数据结构-图

信智学院　陈琰宏

主要内容

01 prim算法

02 Kruskal算法

什么是最小生成树

一个带权连通无向图G（假定每条边上的权值均大于零）中可能有多棵生成树，每棵生成树中所有边上的权值之和可能不同；**图的所有生成树中具有边上的权值之和最小的树称为图的最小生成树**。

按照生成树的定义，n个顶点的连通图的生成树有n个顶点、$n-1$条边。因此构造最小生成树的准则有几条：

（1）必须只使用该图中的边来构造最小生成树；

（2）必须使用且仅使用$n-1$条边来连接图中的n个顶点，生成树一定是连通的；

（3）不能使用产生回路的边。

（4）最小生成树的权值之和是最小的，但一个图的最小生成树不一定是唯一的。

构造最小生成树的方法：**克鲁斯卡尔(Kruskal)**算法和**普里姆(Prim)**算法。都得遵守以上准则

1 普里姆算法

普里姆（Prim）算法是一种构造性算法。假设G=(V,E)是一个具有*n*个顶点的带权无向连通图，T=(U,TE)是G的最小生成树，其中U是T的顶点集，TE是T的边集，则由G构造从起始顶点*v*出发的最小生成树T的步骤如下：

（1）初始化U={*v*}。以*v*到其他顶点的所有边为候选边；

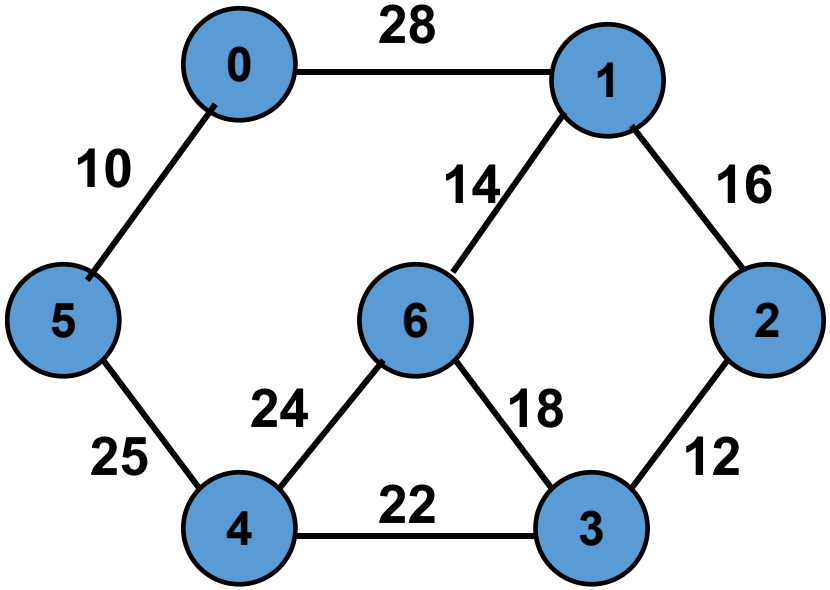
（2）重复以下步骤*n*-1次，使得其他*n*-1个顶点被加入到U中：

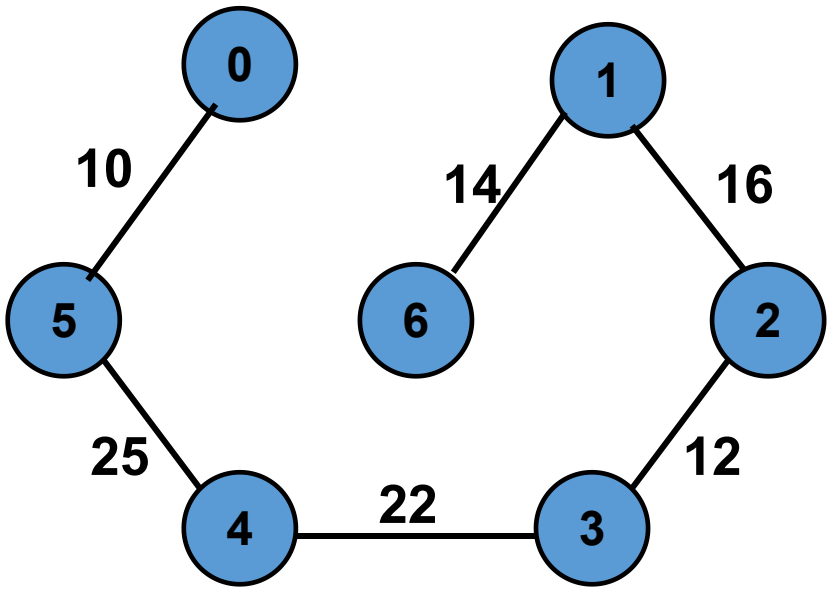
① 从候选边中挑选权值最小的边加入TE，设该边在V-U中的顶点是*k*，将*k*加入U中；

② 考察当前V-U中的所有顶点*j*，修改候选边：若(*k*,*j*)的权值小于原来和顶点*j*关联的候选边，则用(*k*,*j*)取代后者作为候选边。

1.1 prim构造最小生成树的过程

U：当前生成树顶点集合，
V：不属于当前生成树的顶点集合。





10.1.1 prim构造最小生成树的过程

采用邻接矩阵来存储图。为了实现上述思想，必须定义2个辅助数组：

cost[ ]: 存放顶点集合V(生成树外)内的各顶点到顶点集合U(生成树顶点集合)内的权值最小的边的权值。

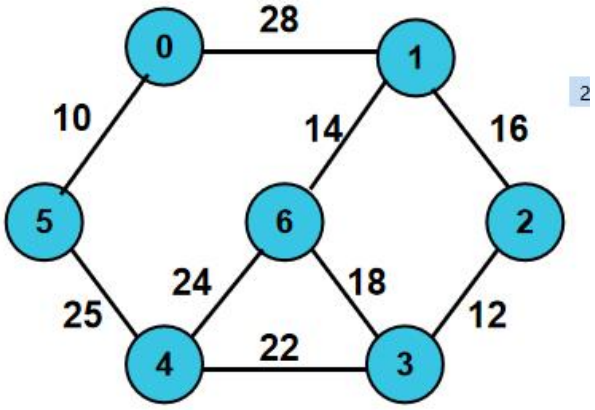
nearvex[ ]: 记录顶点的状态(是否已被选择)

从顶点0出发，2个辅助数组的初始状态为：

	0	1	2	3	4	5	6
cost	0	28	∞	∞	∞	10	∞

	0	1	2	3	4	5	6
nearvex	-1	0	0	0	0	0	0

V=5
选边(0,5)



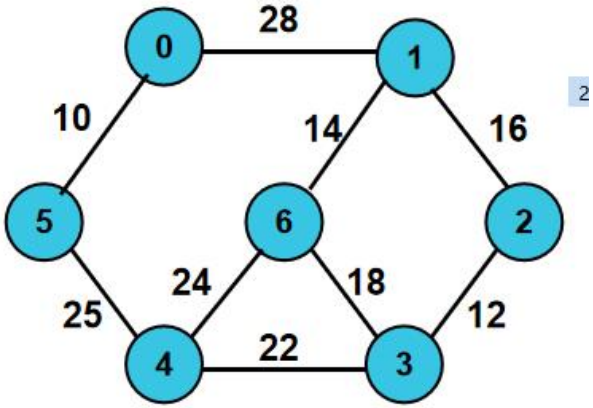
1.1 prim构造最小生成树的过程

在prim算法里要重复做以下工作(简化)：

1. 在cost[ ]中选择nearvex[i]!=-1且lowcost[i]最小的边i，用v标记它。
2. 将nearvex[v]改为-1，表示它已经加入到生成树顶点的集合。
3. 修改cost[ ]。
4. 修改nearvex[ ]。

	0	1	2	3	4	5	6
cost	0	28	∞	∞	∞	10	∞

	0	1	2	3	4	5	6
nearvex	-1	0	0	0	0	0	0



1.1 prim构造最小生成树的过程

在prim算法里要重复做以下工作(具体)：

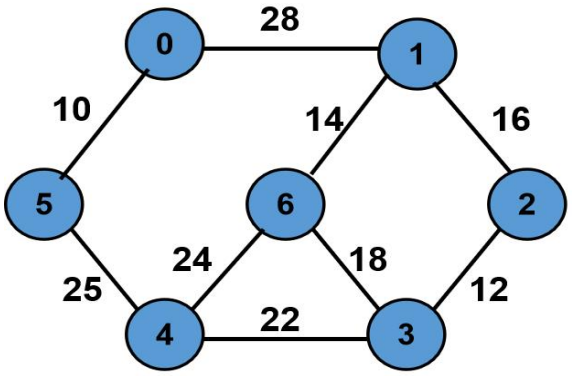
1. 在cost[ ]中选择nearvex[i]!=-1且cost[i]最小的边i，用v标记它。如第一次选中的v=5，则边(0,5)是选中的权值最小的边，相应的权值为cost[5]=10。
2. 将nearvex[v]改为-1，表示它已经加入到生成树顶点的集合，如需要可输出选中的顶点v。
3. 更新cost[ ]：注意，原来顶点v不属于生成树顶点集合，现在v加入进来了，则顶点集合V(生成树外)内的各顶点到顶点集合U(生成树顶点集合)内的权值需要更新，

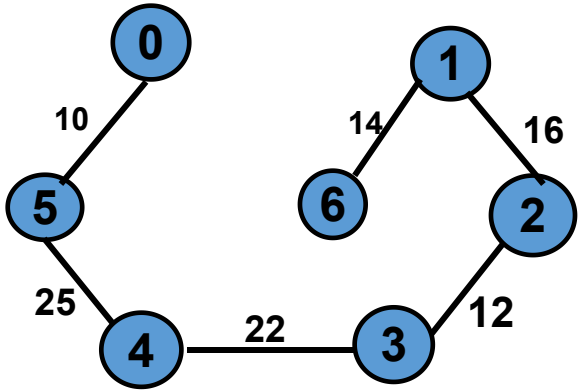
$$cost[i]=min\{cost[i], AdjMatrix[v][j]\},$$

cost[i]表示原权值， AdjMatrix[v][j]表示新加入点v到其他点的权重

1.1 prim构造最小生成树的过程

cost	0					10		nearvex	-1					0		V=5, 选边(0,5)
cost	0	28	∞	∞	**25**	10	∞	nearvex	-1	0	0	0	**5**	**-1**	0	V=4, 选边(5,4)
cost	0	28	∞	**22**	25	10	**24**	nearvex	-1	0	0	**4**	-1	-1	**4**	V=3, 选边(4,3)
cost	0	28	**12**	22	25	10	**18**	nearvex	-1	0	**3**	-1	-1	-1	**3**	V=2, 选边(3,2)
cost	0	**16**	12	22	25	10	18	nearvex	-1	**2**	-1	-1	-1	-1	3	V=1, 选边(2,1)
cost	0	16	12	22	25	10	**14**	nearvex	-1	-1	-1	-1	-1	-1	**1**	V=6, 选边(1,6)
	0	16	12	22	25	10	14	nearvex	-1	-1	-1	-1	-1	-1	-1	





1.2 prim代码实现

```
#define maxn 100
const int inf=0xffff;
int e[maxn][maxn];
cost[maxn];
int vis[maxn];
//U：当前生成树顶点集合，
//V：不属于当前生成树的顶点集合。
//cost[i]表示 i 节点当前加入集合中的最小花费
//相当于vetnearvex数组
//    标记 i 节点是否已加入集合u中
```

```
void prim(){
    int i,j,minn;
    int cnt=1,sum=0;
    for(i=1;i<=n;i++){
        vis[i]=0;//初始全部的点在集合U中
        cost[i]=e[1][i];
    }
    vis[1]=1;//把初始第一点添加到V集合
    while(cnt<n){
        minn=inf;
        for(i=1;i<=n;i++)
            if(vis[i]==0&&cost[i]<minn){//寻找未被处理的最小cost[i]
                minn=cost[i];
                j=i;//找到新的添加到集合u中的点
            }
        vis[j]=1;//把新的点添加到集合u中
        for(i=1;i<=n;i++){//更新cost[i]
        //因为j的加入，比较j到其他点的距离e[j][i]与原cost[i]
            if(vis[i]==0&&e[j][i]<cost[i])
                cost[i]=e[j][i];//
        }
        cnt++;
    }
}
```

1.2 prim代码实现

上述算法的初始化时间是O(1)，k循环中有两个循环语句，其时间大致为：

$$\sum_{k=0}^{n-2}\left(\sum_{j=k}^{n-2}O(1)+\sum_{j=k+1}^{n-2}O(1)\right)\approx 2\sum_{k=0}^{n-2}\sum_{j=k}^{n-2}O(1)$$

令O(1)为某一正常数C，展开上述求和公式可知其数量级仍是 n 的平方。所以，整个算法的时间复杂性是$O(n^2)$

1.3 [3316] 还是畅通工程

浙江大学研究生复试题：某省调查乡村交通状况，得到的统计表中列出了任意两村庄间的距离。省政府“畅通工程”的目标是使全省任何两个村庄间都可以实现公路交通（但不一定有直接的公路相连，只要能间接通过公路可达即可），并要求铺设的公路总长度为最小。请计算最小的公路总长度。

输入

测试输入包含若干测试用例。每个测试用例的第1行给出村庄数目N（< 100）；随后的N(N-1)/2行对应村庄间的距离，每行给出一对正整数，分别是两个村庄的编号，以及此两村庄间的距离。为简单起见，村庄从1到N编号。

当N为0时，输入结束，该用例不被处理。

输出

对每个测试用例，在1行里输出最小的公路总长度。

[3316] 还是畅通工程

样例输入

```
3
1 2 1
1 3 2
2 3 4
4
1 2 1
1 3 4
1 4 1
2 3 3
2 4 2
3 4 5
0
```

样例输出

```
3
5
```

[3316] 还是畅通工程

```
int main(){
    while(cin>>n){
        if(n==0)
            break;
        m=n*(n-1)/2;
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                if(i==j)
                    e[i][j]=0;
                else
                    e[i][j]=inf;
        int u,v,w;
        for(int i=1;i<=m;i++){
            cin>>u>>v>>w;
            e[u][v]=e[v][u]=w;
        }
        prim();
    }
    return 0;
}
```

```
#include<iostream>
using namespace std;
#define maxn 100
const int inf=0xffff;
int e[maxn][maxn],cost[maxn];
int vis[maxn];
int m,n,j;
// cost[i] 表示 i 节点当前加入集合中的最小花费
// vis [i] 标记 i 节点是否已加入集合u中
```

[3316] 还是畅通工程

```
void prim(){
    int i,j,minn;
    int cnt=1,sum=0;
    for(i=1;i<=n;i++){
        vis[i]=0;//初始全部的点在集合A中
        cost[i]=e[1][i];
    }
    vis[1]=1;//把初始第一点添加到B集合
    while(cnt<n){
        minn=inf;
        for(i=1;i<=n;i++)
            if(vis[i]==0&&cost[i]<minn){//在集合v中找d[i]最小的权值
                minn=cost[i];
                j=i;//找到新的添加到集合u中的点
            }
        vis[j]=1;//把新的点添加到集合u中
        sum=sum+cost[j];//求和，计算最小距离
        cnt++;
        for(i=1;i<=n;i++){//更新cost[i]
            if(vis[i]==0&&e[j][i]<cost[i])
                cost[i]=e[j][i];
        }
    }
    cout<<sum<<endl;
}
```

[2910] 城市公交网建设问题

样例输入

```
5 8
1 2 2
2 5 9
5 4 7
4 1 10
1 3 12
4 3 6
5 3 3
2 3 8
```

样例输出

```
1 2
2 3
3 4
3 5
```

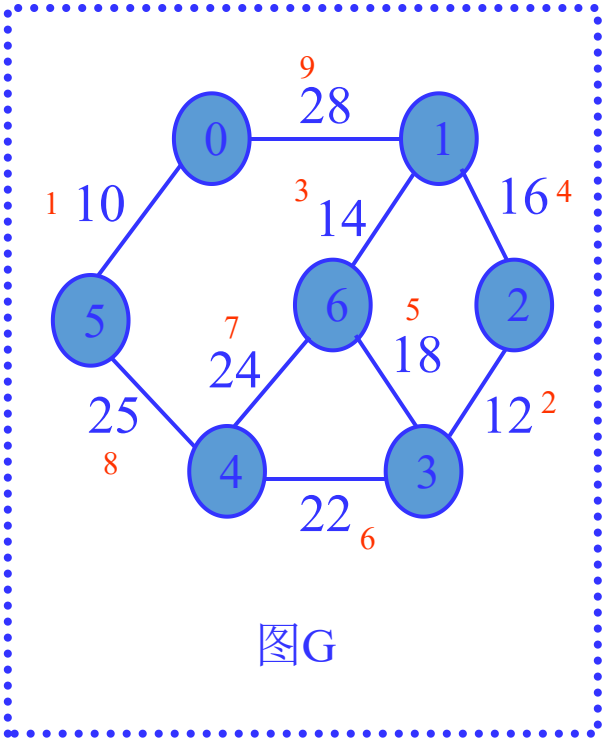
2 kruskal算法

克鲁斯卡尔（Kruskal）算法是一种按权值的递增次序选择合适的边来构造最小生成树的方法。假设G=(V,E)是一个具有*n*个顶点的带权连通无向图，T=(U,TE)是G的最小生成树，则构造最小生成树的步骤如下：

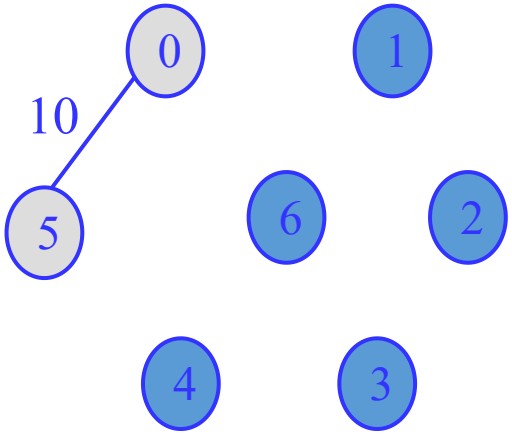
（1）置U的初值等于V（即包含有G中的全部顶点），TE的初值为空集（即图T中每一个顶点都构成一个分量）。

（2）将图G中的边按权值从小到大的顺序依次选取：若选取的边未使生成树T形成回路，则加入TE；否则舍弃，直到TE中包含*n*-1条边为止。

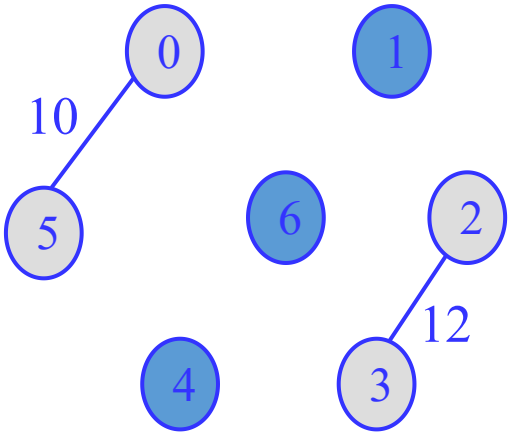
2.1 kruskal算法求解过程



图G



取1号边

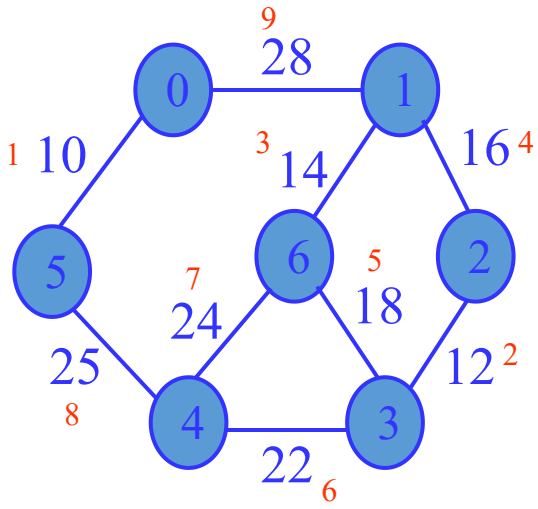


取2号边

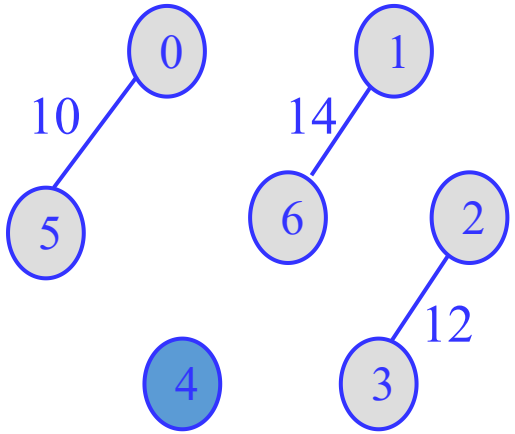
按边大小递减排序

克鲁斯卡尔算法求解最小生成树的过程（1）

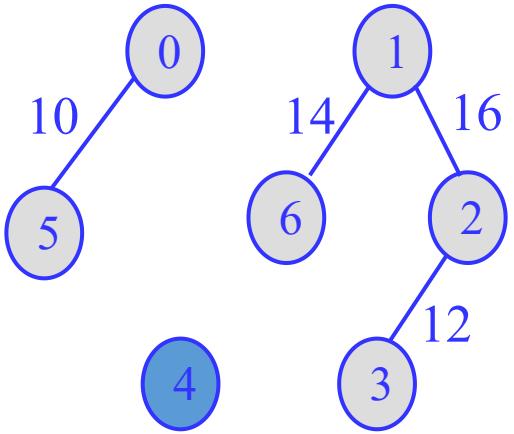
2.1 kruskal算法求解过程



图G



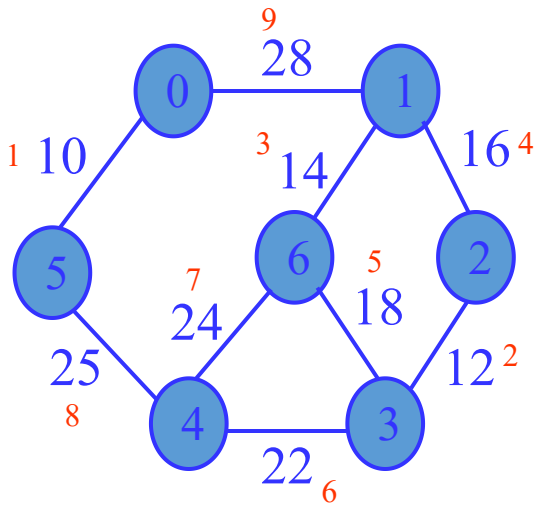
取3号边



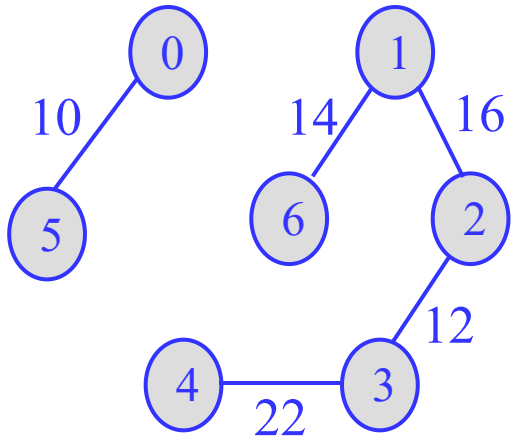
取4号边

克鲁斯卡尔算法求解最小生成树的过程（2）

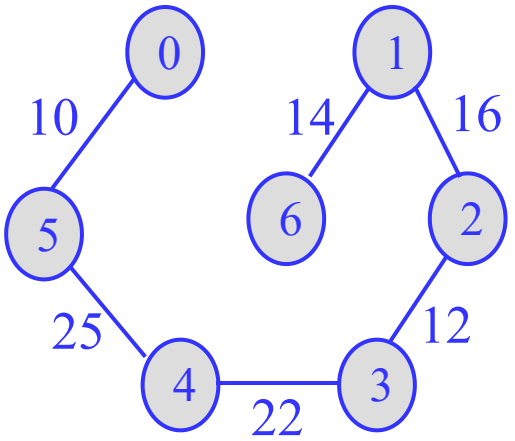
2.1 kruskal算法求解过程



图G



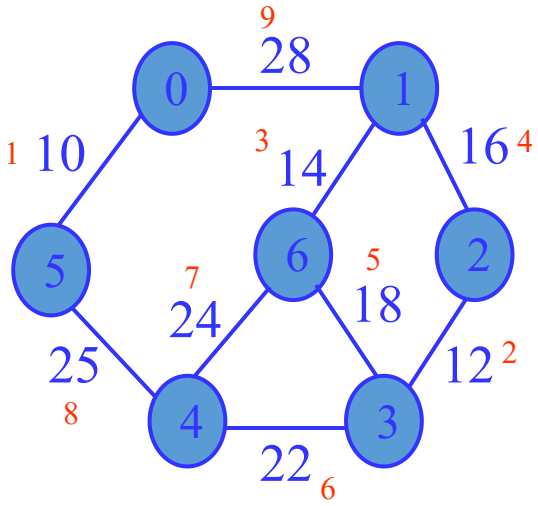
取6号边（不能取5号边）

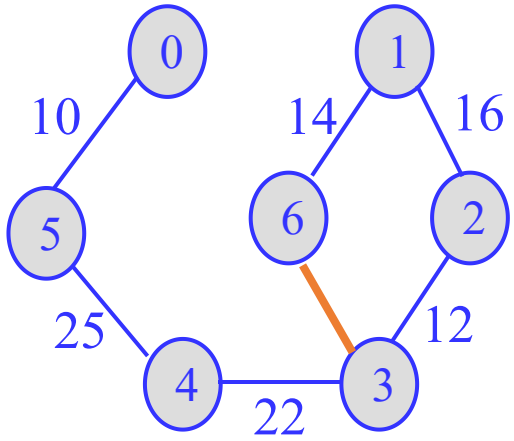


取8号边（不能取7号边）

克鲁斯卡尔算法求解最小生成树的过程（3）

如何解决出现回路的问题？





Kruskal （克鲁斯卡尔）算法的伪代码：

```
void Kruskal ( Graph G )
{  T = {空集};
   while ( T 中收集的边不到n–1条 && 原图的边集E非空 ) {
      从E中选择最小代价边(v, w);
      从E中删除此边(v, w);
      if ( (v, w)的选取将在T中构成回路 )
                          /* 判断由并查集的Find完成 */
            将(v, w)加入T；  /* 此
      else
            彻底丢弃(v, w) ;
   }  /* 结束while */
   if ( T 中收集的边不到n–1条 )
        printf( “图不连通” ) ;
}
```

时间复杂性主要体现在while循环中，循环|E|次；带路径压缩的Find函数和Union函数完成需要$O(\log|V|)$；总体时间复杂度为$O(|E|\cdot\log|V|)$。

❖Kruskal 核心代码

在实现克鲁斯卡尔算法Kruskal()时，用一个数组E存放图G中的所有边，要求它们是按权值递增排列的，为此先从图G的邻接矩阵中获取所有边集E，再采用直接插入排序法对边集E按权值递增排序。克鲁斯卡尔算法如下：

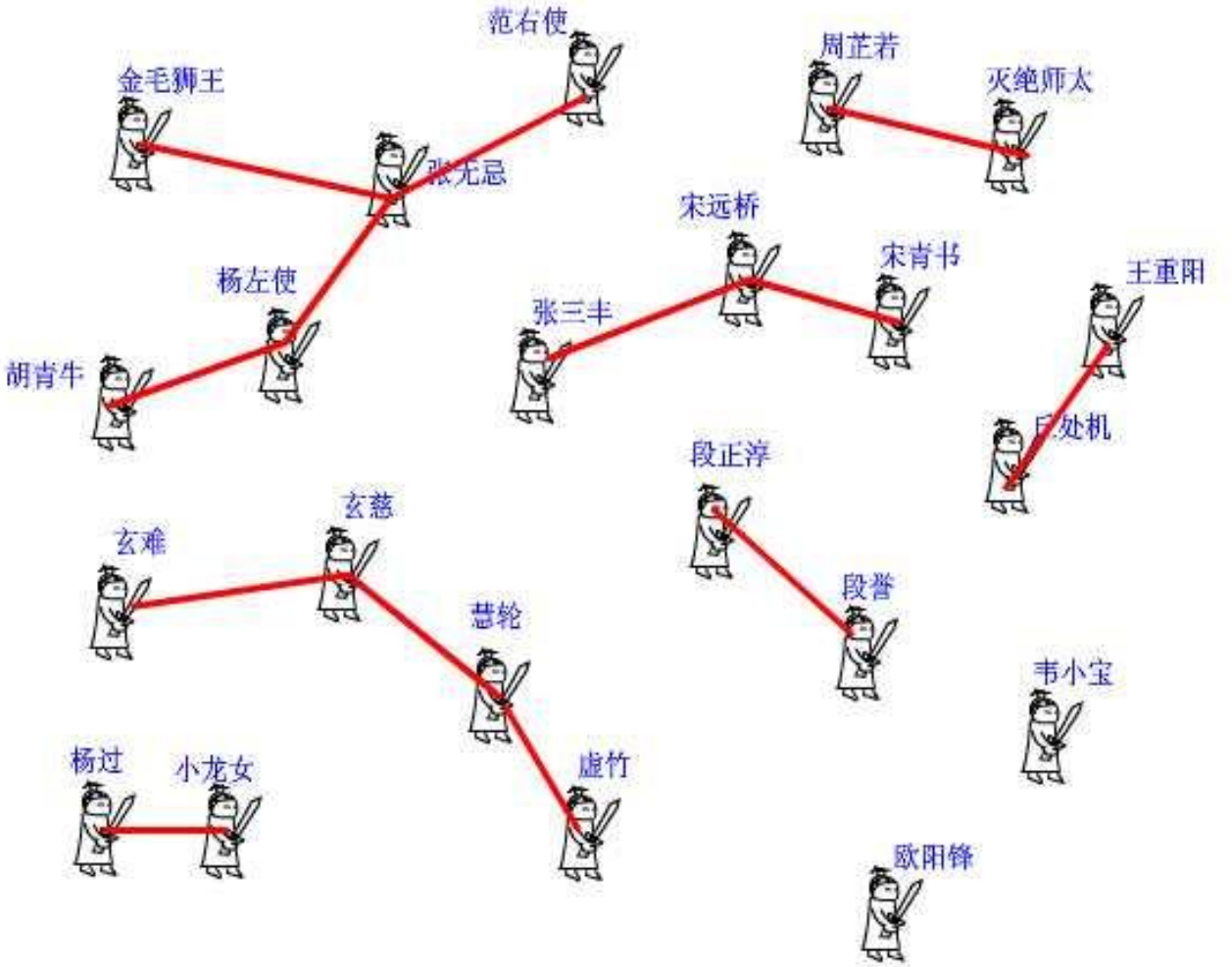
```
int kruskal() {
    int ans = 0;
    sort(e, e+n*(n-1)/2, cmp);
    for(int i = 1; i <= n; i ++) fa[i] = i;// 并查集初始化

    for(int i = 0; i <= n*(n-1)/2; i ++) {
        if(Union(e[i].u, e[i].v))
            ans += e[i].dis;
    }
    return ans;
}
```

3 并查集

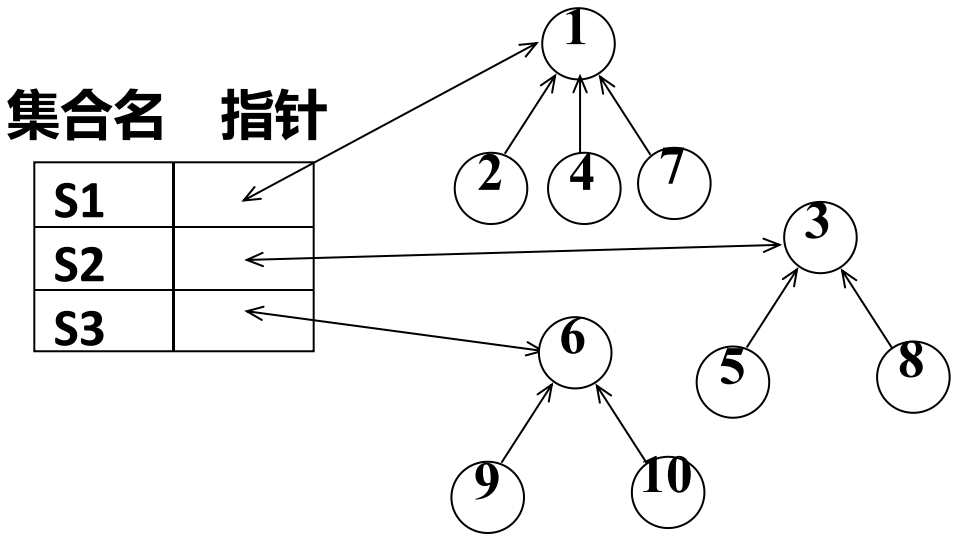
在一些有N个元素的集合应用问题中，我们通常是在开始时让每个元素构成一个单元素的集合，然后按一定顺序将属于同一组的元素所在的集合合并，其间要反复查找一个元素在哪个集合中。这一类问题看似并不复杂，但数据量极大，若用正常的数据结构来描述的话，往往超过了空间的限制，计算机无法承受；即使在空间上能勉强通过，运行的时间复杂度也极高，根本不可能在规定的运行时间内计算出试题需要的结果，只能采用一种特殊数据结构——并查集来描述。

3 并查集


范右使
金毛狮王
张无忌
杨左使
胡青牛
周芷若
灭绝师太
宋远桥
宋青书
张三丰
王重阳
段正淳
玄慈
玄难
慧轮
段誉
韦小宝
杨过
小龙女
虚竹
欧阳锋

3.1 并查集存储

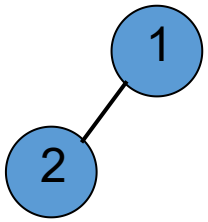
- 逻辑上，可以用**树结构**表示集合，树的每个结点代表一个集合元素。
- 例如，有三个互不相交的整数集合S1={1,2,4,7}、S2={3,5,8}、S3={6,9,10}，
- 这三个集合的多叉树表示形式：



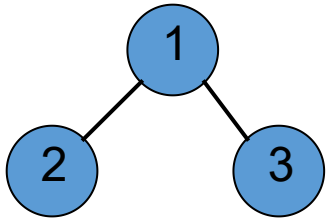
3.1 并查集存储

元素的合并图示：

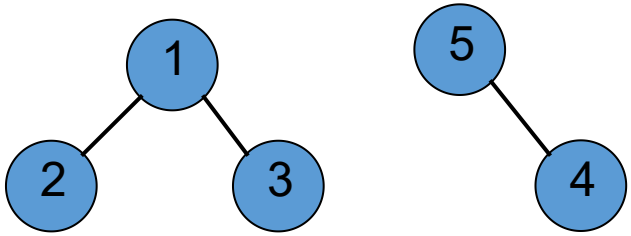
① 合并1和2



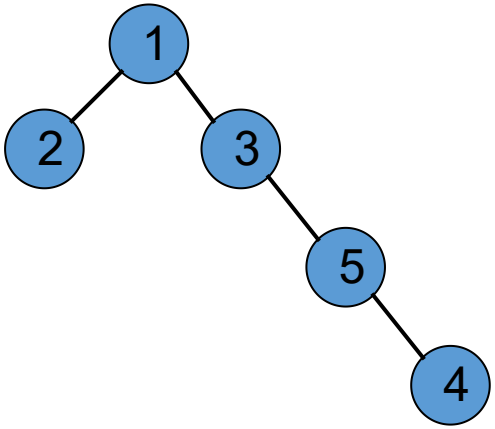
② 合并1和3



③ 合并5和4



④ 合并3和5



用father[i]表示元素i的父亲结点，
进行不断并到不同的集合中

3.2 核心代码

寻找父节点：

```
 7  int find(int x){
 8      while(x!=father[x])
 9          x=father[x];//寻找根节点
10      return x;
11  }
```

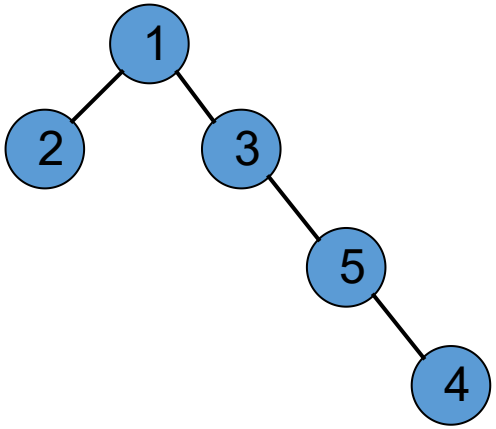
```
20  int find(int x)
21  {
22      if(x==parent[x])
23          return x;
24      int fx=find(parent[x]);
25      parent[x]=fx;
26      return fx;
27  }
```

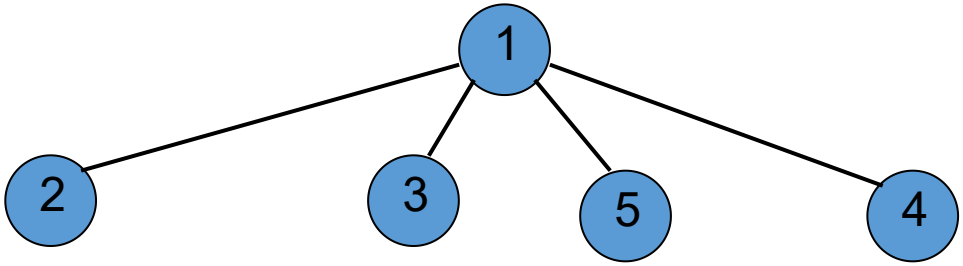
合并成一个集合：

```
12  void union1(int r1,int r2){
13      father[r2]=r1;//合并
14  }
```

优化：路径压缩

两种合并方式





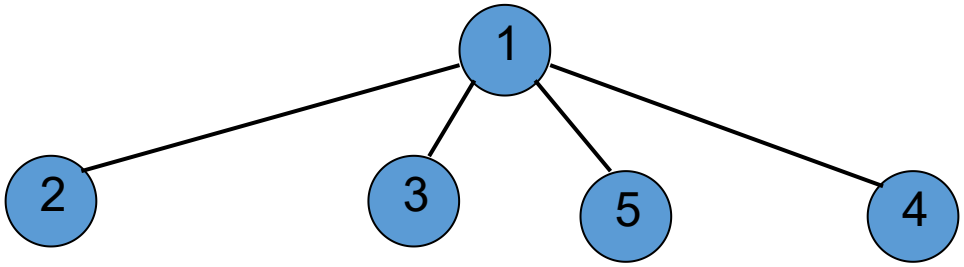
路径压缩实际上是在找完根结点之后，在递归回来的时候顺便把路径上元素的父亲指针都指向根结点

不同的合并方式给查询操作带来不同的效率

优化：路径压缩

寻找根结点编号并压缩路径：

```
int find (int x)
{
    if (x!=father[x])
        father[x] = find (father[x]);
    return father[x];
}
```



合并两个集合：

```
void unionn(int x,int y)
{
    x = find(x);y = find(y);
    if(x!=y)
    father[y] = x;
}
```

非递归路径压缩

```
 8  int find(int x)
 9  {
10      int k,g;
11      g=x;
12      while(g!=father[g])//查找根节点
13      {
14          g=father[g];
15      }     //找到根节点，用g记录下
16      while(x!=g)//非递归路径压缩操作
17      {
18          k=father[x];
19          father[x]=g; //father[x]指向根节点
20          x=k;
21      }
22      return g;   //返回根节点的值
23  }
```

```
22  int get_f(int x)
23  {//寻找父节点，并做压缩
24      while(x != f[x])
25          x = f[x] = f[f[x]];
26      return x;
27  }
```

3.2 路径压缩举例

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  int father[11];
4  int root1,root2;
5  void union1(int r1, int r2){
6      father[r2]=r1;//合并
7  }
8  int find(int x){//路径压缩，每次更新节点的父节点，
9  //将所有链上的子节点全部链接到父节点上
10     if(father[x]!=x)father[x]=find(father[x]);
11     return father[x];//查找
12 }
```

```
链1：1 2 3 4 5
链2：6 7 8 9 10
5和10普通合并后链的父节点：1 2 3 10 6 6 7 8 9 10
```

```
链1：1 2 3 4 5
链2：6 7 8 9 10
5和10压缩合并后链的父节点：1 1 1 1 1 1 6 6 6 6
```

3.2 路径压缩举例

```
14  int main(){
15      for(int i=1;i<=10;i++)father[i]=i;//初始化
16      for(int i=10;i>6;i--){
17          father[i]=i-1;//形成6->7->8->9->10-的链
18          father[i-5]=i-6;//形成1->2->3->4->5的链
19      }
20      cout<<"链1: ";
21      for(int i=2;i<=5;i++)cout<<father[i]<<" ";
22      cout<<5<<endl;
23      cout<<"链2: ";
24      for(int i=7;i<=10;i++)cout<<father[i]<<" ";
25      cout<<10<<endl;
26      cout<<"5和10压缩合并后链的父节点: ";
27      //union1(10,5);//把5,10合并 ,普通合并
28      //for(int i=2;i<=10;i++)cout<<father[i]<<" ";
29      //cout<<10<<endl;
30      int r1=find(5),r2=find(10);
31      if(r1!=r2)union1(r1,r2);
32      for(int i=1;i<=10;i++)cout<<father[i]<<" ";
33      return 0;
34  }
```

3.3 [2903]亲戚(relation)

或许你并不知道，你的某个朋友是你的亲戚。他可能是你的曾祖父的外公的女婿的外甥女的表姐的孙子。如果能得到完整的家谱，判断两个人是否亲戚应该是可行的，但如果两个人的最近公共祖先与他们相隔好几代，使得家谱十分庞大，那么检验亲戚关系实非人力所能及。在这种情况下，最好的帮手就是计算机。为了将问题简化，你将得到一些亲戚关系的信息，如Marry和Tom是亲戚，Tom和Ben是亲戚，等等。从这些信息中，你可以推出Marry和Ben是亲戚。请写一个程序，对于我们的关于亲戚关系的提问，以最快的速度给出答案。

输入格式】

输入由两部分组成。

第一部分以N，M开始。**N为问题涉及的人的个数(1≤N≤20000)**。这些人的编号为1,2,3,…，N。下面有M行(1≤M≤1 000 000)，每行有两个数**a_i，b_i，表示已知a_i和b_i是亲戚。**

第二部分以Q开始。以下Q行有**Q个询问**(1≤Q≤1 000 000)，每行为c_i，d_i，表示询问c_i和d_i是否为亲戚

3.3 [2903]亲戚(relation)

【输出格式】
对于每个询问ci, di，输出一行：若ci和di为亲戚，则输出“Yes”，否则输出“No”。

【输入样例】
```
10 7
2 4
5 7
1 3
8 9
1 2
5 6
2 3
3
3 4
7 10
8 9
```

【输出样例】
```
Yes
No
Yes
```

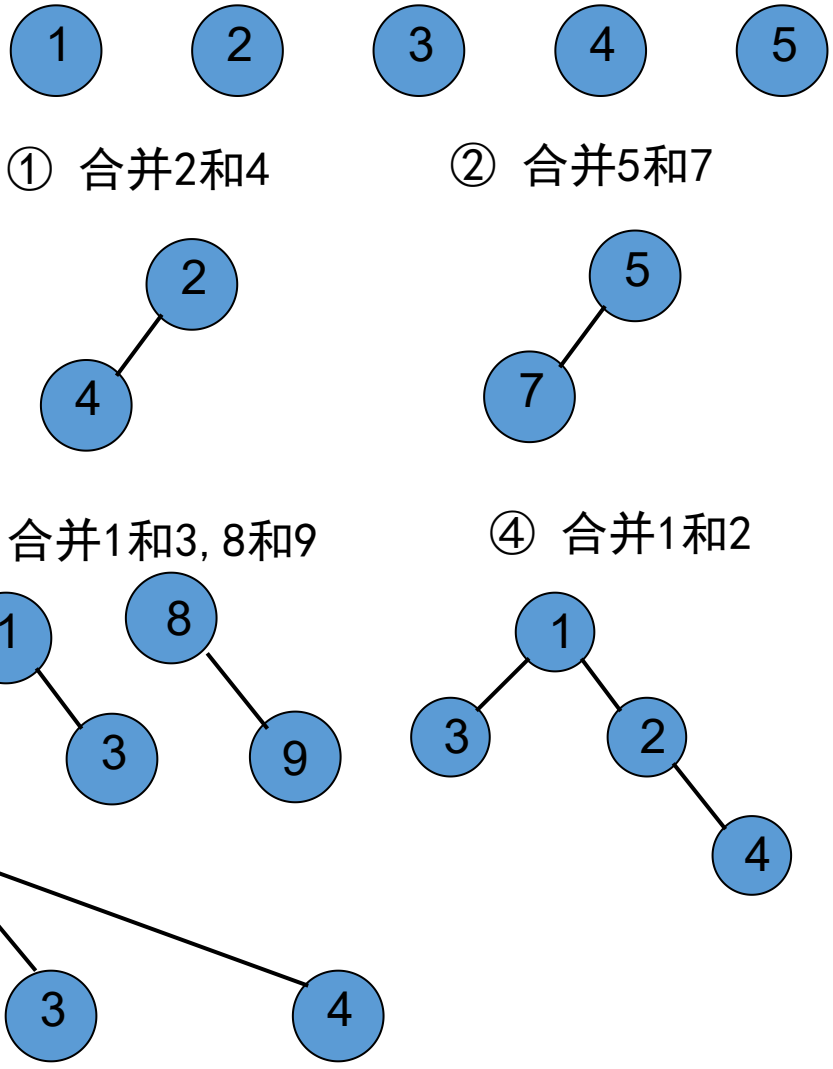
3.3 [2903]算法分析

将**每个人抽象成为一个点，数据给出M个边的关系**，两个人是亲戚的时候两点间有一条边。很自然的就得到了一个**N个顶点M条边的图论模型**。

用集合的思路，对于每个人建立一个集合，开始的时候集合元素是这个人本身，表示开始时不知道任何人是他的亲戚。以后每次给出一个亲戚关系时，就将两个集合合并。这样实时地得到了在当前状态下的集合关系。

3.3 [2903]算法分析

输入关系	分离集合
初始状态	{1}{2}{3}{4}{5}{6}{7}{8}{9}{10}
(2,4)	{1}{2,4}{3}{5}{6}{7}{8}{9}{10}
(5,7)	{1}{2,4}{3}{5,7}{6}{8}{9}{10}
(1,3)	{1,3}{2,4}{5,7}{6}{8}{9}{10}
(8,9)	{1,3}{2,4}{5,7}{6}{8,9}{10}
(1,2)	{1,2,3,4}{5,7}{6}{8,9}{10}
(5,6)	{1,2,3,4}{5,6,7}{8,9}{10}
(2,3)	{1,2,3,4}{5,6,7}{8,9}{10}



3.3 [2903]代码

```
3   int father[20000+5];
4   void union1(int r1, int r2){
5       father[r2]=r1;//合并
6   }
7   int find(int x){
8       while(father[x]!=x)x=father[x];
9       return x;//查找
10  }
```

```
12  int main()
13  {
14      int n,m,q,x,y;
15      cin>>n>>m;
16      for(int i=1;i<=n;i++)
17          father[i]=i;//初始化每个结点
18      for(int i=1;i<=m;i++){
19          scanf("%d %d",&x,&y);
20          int r1=find(x),r2=find(y);
21          if(r1!=r2)union1(r1,r2);
22      }//完成合并操作
23      cin>>q;
24      for(int i=1;i<=q;i++){
25          scanf("%d %d",&x,&y);
26          int r1=find(x),r2=find(y);
27          if(r1==r2)printf("Yes\n");
28          else printf("No\n");
29      }
30      return 0;
31  }
```

4 kruskal算法实现

```
void Kruskal ( Graph G )
{   T = {空集};
    while ( T 中收集的边不到n–1条 && 原图的边集E非空 ) {
        从E中选择最小代价边(v, w);
        从E中删除此边(v, w);
        if ( (v, w)的选取不在T中构成回路 )
                    /* 判断由并查集的Find完成 */
            将(v, w)加入T；  /* 此
        else
            彻底丢弃(v, w) ;
    }   /* 结束while */
    if ( T 中收集的边不到n–1条 )
        printf( “图不连通” ) ;
}
```

时间复杂性主要体现在while循环中，循环|E|次；带路径压缩的Find函数和Union函数完成需要$O(\log|V|)$；总体时间复杂度为$O(|E|\cdot\log|V|)$。

4.1 [3316] 还是畅通工程

浙江大学研究生复试题：某省调查乡村交通状况，得到的统计表中列出了任意两村庄间的距离。省政府“畅通工程”的目标是使全省任何两个村庄间都可以实现公路交通（但不一定有直接的公路相连，只要能间接通过公路可达即可），并要求铺设的公路总长度为最小。请计算最小的公路总长度。

输入

测试输入包含若干测试用例。每个测试用例的第1行给出村庄数目N（< 100）；随后的N(N-1)/2行对应村庄间的距离，每行给出一对正整数，分别是两个村庄的编号，以及此两村庄间的距离。为简单起见，村庄从1到N编号。

当N为0时，输入结束，该用例不被处理。

输出

对每个测试用例，在1行里输出最小的公路总长度。

[3316] 还是畅通工程

样例输入

```
3
1 2 1
1 3 2
2 3 4
4
1 2 1
1 3 4
1 4 1
2 3 3
2 4 2
3 4 5
0
```

样例输出

```
3
5
```

```
#include <iostream>
#include <algorithm>
using namespace std;
const int maxn = 1e2 + 10;
const int inf = 0x3f3f3f3f;
int n, m, x, y, z, fa[maxn];
struct node {
    int u, v, dis;
} e[maxn*maxn];
//定义边数组，maxn个顶点，最多 maxn*maxn条边
bool cmp (node a,node b) {
int find_fa(int x) { //查找父节点
int Union (int x, int y) {
void kruskal() {
int main()
{
    while(scanf("%d", &n) && n) {
        m = n * (n-1) / 2;
        for(int i = 1; i <= m; i ++)
            scanf("%d%d%d", &e[i].u, &e[i].v, &e[i].dis);
        kruskal();
    }
    return 0;
}
```

```
bool cmp (node a,node b) {
    return a.dis < b.dis;
} // 对边的权值做从小到大排序
int find_fa(int x) { //查找父节点
    if(x == fa[x]) return x;
    return fa[x] = find_fa(fa[x]);
}
int Union (int x, int y) {

    int fx = find_fa(x), fy = find_fa(y);
    if(fx == fy) return 0;
    fa[fx] = fy;
    return 1;
}
void kruskal() {
    int ans = 0;
    for(int i = 1; i <= n; i ++) fa[i] = i;
    sort(e+1, e+1+m, cmp);
    for(int i = 1; i <= m; i ++) {
        if(Union(e[i].u, e[i].v)) {
            ans += e[i].dis;
        }
    }
    printf("%d\n", ans);
}
```

4 [2910]城市公交网建设问题

有一张城市地图，图中的顶点为城市，无向边代表两个城市间的连通关系，边上的权为在这两个城市之间修建高速公路的造价，研究后发现，这个地图有一个特点，即任一对城市都是连通的。现在的问题是，要修建若干高速公路把所有城市联系起来，问如何设计可使得工程的总造价最少？

输入

n（城市数，1<≤n≤100）

e（边数）

以下e行，每行3个数i,j,wiji,j,wij ，表示在城市i,j之间修建高速公路的造价。

输出

n-1行，每行为两个城市的序号，表明这两个城市间建一条高速公路。

注意看测试数据的输出顺序，有排序要求。

4 [2910]城市公交网建设问题

样例输入

```
5 8
1 2 2
2 5 9
5 4 7
4 1 10
1 3 12
4 3 6
5 3 3
2 3 8
```

样例输出

```
1 2
2 3
3 4
3 5
```

输入

n（城市数，1<≤n≤100）

e（边数）

以下e行，每行3个数i,j,wij，表示在城市i,j之间修建高速公路的造价。

输出

n-1行，每行为两个城市的序号，表明这两个城市间建一条高速公路。

注意看测试数据的输出顺序，有排序要求。

4 [2910]城市公交网建设问题

```
#include <bits/stdc++.h>
using namespace std;
#define maxn 10010
struct edge
{
    int from,to,flag,value;
    // flag标记改变是否被选择 value权值
}num[maxn];
int parent[110], n,m;
bool cmp1(edge x,edge y)
{//按照权值排序
    return x.value<y.value;
}
bool cmp2(edge x,edge y)
{//按照边排序
    if(x.from==y.from)
        return x.to<y.to;
    return x.from<y.from;
}
```

4 [2910]城市公交网建设问题

```
20  int find(int x)
21  {//找到父节点
22      if(x==parent[x])
23          return x;
24      int fx=find(parent[x]);
25      parent[x]=fx;
26      return fx;
27  }
28  void kruskal()
29  {
30      for(int i=1;i<=m;i++)
31      {
32          int fx=find(num[i].from);
33          int fy=find(num[i].to);
34          if(fx!=fy)
35          {
36              parent[fx]=fy;
37              num[i].flag=1;//选取的边设置为1
38          }
39      }
40  }
```

4 [2910]城市公交网建设问题

```
43  int main()
44  {
45      scanf("%d%d",&n,&m);
46      for(int i=1;i<=n;i++)
47          parent[i]=i;//初始自己为父节点
48      for(int i=1;i<=m;i++)
49      {
50          int x,y;
51          scanf("%d%d%d",&x,&y,&num[i].value);
52          num[i].from=min(x,y);
53          num[i].to=max(x,y);
54          num[i].flag=0;
55      }
56      sort(num+1,num+1+m,cmp1);//按照权值排序
57      kruskal();
58      sort(num+1,num+1+m,cmp2);//按照起始点排序
59      for(int i=1;i<=m;i++)
60      {
61          if(num[i].flag)
62              printf("%d %d\n",num[i].from,num[i].to);
63      }
64      return 0;
65  }
```

通过改进可以降低该算法的时间复杂度，通常认为克鲁斯卡尔算法的时间复杂度为$O(e\log_2 e)$。由于与n无关，所以克鲁斯卡尔算法特别适合于稀疏图求最小生成树。

这是一个最短路径树 的典例之一。题意很明确——问你有多少棵最短路径树。虽然看起来n比较大，但是是单源的所以不用担心复杂度会爆掉。求一遍最短路，然后枚举每个点以及连出去的边，看看有多少条到达这个点的路径长度是等于最短路径的。最后累乘起来就行了

今天的课程结束啦……

下课了…
同学们再见！