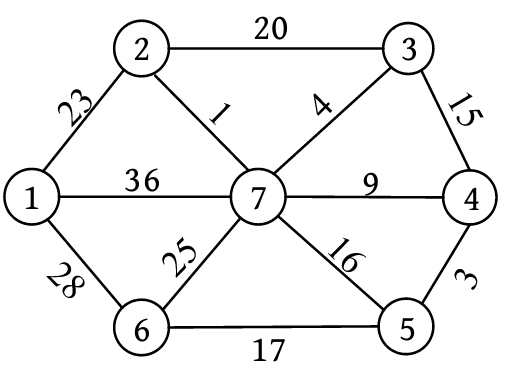
**最小生成树**

校园网是为学校师生提供资源共享、信息交流和协同工作的计算机网络。如果一所学校包括多个专业学科及部门，则也可以形成多个局域网络，并通过有线或无线方式连接起来。原来的网络系统只局限于以学院、图书馆为单位的局域网，不能完成集中管理及对各种资源的共享，个别院校还远离大学本部，这些情况都严重阻碍了该校的网络化进程。现在需要设计网络电缆布线，将各个单位连通起来，如何设计才能使布线费用最少呢？

可以用无向连通图G=(V,E)表示通信网络，V表示节点集，E表示边集。把各个单位都抽象为图中的节点，把单位之间的通信网络抽象为节点与节点之间的边，边的权值表示布线费用。如果两个节点没有连线，则代表在这两个单位之间不能布线，费用为无穷大，如下图所示。



**那么如何设计网络电缆布线，将各个单位连通起来，使布线费用最少呢？**对于有n个节点的连通图，只需n-1条边就可以使这个图连通，在n-1条边中要想保证图连通，就必须不包含回路，所以**只需找出n-1条权值最小且无回路的边即可**。需要说明以下几个概念。

**• 子图：**从原图中选中一些由节点和边组成的图，称之为原图的子图。

**• 生成子图：**选中一些由边和所有节点组成的图，称之为原图的生成子图。

**• 生成树：**如果生成的子图恰好是一棵树，则称之为生成树。

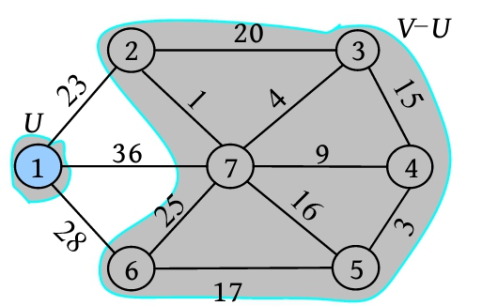
**• 最小生成树：**权值之和最小的生成树，称之为最小生成树。

**求解最小生成树算法有两种：Prim算法和Kruskal算法。**

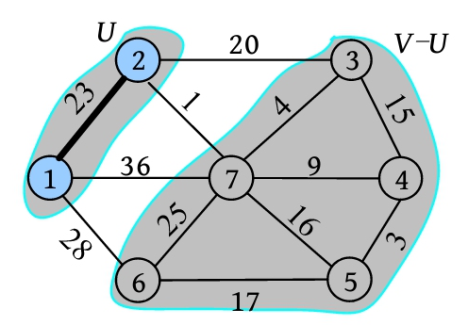
**一、　Prim算法**

找出n-1条权值最小的边很容易，那么怎么保证无回路呢？如果在一个图中通过深度搜索或广度搜索判断有没有回路，则工作繁重。有一种很好的办法——集合避圈法。在生成树的过程中，我们把已经在生成树中的节点看作一个集合，把剩下的节点看作另一个集合，从连接两个集合的边中选择一条权值最小的边即可。

首先任选一个节点，例如节点1，把它放在集合U中，U={1}，那么剩下的节点即V-U={2,3,4,5,6,7}，集合V是图的所有节点集合，如下图所示。



现在只需看看在连接两个集合（V和V-U）的边中，哪一条边的权值最小，把权值最小的边关联的节点加入集合U中。从上图可以看出，在连接两个集合的3条边中，1-2的边的权值最小，选中它，把节点2加入集合U中，U={1,2}，V-U={3,4,5,6,7}，如下图所示。

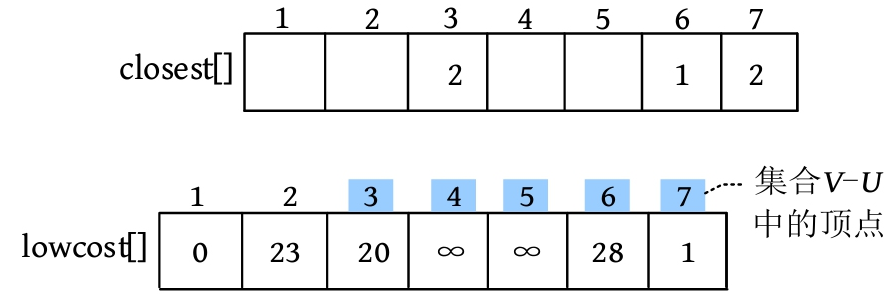


再从连接两个集合（V和V−U）的边中选择一条权值最小的边。从上图可以看出，在连接两个集合的4条边中，节点2到节点7的边的权值最小，选中这条边，把节点7加入集合U={1,2,7}中，V−U={3,4,5,6}。

如此下去，直到U=V结束，选中的边和所有的节点组成的图就是最小生成树。这就是**Prim算法，1957年由Robert C.Prim发现**。那么如何用算法来实现呢？

直观地看图，很容易找出集合U到集合V−U的边中哪条边的权值是最小的，但是**在程序中如果穷举这些边，再找最小值，则时间复杂度太高，该怎么办呢？**可以通过设置两个数组巧妙地解决这个问题，closest[j]表示集合V−U中的节点j到集合U中的最邻近点，lowcost[j]表示集合V−U中的节点j到集合U中的最邻近点的边值，即边(j,closest[j])的权值。

例如在上图中，节点7到集合U中的最邻近点是2，closest[7]=2。节点7到最邻近点2的边值为1，即边(2,7)的权值，记为lowcost[7]=1，如下图所示。



所以只需在集合V−U中找到lowcost[]值最小的节点即可。

**1. 算法步骤**

（1）初始化。令集合U={u0}，u0∈V，并初始化数组closest[]、lowcost[]和s[]。

（2）在集合V−U中找lowcost值最小的节点t，即lowcost[t]=min{lowcost[j]|j∈V−U}，满足该公式的节点t就是集合V−U中连接集合U的最邻近点。

（3）将节点t加入集合U中。

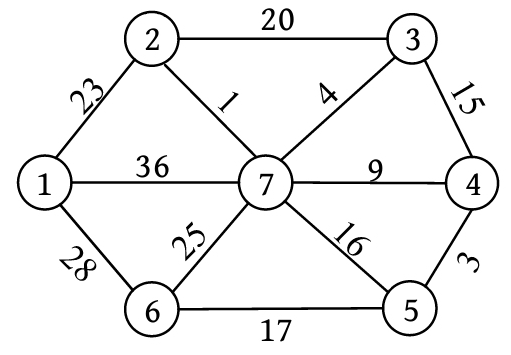
（4）如果集合V−U为空，则算法结束，否则转向步骤5。

（5）对集合V−U中的所有节点j都更新其lowcost[]和closest[]。更新if(C[t][j]<lowcost [j]){ lowcost[j]=C[t] [j]; closest[j]=t;}，转向步骤2。

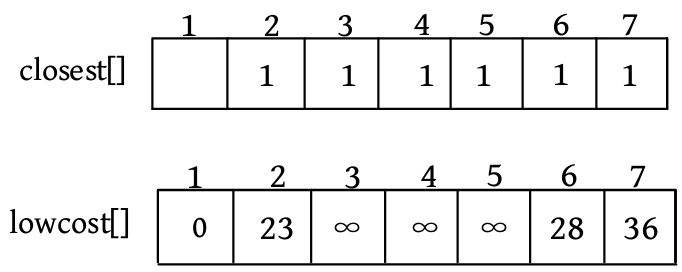
按照上述步骤，最终可以得到一棵权值之和最小的生成树。

**2. 图解**

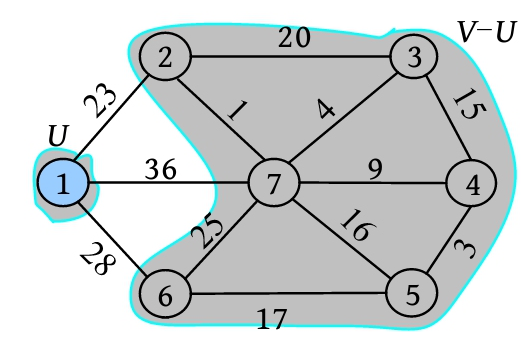
图G（G=(V, E)）是一个无向连通带权图，如下图所示。



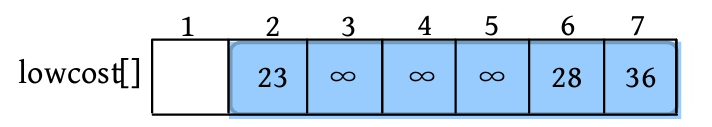
（1）初始化。假设u0=1，令集合U={1}，集合V−U={2,3,4,5,6,7}，TE={}，s[1]=true，初始化数组closest[]：除了节点1，其余节点均为1，表示集合V−U中的节点到集合U的最邻近点均为1。lowcost[]：节点1到集合V−U中节点的边值。closest[]和lowcost[]如下图所示。



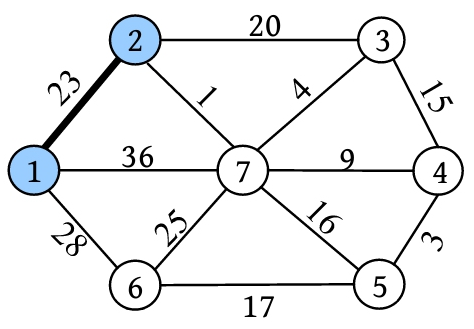
初始化后如下图所示。



（2）找lowcost最小的节点。在集合V−U={2,3,4,5,6,7}中，依照贪心策略寻找集合V−U中lowcost最小的节点t。找到的最小值为23，对应的节点t=2，如下图所示。



选中的边和节点如下图所示。



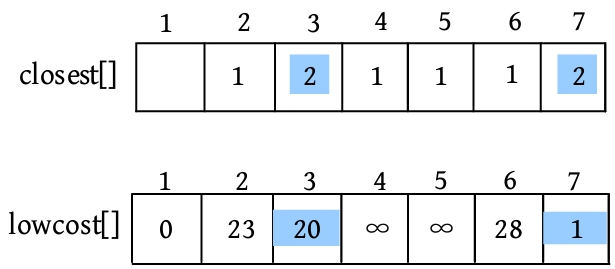
（3）加入集合U中。将节点t加入集合U中，U={1,2}，同时更新V−U={3,4,5,6,7}。

（4）更新。对t在集合V−U中的每一个邻接点j，都可以借助t更新。节点2的邻接点是节点3和节点7：

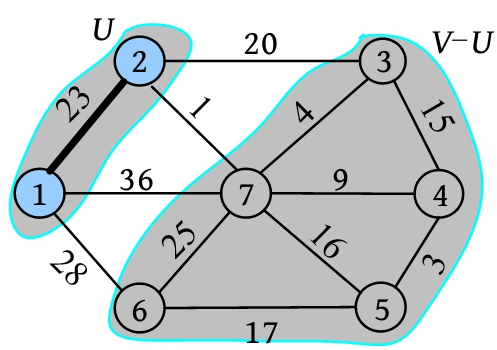
• C[2][3]=20<lowcost[3]=∞，更新最邻近距离lowcost[3]=20，最邻近点closest[3]=2；

• C[2][7]=1<lowcost[7]=36，更新最邻近距离lowcost[7]=1，最邻近点closest[7]=2。

更新后的closest[]和lowcost[]数组如下图所示。

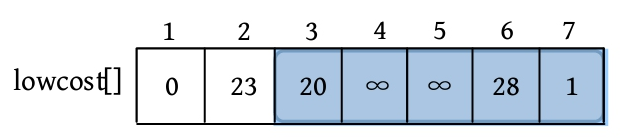


更新后的集合如下图所示。

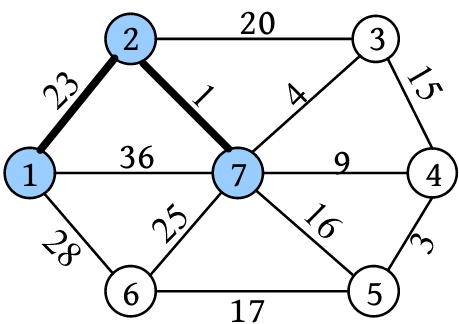


closest[j]和lowcost[j]分别表示集合V−U中节点j到集合U的最邻近节点和最邻近距离。节点3到集合U的最邻近点为2，最邻近距离为20；节点4、5到集合U的最邻近点仍为初始化状态1，最邻近距离为∞；节点6到集合U的最邻近点为1，最邻近距离为28；节点7到集合U的最邻近点为2，最邻近距离为1。

（5）找lowcost最小的节点。在集合V−U={3,4,5,6,7}中，依照贪心策略寻找集合V−U中lowcost最小的节点t，找到的最小值为1，对应的节点t=7，如下图所示。



选中的边和节点如下图所示：



（6）加入集合U中。将节点t加入集合U中，U={1,2,7}，同时更新V−U={3,4,5,6}。

（7）更新。对t在集合V−U中的每一个邻接点j，都可以借t更新。节点7在集合V−U中的邻接点是节点3、4、5、6：

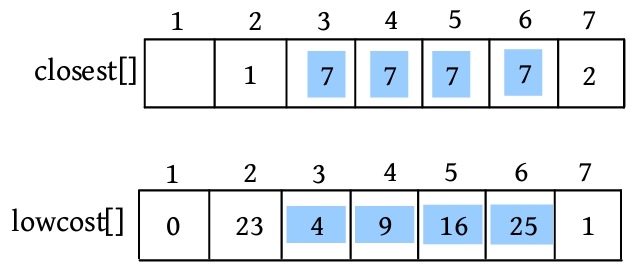
• C[7][3]=4<lowcost[3]=20，更新最邻近距离lowcost[3]=4，最邻近点closest[3]=7；

• C[7][4]=9<lowcost[4]=∞，更新最邻近距离lowcost[4]=9，最邻近点closest[4]=7；

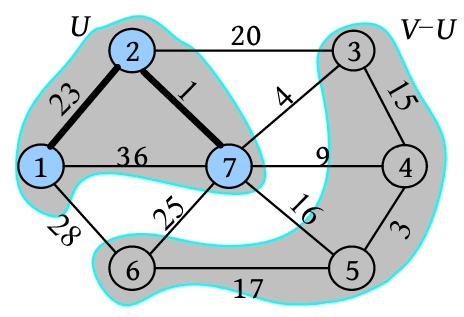
• C[7][5]=16<lowcost[5]=∞，更新最邻近距离lowcost[5]=16，最邻近点closest[5]=7；

• C[7][6]=25<lowcost[6]=28，更新最邻近距离lowcost[6]=25，最邻近点closest[6]=7。

更新后的closest[]和lowcost[]数组如下图所示：

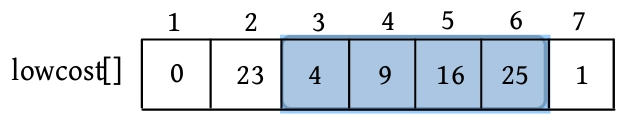


更新后的集合如下图所示：

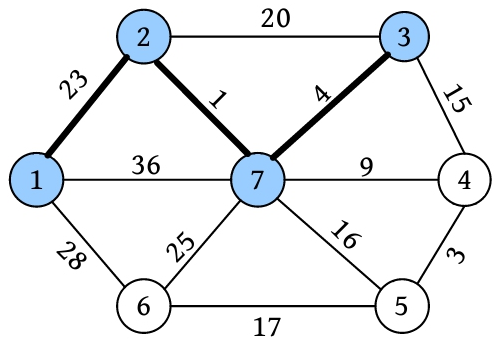


节点3到集合U的最邻近点为7，最邻近距离为4；节点4到集合U的最邻近点为7，最邻近距离为9；节点5到集合U的最邻近点为7，最邻近距离为16；节点6到集合U的最邻近点为7，最邻近距离为25。

（8）找lowcost最小的节点。在集合V−U={3,4,5,6}中，依照贪心策略寻找集合V−U中lowcost最小的节点t，找到的最小值为4，对应的节点t=3，如下图所示。



选中的边和节点如下图所示：



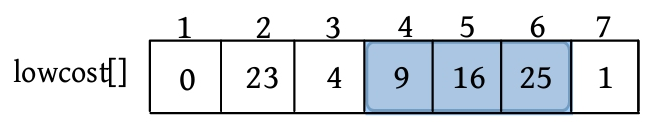
（9）加入集合U中。将节点t加入集合U中，U ={1,2,3,7}，同时更新V−U={4,5,6}。

（10）更新。对t在集合V−U中的每一个邻接点j，都可以借助t更新。节点3在集合V−U中的邻接点是节点4：C[3][4]=15>lowcost[4]=9，不更新；closest[j]和lowcost[j]数组不改变。

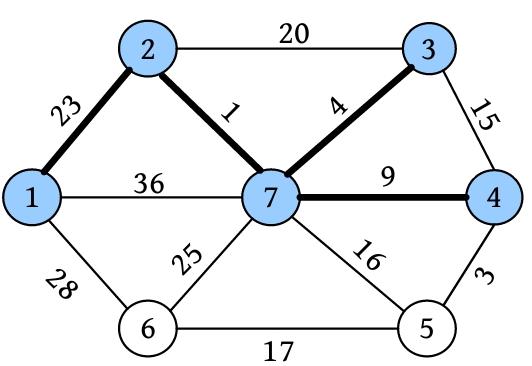
更新后的集合如下图所示。

节点4到集合U的最邻近点为7，最邻近距离为9；节点5到集合U的最邻近点为7，最邻近距离为16；节点6到集合U的最邻近点为7，最邻近距离为25。

（11）找lowcost最小的节点。在集合V−U={4,5,6}中，依照贪心策略寻找集合V−U中lowcost最小的节点t，找到的最小值为9，对应的节点t=4，如下图所示。

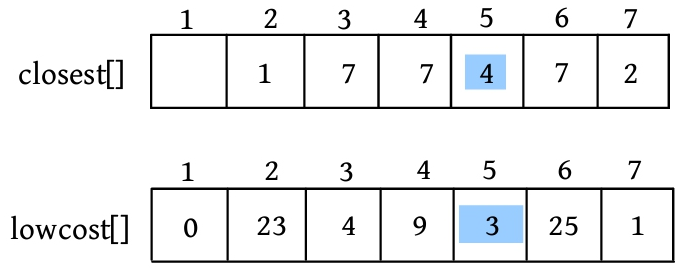


选中的边和节点如下图所示。

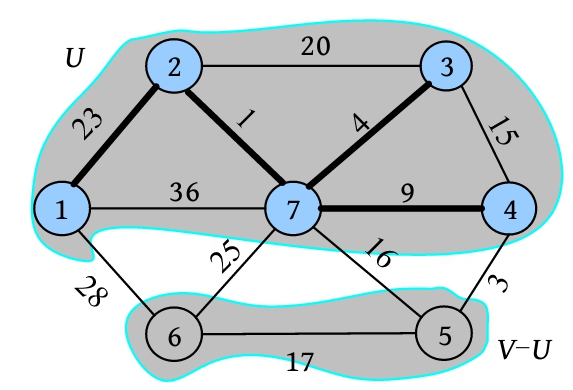


（12）加入集合U中。将节点t加入集合U中，U ={1,2,3,4,7}，同时更新V−U={5,6}。

（13）更新。对t在集合V−U中的每一个邻接点j，都可以借助t更新。节点4在集合V−U中的邻接点是节点5：C[4][5]=3<lowcost[5]=16，更新最邻近距离lowcost[5]=3，最邻近点closest[5]=4；更新后的closest[]和lowcost[]数组如下图所示。

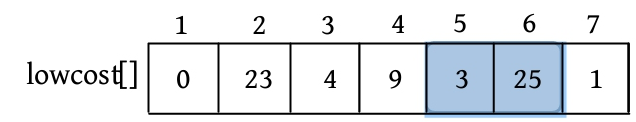


更新后的集合如下图所示。

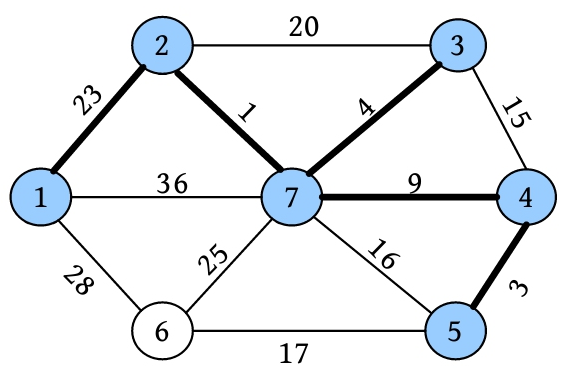


节点5到集合U的最邻近点为4，最邻近距离为3；节点6到集合U的最邻近点为7，最邻近距离为25。

（14）找最小。在集合V−U={5,6}中，依照贪心策略寻找集合V−U中lowcost最小的节点t，找到的最小值为3，对应的节点t=5，如下图所示。

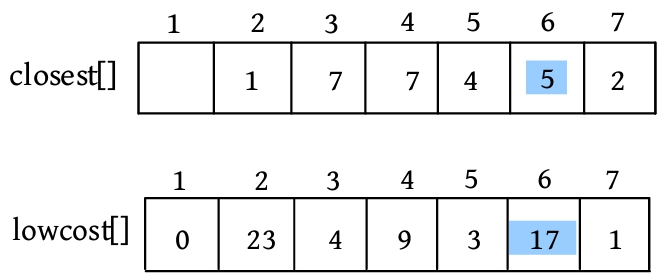


选中的边和节点如下图所示。

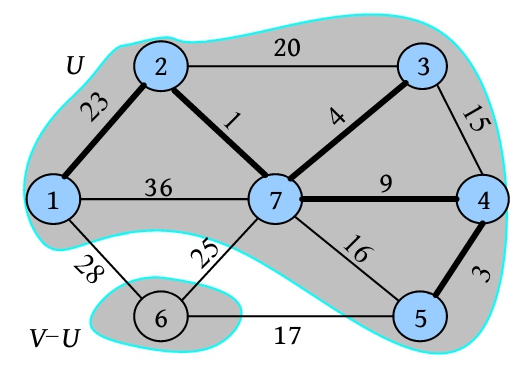


（15）加入集合U中。将节点t加入集合U中，U={1,2,3,4,5,7}，同时更新V−U={6}。

（16）更新。对节点t在集合V−U中的每一个邻接点j，都可以借助t更新。节点5在集合V−U中的邻接点是节点6：C[5][6]=17<lowcost[6]=25，更新最邻近距离lowcost[6]=17，最邻近点closest[6]=5；更新后的closest[]和lowcost[]数组如下图所示。

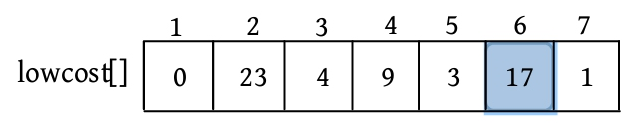


更新后的集合如下图所示。

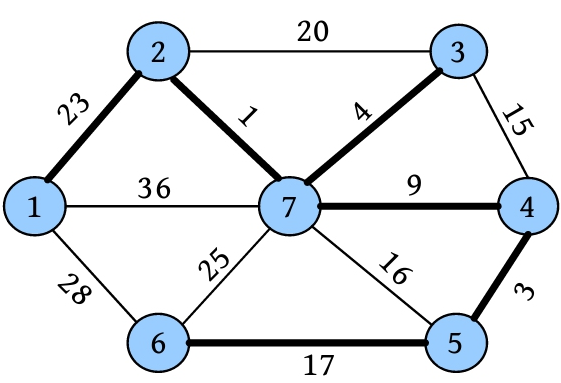


节点6到集合U的最邻近点为5，最邻近距离为17。

（17）找lowcost最小的节点。在集合V−U={6}中，依照贪心策略寻找集合V−U中lowcost最小的节点t，找到的最小值为17，对应的节点t=6。

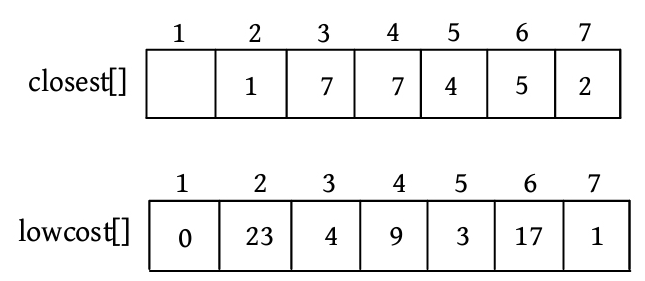


选中的边和节点如下图所示。

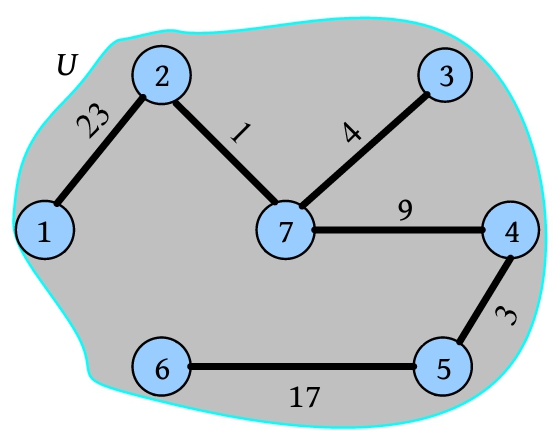


（18）加入集合U中。将节点t加入集合U中，U ={1,2,3,4,5,6,7}，同时更新V−U={}。

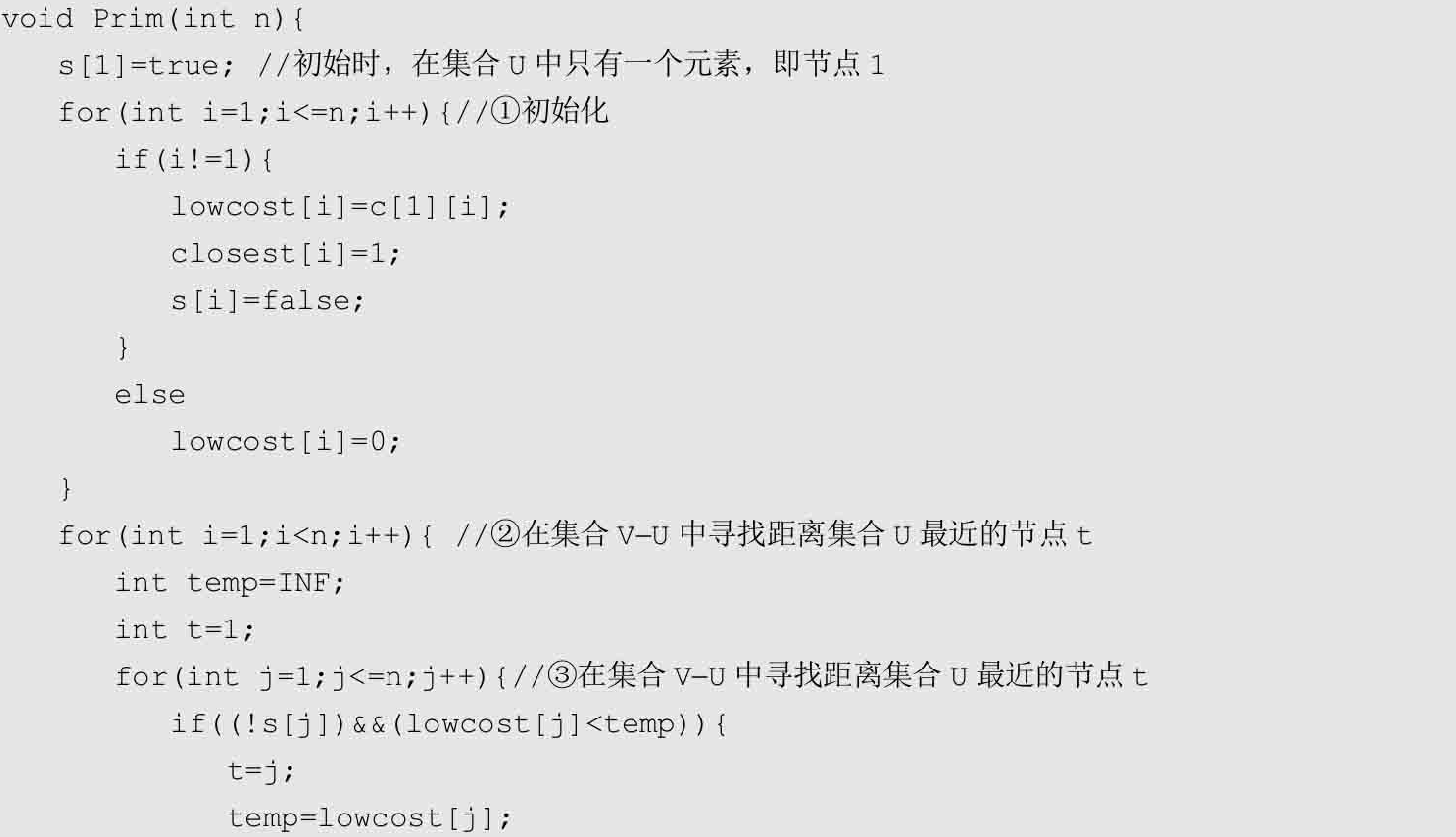
（19）更新。对t在集合V−U中的每一个邻接点j，都可以借t更新。节点6在集合V−U中无邻接点，因为V−U={}。更新后的closest[]和lowcost[]数组如下图所示。

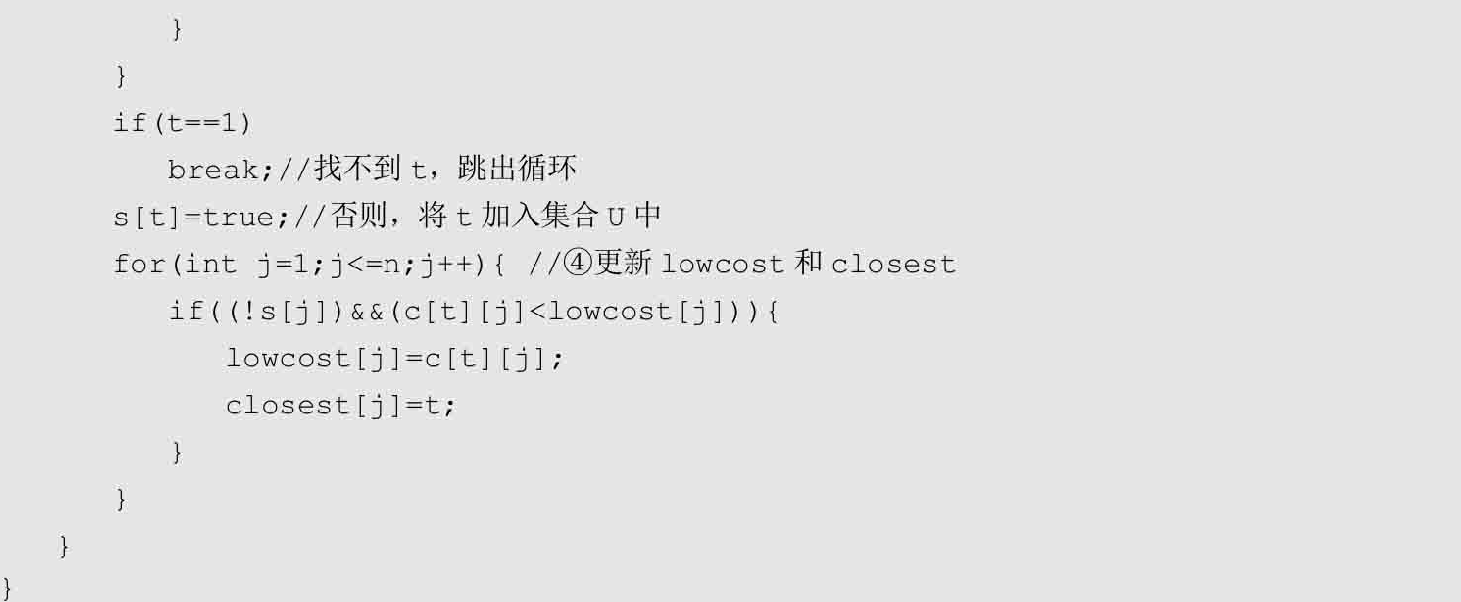


（20）得到的最小生成树如下图所示。最小生成树的权值之和为57，即把lowcost[]数组中的值加起来。



**3. 算法实现**

****

****

**4. 算法分析**

时间复杂度：在Prim(int n, int u0, int c[N][N])算法中，共有4个for语句，①for语句的执行次数为n；②在for语句里面嵌套了两个for语句③、④，它们的执行次数均为n，对算法的运行时间贡献最大，当外层循环标号为1时，③、④for语句在内层循环的控制下均执行n次，外层循环②从1～n，因此，该语句的执行次数为n2，时间复杂度为O(n2)。

空间复杂度：算法所需要的辅助空间包含lowcost[]、closest[]和s[]，空间复杂度为O(n)。

二、　Kruskal算法

构造最小生成树还有一种算法，即Kruskal算法：设图G（G=(V,E)）是无向连通带权图，V={1,2,…,n}；设最小生成树T=(V,TE)，该树的初始状态为只有n个节点而无边的非连通图T=(V,{})，Kruskal算法将这n个节点看成n个孤立的连通分支。它首先将所有的边都按权值从小到大排序，然后只要在T中选的边数不到n−1，就做这样的**贪心选择**：在边集E中选取权值最小的边(i,j)，如果将边(i,j)加入集合TE中不产生回路（圈），则将边(i,j)加入边集TE中，即用边(i,j)将这两个连通分支合并连接成一个连通分支；否则继续选择下一条最短边。把边(i,j)从集合E中删去，继续上面的贪心选择，直到T中的所有节点都在同一个连通分支上为止。此时，选取的n−1条边恰好构成图G的一棵最小生成树T。

**那么，怎样判断加入某条边后图T会不会出现回路呢？**该算法对于手工计算十分方便，因为肉眼可以很容易看出挑选哪些边能够避免回路（避圈法），但计算机程序需要一种机制进行判断。Kruskal算法用了一种非常聪明的方法，就是**运用集合避圈**：如果所选择加入的边的起点和终点都在T的集合中，就可以断定会形成回路（圈）。这其实就是前面提到的“避圈法”：**边的两个节点不能属于同一个集合**。

**1. 算法步骤**

（1）初始化。将所有边都按权值从小到大排序，将每个节点的集合号都初始化为自身编号。

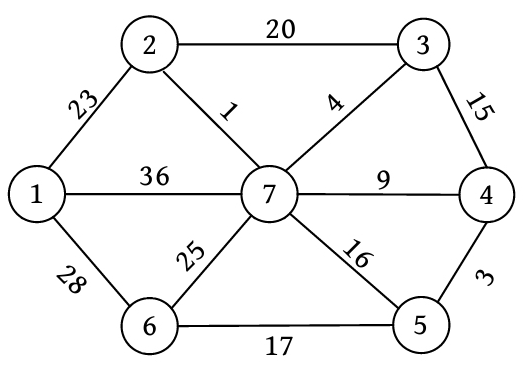
（2）按排序后的顺序选择权值最小的边(u,v)。

（3）如果节点u和v属于两个不同的连通分支，则将边(u,v)加入边集TE中，并将两个连通分支合并。

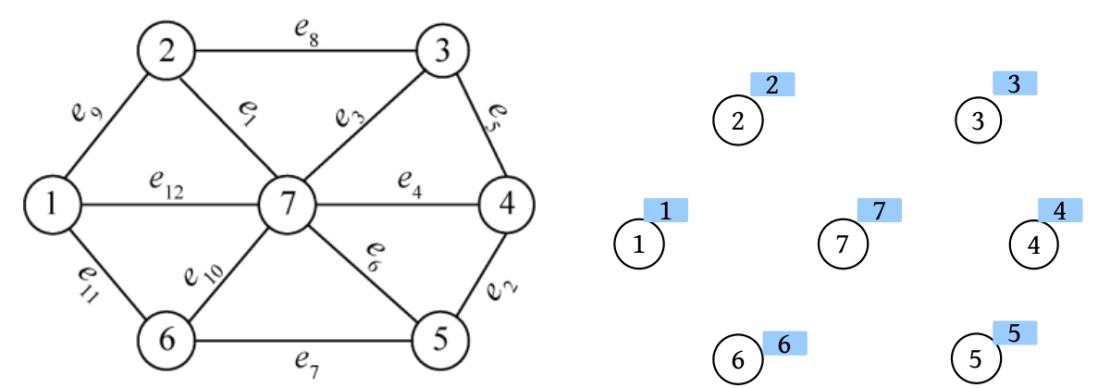
（4）如果选取的边数小于n−1，则转向步骤2，否则算法结束。

**2. 图解**

设图G（G =(V, E)）是无向连通带权图，如下图所示。

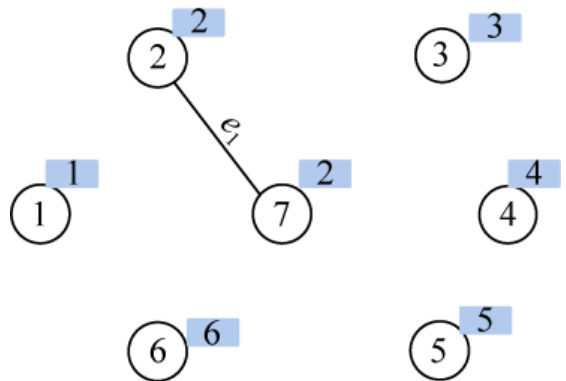


（1）初始化。将所有边都按权值从小到大排序，如下图所示。将每个节点都初始化为一个孤立的分支，即一个节点对应一个集合，集合号为该节点的序号，如下图所示。



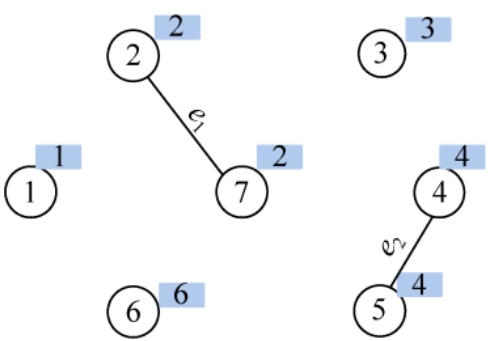
（2）找最小。在E中寻找权值最小的边e1(2,7)，边值为1。

（3）合并。节点2和节点7的集合号不同，即属于两个不同的连通分支，将边(2,7)加入边集TE中，执行合并操作，将两个连通分支的所有节点都合并为一个集合；假设把小的集合号赋值给大的集合号，以下均做如此处理，那么将节点7的集合号也改为2，如下图所示。



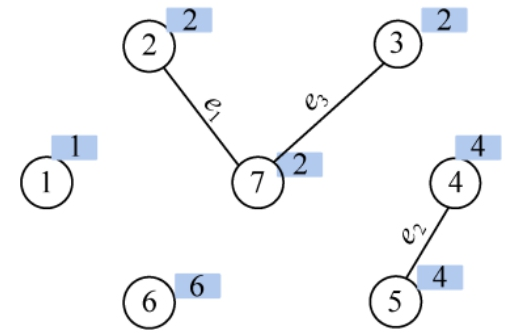
（4）找最小。在E中寻找权值最小的边e2(4,5)，边值为3。

（5）合并。节点4和节点5的集合号不同，即属于两个不同的连通分支，将边(4,5)加入边集TE中，执行合并操作，将两个连通分支的所有节点都合并为一个集合，将节点5的集合号也改为4，如下图所示。



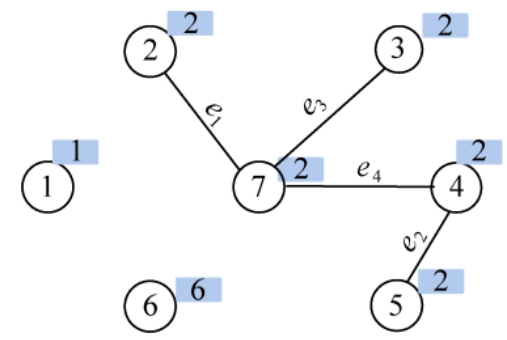
（6）找最小。在E中寻找权值最小的边e3(3,7)，边值为4。

（7）合并。节点3和节点7的集合号不同，即属于两个不同的连通分支，将边(3,7)加入边集TE中，执行合并操作，将两个连通分支的所有节点都合并为一个集合，将节点3的集合号也改为2，如下图所示。



（8）找最小。在E中寻找权值最小的边e4(4,7)，边值为9。

（9）合并。节点4和节点7的集合号不同，即属于两个不同的连通分支，将边(4,7)加入边集TE中，执行合并操作，将两个连通分支的所有节点都合并为一个集合，将节点4、5的集合号都改为2，如下图所示。



（10）找最小。在E中寻找权值最小的边e5(3,4)，边值为15。

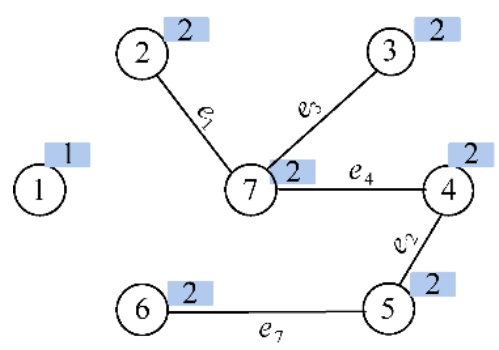
（11）合并。节点3和节点4的集合号相同，属于同一连通分支，不能选择，否则会形成回路。

（12）找最小。在E中寻找权值最小的边e6(5,7)，边值为16。

（13）合并。节点5和节点7的集合号相同，属于同一连通分支，不能选择，否则会形成回路。

（14）找最小。在E中寻找权值最小的边e7(5,6)，边值为17。

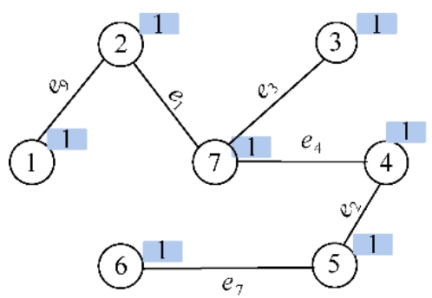
（15）合并。节点5和节点6集合号不同，即属于两个不同的连通分支，将边(5,6)加入边集TE中，执行合并操作，将两个连通分支的所有节点都合并为一个集合，将节点6的集合号改为2，如下图所示。



（16）找最小。在E中寻找权值最小的边e8(2,3)，边值为20。

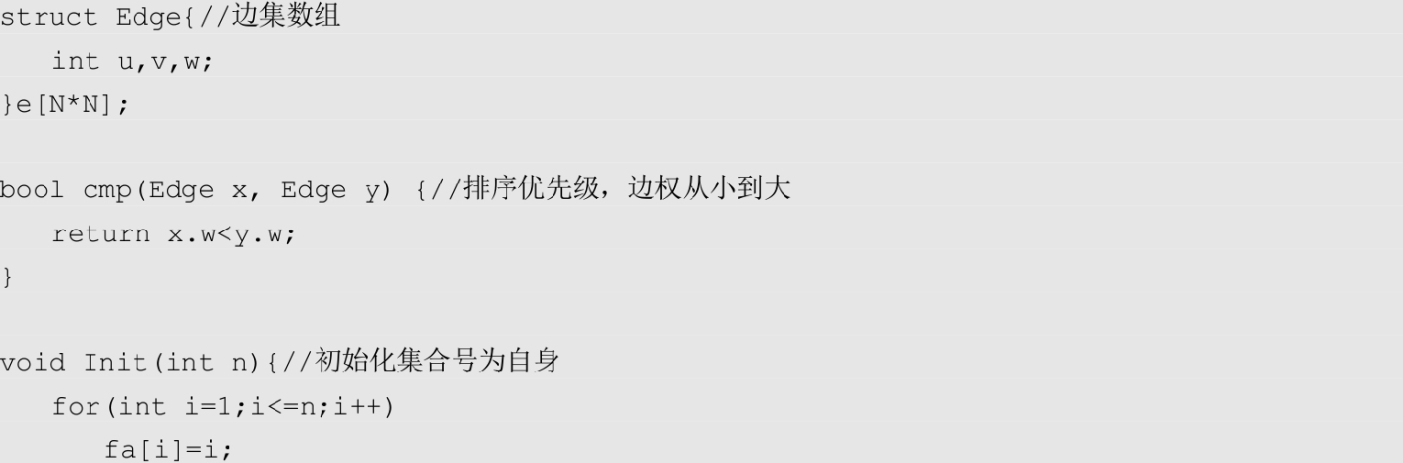
（17）合并。节点2和节点3的集合号相同，属于同一连通分支，不能选择，否则会形成回路。

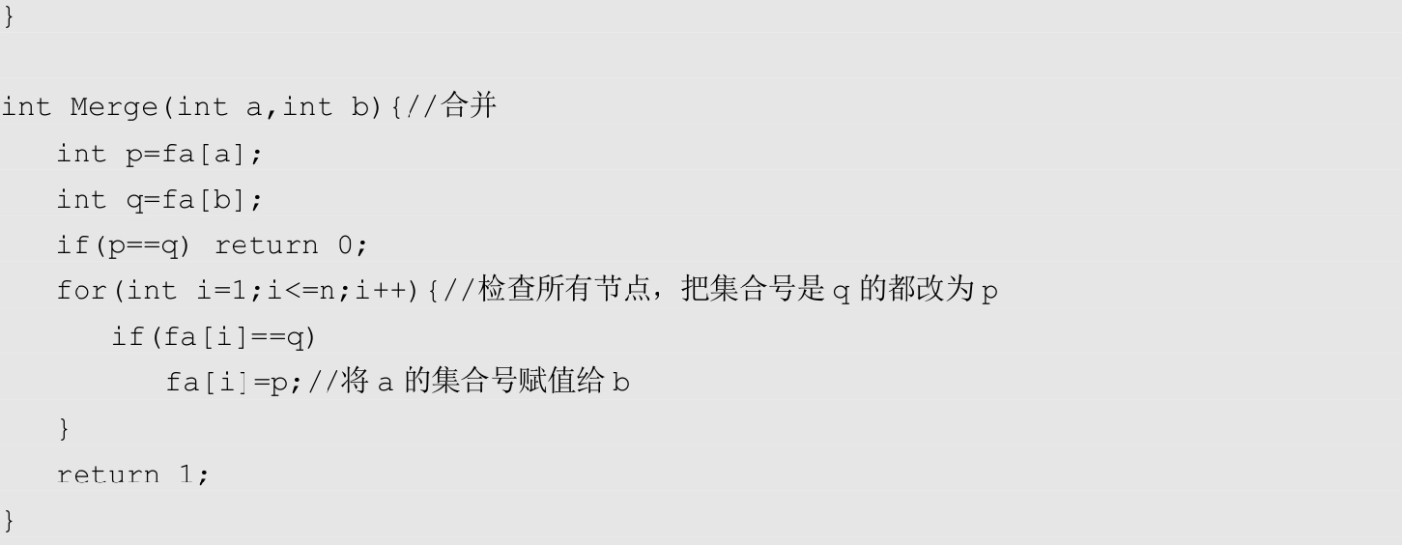
（18）找最小。在E中寻找权值最小的边e9(1,2)，边值为23。

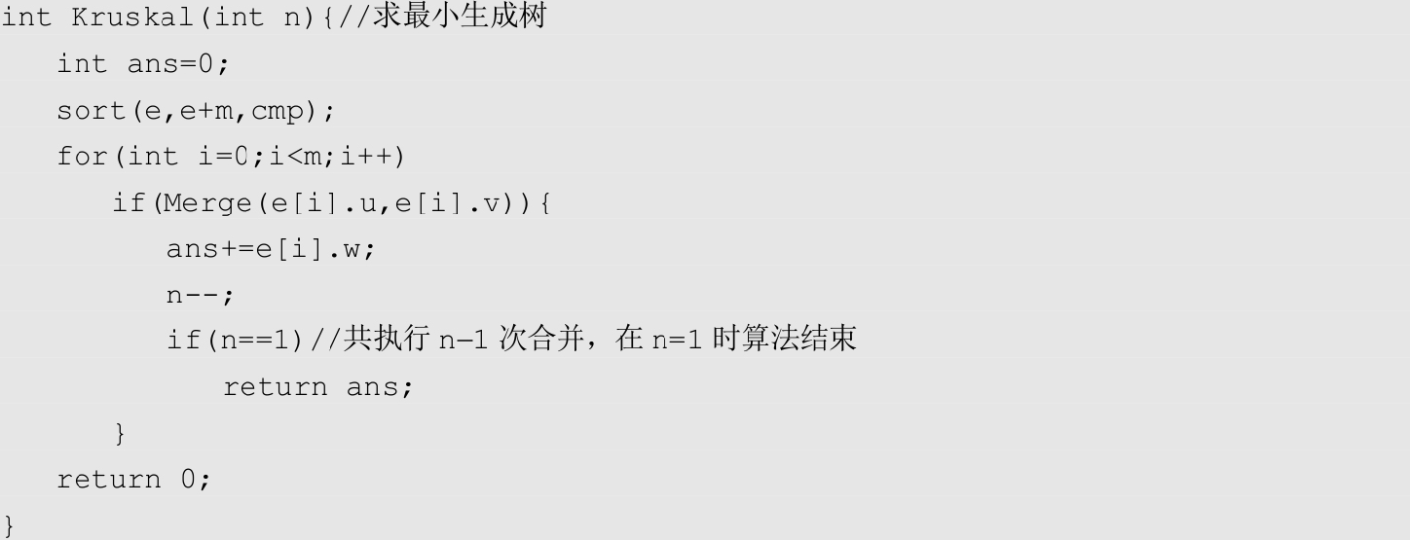
（19）合并。节点1和节点2的集合号不同，即属于两个不同的连通分支，将边(1,2)加入边集TE中，执行合并操作，将两个连通分支的所有节点都合并为一个集合，将节点2、3、4、5、6、7的集合号都改为1，如下图所示。

（20）选中的各边和所有的节点就是最小生成树，各边权值之和就是最小生成树的代价。

**3. 算法代码**

****

****

****

**4. 算法分析**

**时间复杂度：**在该算法中需要对边进行排序，若使用快速排序，则算法的时间复杂度为O(mlogm)。而合并集合需要n−1次合并，每次合并的时间复杂度都为O(n)，合并集合的时间复杂度为O(n2)。总的时间复杂度为O(mlogm)。

如果使用并查集优化合并操作，则每次合并的时间复杂度都为O(logn)。

**空间复杂度：**辅助空间包括一些变量和集合号数组fa[]，空间复杂度为O(n)。