**数据结构课程设计**

**项目说明文档**

**电网建设造价模拟系统**

作 者 姓 名： 苏家铭

学 号： 2151299

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

**Tongji University**



**目录**

[1 项目分析 1](#_Toc24363)

[1.1 项目背景 1](#_Toc14852)

[1.2项目要求 1](#_Toc23242)

[1.2.1 功能要求 1](#_Toc3377)

[1.2.2 项目实例 1](#_Toc32195)

[2 项目设计及实现 2](#_Toc749)

[2.1 数据结构设计思路 2](#_Toc25667)

[2.2 类设计 2](#_Toc19827)

[2.2.1 图 2](#_Toc2872)

[2.2.2 堆 5](#_Toc17178)

[2.2.3 最小生成树 6](#_Toc28758)

[2.3 项目算法 8](#_Toc12461)

[2.3.1 实现思路 8](#_Toc888)

[2.3.2 代码实现 12](#_Toc19410)

[3 项目测试 14](#_Toc31744)

[3.1 创建电网顶点 14](#_Toc11635)

[3.2 添加边测试 15](#_Toc31251)

[3.3 构造最小生成树(prim) 16](#_Toc15480)

[3.4 显示最小生成树 16](#_Toc23276)

[3.5 构造最小生成树(kruscal) 17](#_Toc29530)

[3.6 显示最小生成树(kruscal) 18](#_Toc8349)

[3.7 输入操作错误 18](#_Toc32605)

[4 算法性能分析 19](#_Toc4338)

[4.1 正确性 19](#_Toc28264)

[4.2 可使用性 19](#_Toc3157)

[4.3 可读性 19](#_Toc2264)

[4.4 效率 20](#_Toc25823)

[4.5 健壮性 20](#_Toc7880)

# 1 项目分析

## 项目背景

国家电网是中国非常有名的企业，我们日常生活中离不开电的使用，对于企业来说，如何铺设电网能最大限度地降低成本是一个非常值得深思的问题。铺设电线的成本降低了，人们在用电上的花费才能降低，幸福水平才能提高。由此观之，在铺设电路前找到一种最优解是一个利国利民、具有巨大社会效益的工作。

电网的各个点之间用线段连接起来，类似于图(graph)结构。在数据结构中，图是一种非线性的数据结构，利用好图的特性，可以帮助我们求出电网图中的最小生成树，以达到经济效益最大化的目的。

## 1.2项目要求

### 1.2.1 功能要求

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。请设计一个能够满足要求的造价方案。

在每个小区之间都可以设置一条电网线路，都要付出相应的经济代价。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。

### 1.2.2 项目实例



# 2 **项目设计及实现**

## 2.1 **数据结构设计思路**

根据电网的特性，给定若干地点，而后用若干线段将某些地点连接起来的结构与图相吻合。本项目中设计了图结构，而图分为有向图和无向图，对于电路的流通应当是双向流动，固采用无向图。

不仅如此，图的存储结构通常有邻接矩阵和邻接表两种方式（当然也有十字链表、邻接多重表、边集数组、逆邻接表等多种方式）。对于这两种存储方式的对比如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 存储结构 | 存储 | 遍历 | 查找 | Prim | Kruscal |
| 邻接表 | 适用于点稀疏 | O(n+e) | O(n+e) | O(n+e) |  |
| 邻接矩阵 | 适用于边稀疏 | O(n2) | O(n2) | O(n2) |  |

为了降低时间复杂度，本项目利用邻接表作为存储结构。

## 2.2 **类设计**

### 2.2.1 图

图是由顶点的有穷非空集合和顶点之间的边的集合组成，通常表示成：G=(V,E),其中G表示一个图，V是图G中顶点的集合，E是图G中边的集合。

图的存储方式一般为邻接矩阵和邻接表，本项目中选择邻接表作为图的存储结构，为了往后能实现两种存储方式又不用重复定义相同成员，本项目公有继承基类GRAFH，实现了派生类grafh。

GRAFH类数据成员：

int max\_vertices\_num;//图中最大顶点数

int edges\_num;//当前图中边数

int vertices\_num;//当前图中顶点数

Grafh类数据成员：

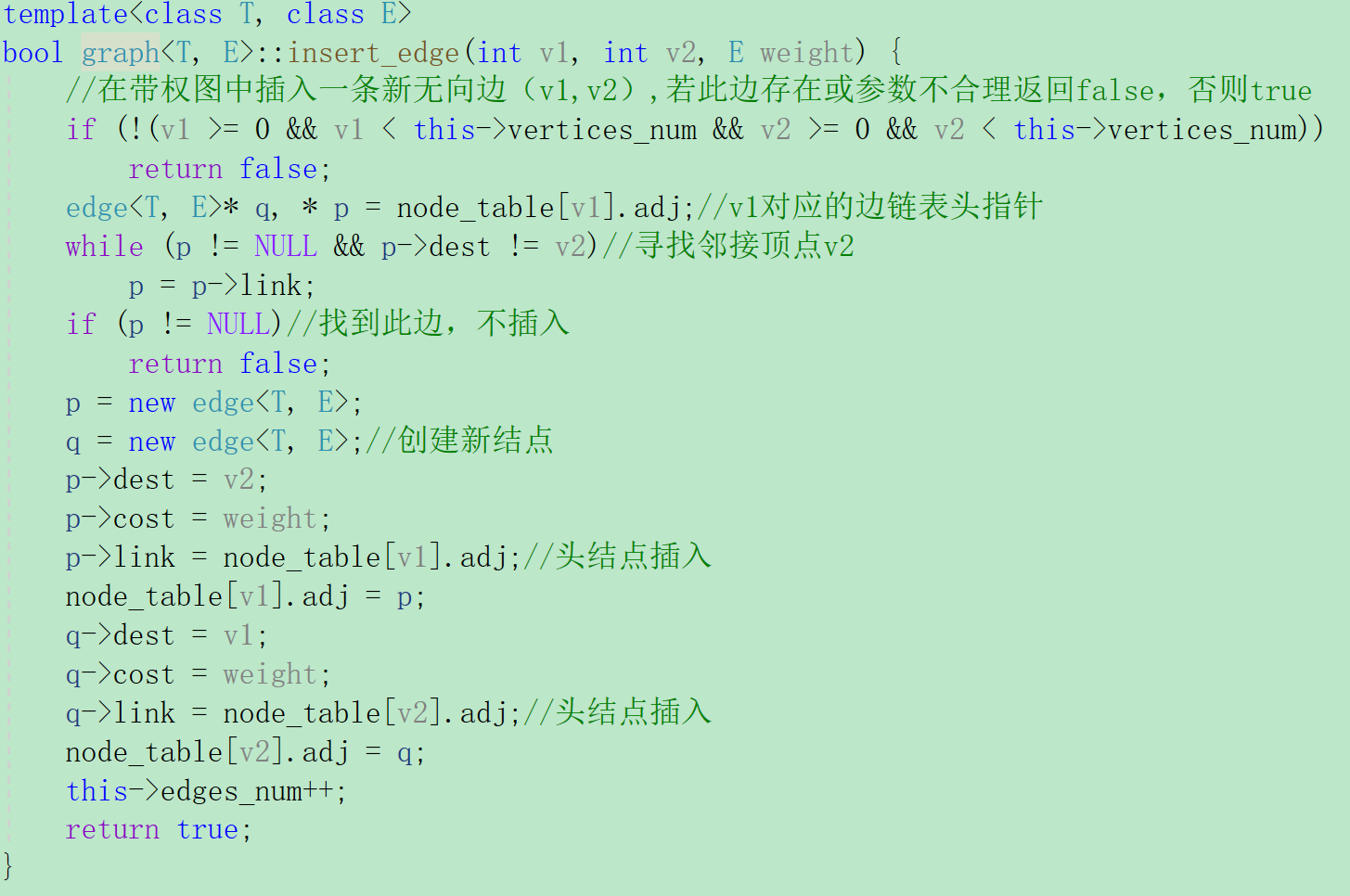
vertex<T, E>\* node\_table;//顶点表（各边链表的头结点）

与为了达到项目的要求，图实现了包括返回结点的值、返回边权、插入顶点、插入边（有向/无向）、删除顶点、删除边、左移/右移运算符重载等17种功能。

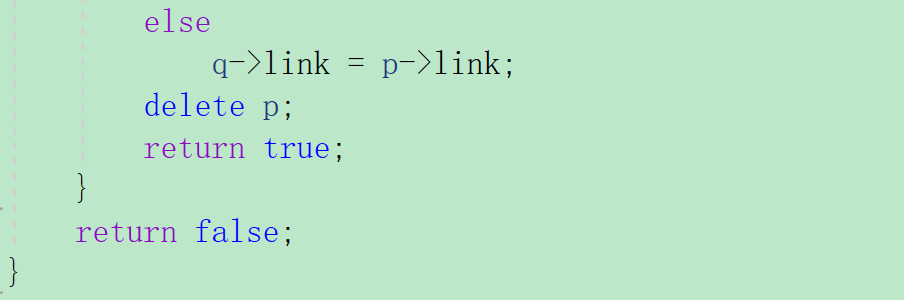
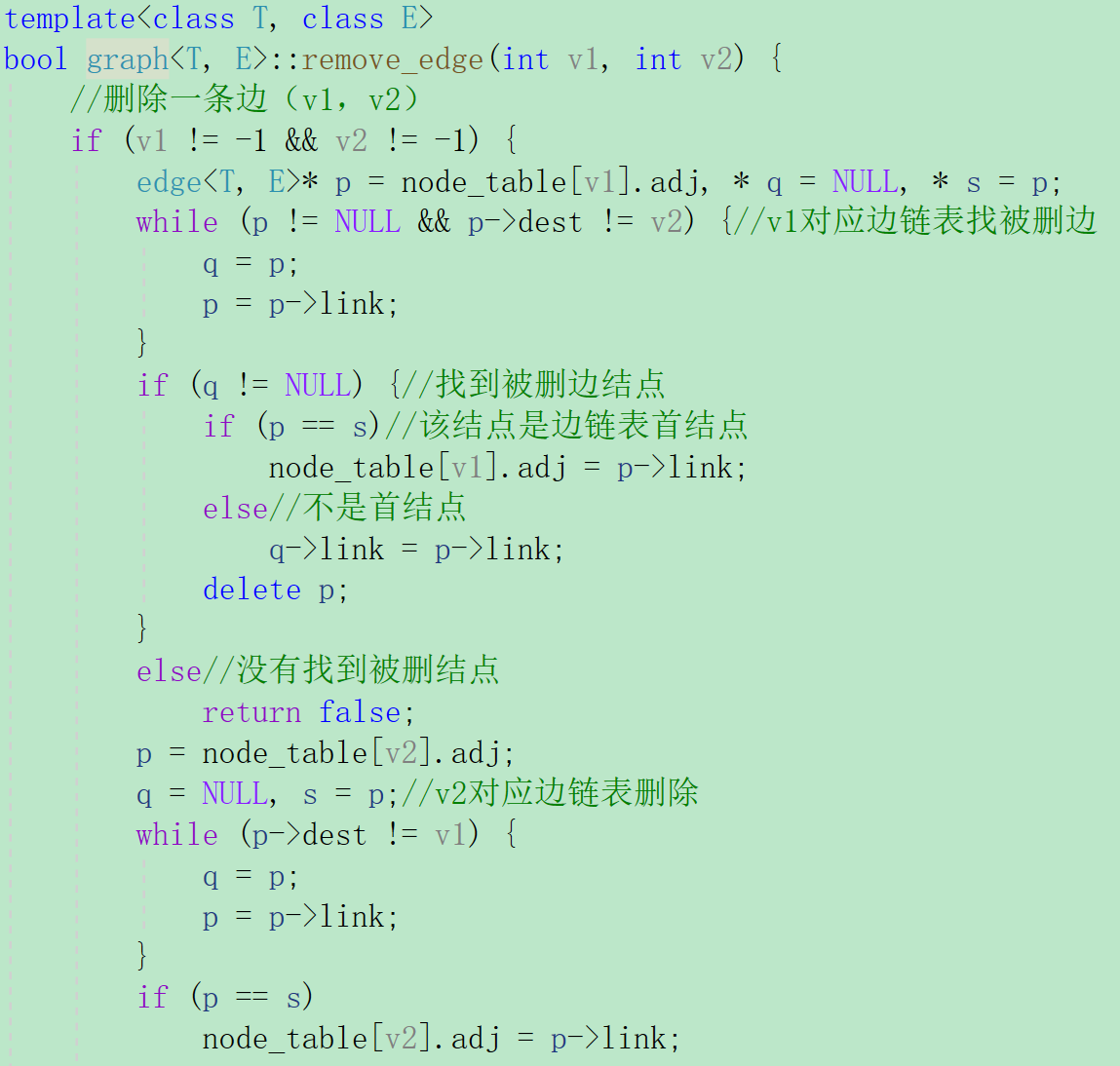
函数声明如下：

* bool empty()const;//判空
* bool full()const;//判满
* int vertices\_number(); //返回当前结点数
* int edges\_number();//返回当前边数量
* T get\_value(int i);//取位置为i的顶点中的值
* E get\_weight(int v1, int v2);//返回边（v1,v2）上的权值
* bool insert\_vertex(const T& vertex);//在图中插入一个顶点vertex
* bool remove\_vertex(int v);//在图中删除一个顶点v,此时v是顶点号
* bool insert\_edge(int v1, int v2, E cost);//在图中插入一条新无向边(v1,v2)
* bool insert\_directed\_edge(int v1, int v2, E cost);//在图中插入一条新有向边<v1,v2>
* bool remove\_edge(int v1, int v2);//在图中删除一条边(v1,v2)
* int get\_vertex\_pos(const T vertex);//返回顶点vertex在图中的位置
* int get\_first\_neighbor(int v);//取顶点v的第一个邻接顶点
* int get\_next\_neighbor(int v, int w);//取v的邻接顶点w的下一邻接顶点
* int get\_neighbor\_num(int v);//返回该点的邻接边数量
* friend istream& operator>>(istream& in,const graph& G);
* friend ostream& operator<< (ostream& out,const graph& G);

插入算法代码实现：



删除算法代码实现：



主要算法的时间复杂度：

* 插入顶点：O(n)
* 插入边：O(n+m)
* 删除顶点：O(n+m)
* 删除边：O(n+m)
* 返回顶点值：O(n)
* 返回边权：O(n+m)

其中，n为顶点数量，m为边数量

### 2.2.2 堆

堆(heap):若有一个关键码的集合K = {k0，k1，k2，…，kn-1}，把它的所有元素按[完全二叉树](https://so.csdn.net/so/search?q=%E5%AE%8C%E5%85%A8%E4%BA%8C%E5%8F%89%E6%A0%91&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/zgdlxs/article/details/_blank)的顺序存储方式存储在一个一维数组中，并满足：Ki <= K2i+1 且 Ki <= K2i+2，则称为小堆(或大堆)。将根节点最大的堆叫做最大堆或大根堆，根节点最小的堆叫做最小堆或小根堆。

本项目中堆基于一维数值来实现，通过定义建堆、调整堆来完成了一个堆。

最小堆类数据成员：

template <class T>

T\* pheap;//存放小根堆中元素的数组

int current\_size;//小根堆中当前元素个数

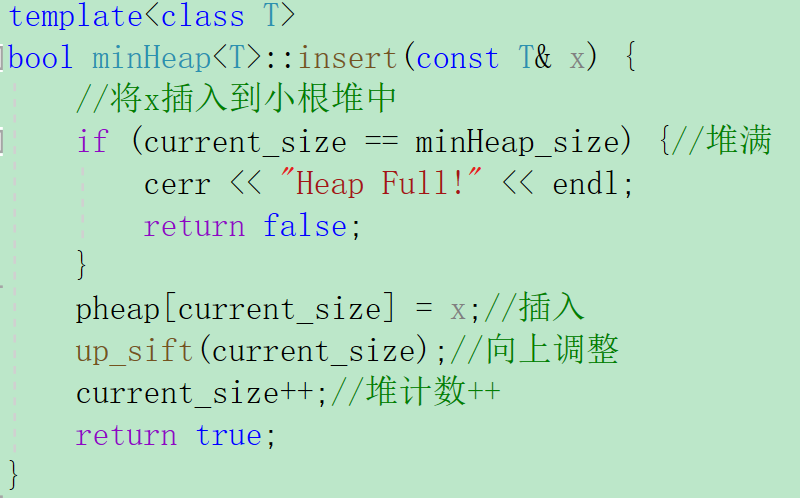
int minHeap\_size;//小根堆最多允许元素个数

该最小堆已经实现8种功能：向上调整、向下调整、插入元素、删除堆顶元素、判空、判满、置空堆、堆排序。

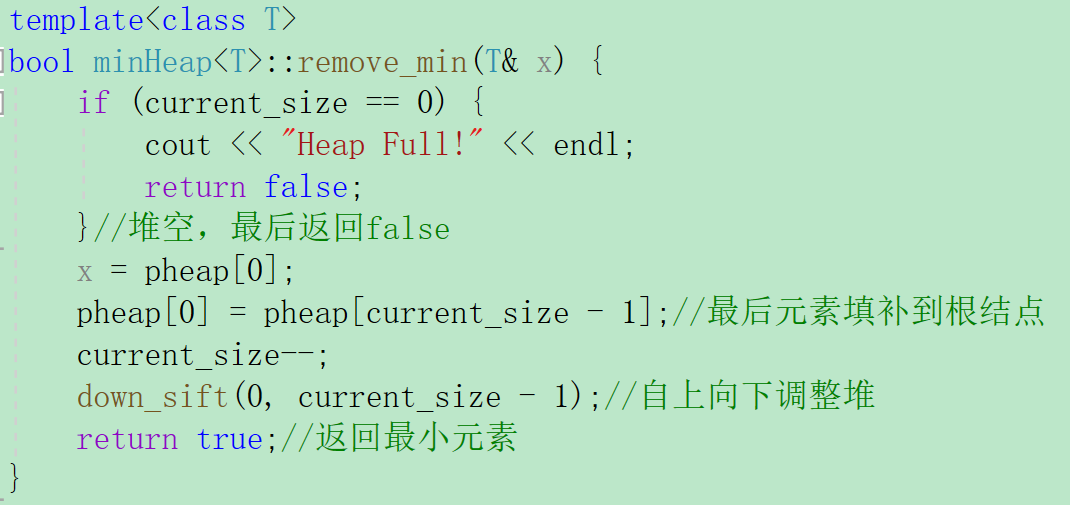
具体函数声明为：

* void down\_sift(int start, int m);//自上向下调整
* void up\_sift(int start);//自下向上调整
* void makeEmpty();//置空
* bool insert(const T& x);//将x插入到小根堆中
* bool remove\_min(T& x);//删除小根堆上的最小元素
* bool empty()const ;//判断堆是否为空
* bool full()const ;//判断堆是否达到最大限制个数
* void make\_empty();//置空堆
* void rank\_heap(T rank[]);//堆排序

插入算法代码实现：



删除算法代码实现：



主要算法的时间复杂度：

* 插入：O(log2n)
* 删除：O(log2n)
* 置空：O(n)
* 判断空否：O(1)

在最后为了能在优先级队列中访问堆的数据成员和成员函数，将优先级队列声明为友元函数：

//友元类

template <class E>

friend class minPQueue;

### 2.2.3 最小生成树

对于一个带权连通无向图G=(V,E)，生成树不同，每棵树的权（树中所有边上的权值和）也不同，设R为G的所有生成树的集合，若T为R中权值和最小的生成树，则T称为G的最小生成树（Minimum-Spanning-Tree，MST）。

最小生成树满意以下4个特性：

1、最小生成树可能有多个，但边的权值之和总是唯一且最小的

2、最小生成树的边数=定点数-1，砍掉一条则不连通，增加一条则会出现回路

3、若一个连通图本身就是一颗树，则其最小生成树就是它本身

4、只有连通图才有生成树，非连通图只有生成森林

本项目利用一维边值数组来表示最小生成树，树的边结点通过结构体struct储存。

边结点结构体数据成员：

int tail, head;//两顶点的位置

E key;//边上的权值，为关键码

为了能比较图的各个边权值的大小，在该结构体中重载了<、>、<=、>=、==五个运算符，函数声明如下：

friend bool operator<(const MST\_edge\_node& a, const MST\_edge\_node& b);

friend bool operator>(const MST\_edge\_node& a, const MST\_edge\_node& b);

bool operator==(const MST\_edge\_node& b);

bool operator<=(const MST\_edge\_node& b);

bool operator>=(const MST\_edge\_node& b);

最小生成树类MST的数据成员为：

MST\_edge\_node<T, E>\* edge\_value;//用边值数组表示树

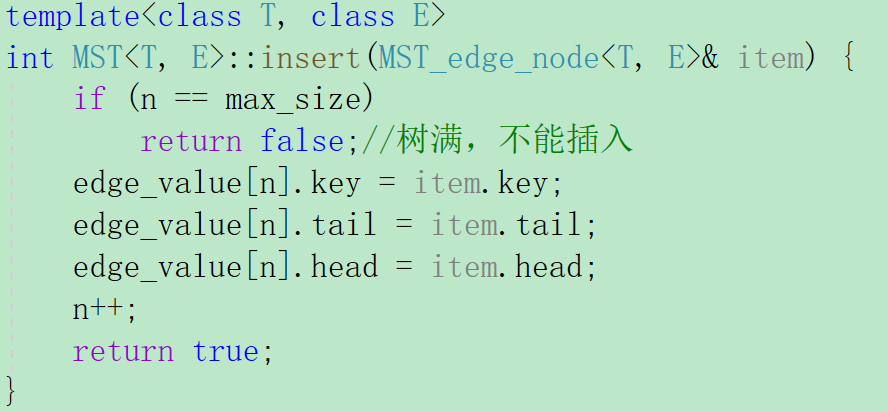
int max\_size, n;//数组的最大元素个数和当前个数

该最小生成树类完成了9项功能：插入边、扩大树、置空、返回最多可容纳边数、返回当前边数、返回边值数组首地址、返回第i条边的头顶点/尾顶点/权值。

具体函数声明为：

* int insert(MST\_edge\_node<T,E>& item);//插入一条边
* void enlarge(int size);//扩大最下生成树
* bool empty();//置空
* int get\_max\_size();//返回最大元素个数
* int get\_n();//返回当前元素个数
* MST\_edge\_node<T, E>\* get\_edge\_value();//返回边值数组的地址
* int get\_edge\_node\_tail(int i);//返回第i条边的尾
* int get\_edge\_node\_head(int i);//返回第i条边的头
* int get\_edge\_node\_key(int i);//返回第i条边的权值

插入算法代码实现：



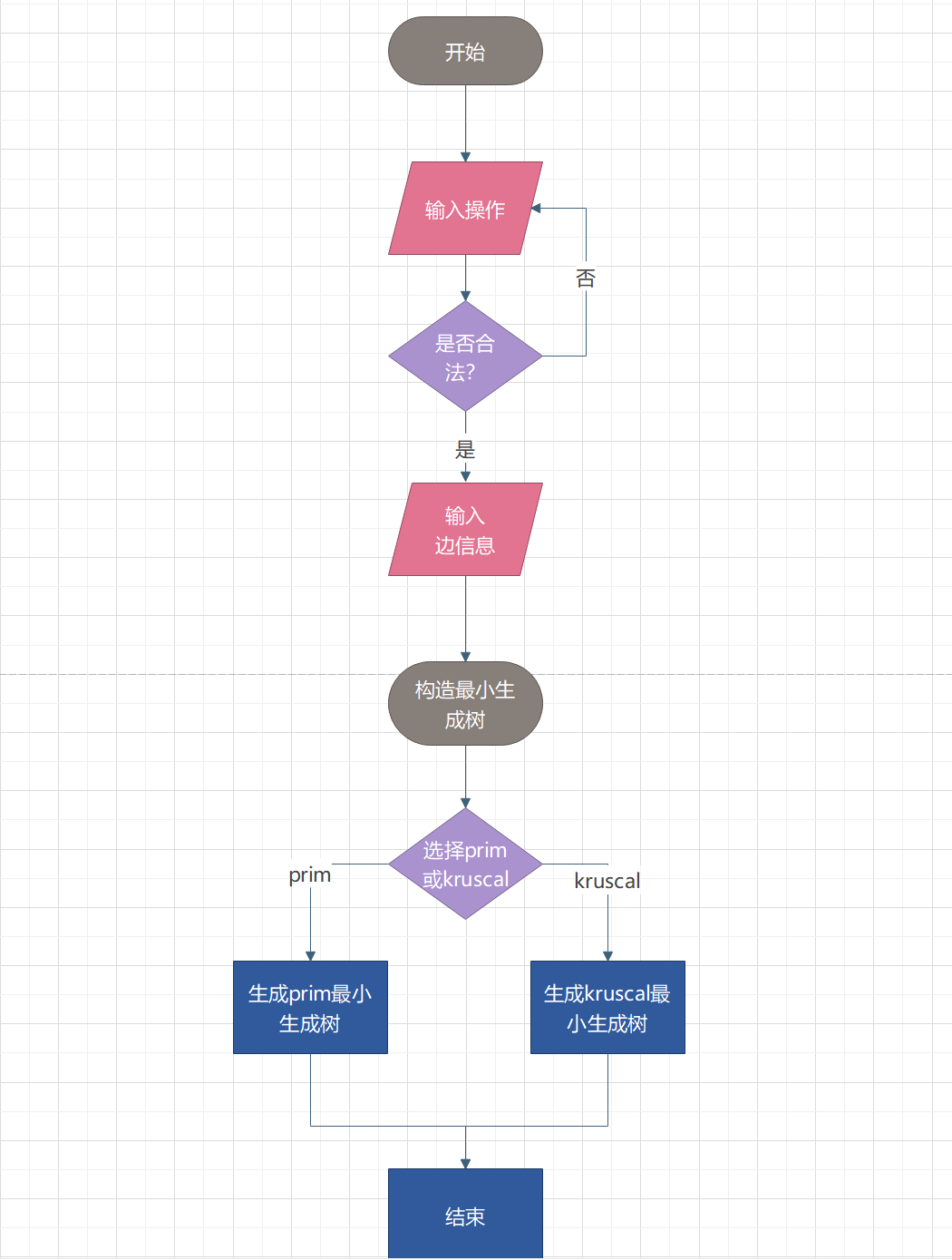
主要算法的时间复杂度：

* 插入：O(n)
* 返回边数据：O(n)

## 2.3 **项目算法**

### 2.3.1 实现思路

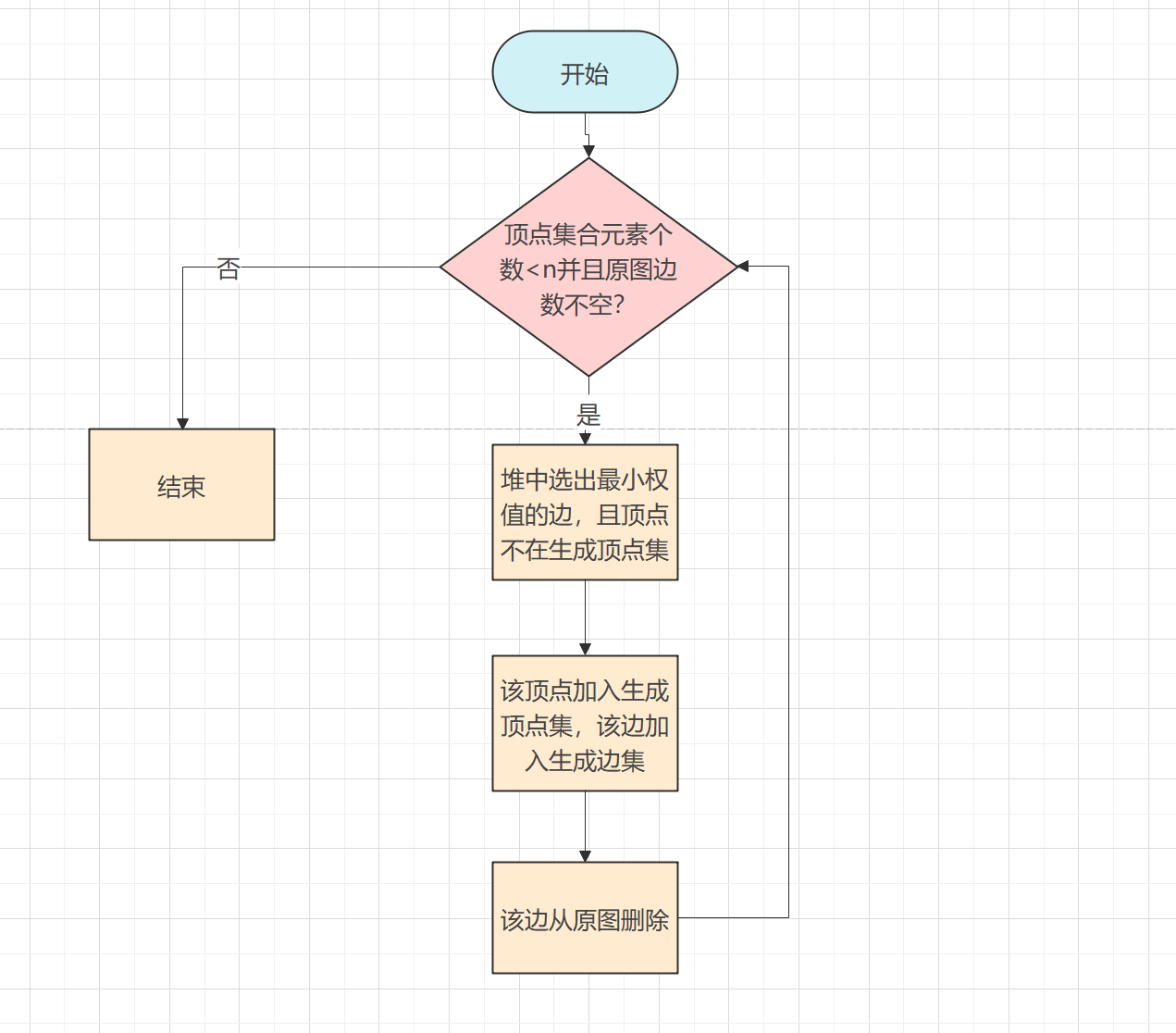
项目总体流程图如下：



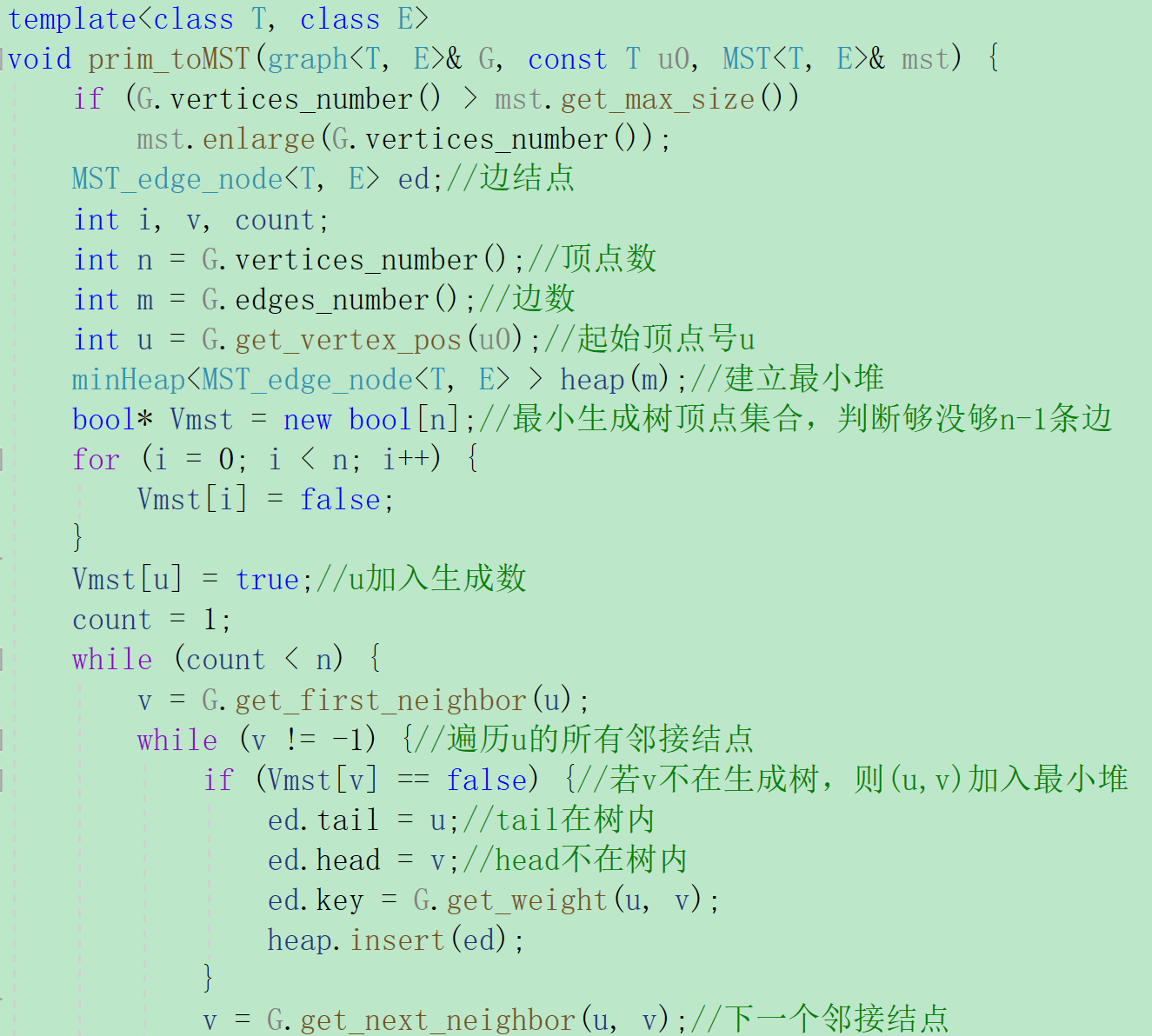
Prim生成最小生成树算法：

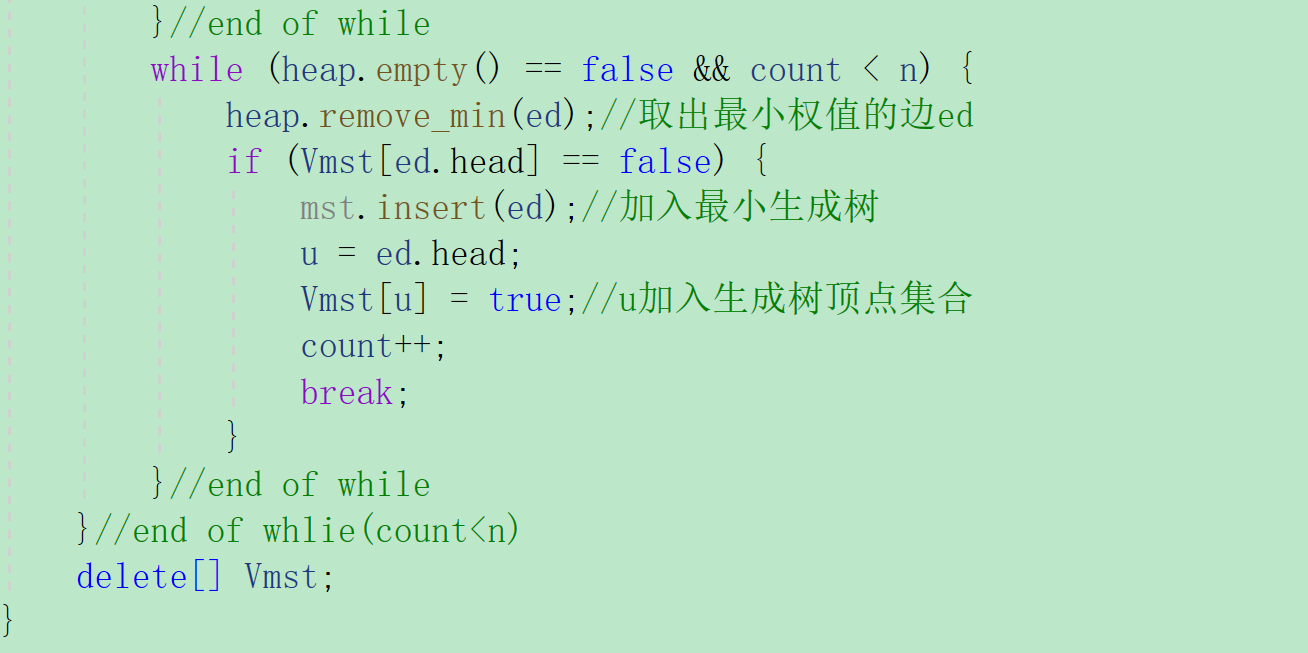
每次从最小堆中选出一个端点在生成树中，一个端点不在生成树中的权值最小的边(u,v)，加入生成树。然后再把以v为起始顶点的，不在生成树中的端点为另一个顶点的边加入最小堆。下一循环中再选取堆顶边，不断迭代，最后选出n个顶点，即可建立最小生成树。

流程图如下：



代码实现：

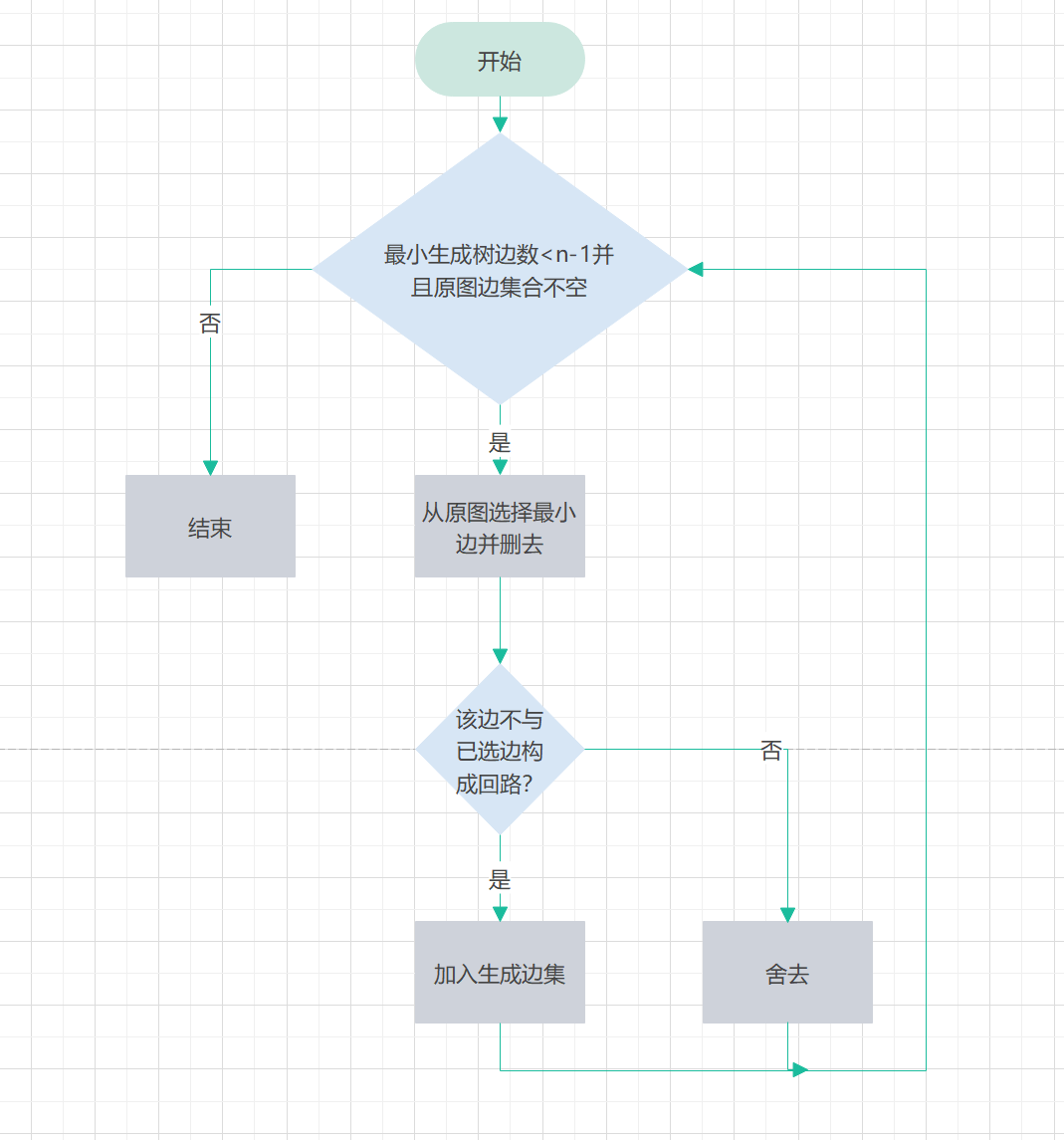


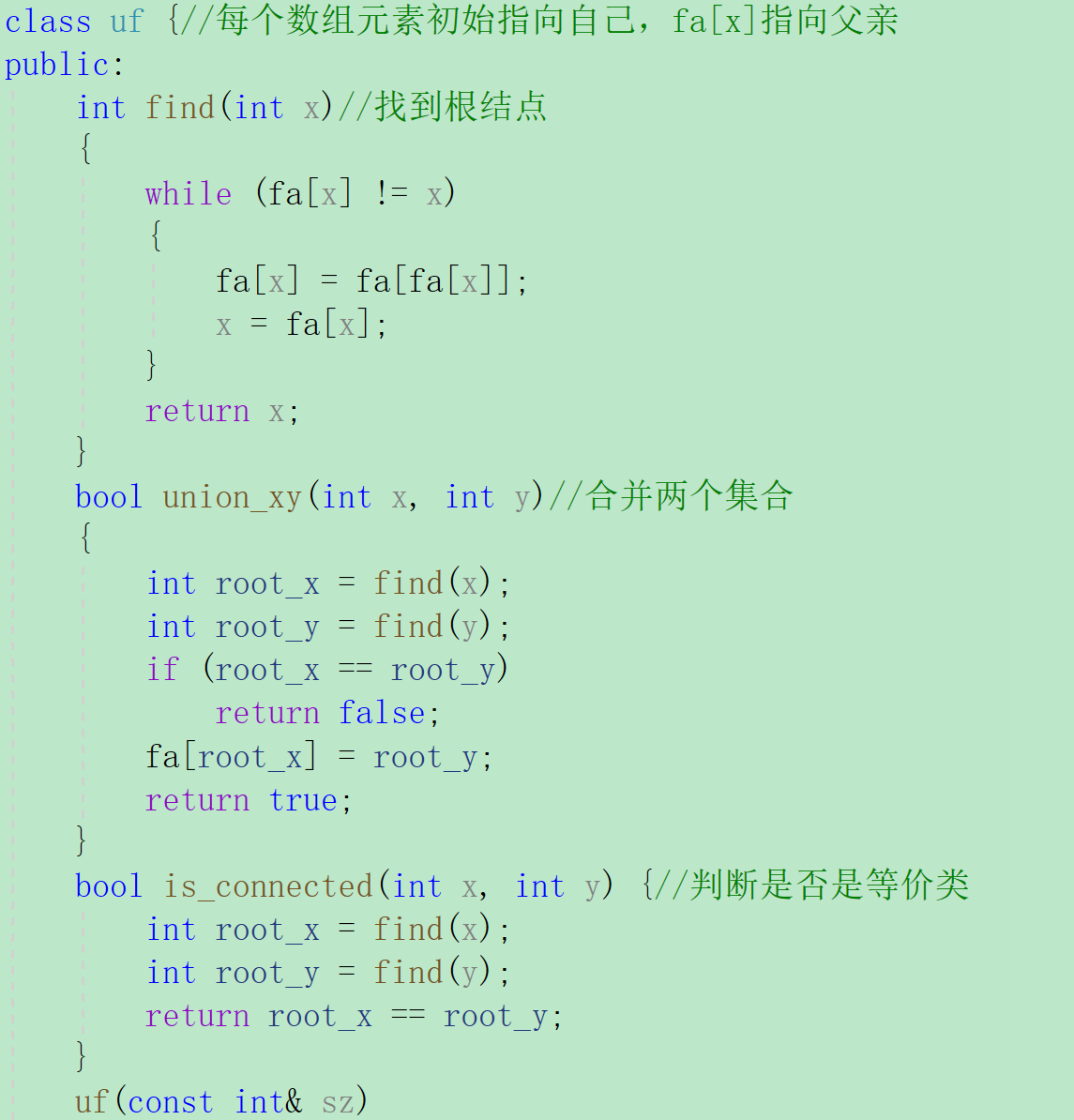


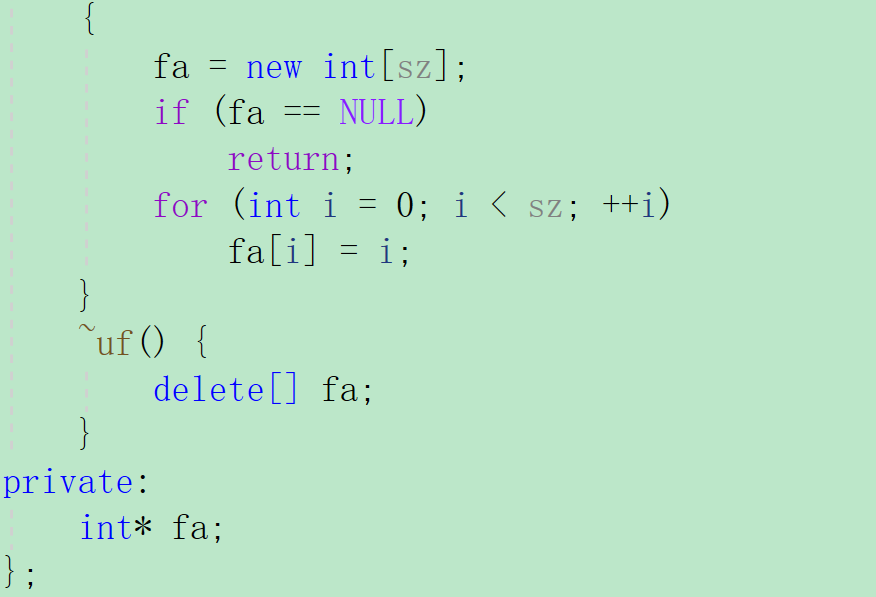
Kruscal生成最小生成树算法：

构造出只有n个顶点没有边的森林(图)，不断从原图中找最小权值的边，如果该边的两个顶点处于两个不同的连通分量，则加入生成图。重复n-1次即可构成一棵最小生成树。

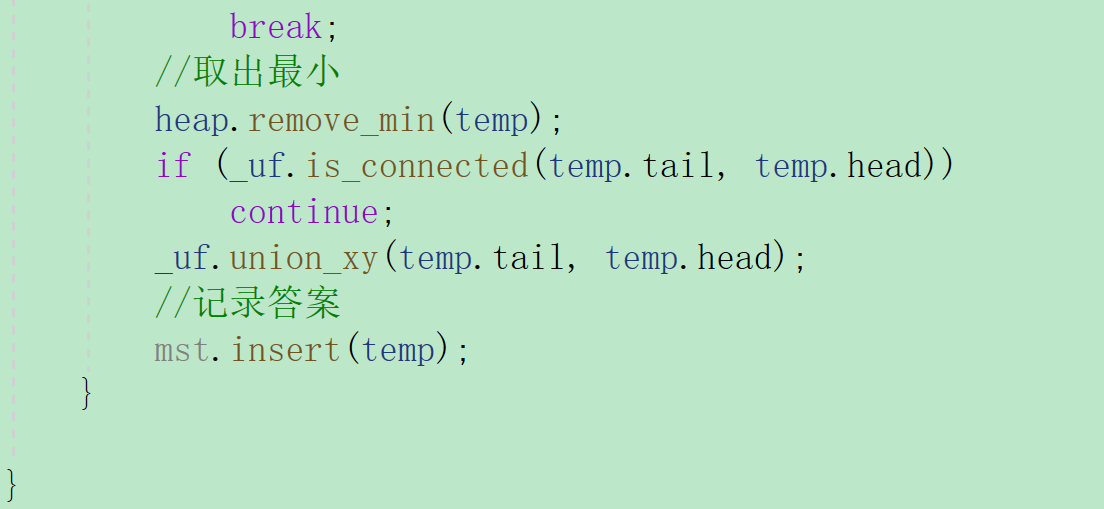
路程图如下：









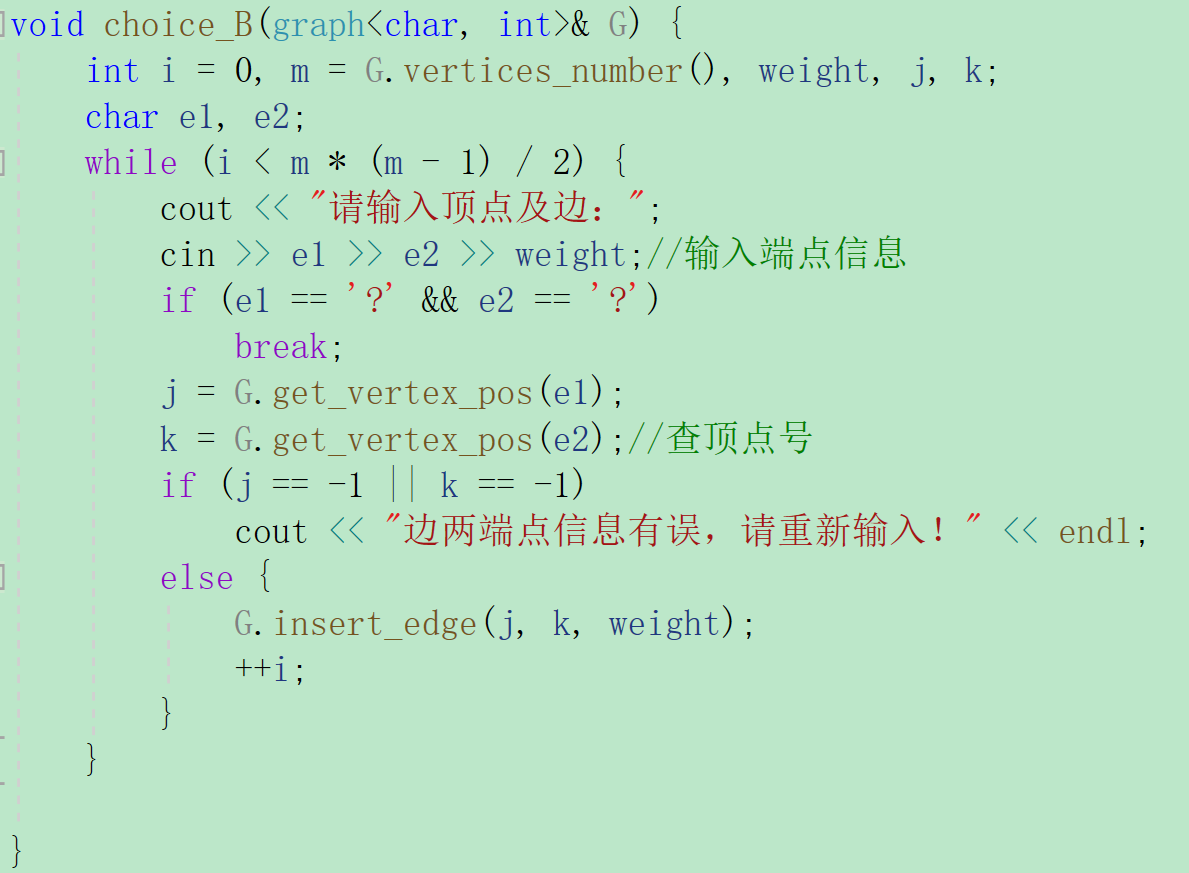


### 2.3.2 代**码实**现

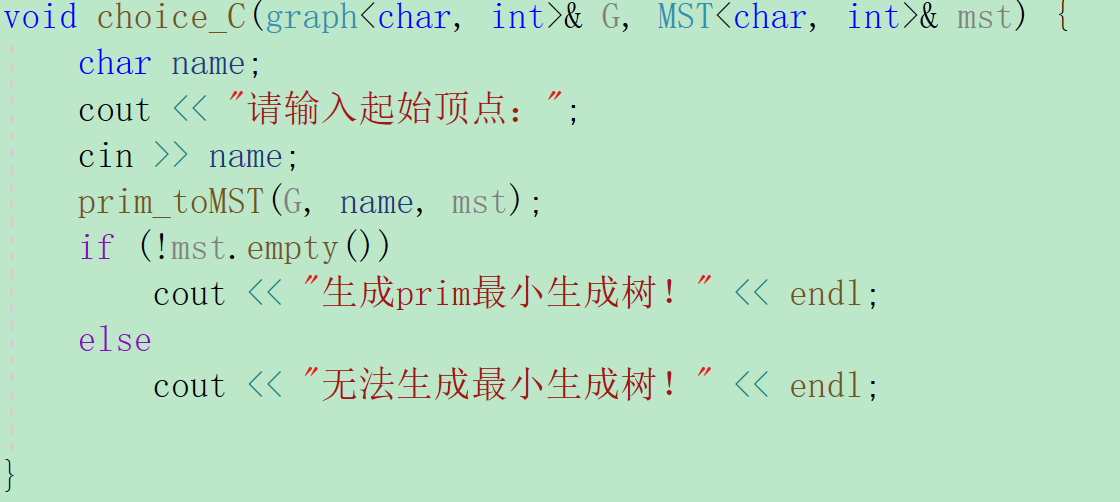
* 创建电网顶点：



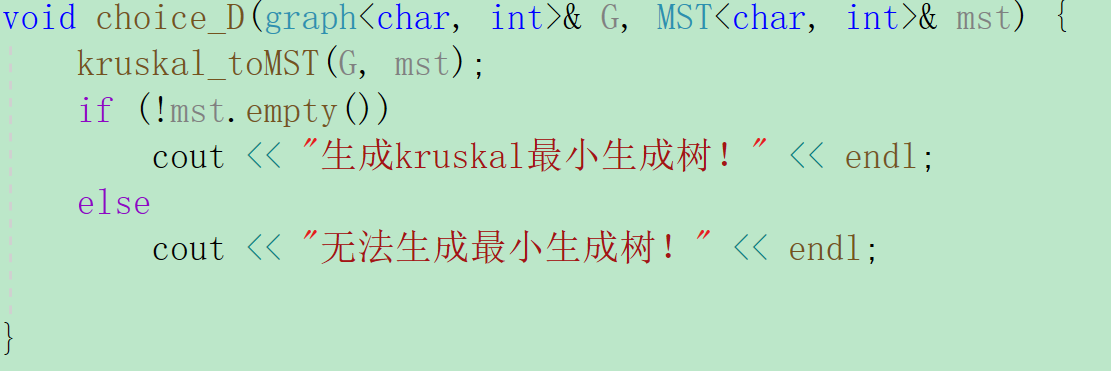
* 添加电网的边：



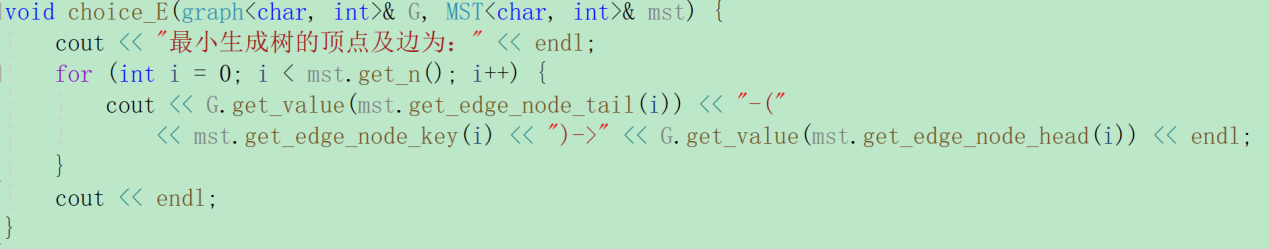
* 构造最小生成树(prim)：



* 构造最小生成树（kruscal）：



* 显示最小生成树：



# 3 项目测试

## 3.1 创建电网顶点

测试输入：

A

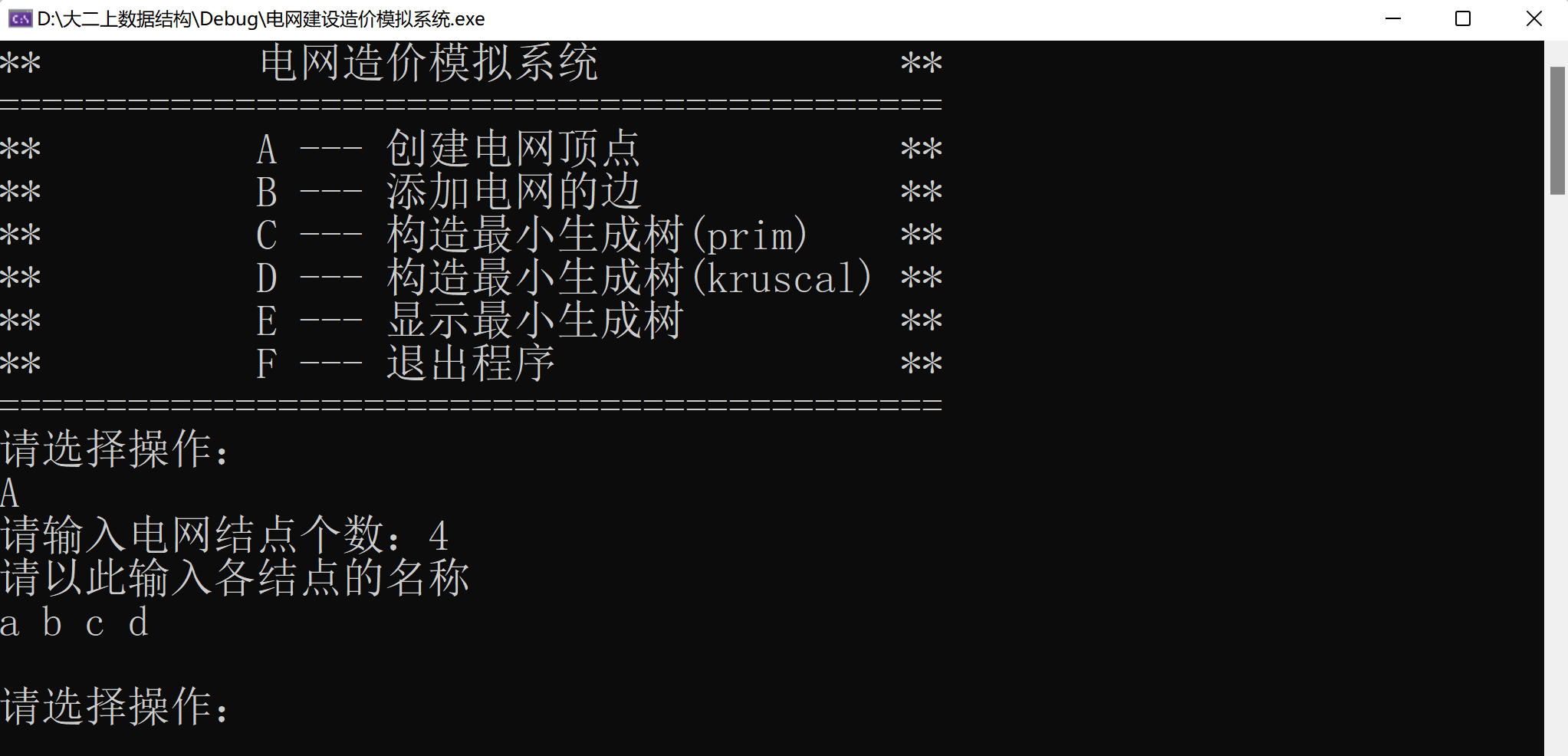
4

A b c d

预期结果

完成添加顶点

实验结果：



## 3.2 添加边测试

测试输入：

B

a b 8

b c 7

c d 5

d a 11

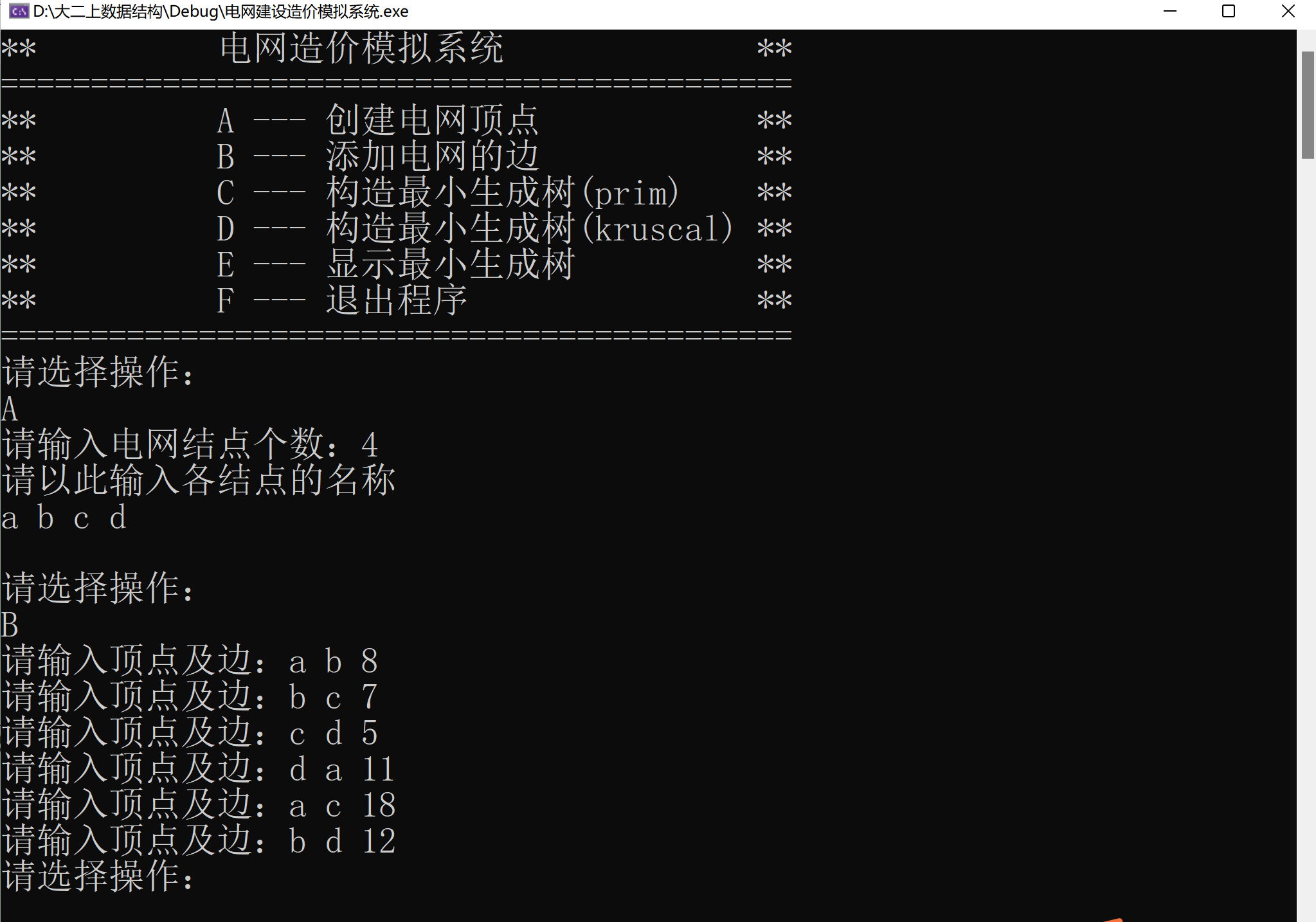
a c 18

b d 12

预期输出：

完成添加顶点

实验结果：



## 3.3 构造最小生成树(prim)

测试输入：

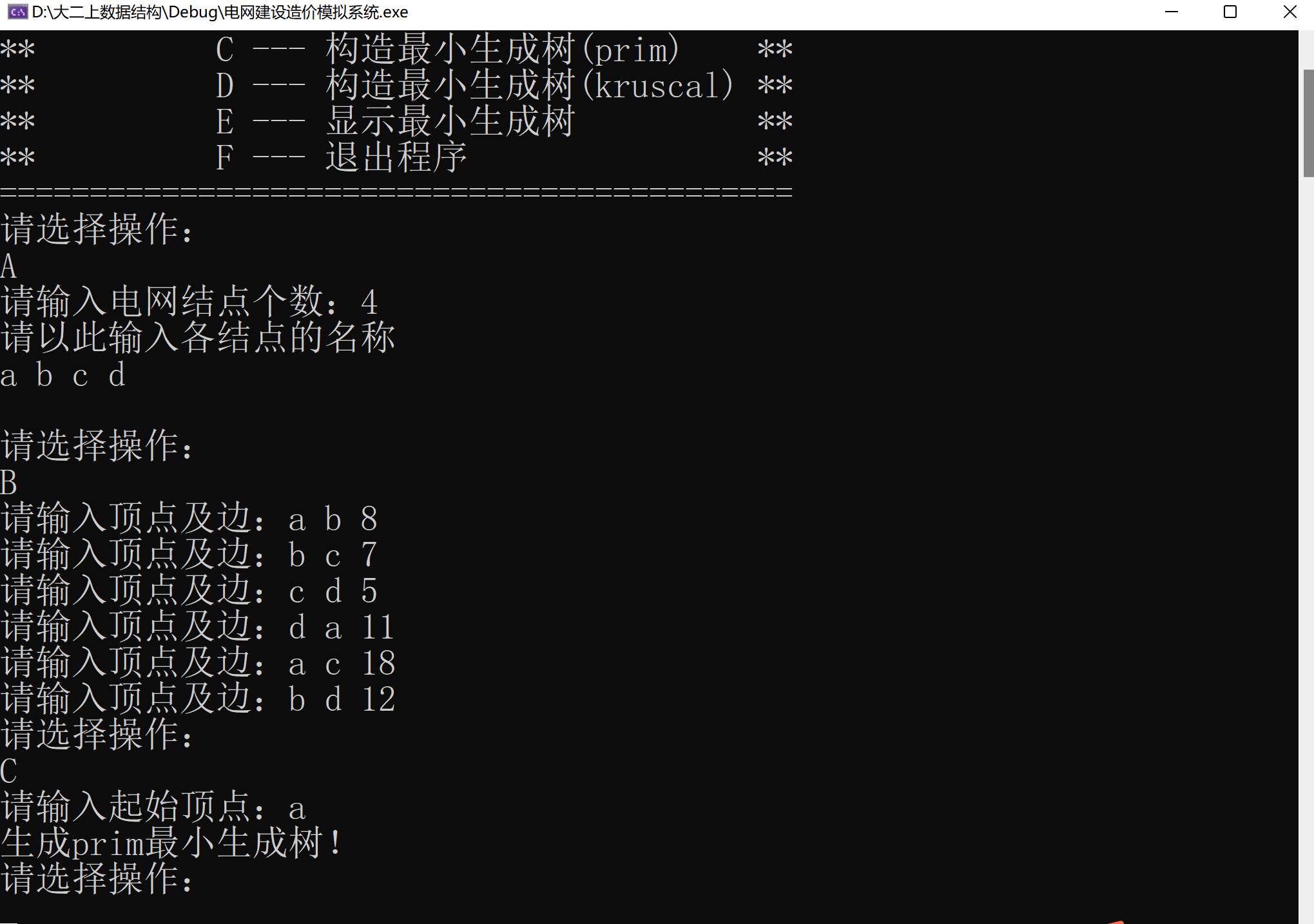
C

a

预期输出：

生成prim最小生成树！

实验结果：



## 3.4 显示最小生成树

测试输入：

E

预期输出：

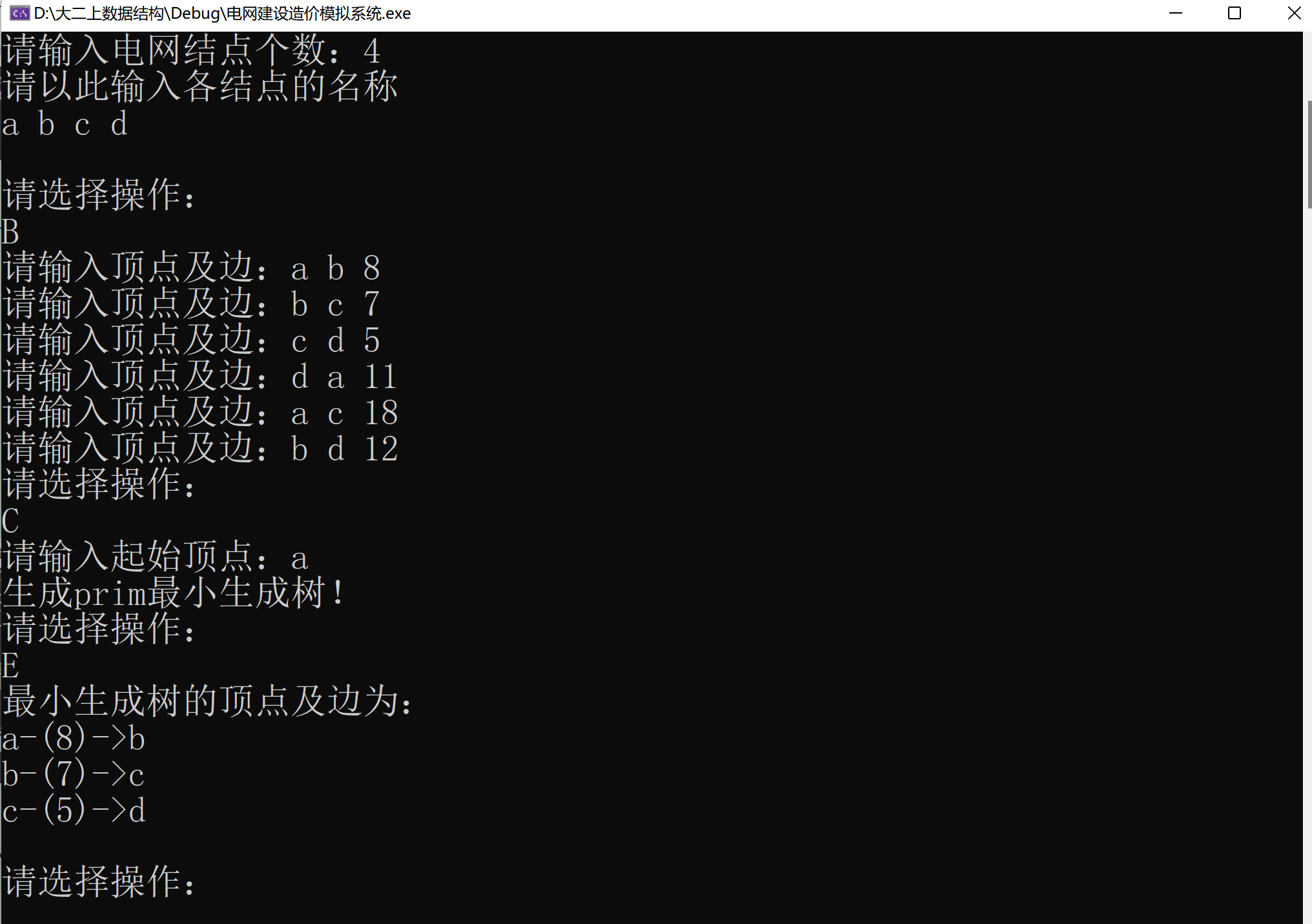
最小生成树的顶点及边为：

a-(8)->b

b-(7)->c

c-(5)->d

实验结果：



## 3.5 构造最小生成树(kruscal)

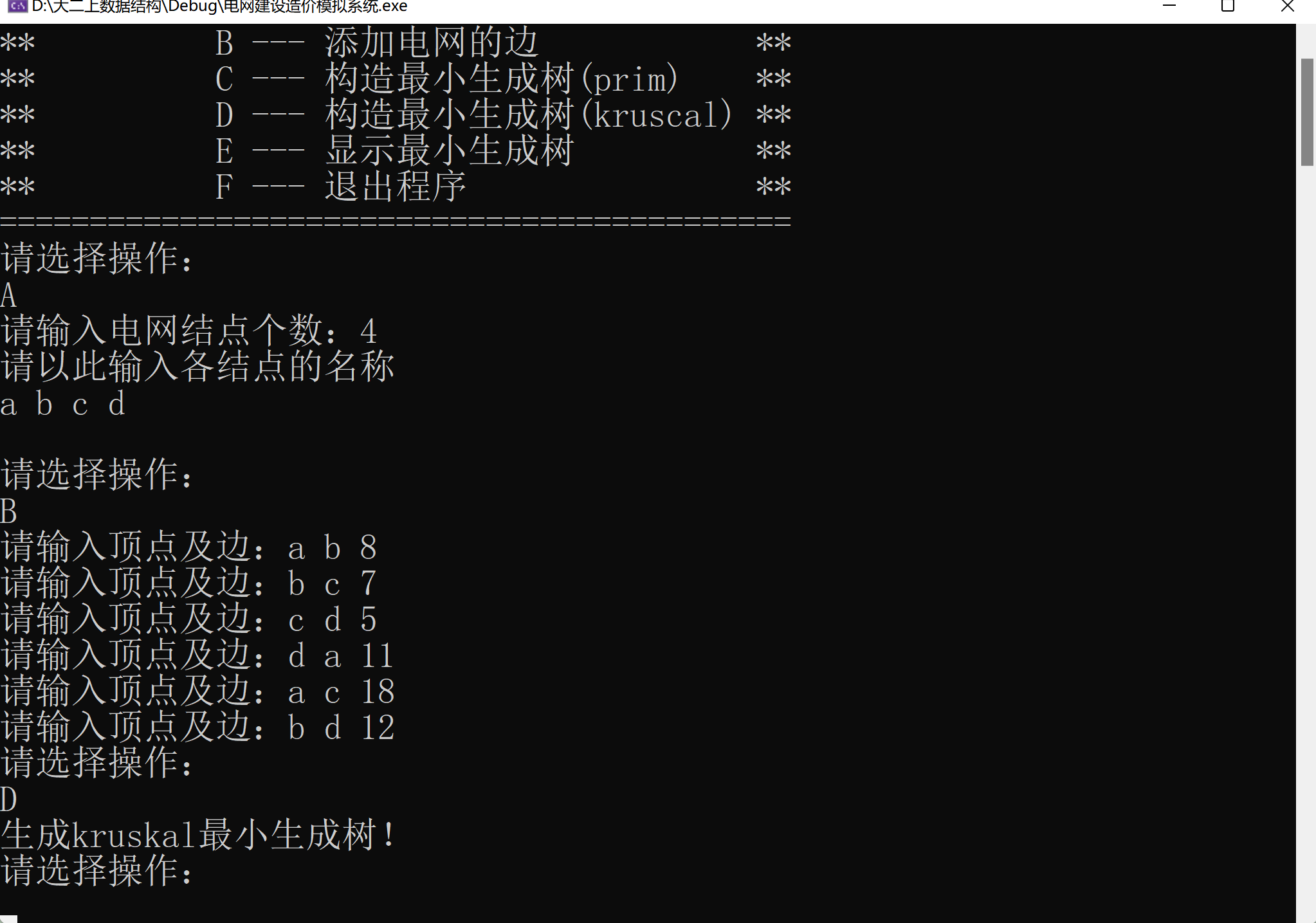
测试输入：

D

预期输出：

生成kruskal最小生成树！

实验结果：



## 3.6 显示最小生成树(kruscal)

测试输入：

E

预期输出：

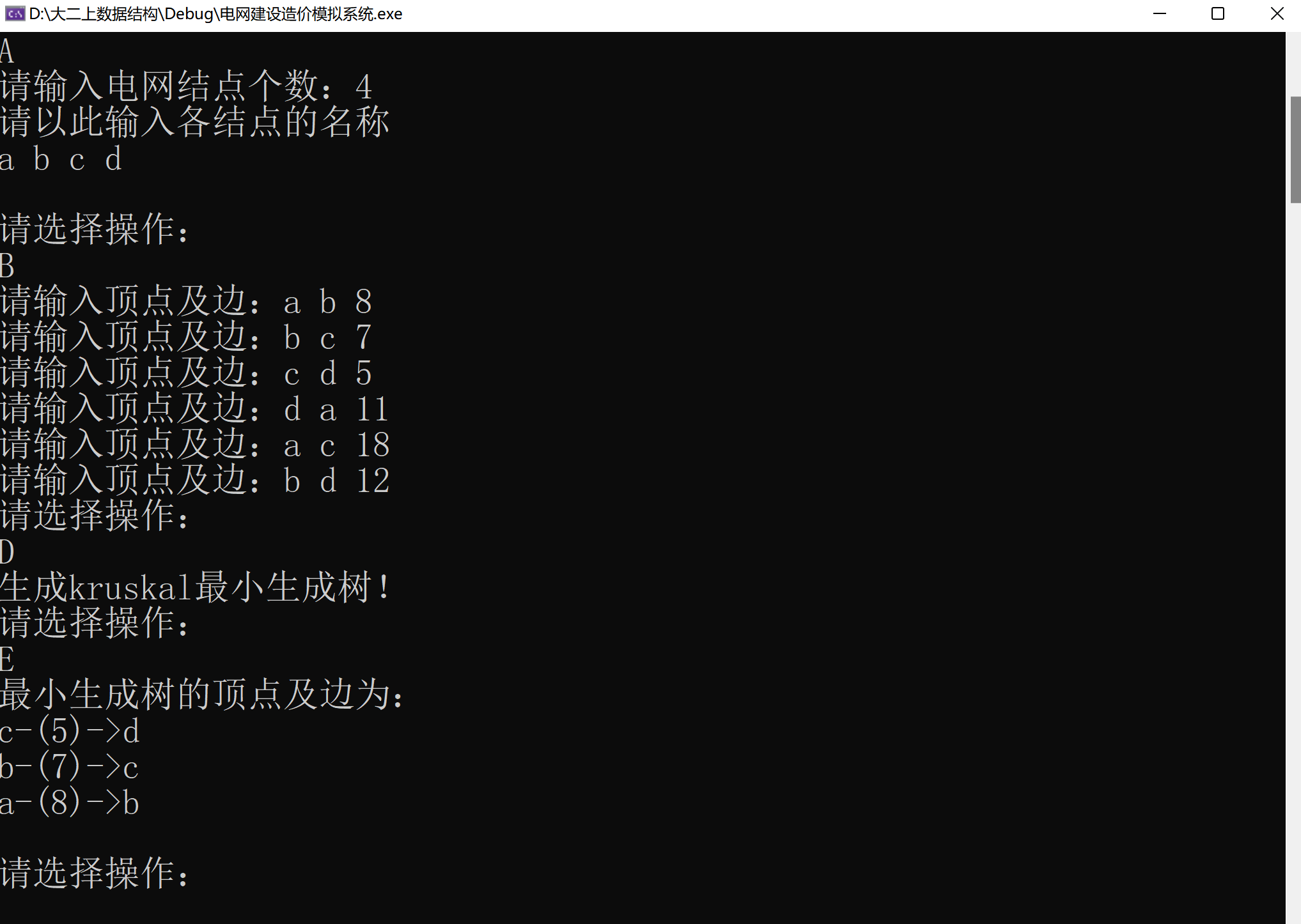
最小生成树的顶点及边为：

c-(5)->d

b-(7)->c

1. (8)->b

实验结果：



## 3.7 输入操作错误

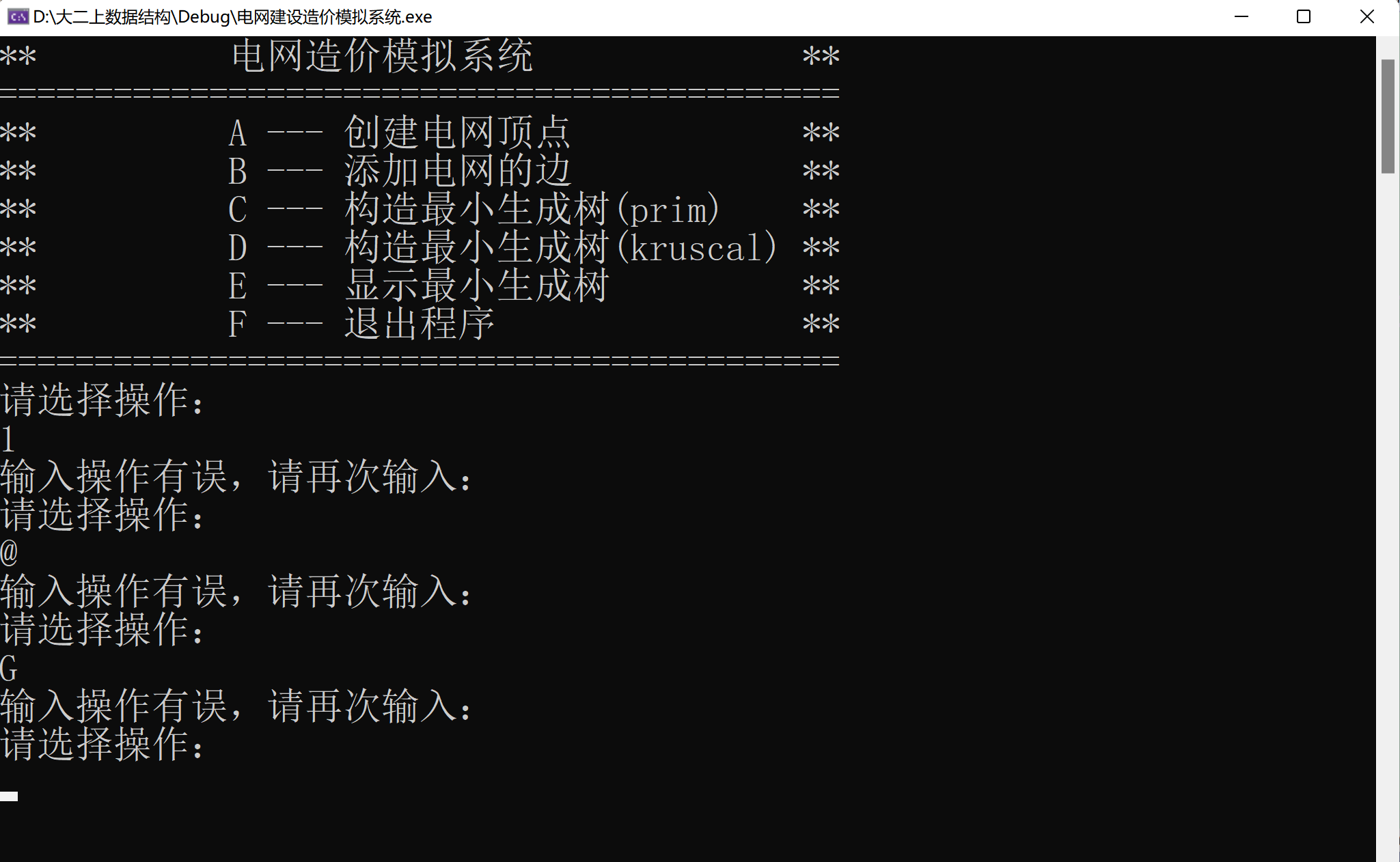
测试输入：

1 @ G

预期输出：

错误提示+重新输入

实验结果：



# 4 算法性能分析

## 4.1 正确性

本算法能正确地执行预定的功能和性能要求，由大量实验结果可知，不论是prim算法还是kruscal算法都和正确结果吻合，满足最小生成树的要求，即满足正确性。

## 4.2 可使用性

本算法可以很方便地使用，两类算法求最小生成树的功能已各自封装在一个函数中。并且该算法有良好的界面和完备的用户文档。没有使用公用变量或全局变量。

## 4.3 可读性

本算法逻辑清晰、简单、且结构化，所有命名与函数名都具有实际含义，让人见名知义。且算法中包含了大量注释，简要说明了算法功能、输入与输出参数的使用规则、重要数据的作用、算法中各程序段完成的功能。

## 4.4 效率

本项目的prim算法利用堆进行了优化，时间复杂度从O(n2)降为O(nlong2n)，适合稠密图，而kruscal中堆的建立以及提取时间复杂度为O(elog2e),并查集操作的时间复杂度为O(log2n),若采用邻接表作为图的存储结构，那么图的遍历时间复杂度为O(n+e)，若存储结构为邻接矩阵，图的遍历的时间复杂度为O(n2),所以总的时间复杂度为O(elog2e+elog2n+n+e)，适合简单图。

## 4.5 健壮性

本算法对于边界条件，诸如：选择操作非法，无法生成最小生成树都有相应的判断，并且对于新结点的申请失败也有相应的错误提示。