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： 崔鑫宇

学 号： 1853444

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

# 1 分析

## 1.1 背景分析

电网的建设是当今社会中比较常见的一种工程，通常为了使一块区域内电网建设的成本，我们需要找到这块区域内可以将所有单位连接起来的最短线路，而如果人工对最短线路进行计算的话，如果区域较大便无法保证计算结果的稳定性，因此我们有必要开发出一款程序来代替我们计算最佳的电网建设方式。

## 1.2 功能分析

作为一个最简易的电网建设模拟系统，假设一个城市有n个小区，如果要实现n个小区之间的电网都能够相互接通的话。首先在每个小区之间都可以设置一条电网线路，因此n个小区之间要从中选出n-1条线路使总的耗费最少。

因此,我们的电网造价模拟建设系统应该有创建电网顶点、添加电网的边、构建最小生成树以及显示最小生成树的功能。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该系统需要用到的数据结构就是最小生成树。

## 2.2 类结构设计

在本次设计中我设计了三个类分别是堆(Heap)类、图(Graph)类、以及最小生成树(MinSpanTree)类。

## 2.3 成员与操作设计

**堆类(Heap)**

public:

Heap(int size = DefaultValue);

~Heap() { delete[]heap; }

bool insert(const Node<T1, T2>& x);//插入元素

bool extract(Node<T1, T2>& item);//提取元素

//调整堆中元素的位置

void siftDown(int start, int end);

void siftUp(int start);

int length() { return currentSize; }//返回目前堆的长度

bool isEmpty() { return currentSize == 0; }//判断是否为空

private:

Node<T1, T2> \*heap;

int currentSize;//堆的目前大小

int maxSize;//堆的容量

**图类（Graph）**

public:

Graph();

~Graph() = default;

int getNumVertices() { return numVertices; }//取得点

int getNumEdges() { return numEdges; }//取得边

//取值

T1 getValue(int i) {

return (i >= 0 && i < numVertices) ? NodeTable[i].data : '0';

}

//取长度

T2 getWeight(int v1, int v2);

//插入点

bool insertVertice(const T1& vertices);

//插入边

bool insertEdge(int v1, int v2, T2 cost);

//取得第一个相邻单位

int getFirstNeighbor(int v);

//取得下一个相邻单位

int getNextNeighbor(int v, int w);

//取得点的位置

int getVerticePos(const T1 vertices) {

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

if (NodeTable[i].data == vertices) {

return i;

}

}

return -1;

}

private:

Vertices<T1, T2> \*NodeTable;

int numVertices;//点

int maxVertices;//最大点数

int numEdges;//边

**最小生成树类（MinSpanTree）**

public:

Node<T1, T2>\* value;//容器

MinSpanTree(int sz = DefaultValue - 1) : maxSize(sz), n(0) {

value = new Node<T1, T2>[sz];

}

//插入数据

bool InsertNode(Node<T1, T2>& node) {

if (n == maxSize) {

cerr << "达到可容纳的最大值" << endl;

return false;

}

else {

value[n] = node;

n++;

return true;

}

}

private:

//最小生成树的最大容量

int maxSize, n;

## 2.4 系统设计

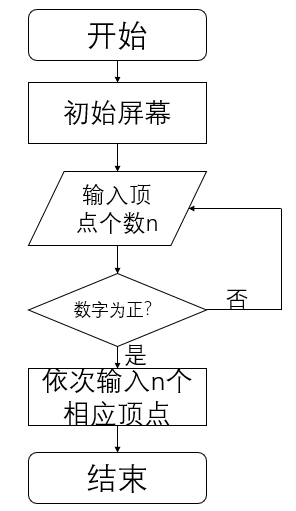
系统首先调用printScreen()函数实现对屏幕的初始化，完成根据用户所输入的操作码执行所设计的对应的函数。

# 

# 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

cout << "请输入顶点的个数：";

int n = 0;

while (n <= 0) {

cin >> n;

if (n <= 0) {

cout << "点的个数应大于0！请重新输入:";

}

}

num = n;

cout << "请输入各个顶点的名称：";

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

char name;

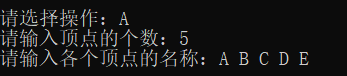
cin >> name;

G.insertVertice(name);

}

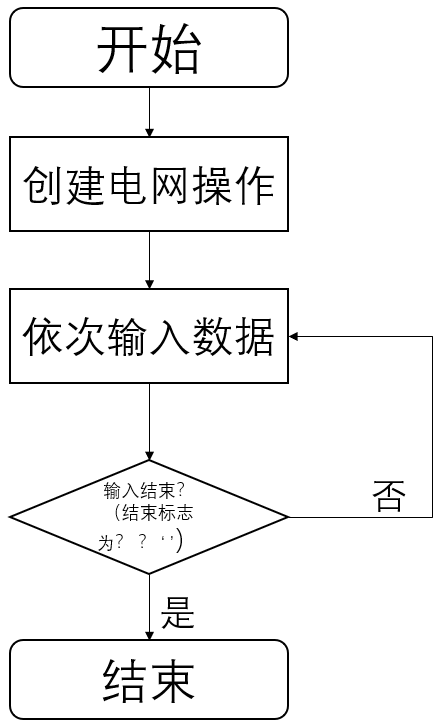
### 

### 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

bool i = 1;

while (i) {

cout << "请输入两个顶点及边：";

char v1,v2;

int weight;

cin >> v1 >> v2 >> weight;

if (v1 == '?' || v2 == '?') {

i = 0;

break;

}

int v1\_pos = G.getVerticePos(v1);

int v2\_pos = G.getVerticePos(v2);

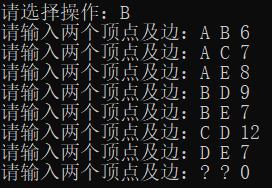
if (!G.insertEdge(v1\_pos, v2\_pos, weight)) {

cerr << "顶点创建失败，请检查后重新输入!" << endl;

}

}

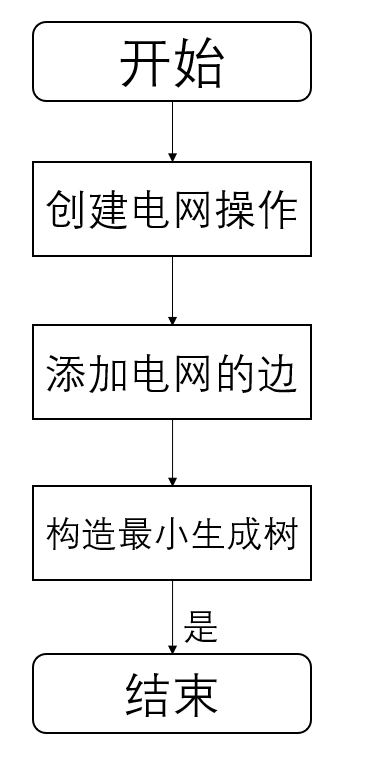
### 3.2.3 添加电网的边功能截屏示例



## 

## 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

Node<T1, T2> tn;

int a, b, count;

int n = G.getNumVertices();

int m = G.getNumEdges();

a = G.getVerticePos(x);

Heap<T1, T2> H(m);

bool \*v = new bool[n];

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

v[i] = false;

}

v[a] = true;

count = 1;

while (count < n) {

b = G.getFirstNeighbor(a);

while (b != -1) {

if (v[b] == false) {

tn.head = b;

tn.tail = a;

tn.key = G.getWeight(a, b);

H.insert(tn);

}

b = G.getNextNeighbor(a, b);

}

while (!(H.isEmpty()) && count < n) {

H.extract(tn);

if (v[tn.head] == false) {

MinSizeTree.InsertNode(tn);

a = tn.head;

v[a] = true;

count++;

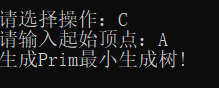
break;

}

}

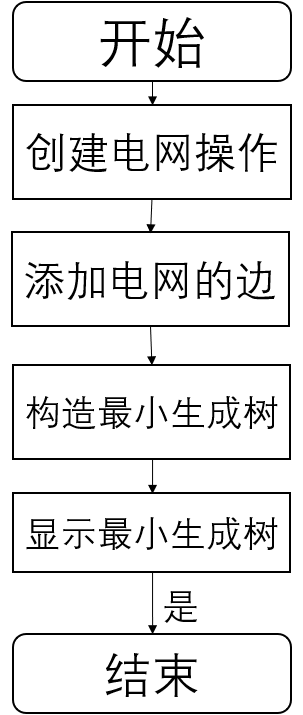
### }

### 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

cout << "最小生成树的顶点及边分别为：" << endl;

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

char head = G.getValue(MSTree.value[i].tail);

char tail = G.getValue(MSTree.value[i].head);

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

if (i != numVertices - 2) {

cout << "\t";

}

}

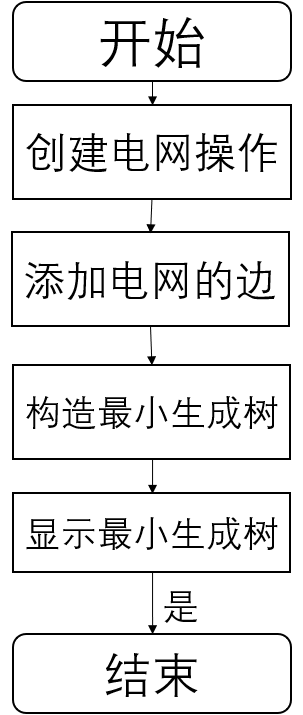
### cout << endl;

### 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

printScreen();

Graph<char, int> G;

MinSpanTree<char, int> MSTree;

int numVertices = 0;

char opt;

bool i = 1;

int isBuilt = 0;

//各种功能的实现

while (i == 1) {

cout << endl << "请选择操作：";

cin >> opt;

switch (opt) {

case 'A':

if (isBuilt == 1) {

cerr << "不要重复创建电网!" << endl;

}

else {

addVertexes(G, numVertices);

isBuilt = 1;

}

break;

case 'B':

if (isBuilt == 0) {

cerr << "请先创建电网!" << endl;

}

else {

addEdges(G);

}

break;

case 'C':

if (isBuilt == 0) {

cerr << "请先创建电网!" << endl;

break;

}

cout << "请输入起始顶点：";

char begin;

cin >> begin;

Prim(begin, G, MSTree);

cout << "生成Prim最小生成树!" << endl;

break;

case 'D': {

if (isBuilt == 0) {

cout << "请先创建电网!" << endl;

break;

}

else {

cout << "最小生成树的顶点及边分别为：" << endl;

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

char head = G.getValue(MSTree.value[i].tail);

char tail = G.getValue(MSTree.value[i].head);

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

if (i != numVertices - 2) {

cout << "\t";

}

}

cout << endl;

break;

}

}

case 'E':

i = 0;

break;

default:

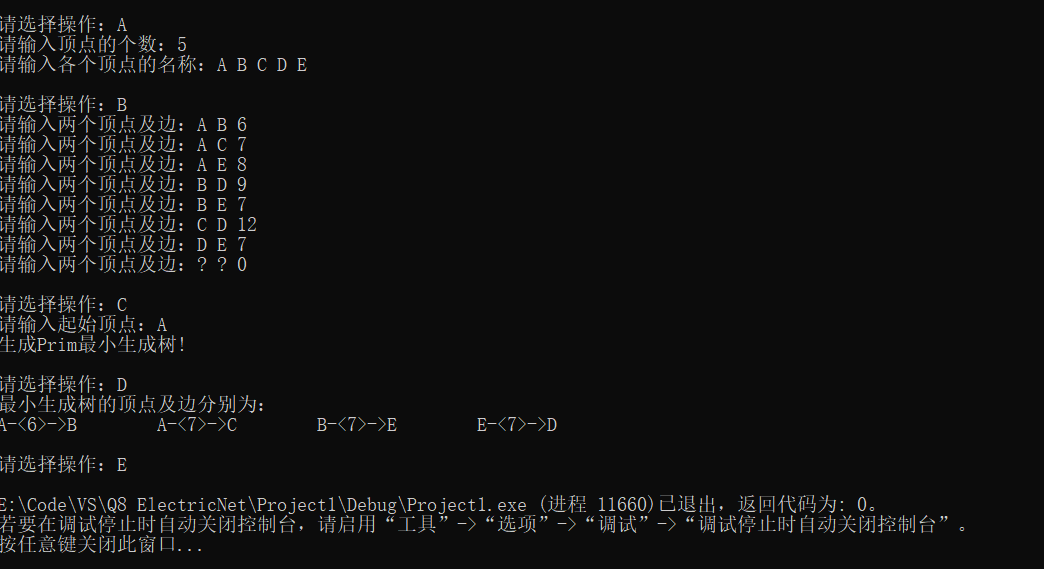
cout << "选项不存在，请重新输入" << endl;

break;

}

### }

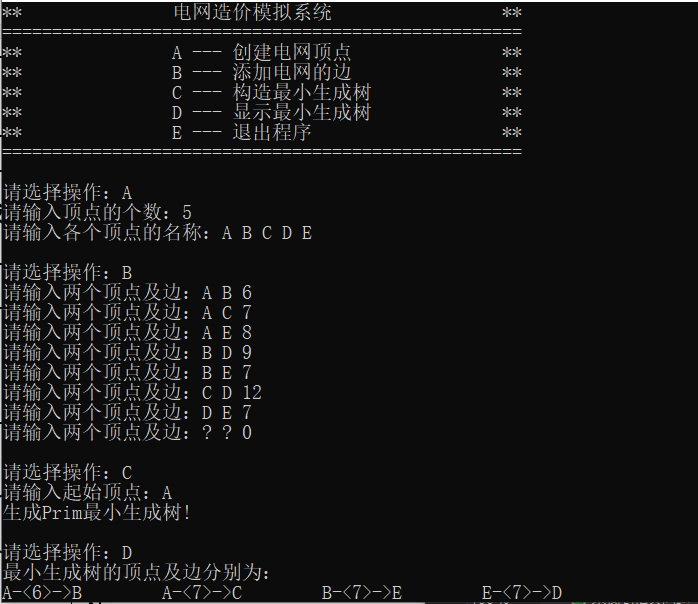
### 3.6.3 总体系统截屏示例



# 

# 4 测试

## 4.1 常规功能测试



## 

## 4.2 出错测试

### 4.2.1 选项不存在

**测试用例：**不存在的选项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：**

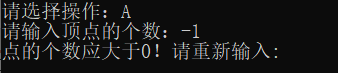


### 4.2.2 添加顶点数小于0

**测试用例：**-1

**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：**



### 4.2.3 重复创建电网

**测试用例：**创建电网后重新输入操作码A

**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：**

****