,4,3) (2,5,4) prim算法的最低造价电网和为: 11
-----电网建设造价设计-----
请选择你要使用的算法
1. prim 算法
2. krusKal算法
3. KrusKal算法测试
4. prim算法测试
5. 按0退出
```

验证：



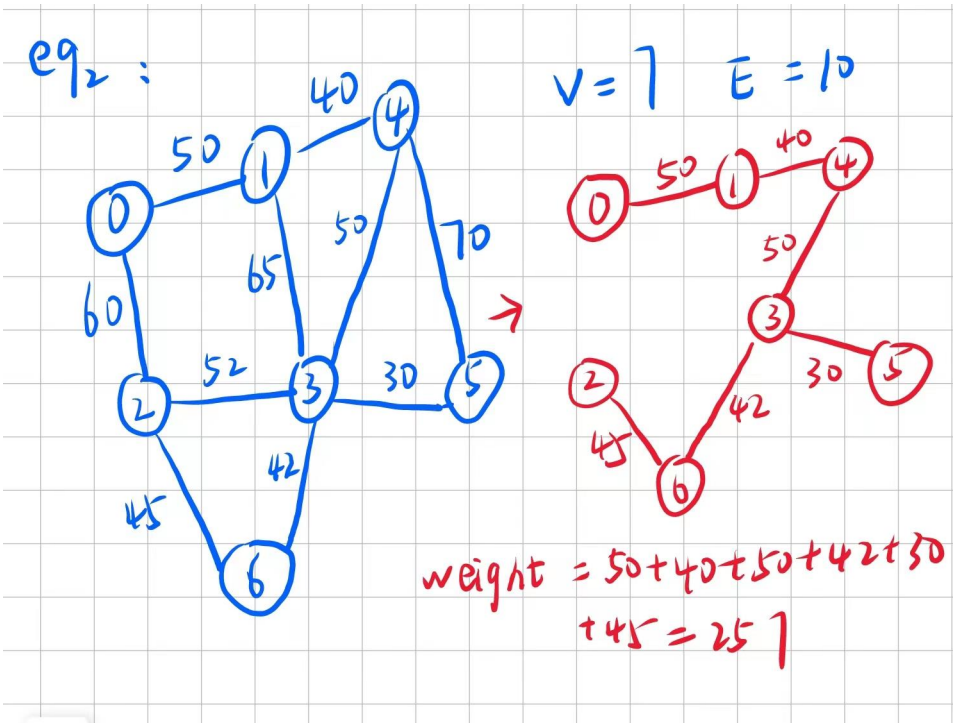
验证通过

样例测试 2：

这里直接采用测试集数据

```
-----电网建设造价设计-----
请选择你要使用的算法
1. prim 算法
2. krusKal算法
3. KrusKal算法测试
4. prim算法测试
5. 按0退出
4
测试图的邻接矩阵为：
0 50 60 99 99 99 99
50 0 99 65 40 99 99
60 99 0 52 99 99 45
99 65 52 0 50 30 42
99 40 99 50 0 70 99
99 99 99 30 70 0 99
99 99 45 42 99 99 0
Prim算法的最低造价电网图为（按“权”排序之后）
(3,5,30) (1,4,40) (3,6,42) (6,2,45) (4,3,50) (0,1,50) prim算法的最低造价电网和为: 257
-----电网建设造价设计-----
```

验证：



验证通过

5.2 Kruskal 算法测试

输入：小区个数（顶点数）、小区线路线路数（边数）、小区具体线路信息（起点 终点 权重）。

输出：边集数组、Kruskal 算法最低造价电网图、电网最低造价和

样例测试 1：

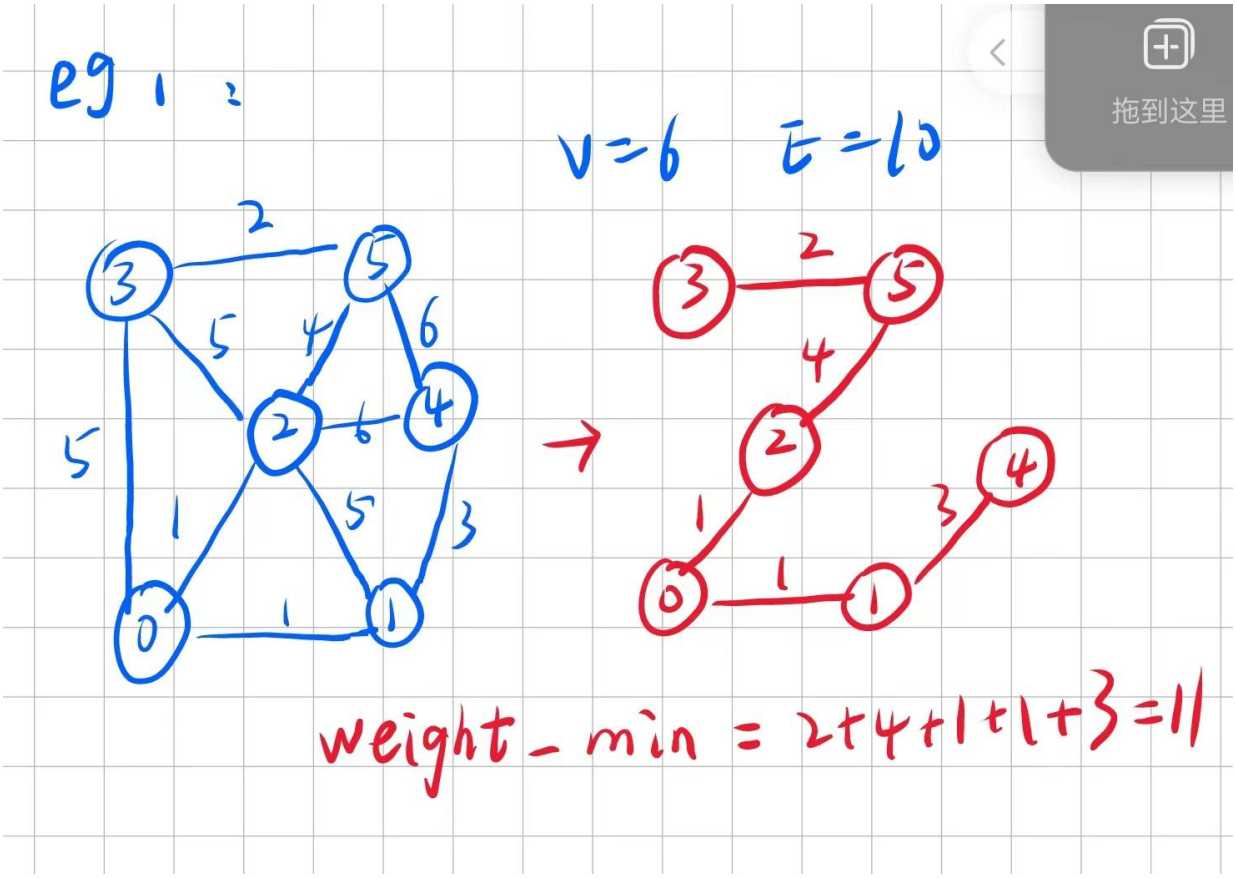
输入：

```
-----电网建设造价设计-----
请选择你要使用的算法
1. prim 算法
2. krusKal算法
3. KrusKal算法测试
4. prim算法测试
5. 按0退出
2
请输入小区个数(顶点数)
6
请输入小区之间的路径数(边数)
10
请依次输入权重不为0的边的起点，终点，权重：
直到输入一条权重为0的边后退出输入
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
0 1 1
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
0 2 1
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
0 3 5
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
1 2 5
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
1 4 3
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
2 3 5
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
2 5 4
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
2 4 6
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
3 5 2
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
4 5 6
```

输出：

```
创建的边集数组为:
(0,1,1) (0,2,1) (0,3,5) (1,2,5) (1,4,3) (2,3,5) (2,5,4) (2,4,6) (3,5,2) (4,5,6)
Kruskal算法的最低造价为11
KrusKal算法的最低造价电网图为（按“权”从小到大排序）
(0,2,1) (0,1,1) (3,5,2) (1,4,3) (2,5,4)
-----电网建设造价设计-----
```

验证：

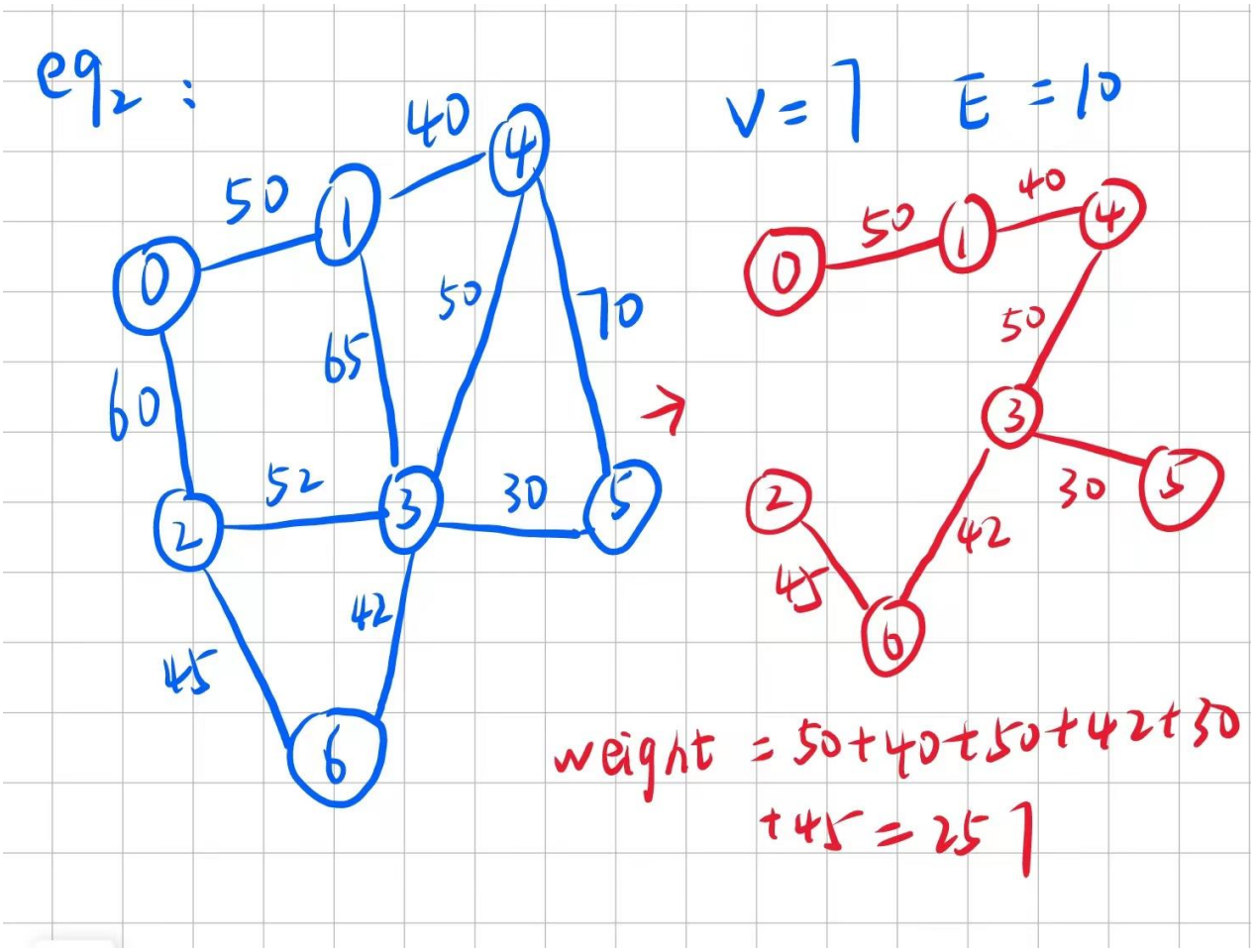


验证通过

样例测试 2

```
-----电网建设造价设计-----
请选择你要使用的算法
1. prim 算法
2. krusKal算法
3. KrusKal算法测试
4. prim算法测试
5. 按0退出
3
测试图的邻接矩阵为:
0 50 60 99 99 99 99
50 0 99 65 40 99 99
60 99 0 52 99 99 45
99 65 52 0 50 30 42
99 40 99 50 0 70 99
99 99 99 30 70 0 99
99 99 45 42 99 99 0
Kruskal算法的最低造价为257
KrusKal算法的最低造价电网图为（按“权”从小到大排序）
(5,3,30) (4,1,40) (6,3,42) (6,2,45) (1,0,50) (4,3,50)
```

验证：



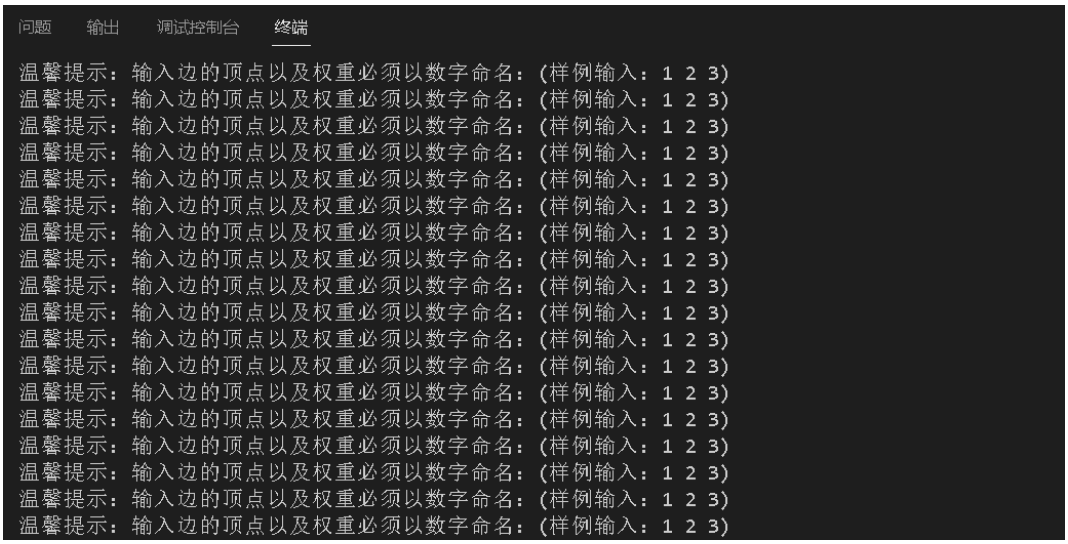
验证通过

5.2 异常测试数据

1.异常样例一

输入电网线路信息：

a b 3



导致原因：

输入的路径信息，顶点目前只支持数字，不支持其他格式

2. 异常样例 2

```
5. 按0退出
1
请输入小区个数(顶点数)
6
请输入小区之间的路径数(边数)
10
请依次输入权重不为0的边的起点，终点，权重：
直到输入一条权重为0的边后退出输入
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
2 5 6
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
2 25 3
输入错误，请重新输入
2 7 6
输入错误，请重新输入
2 6 100
输入错误，请重新输入
2 6 3
温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)
5 5 0
创建的邻接矩阵为：
0 99 99 99 99 99
99 0 99 99 99 99
99 99 0 99 99 6
99 99 99 0 99 99
99 99 99 99 0 99
99 99 6 99 3 0
Prim算法的最低造价电网图为（按“权”排序之后）
prim算法的最低造价电网和为：0
```

导致原因：

输入的电网信息图本身数据有残缺或问题，不是连通图。

6 总结

6.1 发现的问题和解决方法

本次程序设计从思想框架构建到代码实现，经过了反复的调试和验证终于圆满谢幕。其中从撰写到调试到验证确实遇见了许多莫名奇妙的 bug，好在最后都能够成功解决。

问题一：

在写邻接矩阵的创建这个函数的时候，由于需要用户输入，但是刚开始的时候没有设置对用户输入数据的一个范围以及格式的限定，导致后续在去验证以及测试的时候，程序报错，无法正常运行等，发现了后加上判定条件，如果格式或范围不符号要求的话会及时提醒用户重新输入。

问题二：

在写边集数组创建的时候，由于边集数组是一个结构体数组，在将用户输入的线路信息依次添加到边集数组中，出现了无法赋值成功的情况。假设 pn 就是我们所说的边集结构体数组，我设置了一个循环依次让用户输入线路信息并赋值，pn->start_vex=用户输入的值，结果输出的结果与理想值大相径庭，后来我输出了用户输入创建的边集数组，发现边集数组数据离谱至极，在排查原因的时候才发现我忽略了 pn 是一个数组，正确赋值方式应该是 pn[i].start_vex=用户输入的值，修改之后结果就符合情况了。

问题三：

在写 Kruskal 算法中判断是否成环的函数时，设置了一个数组 vend，用于将每次新添加边的两个端点划分到一起。结果出现了异常。起初我是将 vend 数组全部初始化为 0 的，然后 0 代表的就是该顶点还没有被访问。但是这样其实是不合理的，因为在判断成环的过程中有一个重要操作就是 vends[m] = n，而 m 和 n 由于是索引的关系，也是有可能为 0 的，那么这样就会对我们的初始值的代表意义产生一个干扰。所以后来将 vend 数组的初始值改为 99 就可以很好的避免这个问题。

主要遇到的问题就是这几个了，其实过程中还遇到了很多其他的小问题，在这里就不一一赘述了。

6.2 学到的内容

通过本次程序设计，我学会了多文件编写程序，不再单一的将程序全部都集中在一个 cpp 文件中，我学会了合理利用头文件、源文件、以及函数的意义、类的意义，将程序结构划分的简洁清楚。

其次，我清楚的掌握了 Prim 算法以及 Kruskal 算法的核心原理，并且能够以多种方法用代码实现他们，对图的存储结构也有了更多元化的认识，不仅仅是邻接表、邻接矩阵，期间还尝试用其他编程

语言如 python 的字典结构去实现图的结构。

通过本次程序设计，我也对 c++的类和面向对象以及指针、函数等的应用更加熟练，包括类外函数声明，多文件之间如何调用等问题，通过这次实践，大大提高了我对他们的认识。并且也对我所使用的编程工具 vscode 有了更多的认识。

6.3 未来完善

本次程序设计功能虽然较为完善，但个人觉得 UI 界面不是很美观，正在学习一些 UI 开发内容，但由于时间有限，目前只能使用 c++的 QT 开发一些简单的界面，未来会为该项目加上一个合理的 UI 界面，如果有条件，我也会尝试用前端框架为该项目编写一个界面，以方便用户使用。

至于功能模块，预计未来会越来越完善，为其增添其他功能，比如线路信息顶点的命名目前支持数字，不支持其他格式，未来会新建一个函数完成其他格式之间与数字之间的转换。使应用更为广泛，不断完善使其成为一个更加全面的功能体，作为在处理最小生成树问题的一个快捷库。

7. 参考文献

[1].《数据结构》 主编：严蔚敏　　2013 年 9 月第 1 版　人民邮电出版社

[2]. 韩丽霞,王宇平.求解度约束最小生成树的新的遗传算法[J].计算机工程与应用,2006,42(31):13-15.

[3].段智,袁振洲.基于 Prim 算法的农村公路网布局重要度最大树求解方法[J].公路,2007(5):111-114.

[4].李国奇.基于堆的无向带权图最小生成树的 PRIM 方法[J].电脑学习,2007(6):61-62.

[5].刘平原,张霓.普里姆(Prim)算法另解[J].科学中国人,2007(7):125-126.

[6].董跃华,李云浩,姜在东.用破圈法实现普里姆算法[J].江西理工大学学报,2008,29(4):20-22.

8. 附录（代码清单）

主函数入口——main.cpp

```
#include"graph.cpp"
int n=MaxV;int G[MaxV][MaxV];   //n 为顶点数，MaxV 为最大顶点数  G 为邻接矩阵
int vis[MaxV] = {0};    //标记数组，vis[i]==true 表示已访问。初始值均为 false
Edge edge_prim[num]; //边集数组
int edge_num;//边数

int main()
{
    int choice=99;
     while(choice!=0)//按 0 退出
     {
     std::cout<<"-----电网建设造价设计-----"<<endl;
     Solution A;//创建一个对象 A
     std::cout<<"请选择你要使用的算法"<<endl;
     std::cout<<"1. prim 算法"<<endl;
     std::cout<<"2. krusKal 算法"<<endl;
     std::cout<<"3. KrusKal 算法测试"<<endl;
     std::cout<<"4. prim 算法测试"<<endl;
     std::cout<<"5. 按 0 退出"<<endl;
     cin>>choice;
     switch (choice)
     {
     case 1:
         std::cout<<"请输入小区个数(顶点数)"<<endl;cin>>n;A.vex=n;
         std::cout<<"请输入小区之间的路径数(边数)"<<endl;cin>>edge_num;
         //初始化邻接矩阵和边集数组
         A.initial_adj_matrix(edge_prim,n,edge_num);
         //创建邻接矩阵和边集数组
         A.create_adj_matrix(edge_prim,edge_num,n);
         A.adj_prim(vis,n,A.adj_matrix,edge_prim);
         break;
     case 2:
         std::cout<<"请输入小区个数(顶点数)"<<endl;cin>>n;A.vex=n;
         std::cout<<"请输入小区之间的路径数(边数)"<<endl;cin>>edge_num;
         //初始化邻接矩阵和边集数组
         A.initial_adj_matrix(edge_prim,n,edge_num);
         //创建边集数组
         A.create_adj_pn(A.adj_matrix,edge_num,n);
         A.adj_KrusKal(A.adj_pn,A.adj_matrix,edge_num);
```

```
        break;
    case 3://测试 KrusKal 算法
        A.G(A.adj_matrix,edge_prim);//测试数据函数
        A.adj_KrusKal(edge_prim,A.adj_matrix,10);//调用 Kruskal 算法函数
        break;
    case 4://测试 prim 算法
        A.G(A.adj_matrix,edge_prim);//测试数据函数
        A.adj_prim(vis,n,A.adj_matrix,edge_prim);//调用 Prim 算法函数
        break;
    default:
        break;
    }
    }
}
```

头文件——graph.h

```
//设置最大顶点数
const int MaxV=7;
const int num=20;

//设置最大权重
//默认最大权值即无法到到达
const int MaxValue=99;

//设置边的数组元素类型
typedef struct
{
    int start_vex;int stop_vex;//边的起点和终点
    int weight;//边的权
}Edge;

//设置一个图类定义
class Solution
{
    public://公有属性
    int edge_num;//边数
    int adj_matrix[MaxV][MaxV];//邻接矩阵
    Edge adj_pn[num];
    int vex=MaxV;//顶点数


    public://公有属性
    //1.初始化邻接矩阵(边集合数组，顶点数，边数)
    int initial_adj_matrix(Edge*pn,int node,int edge);

```

```
    //2.创建一个检查函数：(起点，终点，权重，顶点数)
    void check(int i,int j,int w,int node);

    //3.创建邻接矩阵(边集数组结构体指针，边数，顶点数)
    void create_adj_matrix(Edge*pn,int edge,int node);

    //4.将边集数组从小到大排序(边集数组，边的个数)
    void sort_edge(Edge*pn,int edge);

    //5.prim 算法实现图的最小生成树(访问数组，顶点数，邻接矩阵，边集数组)
    void adj_prim(int vis[MaxV],int n,int G[MaxV][MaxV],Edge edge_prim[MaxV]);

    //6.邻接矩阵测试图数据函数(邻接矩阵)
    void G(int G[MaxV][MaxV],Edge*pn);

    //7.KrusKal 算法(边集数组，邻接矩阵，边数)
    void adj_KrusKal(Edge* pn,int adj_matrix[MaxV][MaxV],int edge);

    //8.判断是否围成圈(标记访问数组，遍历索引)
    int get_end(int vends[], int i);

    //9.创建边集数组（Kruskal）(邻接矩阵，边数，顶点数)
    void create_adj_pn(int G[MaxV][MaxV],int edge_num,int node);
};
```

源文件——graph.cpp

```
#include "graph.h"
#include<iostream>
#include<string.h>
using namespace std;
//1.初始化邻接矩阵和边集数组,邻接矩阵都不可达，边集数组权重都设置为0
int Solution ::initial_adj_matrix(Edge*pn,int node,int edge)
{
    for(int i=0;i<node;i++)//i 表示行
    {
        for(int j=0;j<node;j++)//j 表示列
        {
            if(i==j) adj_matrix[i][j]=0;//自身到自身设置为0
            else adj_matrix[i][j]=MaxValue;//初始化均设置为最大权值
        }
    }
    for(int k;k<edge;k++)
    {
        pn[k].weight=0;//将每条边的权值都设置为0
    }
```

```
    return 0;
}

/*2.创建一个检查函数：(起点，终点，权重，顶点数)
用于检查输入的数字是否符合要求*/
void Solution::check(int i,int j,int w,int node)
{
    while(true)
    {
        if(i<0||j<0||i>node||j>node||w>MaxValue||w<0)//设置数据异常的情况
        std::cout<<"输入错误，请重新输入"<<endl;
        else return;//数据正常就可以返回了
        cin>>i>>j>>w;//数据输入错误就继续输入
    }
}

//3.创建邻接矩阵,pn 为边集结构体数组指针,0 和 99 的都不用输入
void Solution ::create_adj_matrix(Edge*pn,int edge,int node)
{
    std::cout<<"请依次输入权重不为 0 的边的起点，终点，权重："<<endl;
    std::cout<<"直到输入一条权重为 0 的边后退出输入"<<endl;
    int a;int b;int c;
    do
    {
        int k=0;
        cout<<"温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)"<<endl;
        cin>>a>>b>>c;//输入起点，终点，权重
        check(a,b,c,node);
        if(c==0) break;//如果输入的权重为 0，则停止
        else{
             adj_matrix[a][b]=c;//将输入的权重赋值给邻接矩阵
        adj_matrix[b][a]=c;
        k++;
        }

    }while(true);

    std::cout<<"创建的邻接矩阵为："<<endl;
    for(int i =0;i<node;i++)
    {
        for(int j=0;j<node;j++)
        {
            std::cout<<adj_matrix[i][j]<<" ";
        }
        std::cout<<endl;
```

```
    }
}

//4.将边集数组从小到大排序(边集数组，边的个数)
void Solution ::sort_edge(Edge*pn,int edge)
{
    for(int i=0;i<edge-1;i++)//一共 edge-1 躺
    {
        for(int k=0;k<edge-i-1;k++)//每趟的比较次数
        {
            Edge tmp;
            if(pn[k].weight>=pn[k+1].weight)//结构体排序
            {
                tmp=pn[k];
                pn[k]=pn[k+1];
                pn[k+1]=tmp;
            }
        }
    }
}


//5.prim 算法
void Solution::adj_prim(int vis[MaxV],int n,int G[MaxV][MaxV],Edge edge_prim[MaxV])
{
    vis[0]=1; int flag=0; int weight=99;int t=0;int weight_sum=0;//(t 用于记录 j)
    for(int m=0;m<MaxV;m++)
    {
        weight=99;
        for(int i=0;i<n&&flag<MaxV;i++)//每一行的
        {
            if(vis[i]==1)//代表其已经访问
            {
                for(int j=0;j<n&&flag<=MaxV;j++)//每一列的
                {
                        if (vis[j]==0&&G[i][j]<weight)//代表其未访问
                        {
                            edge_prim[flag].weight=G[i][j];
                            edge_prim[flag].start_vex=i;
                            edge_prim[flag].stop_vex=j;
                            t=j;
                            weight=G[i][j];
                        }
                    }
                }
```

```
110.                 else continue;
111.
112.             }
113.         if(weight==99)
114.             continue;//说明它无法构成最小生成树
115.             else
116.             {
117.                 vis[t]=1;
118.                 flag++;
119.             }
120.         }
121.         // std::cout<<"Prim 算法的最小生成树为（未排序）"<<endl;
122.         // for(int k=0;k<flag;k++)//flag 初始化为 1，所以从 1 开始输出
123.         // {
124.         //  std::cout<<"("<<edge_prim[k].start_vex<<","<<edge_prim[k].stop_vex<<","<<edge_pr
    im[k].weight<<")"<<" ";
125.         //  weight_sum=weight_sum+edge_prim[k].weight;
126.         // }
127.         // std::cout<<"prim 算法所得最小权值之和为"<<weight_sum<<endl;
128.         sort_edge(edge_prim,flag);
129.         std::cout<<"Prim 算法的最低造价电网图为（按“权”排序之后）"<<endl;
130.         weight_sum=0;
131.         for(int k=0;k<flag;k++)
132.         {
133.             std::cout<<"("<<edge_prim[k].start_vex<<","<<edge_prim[k].stop_vex<<","<<edge_pr
    im[k].weight<<")"<<" ";
134.             weight_sum=weight_sum+edge_prim[k].weight;
135.         }
136.         std::cout<<"prim 算法的最低造价电网和为：“<<weight_sum<<endl;
137.     }
138.
139.     //6.图的测试数据函数
140.     void Solution ::G(int G[MaxV][MaxV],Edge*pn)
141.     {
142.         fill(G[0],G[0]+7*7,99);
143.         G[1][0]=50;G[0][1]=50;pn[0].start_vex=1;pn[0].stop_vex=0;pn[0].weight=50;
144.         G[2][0]=60;G[0][2]=60;pn[1].start_vex=2;pn[1].stop_vex=0;pn[1].weight=60;
145.         G[3][1]=65;G[1][3]=65;pn[2].start_vex=3;pn[2].stop_vex=1;pn[2].weight=65;
146.         G[3][2]=52;G[2][3]=52;pn[3].start_vex=3;pn[3].stop_vex=2;pn[3].weight=52;
147.         G[4][1]=40;G[1][4]=40;pn[4].start_vex=4;pn[4].stop_vex=1;pn[4].weight=40;
148.         G[4][3]=50;G[3][4]=50;pn[5].start_vex=4;pn[5].stop_vex=3;pn[5].weight=50;
149.         G[5][3]=30;G[3][5]=30;pn[6].start_vex=5;pn[6].stop_vex=3;pn[6].weight=30;
150.         G[5][4]=70;G[4][5]=70;pn[7].start_vex=5;pn[7].stop_vex=4;pn[7].weight=70;
151.         G[6][2]=45;G[2][6]=45;pn[8].start_vex=6;pn[8].stop_vex=2;pn[8].weight=45;
152.         G[6][3]=42;G[3][6]=42;pn[9].start_vex=6;pn[9].stop_vex=3;pn[9].weight=42;
```

```
153.      for(int i =0;i<7;i++)
154.      {
155.          for(int j=0;j<7;j++)
156.          {
157.              if(i==j)
158.              G[i][j]=0;
159.              else continue;
160.          }
161.      }
162.      std::cout<<"测试图的邻接矩阵为: "<<endl;
163.      for(int i =0;i<7;i++)
164.      {
165.          for(int j=0;j<7;j++)
166.          {
167.              std::cout<<G[i][j]<<" ";
168.          }
169.          std::cout<<endl;
170.      }
171.  }
172.  //7.判断是否成圈
173.  int Solution::get_end(int vends[], int i)
174.  {
175.      while (vends[i] != 99)
176.      i = vends[i];//如果该位置有人，则找到与它同一个阵营的位置去看看，直到那个阵营没人。
177.      //std::cout<<i<<" ";
178.      return i;
179.  }
180.
181.
182.  //8.KrusKal 算法
183.  void Solution::adj_KrusKal(Edge* pn,int adj_matrix[MaxV][MaxV],int edge)
184.  {
185.      //std::cout<<"开始执行 KrusKal 算法"<<endl;
186.      int i,m,n,p1,p2;
187.      int length;
188.      int index = 0;          // rets 数组的索引
189.      int vends[MaxV];     // 用于保存"已有最小生成树"中每个顶点在该最小树中的终点。
190.      std::fill(vends,vends+MaxV,99);
191.      Edge rets[MaxV];        // 结果数组，保存 kruskal 最小生成树的边
192.      // 将边按照"权"的大小进行排序(从小到大)
193.      sort_edge(pn, edge);
194.
195.      for (i=0; i<edge; i++)
196.      {
```

```
197.          //std::cout<<"("<<pn[i].start_vex<<","<<pn[i].stop_vex<<","<<pn[i].weight<<")"<<
    " ";// 输出所有路径
198.          p1 = pn[i].start_vex;   // 获取第 i 条边的"起点"的序号
199.          p2 = pn[i].stop_vex;     // 获取第 i 条边的"终点"的序号
200.
201.          m = get_end(vends, p1);                 // 获取 p1 在"已有的最小生成树"中的终点
202.          n = get_end(vends, p2);                 // 获取 p2 在"已有的最小生成树"中的终点
203.          // 如果 m!=n，意味着"边 i"与"已经添加到最小生成树中的顶点"没有形成环路
204.          if (m != n)
205.          {
206.              //std::cout<<"("<<m<<","<<n<<")"<<endl;
207.              vends[m] = n;                       // 设置 m 在"已有的最小生成树"中的终点为 n
208.              rets[index++] = pn[i];            // 保存结果
209.          }
210.      }
211.
212.      // 统计并打印"kruskal 最小生成树"的信息
213.      length = 0;
214.      for (i = 0; i < index; i++)
215.          length += rets[i].weight;
216.      std::cout<<"Kruskal 算法的最低造价为"<<length<<endl;
217.      std::cout<<"KrusKal 算法的最低造价电网图为（按“权”从小到大排序）"<<endl;
218.      for (i = 0; i < index; i++)
219.          std::printf("(%d,%d,%d) ", rets[i].start_vex, rets[i].stop_vex,rets[i].weight);

220.      std::printf("\n");
221.  }
222.  //9. 创建边集数组
223.  void Solution:: create_adj_pn(int G[MaxV][MaxV],int edge_num,int node)
224.  {
225.      std::cout<<"请依次输入权重不为 0 的边的起点，终点，权重："<<endl;
226.      std::cout<<"直到输入一条权重为 0 的边后退出输入"<<endl;
227.      int flag=0;int a;int b;int c=99;
228.      for(int i=0;i<edge_num;i++)
229.      {
230.          cout<<"温馨提示：输入边的顶点以及权重必须以数字命名：(样例输入：1 2 3)"<<endl;
231.          cin>>a>>b>>c;
232.          check(a,b,c,node);
233.          if(c==0){break;}
234.          else
235.          {
236.              adj_pn[flag].start_vex=a;
237.              adj_pn[flag].stop_vex=b;
238.              adj_pn[flag].weight=c;
239.              flag++;
```

```
240.            }
241.
242.        }
243.        std::cout<<"创建的边集数组为：："<<endl;
244.        for(int k=0;k<edge_num;k++)
245.        {
246.            std::cout<<"("<<adj_pn[k].start_vex<<","<<adj_pn[k].stop_vex<<","<<adj_pn[k].wei
    ght<<")"<<" ";
247.        }
248.        std::cout<<endl;
249.    }
```