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： 王加炜

学 号： 2150265

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

# 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。请设计一个能够满足要求的造价方案。

## 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。

# 设计

## 2.1数据结构设计

如上功能分析所示，需要一个数据结构来存储无向边构成的图；由于prim算法需要将各条边上的权值进行比较，所以采用最小堆的数据结构来进行权值之间的比较，每次可以选出具有最小权值的边；通过最小生成树的数据结构将经过prim算法得到的各边信息存储在一个结构体数组中。综上，本题共使用到了三种数据结构。

## 2.2类结构设计

相较邻接表，邻接矩阵更加直观易懂，该存储结构也可以方便地找到相邻顶点。因此我决定采用邻接矩阵来存储图。同时邻接矩阵的类中也存储了顶点信息以及边的信息。

最小堆中包括了向上调整算法，向下调整算法以及插入和删除的功能。需要注意的是，最小堆中存储的是边的结构体元素，而参与比较时比较的是边中的权值信息，所以对于边的结构体需要进行一个运算符的重载操作。

最小生成树的类包括插入函数和输出函数。前者负责将经过prim生成的边信息存入到最小生成树中，后者负责最小生成树的输出。

## 2.3成员与操作设计

**Edge结构体**：

struct edge {//边信息

char head;

char tail;

int weight;//权值

edge& operator=(edge& x)//=运算符重载

{

head = x.head;

tail = x.tail;

weight = x.weight;

return \*this;

}

bool operator<=(edge& x)//<=运算符重载

{

return (weight <= x.weight);

}

bool operator>(edge& x)//>运算符重载

{

return(weight > x.weight);

}

};

**Minheap类：**

class Minheap//最小堆

{

public:

Minheap()//构造函数，初始化最小堆

{

edge\_num = 0;

element = new edge[defaultsize];

}

~Minheap()

{

delete[]element;

}

bool removemin(edge& a);//移除堆顶元素

void insert(edge a);//插入元素

void siftdown(int start);//从上到下调整

void siftup(int start);//从下到上调整

private:

edge\* element;//存放边信息的数组

int edge\_num;//存放边的数目

};

**Graphmtx类：**

class graphmtx//邻接矩阵

{

protected:

char\* vertex;//顶点数组

int vertex\_num;//顶点个数

int edge\_num;//边个数

int edge\_[defaultsize][defaultsize];//邻接矩阵

public:

graphmtx()

{

vertex\_num = 0;

edge\_num = 0;

vertex = new char[defaultsize];

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

{

for (int j = 0; j < defaultsize; j++)

{

edge\_[i][j] = (i == j) ? 0: maxweight;

}

}

}

char get\_vertex(int i)

{

return vertex[i];

}

int get\_edge\_num()

{

return edge\_num;

}

int get\_vertex\_num()

{

return vertex\_num;

}

void insert\_vertex(char a);

void insert\_edge(edge a);

int getvertexpos(char a);

int getfirstneighbor(int a);

int getnextneighbor(int w, int v);

int getweight(int a, int b)

{

return edge\_[a][b];

}

};

**Minspantree类：**

class Minspantree//最小生成树

{

protected:

edge\* element;//边信息

int edge\_num;//边的个数

public:

Minspantree()

{

edge\_num = 0;

element = new edge[defaultsize];

}

~Minspantree()

{

delete []element;

}

void insert(edge a);

void output();

};

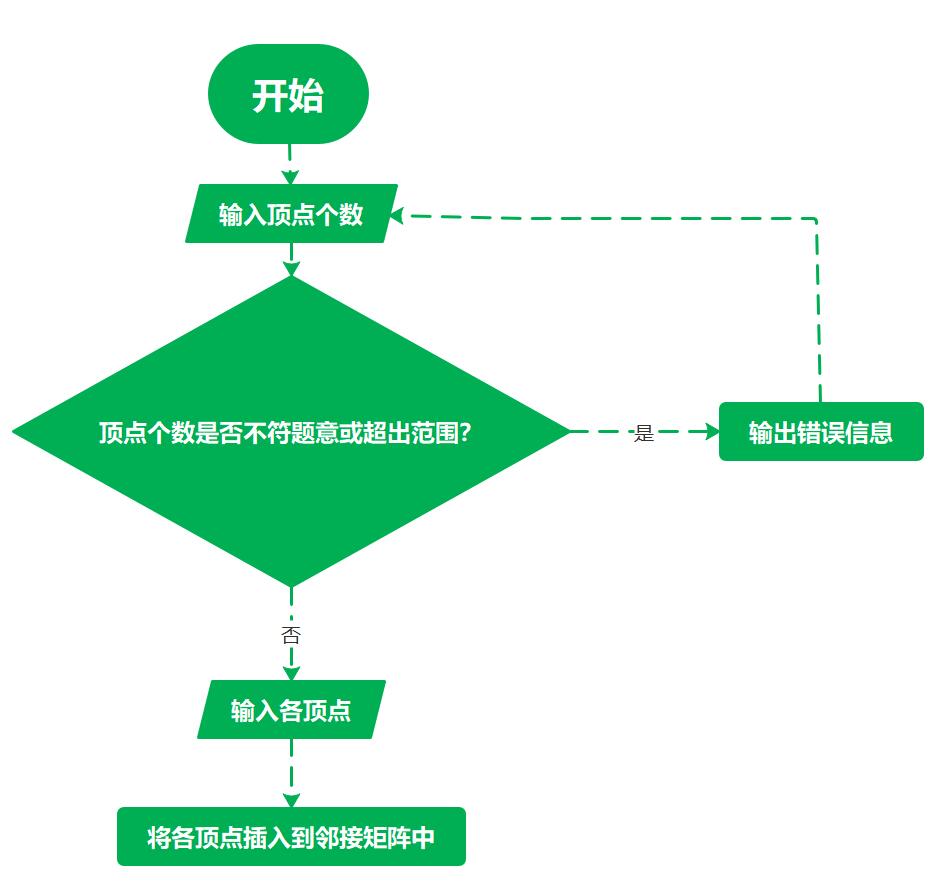
## 2.4系统设计

用户首先在main函数中生成一个实例化邻接矩阵和生成树，调用menu函数输出可视化菜单，通过input函数，让用户输入操作码来执行相应操作。

# 实现

## 3.1创建顶点

### 3.1.1流程图



### 3.1.2代码实现

void create\_vertex(graphmtx& c)

{

int n = 0;

char a;

while (1)

{

cout << "请输入顶点的个数：" << endl;

cin >> n;

if (n > 0 && n < defaultsize)

{

break;

}

else

{

cout << "顶点个数有误，请重新输入！" << endl;

}

}

cout << "请依次输入各顶点的名称：" << endl;

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

{

cin >> a;

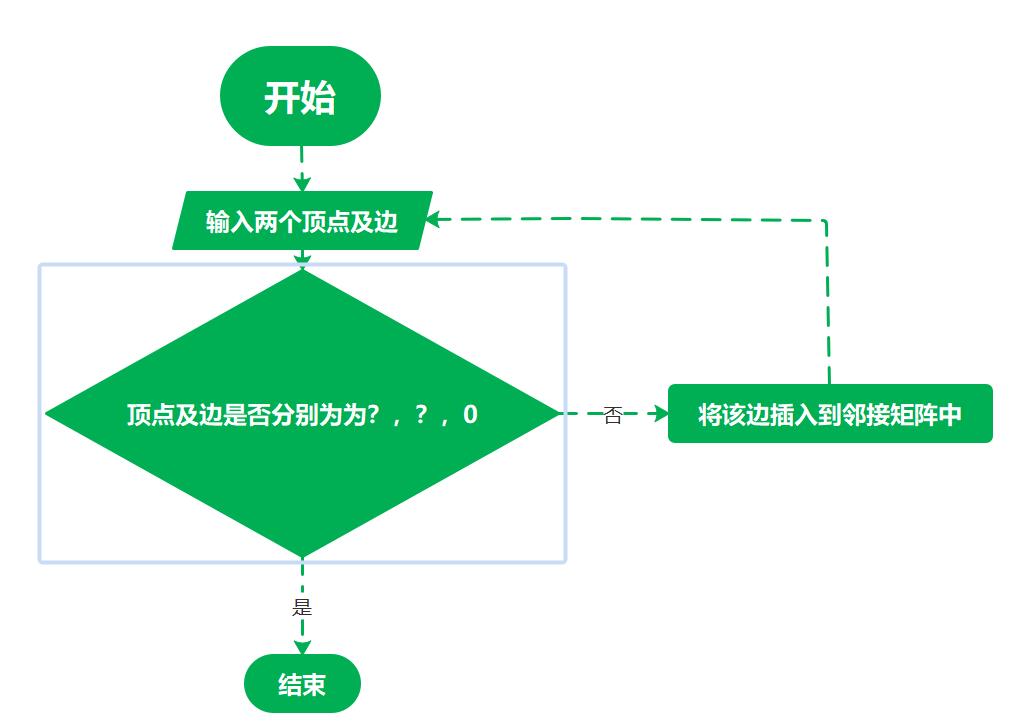
c.insert\_vertex(a);

}

}

## 3.2创建边

### 3.2.1流程图



### 3.2.2代码实现

void create\_edge(graphmtx& c)

{

char a, b;

int n;

edge x;

while (1)

{

cout << "请输入两个顶点及边：";

cin >> a;

cin >> b;

cin >> n;

if (a == '?' || b == '?' || n == 0)

{

break;

}

else

{

x.head = a;

x.tail = b;

x.weight = n;

c.insert\_edge(x);

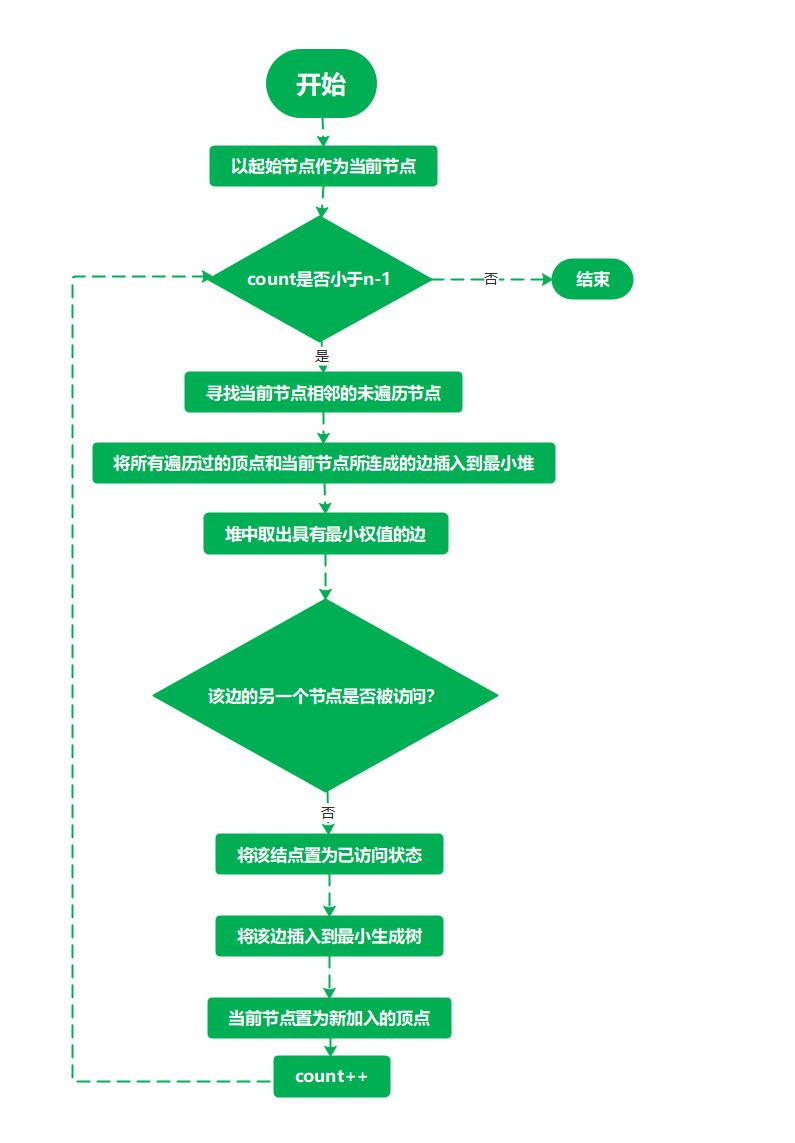
}

}

}

## 3.3构造最小生成树(Prim算法实现）

### 3.3.1流程图



### 3.3.2代码实现

void Prim(graphmtx& G, char u0, Minspantree& MST)//求最小生成树

{

Minheap H;

edge x;

int n = G.get\_vertex\_num();

int m = G.get\_edge\_num();

int u = G.getvertexpos(u0);

int v = 0;

int p = 0;

int q = 0;

bool\* visited = new bool[n];

for (int i = 0; i < n; i++)//给遍历数组赋初值

{

visited[i] = false;

}

visited[u] = true;//顶点已访问

int count = 0;

while (count < n - 1)//生成最小生成树

{

v = G.getfirstneighbor(u);//取当前位置的相邻位置

while (v != -1)//若存在

{

if (visited[v] == false)//未被访问

{

x.head = G.get\_vertex(u);

x.tail = G.get\_vertex(v);

x.weight = G.getweight(u, v);

H.insert(x);//将该边送入最小堆

}

v = G.getnextneighbor(u, v);//获取下一个位置

}

H.removemin(x);//取出最小堆的最小元素

p = G.getvertexpos(x.tail);

q = G.getvertexpos(x.head);

if (u == p)

{

v = q;

}

else

{

v = p;

}

if (visited[v] == false)

{

visited[v] = true;//该位置已经遍历过，做标记

MST.insert(x);//插入最小生成树

count++;

u = v;

}

}

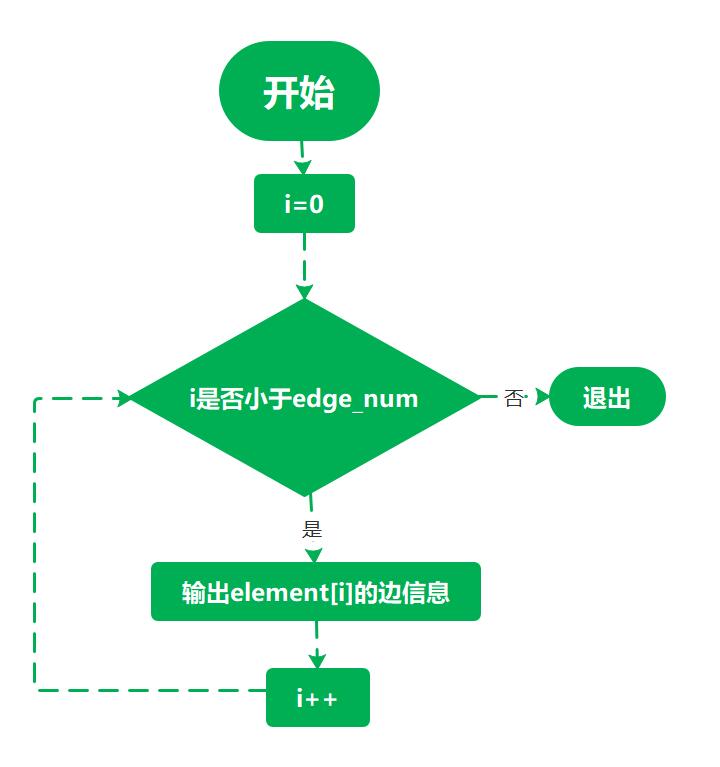
}

### 3.3.3文字说明

prim算法(普利姆算法)：对图G(V,E)设置集合S，存放已访问的顶点，然后每次从集合V-S中选择与集合S的最短距离最小的一个顶点(记为u)，访问并加入集合S。之后，令顶点u为中介点，优化所有从u能到达的顶点v与集合S之间的最短距离。执行n次(n为顶点个数)，直到集合S已包含所有顶点。

## 3.4展示最小生成树

### 3.4.1流程图



### 3.4.2代码实现

void Minspantree::output()

{

cout << "最小生成树的顶点及边为：" << endl;

for (int i = 0; i < edge\_num; i++)

{

cout << element[i].head << "-<" << element[i].weight << ">->" << element[i].tail<<" ";

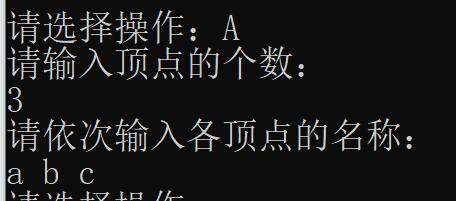
}

}

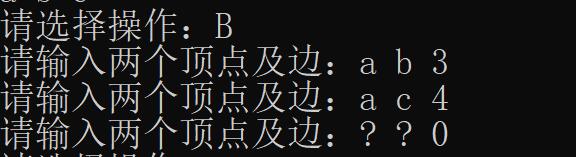
# 测试

## 4.1功能测试

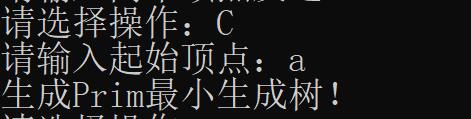
### 4.1.1创建顶点



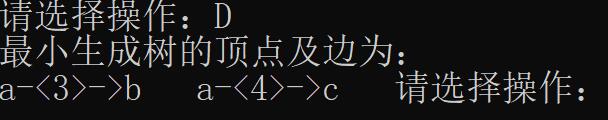
### 4.1.2创建边



### 4.1.3构造最小生成树

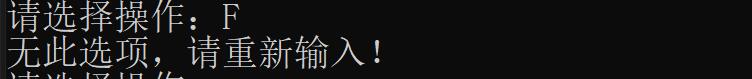


### 4.1.4展示最小生成树

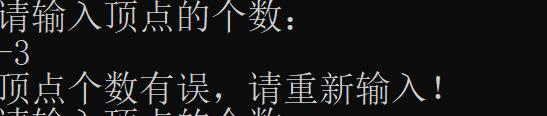


## 4.2错误测试

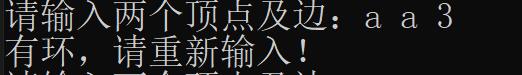
### 4.2.1操作码输入错误



### 4.2.2顶点个数输入不正确



### 4.2.3有环产生



### 4.2.4起始顶点不存在

