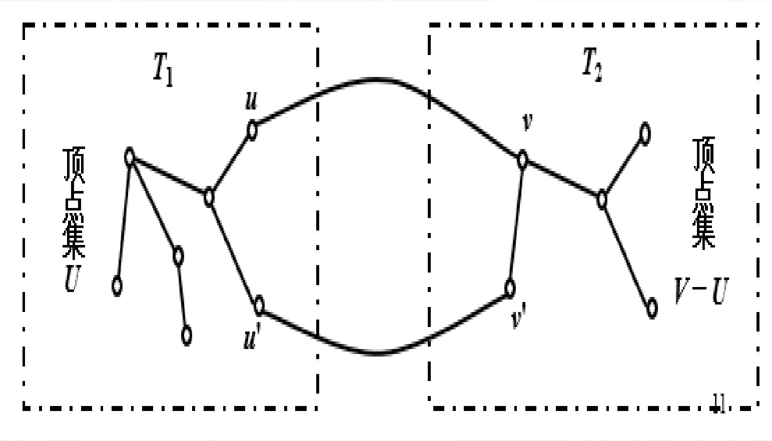
最小生成树(MST)问题的扩展

Prim算法的正确性证明

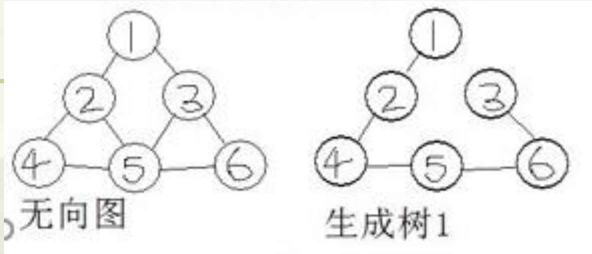
如果顶点集U中的所有已经选择的边的集合E全部是某个最小生成树T的边，而u和v是连接U和V-U的最短边，那么E+uv一定是某个最小生成树T’的边。

证明：反证法。
T的顶点被分成两部分，U和V-U。
假设T没有选择uv这条最短的边，我们在T中将uv连接，必然形成一个环，如图，假设u’v’是该环中的另一条边，其中u’在U中，v’在V-U中（这样的u’v’必然存在）。
如果len(u’v’)>len(uv)，我们将u’v’从T中去掉，剩下的生成树长度比T更小，矛盾。
所以只能有len(u’v’)=len(uv)，此时，我们将u’v’从T中去掉，得到一颗和T长度一样的最小生成树T’，且E和uv都在T’中，结论得证。



将图的顶点任意分成两部分，其中两部分之间最短的边长一定在**任意**最小生成树中.

最小生成树的任意一条边都不可能是图中某个环的最大边。



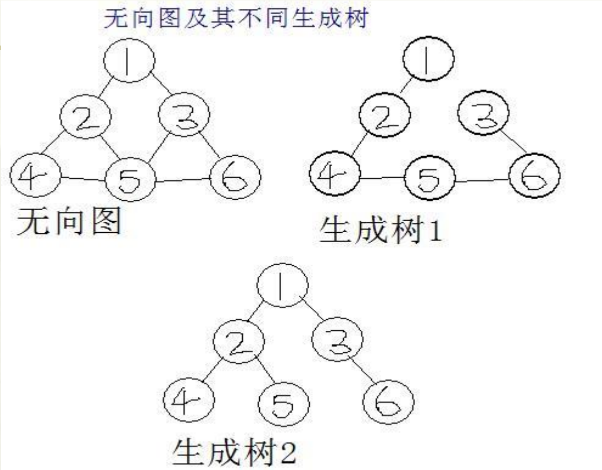
证明：反证法。

假设最小生成树(生成树1)中45是图G(无向图)中某环12453的最大边（比环中其他所有的边都严格大于）。

在最小生成树中，将45这条边去掉，最小生成树将分成124和563两部分。那么在图G的环12453中，一定有一条45之外的边可以连接124和563（因为45是在环中），上图中是13。

我们在最小生成树中连接13可以得到一颗新的生成树，比原来包含45的生成树更小（环中，边45比13更长），与原生成树为最小生成树矛盾矛盾。

最小生成树的最长边一定不大于其他生成树的最长边。



证明：反证法。

假设生成树1是最小生成树，生成树2是任意一颗生成树，生成树2的最长边的长度小于生成树1的最长边45的长度，从而生成树2中的每一条边都小于生成树1中边45的长度。

在生成树2中，加入边45必然形成一个环，在这个环中，边45是长度最长的，与生成树1为最小生成树矛盾（利用上一页的结论）。

推论：所有最小生成树的最长边的权值都是相等的。

权值为a的边在所有最小生成树中数量是相等的。

证明： 首先，假设按照Kruscal算法得到的最小生成树T的最长边为d,我们在原图G中删除所有长度大于或者等于d的边，得到G’，重新应用Kruscal算法，假设得到k（k>1）个连通分支的最小生成树T’，T与T’的差别就是T’没有边长为d的边, len(T)-len(T’)=（k-1）d。

一般地，对于图G的任意一颗最小生成树S，如果去掉S中所有长度为d的最长边，得到的也是G’的k个连通分支的最小生成树S’，由于 G’ 的 k 个 连 通 分 支 的 最 小 生 成 树 的 权 值 和 是 相 等 的 , 即 len(T’)=len(S’)，所以，len(S)-len(S’)= len(T)-len(T’)=（k-1）d，说明T和S中的最长边的数量均为k-1,是相等的。

将以上方法依次应用在G’的k个连通分支上，可得结论。

推论：假设图G的最小生成树T中权值大于a的边有k-1条，那么，从图G中去掉所有权值大于a的边，将会得到k个连通分支。

T是图G的任何一颗最小生成树，任何一颗生成树T0,可以通过如下一系列变换T0-> T1->...->Ts=T变成T。每次变换将Ti中的一条不属于T的边替换成T中的一条长度较小或相等的边，len(Ti)>=len(Ti+1)。

取Ti-> Ti+1的变换为：

- 从Ti中任取一条不属于T的边uv，断开uv，Ti必然被分成两部分。
- 在T中加入边uv,形成一个环，在该环中存在uv,之外的边u’v’，其中u’和v’分别在Ti不同的部分中。
- 可以看出，len(u’v’)<=len(uv),否则，与T为最小生成树矛盾
- 在Ti中连接两点u’ v’得到Ti+1.
- 以上过程将Ti中的边uv换成T中的边u’v’。

任意次优最小生成树都与某最小生成树相同或者仅相差一条边；反之任意一颗最小生成树，可以仅通过调整一条边变为次优最小生成树。

次优最小生成树是由最小生成树而来的，含义就是所有的生成树集合中，除去最小的那棵，剩下的集合中最小的生成树。

证明：

- 如果最小生成树不唯一，那么次优最小生成树也是最小生成树。
- 如果最小生成树唯一，那么次优最小生成树的权值小于最小生成树。取T0为一颗次优最小生成树，T为任意最小生成树，那么T0-> T1->…->Ts=T的变化中，只有唯一一个i，使得Ti的权值大于Ti+1，其中更换的边为uv到u'v'，T0中只有uv的权值与T中u'v'的权值不相等，在T0中，将uv换成u'v'得到一颗最小生成树。
- 反之亦然。

最小生成树的个数

分析：

最小生成树的两个性质：

同一个图的最小生成树，满足：

1）同一种权值的边的个数相等

2）用Kruscal按照从小到大，处理完某一种权值的所有边后，图的连通性相等

这样，

- 先做一次Kruscal求出每种权值的边的条数，
- 再按照权值从小到大，对每种边进行 DFS， 求出这种权值的边有几种选法。
- 最后根据乘法原理将各种边的选法数乘起来就可以了。

特别注意：在DFS中为了在向下DFS之后消除决策影响，恢复f[]数组之前的状态，在DFS中调用的Find()函数不能路径压缩。

最小生成树计数（洛谷-P4208）

```
struct Edge {
int x,y;
int dis;
bool operator < (const Edge &rhs)const{
return dis<rhs.dis;
}
} edge[N];
struct build {
int l,r;
int cnt;
} a[N];
int tot;
int n,m;
int father[N];
int sum;
int Find(int x) {//不要压缩路径
if(father[x]!=x)
return Find(father[x]);
return x;
}
```

```
int Kruskal() {
sort(edge+1,edge+m+1);
for(int i=1; i<=n; i++)
father[i]=i;
int cnt=0;
for(int i=1; i<=m; i++) {
if(edge[i].dis!=edge[i-1].dis) {
tot++;
a[tot].l=i;//第tot个权值的起始边
a[tot-1].r=i-1; //第tot个权值的结束边
}
int x=Find(edge[i].x);
int y=Find(edge[i].y);
if(x!=y) {
father[y]=x;
a[tot].cnt++;//第tot个权值在最小生成树中的边数，权值相等的边出现次数相等，用于判断dfs路径是否合法
cnt++;
}
}
a[tot].r=m;
return cnt;
}
```

```
void dfs(int x,int now,int num) {
//x:第几个权值，now:到第几条边，num:已经有几条边了。
if(now==a[x].r+1) {
if(num==a[x].cnt)//选指定的条数
sum++;
return ;
}
int fx=Find(edge[now].x);
int fy=Find(edge[now].y);
if(fx!=fy) {//选
father[fx]=fy;
dfs(x,now+1,num+1);
father[fx]=fx;//恢复并查集
father[fy]=fy;
}
dfs(x,now+1,num);//不选
}
```

```
int main() {
scanf("%d%d",&n,&m);
int x,y,z;
for(int i=1; i<=m; i++)
scanf("%d%d%d",&edge[i].x,&edge[i].y,&edge[i].dis);
int num=Kruskal();
if(num!=n-1) {
printf("0");
return 0;
}
else{
for(int i=1; i<=n; i++)
father[i]=i;
int res=1;
for(int i=1; i<=tot; i++) {
sum=0;
dfs(i,a[i].l,0);
res=res*sum%MOD;
for(int j=a[i].l; j<=a[i].r; j++) {
int x=Find(edge[j].x);
int y=Find(edge[j].y);
if(x!=y)
father[y]=x;
}
}
printf("%d",res);
}
return 0;
}
```

谢谢