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](#_Toc91257802)

[1.1 背景分析 1](#_Toc91257803)

[1.2 功能分析 1](#_Toc91257804)

[2 设计 1](#_Toc91257805)

[2.1 使用邻接表实现的图模板类 1](#_Toc91257806)

[2.1.1 整体框架 1](#_Toc91257807)

[2.1.2实现方法 1](#_Toc91257808)

[2.1.3 成员函数的实现 2](#_Toc91257809)

[2.1.3.1 插入顶点 2](#_Toc91257810)

[2.1.3.2插入边 2](#_Toc91257811)

[2.1.3.3删去结点 2](#_Toc91257812)

[2.1.3.4删去边 2](#_Toc91257813)

[2.1.3.5 实现细节 2](#_Toc91257814)

[2.1.4 部分代码 3](#_Toc91257815)

[2.2 最小堆模板类（MinHeap） 3](#_Toc91257816)

[2.2.1 整体框架 3](#_Toc91257817)

[2.2.2 成员函数的实现 4](#_Toc91257818)

[2.2.2.1 插入元素及向上调整 4](#_Toc91257819)

[2.2.2.2 删除并返回堆顶元素及向下调整 4](#_Toc91257820)

[2.2.3 部分代码 4](#_Toc91257821)

[2.3 压缩路径的并查集模板类 4](#_Toc91257822)

[2.3.1 整体框架 4](#_Toc91257823)

[2.3.2 成员函数的实现 5](#_Toc91257824)

[2.3.2.1 寻找祖先 5](#_Toc91257825)

[2.3.2.2 压缩路径寻找祖先 5](#_Toc91257826)

[2.3.2.3 合并两个结点所在集合 5](#_Toc91257827)

[2.3.3 部分代码 5](#_Toc91257828)

[2.4 最小生成树模板类 5](#_Toc91257829)

[2.4.1 整体框架 5](#_Toc91257830)

[2.4.2 Kruskal算法 6](#_Toc91257831)

[2.4.3 部分代码 6](#_Toc91257832)

[3 测试 7](#_Toc91257833)

[3.1 功能测试 7](#_Toc91257834)

[3.1.1 手动创建图的结点与边 7](#_Toc91257835)

[3.1.2 构造最小生成树 7](#_Toc91257836)

[3.1.3 再次添加结点和边 8](#_Toc91257837)

[3.1.4 删除边 8](#_Toc91257838)

[3.1.5 删除结点 8](#_Toc91257839)

[3.2 出错测试 9](#_Toc91257840)

[3.2.1 结点已存在 9](#_Toc91257841)

[3.2.2 边已存在 9](#_Toc91257842)

[3.2.3 删除边、结点不存在 9](#_Toc91257843)

[3.2.4 最小生成树不存在 10](#_Toc91257844)

# 1 分析

## 1.1 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删去结点

删去结点，则需要删除所有与结点关联的边及结点本身。首先遍历该结点相连的所有边，对于每一条边p，k为这条边的另一个结点，因此k的邻接表中也存在这条边，先找出这条边并将其删去，而后删去“被删结点”邻接表中的边p，依次遍历所有的边。最后删除结点本身。

为了使得结点表连续，将最后一个结点填入被删结点的位置，然后将所有与此结点相关的邻接表中的结点序号进行更新。

### 2.1.3.4删去边

删去边，即删除对应的两个结点的邻接表中的这条边即可。

### 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。

### 2.1.4 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));

heap.Insert(edge);

}

}

}

while (currentSize < maxSize) {

MSTEdge<T> addEdge;

heap.RemoveMin(addEdge);

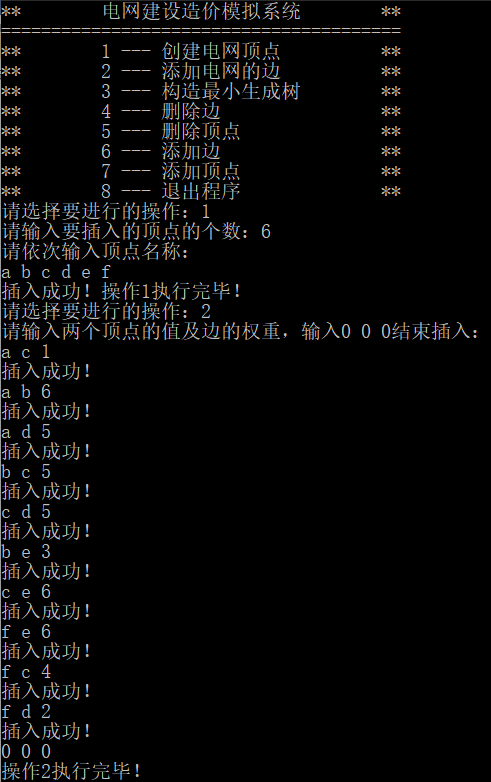
u = ufsets.CollapsingFind(addEdge.\_tail);

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

if (u != v) {

ufsets.Union(u, v);

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

 }

}

return true;

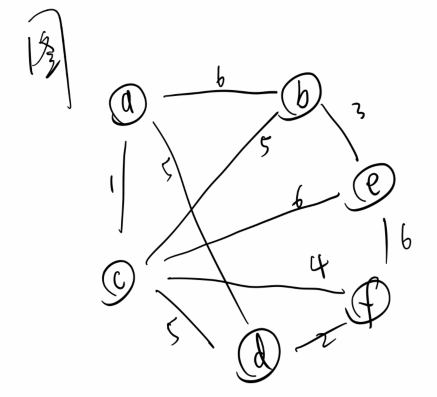
}

# 3 测试

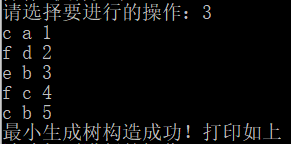
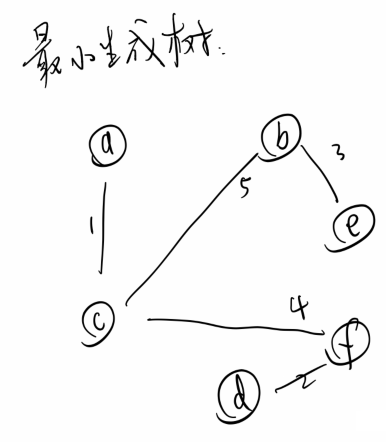
## 3.1 功能测试

### 3.1.1 手动创建图的结点与边

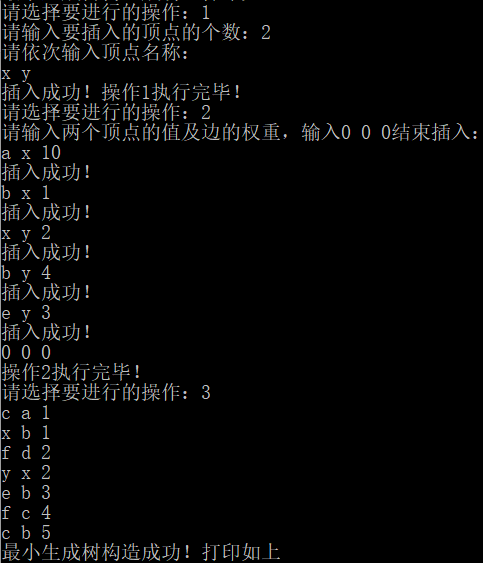
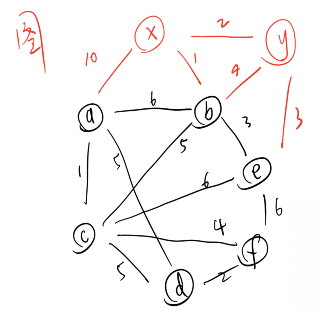
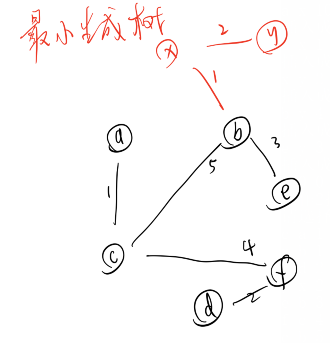
图的可视化：



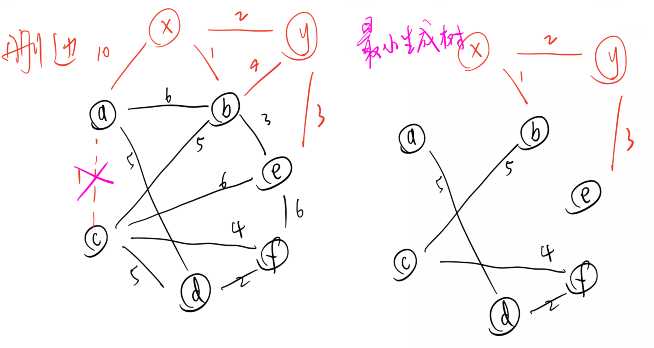
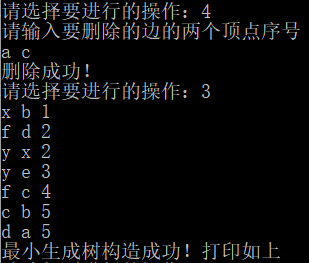
### 3.1.2 构造最小生成树



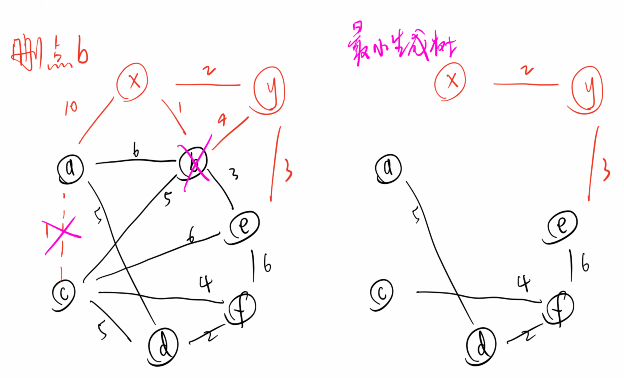
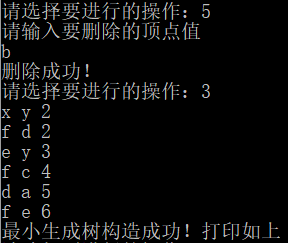
### 3.1.3 再次添加结点和边



### 3.1.4 删除边



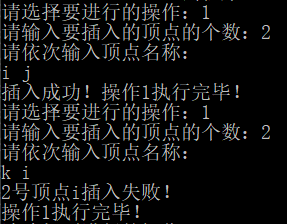
### 3.1.5 删除结点



## 3.2 出错测试

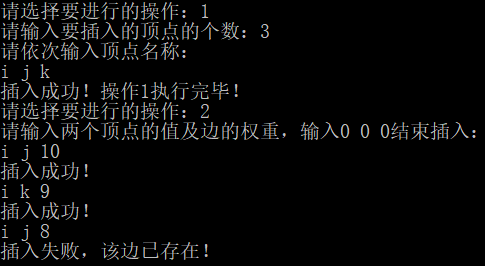
### 3.2.1 结点已存在

连续插入两个相同的结点，程序给出插入失败提示，并表明结点序号。



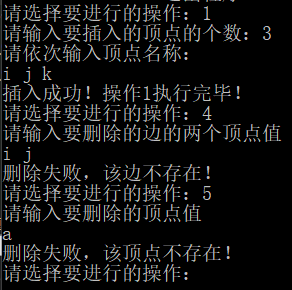
### 3.2.2 边已存在

连续插入相同结点之间的边，程序给出插入失败提示。



### 3.2.3 删除边、结点不存在

若要进行删除的边或结点不存在，程序给出错误提示。



### 3.2.4 最小生成树不存在

若图不连通，则最小生成树不存在，程序给出提示。

