# 1、解题思路

1.1、需求分析：

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。

1.2、实验原理：  
Kruskal算法利用了MST性质:假设N=(V，{E})是一个连通网，U是顶点集V的一个非空子集。若(u,v)是一条具有最小权值（代价)的边，其中u∈U，v∈V-U，则必存在一棵包含边(u,v)的最小生成树。

MST证明：

可以用反证法证明之。假设网N的任何一棵最小生成树都不包含(u,v)。设T是连通网上的一棵最小生成树，当将边(u,v)加入到T中时，由生成树的定义，T中必存在一条包含(u,z)的回路。另一方面，由于T是生成树，则在T上必存在另一条边(u',v')，其中u'∈U，v'∈V-U，且u和u'之间，和v'之间均有路径相通。删去边(u',v')，便可消除上述回路，同时得到另一棵生成树T’。因为(u,v)的代价不高于(u',v')，则T;的代价亦不高于T，T’是包含(u,v)的一棵最小生成树。由此和假设矛盾。

## 1.3、设计思路：

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：

(1)假设该图G是不连通的，对该图的边以权值非降序重新排列；

(2)对于排序表中的每条边，如果现在把它放入最小生成树T不会形成回路的话，则把它加入到T中；否则丢弃；

(3)输出最小生成树T的结果，得到想要的答案。

# 2、数据结构：

该实验使用的Kruskal算法基于图这种逻辑结构。

而在存储结构上，使用十字链表。

十字链表(Orthogonal List)是有向图的一种链式存储结构。可以看成是将有向图的邻接表和逆邻接表结合起来得到的一种链表。在十字链表中,对应于有向图中每一条弧有一个结点，对应于每个顶点也有一个结点。

代码：

struct EBox

{

int ivex, jvex;

int len;

};

# 3、核心算法：

利用并查集提高效率。

并查集的作用是维护不相交集合，可以高效地实现以下两种操作：

* + 查询元素与元素是否在同一集合（查）
  + 合并元素与元素所在的集合（并）

并查集的逻辑结构：树形结构。

可以采取的存储结构：静态数组。

具体实现上，使用parent[]，储存每一个点的父辈。初始时，将所有节点的parent设置为其自身。

查找算法：查找每一个点的祖先

* 若这个节点的parent是其自身，则该节点就是祖先，这是平凡情况
* 若这个节点的parent不是自身，则可以继续递归地查找parent的祖先，因为这个节点和其parent有相同的祖先。
* 递归边界是平凡情况

归并算法：

将一个并查集的祖先设置为另一个并查集的祖先。

查找两个元素是否在同一个集合中：

查找它们是否有相同的祖先。

算法优化——路径压缩

在特殊情况下，并查集中的树可能会退化成一条长链。此时，若多次对链尾结点执行find操作，每次都会遍历整条链，导致效率低下。看起来这是一个很棘手的问题，但由于树的形态无关紧要，因此我们在“查找”的同时，将经过的结点直接连接到根上，实现路径压缩，可以大大提高算法的效率。

代码：

int\* parent;

int find(int x)

{

if (x == parent[x])return x;

return parent[x] = find(parent[x]);

}

void merge(int x, int y)

{

int m = find(x), n = find(y);

parent[m] = find(n);

}

# 4、程序测试：

## a、基本测试：

1、测试输入：

7 12

1 2 20

2 3 15

3 4 3

4 5 17

5 6 28

6 1 23

1 7 1

2 7 4

3 7 9

4 7 16

5 7 25

6 7 36

实验预期：

1--1--7

3--3--4

2--4--7

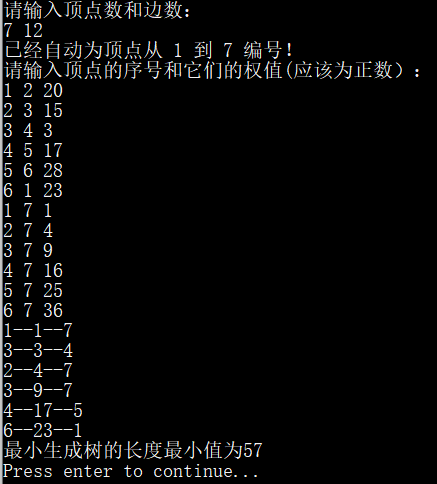
3--9--7

4--17--5

6--23--1

最小生成树的长度最小值为57

实验结果：



## b、出错测试：

1、输入不连通图

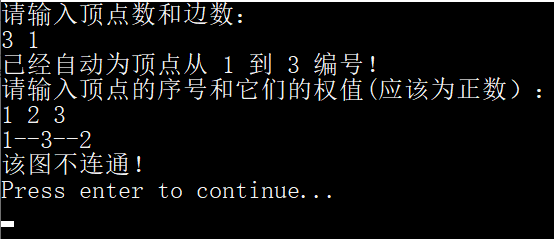
测试输入：

3 1

1 2 3

预期：错误提示

实验结果：



2、输入错误数据

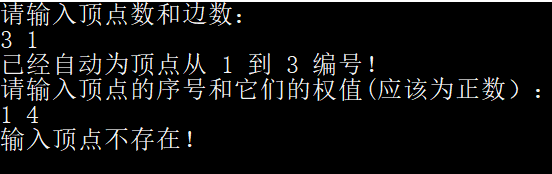
测试输入：

3 1

1 4

预期：错误提示

实验结果：



3、输入非法字符

测试输入：a

预期：错误提示

实验结果：

