同濟大學
TONGJI UNIVERSITY

项目说明文档

电网造价模拟系统

指导教师：张颖

1751984 王舸飞

1.分析

1.1 背景分析

假设一个城市有n个小区，要实现n个小区的电网都能够互相接通，构造这个城市n个小区之间的电网，使总工程造价最低。要设计一个能够满足条件的造价方案。

1.2 功能分析

本次要求设计一个造价系统的模拟，除了给出最短路的算法，还应包含其他相关输入输出操作，即建立一个数据结构对输入数据进行存储，在输入的过程中判断合法性。对于最重要的最短路算法采用规定的prim算法，并在生成最短路之后将它输出。

2.设计

2.1 数据结构设计

n个小区之间通过电网连通，很容易让人想到图的结构，由于电网的特性和本题中实现造价的要求，将电网定义为带权无向图。

此外，还应定义一个数组结构用于存储生成的最短路。

由于prim算法中要用到最小堆，还应设一个最小堆的数据结构。

2.2 类和结构体设计

承接上文，本次对小区的定义为带权无向图，对它设计了一个类，由于要涉及大量插入删除操作，采用邻接表进行存储。

定义了三个结构体，分别为图的节点，邻接表中的边和最小生成树中的边。

此外，还对于最小堆和最小生成树定义了两个类。

2.3 成员与操作设计

邻接表边的结构体(Edge)

```
struct Edge{
    int another;//这条边的另一个节点名称
    int cost;//这条边的权值
    Edge *link;//下一条边
    Edge(int num,int weight):another(num),cost(weight),link(NULL){};//构造函数
};//边的初始化定义
```

最小生成树边的结构体(MST_EDGE)

```
struct MST_EDGE{
    string head;//在最小生成树中的头端点名称
    string tail;//在最小生成树中的尾端点名称
    int cost;//权值
};
```

节点的结构体(Vertex)

```
struct Vertex{
    string name;//顶点的名称

    Edge *first_edge;//顶点的第一条边

    Vertex(){first_edge=NULL;}//构造函数

};//结点的初始化定义
```

邻接表类(Link_Graph)

```
class Link_Graph{
public: Link_Graph(){NodeTable=(Vertex *)malloc(10000*sizeof(Vertex));length=0;}//首先对于邻接表分配足够的内存
    int GetVertexPos(string p_name);//给出节点名称返回它在邻接表中的位置
    string GetName(int p_number);//给出位置返回它在邻接表中的名称
    void InitVertex();//初始化节点，即生成邻接表
    void InsertEdge(int v1,int v2,int cost);//给出两条边的位置和权值将其插入邻接表
    void InitEdge();//初始化邻接表的边
    int Getlength(){return length;}//返回邻接表的长度
    int GetFirstNeighbor(int v);//获得一个节点的第一个边结点位置
    int GetNextNeighbor(int v,int w);//在v节点的边中，获得位置为w节点右边的节点位置
    int GetWeight(int v1,int v2);//给出两节点位置，返回它们的边权重
    void Clear(){for(int i=0;i<length;i++)NodeTable[i].first_edge=NULL;} //将表的边刷新
private:Vertex *NodeTable;//邻接表数组的指针
    int length;//代表邻接表的长度
};
```

最小堆类(Minheap)

```
//最小堆的定义
class MinHeap{

public:MinHeap(){length=0;}//初始化长度为0

    void Renew(){length=0;}//更新长度为0

    void Insert(MST_EDGE);//插入一个最小生成树节点

    bool IsEmpty(){return length==0?true:false;}//判空

    void Remove(MST_EDGE &E1);//取堆顶元素存于E1中

private:int length;//长度

    MST_EDGE heap[1000];//存储用数组
};
```

最小生成树类(MST)

```
class MST{
public:

    void SetLength(int p_length){length=p_length;}//设置最小生成树的长度

    void Display(){for(int i=0;i<length;i++)cout<<E[i].head<<"-<"<<E[i].cost<<">-"<<E[i].tail<<"    ";}

    //输出展示最小生成树


    void Insert(MST_EDGE p_MST_EDGE){

        E[length]=p_MST_EDGE;

        length++;
    }

    //插入最小生成树

private:int length;//最小生成树的长度

    MST_EDGE E[1000];//存储边的数组

};
```

其他用到的函数：

初始化函数(init())

```
void init(){
    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;
    cout<<"====================================="<<endl;
    cout<<endl;
}
```

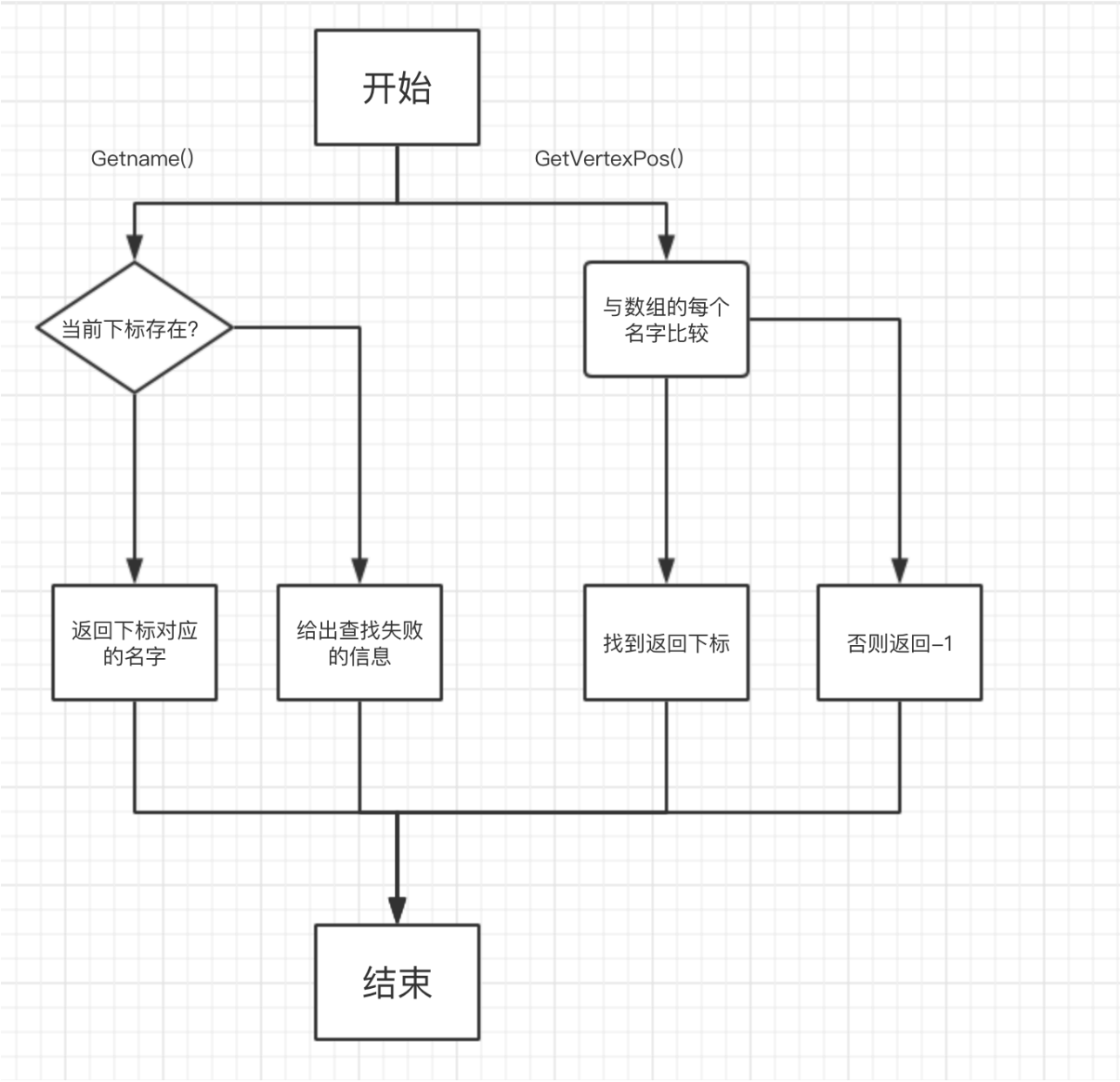
2.4 系统设计

程序运行之后，系统会给出提示信息并创建一个邻接表类和一个最小生成树类，利用构造函数将它们初始化。之后会要求用户按照提示信息给出操作码的输入。当输入的操作码不为结束提示符'E'时给出相应的操作，否则退出程序。

3.实现

3.1 邻接表返回功能的实现

3.1.1 返回功能流程图

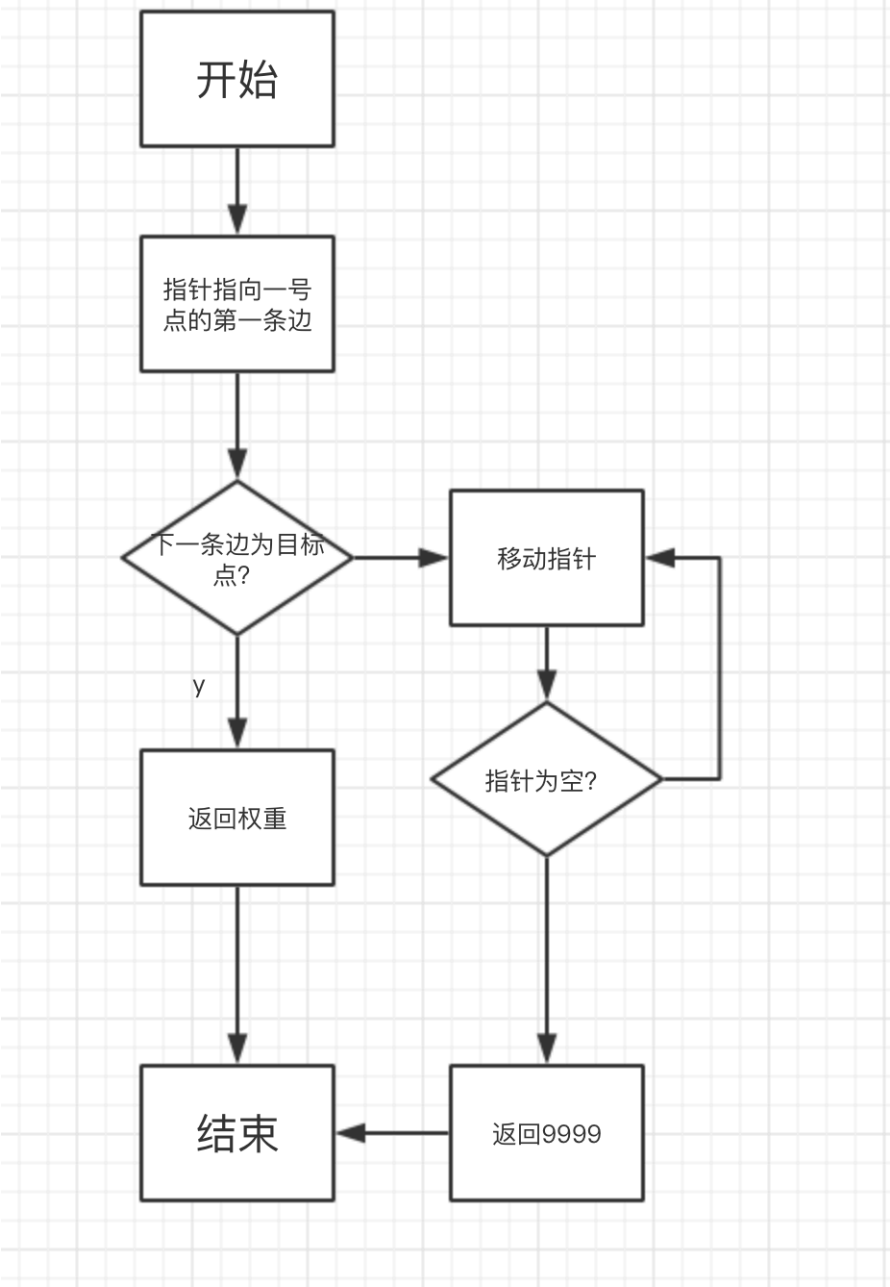


3.1.2 返回功能核心代码

```
int Link_Graph::GetVertexPos(string p_name){

    for(int i=0;i<length;i++)if(strcmp(NodeTable[i].name.c_str(),p_name.c_str())==0)return i;

    return -1;//如果没有找到返回-1
    }


string Link_Graph::GetName(int p_number){

    if(p_number>=0&&p_number<length)return NodeTable[p_number].name;

    else return "can not find";//如果没有找到返回提示信息
}
```

3.2 返回权重功能的实现

3.2.1 返回权重功能流程图

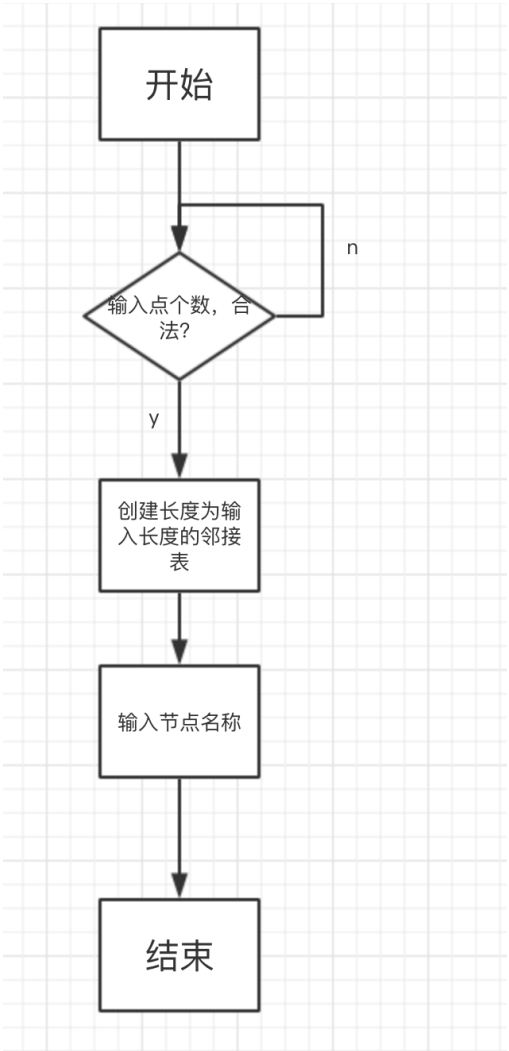


3.2.2 返回权重功能核心代码

```
int Link_Graph::GetWeight(int v1,int v2){
    if(v1!=-1&&v2!=-1){
        //首先判断两位置合法

        Edge *p=NodeTable[v1].first_edge;

        while(p!=NULL&&p->another!=v2)p=p->link;

        if(p!=NULL)return p->cost;
    }
    return 9999;//否则不存在边将两者相连，可认为无穷大
}
```

3.3 节点初始化功能的实现

3.3.1 节点初始化功能流程图

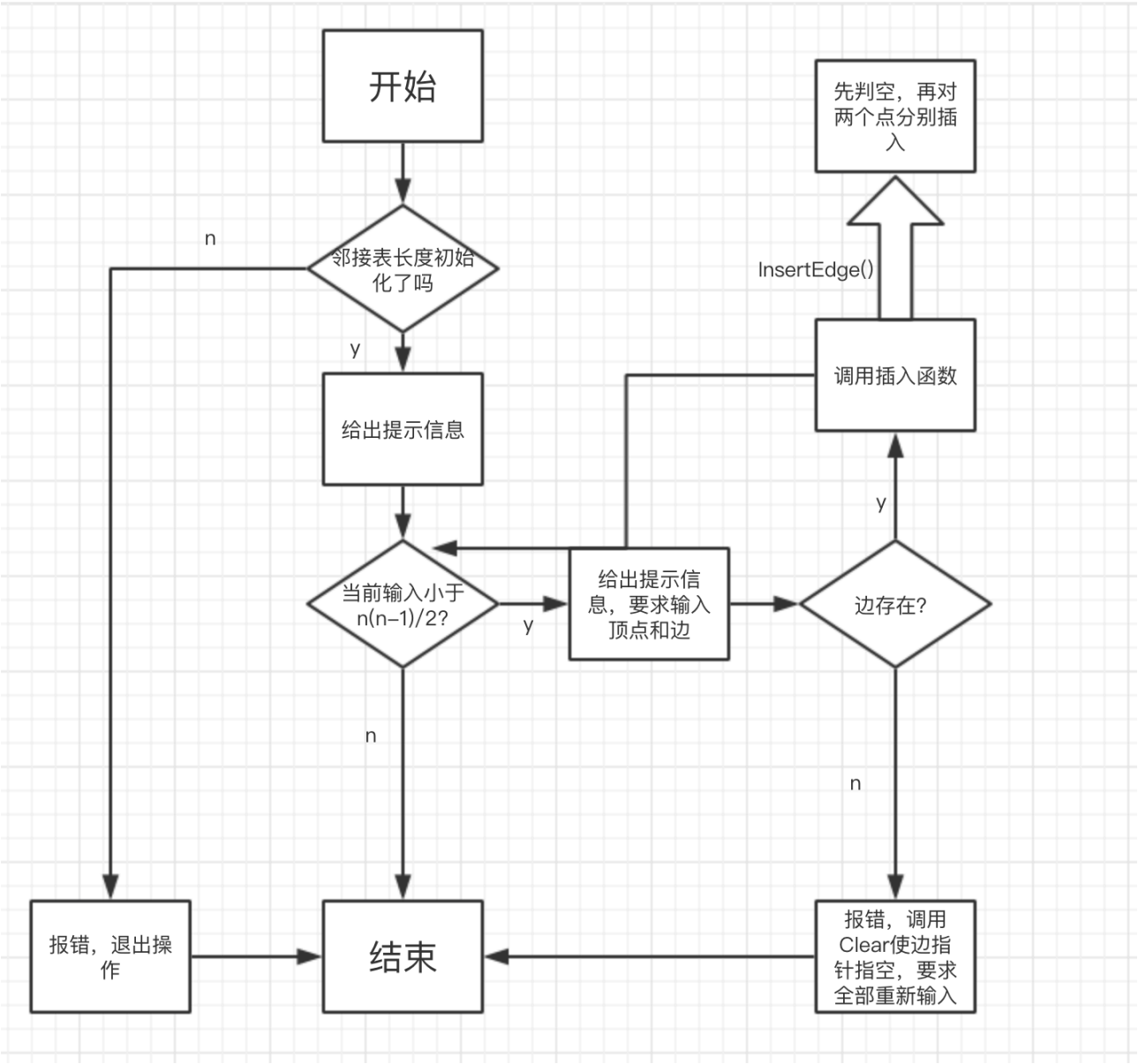


3.3.2 节点初始化功能核心代码

```
//初始化邻接表

void Link_Graph::InitVertex(){

    cout<<"请输入顶点的个数：";

    cin>>length;

    while(length<=0){
        cout<<"输入的顶点个数不合法，请重新输入：";

        cin>>length;
    }
    cout<<"请依次输入各顶点的名称："<<endl;

    for(int i=0;i<length;i++)cin>>NodeTable[i].name;
}
```

3.4 边初始化功能的实现

3.4.1 边初始化功能流程图



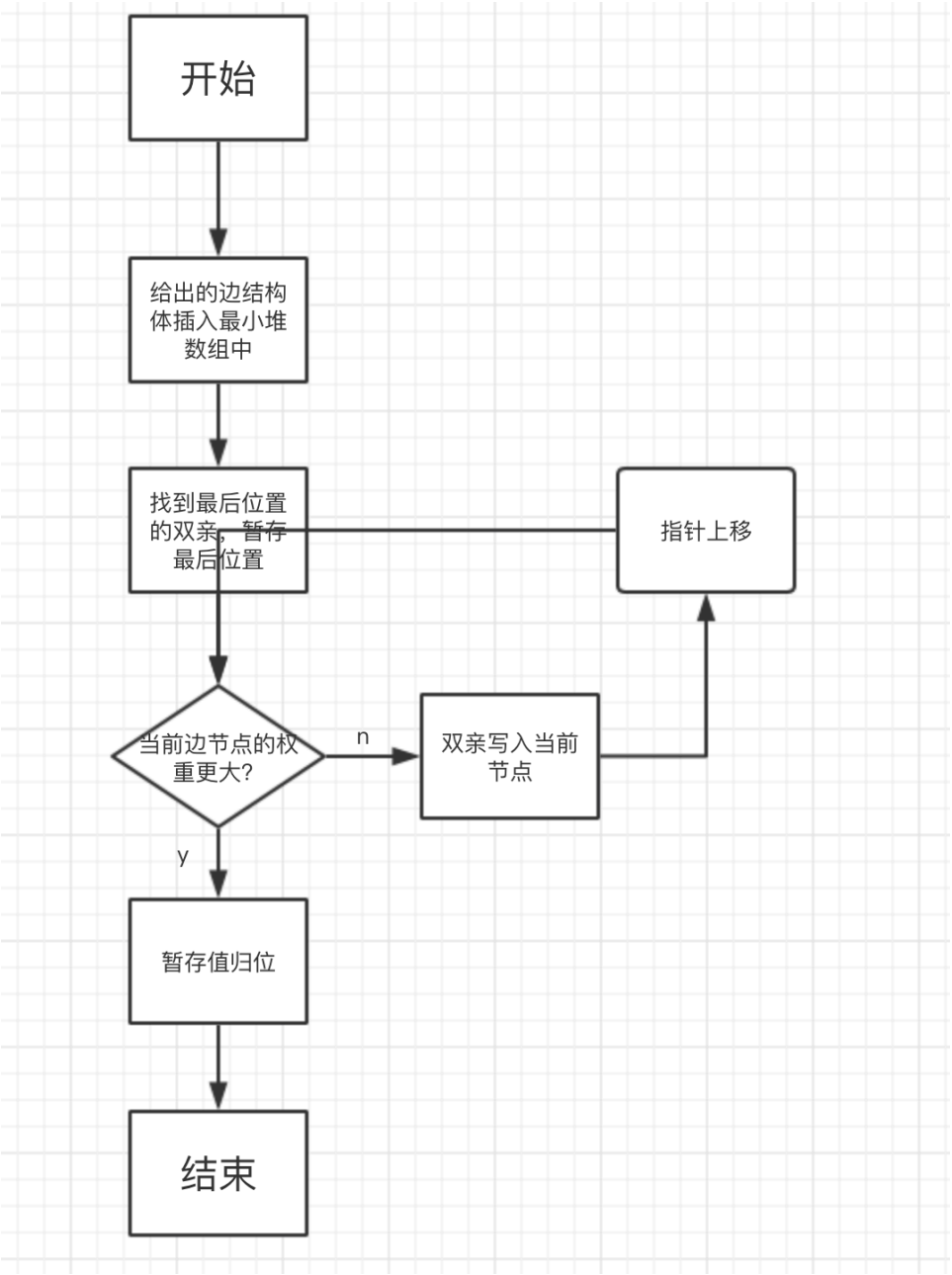
3.4.2 边初始化功能核心代码

```
void Link_Graph::InsertEdge(int v1,int v2,int cost){//初始化两个要插入的指针
    Edge *v1e=new Edge(v2,cost);
    Edge *v2e=new Edge(v1,cost);
    if(NodeTable[v1].first_edge==NULL){NodeTable[v1].first_edge=v1e;}//分类分别对两顶点进行头指针插入法
    else{v1e->link=NodeTable[v1].first_edge;NodeTable[v1].first_edge=v1e;}
    if(NodeTable[v2].first_edge==NULL){NodeTable[v2].first_edge=v2e;}
    else{v2e->link=NodeTable[v2].first_edge;NodeTable[v2].first_edge=v2e;}
}
```

```
    void Clear(){for(int i=0;i<length;i++)NodeTable[i].first_edge=NULL;} //将表的边刷新



//初始化边的函数


void Link_Graph::InitEdge(){//不可直接进行插入边操作

    if(length<=0){cout<<"在没有初始化节点的情况下不可以初始化边"<<endl;return;}

    else{
        cout<<"警告：在输入过程中如果出现相同顶点两次输入的情况会造成覆盖"<<endl;

        cout<<"请按完全图产生边数输入，对于不存在的边输入较大的权值即可"<<endl;

        int count=0;

        //共计n(n-1)/2条边
        while(count<length*(length-1)/2){

            string p1,p2;int p_cost;

            cout<<"请输入两个顶点及边:";

            cin>>p1>>p2;cin>>p_cost;
            int v1=GetVertexPos(p1);
            int v2=GetVertexPos(p2);

            if(v1==-1||v2==-1){cout<<"不存在的边,请全部重新输入"<<endl;Clear();break;}//刷新后要求重新输入

            else {InsertEdge(v1,v2,p_cost);count++;}
        }


    }

    }
```

3.5 最小堆插入功能的实现

3.5.1 最小堆插入功能流程图

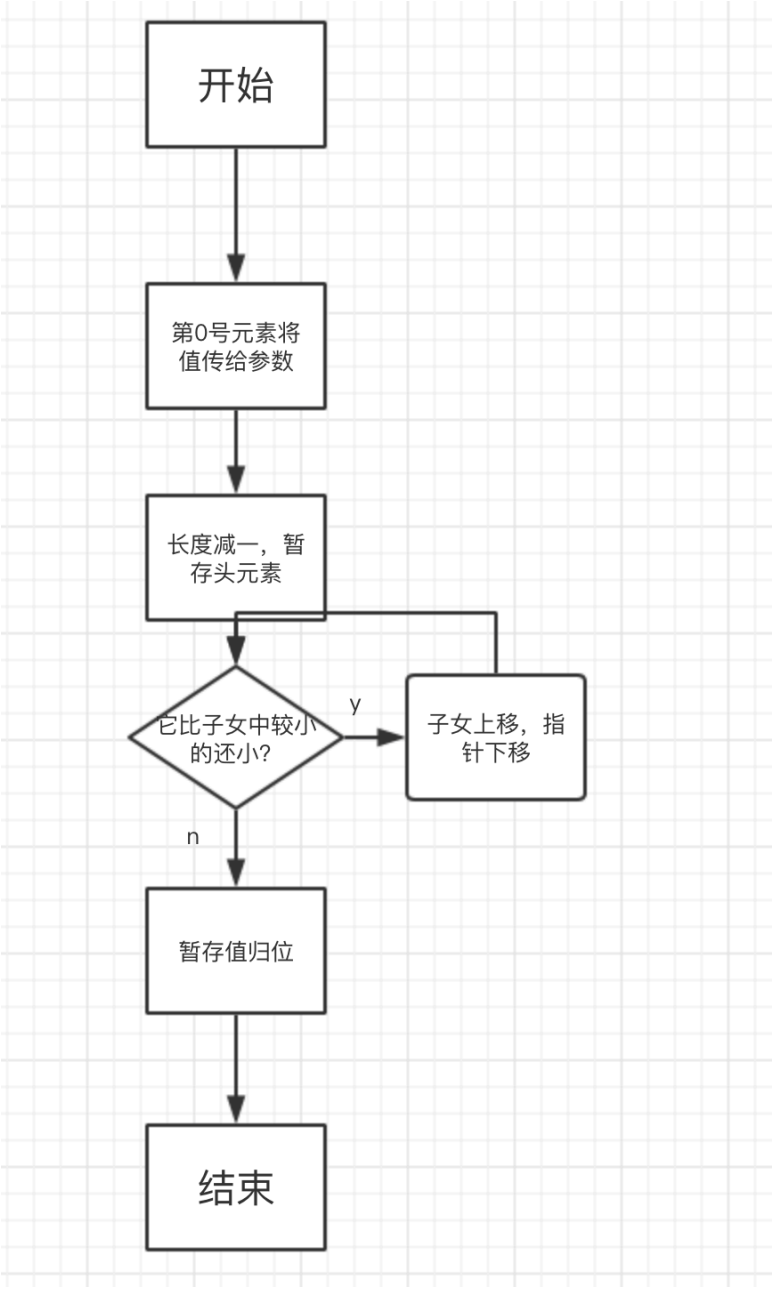


3.5.2 最小堆插入功能核心代码

```
void MinHeap::Insert(MST_EDGE E1){
    heap[length]=E1;
        int j=length,i=(j-1)/2;MST_EDGE temp=heap[j];//自底向上调整
    while(j>0){
        if(heap[i].cost<=temp.cost)break;
        else{heap[j]=heap[i];j=i;i=(i-1)/2;}
    }
    heap[j]=temp;
    length++;
    }
```

3.6 最小堆删除功能的实现

3.6.1 最小堆删除功能流程图

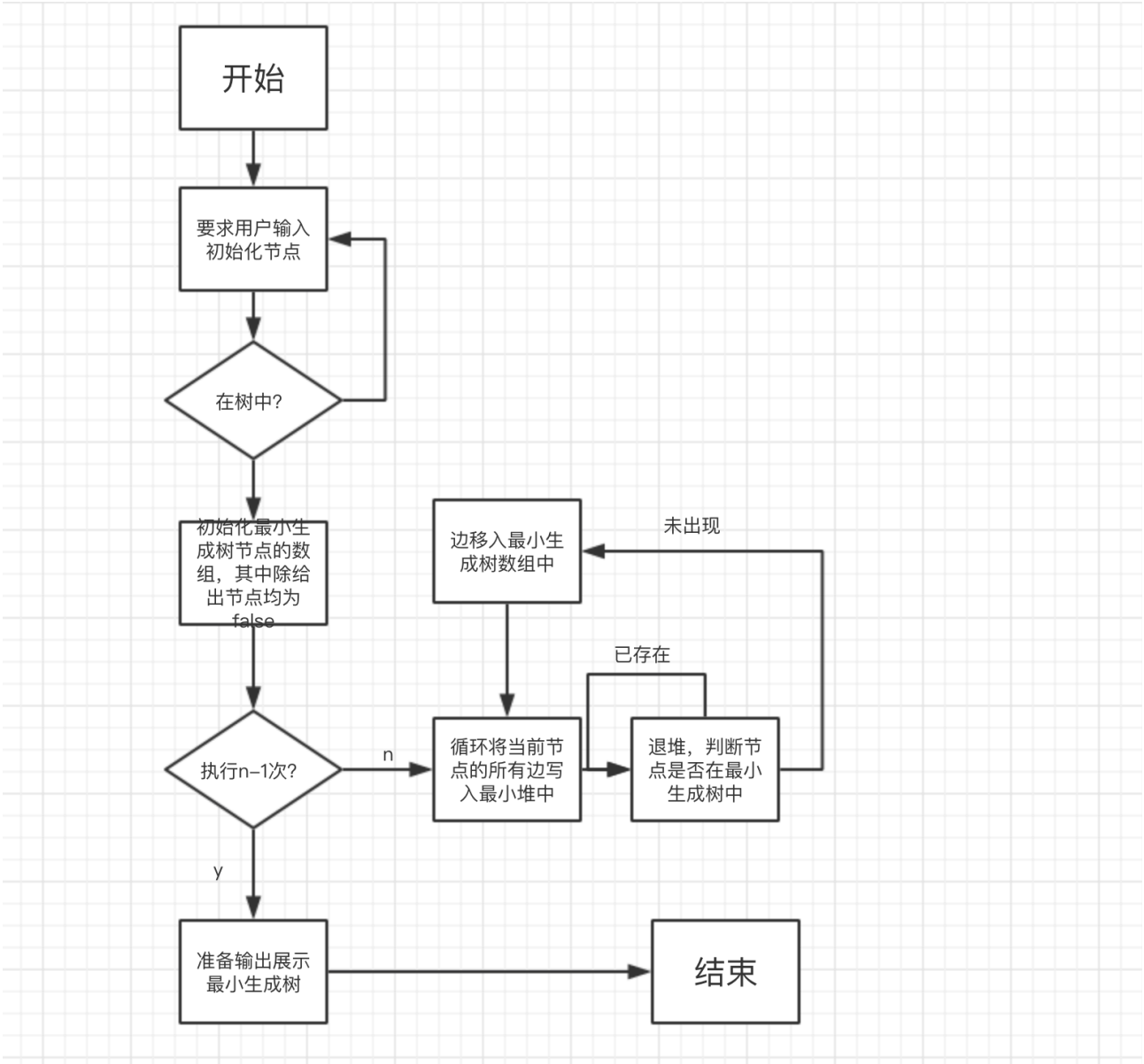


3.6.2 最小堆删除功能核心代码

```
void MinHeap::Remove(MST_EDGE &E1){
    E1=heap[0];

    heap[0]=heap[length-1];

    length--;

    int i=0,j=2*i+1;//自顶向下调整

    MST_EDGE temp=heap[i];

    while(j<=length-1){

        if(j<length-1&&heap[j].cost>heap[j+1].cost)j++;

        if(temp.cost<=heap[j].cost)break;

        else{heap[i]=heap[j];i=j;j=2*j+1;}
    }

    heap[i]=temp;


}
```

3.7 prim算法的实现

3.7.1 prim算法流程图



3.7.2 prim算法核心及相关代码

```
void Display(){for(int i=0;i<length;i++)cout<<E[i].head<<"-<"<<E[i].cost<<">-"<<E[i].tail<<"    ";}
//输出展示最小生成树

void Insert(MST_EDGE p_MST_EDGE){
    E[length]=p_MST_EDGE;
    length++;
}
```

```
void Prim(Link_Graph G,MST &M){

    M.SetLength(0);//进行算法前长度为0

    bool Vmst[G.Getlength()];//判断节点是否在最小生成树中

    for(int i=0;i<G.Getlength();i++)Vmst[i]=false;

    string p_point;

    cout<<"请输入起始顶点：";cin>>p_point;
     int u=G.GetVertexPos(p_point);
    while(u==-1){cout<<"给出的顶点不存在，请重新输入：";cin>>p_point;u=G.GetVertexPos(p_point);}

    int v;//分别给出要用到的表示位置的整数，存储边的变量和最小堆
    MST_EDGE ed;
    MinHeap H;


    Vmst[u]=true;//从u开始故u在堆中

    int count=1;//共进行n-1次

    do{
        v=G.GetFirstNeighbor(u);
        while(v!=-1){
            if(Vmst[v]==false){//将边置入最小堆中
                ed.tail=G.GetName(u);ed.head=G.GetName(v);ed.cost=G.GetWeight(u,v);
                H.Insert(ed);
            }
            v=G.GetNextNeighbor(u,v);//循环写入所有没用过的边
        }

        while(!H.IsEmpty()&&count<G.Getlength()){

            H.Remove(ed);//退值给ed

            if(Vmst[G.GetVertexPos(ed.head)]==false){//之前加入最小生成树的点不能再出现

                M.Insert(ed);

                u=G.GetVertexPos(ed.head);Vmst[u]=true;//u后移

                count++;break;
            }
        }


    }while(count<G.Getlength());

    cout<<"生成Prim最小生成树！"<<endl;
```

4.测试

4.1 常规测试

4.1.1 题目要求测试

```
**            电网造价模拟系统                **
====================================
**          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
警告：在输入过程中如果出现相同顶点两次输入的情况会造成覆盖
请按完全图产生边数输入，对于不存在的边输入较大的权值即可
请输入两个顶点及边:a b 8
请输入两个顶点及边:b c 7
请输入两个顶点及边:c d 5
请输入两个顶点及边:d a 11
请输入两个顶点及边:a c 18
请输入两个顶点及边:b d 12

请选择操作：C
请输入起始顶点：a
生成Prim最小生成树!

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

4.1.2 课本要求测试

实验结果：

```
请选择操作：a
请输入顶点的个数：7
请依次输入各顶点的名称：
0 1 2 3 4 5 6

请选择操作：B
警告：在输入过程中如果出现相同顶点两次输入的情况会造成覆盖
请按完全图产生边数输入，对于不存在的边输入较大的权值即可
请输入两个顶点及边:0 1 28
请输入两个顶点及边:0 2 999
请输入两个顶点及边:0 3 999
请输入两个顶点及边:0 4 999
请输入两个顶点及边:0 5 10
请输入两个顶点及边:1 2 16
请输入两个顶点及边:1 3 999
请输入两个顶点及边:1 4 999
请输入两个顶点及边:1 5 999
请输入两个顶点及边:2 3 12
请输入两个顶点及边:2 4 999
请输入两个顶点及边:2 5 999
请输入两个顶点及边:3 4 22
请输入两个顶点及边:3 5 999
请输入两个顶点及边:3 6 18
请输入两个顶点及边:1 6 14
请输入两个顶点及边:2 6 999
请输入两个顶点及边:4 5 25
请输入两个顶点及边:4 6 24
请输入两个顶点及边:5 6 999
请输入两个顶点及边:0 6 999

请选择操作：C
请输入起始顶点：0
生成Prim最小生成树!

请选择操作：D
5-<10>-0    4-<25>-5    3-<22>-4    2-<12>-3    1-<16>-2    6-
<14>-1
```

4.2 错误测试

4.2.1 非法节点个数

实验结果：

请选择操作： A
请输入顶点的个数： 0
输入的顶点个数不合法，请重新输入：

4.2.2 无法添加的边

实验结果：

请选择操作： B
在没有初始化节点的情况下不可以初始化边

请选择操作：

4.2.3 错误的边节点输入

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
警告：在输入过程中如果出现相同顶点两次输入的情况会造成覆盖
请按完全图产生边数输入，对于不存在的边输入较大的权值即可
请输入两个顶点及边：e f 7
不存在的边，请全部重新输入

4.2.4 最小生成树为空

请选择操作： D

请选择操作：