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： 肖杨

学 号： 1950430

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 3

2 设计 3

3 实现 12

4 测试 16

1 分析

1.1 背景分析

在电力工程管理中，造价管理作为一个重要管理环节，主要是依靠基于各种经济技术和法律于段 而制定的科学、适宜的策略，加强对电力造价的管理，以此达到有效控制电力成本的目的，从而 确保电力企业能够获得更人的经济利益。现阶段，电力造价管理因受成本上涨、融资渠道减少等 多方而因素的影响，导致其的运行和管理出现了问题。针对这些问题，应利用更加科学、有效地 控制策略，加强对电力造价成本的管理，以便确保电力企业能够从电力工程中获得更人的利益。

1.2 功能分析

本实验项目场景是在一个城市中有n个小区，要实现n个小区之间的电网的相互接通，构造这个城市n个小区之间的电网，使得总工程造价最低，设计一个能够满足要求的造价方案。据此可知所需的功能有：

1.创建电网节点

2.向电网中添加边

3.构造最小生成树

4.显示最小生成树

2 设计

2.1 算法设计

一个连通图的生成树是一个极小的连通子图，它包含图中全部的顶点（n个顶点），但只有n-1条边。

最小生成树：构造连通网的最小代价（最小权值）生成树。

可以看出，我们要求的最小电网构造就是求电路网络的最小生成树。

在这里，我们选用Kruskal算法求电路网络的最小生成树。

Kruskal算法过程为：

首先，对一个有n个端点的连通图，将其边按权值进行排序。

取出权值最小的边，判断其两个端点是否在一个等价类中，若在，则取出下一个边，否则将边加入答案数组，并将两个端点的等价类合并。

当拥有n-1条边时，则生成完毕。

当边遍历完毕仍不够n-1条边，说明图不连通。

2.2 数据结构设计

如上功能分析所述，我们需要存储小区以及小区和小区之间电网的造价。所以考虑采用图的数据结构，用顶点表示小区，用边上的权值表示小区之间电网的造价。同时考虑到处理的情况大多为稀疏图，这里还选用了堆来优化Kruskal算法。同时使用并查集来判断是否在一个等价类。

2.3 类结构设计

对于并查集，我们使用一个哈希表来进行储存以加速查找、合并操作。对于最小堆，使用模板类来保证泛用性。边类储存边的各种数据。同时最小生成树不需要实现一个完整的图类，我们另外实现一个电网类在内部进行各种操作。在电网类中需要一个哈希集合进行输入检测。

2.4 成员与操作设计

哈希表类

template<class v>

class unordered\_map

{

private:

class Node

{

public:

Node(string ikey, v ivalue)

{

key = ikey;

value = ivalue;

next = nullptr;

}

Node()

{

next = nullptr;

}

~Node()

{

while (next)

{

Node\* temp = next;

next = temp->next;

delete temp;

}

}

string key;

v value;

Node\* next;

};

int hashSize;

Node\* hashnode;

int hash(string key);

public:

unordered\_map(int size);

~unordered\_map();

Node\* find(string key);

v& operator [] (string key);

};

内部类：Node

私有成员：

int hashSize;

//哈希表大小

Node\* hashnode;

//储存开散列的首节点

私有操作：

int hash(string key);

//哈希化

公有操作：

unordered\_map(int size);

//用桶个数构造函数

Node\* find(string key);

//查找元素并返回地址

v& operator [] (string key);

//重载中括号

哈希集合类

class unordered\_set

{

private:

class Node

{

public:

Node(string ivalue)

{

value = ivalue;

next = nullptr;

}

Node()

{

next = nullptr;

}

string value;

Node\* next;

};

int hashSize;

Node\* hashnode;

int hash(string value);

public:

unordered\_set(int size);

~unordered\_set();

void insert(string value);

Node\* find(string value);

};

私有成员：

int hashSize;

//哈希集合大小

Node\* hashnode;

//储存开散列的首节点

私有操作：

int hash(string key);

//哈希化

unordered\_set(int size);

//用桶个数构造函数

void insert(string value);

//向集合中插入键

Node\* find(string value);

//查找键是否在集合中

并查集类

template<class T>

class UnionFindSet

{

private:

class VW

{

public:

T value;

int weight;

VW(T v, int w) :value(v), weight(w) {}

VW() = default;

};

int num;

unordered\_map<VW> buffer;

public:

UnionFindSet(int size, T\* elements);

void unionElement(T a, T b);

T find(T a);

bool isUnioned(T a, T b);

};

内部类：VW 用于存放其所在等价类和该等价类所含有的元素数量

私有成员：

int num;

//拥有的成员数目

unordered\_map<VW> buffer;

//储存每个元素和其所在等价类的哈希表

公有操作：

UnionFindSet(int size, T\* elements);

//用一定大小的数组进行初始化的构造函数

void unionElement(T a, T b);

//将两个成员所在等价类进行合并

T find(T a);

//查找a所在等价类的根元素

bool isUnioned(T a, T b);

//判断是否在一个等价类

最小堆类

template<class T>

class MinHeap

{

public:

MinHeap(int maxSize);

MinHeap(T arr[], int num);

~MinHeap();

void push(const T& elem);

T pop();

bool isEmpty();

bool isFull();

private:

int size;

int number;

T\* buffer;

void down(int start, int end);

void up(int start);

};

私有成员：

int size;

//堆的大小

int number;

//堆中已有元素

T\* buffer;

//连续地址用于储存堆的元素

私有操作：

void down(int start, int end);

//向下调整

void up(int start);

//向上调整

公有操作：

MinHeap(int maxSize);

//构造空堆

MinHeap(T arr[], int num);

//用数组初始化堆

void push(const T& elem);

//入堆

T pop();

//最小值出堆

bool isEmpty();

//判空

bool isFull();

//判满

边类

class Edge

{

public:

string name1;

string name2;

int cost = 0;

Edge(string a, string b, int number);

Edge() = default;

~Edge() = default;

bool operator<(const Edge &right);

};

公有成员：

string name1;

//顶点1

string name2;

//顶点2

int cost = 0;

//权值

公有操作：

Edge(string a, string b, int number);

//构造函数

Edge() = default;

//默认构造函数

bool operator<(const Edge &right);

//重载小于号用于构建堆

电网类

class electricity

{

private:

int cityNumber = 0;

int edgeNumber = 0;

string\* cities;

unordered\_set\* check;

Edge\* edges;

Edge\* ans;

UnionFindSet<string>\* ufs;

MinHeap<Edge>\* heap;

void getNumber(int& temp);

public:

electricity();

void span();

void printGraph();

};

私有成员：

int cityNumber = 0;

//城市（顶点）数量

int edgeNumber = 0;

//边数量

string\* cities;

//储存城市名称

unordered\_set\* check;

//用于检查后续输入的城市是否在集合中

Edge\* edges;

//储存边

Edge\* ans;

//储存最小生成树的边

UnionFindSet<string>\* ufs;

//并查集

MinHeap<Edge>\* heap;

//最小堆

私有操作：

void getNumber(int& temp);

//输入数字时进行检查

公有操作：

electricity();

//构造函数，进行各种输入

void span();

//生成最小生成树

void printGraph();

//输出最小生成树

2.5 系统设计

系统首先调用电网类构造函数，随后执行span()，若能生成则执行printGraph()函数，否则提示不连通。

3 实现

3.1 添加顶点功能的实现

3.1.1 添加顶点流程

首先，输入正确的顶点数，若不正确则要求重新输入。

随后，输入每个顶点的名称，若出现重名，则要求重新输入重名的名称。顶点名称储存在顶点数组中，并加入哈希集合以进行后续检测。

最后，用顶点数组建立并查集。

3.1.2 核心代码

cout << "请输入城市个数" << endl;

getNumber(cityNumber);

while (cityNumber <= 1)

{

cout << "发生错误，请输入大于1小于INT\_MAX的正整数" << endl;

getNumber(cityNumber);

}

cities = new string[cityNumber];

check = new unordered\_set(cityNumber);

cout << "请输入各城市名称" << endl;

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

{

cin >> cities[i];

if (check->find(cities[i]))

{

cout << "请不要重名" << endl;

i--;

continue;

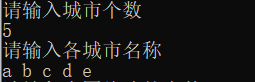
}

check->insert(cities[i]);

}

ufs = new UnionFindSet<string>(cityNumber, cities);

3.1.3 截屏示例



3.2 添加边功能的实现

3.2.1 添加边的流程

首先输入正确的电网线路的个数，若输入不正确则会要求重新输入。

随后，输入每个边的两个顶点和边的权值，过程中要求每个顶点必须在哈希集合中且边的权值合法，以此建立边并存入边数组。

最后，用边数组建立最小堆，并提示输入完毕。

3.2.2 核心代码

cout << "请输入电网线路的个数" << endl;

edgeNumber = 0;

getNumber(edgeNumber);

while (edgeNumber <= 0)

{

cout << "发生错误，请输入小于INT\_MAX的正整数" << endl;

getNumber(edgeNumber);

}

edges = new Edge[edgeNumber];

string temps1, temps2;

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

{

int temp = 0;

cout << "请输入两个城市名与线路价格：" ;

cin >> temps1 >> temps2;

getNumber(temp);

if (check->find(temps1) && check->find(temps2) && temp > 0)

{

edges[i] = Edge(temps1, temps2, temp);

}

else

{

cout << "输入不合法，请重新输入" << endl;

i--;

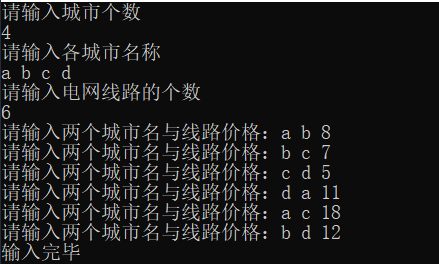
}

}

heap = new MinHeap<Edge>(edges, edgeNumber);

cout << "输入完毕" << endl;

3.2.3 截屏示例



3.3 构建最小生成树并输出功能的实现

3.3.1 构建最小生成树并输出的流程

首先从最小堆中弹出权值最小的边，若边的两个端点在不同的集合中，则将该边加入答案数组中，否则则跳过该边，弹出下一个权值最小的边。

随后，每加入一个边使计数加一，重复弹出操作直到边数为顶点数减一或堆已空。

若边数为顶点数减一，则提示已找到最佳线路并进行答案数组的输出，否则提示线路不连通。

3.3.2 核心代码

void electricity::span()

{

int num = 0;

ans = new Edge[cityNumber - 1];

Edge min;

while (!heap->isEmpty() && num < cityNumber - 1)

{

min = heap->pop();

if (ufs->isUnioned(min.name1, min.name2))

{

continue;

}

ufs->unionElement(min.name1, min.name2);

ans[num++] = min;

}

if (num == cityNumber - 1)

{

cout << "已找到最佳线路" << endl;

printGraph();

}

else

{

cout << "该地区电网不连通，无法找到最佳线路" << endl;

}

}

void electricity::printGraph()

{

cout << "最佳线路为：" << endl;

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

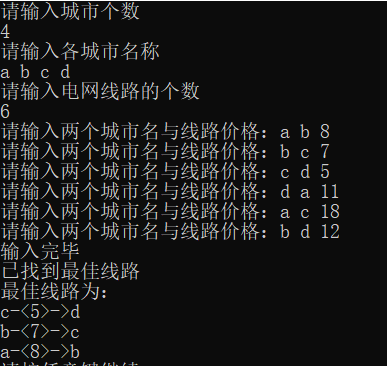
{

cout << ans[i].name1 << "-<" << ans[i].cost << ">->" << ans[i].name2 << endl;

}

}

3.3.3 截屏示例



4 测试

4.1 基本测试

**测试用例**：

4

a b c d

6

a b 8

b c 7

c d 5

d a 11

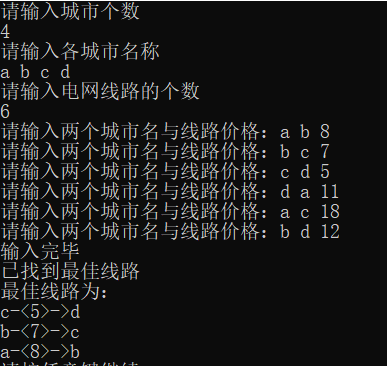
a c 18

b d 12

**预期结果**：

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

**实验结果：**



**测试用例**：

15

a b c d e f g h i j k l m n o

30

a b 2

a c 7

a d 5

a f 13

a i 12

b c 3

b e 9

b k 11

b l 5

b o 4

c h 7

c g 3

c j 14

f m 11

g n 4

g h 7

g o 5

h k 10

h n 2

h j 8

h m 15

h o 3

i j 5

i k 6

i o 20

j m 7

j l 13

m n 3

m o 1

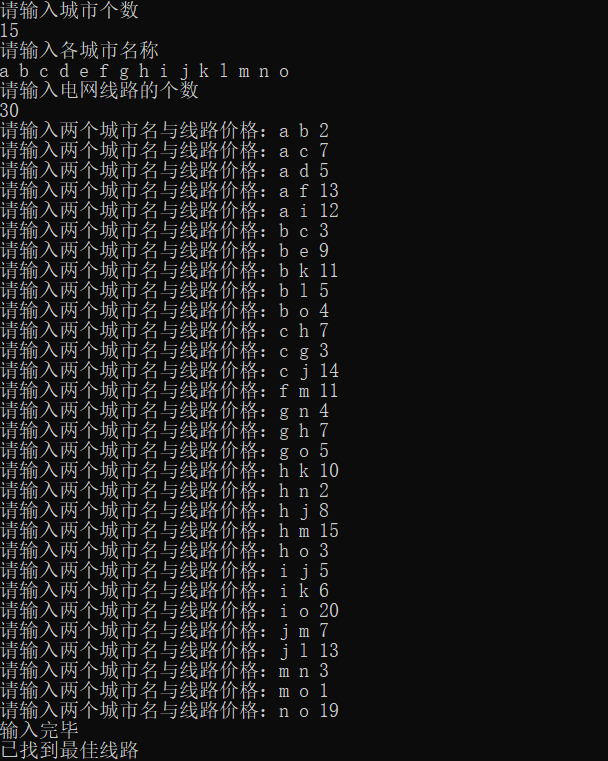
n o 19

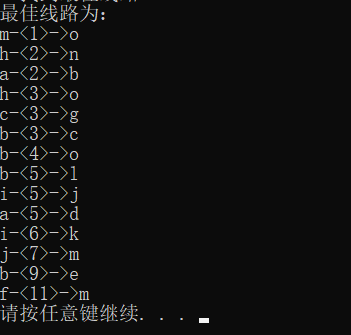
**预期结果**：

m-<1>->o h-<2>->n a-<2>->b h-<3>->o c-<3>->g b-<3>->c b-<4>->o b-<5>->l

i-<5>->j a-<5>->d i-<6>->k j-<7>->m b-<9>->e f-<11>->m

**实验结果：**





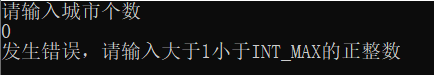
4.2 边界测试

4.2.1 顶点数小于2

**测试用例：**新建顶点数为0

**预期结果：**给出错误提示，程序运行正常不崩溃。

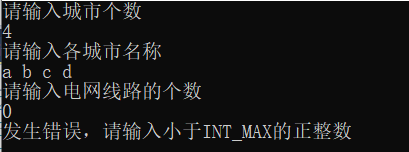
**实验结果：**



4.2.2 添加边数为0

**测试用例：**添加边数为0

**预期结果：**给出错误提示，程序运行正常不崩溃。



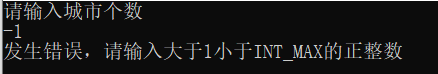
4.3 错误测试

4.3.1 顶点数错误

**测试用例：**新建-1个顶点

**预期结果：**程序给出提示信息，程序正常运行不崩溃，要求重新输入

**实验结果：**

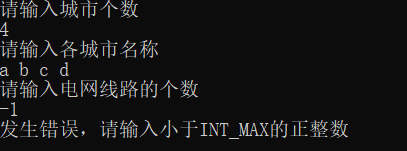


4.3.2 边数错误

**测试用例：**添加-1条边

**预期结果：**程序给出提示信息，程序正常运行不崩溃，要求重新输入。

**实验结果：**



4.3.3 顶点重名

**测试用例**：

4

a b c c

d

6

a b 8

b c 7

c d 5

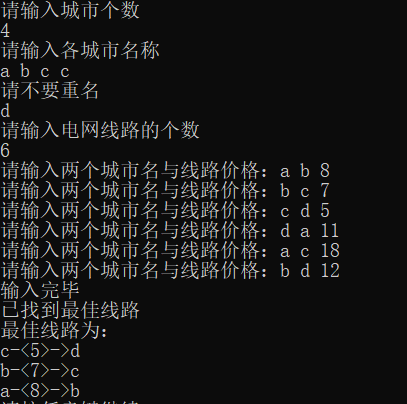
d a 11

a c 18

b d 12

**预期结果**：提示重名并要求重新输入

**实验结果：**



4.3.4 图不连通

**测试用例**：

5

a b c d e

5

a b 8

b c 7

a c 6

c d 5

d a 3

**预期结果**：提示图不连通

**实验结果：**

