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： 刘畅

学 号： 2054164

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](#_Toc89372204)

[1.1 背景分析 1](#_Toc89372205)

[1.2 功能分析 1](#_Toc89372206)

[2 设计 1](#_Toc89372207)

[2.1 使用邻接表实现的图模板类 1](#_Toc89372208)

[2.1.1 整体框架 1](#_Toc89372209)

[2.1.2实现方法 1](#_Toc89372210)

[2.1.3 成员函数的实现 2](#_Toc89372211)

[2.1.3.1 插入顶点 2](#_Toc89372212)

[2.1.3.2插入边 2](#_Toc89372213)

[2.1.3.3删去顶点 2](#_Toc89372214)

[2.1.3.4删去边 2](#_Toc89372215)

[2.1.3.5 实现细节 2](#_Toc89372216)

[2.1.2 部分代码 2](#_Toc89372217)

[2.2 最小堆模板类（MinHeap） 3](#_Toc89372218)

[2.2.1 整体框架 3](#_Toc89372219)

[2.2.2 成员函数的实现 3](#_Toc89372220)

[2.2.2.1 插入元素及向上调整 3](#_Toc89372221)

[2.2.2.2 删除并返回堆顶元素及向下调整 4](#_Toc89372222)

[2.2.3 部分代码 4](#_Toc89372223)

[2.3 优化的并查集模板类 4](#_Toc89372224)

[2.3.1 整体框架 4](#_Toc89372225)

[2.3.2 成员函数的实现 4](#_Toc89372226)

[2.3.2.1 寻找祖先 4](#_Toc89372227)

[2.3.2.2 压缩路径寻找祖先 5](#_Toc89372228)

[2.3.2.3 合并两个结点所在集合 5](#_Toc89372229)

[2.3.3 部分代码 5](#_Toc89372230)

[3 实现 5](#_Toc89372231)

[3.1 插入功能的实现 5](#_Toc89372232)

[3.1.1 插入功能流程图 5](#_Toc89372233)

[3.1.2 插入功能核心代码 6](#_Toc89372234)

[3.1.3 插入功能截屏示例 6](#_Toc89372235)

[3.2 删除功能的实现 8](#_Toc89372236)

[3.2.1 删除功能流程图 8](#_Toc89372237)

[3.2.2 删除功能核心代码 8](#_Toc89372238)

[3.2.3 删除功能截屏示例 9](#_Toc89372239)

[3.3 查找功能的实现 11](#_Toc89372240)

[3.3.1 查找功能流程图 11](#_Toc89372241)

[3.3.2 查找功能核心代码 11](#_Toc89372242)

[3.3.3 查找功能截图示例 12](#_Toc89372243)

[3.4 修改功能的实现 13](#_Toc89372244)

[3.4.1 修改功能流程图 13](#_Toc89372245)

[3.4.2 修改功能核心代码 13](#_Toc89372246)

[3.4.3 修改功能截屏示例 14](#_Toc89372247)

[3.5 统计功能的实现 15](#_Toc89372248)

[3.5.1 统计功能流程图 15](#_Toc89372249)

[3.5.2 统计功能核心代码 15](#_Toc89372250)

[3.5.3 统计功能截屏示例 16](#_Toc89372251)

[3.6 总体系统的实现 16](#_Toc89372252)

[3.6.1 总体系统流程图 16](#_Toc89372253)

[3.6.2 总体系统核心代码 17](#_Toc89372254)

[3.6.3 总体系统截屏示例 17](#_Toc89372255)

[4 测试 19](#_Toc89372256)

[4.1 功能测试 19](#_Toc89372257)

[4.1.1 插入功能测试 19](#_Toc89372258)

[4.1.2 删除功能测试 19](#_Toc89372259)

[4.1.3 查找功能测试 20](#_Toc89372260)

[4.1.4 修改功能测试 20](#_Toc89372261)

[4.1.5 统计功能测试 21](#_Toc89372262)

[4.2 边界测试 21](#_Toc89372263)

[4.2.1 初始化无输入数据 21](#_Toc89372264)

[4.2.2 删除头结点 22](#_Toc89372265)

[4.2.3 删除后链表为空 22](#_Toc89372266)

[4.3 出错测试 23](#_Toc89372267)

[4.3.1 考生人数错误 23](#_Toc89372268)

[4.3.2 操作码错误 23](#_Toc89372269)

[4.3.3 插入位置不存在 23](#_Toc89372270)

[4.3.4 删除考号不存在 24](#_Toc89372271)

[4.3.5 查找考号不存在 24](#_Toc89372272)

[4.3.6 修改考号不存在 25](#_Toc89372273)

# 1 分析

## 背景分析

假设有个城市有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建设方案。

## 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。

存储结构可以采用图的方式，结点表示小区，结点之间的边权重表明这两个小区之间要连通所付出的经济代价。先题意即求该图的最小生成树。

# 2 设计

## 2.1 使用邻接表实现的图模板类

### 2.1.1 整体框架

如上功能分析，首先得设计一个图的模板类，结点的数据类型和边的权值类型待定，并且采用邻接表存储图，即template<class T, class E>class LinkedGraph{};，图的模板类中主要功能有获取某条边的权值、获取某个顶点的数据、插入某个顶点、插入某条边、删去某个顶点、删去某条边。（删点和删边操作可能在此题中没有必要，但为了模板类的完整性，给予实现）

对于图的结点类，首先必须包含该结点的数据，并且由于采用邻接表存储，需要指向一条边作为该结点引出的第一条边；对于图的边类，需要包含边的权值、该边指向的第二个结点以及下一条边的位置。

### 2.1.2实现方法

图中所有的结点，都有自己的一个编号，即若图中结点数为n，那么这n个结点分别编号0~n-1，我们采用顺序结构存储这n个结点以方便查找。对于第i个结点，我们可以使用getVertexValue(int i)获取其结点。同时又由于是顺序存储，当达到最大容量时扩展极为不便（当然也有方法可以实现），在本项目中便不加以实现。

边的存储采用邻接表的形式，即每个结点包含从该结点引出的第一条边，每条边包含该边的另一个结点以及下一条边的指针；若图中有n条边，则需要使用2n个边对象来存储，即每条边作为两个结点的出度被存储了两次。

### 2.1.3 成员函数的实现

### 2.1.3.1 插入顶点

只需要在类的结点数组中新建一个结点，并自增结点数目即可。

### 2.1.3.2插入边

在两个顶点的边链表表头插入即可，省去了遍历整个边链表的时间。

### 2.1.3.3删去顶点

### 2.1.3.4删去边

### 2.1.3.5 实现细节

值得指出的细节是，由于不知道图的结点数据和边权值是什么类型，也就不知道该类型的默认值，导致在判断顶点是否连通、顶点是否存在等问题上，当答案是false的返回值问题，因此在整个模板图类的构造函数中，要求传入两种数据类型的默认值，便于在找不到顶点或者边不连通时返回默认值，从而防止混淆。

在删除顶点时，可以传入的参数有顶点序号和顶点的data值，每个顶点的序号不存在重复的情况，而不同顶点的data值可能是同一个，但结合实际问题不同小区的名称不应相同，因此在插入结点函数中判断data值是否已经存在，以确保每一个结点的data值都不相同，从而保证删除时最多只需要删除一个结点。

### 2.1.2 部分代码

template<class T,class E>

class LinkedGraph {

public:

LinkedGraph(T dafaultValue\_T = 0,E defaultValue\_E=0, int maxVertices = DefaultSize);

~LinkedGraph();

E getVertexValue(int i)const; //获取某个结点的值

T getEdgeWeight(int v1, int v2)const; //获取某条边的权值

bool insertVertex(const E& vertex); //插入结点，数据为vertex

bool removeVertex(int v); //删除值为vertex的结点

bool removeVertex(const E& vertex); //删除第v个结点

bool insertEdge(int v1, int v2, const T& weight); //在v1和v2之间插入权值为weight的边，若边已存在则返回false

bool removeEdge(int v1, int v2); //删除v1和v2之间的边，若不存在这条边则返回false

int getFirstNeighbour(int v); //获取结点v引出的第一条边

int getNextNeighbour(int v, int w); //获取结点v引出的边链表中，w的下一条边

private:

Vertex<T,E>\* \_NodeTable; //各边链表的头结点

int getVertexPos(const E& vertex); //获取数据为vertex的结点序号

};

## 2.2 最小堆模板类（MinHeap）

在最小生成树的获取过程中，需要不断获取边权值最小的边，而使用堆进行最小值的弹出十分简便，因此采用模板堆存储边的权值。

## 2.2.1 整体框架

堆是一个半有序的树型结构，因此采用树的存储方法，这里使用顺序存储。将n个结点值存入0~n-1的数组中。由于堆是一个完全二叉树，因此结点序号和父子关系之间有良好的对应关系，即对于结点i，若i=0则为根结点，不然则其父节点为(i-1)/2，左子女为2\*i+1，右子女为2\*i+2。对于最小堆，每个结点的关键码均小于或等于它左右子女的关键码，因此堆顶元素即为最小值。

对于堆我们主要需要两个对外接口，即插入元素和删除并返回堆顶元素，为了维护堆的最小性，在每次删除和插入时都需要对其进行结点的调整，即siftUp()和siftDown()函数。

## 2.2.2 成员函数的实现

## 2.2.2.1 插入元素及向上调整

若堆满则返回false，不然将要插入的元素放在数组的最后，然后调用siftUp()向上调整堆。首先将新元素作为操作对象，将其与父结点比较，若大于或等于则停止，不然则将其与父结点交换，而后将交换后的新结点作为对象继续如上操作，直至其大于或等于父结点或成为堆顶元素为止。

## 2.2.2.2 删除并返回堆顶元素及向下调整

在该函数中传入一个引用类型的变量将其赋值为堆顶元素，从而实现堆顶元素的返回；而后将堆顶元素赋值为堆尾元素，再调用siftDown()向下调整堆。首先将堆顶元素作为操作对象，将其与较小的子女进行比较，若本身更大，交换两者，然后将交换后的新结点作为对象继续如上操作，直至结点本身较小或已经到达堆尾为止。

## 2.2.3 部分代码

bool Insert(const T& data); //插入元素data，插入成功返回true，失败返回false

bool RemoveMin(T& x); //若堆空，则返回false，不然则退回最小元素并返回true

void siftUp(); //从堆尾向上调整

void siftDown(); //从堆顶向下调整

## 2.3 压缩路径的并查集模板类

在最小生成树的获取过程中，需要不断判断选出的最小权值边对应的两个结点是否已经连通，而连通是一种等价关系，我们可以将所有连通的结点合并为一个集合，利用并查集判断一条边的两个结点是否在同一个结点中，从而决定该边是否需要加入最小生成树。

## 2.3.1 整体框架

利用树形结构，两个结点处于同一个集合，当且仅当它们拥有共同祖先。对于一个并查集，我们需要判断两个结点是否属于同一集合，并且如果两者不属于同一集合，则可以将其合并为同一集合。

## 2.3.2 成员函数的实现

## 2.3.2.1 寻找祖先

并查集的每个结点都有parent指针，当且仅当其为根结点时，parent指针等于0。因此只需不断向上获取其parent直至parent为零。

## 2.3.2.2 压缩路径寻找祖先

由于某个结点里根的距离可能很远，这样会增大寻找祖先时的时间开销，因此在每次寻找时都将路径上的结点直接作为祖先的第一层子女，即每次查找时都将树压缩为只有两层。

## 2.3.2.3 合并两个结点所在集合

先判断这两个结点的祖先是否为同一个，若为同一个则返回false，因为已经在同一个集合之中了；不然则将其中一个结点的祖先设置为另一个祖先的子女。

## 2.3.3 部分代码

int Find(int x); //搜索并返回包含x的树的根

int CollapsingFind(int x); //搜索并返回包含x的树的根，并压缩路径

bool Union(int x1,int x2); //合并结点x1与x2所在集合

# 2.4 最小生成树模板类

用于获取某个图的最小生成树

## 2.4.1 整体框架

最小生成树特有的边类MSTEdge，包含边的两个结点和边的权值；以及最小生成树类。

初始化最小生成树时使用某个图的结点数，以开辟最小生成树的边空间，一颗拥有n个结点的图，其最小生成树拥有n-1条边。

使用Kruskal算法获取最小生成树，参数为一个图。

打印最小生成树，每条边使用结点1数据、结点2数据和连通这两个结点的边的权值描述，打印n-1行。

## 2.4.2 Kruskal算法

首先判断树是否有被初始化，若有则maxSize不为-1，而后获取图的结点数n和边数m，构造大小为m的最小堆和大小为n的并查集。并将所有的边添加到最小堆中。（注意这个地方的边是以最小生成树MSTEdge特有的边形式表示的，而在最小堆中需要进行大小比较，因此要重载MSTEdge的大小比较运算符）这样以后，是Kruskal算法的核心：取出堆顶元素，即权值最小的边，判断该边的两个结点是否处于同一集合，若否，则将此边加入到最小生成树的边数组中，若是，则继续取出下一条权值最小的边。直至最小生成树的边数组拥有n-1条边。

## 2.4.3 部分代码

bool Kruskal(const LinkedGraph<T, E>& graph) {

if (maxSize == -1) { return false; }

int u, v, n = graph.getNumVertices(), m = graph.getNumEdges();

MinHeap<MSTEdge<T> >heap(m);

UFSets ufsets(n);

for (u = 0; u < n; u++) {

for (v = u + 1; v < n; v++) {

if (graph.getEdgeWeight(u, v) != graph.getDefaultValueOfEdge()) {

MSTEdge<T> edge(u, v, graph.getEdgeWeight(u, v));

//e.\_tail = u; e.\_head = v;

//e.\_weight = graph.getWeight(u, v);

heap.Insert(edge);

}

}

}

while (currentSize < maxSize) {

MSTEdge<T> addEdge;

heap.RemoveMin(addEdge);

u = ufsets.CollapsingFind(addEdge.\_tail);

v = ufsets.CollapsingFind(addEdge.\_head);

if (u != v) {

ufsets.Union(u, v);

this->\_edges[currentSize++] = addEdge;

}

}

return true;

}

### 3.1.2 插入功能核心代码

LNode \*temp = new LNode(studentID, studentName, sex, age, category);

LNode \*current = m\_first;

//寻找插入位置前一个节点

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

{

current = current->m\_next;

}

//进行插入操作

++m\_length;

temp->m\_next = current->m\_next;

current->m\_next = temp;

### 3.1.3 插入功能截屏示例

## 3.2 删除功能的实现

### 3.2.1 删除功能流程图

### 3.2.2 删除功能核心代码

bool LinkList::Delete()

{

LNode \*current = m\_first->m\_next;//要删除的节点

LNode \*prior = m\_first;//要删除节点的前一个节点

int flag = 0; //找到的标志

while ((flag == 0) && (current != nullptr))

{

if (current->m\_studentID == studentID)

{

flag = 1;

}

else

{

current = current->m\_next;

prior = prior->m\_next;

}

}

//进行删除操作

{

--m\_length;

cout << "你删除的考生的信息是："

<< current->m\_studentID << " "

<< current->m\_studentName << " "

<< current->m\_sex << " "

<< current->m\_age << " "

<< current->m\_category << " "

<< endl;

prior->m\_next = current->m\_next;

delete current;

}

}

### 3.2.3 删除功能截屏示例

## 3.3 查找功能的实现

### 3.3.1 查找功能流程图

### 3.3.2 查找功能核心代码

bool LinkList::Find() const

{

LNode \*current = m\_first->m\_next;//当前寻找的节点

int flag = 0; //找到的标志

while ((flag == 0) && (current != nullptr))

{

if (current->m\_studentID == studentID)

{

flag = 1;

}

else

{

current = current->m\_next;

}

}

}

### 3.3.3 查找功能截图示例

## 3.4 修改功能的实现

### 3.4.1 修改功能流程图

### 3.4.2 修改功能核心代码

bool LinkList::Change()

{

LNode \*current = m\_first->m\_next;//当前修改的节点

int flag = 0; //找到的标志

while ((flag == 0) && (current != nullptr))

{

if (current->m\_studentID == studentID)

{

flag = 1;

}

else

{

current = current->m\_next;

}

}

{

current->m\_studentID = studentID;

current->m\_studentName = studentName;

current->m\_sex = sex;

current->m\_age = age;

current->m\_category = category;

}

}

### 3.4.3 修改功能截屏示例

## 3.5 统计功能的实现

### 3.5.1 统计功能流程图

### 3.5.2 统计功能核心代码

while (current != nullptr)

{

if (current->m\_studentID != "")

{

cout << current->m\_studentID << " "

<< current->m\_studentName << " "

<< current->m\_sex << " "

<< current->m\_age << " "

<< current->m\_category << " "

<< endl;

}

current = current->m\_next;//移动到下一个节点

}

### 3.5.3 统计功能截屏示例

## 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

while (operateCode != '0')

{

switch (operateCode)

{

case '1': L.Insert(); break;

case '2': L.Delete(); break;

case '3': L.Find(); break;

case '4': L.Change(); break;

case '5': L.Display(); break;

//如果用户输入操作要求错误，提示用户重新输入

default: cout << "抱歉，您想要进行的操作不存在，请重新输入！" << endl; break;

}

cout << "请选择您要进行的操作" << endl;

cin >> operateCode;

}

### 3.6.3 总体系统截屏示例

# 4 测试

## 4.1 功能测试

### 4.1.1 插入功能测试

**测试用例**：3 stu3 男 22 网络工程师

**预期结果**：

1 stu1 男 20 软件开发师

2 stu2 女 21 软件测试员

3 stu3 男 22 网络工程师

**实验结果**

### 4.1.2 删除功能测试

**测试用例：**删除考号为4的考生

**预期结果：**

1 stu1 男 20 软件开发师

2 stu2 女 21 软件测试员

3 stu3 男 22 网络工程师

**实验结果：**

### 4.1.3 查找功能测试

**测试用例：**查找考号为2的考生

**预期结果：**

2 stu2 女 21 软件测试员

**实验结果：**

### 4.1.4 修改功能测试

**测试用例：**将考号1修改为性别女，年龄20，报考种类移动开发员。

**预期结果：**

1 stu1 女 20 移动开发员

**实验结果：**

### 4.1.5 统计功能测试

**测试用例：**统计当前数据

**预期结果：**

1 stu1 女 20 移动开发员

2 stu2 女 21 软件测试员

3 stu3 男 22 网络工程师

**实验结果：**

## 4.2 边界测试

### 4.2.1 初始化无输入数据

**测试用例：**初始无输入数据

**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 删除头结点

**测试用例：**删除头结点

**预期结果：**程序正常运行，不崩溃。

**实验结果：**

### 4.2.3 删除后链表为空

**测试用例：**删除前链表只有一个结点，删除后链表为空

**预期结果：**程序正常运行，不崩溃。

**实验结果：**

## 4.3 出错测试

### 4.3.1 考生人数错误

**测试用例：**输入考生人数为负数

**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 插入位置不存在

**测试用例：**链表里有两条记录，向链表的第四个位置插入结点

**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：**

### 4.3.4 删除考号不存在

**测试用例：**要删除的考号不存在

**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：**

### 4.3.5 查找考号不存在

**测试用例：**要查找的考号不存在

**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：**

### 4.3.6 修改考号不存在

**测试用例：**要修改的考号不存在

**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：**

# 