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： 王星洲

学 号： 1652977

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](#_Toc534066875)

[1.1 背景分析 1](#_Toc534066876)

[1.2 功能分析 1](#_Toc534066877)

[2 设计 2](#_Toc534066878)

[2.1 数据结构设计 2](#_Toc534066879)

[2.2 类结构设计 2](#_Toc534066880)

[2.3 成员与操作设计 2](#_Toc534066881)

[2.4 系统设计 3](#_Toc534066882)

[3 实现 4](#_Toc534066883)

[3.1 创建链表 4](#_Toc534066884)

[3.1.1 创建链表核心代码 4](#_Toc534066885)

[3.1.2 创建说明 4](#_Toc534066886)

[3.1.3 初始化截屏示例 4](#_Toc534066887)

[3.2 插入功能的实现 5](#_Toc534066888)

[3.2.1 插入功能流程图 5](#_Toc534066889)

[3.2.2 插入功能核心代码 6](#_Toc534066890)

[3.2.3 插入功能截屏示例 7](#_Toc534066891)

[3.3 删除功能的实现 8](#_Toc534066892)

[3.3.1 删除功能流程图 8](#_Toc534066893)

[3.3.2 删除功能核心代码 9](#_Toc534066894)

[3.3.3 删除功能截屏示例 10](#_Toc534066895)

[3.4 查找功能的实现 11](#_Toc534066896)

[3.4.1 查找功能流程图 11](#_Toc534066897)

[3.4.2 查找功能核心代码 12](#_Toc534066898)

[3.4.3 查找功能截图示例 13](#_Toc534066899)

[3.5 修改功能的实现 14](#_Toc534066900)

[3.5.1 修改功能流程图 14](#_Toc534066901)

[3.5.2 修改功能核心代码 15](#_Toc534066902)

[3.5.3 修改功能截屏示例 16](#_Toc534066903)

[3.6 统计功能的实现 17](#_Toc534066904)

[3.6.1 统计功能流程图 17](#_Toc534066905)

[3.6.2 统计功能核心代码 18](#_Toc534066906)

[3.6.3 统计功能截屏示例 18](#_Toc534066907)

[3.7 总体系统的实现 19](#_Toc534066908)

[3.7.1 总体系统流程图 19](#_Toc534066909)

[3.7.2 总体系统核心代码 20](#_Toc534066910)

[4 测试 23](#_Toc534066911)

[4.1 功能测试 23](#_Toc534066912)

[4.1.1 插入功能测试 23](#_Toc534066913)

[4.1.2 删除功能测试 24](#_Toc534066914)

[4.1.3 查找功能测试 24](#_Toc534066915)

[4.1.4 修改功能测试 25](#_Toc534066916)

[4.1.5 统计功能测试 26](#_Toc534066917)

[4.2 边界测试 27](#_Toc534066918)

[4.2.1 初始化无输入数据 27](#_Toc534066919)

[4.2.2 删除后链表为空 27](#_Toc534066920)

[4.3 出错测试 28](#_Toc534066921)

[4.3.1 考生人数错误 28](#_Toc534066922)

[4.3.2 操作码错误 28](#_Toc534066923)

[4.3.3 插入位置不存在 28](#_Toc534066924)

[4.3.4 删除考号不存在 29](#_Toc534066925)

[4.3.5 查找考号不存在 29](#_Toc534066926)

[4.3.6 修改考号不存在 30](#_Toc534066927)

[5 亮点 31](#_Toc534066928)

[5.1 人机交互 31](#_Toc534066929)

[5.2 带头结点链表的使用 31](#_Toc534066930)

# 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。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。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该系统要求实现建立最小生成树，因此需要使用克鲁斯卡尔和普里姆算法之一，这里我选择Prim算法。Prim算法需要一个最小堆存储所有边，这样就可以找出最小的一条边。因此设计了MinHeap类。而用户的输入为string类型的变量，而对于之后的处理，string明显不便，用int指代对应的顶点更加便利，于是我们创建一个vector<string>name,一个map<int, string>points，这样就能够使结点名称和索引关联起来，之后解决问题就可以使用索引来解决。接受完用户的输入后，我们需要将图保存来进行Prim算法，而这里存储图的方式我选择了使用邻接矩阵，邻接矩阵本身是个二维数组，因此使用vector<vector<int> >edges来保存，在接收用户输入及Prim算法中很多位置需要用到结点数，因此结点数用一个全局变量int number来储存，使用以上数据结构即可以解决问题。

## 2.2 类结构设计

本题使用的类是MinHeap，它需要存入图中的边，而边需要用前后两个结点和边的权值来表示，所以使用一个结构struct ed来存储边两个结点的索引即自身权值三个int变量。这样在堆中对结点的权值进行排序，就可以得到最短的边以及其两边的结点，便于后续处理。

## 2.3 成员与操作设计

//边的结构

struct ed {

int tail;

int head;

int key;

};

//最小堆

class MinHeap{

public:

MinHeap(int sz = DefaultSize);//构造函数：建立空堆

~MinHeap()

{

delete[]heap;

}

bool Insert(const ed &x);

bool RemoveMin(ed &x);

bool IsEmpty()const {

return currentSize == 0;

}

bool IsFull()const {

return currentSize == maxHeapSize;

}

void MakeEmpty() {

currentSize = 0;

}

void output() {//自定义函数，顺序输出最小堆元素

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

cout << heap[i].head << "-<" << heap[i].key << ">-" << heap[i].tail << " ";

cout << endl;

}

private:

ed \*heap; //存放最小堆中元素的数组

int currentSize; //最小堆中当前元素个数

int maxHeapSize; //最小堆最多允许元素个数

void siftDown(int start, int m);//从start到m下滑调整成为最小堆

void siftUp(int start); //从start到0上滑调整成为最小堆

};

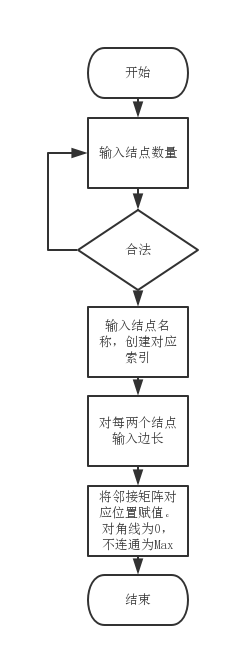
## 2.4 系统设计

系统通过循环和选择语句接收用户的指令，直到退出系统。但是代码中对执行顺序有限制，必须要满足程序按照a。b。c。d的顺序执行，所以相当于一个线性过程。

# 3 实现

## 3.1 接收输入完成图的创建

### 3.1.1 创建功能流程图



### 3.1.2 创建功能核心代码

void inputPoint(map<int, string>& points, vector<string> & nameArray)

{

if (number != 0)

{

cout << "您已经输入过结点，请勿重复输入" << endl;

return;

}

string name;

cout << "请输入顶点的个数：";

while (1)

{

if (!(cin >> number))

{

cout << "输入非法，请重新输入" << endl;

cin.clear();

while (cin.get() != '\n')

{

continue;

}

}

else

{

if (number <= 1)

{

cout << "请至少输入两个点用于计算" << endl;

}

else

{

break;

}

}

}

cout << "请依次输入各顶点名称：" << endl;

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

{

cin >> name;

points.insert(pair<int, string>(i, name));

nameArray.push\_back(name);

}

}

void inputEdges(map<int, string>& points, vector<vector<int> >& edges, vector<string>& name)

{

if (number <= 0)

{

cout << "请先输入结点" << endl;

return;

}

if (hasInput == true)

{

cout << "您已输入过边，请勿重复输入" << endl;

return;

}

int number = points.size();

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

cout << "请输入给出的两个顶点间的距离，不存在路径则输入-1: ";

edges[i][i] = 0;

for (int k = i + 1; k < number; k++) {

cout << name[i] << " ";

cout << name[k] << " ";

cin >> edges[i][k];

if (edges[i][k] == -1)

edges[i][k] = Max;

if (k != number - 1)

cout << "请输入给出的两个顶点间的距离，不存在路径则输入-1: ";

}

}

for (int i = 0; i < number; i++) {

for (int k = 0; k < i; k++) {

edges[i][k] = edges[k][i];

}

}

hasInput = true;

cout << endl;

/\*for (int i = 0; i < number; i++)

{

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

{

cout << edges[i][j];

}

cout << endl;

}\*/

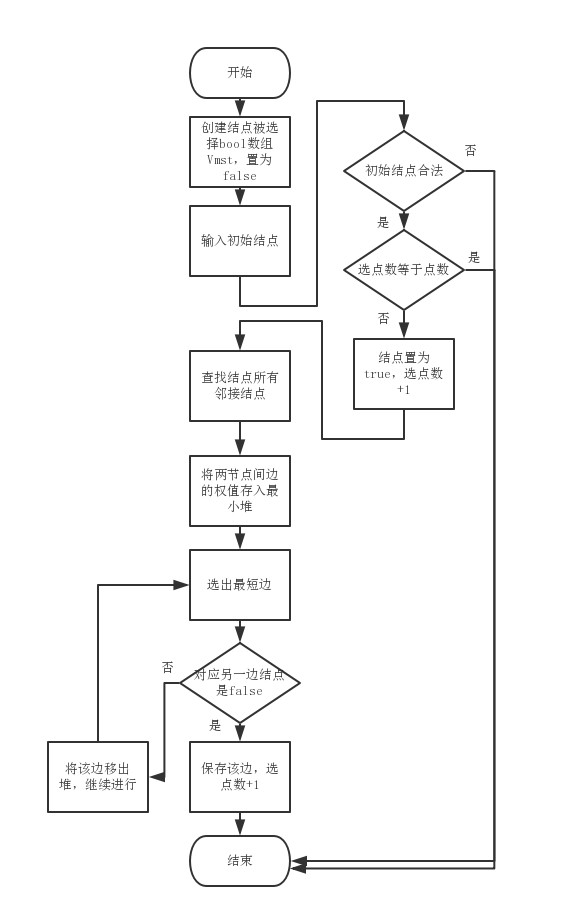
}

### 3.1.3 创建说明

创建过程分为了两个步骤，分别是接收用户输入的点储存为图的结点，接收用户输入的边将邻接矩阵中对应的位置进行赋值。同时，创建代码中有对于合法性的判断，保证程序的健壮性。

## 3.2 普里姆算法的实现

### 3.2.1 普里姆算法流程图



### 3.2.2 普里姆算法核心代码

void Prim(map<int, string>& points, vector<vector<int> >& edges, vector<string>& name, vector<ed>& MST)

{

if (number == 0)

{

cout << "请先输入结点" << endl;

return;

}

if (hasInput == false)

{

cout << "请先输入边" << endl;

return;

}

if (MST.size() != 0)

{

cout << "您已生成过最小生成树，赶快按d查看吧" << endl;

return;

}

ed helper;

string first;

int count, v, col;

int edgesNumber = number \* (number - 1) / 2;

int i = 0;

vector<bool> Vmst(number);

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

{

Vmst[j] = false;

}

MinHeap H(edgesNumber);

cout << "请输入起始点:";

cin >> first;

for (i = 0; i < name.size(); i++)

{

if (first == name[i])

break;

}

if (i == name.size())

{

cout << "没有找到该结点，请检查您的输入" << endl;

return;

}

else

{

Vmst[i] = true;

count = 1;

do {

if (i != -1)

{

for (col = 0; col < number; col++)

{

if (edges[i][col] > 0 && edges[i][col] < Max)

{

v = col; break;

}

}

if (col == number)

{

v = -1;

}

}

while (v != -1)

{

if (Vmst[v] == false)

{

helper.tail = i; helper.head = v;

helper.key = (edges[i][v]);

H.Insert(helper);

}

if (i != -1 && v != -1)

{

for (col = v + 1; col < number; col++)

{

if (edges[i][col] > 0 && edges[i][col] < Max)

{

v = col;

break;

}

}

if (col == number)

{

v = -1;

}

}

}

while (H.IsEmpty() == false && count < number)

{

H.RemoveMin(helper);

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

{

MST.push\_back(helper);

i = helper.head;

Vmst[i] = true;

count++; break;

}

}

} while (count < number);

cout << "生成Prim最小树！" << endl;

return;

}

}

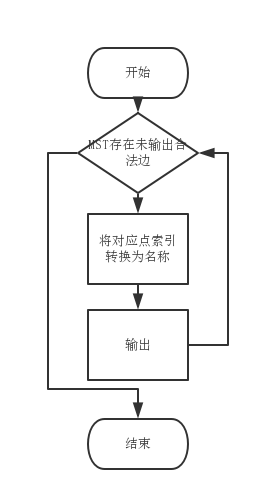
### 3.2.3 普里姆算法说明

普里姆算法又称为选点法，从一个指定的顶点开始，将它所有相邻的边加入最小堆，然后从最小堆中选出堆顶的边，看它的另一个顶点是否已经被选择过，如果没有，那么这条边就作为结果的一条边，并且将对应的顶点记录下来，重复执行直到所有点都被选完。

在本题的实现中我采用了bool数组记录哪些顶点被选中过，并妥善使用了最小堆H和边结构helper实现了prim算法。

## 3.3 输出功能的实现

### 3.3.1 输出功能流程图



### 3.3.2 输出功能核心代码

void showMST(map<int, string>& points, vector<ed>& MST)

{

if (MST.size() == 0)

{

cout << "并没有可见的最小生成树，请检查之前的步骤" << endl;

return;

}

cout << "最小生成树的顶点及边为：" << endl;

cout << endl;

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

{

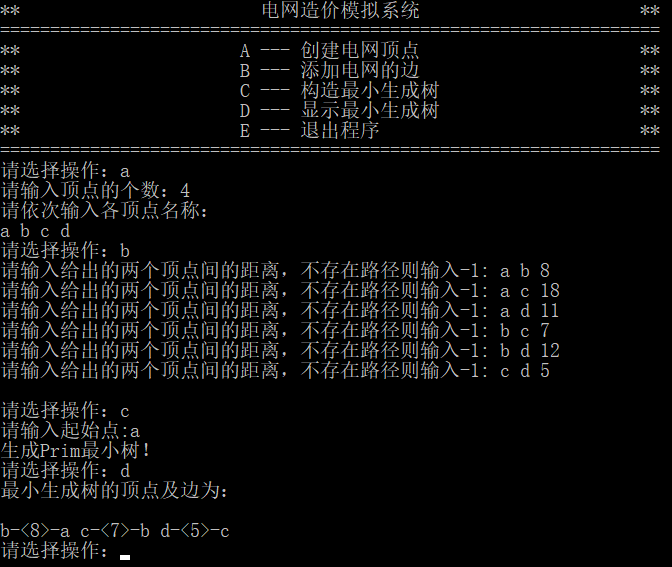
cout << points[MST[i].head] << "-<" << MST[i].key << ">-" << points[MST[i].tail] << '\t';

}

cout << endl;

}

### 3.3.3 输出功能截屏示例



# 4 测试

## 4.1 功能测试

### 4.1.1 一般功能测试

**测试用例：**

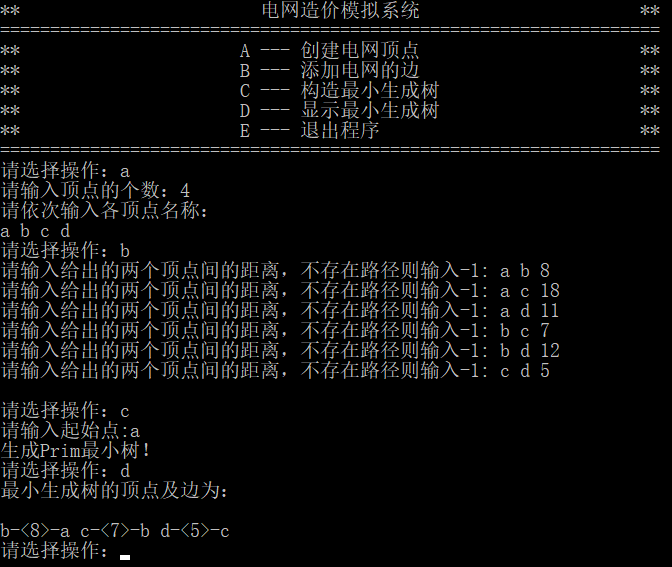
4个顶点a b c d

距离 ab 8 ac 18 ad 11 bc 7 bd 12 cd 5

**预期结果：**

b-8-a c-7-b d-5-c

**实验结果：**



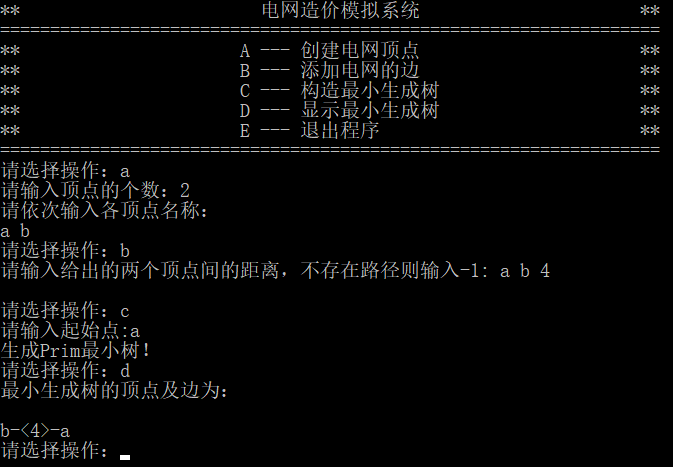
## 4.2 边界测试

### 4.2.1 只有两点的情况

**测试用例：**只输入两个结点a b 边ab=4

**预期结果：**b-4-a

**实验结果：**



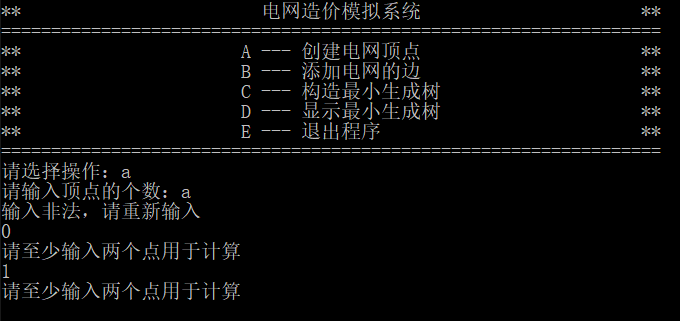
## 4.3 出错测试

### 4.3.1 点数不在范围内

**测试用例：**输入点数为字母或小于2的数字

**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：**

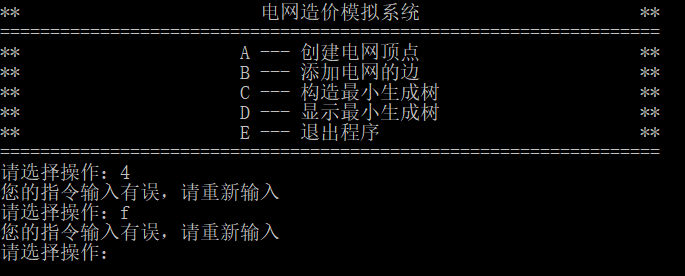


### 4.3.2 操作码错误

**测试用例：**输入操作码错误

**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：**

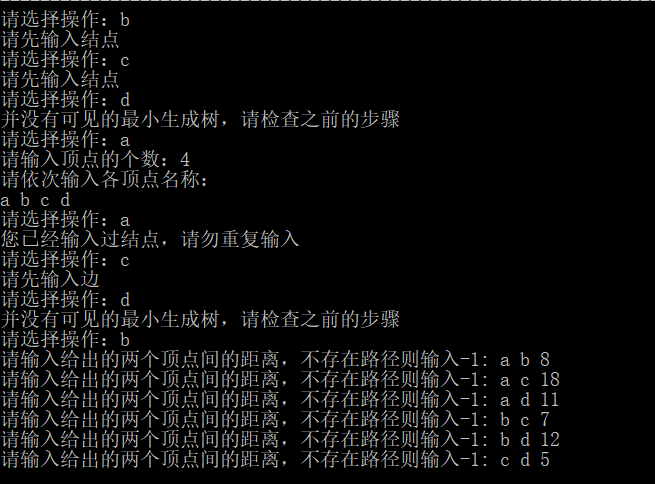


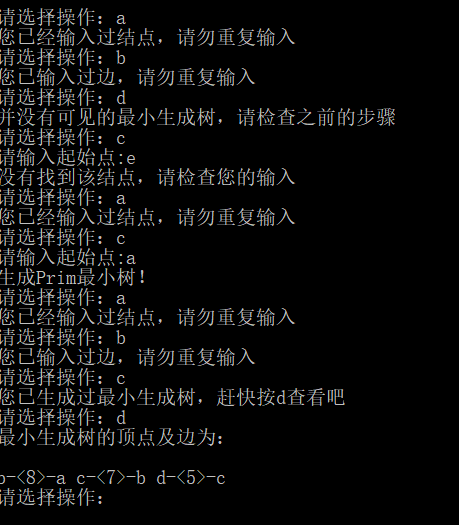
### 4.3.3 未按顺序执行算法

**测试用例：**还未插入结点/创建边/生成最小树就打算输出

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**





# 5 亮点

## 5.1 人机交互

本题中我添加了大量人机代码满足人机交互需求，防止意外退出或者崩溃。详情见4.3

## 5.2 最小堆的使用

本题中，为了使链表的使用更加方便快捷，我使用了最小堆MinHeap类，详情见2.2 3.2