项目说明文档

数据结构课程设计

——电网造价模拟系统

作 者 姓 名： 张喆

学 号： 1754060

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](#_Toc533280521)

[1.1 背景分析 1](#_Toc533280522)

[1.2 功能分析 1](#_Toc533280523)

[2 设计 2](#_Toc533280524)

[2.1 数据结构设计 2](#_Toc533280525)

[2.2 类结构设计 2](#_Toc533280526)

[2.3 成员与操作设计 2](#_Toc533280527)

[2.4 系统设计 3](#_Toc533280528)

[3 实现 4](#_Toc533280529)

[3.1 插入功能的实现 4](#_Toc533280530)

[3.1.1 插入功能流程图 4](#_Toc533280531)

[3.2 删除功能的实现 7](#_Toc533280532)

[3.2.1 删除功能流程图 7](#_Toc533280533)

[3.2.2 删除功能核心代码 7](#_Toc533280534)

[} 8](#_Toc533280535)

[3.2.3 删除功能截屏示例 8](#_Toc533280536)

[3.3 查找功能的实现 9](#_Toc533280537)

[3.3.1 查找功能流程图 9](#_Toc533280538)

[3.3.2 查找功能核心代码 9](#_Toc533280539)

[return false; 10](#_Toc533280540)

[3.3.3 查找功能截图示例 10](#_Toc533280541)

[3.4 修改功能的实现 11](#_Toc533280542)

[3.4.1 修改功能流程图 11](#_Toc533280543)

[3.4.2 修改功能核心代码 11](#_Toc533280544)

[} 12](#_Toc533280545)

[3.4.3 修改功能截屏示例 12](#_Toc533280546)

[3.5 统计功能的实现 13](#_Toc533280547)

[3.5.1 统计功能流程图 13](#_Toc533280548)

[3.5.2 统计功能核心代码 13](#_Toc533280549)

[} 14](#_Toc533280550)

[3.5.3 统计功能截屏示例 14](#_Toc533280551)

[3.6 总体系统的实现 14](#_Toc533280552)

[3.6.1 总体系统流程图 14](#_Toc533280553)

[3.6.2 总体系统核心代码 14](#_Toc533280554)

[cout << "操作结束" << endl; 16](#_Toc533280555)

[3.6.3 总体系统截屏示例 16](#_Toc533280556)

[4 测试 17](#_Toc533280557)

[4.1 功能测试 17](#_Toc533280558)

[4.1.1 插入功能测试 17](#_Toc533280559)

[4.1.2 删除功能测试 18](#_Toc533280560)

[4.1.3 查找功能测试 19](#_Toc533280561)

[4.1.4 修改功能测试 19](#_Toc533280562)

[4.1.5 统计功能测试 19](#_Toc533280563)

[4.2 边界测试 20](#_Toc533280564)

[4.2.1 初始化无输入数据 20](#_Toc533280565)

[4.2.2 删除头结点 20](#_Toc533280566)

[4.2.3 删除后链表为空 21](#_Toc533280567)

[4.3 出错测试 22](#_Toc533280568)

[4.3.1 输入操作码错误 22](#_Toc533280569)

[4.3.2 输入顶点个数为负数 22](#_Toc533280570)

[4.3.3 创建电网顶点时输入的顶点名称个数与顶点个数不匹配 22](#_Toc533280571)

[4.3.4 创建电网顶点时输入的顶点名称中有非字母的名称 23](#_Toc533280572)

[4.3.5 添加的边的某个顶点不是电网的顶点 23](#_Toc533280573)

[4.3.6 添加的边中两顶点的花费不是正整数 24](#_Toc533280574)

[4.3.7 构建最小生成树的顶点不在电网中 24](#_Toc533280575)

[4.3.8 电网不连通 24](#_Toc533280576)

# 1 分析

## 1.1 背景分析

电网建设造价模拟系统是要实现已知某个城市中有n个小区，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。

这个系统对政府机关的决策以及施工队的预算处理等很多方面都有很重要的作用，传统的手工估算不仅耗费时间很长，而且错误率很高，如果是放在全中国范围研究，对手工来说几乎是不可能的事情；随着计算机的发展，我们很有必要开发这样一个电网建设造价模拟系统已取代人工的手工劳动，使成本降到最低，并能得到作为精确的解。

## 1.2 功能分析

作为一个简易的电网建设造价模拟系统，首先应该有的功能就是可以正确储存城市中各个小区的名字，即电网的结点，并能对电网结点的输入合理与否进行一个最简单的判断。

其次，电网造价模拟系统还应该能正确储存各个小区之间的距离，即可以正确无误的添加电网的边。

之后可以使用正确而高效的方法实现最小生成树的构建并予以显示，当各个小区并不连通的时候可以提醒用户。

最后系统应能正常退出，并保持一定的健壮和稳定性。

# 2 设计

## 2.1 数据结构设计

综上所述，该系统要能正确储存电网的顶点和添加顶点的边，该系统使用vector储存电网的顶点，把各个边的长度以及断点作为一个结构体共同储存。

同时在Prim算法中由于需要每次寻找树外到树内距离最近的顶点，因此使用优先队列priority\_queue把与树内结点关联的所有边压入队列，每次可以便捷高效的取出花费最少的边。

最后把用于构建最小生成树的所有边及它的两个端点一同储存，这样可以更便捷的予以显示。

## 2.2 类结构设计

电网的边类Edge：用于记录这条边的两个端点及它的长度，并可以判断是否为空边

顶点和与改定关联的所有边构成的类Branch：储存某个顶点，和在电网所有边中，以这个顶点作为某个端点的所有边构成的优先队列。

电网类ProwerGrid：用于储存电网中所有的顶点、边及最小生成树的信息，并具有创建电网顶点、添加电网的边、构建最小生成树、显示最小生成树等方法。

用于构建仿函数类cmp

## 2.3 成员与操作设计

**电网的边类（Edge）**

**公有成员：**

Vertex v1, v2; //边的两个端点

int cost; //边的长度

**电网边的构造函数：**

Edge() = default;

Edge(int buf1, int buf2, int bufcost);

Edge::Edge(int buf1, int buf2, int bufcost)

:v1(buf1), v2(buf2), cost(bufcost) {}

**重载运算符：**

bool operator>(const Edge &buf) const

{return this->cost > buf.cost;}

**公有操作：**

bool isemptyEdge() { return (v1 == '?' && v2 == '?' && cost == 0); } //判断某条边是否为空边

**顶点和与改定关联的所有边构成的类（Branch）**

**公有成员：**

Vertex v;

priority\_queue<Edge, vector<Edge>, cmp> priQ; //与该顶点关联的边构成的优先队列

**构造函数：**

Branch() = default;

Branch(const Vertex &buf, const PowerGrid &root);

**仿函数类（cmp）**

**公有操作：**

bool operator()(const Edge &e1, const Edge &e2)const

{

return e1 > e2;

}

**电网类（PowerGrid）**

**私有成员：**

bool isconnect = true; //电网是否连通

vector<Vertex> \_vertex; //顶点集合

vector<Edge> \_edge; //边集合

vector<Edge> \_minSpanTree; //最小生成树

**公有操作：**

int VertexSize()const { return this->\_vertex.size(); } //返回顶点总数

int EdgeSize()const { return this->\_edge.size(); } //返回边的总数

bool InitVertex(int size); //创建电网顶点

void addEdges(); //添加电网的边

bool isVertex(Vertex buf); //判断某个点是否为电网的顶点

bool Prim(Vertex start); //求最小生成树的Prim算法

Edge findminEdge(const vector<Branch> &branch); //求当前的最小cost的边

void growBranch(vector<Branch> &branch); //把树内结点加入到branch中

void showSpanTree(); //显示最小生成树

**友元：**

friend class Branch;

## 2.4 系统设计

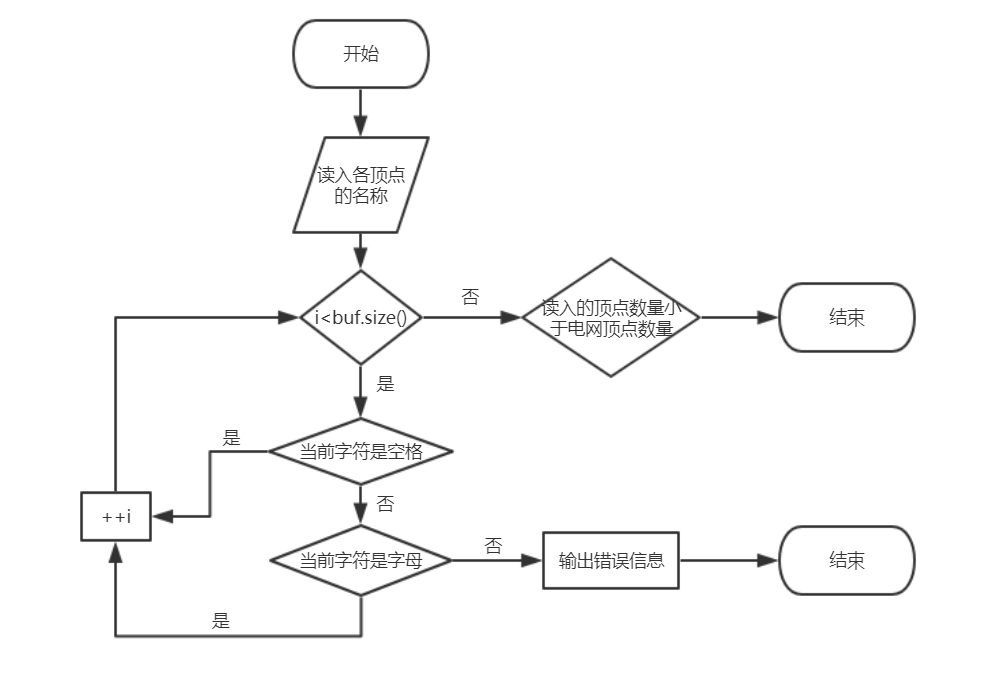
系统首先调用实现对屏幕的初始化，并提示用于各操作码的含义，之后创建电网对象，并根据用于的操作码实现相应的功能，具体包括创建电网顶点、添加电网的边、构造最小生成树和显示最小生成树。

用户结束操作之后可以正常退出程序，并能在用户非法操作的时候提醒用户。

# 3 实现

## 3.1 创建电网顶点功能的实现

### 3.1.1 创建电网顶点功能流程图



### 3.1.2 创建电网顶点功能核心代码

bool PowerGrid::InitVertex(int size)

{

this->\_vertex.resize(size);

cout << "请依次输入各顶点的名称:(单个字母) " << endl;

string buf;

getline(cin, buf);

int cnt = 0;

for (int i = 0; i < buf.size(); ++i)

{

if (isspace(buf[i])) //跳过空格

{ continue; }

else if (isalpha(buf[i]))

//如果是字母的话读入

{

if (cnt < this->\_vertex.size())

//当输入的顶点名称个数小于顶点个数时,储存该名称

{

this->\_vertex[cnt] = buf[i];

++cnt;

}

else

{

cerr << "输入的顶点名称个数不得超过顶点个数, 请重新创建所有顶点" << endl;

return false;

}

}

else

{

cerr << "各顶点的名称只能为字母, 请重新创建所有顶点" << endl;

return false;

}

}

if (cnt < this->\_vertex.size())

{

cerr << "输入的顶点名称个数不得少于顶点个数, 请重新创建所有顶点" << endl;

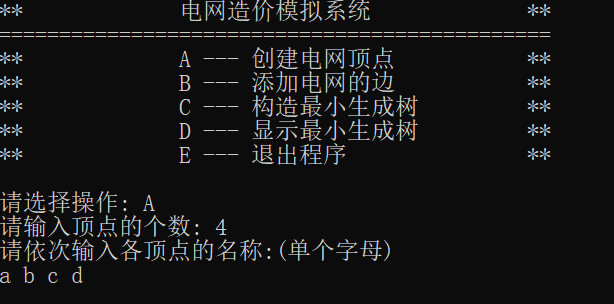
return false;

}

return true;

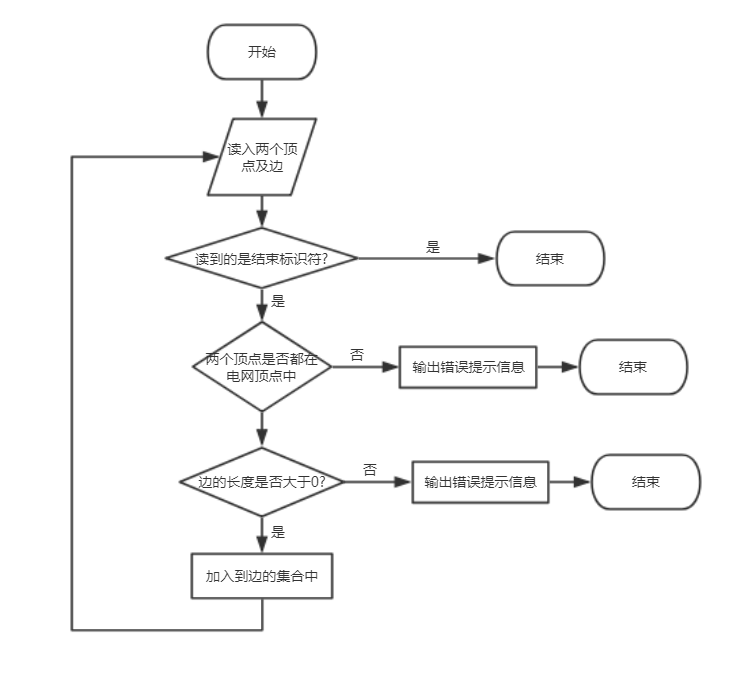
}

### 3.1.3 创建电网顶点功能截屏示例



## 3.2 添加电网的边功能的实现

### 3.2.1 添加电网的边功能流程图



### 3.2.2 添加电网的边功能核心代码

void PowerGrid::addEdges()

{

Edge buf;

cout << "以 ? ? 0 作为结束标识符!" << endl;

while (true)

{

cout << "请输入两个顶点及边: ";

cin >> buf.v1 >> buf.v2 >> buf.cost;

if (buf.isemptyEdge()) //读到结束标识符退出读入

{ break; }

if (isVertex(buf.v1) && isVertex(buf.v2))

//如果输入的两个顶点都在电网顶点中

{

if (buf.cost > 0)

{

this->\_edge.push\_back(buf);

}

else

{

cerr << "两顶点间的花费应为正整数" << endl;

continue;

}

}

else

{

cerr << "请输入电网中存在的顶点" << endl;

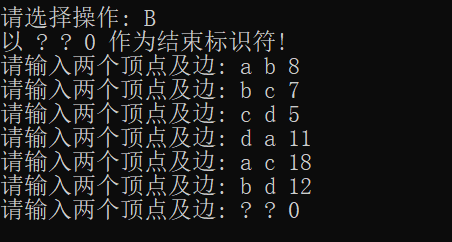
continue;

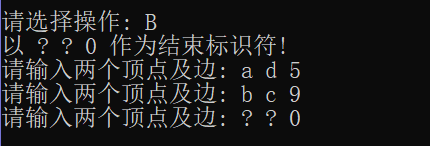
}

}

}

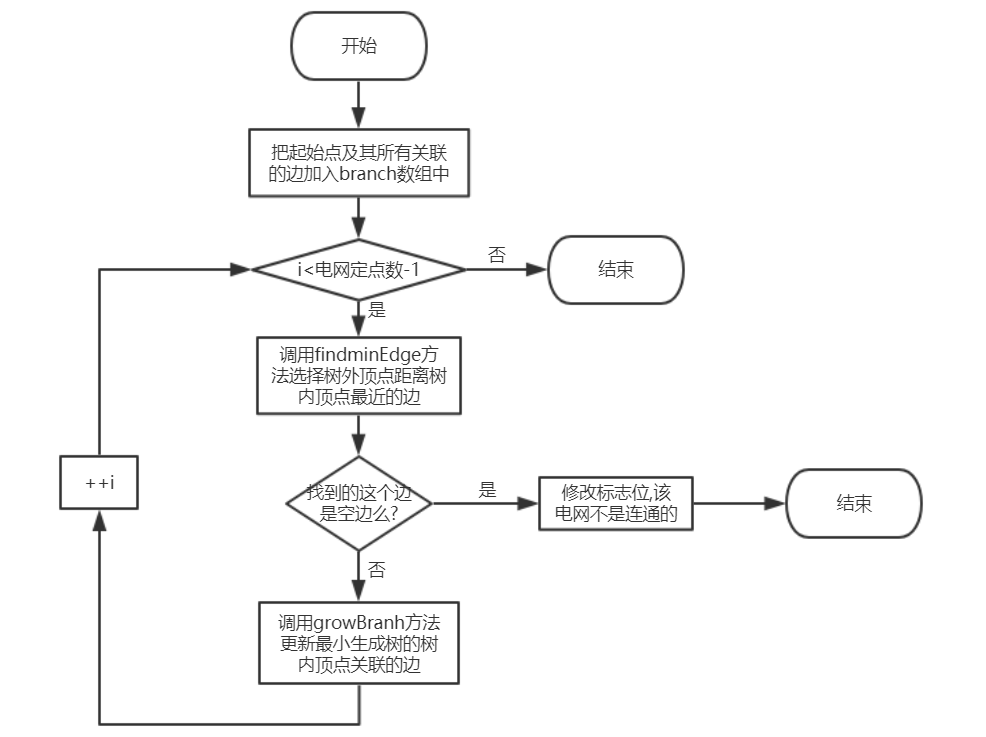
### 3.2.3 添加电网的边功能截屏示例





## 3.3 构造最小生成树功能的实现

### 3.3.1 构造最小生成树功能流程图



### 3.3.2 构造最小生成树功能核心代码

bool PowerGrid::Prim(Vertex start)

{

vector<Branch> branch;

branch.push\_back(Branch(start, \*this)); //先把起始点压入branch中

for (int i = 0; i < this->VertexSize()-1; ++i)

//n个顶点只需选出n-1条边

{

this->\_minSpanTree.push\_back(findminEdge(branch)); //把最小cost的边加入到最小生成树中

if (this->\_minSpanTree.back().isemptyEdge())

//如果存在空边, 该电网不构成网络

{

this->isconnect = false;

return false;

}

growBranch(branch); //更新最小生成树的树内顶点关联的边

}

return true;

}

Edge PowerGrid::findminEdge(const vector<Branch>& branch)

{

int mincost = INFINITE;

Edge minEdge('?', '?', 0);

for (int i = 0; i < branch.size(); ++i)

{

if (branch[i].priQ.empty()) { return minEdge; }

if (branch[i].priQ.top().cost < mincost)

//如果找到更小的边则更新minEdge

{

mincost = branch[i].priQ.top().cost;

minEdge = branch[i].priQ.top();

}

}

return minEdge;

}

void PowerGrid::growBranch(vector<Branch> &branch)

{

Vertex bufv1 = this->\_minSpanTree.back().v1, bufv2 = this->\_minSpanTree.back().v2;

Edge lastEdge = this->\_minSpanTree.back();

for (int i = 0; i < branch.size(); ++i)

{

if (branch[i].v == bufv1)

{

/\*在电网的边中删除这个边\*/

auto iter2 = find(this->\_edge.begin(), this->\_edge.end(), lastEdge);

this->\_edge.erase(iter2, iter2 + 1);

/\*更新一遍刚才选出最小边的位置\*/

branch.erase(branch.begin() + i, branch.begin() + i + 1); //把原来的删除

Branch refresh = Branch(bufv1, \*this); //如果更新后还有与这个点关联的边

if (refresh.v != '?')

{

branch.push\_back(refresh); //把他更新之后的加入

}

branch.push\_back(Branch(bufv2, \*this));

break;

}

else if (branch[i].v == bufv2)

{

/\*在电网的边中删除这个边\*/

auto iter2 = find(this->\_edge.begin(), this->\_edge.end(), lastEdge);

this->\_edge.erase(iter2, iter2 + 1);

branch.erase(branch.begin() + i, branch.begin() + i + 1);

Branch refresh = Branch(bufv2, \*this);

if (refresh.v != '?')

{

branch.push\_back(refresh);

}

branch.push\_back(Branch(bufv1, \*this));

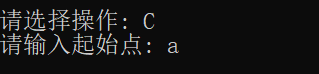
break;

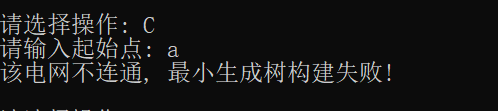
}

}

}

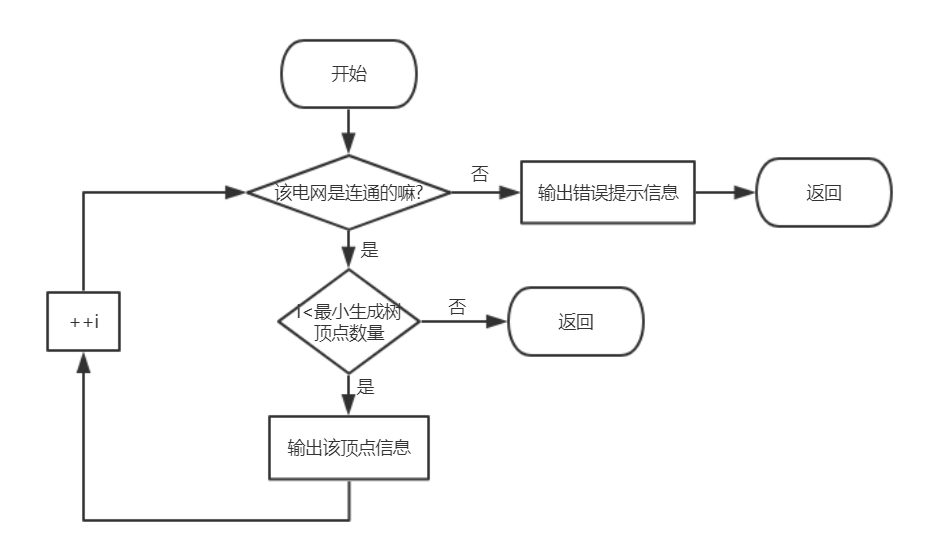
### 3.3.3 构造最小生成树功能截图示例





## 3.4 显示最小生成树功能的实现

### 3.4.1 显示最小生成树功能流程图



### 3.4.2 显示最小生成树功能核心代码

void PowerGrid::showSpanTree()

{

if (!this->isconnect)

{

cerr << "该电网不连通, 最小生成树输出失败!" << endl;

return;

}

for (int i = 0; i < this->\_minSpanTree.size(); ++i)

{

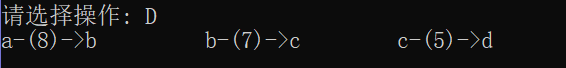
cout << \_minSpanTree[i] << " ";

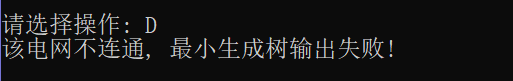
}

cout << endl;

}

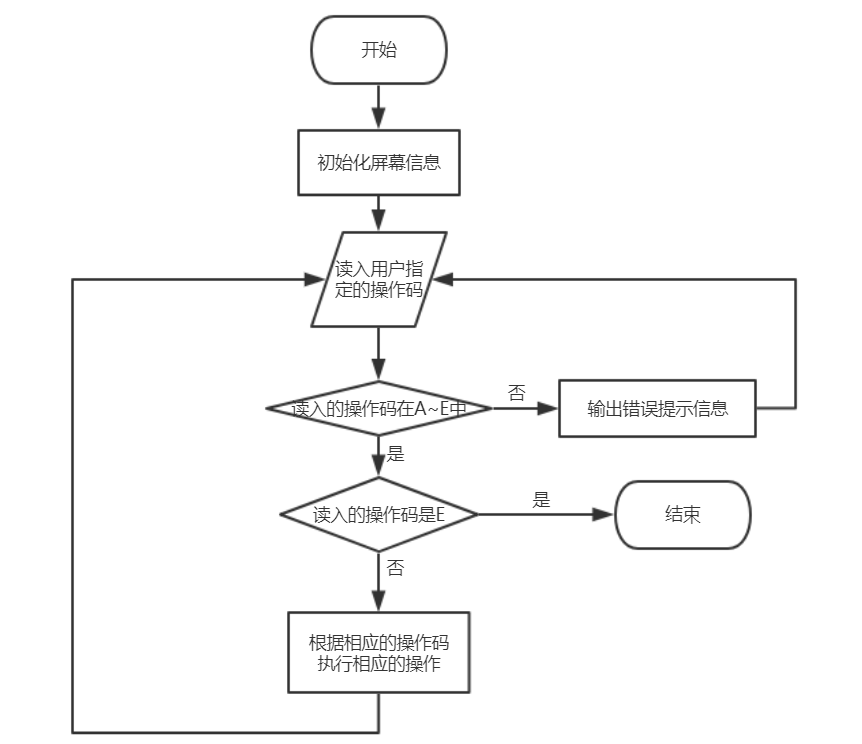
### 3.4.3 显示最小生成树功能截屏示例





## 3.6 总体系统的实现

### 3.6.1 总体系统流程图



### 3.6.2 总体系统核心代码

cout << "\*\* " << "电网造价模拟系统" << " \*\*" << endl;

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

//创建电网对象

PowerGrid grid;

/\*用户操作\*/

char OpCode;

while (1)

{

cout << endl << "请选择操作: ";

cin >> OpCode;

if (OpCode == 'E') { break; }

switch (OpCode)

{

case 'A':

{

int sumVervex;

while (true)

{

cout << "请输入顶点的个数: ";

cin >> sumVervex;

if (sumVervex > 0) { break; }

else

{

cerr << "顶点的个数为非负整数, 请重新输入" << endl;

}

}

getchar();

while (!grid.InitVertex(sumVervex))

{

;

}

break;

}

case 'B':

{

grid.addEdges();

break;

}

case 'C':

{

Vertex start;

while (true)

{

cout << "请输入起始点: ";

cin >> start;

if (grid.isVertex(start)) { break; }

else { cerr << "起始点必须为电网顶点, 请重新输入!" << endl; }

}

if (!grid.Prim(start))

{

cerr << "该电网不连通, 最小生成树构建失败!" << endl;

}

break;

}

case 'D':

{

grid.showSpanTree();

break;

}

default:

{

cout << "请输入大写字母A~E!" << endl;

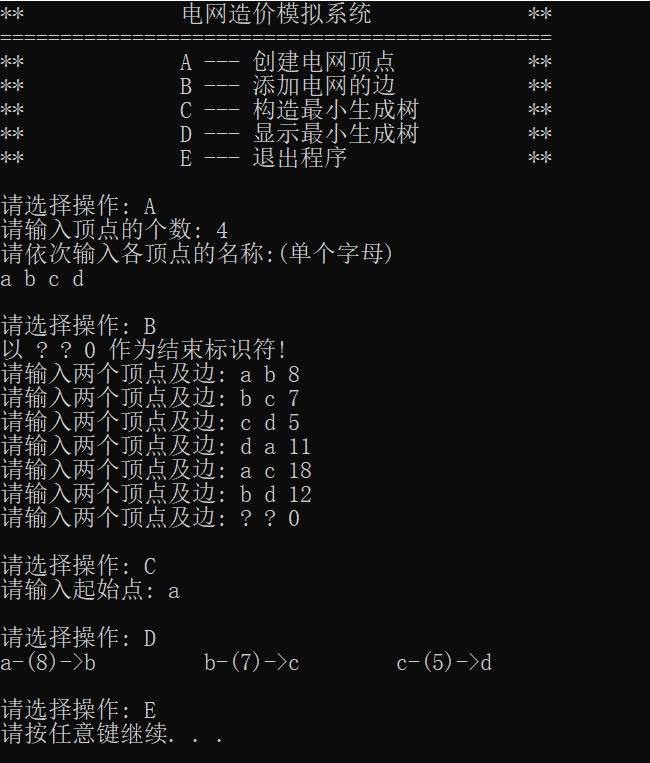
break;

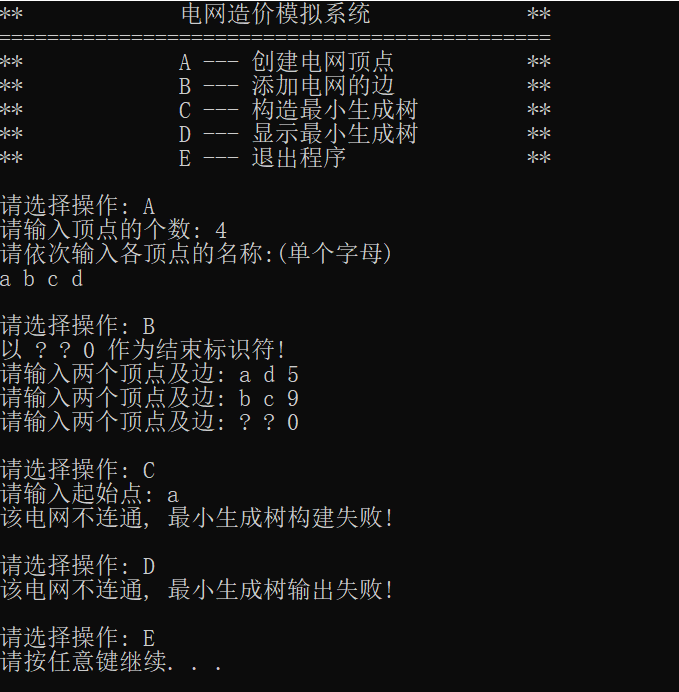
}

}

}

### 3.6.3 总体系统截屏示例





# 4 测试

## 4.1 功能测试

### 4.1.1 创建电网顶点功能测试

**前插法测试用例**：

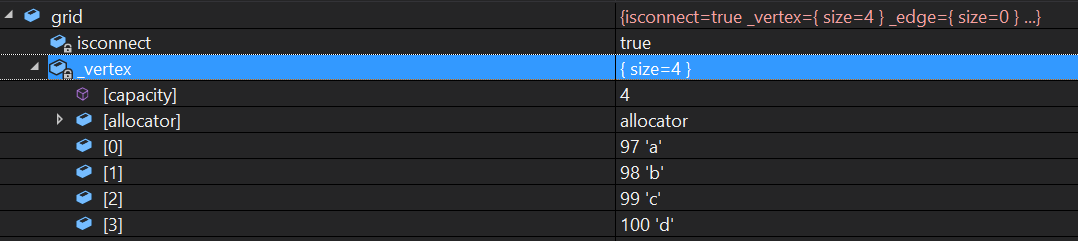
A

4

a b c d

**预期结果**：正确储存四个顶点名称

**实验结果**



### 4.1.2 添加电网的边功能测试

**测试用例：**

B

a b 8

b c 7

c d 5

d a 11

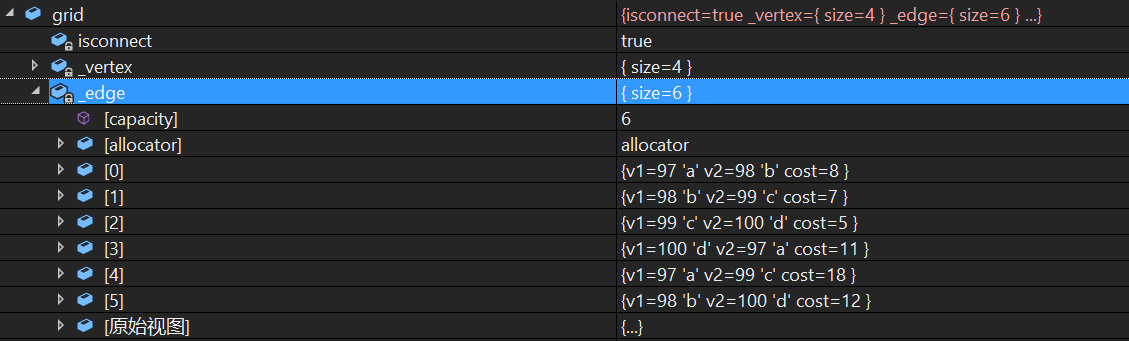
a c 18

b d 12

? ? 0

**预期结果：**正确储存电网的边

**实验结果：**

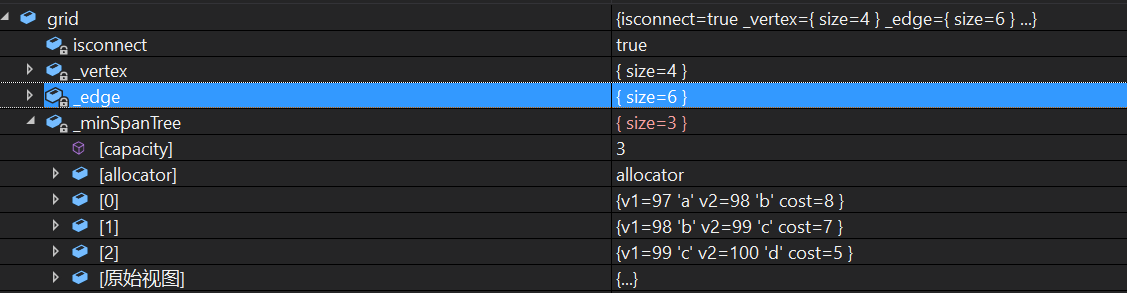


### 4.1.3 构建最小生成树功能测试

**测试用例：**C a

**预期结果：**正确生成该电网的最小生成树

**实验结果：**



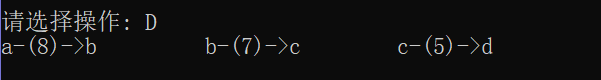
### 4.1.4 显示最小生成树功能测试

**测试用例：**D

**预期结果：**

输出 a-(8)->b b-(7)->c c-(5)->d

**实验结果：**



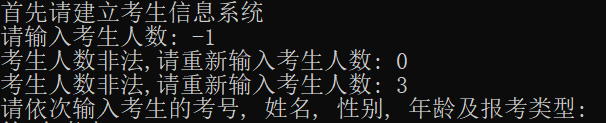
## 4.2 边界测试

### 4.2.1 初始化无输入数据

**测试用例：**考生人数分别输入-1, 0, 3

**预期结果：**给出错误提示，程序运行正常不崩溃，并提示用户重新输入。

**实验结果：**

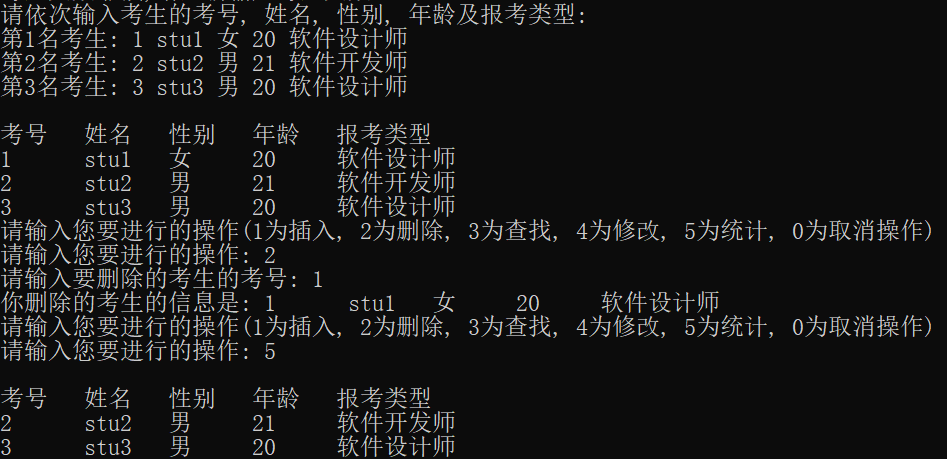


### 4.2.2 删除头结点

**测试用例：**删除头结点

**预期结果：**程序正常运行，不崩溃。

**实验结果：**

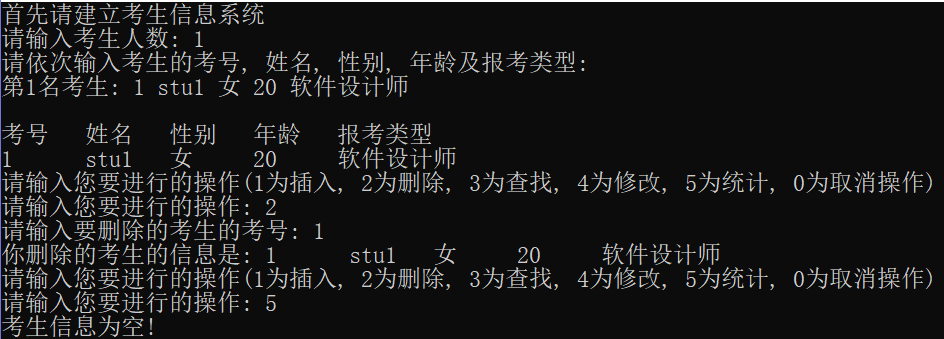


### 4.2.3 删除后链表为空

**测试用例：**删除前链表只有一个结点，删除后链表为空

**预期结果：**程序正常运行，不崩溃。

**实验结果：**



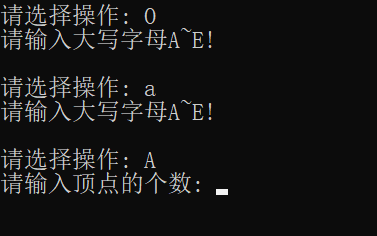
## 4.3 出错测试

### 4.3.1 输入操作码错误

**测试用例：** O a A分别作为操作码

**预期结果：** 如果不是大写字母A~E, 程序不崩溃, 并允许用户重新输入, 直到为合法的操作码为止

**实验结果：**

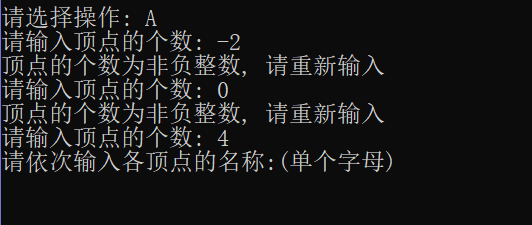


### 4.3.2 输入顶点个数为负数

**测试用例：** -2 0 4

**预期结果：** 只有当输入的顶点个数为正整数时程序继续询问各顶点的名称, 否则程序不崩溃, 并输出提示信息, 允许用户重新输入

**实验结果：**



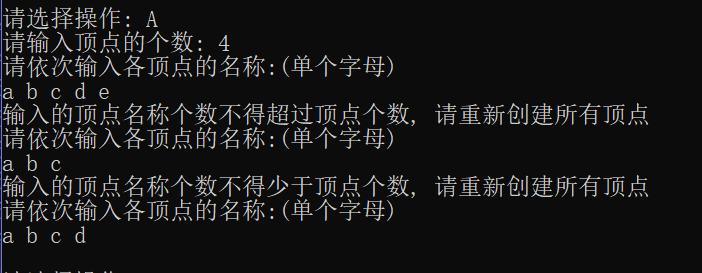
### 4.3.3 创建电网顶点时输入的顶点名称个数与顶点个数不匹配

**超过顶点个数测试用例：**a b c d e

**少于顶点个数测试用例：** a b c

**预期结果：** 当输入的顶点名称个数超过或少于顶点个数时程序不崩溃, 并能输出提示信息, 允许用户重复输入, 直到二者匹配为止

**实验结果：**



### 4.3.4 创建电网顶点时输入的顶点名称中有非字母的名称

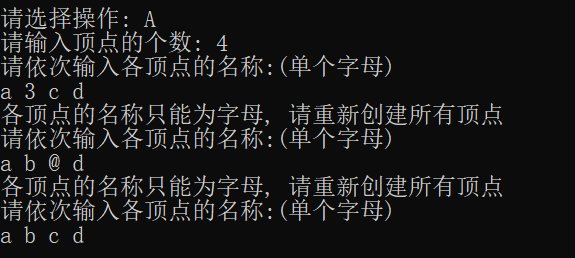
**测试用例：**

a 3 c d

a b @ d

**预期结果：**当输入的顶点名称中有非字母的情况, 程序不崩溃, 并允许用户重复输入, 直到输入的顶点名称全部合法为止

**实验结果：**



### 4.3.5 添加的边的某个顶点不是电网的顶点

**测试用例：**

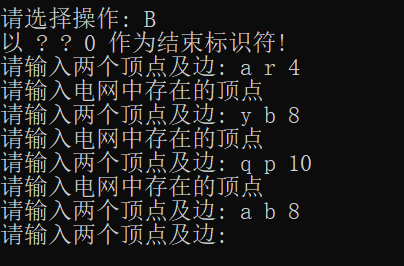
a r 4 //第二个顶点不在电网中

y b 8 //第一个顶点不在电网中

q p 10 //两个顶点都不在电网中

**预期结果：**当任意一个顶点不在电网中时, 程序不崩溃, 并能输出相应的提示信息告知用户输入有误, 并允许继续输入

**实验结果：**



### 4.3.6 添加的边中两顶点的花费不是正整数

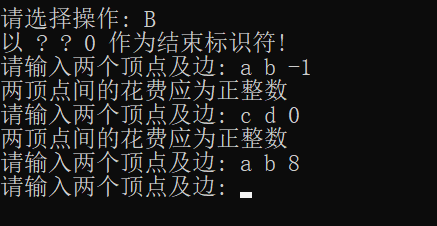
**测试用例：**

a b -1

c d 0

**预期结果：** 当任意两个顶点间的花费不是正整数时, 程序不崩溃, 并能输出相应的提示信息告知用户输入有误, 并允许继续输入

**实验结果：**

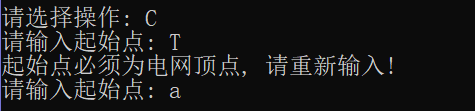


### 4.3.7 构建最小生成树的顶点不在电网中

**测试用例：** T

**预期结果：**程序不崩溃, 并能输出相应的提示信息告知用户输入有误, 并允许继续输入

**实验结果：**



### 4.3.8 电网不连通

**测试用例：**

A

4

a b c d

B

a d 5

b c 9

? ? 0

C

a

D

E

**预期结果：** 程序不崩溃, Prim算法可以正常调用, 并判断出该电网为非连通图, 最小生成树构建失败, 并且不会输出

**实验结果：**

