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：翟晨昊

学　　号：1952216

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

1.1 背景分析

电在我们的日常生活中占有非常重要的地位，如果没有电，无法想象世界会变成什么样子。城市中的电力一般都通过电网来进行运输，而电网的造价成本不菲，在保证全城都能通上电的前提下，如何将电网的造价成本降至最低，是一个非常有经济价值的问题。现在假设一个城市有 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。

1.2 功能分析

系统的功能要求是在每个小区之间都可以设置一条电网线路，但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。因此首先系统需要小区的布局构造出来，随后在小区之间增加线路。增加完毕后，就需要通过计算来得出其中的 n-1 条使总的耗费最少，并将它们显示出来。

综上所述，该电网建设造价模拟系统需要有创建电网顶点，添加电网的边，计算得出 n-1 条线路使总的耗费最少，输出构造方案的功能。

2 设计

2.1 数据结构设计

一个连通图的每一棵生成树，都是原图的一个极大无环子图。如果每个小区可以看作一个顶点，小区与小区之间的电网线路可以看作一条边，每条线路的造价看作边上的权值，这样就可以把整个电网看作一个连通网络。若一个连通网络由 n 个顶点组成，则其生成树必含 n 个结点、n-1 条边。由于我们要求建立一个造价最低的电网系统，那就要找出该联通网络的一棵最小生成树。

构造最小生成树的方法最典型的有两种，一种为 Kruskal 算法，一种为 Prim 算法。本系统采用了 Prim 算法。

Prim 算法是在不断迭代进行的。构建两个集合 V 与 V1，V 代表当前最小生成树的顶点集合，V1 代表不属于当前生成树的顶点集合。首先选定构造最小生成树的起始顶点 u0，将它加入到集合 V 中，随后选择一条边（u，v），要求 u 属于集合 V 而 v 属于集合 V1，且该边为满足该条件的权值最小的边。将 v 从 V1 取出加入到集合 V 中，然后继续这个过程，直到网络中的所有顶点都已经加入到生成树顶点集合 V 中。此时算法过程中选取的边的集合就可以构建出最小生成树。

如上分析所述，该电网造价模拟系统会有大量创建顶点，添加边的操作。但边不会很密集，因此采用用邻接表表示的图的数据结构来保存所有小区与线路。由于 Prim 算法每次都要获得权值最小的边，因此设计了一个最小堆的数据结构，每次从堆顶取出权值最小的边。在添加边等操作中，都需要找到图中的对应顶点，为了能快速找到顶点的位置，本系统设计了一个 AVL 二叉平衡树来存储所有顶点，方便查找。

2.2 类结构设计

如上分析所述，首先本系统有一个图类（Graph 类），以及图中所需的顶点类（Vertex 类）和边类（Edge 类）。其中 Graph 类与 Vertex 类通过友元来建立起联系，这样使得 Graph 类可以访问 Vertex 类。其次，为了给 Prim 算法提供最小权值的边，系统设计了一个堆类（Heap 类）与两个 function object，分别为 Greater 类与 Less 类，来帮助 Heap 类实现最大堆与最小堆；同时，为了方便查找顶点，本系统设计了一个 AVL 树（AVLTree 类）以及它的结点类：AVL 树结点类（AVLTreeNode 类），将 AVL 树都设置为对应结点类的友元，使得 AVL 树可以访问结点类。最后，本系统还设计了电网系统类（PowerGridSystem 类），给用户提供添加顶点，添加边，构建或显示最小生成树等功能的接口。系统运行过程中的数据主要是通过 Vector 向量类来保存。为了使数据结构更具有泛用性，本系统将 AVLTree 类，Heap 类，Graph 类等都设计为了模板类。

2.3 成员与操作设计

向量类（Vector）：

```
template <typename ElementType> class Vector {
public:
    ~Vector();
```

```
    Vector();
    ElementType& operator[](const int x);
    const ElementType& operator[](const int x)const;
    Vector<ElementType>(const Vector<ElementType>& rhs);
    Vector<ElementType>& operator=(const Vector<ElementType>& rhs);
    bool isFull();
    bool isEmpty();
    void pushBack(const ElementType& temp);
    void popBack();
    void clear();
    int getSize();
    void reSize(int newSize);
private:
    void extendSize();
    int size;
    int maxSize;
    ElementType* myVector;
};
```

私有成员：

int size;//Vector 中已经存储的元素数量
int maxSize;//Vector 中已经申请的空间大小，元素数量如果超过要再次申请
ElementType* myVector;//存储的元素序列的起始地址

私有操作：

void extendSize();
//在 Vector 申请的空间已经被占满时再次申请空间，每次扩充至 maxsize*2+1 是因为刚开始的 maxsize 为 0

公有操作：

Vector();
//构造函数，初始化指针并将 size 与 maxSize 置为 0

~Vector();
//析构函数，调用 clear()函数来删除元素，释放内存

ElementType& operator[](const int x);
//重载[]运算符，返回 ElementType&类型

const ElementType& operator[](const int x)const;
//重载[]运算符，返回 const ElementType&类型

Vector<ElementType>(const Vector<ElementType>& rhs);
//复制构造函数，将一个 Vector 复制给另一个 Vector

```
Vector<ElementType>& operator=(const Vector<ElementType>& rhs);
//重载=运算符，可以将一个 Vector 赋给另一个 Vector

bool isFull();
//判断是否 Vector 中申请的内存已经被占满

bool isEmpty();
//判断 Vector 是否为空

void pushBack(const ElementType& temp);
//在 Vector 末尾添加一个元素

void popBack();
//删除 Vector 最末尾的元素

void clear();
//删除 Vector 中的所有元素并释放内存

int getSize();
//返回 Vector 已经存储的元素数量

void reSize(int newSize);
//重设 Vector 的大小，若比之前小，则会抛弃多余的元素
```

AVL 树结点类（AVLTreeNode）：

```
template <typename Key, typename ElementType>
class AVLTreeNode {
public:
    friend class AVLTree<Key, ElementType>;
    AVLTreeNode() = default;
    AVLTreeNode(Key inputKey, ElementType inputData) :
        dataKey(inputKey), data(inputData) {}
    ~AVLTreeNode() = default;
private:
    Key dataKey;
    ElementType data;
    int height = 0;
    AVLTreeNode<Key, ElementType>* left = nullptr;
    AVLTreeNode<Key, ElementType>* right = nullptr;
};
```

私有成员：

Key dataKey;//AVL 树结点的关键码，作为搜索依据
ElementType data;//AVL 树结点中保存的数据信息
int height;//AVL 树结点当前所处的高度，用于保持平衡
AVLTreeNode<Key, ElementType>* left;//指针域，指向当前结点的左子女
AVLTreeNode<Key, ElementType>* right;//指针域，指向当前结点的右子女

公有操作：

```
friend class AVLTree<Key, ElementType>;
//将 AVLTree 声明为友元

AVLTreeNode() = default;
//默认构造函数

AVLTreeNode(Key inputKey, ElementType inputData);
//含参构造函数

~AVLTreeNode() = default;
//默认析构函数
```

AVL 树类（AVLTree）：

```
template <typename Key, typename ElementType>
class AVLTree {
public:
    friend class Genealogy;
    AVLTree() :root(nullptr), size(0) {}
    ~AVLTree();
    ElementType getData(AVLTreeNode<Key, ElementType>* ptr);
    void makeEmpty();
    AVLTreeNode<Key, ElementType>* find(Key findK);
    AVLTreeNode<Key, ElementType>* insert(Key insertK, ElementType inse
rtData);
    bool remove(Key removeK);
    void change(Key changeOldK, Key changeNewK);
private:
    int  max(int x, int y);
    int  getHeight(AVLTreeNode<Key, ElementType>* ptr);
    AVLTreeNode<Key, ElementType>* findPriorNode(AVLTreeNode<Key, Eleme
ntType>* ptr);
    AVLTreeNode<Key, ElementType>* findNextNode(AVLTreeNode<Key, Elemen
tType>* ptr);
    AVLTreeNode<Key, ElementType>* rotateL(AVLTreeNode<Key, ElementType
>* ptr);
```

```
    AVLTreeNode<Key, ElementType>* rotateR(AVLTreeNode<Key, ElementType
>* ptr);
    AVLTreeNode<Key, ElementType>* rotateLR(AVLTreeNode<Key, ElementTyp
e>* ptr);
    AVLTreeNode<Key, ElementType>* rotateRL(AVLTreeNode<Key, ElementTyp
e>* ptr);
    void makeEmpty(AVLTreeNode<Key, ElementType>* ptr);
    AVLTreeNode<Key, ElementType>* find(Key findK, AVLTreeNode<Key, Ele
mentType>* ptr);
    AVLTreeNode<Key, ElementType>* insert(Key insertK, ElementType inse
rtData, AVLTreeNode<Key, ElementType>* ptr);
    AVLTreeNode<Key, ElementType>* remove(Key removeK, AVLTreeNode<Key,
 ElementType>* ptr);
    AVLTreeNode<Key, ElementType>* root;
    int size;
};
```

私有成员：

int size;//AVL 树中的结点个数
AVLTreeNode<Key, ElementType>* root;//AVL 树的根节点

私有操作：

int max(int x, int y);
//返回 x，y 中的最大值，用来修改结点的高度

int getHeight(AVLTreeNode<Key, ElementType>* ptr);
//返回结点 ptr 的高度

AVLTreeNode<Key, ElementType>* findPriorNode(AVLTreeNode<Key,
ElementType>* ptr);
//找到在中序遍历下 ptr 前面的结点

AVLTreeNode<Key, ElementType>* findNextNode(AVLTreeNode<Key,
ElementType>* ptr);
//找到在中序遍历下 ptr 后面的结点

AVLTreeNode<Key, ElementType>* rotateL(AVLTreeNode<Key, ElementType>*
ptr);
//左单旋转

AVLTreeNode<Key, ElementType>* rotateR(AVLTreeNode<Key, ElementType>*
ptr);
//右单旋转

```
AVLTreeNode<Key, ElementType>* rotateLR(AVLTreeNode<Key,
ElementType>* ptr);
//先左后右双旋转

AVLTreeNode<Key, ElementType>* rotateRL(AVLTreeNode<Key,
ElementType>* ptr);
//先右后左双旋转

void makeEmpty(AVLTreeNode<Key, ElementType>* ptr);
//将 ptr 结点及 ptr 结点的所有子女结点全部删除并释放内存
AVLTreeNode<Key, ElementType>* find(Key findK, AVLTreeNode<Key,
ElementType>* ptr);
//找到以结点 ptr 为根节点的子树中关键码为 findK 的结点

AVLTreeNode<Key, ElementType>* insert(Key insertK, ElementType
insertData, AVLTreeNode<Key, ElementType>* ptr);
//将关键码为 insertK，数据信息为 insertData 的结点插入以结点 ptr 为根节
点的子树中

AVLTreeNode<Key, ElementType>* remove(Key removeK, AVLTreeNode<Key,
ElementType>* ptr);
//将以结点 ptr 为根节点的子树中关键码为 removeK 的结点删除
```

公有操作：

```
friend class Genealogy;
//将 Genealogy 声明为友元

AVLTree();
//无参构造函数

~AVLTree();
//析构函数，通过调用 makeEmpty()函数来删除元素，释放内存

ElementType getData(AVLTreeNode<Key, ElementType>* ptr);
//获得结点 ptr 中保存的数据信息

void makeEmpty();
//将 AVL 树中的所有结点删除并释放内存

AVLTreeNode<Key, ElementType>* find(Key findK);
//找到 AVL 树中关键码为 findK 的结点

AVLTreeNode<Key, ElementType>* insert(Key insertK, ElementType
insertData);
```

//将关键码为 insertK，数据信息为 insertData 的结点插入 AVL 树中

bool remove(Key removeK);
//将 AVL 树中关键码为 removeK 的结点删除

void change(Key changeOldK, Key changeNewK);
//将 AVL 树中关键码为 changeOldK 的结点的关键码改为 changeNewK

堆类（Heap）：

```
template<typename ElementType, typename Comparator> class Heap{
public:
    Heap() = default;
    ~Heap();
    int getSize();
    void build(Vector<ElementType>& everyLength);
    void insert(const ElementType& inputData);
    bool pop();
    ElementType getTop();
    void makeEmpty();
private:
    Vector<ElementType> data;
    void siftDown(int start, int max);
    void siftUp(int start);
};
```

私有成员：

Vector<ElementType> data;//保存堆中的数据

私有操作：

void siftDown(int start, int max);
//从结点 start 到结点 max 为止下滑调整堆中数据

void siftUp(int start);
//从结点 start 到最上方上滑调整堆中数据

公有操作：

Heap() = default;
//默认构造函数

~Heap();
//析构函数，调用 makeEmpty()函数来删除元素，释放内存

int getSize();

```
//返回堆已经存储的元素数量

void build(Vector<ElementType>& everyLength);
//通过 Vector 构建堆

void insert(const ElementType& inputData);
//向堆中插入一个元素

bool pop();
//删除堆中的一个元素

ElementType getTop();
//返回堆顶元素

void makeEmpty();
//删除堆中的所有元素并释放内存
```

大于比较类（Greater）：

```
template<typename ElementType> class Greater{
public:
    bool operator()(const ElementType& E1, const ElementType& E2)
    {
        return *E1 > *E2;
    }
};
```

公有操作：

```
bool operator()(const ElementType& E1, const ElementType& E2);
//当 E1>E2 时返回 true
```

小于比较类（Less）：

```
template<typename ElementType> class Less{
public:
    bool operator()(const ElementType& E1, const ElementType& E2)
    {
        return *E1 < *E2;
    }
};
```

公有操作：

```
bool operator()(const ElementType& E1, const ElementType& E2);
//当 E1<E2 时返回 true
```

边类（Edge）：

```
template<typename NameType, typename ElementType>
class Edge {
public:
    Edge() = default;
    Edge(int vertex1, int vertex2, ElementType weight) :
        current(vertex1), dest(vertex2), cost(weight) {}
    friend bool operator<(const Edge<NameType, ElementType>& rhs1,const
 Edge<NameType, ElementType>& rhs2)
    {
        return rhs1.cost < rhs2.cost;
    }
    friend bool operator>(const Edge<NameType, ElementType>& rhs1,const
 Edge<NameType, ElementType>& rhs2)
    {
        return rhs1.cost > rhs2.cost;
    }
    int current;
    int dest;
    ElementType cost;
    Edge<NameType, ElementType>* next = nullptr;
};
```

公有成员：

int current;//边相连的当前顶点位置
int dest;//边相连的另一顶点位置
ElementType cost;//边上的权值
Edge<NameType, ElementType>* next;//当前顶点的下一条边

公有操作：

Edge() = default;
//默认构造函数

Edge(int vertex1, int vertex2, ElementType weight);
//含参构造函数

friend bool operator<(const Edge<NameType, ElementType>& rhs1,const Edge<NameType, ElementType>& rhs2);
//重载小于号，并声明为友元

friend bool operator>(const Edge<NameType, ElementType>& rhs1,const Edge<NameType, ElementType>& rhs2);
//重载大于号，并声明为友元

顶点类（Vertex）：

```
template<typename NameType, typename ElementType>
class Vertex {
public:
    friend class Graph<NameType, ElementType>;
    Vertex() = default;
    Vertex(int inputOrder, NameType inputName) :
        order(inputOrder), name(inputName) {}
private:
    int order;
    NameType name;
    Edge<NameType, ElementType>* head = nullptr;
};
```

私有成员：

int order;//该顶点在图中的位置
NameType name;//该顶点的名称
Edge<NameType, ElementType>* head;//与该顶点相连的第一条边

公有操作：

friend class Graph<NameType, ElementType>;
//将 Graph 声明为友元

Vertex() = default;
//默认构造函数

Vertex(int inputOrder, NameType inputName);
//含参构造函数

图类（Graph）：

```
template<typename NameType, typename ElementType>
class Graph {
public:
    friend class PowerGridSystem;
    Graph() = default;
    ~Graph();
    void clear();
    NameType getName(int order);
    Edge<NameType, ElementType>* getFirstEdge(int order);
    Edge<NameType, ElementType>* getNextEdge(Edge<NameType, ElementType
>* curEdge);
    void insertVertex(NameType inputVertex);
    void insertEdge(int order1, int order2, ElementType weight);
```

```
private:
    bool checkEdge(int order1, int order2);
    Vector<Vertex<NameType, ElementType>*> vertexTable;
};
```

私有成员：

Vector<Vertex<NameType, ElementType>*> vertexTable;
//存储图中的所有顶点

私有操作：

bool checkEdge(int order1, int order2);
//检查两个位置所对应的顶点之间是否已经存在边

公有操作：

friend class PowerGridSystem;
//将 PowerGridSystem 声明为友元

Graph() = default;
//默认构造函数

~Graph();
//析构函数，调用 clear()函数来删除元素，释放内存

void clear();
//删除图中的所有元素并释放内存

NameType getName(int order);
//获得该位置所对应的顶点的名称

Edge<NameType, ElementType>* getFirstEdge(int order);
//返回该位置所对应的顶点的第一条边

Edge<NameType, ElementType>* getNextEdge(Edge<NameType, ElementType>* curEdge);
//返回该边相连的下一条边

void insertVertex(NameType inputVertex);
//向图中加入顶点

void insertEdge(int order1, int order2, ElementType weight);
//向两个位置所对应的顶点之间加入边

电网系统类（PowerGridSystem）：

```
class PowerGridSystem {
public:
    void menu();
    void build();
    void addEdge();
    void buildPrim();
    void display();
private:
    int getOrder();
    int getMaxAddEdge();
    bool checkName(Vector<string>& nodeName);
    int curEdgeNum = 0;
    Graph<string, int> powerGrid;
    AVLTree<string, int> vertexTree;
    Vector<Edge<string, int>*> result;
};
```

私有成员：

int curEdgeNum;//当前图中边的个数
Graph<string, int> powerGrid;//用来储存电网的图
AVLTree<string, int> vertexTree;//保存了所有顶点的 AVL 树
Vector<Edge<string, int>*> result;//计算出的最小生成树中边的集合

私有操作：

int getOrder();
//获得图中能够插入顶点的位置

int getMaxAddEdge();
//计算当前图中所能容纳的最多边数

bool checkName(Vector<string>& nodeName);
//检查创建的新顶点的名称是否重复或者已经在图中存在

公有操作：

void menu();
//用户操作菜单

void build();
//创建电网顶点

void addEdge();
//添加电网的边

```
void buildPrim();
//构造最小生成树

void display();
//显示最小生成树
```

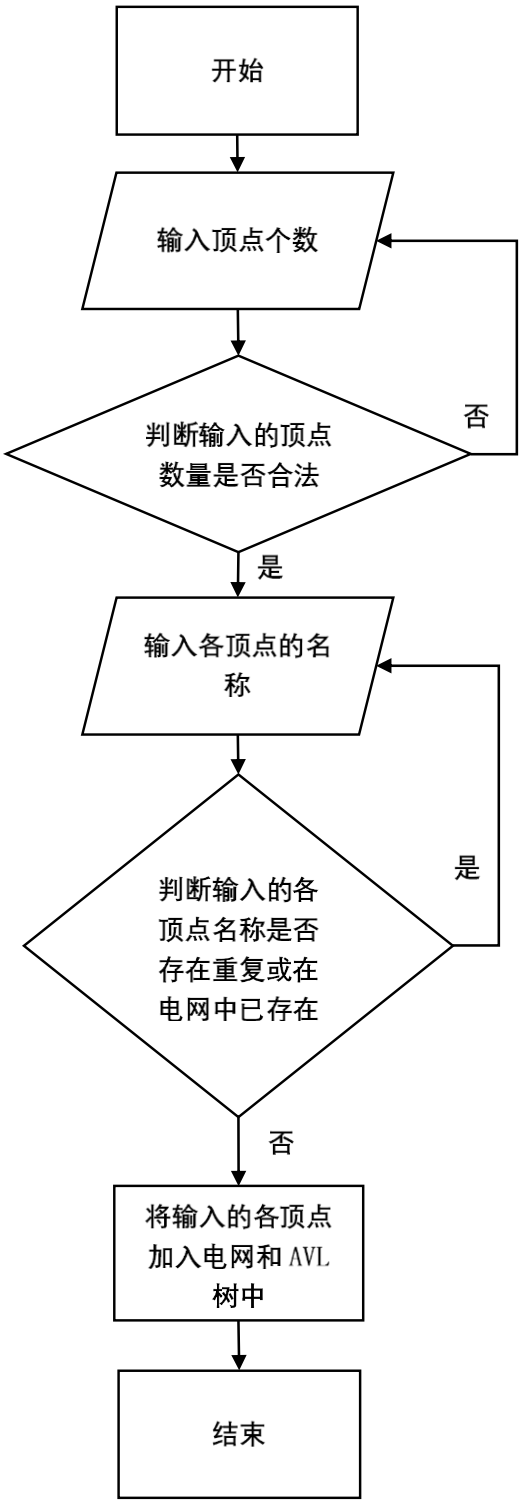
2.4 系统设计

系统首先会在屏幕上显示一个用户操作菜单，根据此菜单，用户通过输入对应的操作码来执行相应的操作。功能有创建电网顶点，添加电网的边，构造最小生成树，显示最小生成树，退出程序等等。

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

PowerGridSystem 类中：

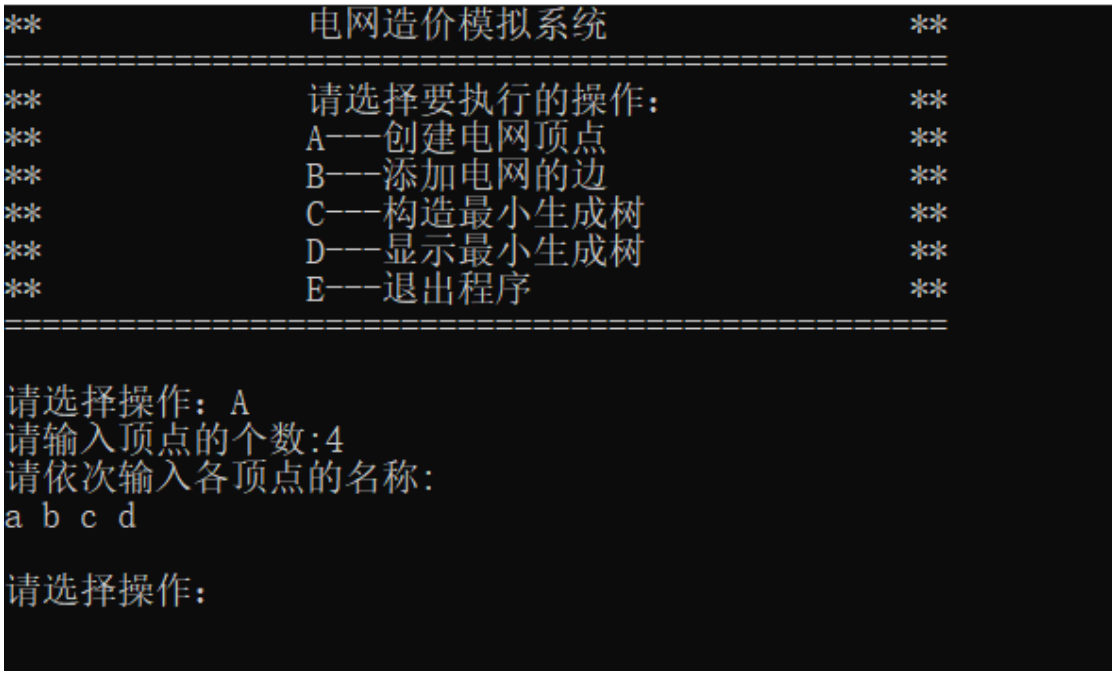
```
void PowerGridSystem::build()
{
    int vertexNum = 0;
    Vector<string> vertexName;
    cout << "请输入顶点的个数:";
    cin >> vertexNum;
    while (vertexNum <= 0 || cin.fail())
    {
        cout << "顶点的个数只能是正整数!" << endl;
        cout << "请重新输入顶点个数:";
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cin >> vertexNum;
    }
    cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    vertexName.reSize(vertexNum);
    cout << "请依次输入各顶点的名称:" << endl;
    for (int i = 0; i < vertexNum; i++)
    {
        cin >> vertexName[i];
    }
    while (!checkName(vertexName))
    {
        cout << "请重新依次输入各顶点的名称:" << endl;
        for (int i = 0; i < vertexNum; i++)
        {
            cin >> vertexName[i];
        }
    }
    for (int i = 0; i < vertexNum; i++)
    {
        vertexTree.insert(vertexName[i], getOrder());
        powerGrid.insertVertex(vertexName[i]);
    }
}

bool PowerGridSystem::checkName(Vector<string>& nodeName)
{
    for (int i = 0; i < nodeName.getSize() - 1; i++)
    {
```

```
        for (int j = i + 1; j < nodeName.getSize(); j++)
        {
            if (nodeName[i] == nodeName[j])
            {
                cout << "创建的新顶点中含有重复的名称!" << endl;
                return false;
            }
        }
    }
    for (int i = 0; i < nodeName.getSize(); i++)
    {
        if (vertexTree.find(nodeName[i]) != nullptr)
        {
            cout << "存在新顶点名称与系统中已存在的顶点名称重复!" << endl;
            return false;
        }
    }
    return true;
}
```

Graph 类中：

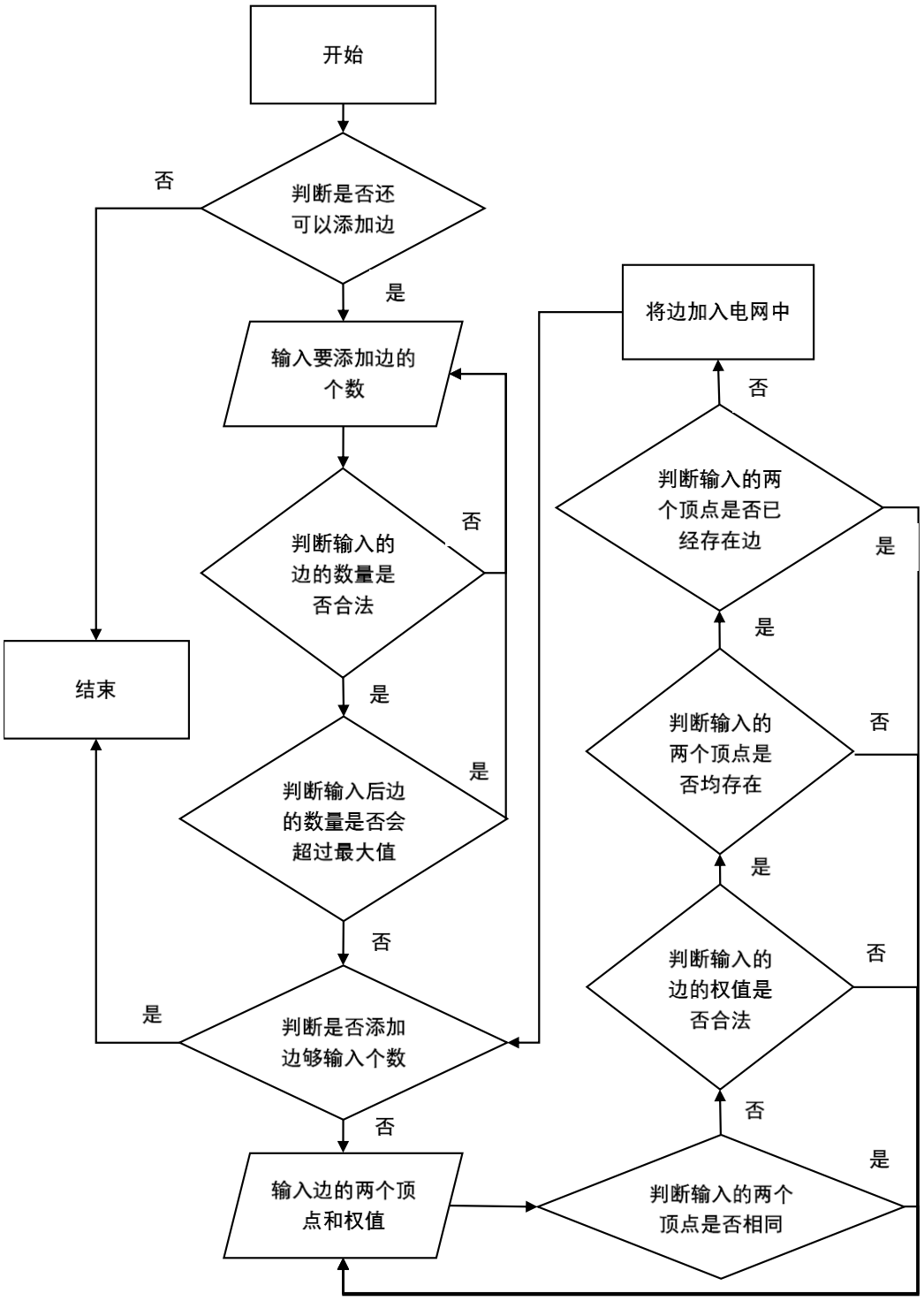
```
template<typename NameType, typename ElementType>
void Graph<NameType, ElementType>::insertVertex(NameType inputName)
{
    Vertex<NameType, ElementType>* newVertex = new Vertex<NameType, Ele
mentType>(vertexTable.getSize(), inputName);
    vertexTable.pushBack(newVertex);
}
```

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

PowerGridSystem 类中：

```
void PowerGridSystem::addEdge()
{
    string name1;
    string name2;
    AVLTreeNode<string, int>* vertex1 = nullptr;
    AVLTreeNode<string, int>* vertex2 = nullptr;
    int order1 = 0;
    int order2 = 0;
    int weight = 0;
    int edgeNum = 0;
    if (curEdgeNum >= getMaxAddEdge())
    {
        cout << "目前边的个数已至最大值，无法再添加！" << endl;
        return;
    }
    cout << "请输入添加边的个数:";
    cin >> edgeNum;
    while (edgeNum <= 0 || cin.fail() || edgeNum > (getMaxAddEdge() - c
urEdgeNum))
    {
        if (edgeNum <= 0 || cin.fail())
        {
            cout << "边的个数只能是正整数!" << endl;
        }
        else
        {
            cout << "输入后边的数量会超过最大值!" << endl;
        }
        cout << "请重新输入边个数:";
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cin >> edgeNum;
    }
    curEdgeNum += edgeNum;
    cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    for (int i = 0; i < edgeNum; i++)
    {
        cout << "请输入两个顶点及边:";
        while (true)
        {
```

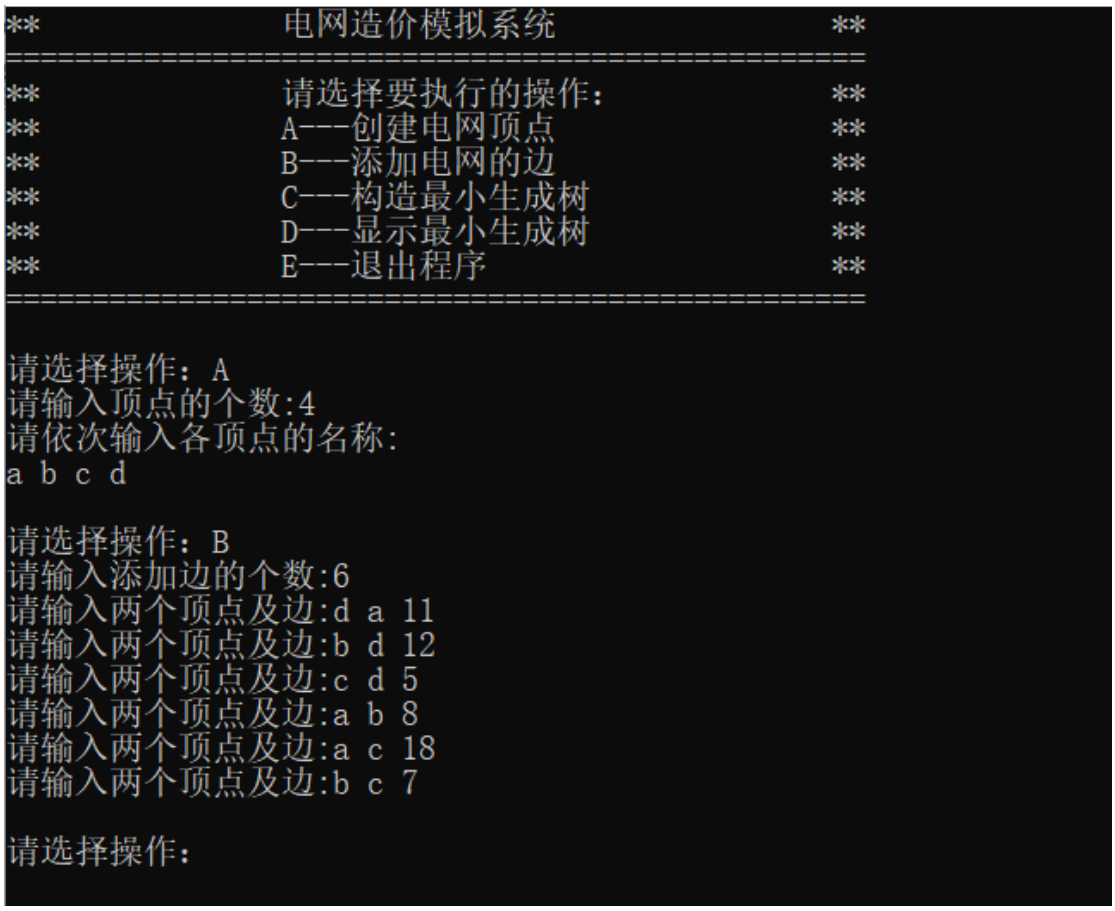
```
            cin >> name1 >> name2 >> weight;
            vertex1 = vertexTree.find(name1);
            vertex2 = vertexTree.find(name2);
            if (name1 == name2)
            {
                cout << "输入的两个顶点相同!";
            }
            else if (vertex1 == nullptr)
            {
                cout << "第一个顶点不存在!";
            }
            else if (vertex2 == nullptr)
            {
                cout << "第二个顶点不存在!";
            }
            else if (weight <= 0 || cin.fail())
            {
                cout << "每一条边的花费必须是正整数!";
            }
            else
            {
                order1 = vertexTree.getData(vertex1);
                order2 = vertexTree.getData(vertex2);
                if (powerGrid.checkEdge(order1, order2))
                {
                    powerGrid.insertEdge(order1, order2, weight);
                    break;
                }
                else
                {
                    cout << "此两个顶点之间已经存在边!";
                }
            }
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << endl;
            cout << "请重新输入两个顶点及边:";
        }
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }
}
```

Graph 类中：

```
template<typename NameType, typename ElementType>
bool Graph<NameType, ElementType>::checkEdge(int order1, int order2)
{
    Vertex<NameType, ElementType>* pFirst = vertexTable[order1];
    Edge<NameType, ElementType>* pTemp = pFirst->head;
    while (pTemp != nullptr)
    {
        if (pTemp->dest == order2)
        {
            return false;
        }
        pTemp = pTemp->next;
    }
    return true;
}

template<typename NameType, typename ElementType>
void Graph<NameType, ElementType>::insertEdge(int order1, int order2, E
lementType weight)
{
    Vertex<NameType, ElementType>* pFirst = vertexTable[order1];
    Vertex<NameType, ElementType>* pSecond = vertexTable[order2];
    Edge<NameType, ElementType>* pTemp = pFirst->head;
    if (pTemp == nullptr)
    {
        pFirst->head = new Edge<NameType, ElementType>(order1, order2,
weight);
    }
    else
    {
        while (pTemp->next != nullptr)
        {
            pTemp = pTemp->next;
        }
        pTemp->next = new Edge<NameType, ElementType>(order1, order2, w
eight);
    }
    pTemp = pSecond->head;
    if (pTemp == nullptr)
    {
        pSecond->head = new Edge<NameType, ElementType>(order2, order1,
 weight);
    }
```

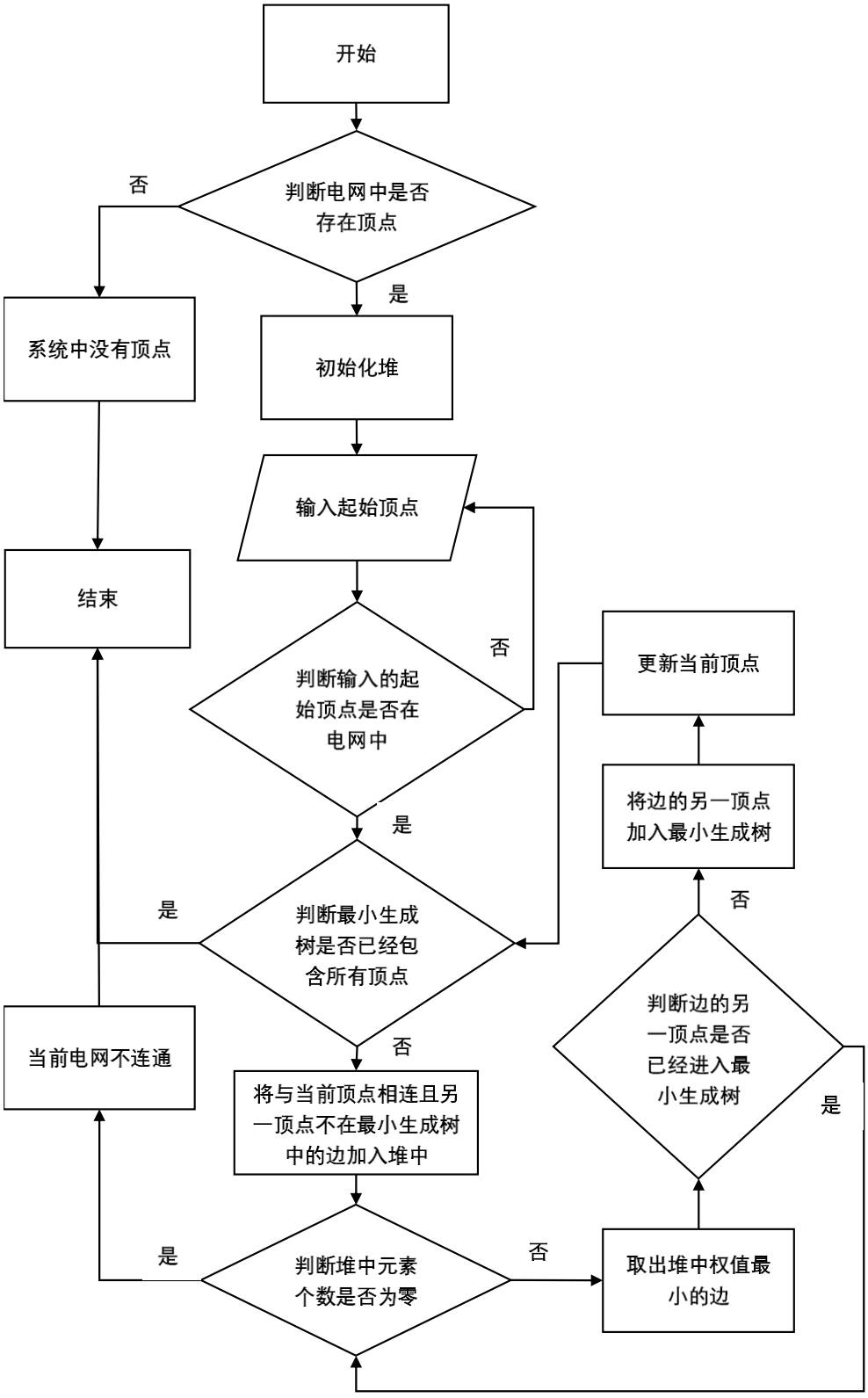
```
    else
    {
        while (pTemp->next != nullptr)
        {
            pTemp = pTemp->next;
        }
        pTemp->next = new Edge<NameType, ElementType>(order2, order1, w
eight);
    }
}
```

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

PowerGridSystem 类中：

```
void PowerGridSystem::buildPrim()
{
    if (getOrder() == 0)
    {
        cout << "系统中没有顶点!" << endl;
        return;
    }
    else if (getOrder() == 1)
    {
        cout << "系统中没有边!" << endl;
        return;
    }
    int num = getOrder();
    int count = 1;
    int order = 0;
    Edge<string, int>* nextEdge = nullptr;
    Edge<string, int>* minEdge = nullptr;
    string init;
    Heap<Edge<string,int>*,Greater<Edge<string, int>*> > heap;
    AVLTreeNode<string, int>* initVertex = nullptr;
    bool* isIn = new bool[num];
    result.clear();
    cout << "请输入起始顶点:";
    cin >> init;
    initVertex = vertexTree.find(init);
    while (initVertex == nullptr)
    {
        cout << "此顶点不在系统之中!" << endl;
        cout << "请重新输入: ";
        cin >> init;
        initVertex = vertexTree.find(init);
    }
    for (int i = 0; i < num; i++)
    {
        isIn[i] = false;
    }
    order = vertexTree.getData(initVertex);
    isIn[order] = true;
    while (count < num)
    {
        nextEdge = powerGrid.getFirstEdge(order);
```

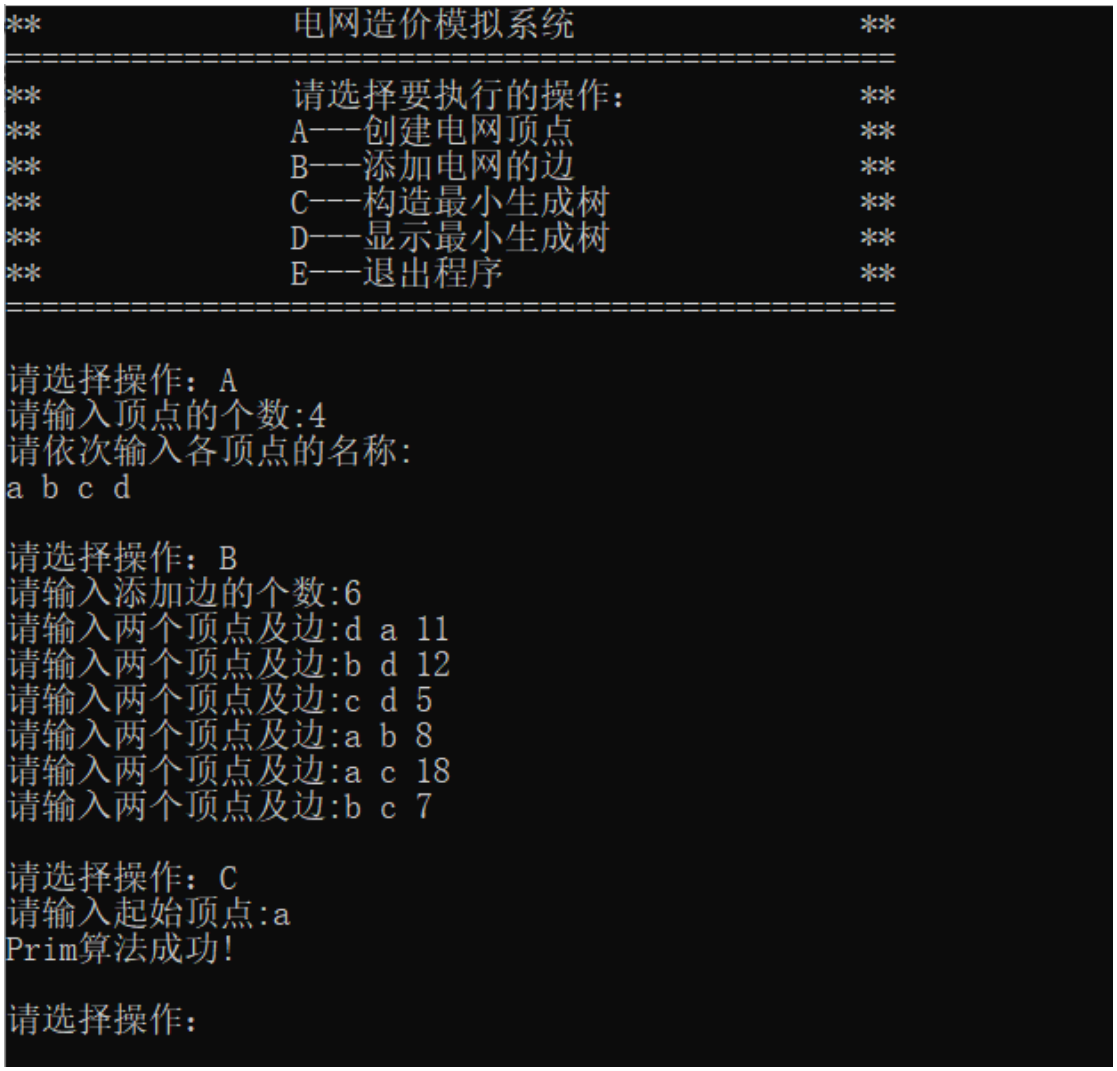
```
        while (nextEdge != nullptr)
        {
            if (isIn[nextEdge->dest] == false)
            {
                heap.insert(nextEdge);
            }
            nextEdge = powerGrid.getNextEdge(nextEdge);
        }
        if (heap.getSize() == 0)
        {
            cout << "目前的电力系统并未连通!" << endl;
            delete[] isIn;
            isIn = nullptr;
            return;
        }
        while (heap.getSize() != 0)
        {
            minEdge = heap.getTop();
            if (isIn[minEdge->dest] == false)
            {
                isIn[minEdge->dest] = true;
                order = minEdge->dest;
                result.pushBack(minEdge);
                heap.pop();
                count++;
                break;
            }
            else
            {
                heap.pop();
            }
        }
    }
    delete[] isIn;
    isIn = nullptr;
    cout << "Prim 算法成功!" << endl;
}
```

Graph 类中：

```
template<typename NameType, typename ElementType>
Edge<NameType, ElementType>* Graph<NameType, ElementType>::getFirstEdge
(int order)
{
    Vertex<NameType, ElementType>* pTemp = vertexTable[order];
```

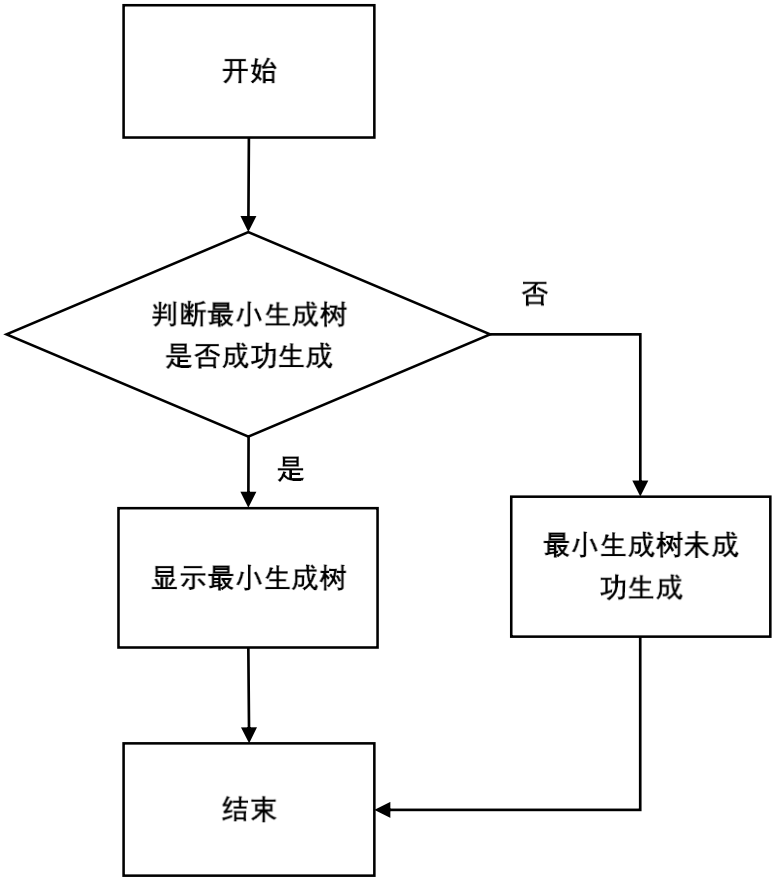
```
    Edge<NameType, ElementType>* pHead = pTemp->head;
    if (pHead != nullptr)
    {
        return pHead;
    }
    else
    {
        return nullptr;
    }
}

template<typename NameType, typename ElementType>
Edge<NameType, ElementType>* Graph<NameType, ElementType>::getNextEdge(
Edge<NameType, ElementType>* curEdge)
{
    return curEdge->next;
}
```

3.3.3 构造最小生成树功能截屏示例



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

PowerGridSystem 类中：

```
void PowerGridSystem::display()
{
    Vertex<string, int>* pCurrent = nullptr;
    Vertex<string, int>* pDest = nullptr;
    if ((result.getSize() != (getOrder() - 1)) || (getOrder() == 1))
    {
        cout << "最小生成树未成功生成!" << endl;
        return;
    }
    cout << "最小生成树的顶点及边为:" << endl;
    for (int i = 0; i < result.getSize(); i++)
    {
        if (i % 4 == 0)
        {
            cout << endl;
        }
        cout << powerGrid.getName(result[i]->current) << "-
           <" << result[i]->cost << ">-
           " << powerGrid.getName(result[i]->dest) << "    ";
    }
}
```

Graph 类中：

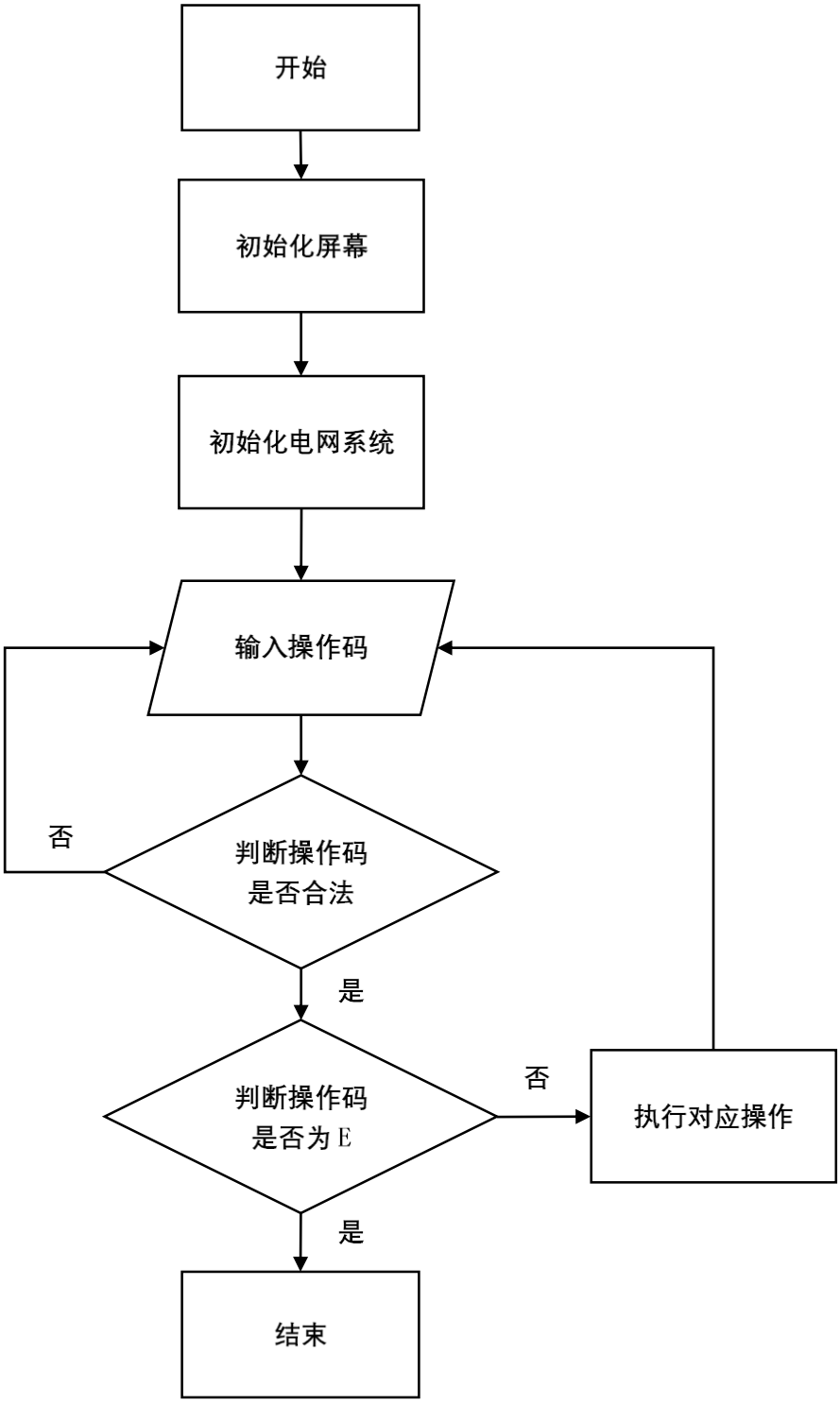
```
template<typename NameType, typename ElementType>
NameType Graph<NameType, ElementType>::getName(int order)
{
    Vertex<NameType, ElementType>* pTemp = vertexTable[order];
    return pTemp->name;
}
```

3.4.3 显示最小生成树功能截屏示例

```
**                电网造价模拟系统                **
==================================================
**                请选择要执行的操作:              **
**                A---创建电网顶点                **
**                B---添加电网的边                **
**                C---构造最小生成树              **
**                D---显示最小生成树              **
**                E---退出程序                    **
==================================================

请选择操作：A
请输入顶点的个数:4
请依次输入各顶点的名称:
a b c d

请选择操作：B
请输入添加边的个数:6
请输入两个顶点及边:d a 11
请输入两个顶点及边:b d 12
请输入两个顶点及边:c d 5
请输入两个顶点及边:a b 8
请输入两个顶点及边:a c 18
请输入两个顶点及边:b c 7

请选择操作：C
请输入起始顶点:a
Prim算法成功!

请选择操作：D
最小生成树的顶点及边为:

a-<8>-b    b-<7>-c    c-<5>-d
请选择操作:
```

3.5 总体功能的实现

3.5.1 总体功能流程图



3.5.2 总体功能核心代码

```
int main()
{
    PowerGridSystem system;
    system.menu();
    string Cmd;
    while (true)
    {
        cout << endl;
        cout << "请选择操作：";
        cin >> Cmd;
        switch (Cmd[0])//string 类型不能用于 switch
        {
        case 'A':
            system.build();
            break;
        case 'B':
            system.addEdge();
            break;
        case 'C':
            system.buildPrim();
            break;
        case 'D':
            system.display();
            break;
        case 'E':
            cout << "成功退出系统！" << endl;
            return 0;
        default:
            cout << "操作数输入不正确，请重新输入!";
            break;
        }
    }
    return 0;
}
```

3.5.3 总体功能截屏示例

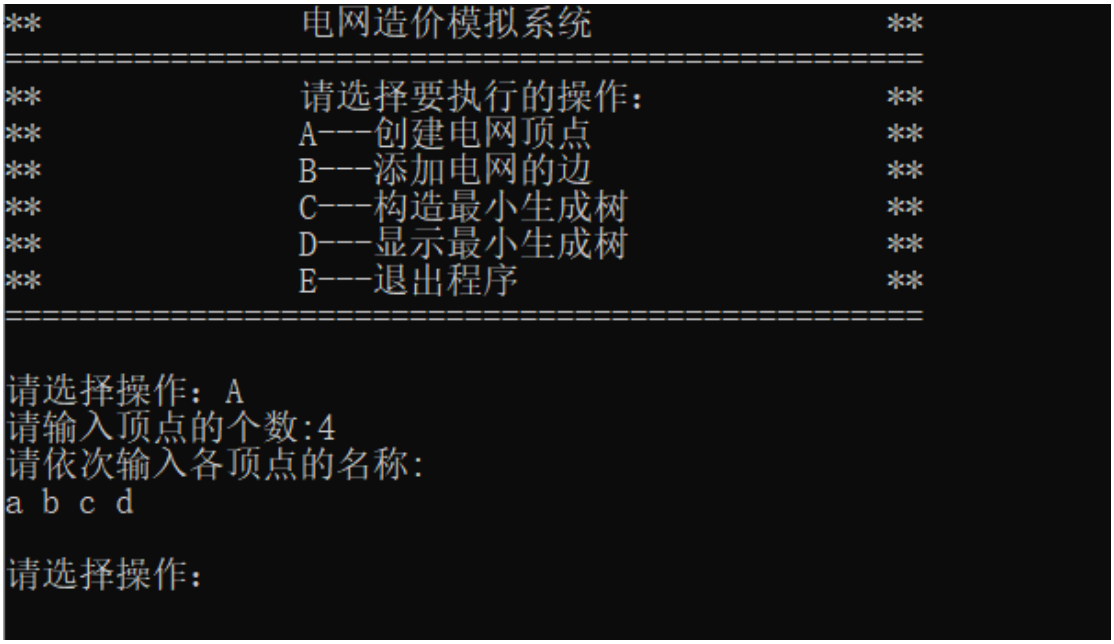
```
**                电网造价模拟系统                **
==================================================
**                请选择要执行的操作:            **
**                A---创建电网顶点                **
**                B---添加电网的边                **
**                C---构造最小生成树              **
**                D---显示最小生成树              **
**                E---退出程序                    **
==================================================

请选择操作：A
请输入顶点的个数:4
请依次输入各顶点的名称:
a b c d

请选择操作：B
请输入添加边的个数:6
请输入两个顶点及边:d a 11
请输入两个顶点及边:b d 12
请输入两个顶点及边:c d 5
请输入两个顶点及边:a b 8
请输入两个顶点及边:a c 18
请输入两个顶点及边:b c 7

请选择操作：C
请输入起始顶点:a
Prim算法成功!

请选择操作：D
最小生成树的顶点及边为:

a-<8>-b    b-<7>-c    c-<5>-d
请选择操作：E
成功退出系统!
```

4 测试

4.1 功能测试

4.1.1 创建电网顶点功能测试

测试用例：
A
4
a b c d
预期结果：完成创建，程序正常运行不崩溃。
实验结果：



4.1.2 添加电网的边功能测试

测试用例：

A
4
a b c d
B
6
d a 11
b d 12
c d 5
a b 8
a c 18
b c 7

预期结果：完成添加，程序正常运行不崩溃。

实验结果：

```
**                电网造价模拟系统                **
================================================
**                请选择要执行的操作：            **
**                A---创建电网顶点                **
**                B---添加电网的边                **
**                C---构造最小生成树              **
**                D---显示最小生成树              **
**                E---退出程序                    **
================================================

请选择操作：A
请输入顶点的个数:4
请依次输入各顶点的名称:
a b c d

请选择操作：B
请输入添加边的个数:6
请输入两个顶点及边:d a 11
请输入两个顶点及边:b d 12
请输入两个顶点及边:c d 5
请输入两个顶点及边:a b 8
请输入两个顶点及边:a c 18
请输入两个顶点及边:b c 7

请选择操作：
```

4.1.3 构造最小生成树功能测试

测试用例：

A
4
a b c d
B
6
d a 11
b d 12
c d 5
a b 8
a c 18
b c 7
C
a

预期结果：完成构造，程序正常运行不崩溃。

实验结果：

```
**                电网造价模拟系统                **
==================================================
**                请选择要执行的操作：            **
**                A---创建电网顶点                **
**                B---添加电网的边                **
**                C---构造最小生成树              **
**                D---显示最小生成树              **
**                E---退出程序                    **
==================================================

请选择操作：A
请输入顶点的个数:4
请依次输入各顶点的名称:
a b c d

请选择操作：B
请输入添加边的个数:6
请输入两个顶点及边:d a 11
请输入两个顶点及边:b d 12
请输入两个顶点及边:c d 5
请输入两个顶点及边:a b 8
请输入两个顶点及边:a c 18
请输入两个顶点及边:b c 7

请选择操作：C
请输入起始顶点:a
Prim算法成功!

请选择操作：
```

4.1.4 显示最小生成树功能测试

测试用例：

A
4
a b c d
B
6
d a 11
b d 12
c d 5
a b 8
a c 18
b c 7
C
a
D

预期结果：显示成功，程序正常运行不崩溃。输出 a-<8>-b　　b-<7>-c　　c-<5>-d

实验结果：

```
**                  电网造价模拟系统                    **
=====================================================
**                  请选择要执行的操作：                **
**                  A---创建电网顶点                    **
**                  B---添加电网的边                    **
**                  C---构造最小生成树                  **
**                  D---显示最小生成树                  **
**                  E---退出程序                        **
=====================================================

请选择操作：A
请输入顶点的个数:4
请依次输入各顶点的名称:
a b c d

请选择操作：B
请输入添加边的个数:6
请输入两个顶点及边:d a 11
请输入两个顶点及边:b d 12
请输入两个顶点及边:c d 5
请输入两个顶点及边:a b 8
请输入两个顶点及边:a c 18
请输入两个顶点及边:b c 7

请选择操作：C
请输入起始顶点:a
Prim算法成功!

请选择操作：D
最小生成树的顶点及边为:

a-<8>-b    b-<7>-c    c-<5>-d
请选择操作：
```

4.1.5 整体功能测试

测试用例：

```
A
10
a b c d e f g h i j
B
25
d i 7
e f 7
j c 15
g a 12
c f 10
b d 10
h c 12
i j 3
g b 7
f a 5
d e 16
f g 13
c e 4
h b 6
e a 9
a b 13
i b 8
f j 5
d j 10
h g 3
c d 6
i f 14
b f 11
j e 6
e g 4
C
a
D
```

预期结果：

显示成功，程序正常运行不崩溃。输出：

```
a-<5>-f     f-<5>-j     j-<3>-i     j-<6>-e
e-<4>-c     e-<4>-g     g-<3>-h     c-<6>-d
h-<6>-b
```

实验结果：

```
**                电网造价模拟系统                **
================================================
**                请选择要执行的操作：            **
**                A---创建电网顶点                **
**                B---添加电网的边                **
**                C---构造最小生成树              **
**                D---显示最小生成树              **
**                E---退出程序                    **
================================================

请选择操作：A
请输入顶点的个数:10
请依次输入各顶点的名称:
a b c d e f g h i j

请选择操作：B
请输入添加边的个数:25
请输入两个顶点及边:d i 7
请输入两个顶点及边:e f 7
请输入两个顶点及边:j c 15
请输入两个顶点及边:g a 12
请输入两个顶点及边:c f 10
请输入两个顶点及边:b d 10
请输入两个顶点及边:h c 12
请输入两个顶点及边:i j 3
请输入两个顶点及边:g b 7
请输入两个顶点及边:f a 5
请输入两个顶点及边:d e 16
请输入两个顶点及边:f g 13
请输入两个顶点及边:c e 4
请输入两个顶点及边:h b 6
请输入两个顶点及边:e a 9
请输入两个顶点及边:a b 13
请输入两个顶点及边:i b 8
请输入两个顶点及边:f j 5
请输入两个顶点及边:d j 10
请输入两个顶点及边:h g 3
请输入两个顶点及边:c d 6
请输入两个顶点及边:i f 14
请输入两个顶点及边:b f 11
请输入两个顶点及边:j e 6
请输入两个顶点及边:e g 4

请选择操作：C
请输入起始顶点:a
Prim算法成功!

请选择操作：D
最小生成树的顶点及边为:

a-<5>-f    f-<5>-j    j-<3>-i    j-<6>-e
e-<4>-c    e-<4>-g    g-<3>-h    c-<6>-d
h-<6>-b
请选择操作：
```

4.2 边界测试

4.2.1 创建电网顶点数为零

测试用例：
A
0
预期结果：程序给出提示信息，程序运行正常不崩溃。
实验结果：

```
**                  电网造价模拟系统                  **
==================================================
**                  请选择要执行的操作：              **
**                  A---创建电网顶点                  **
**                  B---添加电网的边                  **
**                  C---构造最小生成树                **
**                  D---显示最小生成树                **
**                  E---退出程序                      **
==================================================

请选择操作：A
请输入顶点的个数:0
顶点的个数只能是正整数!
请重新输入顶点个数:
```

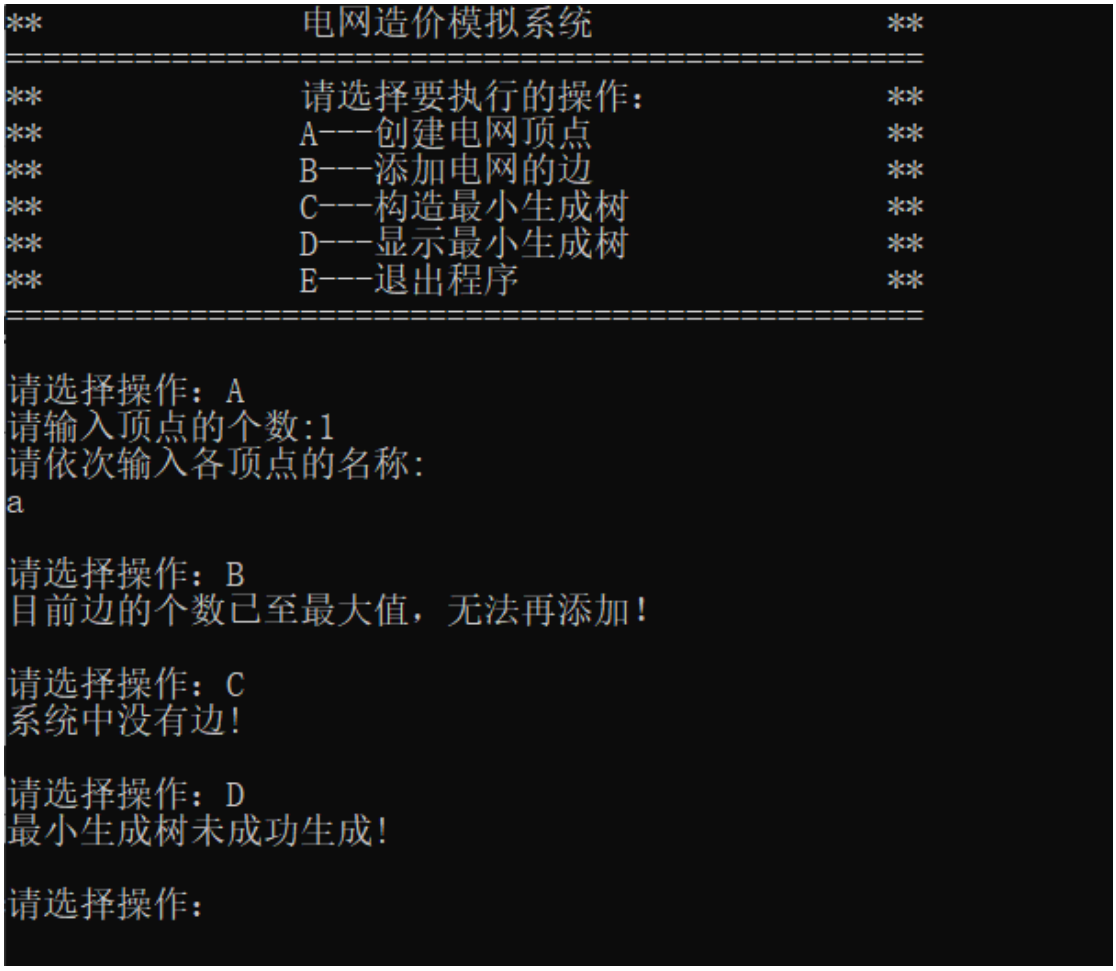
4.2.2 当前电网顶点数为一时进行 B,C,D 操作

测试用例：

A
1
a
B
C
D

预期结果：程序给出提示信息，程序运行正常不崩溃。

实验结果：



4.2.3 当前电网顶点数为零时进行 B,C,D 操作

测试用例：

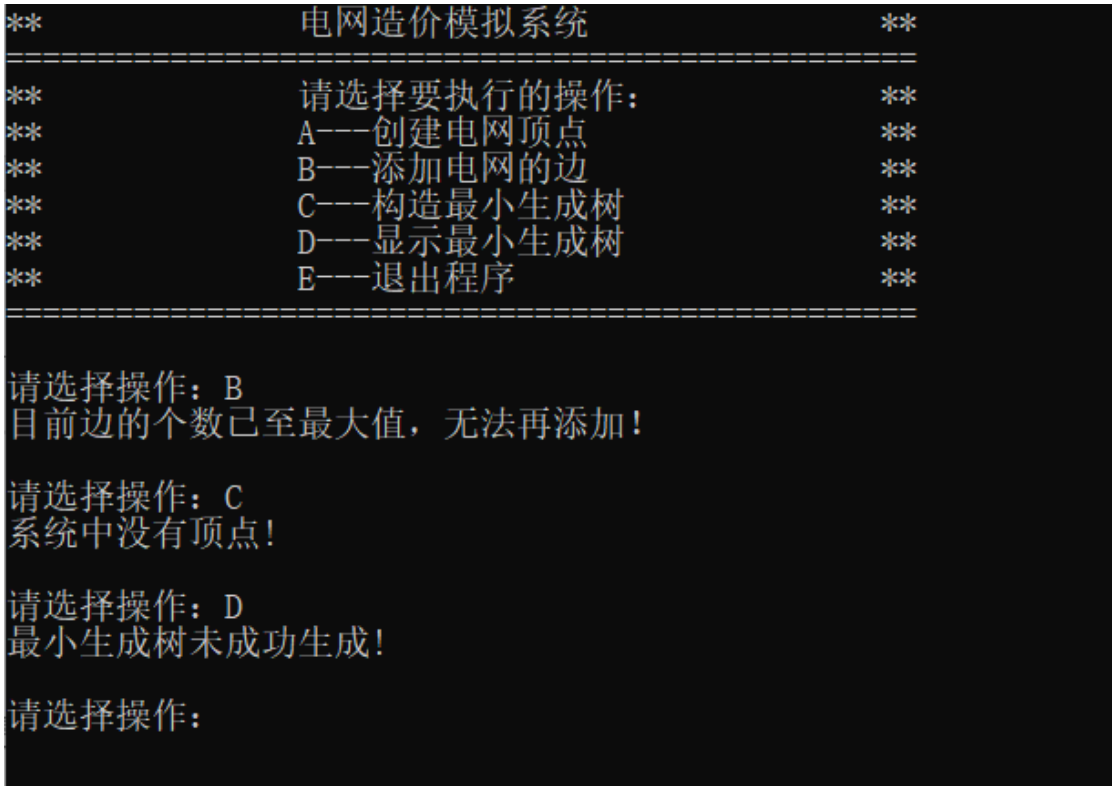
B

C

D

预期结果：程序给出提示信息，程序运行正常不崩溃。

实验结果：



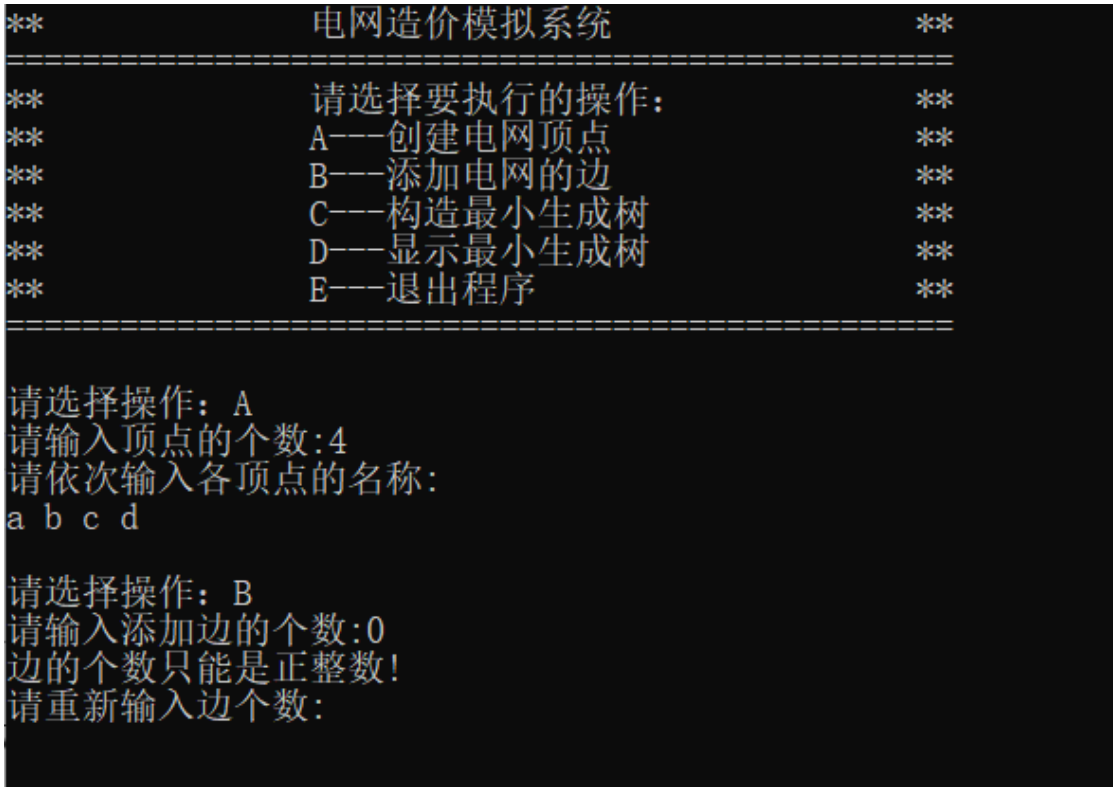
4.2.4 添加电网的边的个数为零

测试用例：

A
4
a b c d
B
0

预期结果：程序给出提示信息，程序运行正常不崩溃。

实验结果：



4.2.5 添加边后边的数量超过最大值

测试用例：

A
4
a b c d
B
4
a b 5
b c 8
c d 7
d a 4
B
4

预期结果：

4 个顶点最多只能有 6 条边
程序给出提示信息，程序运行正常不崩溃。

实验结果：

```
**                  电网造价模拟系统                    **
==============================================================
**                  请选择要执行的操作：                **
**                  A---创建电网顶点                    **
**                  B---添加电网的边                    **
**                  C---构造最小生成树                  **
**                  D---显示最小生成树                  **
**                  E---退出程序                        **
==============================================================

请选择操作：A
请输入顶点的个数:4
请依次输入各顶点的名称:
a b c d

请选择操作：B
请输入添加边的个数:4
请输入两个顶点及边:a b 5
请输入两个顶点及边:b c 8
请输入两个顶点及边:c d 7
请输入两个顶点及边:d a 4

请选择操作：B
请输入添加边的个数:4
输入后边的数量会超过最大值!
请重新输入边个数:
```

4.2.6 存在边的权值为零

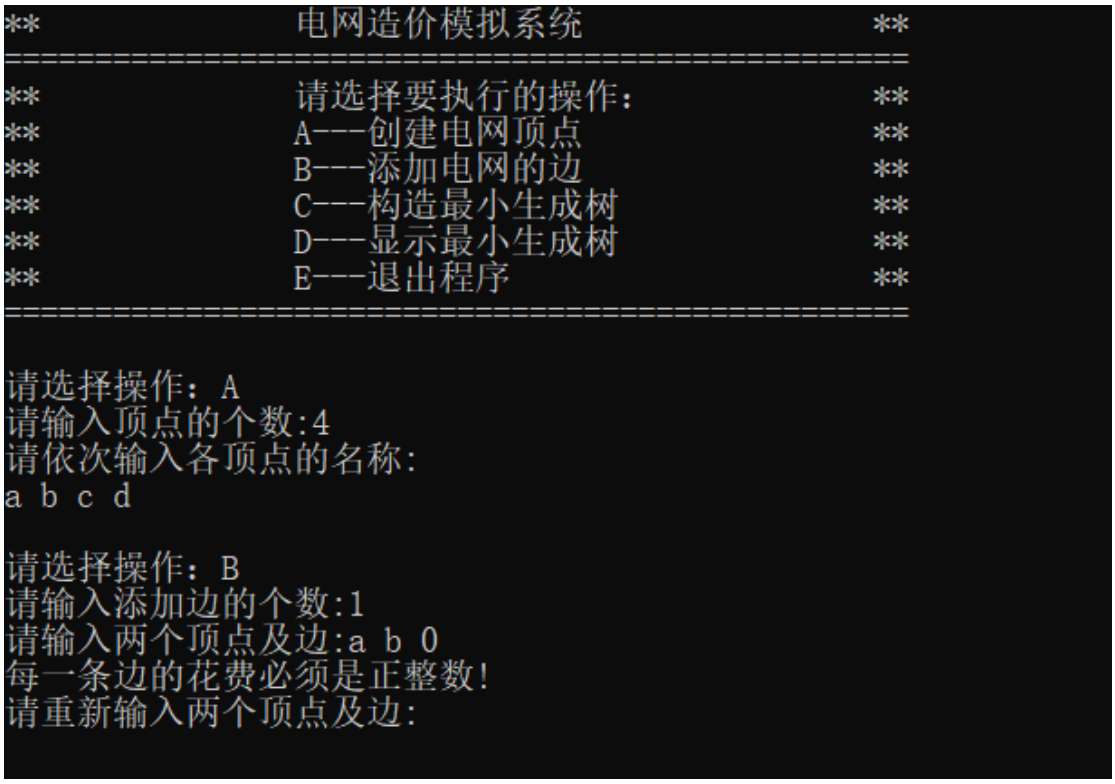
测试用例：

A
4
a b c d
B
1
a b 0

预期结果：

电网每一条线路的构造花费应该都为正数
程序给出提示信息，程序运行正常不崩溃。

实验结果：



4.3 出错测试

4.3.1 输入添加顶点个数错误

测试用例：
A
-5
wasd
测试
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：

```
**                  电网造价模拟系统                  **
==================================================
**                  请选择要执行的操作：              **
**                  A---创建电网顶点                  **
**                  B---添加电网的边                  **
**                  C---构造最小生成树                **
**                  D---显示最小生成树                **
**                  E---退出程序                      **
==================================================

请选择操作：A
请输入顶点的个数:-5
顶点的个数只能是正整数!
请重新输入顶点个数:wasd
顶点的个数只能是正整数!
请重新输入顶点个数:测试
顶点的个数只能是正整数!
请重新输入顶点个数:
```

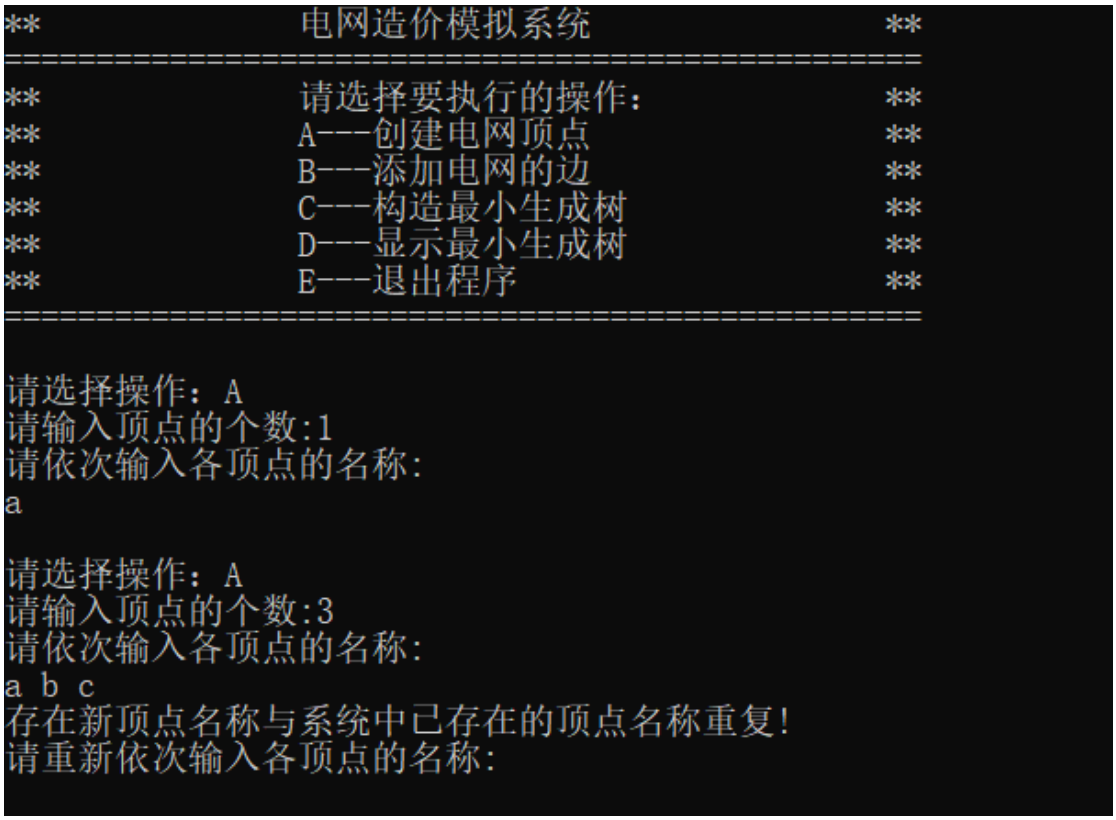
4.3.2 新添加的顶点名称在电网中已存在

测试用例：

A
1
a
A
3
a b c

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 新添加的顶点名称存在重复

测试用例：

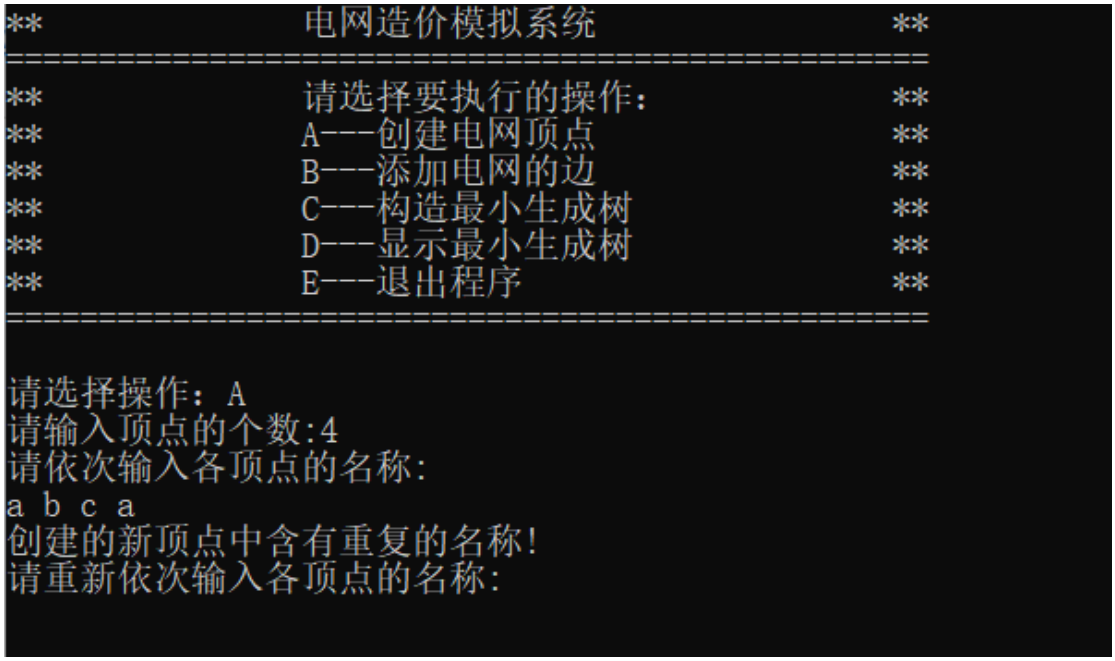
A

4

a b c a

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：



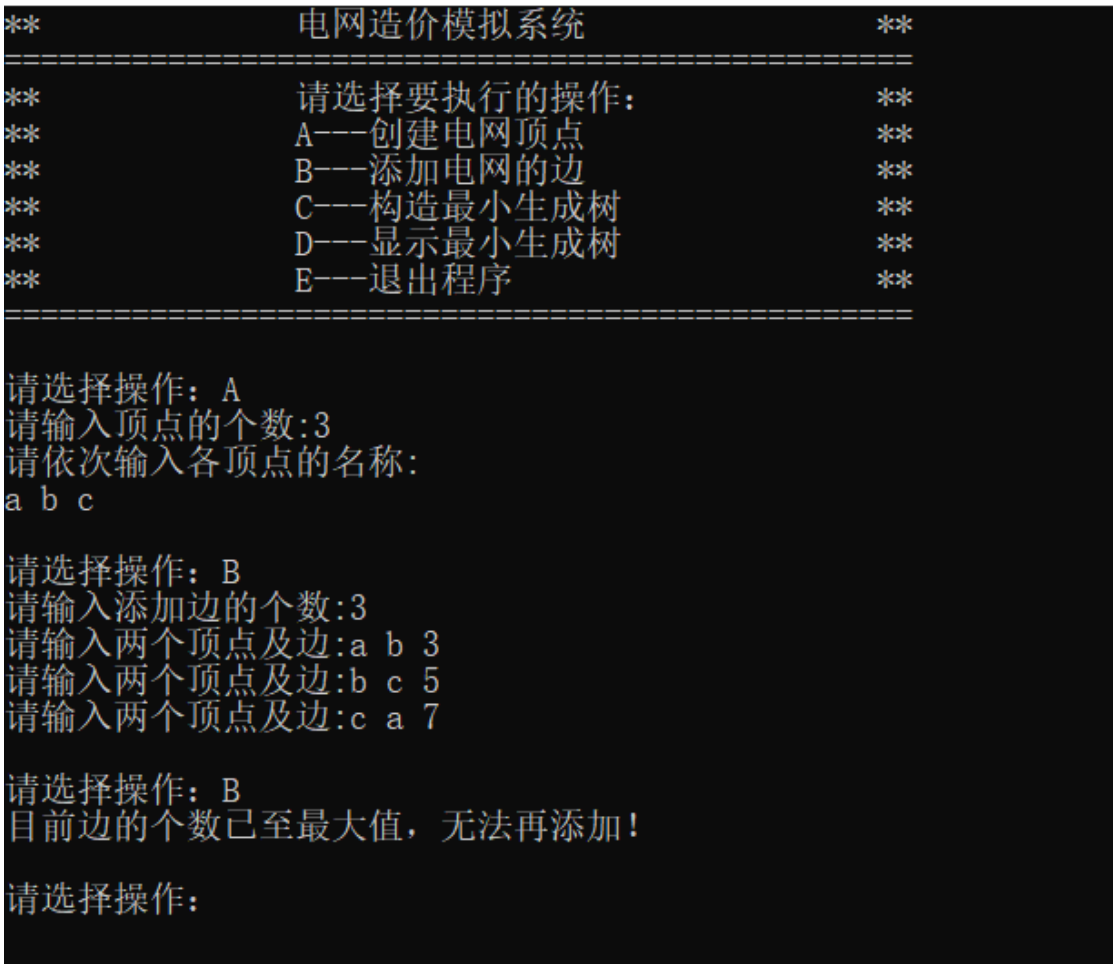
4.3.4 边的个数已经到最大值后进行 B 操作

测试用例：

A
3
a b c
B
3
a b 3
b c 5
c a 7
B

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：



4.3.5 输入添加边个数错误

测试用例：

A
3
a b c
B
-5
wasd
测试

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：

```
**                电网造价模拟系统                **
==================================================
**                请选择要执行的操作：            **
**                A---创建电网顶点                **
**                B---添加电网的边                **
**                C---构造最小生成树              **
**                D---显示最小生成树              **
**                E---退出程序                    **
==================================================

请选择操作：A
请输入顶点的个数:3
请依次输入各顶点的名称:
a b c

请选择操作：B
请输入添加边的个数:-5
边的个数只能是正整数!
请重新输入边个数:wasd
边的个数只能是正整数!
请重新输入边个数:测试
边的个数只能是正整数!
请重新输入边个数:
```

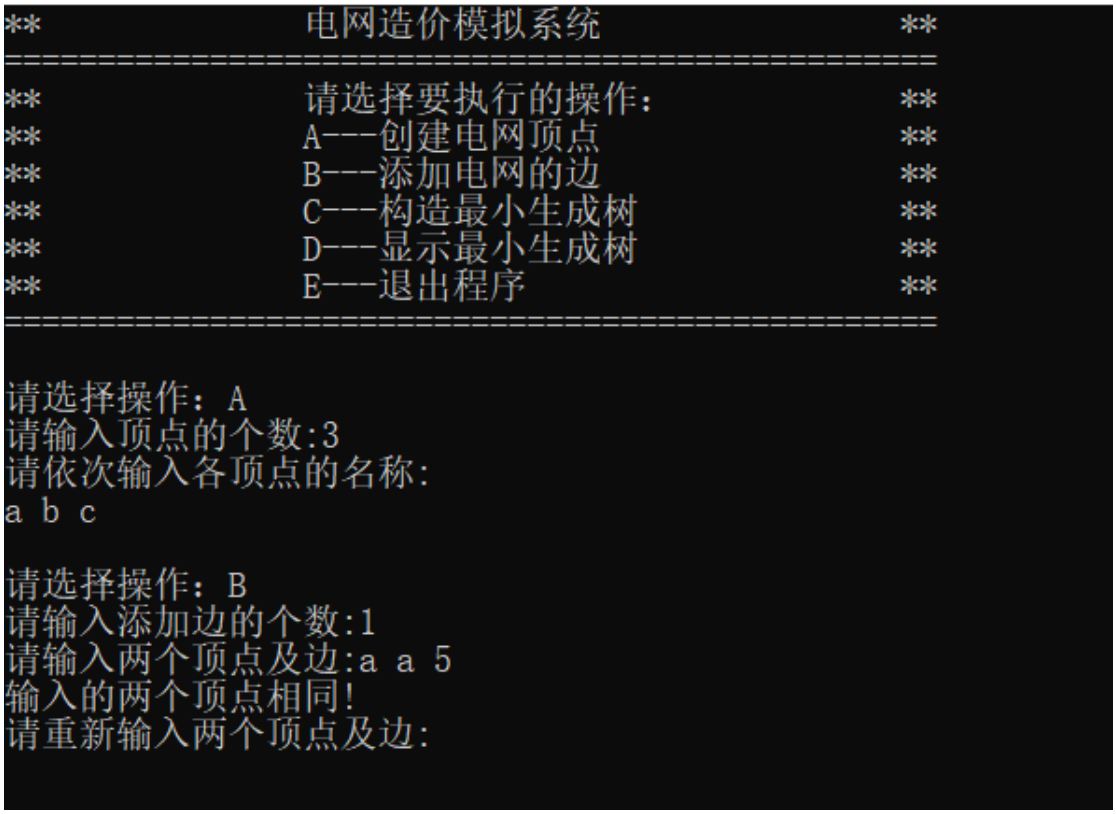
4.3.6 添加边时输入的两个顶点相同

测试用例：

A
3
a b c
B
1
a a 5

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：



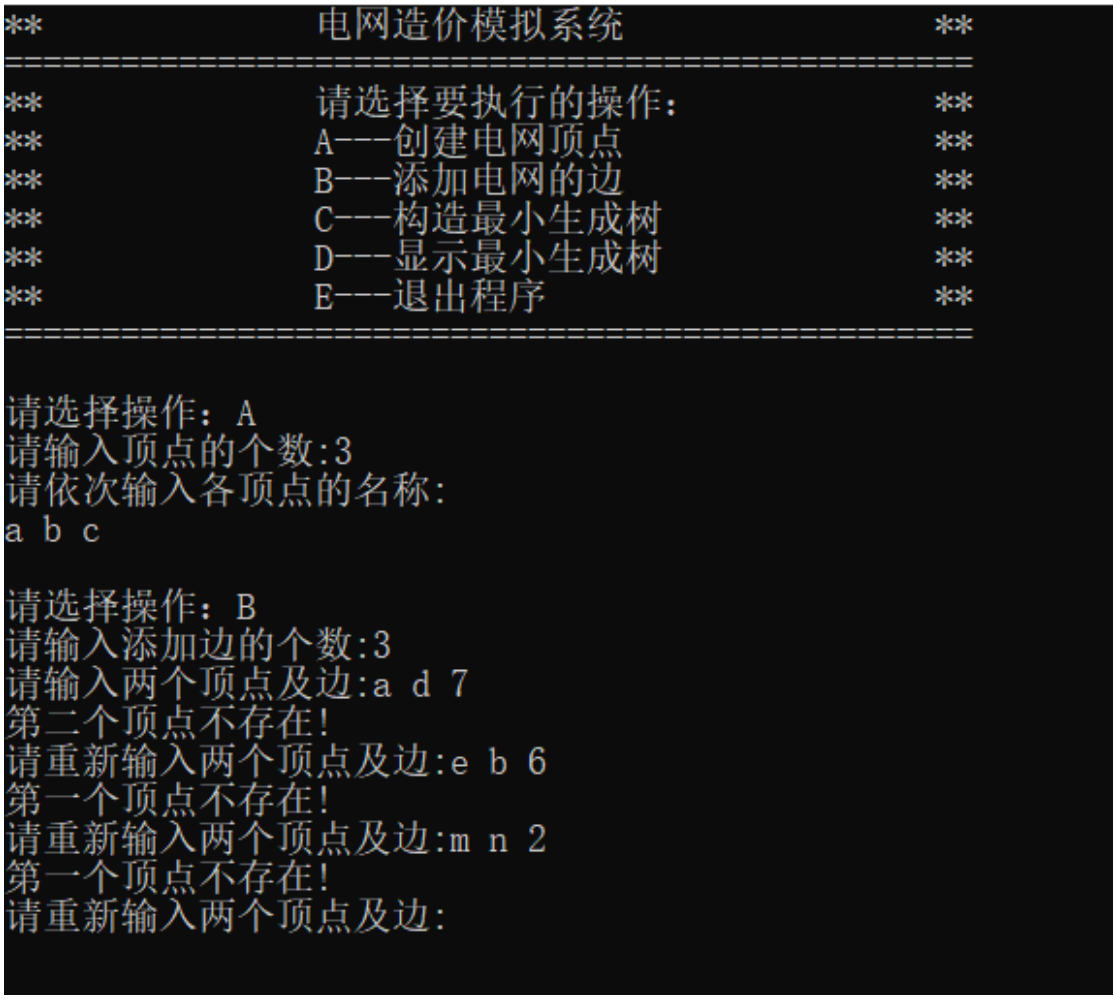
4.3.7 添加边时输入的两个顶点不全部在电网中

测试用例：

A
3
a b c
B
3
a d 7
e b 6
m n 2

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：



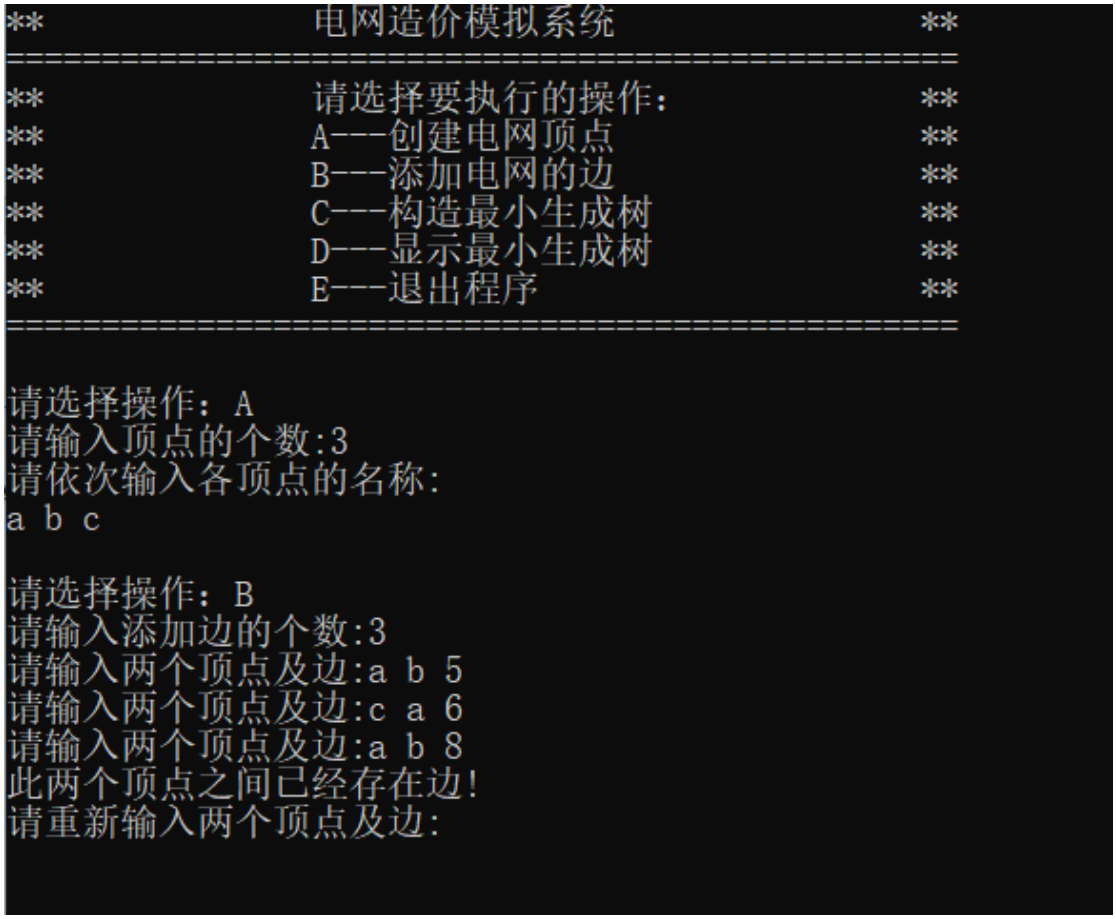
4.3.8 添加边时输入的两个顶点之间已经存在边

测试用例：

A
3
a b c
B
3
a b 5
c a 6
a b 8

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：



4.3.9 构造最小生成树时起始顶点不在电网中

测试用例：

A
3
a b c
B
3
a b 5
c a 6
b c 8
C
d

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：

```
**                  电网造价模拟系统                  **
======================================================
**                  请选择要执行的操作:               **
**                  A---创建电网顶点                  **
**                  B---添加电网的边                  **
**                  C---构造最小生成树                **
**                  D---显示最小生成树                **
**                  E---退出程序                      **
======================================================

请选择操作: A
请输入顶点的个数:3
请依次输入各顶点的名称:
a b c

请选择操作: B
请输入添加边的个数:3
请输入两个顶点及边:a b 5
请输入两个顶点及边:c a 6
请输入两个顶点及边:b c 8

请选择操作: C
请输入起始顶点:d
此顶点不在系统之中!
请重新输入:
```

4.3.10 最小生成树未构造时选择显示最小生成树

测试用例：

A
3
a b c
B
3
a b 5
c a 6
b c 8
D

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：

```
**                电网造价模拟系统                **
==================================================
**                请选择要执行的操作：            **
**                A---创建电网顶点                **
**                B---添加电网的边                **
**                C---构造最小生成树              **
**                D---显示最小生成树              **
**                E---退出程序                    **
==================================================

请选择操作：A
请输入顶点的个数:3
请依次输入各顶点的名称:
a b c

请选择操作：B
请输入添加边的个数:3
请输入两个顶点及边:a b 5
请输入两个顶点及边:c a 6
请输入两个顶点及边:b c 8

请选择操作：D
最小生成树未成功生成!

请选择操作：
```

4.3.11 电网未联通时选择构造最小生成树

测试用例：

A
3
a b c
B
1
a b 5
C
a
D

预期结果：忽略多余的消息，程序正常运行不崩溃。

实验结果：

```
**                  电网造价模拟系统                  **
==================================================
**                  请选择要执行的操作:               **
**                  A---创建电网顶点                  **
**                  B---添加电网的边                  **
**                  C---构造最小生成树                **
**                  D---显示最小生成树                **
**                  E---退出程序                      **
==================================================

请选择操作：A
请输入顶点的个数:3
请依次输入各顶点的名称:
a b c

请选择操作：B
请输入添加边的个数:1
请输入两个顶点及边:a b 5

请选择操作：C
请输入起始顶点:a
目前的电力系统并未连通!

请选择操作：D
最小生成树未成功生成!

请选择操作：
```

4.3.12 输入操作数不合法

测试用例：
F
wasd
1
测试
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：

```
**                  电网造价模拟系统                  **
==============================================================
**                  请选择要执行的操作：              **
**                  A---创建电网顶点                  **
**                  B---添加电网的边                  **
**                  C---构造最小生成树                **
**                  D---显示最小生成树                **
**                  E---退出程序                      **
==============================================================

请选择操作：F
操作数输入不正确，请重新输入!
请选择操作：wasd
操作数输入不正确，请重新输入!
请选择操作：1
操作数输入不正确，请重新输入!
请选择操作：测试
操作数输入不正确，请重新输入!
请选择操作：
```