**《离散数学》求最小生成树项目文档**

1. **题目简介**

如下图所示的赋权图表示某七个城市，预先计算出它们之间的一些直接通信道路造价（单位：万元），试给出一个设计方案，使得各城市之间既能够保持通信，又使得总造价最小，并计算其最小值。

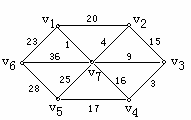


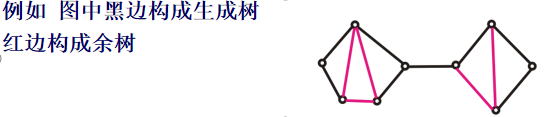
图1 七个城市赋权图

1. **解题思路**

**2.1实验原理**

**2.1.1 生成树的定义**

设G是无向连通图, 若G的生成子图T是一棵树, 则称T是G的生成树. G在T中的边称作T的树枝,不在T中的边，称作T的弦. T的所有弦的集合的导出子图称作T的余树；[注意：余树一般不是树]



**2.1.2 生成树的存在性**

任何无向连通图都有生成树.

证：用破圈法. 若图中无圈, 则图本身就是自己的生成树. 否则删去圈上的任一条边, 不破坏连通性, 重复进行直到无圈为止, 得到图的一棵生成树

推论1 设n阶无向连通图有m条边, 则m>=n-1.

推论2 设n阶无向连通图有m条边, 则它的生成树的余树有m-n+1条边.

**2.1.3 最小生成树的定义**

图G的每一条边e附加一个实数w(e), 称作边e的权. 图G连同附加在边上的权称作带权图, 记作G=<V,E,W>. 设H是G的子图, H所有边的权的和称作H的权, 记作W(H).最小生成树的定义即为带权图权最小的生成树.

使用不同的遍历图的方法，可以得到不同的生成树；从不同的顶点出发，也可能得到不同的生成树。按照生成树的定义，n个顶点的连通网络的生成树有n个顶点、n-1条边；

构造最小生成树的准则：

1、必须只使用该网络中的边来构造最小生成树；

2、必须使用且仅使用n-1条边来联络网络中的n个顶点；

3、不能使用产生回路的边；

**2.2 设计程序**

这是一个求图的最小生成树的问题，因此选择图作为抽象数据结构，并通过链表实现图的邻接表表示，且用到最小生成树类、并查集类；

1. **核心算法**

为了求解最小代价，使花费的总代价最小，这是数学中经典的求解最小(费用)生成树的算法。我们采用Kruskal算法，此算法的核心思想是：

设有一个有n个顶点的连通网络N={V,E},最初先构造一个只有n个顶点，没有边的非连通图T={V,Φ}，图中每个顶点自成一个连通分量。当在E中选到一条具有最小权值的边时,若该边的两个顶点落在不同的连通分量上，则将此边加入到T中;若两个顶点已经在同一个连通分量中，说明顶点之间已经存在边，所以将此边舍去，重新选择一条权值最小的边。如此重复下去，直到所有顶点在同一个连通分量上为止。

算法的框架我们利用最小堆(MinHeap)和并查集(DisjointSets)来实现克鲁斯卡尔算法。

// Kruscal算法

template <class T, class E>

bool MinSpanTree<T, E>::Kruscal(Graphlnk<T, E>& G) {

MSTEdgeNode<T, E> ed; // 边结点辅助单元

int u, v, count;

int n = G.numberOfVertices(); // 图的顶点数

E weight; // 权值

priority\_queue<MSTEdgeNode<T, E>> H; // 最小堆，关键码类型为E

UnionFindSets F(n); // 并查集

// 1.把边全部存到最小堆中

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

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

weight = G.getWeight(u, v);

if (weight > 0 && weight < G.maxWeight) { // 两个顶点存在边

ed.tail = u;

ed.head = v;

ed.weight = weight;

H.push(ed); // 插入最小堆

}

}

}

count = 1; // 最小生成树加入边数计数

while (count < n && H.empty() == false) { // n个顶点，反复执行，取n-1条边;并且最小堆不为空，即还有边时

ed = H.top(); // 从最小堆中退出具最小权值的边ed

H.pop();

u = F.Find(ed.tail); // 取两顶点所在集合的根u和v

v = F.Find(ed.head);

if (u != v) { // 不是同一集合，说明不连通

F.Union(u, v); // 合并，连通它们

Insert(ed); // 该边存入最小生成树

count++;

}

}

if (count < n) {

cout << "不是最小生成树" << endl;

return false;

}

return true; // 是最小生成树

}

1. **所用数据结构**

**4.1 Edge**

图中“边”的类定义，在邻接表表示的存储结构中，是边链表中的边结点。

数据域：目标顶点、边的权值。

指针域：指向下一个边结点的指针。

**4.2 Vertex**

图中“顶点”的类定义，在邻接表表示的存储结构中，是每条边链表的头结点。

数据域：当前顶点的名字。

指针域：指向第一个边结点的指针。

**4.3 Graphlnk**

邻接表实现的图类，

成员数据：一个顶点表（表中元素是各边链表的头结点）

**4.4 MSTEdgeNode**

最小生成树的边结点

记录两个顶点和边的权值

实现了对运算符的重载

**4.5 MinSpanTree**

最小生成树的类定义，

保存边的数量和树中的边。

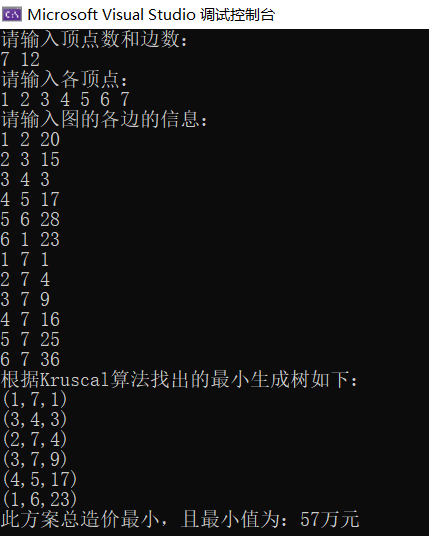
**4.6 UFSets、Tree、SeqQueue、MinHeap**

使用Kruskal算法生成最小生成树的过程中需要使用并查集和最小堆配合实现。

通过DFS建立生成森林的过程中需要用到Tree

使用BFS求所有连通分量的过程中需要用到SeqQueue

1. **实验结果**



1. **心得体会**

本次项目涉及知识点较多，且和数据结构相关，项目难度较大。直接把数据结构课程设计项目中用到的并查集、最小生成树以及用邻接表表示的图类搬运过来，根据网上的资料总结得出Kruscal算法，最终实现；在实现的过程中，因为涉及的类较多，变量也多，很容易在一些小细节上出现错误，这就要求我们在编程中要考虑周到，思路清晰，了解整个运算过程再下手。