项目说明文档

数据结构课程设计

——电网建造

作 者 姓 名： 谢宇翔

学 号： 1951708

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 分析 1](#_Toc9489)

[1.1 背景分析 1](#_Toc6953)

[1.2 功能分析 1](#_Toc21079)

[2 设计 1](#_Toc15128)

[2.1 数据结构设计 1](#_Toc1509)

[2.2 类结构设计 1](#_Toc18051)

[2.3 成员与操作设计 1](#_Toc20745)

[2.4 系统设计 6](#_Toc32262)

[3 实现 7](#_Toc15443)

[3.1 建立顶点 7](#_Toc28874)

[3.1.1 功能流程图 7](#_Toc1981)

[3.1.2 核心代码 7](#_Toc24208)

[3.1.3 功能截屏示例 9](#_Toc7424)

[3.2 添加电网的边 10](#_Toc5923)

[3.2.1 功能流程图 10](#_Toc19924)

[3.2.2 添加电网的边核心代码 10](#_Toc10791)

[3.2.3添加电网的边截屏示例 13](#_Toc1256)

[3.3 构建最小生成树的实现 15](#_Toc4775)

[3.3.1构建最小生成树流程图 15](#_Toc2001)

[3.3.2 构建最小生成树核心代码 16](#_Toc9169)

[3.3.3 构建最小生成树截图示例 18](#_Toc6470)

[3.4 展示最小生成树的实现 18](#_Toc26809)

[3.4.1 展示最小生成树流程图 18](#_Toc1630)

[3.4.2 展示最小生成树核心代码 18](#_Toc30498)

[3.4.3展示最小生成树截屏示例 19](#_Toc16051)

[3.5 系统总体功能的实现 20](#_Toc30103)

[3.5.1系统总体功能流程图 20](#_Toc22160)

[3.5.2系统总体功能核心代码 20](#_Toc22375)

[4 测试 23](#_Toc2961)

[4.1 功能测试 23](#_Toc9520)

[4.2 边界测试 25](#_Toc26475)

[4.2.1 只有2个小区 25](#_Toc7972)

[4.2.2 所有边的权值一样大 26](#_Toc14762)

[4.3 出错测试 27](#_Toc11830)

[4.3.1 节点个数错误 27](#_Toc10409)

[4.3.2 节点个数只有1个 27](#_Toc961)

[4.3.3 边的权值输入错误 27](#_Toc23317)

[4.3.4 增加边时，节点不存在 28](#_Toc883)

[4.3.4 没有完成前期必要工作就进行后续操作 28](#_Toc1719)

# 1 分析

## 1.1 背景分析

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。在生活中也经常遇到一个类似图的结构，然后需要我们去求解最优的连接使得成本最低，此程序就是为了解决这一类问题

## 1.2 功能分析

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。

# 2 设计

## 2.1 数据结构设计

经过分析，可以用图这种数据结构来表示这个城市，顶点就是各个小区，顶 点间的边就是小区间的电网的路，边的 cost 值就是电网的造价，因此可以用Prim 算法求出该图的最小生成树以计算此电网的最低造价

## 2.2 类结构设计

设计graph类，用于存储图，每个节点是Vertex结构体，边是Edge结构体，相互包含嵌套让graph可以使用，同时设计了最小堆MinHeap类，节点边类MSTEdgeNode类，MinSpanTree类来构建以及储存最小生成树。

## 2.3 成员与操作设计

**图类graph**

公有成员：

int maxVertices;//最大顶点数

int numEdge;//当前边数

int numVertices;//当前点数

const int maxWeight = 9999;

MinSpanTree MST;//最小生成树

私有成员：

Vertex\* NodeTable;//邻接表

bool EdgeExist = 0;//边存在

int currentpos = 0;//当前位置

公有操作：

bool graphIsEmpty()

{

if (numEdge == 0)return true;

else return false;

}

bool graphIsFull()//判断图满

{

if (numVertices == maxVertices || numEdge == maxVertices \* (maxVertices - 1) / 2)return true;

else return false;

}

int getFirstNeighbor(int v);//获取第一个邻接点

int getNextNeighbor(int v, int now);//获取v的邻接点中now的下一个

int getWeight(int u, int v);//获取u和v节点之间的边的权值

私有操作：

int getVertexPos(char A);//给出顶点vertex在图中的位置

void CreateNode();//建立顶点

void AddEdge();//建立边

void insertEdge(int v1, int v2, int weight);//插入边

void CreateMintree();//建立最小生成树

void ShowMintree();//展示最小生成树

**边结构体Edge**

int dest;//边的另一个顶点的位置

int cost;//边的权值

Edge\* link;//下一个边链的指针

Edge() { dest = -1; cost = -1; link = NULL; }

Edge(int num,int weight):dest(num),cost(weight),link(NULL){}

bool operator!=(Edge& R)const {

return (dest != R.dest) ? true : false;

}

**节点结构体Vertex**

char data='\0';

Edge\* adj=NULL;//边链表的头指针

**最小堆类MinHeap**

私有成员：

MSTEdgeNode\* heap;//最小堆里的数据是边节点类

int currentsize;

int maxheapsize;

私有操作：

void siftDown(int start, int m);//由上到下修改顺序

void siftUp(int start);//由下到上修改顺序

公有操作：

MinHeap(int sz = 30);

bool insert(MSTEdgeNode& x);//插入一个节点

bool removeMin(MSTEdgeNode& x);//移除一个最小的节点

bool isEmpty()

{

return(currentsize == 0) ? true : false;

}

bool isFull()

{

return (currentsize == maxheapsize) ? true : false;

}

**边节点类MSTEdgeNode**

int tail, head, key;//头结点的位置，尾节点的位置，边的权值

MSTEdgeNode():tail(-1),head(-1),key(0){}

bool operator<=(MSTEdgeNode& R) { return key <= R.key; }

bool operator>(MSTEdgeNode& R) { return key > R.key; }

**最小生成树类MinSpanTree**

保护成员：

int maxSize, currentpos;

公有成员：

MSTEdgeNode\* edgevalue;//存储着生成树的节点链

公有操作：

MinSpanTree(int sz=30):maxSize(sz),currentpos(0){

edgevalue = new MSTEdgeNode[sz];

}

void insert(MSTEdgeNode& item);//插入新的连接

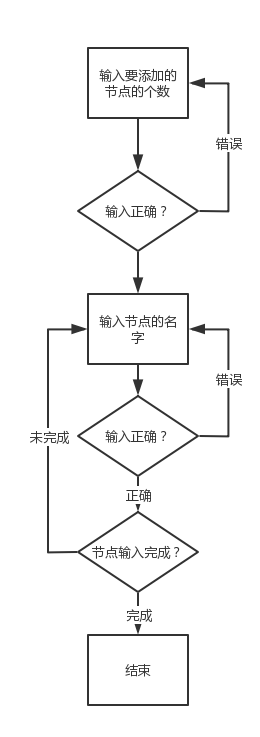
## 2.4 系统设计

首先输出menu，然后根据用户的选择建立顶点，建立边和构建最小生成树，输出最小生成树，退出的操作，在执行条件不满足时会提供错误提示要求重新输入。

# 3 实现

## 3.1 建立顶点

### 3.1.1 功能流程图



### 3.1.2 核心代码

int num;

while (1)

{

cout << "请输入顶点的个数:";

cin >>num;

if (!cin.good())

{

cout << "输入错误！请重新输入" << endl;

cin.clear();//重置cin流

cin.ignore(100, '\n');//清空输入缓冲区

}

else if (num <= 1)

{

cout << "请输入大于1的正整数!"<<endl;

cin.clear();//重置cin流

cin.ignore(100, '\n');//清空输入缓冲区

}

else

break;

}

numVertices += num;

cout << "请依次输入各顶点的名称:" << endl;

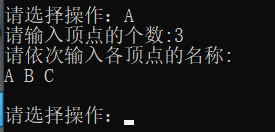
for (; currentpos < numVertices; currentpos++)

{

cin >> NodeTable[currentpos].data;

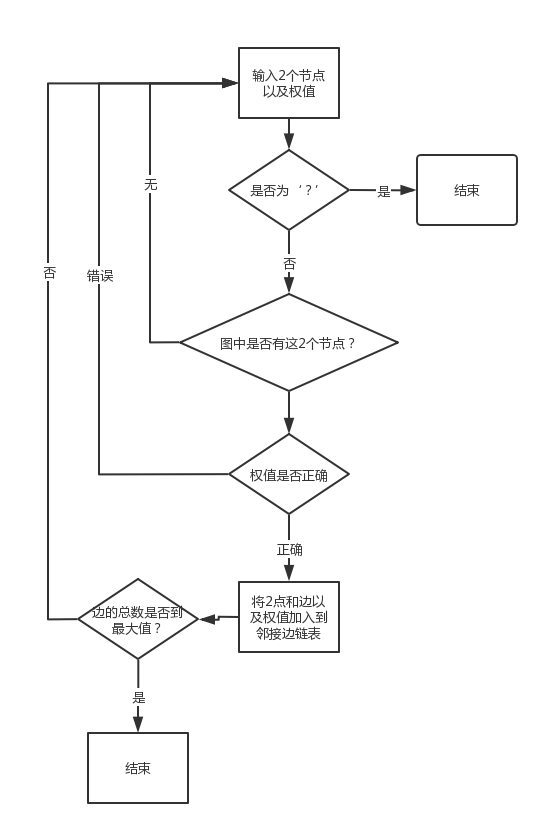
}

### 3.1.3 功能截屏示例



## 3.2 添加电网的边

### 3.2.1 功能流程图



### 3.2.2 添加电网的边核心代码

void graph::insertEdge(int v1, int v2, int weight)

{//因为已经确认了v1,v2是合法的，所以不用if判定是否合法

Edge\* q, \* p = NodeTable[v1].adj;//v1对应的边链表头节点

while (p != NULL && p->dest != v2)//寻找v2

{

p = p->link;

}

if (p != NULL)

{

cout << "2节点已经有边,无法再建立边！"<<endl;

return;//v2已经是v1的边了

}

p = new Edge;

q = new Edge;

p->dest = v2, p->cost = weight;

p->link = NodeTable[v1].adj;

NodeTable[v1].adj = p;

q->dest = v1; q->cost = weight;

q->link = NodeTable[v2].adj;

NodeTable[v2].adj = q;

numEdge++; return;

}

void graph::AddEdge()

{

int Maxnum = numVertices \* (numVertices - 1) / 2;

if (numVertices == 0)

{

cout << "图中尚无顶点！" << endl;

return;

}

while (1)

{

if (numEdge == Maxnum)//边数到达最大

return;

cout << "请输入两个顶点及边:（输入“?”退出，或者到达最大输入数自动退出）";

char want1, want2;

cin >> want1>>want2;

if (want1 == '?')//读到问号

{

cin.ignore(1000, '\n');

return;

}

int find1 = -1, find2 = -1;//判断是否找到

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

{

if (NodeTable[i].data == want1)

find1 = i;

else if (NodeTable[i].data == want2)

find2 = i;

}

if (find1 == -1 || find2 ==-1)//只要有一个没找到就报错

{

cout << "错误的顶点！请重新输入" << endl;

cin.ignore(1000, '\n');

continue;

}

else

{

int value;

cin >> value;//输入权值

if (cin.fail() || value < 0)

{

cout << "输入错误，请重新输入!" << endl;

cin.clear();

cin.ignore(1000, '\n');

continue;

}

else if (value == 0)

{

cout << "请输入正整数，请重新输入!" << endl;

cin.clear();

cin.ignore(1000, '\n');

continue;

}

else

{

EdgeExist = 1;

insertEdge(find1, find2, value);

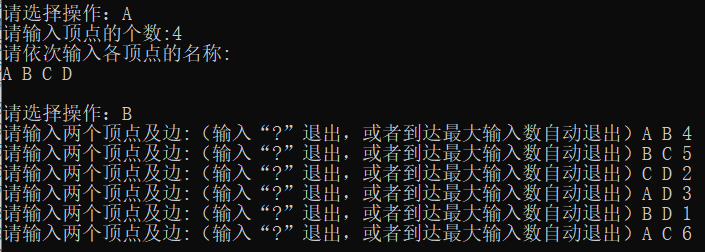
}

}

}

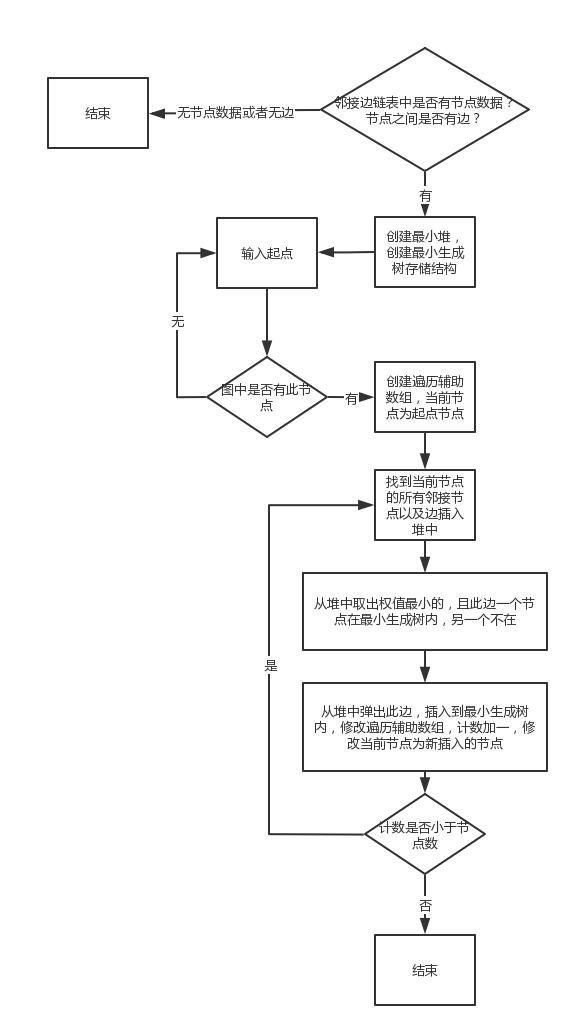
}

### 3.2.3添加电网的边截屏示例



## 3.3 构建最小生成树的实现

### 3.3.1构建最小生成树流程图



### 3.3.2 构建最小生成树核心代码

void graph::CreateMintree()

{

if (NodeTable[0].data == '\0')

{

cout << "无法生成图！因为没有顶点！" << endl;

return;

}

else if (!EdgeExist)

{

cout << "无法生成图！因为没有边！" << endl;

return;

}

MSTEdgeNode ed;

MinHeap H(numEdge);

int count;

cout << "请输入起始顶点: ";

char a;

int u,v,min=INT\_MAX;

while (1)

{

cin >> a;

u = getVertexPos(a);//u记录a的位置

if(u==-1)

{

cout << "无此顶点请重新输入" << endl;

cin.ignore(1000, '\n');

continue;

}

else

{

bool\* Vmst = new bool[numVertices];

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

Vmst[i] = false;

Vmst[u] = true;

count = 1;

do{

v = getFirstNeighbor(u);

while (v != -1)

{

if (Vmst[v] == false)//未访问过

{

ed.tail = u, ed.head = v;

ed.key = getWeight(u, v);

H.insert(ed);//插入堆

}

v = getNextNeighbor(u, v);//对其他邻接顶点操作

}

while (H.isEmpty() == false && count < numVertices)//构建最小生成树

{

H.removeMin(ed);//取出

if (Vmst[ed.head] == false)

{

MST.insert(ed);//插入最小生成树

u = ed.head;

Vmst[u] = true;

count++;

break;//成功插入一个点，要退出，重新寻找图中距离最小生成树最短的顶点

}

}

} while (count < numVertices);

break;

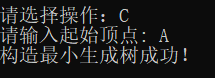
}

}

cout << "构造最小生成树成功！" << endl;

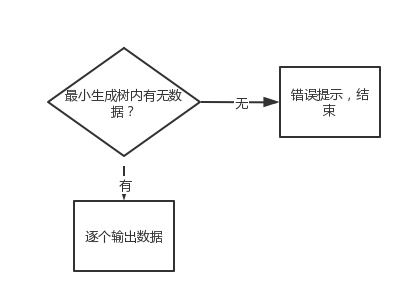
}

### 3.3.3 构建最小生成树截图示例



## 3.4 展示最小生成树的实现

### 3.4.1 展示最小生成树流程图



### 3.4.2 展示最小生成树核心代码

void graph::ShowMintree()

{

if (MST.edgevalue[0].head == -1)

{

cout << "尚未有最小生成树！" << endl;

return;

}

cout << "最小生成树的顶点及边为:" << endl << endl;

for (int i = 0; i < numVertices-1; i++)

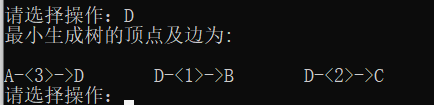
{

cout << NodeTable[MST.edgevalue[i].tail].data << "-<" << MST.edgevalue[i].key << ">->" << NodeTable[MST.edgevalue[i].head].data << " ";

}

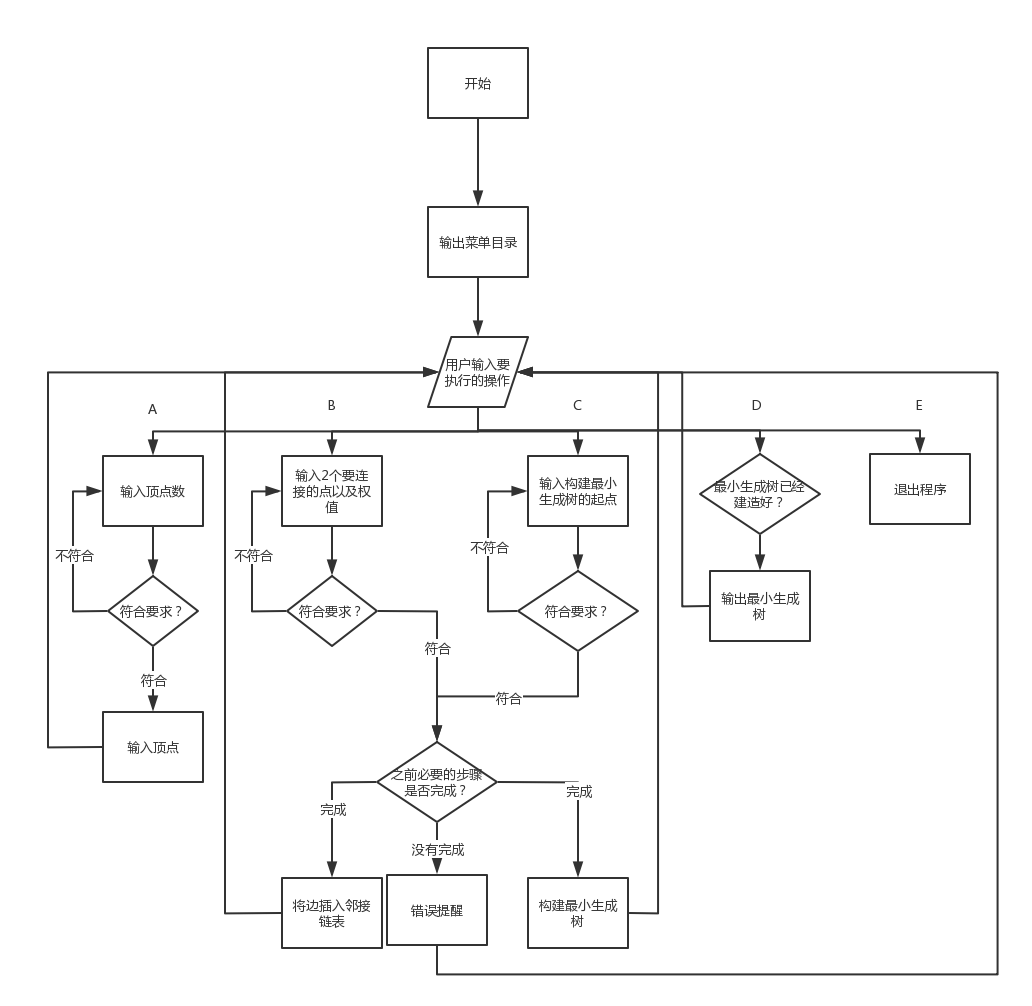
}

### 3.4.3展示最小生成树截屏示例



## 3.5 系统总体功能的实现

### 3.5.1系统总体功能流程图



### 3.5.2系统总体功能核心代码

void graph::menu()

{

cout << "\*\* 电网造价模拟系统 \*\*" << endl;

cout << "=============================================" << endl;

cout << "\*\* 请选择以下的操作 \*\*" << endl;

cout << "\*\* A---创建电网顶点 \*\*" << endl;

cout << "\*\* B---添加电网的边 \*\*" << endl;

cout << "\*\* C---构造最小生成树 \*\*" << endl;

cout << "\*\* D---显示最小生成树 \*\*" << endl;

cout << "\*\* E---退出程序 \*\*" << endl;

cout << "=============================================" << endl;

while (1)

{

char choice;

while (1)

{

cout << endl << "请选择操作：";

cin >> choice;

if (!cin.good() ||!(( choice >= 'A' && choice<='E')||(choice>='a'&&choice<='e')))

{

cout << "输入错误！请重新输入" << endl;

cin.clear();//重置cin流

cin.ignore(100, '\n');//清空输入缓冲区

}

else

break;

}

switch (choice)

{

case 'A':

case 'a':

CreateNode();

break;

case 'B':

case 'b':

AddEdge();

break;

case 'C':

case 'c':

CreateMintree();

break;

case'd':

case'D':

ShowMintree();

break;

case 'e':

case'E':

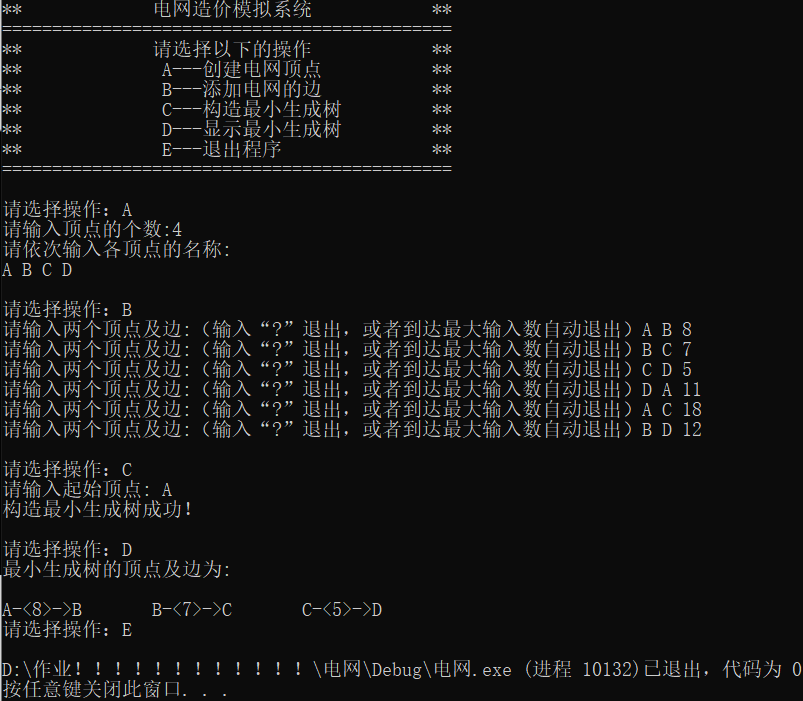
return;

}

}

}

3.5.3系统总体功能截屏示例



# 4 测试

## 4.1 功能测试

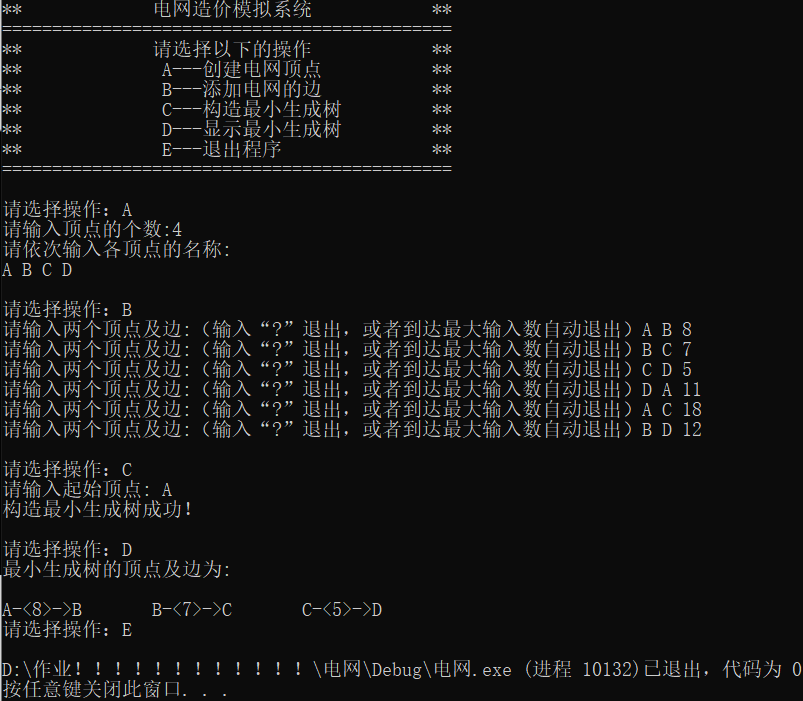
**测试用例**：

**预期结果**：



（此图为作业里的示例图）

**实验结果**



## 4.2 边界测试

### 4.2.1 只有2个小区

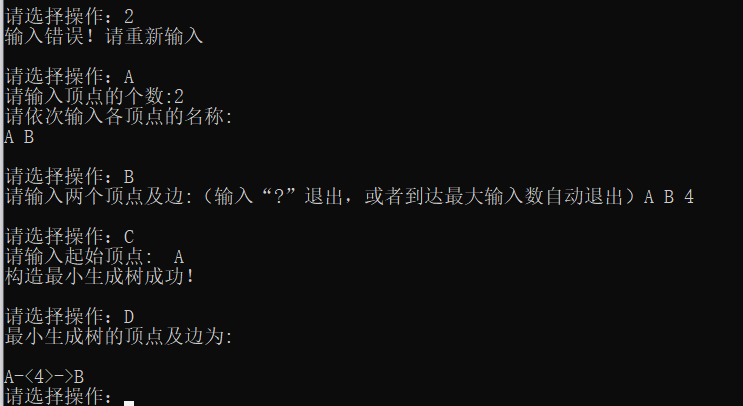
**测试用例：**

只有2个小区的图

**预期结果：**

正常生成最小生成树并输出

**实验结果：**



### 4.2.2 所有边的权值一样大

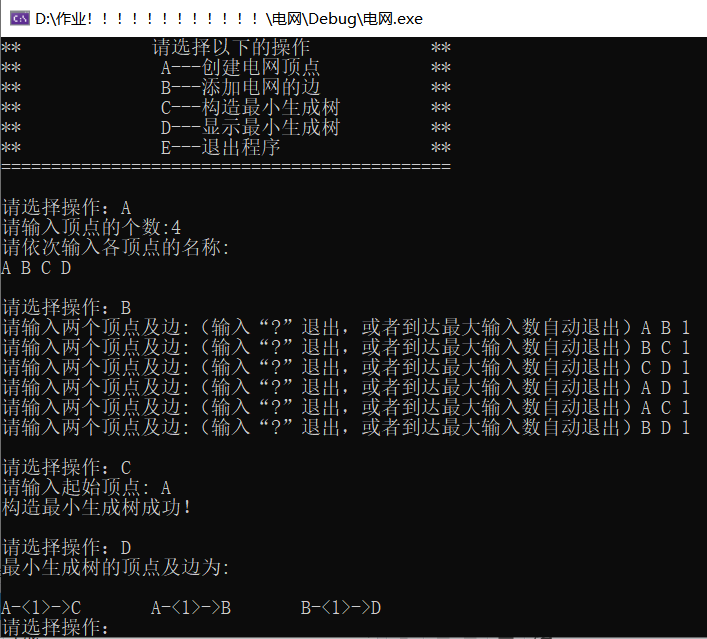
**测试用例：**

所有小区之间的边权值都一样大

**预期结果：**

正常生成最小生成树并输出

**实验结果：**



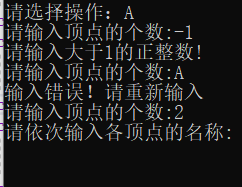
## 4.3 出错测试

### 4.3.1 节点个数错误

**测试用例：**输入节点个数错误

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

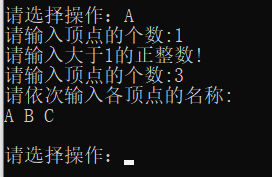


### 4.3.2 节点个数只有1个

**测试用例：**输入节点个数1

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

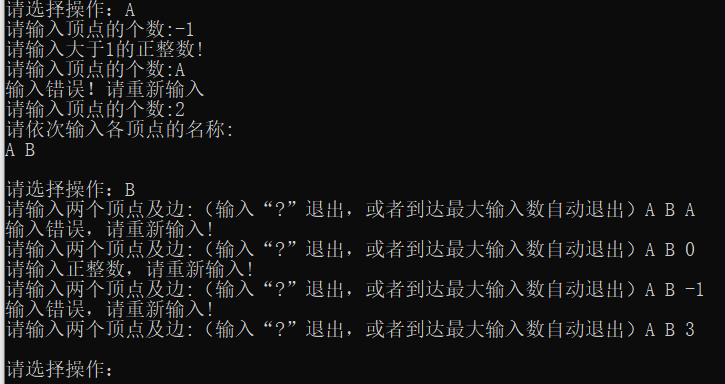


### 4.3.3 边的权值输入错误

**测试用例：**输入边的权值错误

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

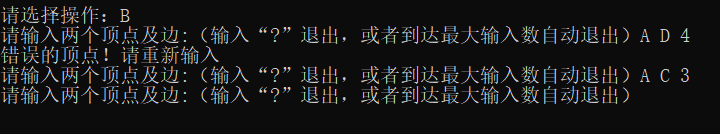


### 4.3.4 增加边时，节点不存在

**测试用例：**增加边时，节点不存在

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

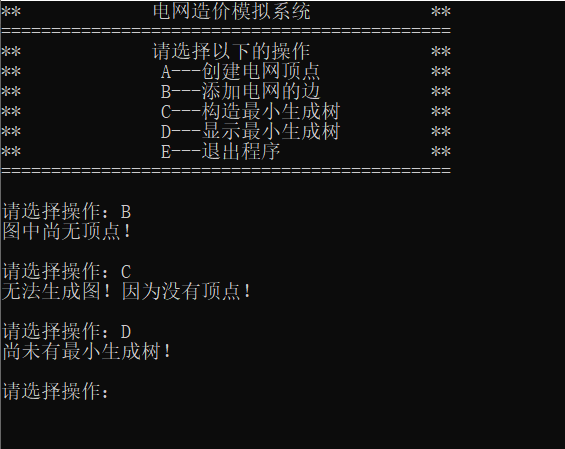
**实验结果：**

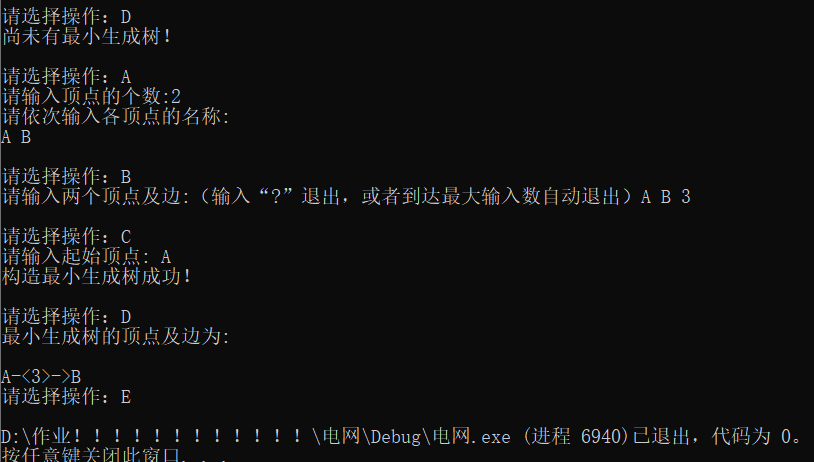


### 4.3.4 没有完成前期必要工作就进行后续操作

**测试用例：**没有完成前期必要工作就进行后续操作

**预期结果：**程序给出提示信息，程序正常运行不崩溃,在完成必要操作后，程序正确运行。

**实验结果：**



# 