最小生成树K算法

主讲老师：党东

Kruskal算法---算法思想

Kruskal算法也是一种**贪心**算法，它是**将边按权值排序**，每次从剩下的边集中**选择权值最小且两个端点不在同一集合的边**加入生成树中，反复操作，直到加入了n-1条边。

Kruskal算法---算法步骤

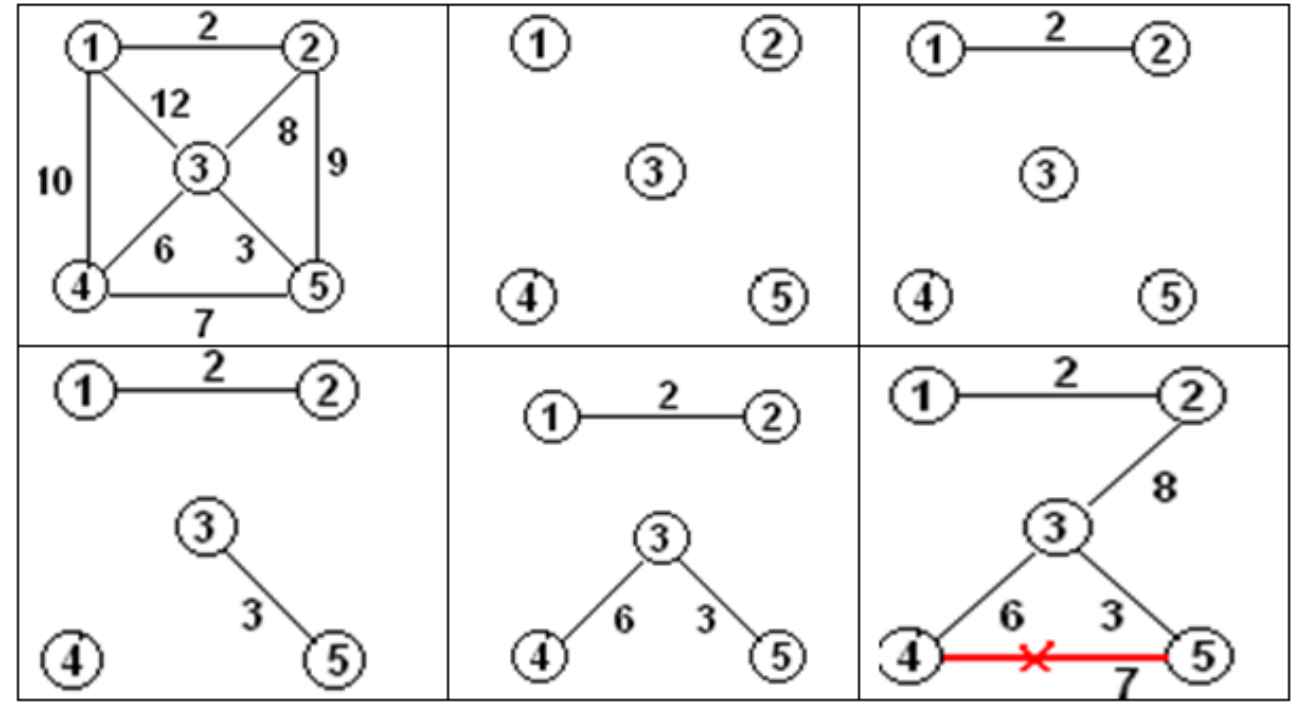
①将图G中的边按权值从小到大快排；

②按照权值从小到大依次选边。若当前选取的边加入后使生成树T形成环，则舍弃当前边；否则标记当前边并计数。

③重复②的操作，直到生成树T中包含n-1条边为止。否则当遍历完所有的边后，都不能选取n-1条边，表示最小生成树不存在。

Kruskal算法---算法步骤

下面给出了的**kruskal算法**的执行过程图：



算法的关键在于如何判定新加入的边会不会使图G'产生环，在这里使用并查集，如果新加入边的两个端点在并查集的同一个集合中，说明存在环，需要舍弃这条边；否则保留当前边，并合并涉及的两个集合。

用并查集优化后总的时间复杂度为O(mlogm+mα(n))

Kruskal算法---核心代码


```
bool cmp(const Edge &x,const Edge &y){return x.z<y.z;}
int Getfather(int x)//查找祖先
{   if(prt[x]==x)return x;
    prt[x]=Getfather(prt[x]);
    return prt[x];
}
int main()
{   cin>>n>>m; ans=0;bj=1;
   for(int i=1;i<=m;i++)cin>>a[i].x>>a[i].y>>a[i].z;
   sort(a+1,a+m+1,cmp);
   kruskal();
   if(bj)cout<<ans<<endl;
}
```

Kruskal算法---核心代码


```
void kruskal()      //kruskal核心程序
{   int f1,f2,k,i;
    k=0;
    for(i=1;i<=n;i++)prt[i]=i;//初始化
    for(i=1;i<=m;i++)
    {    f1=Getfather(a[i].x);f2=Getfather(a[i].y);
         if(f1!=f2)
         {    ans=ans+a[i].z;
              prt[f1]=f2;  //合并不相同的两个集合
              k++;
              if(k==n-1)break;
         }
    }
    if(k<n-1){cout<<"Impossible"<<endl;bj=0;return;}
}
```

【例1】修复公路 --1578

Description

A地区在地震过后，连接所有村庄的公路都造成了损坏而无法通车。政府派人修复这些公路。

给出A地区的村庄数N，和公路数M，公路是双向的。并告诉你每条公路的连着哪两个村庄，并告诉你什么时候能修完这条公路。问最早什么时候任意两个村庄能够通车，即最早什么时候任意两条村庄都存在至少一条修复完成的道路（可以由多条公路连成一条道路）

Input

第1行两个正整数N，M（N<=1000，M<=100000）

下面M行，每行3个正整数x, y, t，告诉你这条公路连着x,y两个村庄，在时间t时能修复完成这条公路。（x<=N，y<=N，t<=100000）

Output

如果全部公路修复完毕仍然存在两个村庄无法通车，则输出-1，否则输出最早什么时候任意两个村庄能够通车

【例1】修复公路 --1578

Sample Input

4 4

1 2 6

1 3 4

1 4 5

4 2 3

Sample Output

5

【例1】修复公路 --1578


```
struct Edge{ int x,y,z;  }a[200005];
int ans=-INT_MAX,bj=1,prt[200005]={0},n,m;
bool cmp(const Edge &x,const Edge &y){return x.z<y.z;}
int Getfather(int x)
{
        if(prt[x]==x) return x;
        prt[x]=Getfather(prt[x]);
        return prt[x];
}
int main()
{
        scanf("%d%d",&n,&m);
        for(int i=1;i<=m;i++)
                scanf("%d%d%d",&a[i].x,&a[i].y,&a[i].z);
        sort(a+1,a+m+1,cmp);
        kruskal();
        if(bj) cout<<ans<<endl;
}
```

【例1】修复公路 --1578


```
void kruskal()
{
        int f1,f2,k,i; k=0;
        for(i=1;i<=n;i++) prt[i]=i;
        for(i=1;i<=m;i++)
        {       f1=Getfather(a[i].x);
                f2=Getfather(a[i].y);
                if(f1!=f2)
                {       ans=max(ans,a[i].z);
                        prt[f1]=f2;
                        k++;
                        if(k==n-1)      break;
                }
        }
        if(k<n-1)
        {  cout<<"-1"<<endl;
                bj=0;  return;
        }
}
```

【例2】联络员 --1455

Description

Tyvj已经一岁了，网站也由最初的几个用户增加到了上万个用户，随着Tyvj网站的逐步壮大，管理员的数目也越来越多，现在你身为Tyvj管理层的联络员，希望你找到一些通信渠道，使得管理员两两都可以联络（直接或者是间接都可以）。Tyvj是一个公益性的网站，没有过多的利润，所以你要尽可能的使费用少才可以。

目前你已经知道，Tyvj的通信渠道分为两大类，一类是必选通信渠道，无论价格多少，你都需要把所有的都选择上；还有一类是选择性的通信渠道，你可以从中挑选一些作为最终管理员联络的通信渠道。数据保证给出的通行渠道可以让所有的管理员联通。

Input

第一行n，m表示Tyvj一共有n个管理员，有m个通信渠道;

第二行到m+1行，每行四个非负整数，p,u,v,w 当p=1时，表示这个通信渠道为必选通信渠道；当p=2时，表示这个通信渠道为选择性通信渠道；u,v,w表示本条信息描述的是u，v管理员之间的通信渠道，u可以收到v的信息，v也可以收到u的信息，w表示费用。

Output

最小的通信费用。

【例2】联络员 --1455

Sample Input

5 6
1 1 2 1
1 2 3 1
1 3 4 1
1 4 1 1
2 2 5 10
2 2 5 5

Sample Output

9

【样例解释】

1-2-3-4-1存在四个必选渠道，形成一个环，互相可以到达。需要让所有管理员联通，需要联通2号和5号管理员，选择费用为5的渠道，所以总的费用为9。

【注意】

U,v之间可能存在多条通信渠道，你的程序应该累加所有u,v之间的必选通行渠道

【数据范围】对于100%的数据，n<=2000 m<=10000

【例3】构造完全图(tree) --1579

Description

对于完全图 G, 若有且仅有一棵最小生成树为 T， 则称完全图 G 是树 T 的扩展出的。
给你一棵树 T, 找出 T 能扩展出的边权和最小的完全图 G 。

Input

第一行 N 表示树 T 的点数。
接下来 N-1 行： Si， Ti， Di； 描述一条边（ Si,Ti ）权值为 Di 。
保证输入数据构成一棵树。

Output

一个数，表示最小的图 G 的边权和。

Sample Input

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

Sample Output

```
12
```

【样例解释】 添加 D(2,3)=2,D(3,4)=3,D(2,4)=3 即可。

【 数据范围】 对于 100% 的数据， N<=100000 ， 1<=Di<=100000

【例3】构造完全图(tree) --1579


```
struct tree{int a,b,c;}t[100005];
int n,f[100005],s[100005];
bool cmp(const tree &x,const tree &y){return x.c<y.c;}
int Getfather(int x)//查找祖先
{  if(f[x]==x)return x;   f[x]=Getfather(f[x]);  return f[x];  }
int main()
{  scanf("%d",&n);
   for(int i=1;i<=n-1;i++)scanf("%d%d%d",&t[i].a,&t[i].b,&t[i].c);
   sort(t+1,t+n,cmp);
   for(int i=1;i<=n;i++){f[i]=i;s[i]=1;}
   long long ans=0;
   for(int i=1;i<=n-1;i++)
   {  int x=Getfather(t[i].a),y=Getfather(t[i].b);
      ans=ans+((long long)(s[x])*s[y]-1)*(t[i].c+1)+t[i].c;
      f[x]=y;
      s[y]+=s[x];
   }
   cout<<ans<<endl;
}
```