最小生成树(MST)

什么是最小生成树问题?

我们给n个点m条边，选出n-1条边把所有点连起来（默认只有一块），使得边权之和最小，这就是最小生成树(Minimum Spanning Tree)问题。

该问题有两种比较常见的作法：kruskal和Prim

Kruskal

步骤

基于贪心思想，先把边从小到大排列，然后按顺序选边并且合并每条边相连的两点所在集合（若两点处于同一集合内，则该边不必再选），直到所有点处于同一集合为止。

前置知识

并查集

代码模板

```
    sort(e+1,e+1+cnt); //按边权从小到大排序
    for(int i=1;i<=cnt;i++){
        int u=e[i].from,v=e[i].to;
        if(find(u)!=find(v)){ //Kruskal算法，找到未合并的两点合并
            unite(u,v);
            mx=e[i].dis; //更新最大边
            num++;
            sum+=e[i].dis;
        }
        if(num==n-1) break; //可以在连了n-1条边时即使跳出
    }
```

Prim算法

步骤

从任意顶点开始，选择一个与当前顶点最近的一个顶点，连接两顶点的边加入集合中，然后将两顶点合并成一个点。重复上述操作至点都合并到了一个集合中。

代码模板

```
int prim(int graph[][MAX], int n)
{
    int lowcost[MAX];
    int mst[MAX];
    int i, j, min, minid, sum = 0;
    for (i = 2; i <= n; i++)
    {
        lowcost[i] = graph[1][i];
        mst[i] = 1;
    }
    mst[1] = 0;
    for (i = 2; i <= n; i++)
    {
        min = MAXCOST;
        minid = 0;
        for (j = 2; j <= n; j++)
        {
            if (lowcost[j] < min && lowcost[j] != 0)
            {
                min = lowcost[j];
                minid = j;
            }
        }
        cout << "V" << mst[minid] << "-V" << minid << "=" << min << endl;
        sum += min;
        lowcost[minid] = 0;
        for (j = 2; j <= n; j++)
```

```
        {
            if (graph[minid][j] < lowcost[j])
            {
                lowcost[j] = graph[minid][j];
                mst[j] = minid;
            }
        }
    }
    return sum;
```

次小生成树

非严格次小生成树求解方法

设最小生成树T的权值和为M

只需在未选择的边e=(u,v,w)替换原来连接u或v的边，可得到权值和为M'=M+w-w'的生成树T'。对所有替换得到的M'取最小值即可。

求u,v路径上的边权最大值

用倍增来维护，处理出每个节点的2^i级祖先以及到达2^i级祖先路径上最大的边权，这样在倍增求LCA的过程中就可以直接求得。

严格次小生成树求解

我们维护到2^i级祖先路径上的最大边权的同时维护严格次大边权，当用于替换的边权的权值与原生成树中路径最大边权相等时，我们用严格次大值来替换即可。

时间复杂度$O(mlogm)$

代码

```
//luguo P4180 [BJWC2010]严格次小生成树
#include<bits/stdc++.h>
#define int long long
#define INF 98765432109876543

using namespace std;
const int maxn=3e5+7;

inline void read(int &data){ //快读优化
    int x=0,f=1;char ch=getchar();
    while(ch<'0'||ch>'9'){
        if(ch=='-') f=f*-1;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9'){
        x=x*10+ch-'0';
        ch=getchar();
    }
    data=x*f;
}

struct node{
```

```
    int to,w,nxt;
}e[maxn<<2]; //这里存最小生成树

struct Node{
    int u,v,w;
    inline bool operator < (const Node &x)const{ //重定义<运算方便排序
        return w<x.w;
    }
}a[maxn]; //这是一开始的图

int n,m,cnt,head[maxn],fa[maxn],sum;
int dep[maxn];
int f[maxn][20]; //存储祖先信息
int g[maxn][20]; //存储最大距离
int h[maxn][20]; //存储次大距离
bool vis[maxn];

void addedge(int u,int v,int w){ //链式前向星存图
    e[++cnt]=(node){v,w,head[u]};
    head[u]=cnt;
}

int find(int x){ //并查集的查找操作
    return x==fa[x]?x:fa[x]=find(fa[x]);
}

void kruskal(){ //得到最小生成树
    int num=0,u,v,w,x,y;
    for(int i=1;i<=m;i++){
        u=a[i].u,v=a[i].v,w=a[i].w;
        x=find(u);y=find(v);
        if(x!=y){
            vis[i]=1; //表示这条边被选中
            num++;
            fa[x]=y; //合并两集合
            sum+=w; //记录最小生成树的权值和
            addedge(u,v,w); //构建最小生成树
            addedge(v,u,w);
            if(num==n-1) break; //边数=点数-1跳出
        }
    }
}

void dfs(int u,int fa,int w){ //倍增来维护每个点的信息
    dep[u]=dep[fa]+1; //u点的深度
    f[u][0]=fa; //u的上一级父亲
    g[u][0]=w; //到上一级祖先的边的最大值
    h[u][0]=-INF;//到上一级祖先的边的次大值
    for(int i=1;i<=18;i++){
        f[u][i]=f[f[u][i-1]][i-1];
        g[u][i]=max(g[u][i-1],g[f[u][i-1]][i-1]);
        h[u][i]=max(h[u][i-1],h[f[u][i-1]][i-1]);
        if(g[u][i-1]>g[f[u][i-1]][i-1]) h[u][i]=max(h[u][i],g[f[u][i-1]][i-1]);
//替换次大值
        else if(g[u][i-1]<g[f[u][i-1]][i-1]) h[u][i]=max(h[u][i],g[u][i-1]);
    }
    for(int i=head[u];i;i=e[i].nxt){
        int v=e[i].to,w=e[i].w;
```

```
        if(v==fa) continue;
        dfs(v,u,w);  //遍历下一个节点
    }
}

int LCA(int x,int y){ //lca求公共祖先
    if(dep[x]<dep[y]) swap(x,y); //x的深度比y大
    for(int i=15;i>=0;i--){
        if(dep[f[x][i]]>=dep[y]) x=f[x][i];  //拉近到同一深度
    }
    if(x==y) return x; //y是x的祖先，直接返回
    for(int i=15;i>=0;i--){
        if(f[x][i]!=f[y][i]) x=f[x][i],y=f[y][i];//同时向上跳
    }
    return f[x][0];
}

int get(int u,int v,int mx,int ans=-INF){
    for(int i=18;i>=0;i--){
        if(dep[f[u][i]]>=dep[v]){
            if(mx!=g[u][i]) ans=max(ans,g[u][i]); //比较ans和u跳2^i路径的最大值
            else ans=max(ans,h[u][i]);//比较ans和u跳2^i路径的次大值
            u=f[u][i]; //往上跳
        }
    }
    return ans;
}

void output(){
    int u,v,w,lca,x,y,ans=INF;
    for(int i=1;i<=m;i++){
        if(vis[i]) continue; //如果是最小生成树上的边直接跳过
        u=a[i].u,v=a[i].v,w=a[i].w,
        lca=LCA(u,v); //找到这两点的LCA
        x=get(u,lca,w); //获取u到lca的最大边权
        y=get(v,lca,w);
        ans=min(ans,sum-max(x,y)+w); //最小生成树权值和-所选最大边+新边
    }
    for(int i=1;i<=m;i++){
        if(vis[i]) continue;
        u=a[i].u,v=a[i].v,w=a[i].w,lca=LCA(u,v);
        x=get(u,lca,w);
        y=get(v,lca,w);
        ans=min(ans,sum-max(x,y)+w);
    }
    printf("%lld",ans);
}

signed main(){
    read(n);read(m);
    for(int i=1;i<=n;i++) fa[i]=i; //一开始父亲指向自己
    for(int i=1;i<=m;i++){
        read(a[i].u);read(a[i].v);read(a[i].w);
    } //存图
    sort(a+1,a+1+m); //对边排序
    kruskal(); //最小生成树
    dfs(1,0,0); //以1为根遍历这棵树
    output();
```

```
    return 0;
}
```

最小生成树计数

根据最小生成树的两个性质：

（1）不同的最小生成树中，每种权值的边出现的个数都是确定的。

（2）不同的生成树中，某一种权值的边连接完成后，形成的连通块状态是一样的。

那么我们可以把每种权值的处理看成是分开的好几步，将得到的结果相乘。

将已经计算好的连通块缩成一个点，那么就变成了一个独立的图的生成树问题，可以用矩阵树定理来求解

矩阵树定理(Kirchhoff's Matrix Tree Theorem)

声明：无论有向无向，都允许重边，不允许自环。

解决了：一张图中生成树计数问题

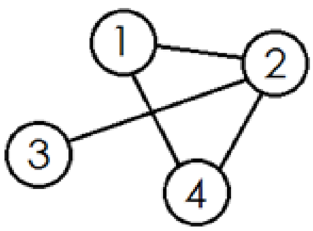
前置知识：线性代数

图的生成树数量等于调和矩阵的行列式。

矩阵-树定理(matrix-tree theorem)是一个计数定理.若连通图G的邻接矩阵为A，将A的对角线(i,i)元素依次换为节点i的度d(i)，其余元素(i,j) (j!=i) 取Aij的相反数，所得矩阵记为M，则M的每个代数余子式相等，且等于G的生成树的数目.这就是矩阵一树定理.我们常常称矩阵M为基尔霍夫矩阵。

基尔霍夫Kirchhoff矩阵 K =度数矩阵 D - 邻接矩阵 A

- 对于一个无向图 G ，它的生成树个数等于其基尔霍夫Kirchhoff矩阵任何一个N-1阶主子式的行列式的绝对值。
- 已经得出了基尔霍夫Kirchhoff矩阵，那么随便去掉某一行一列并计算出新矩阵的行列式，其绝对值即为生成树个数。



$$
\begin{array}{c|cccc|}
D & 1 & 2 & 3 & 4 \\ \hline
1 & 2 & 0 & 0 & 0 \\
2 & 0 & 3 & 0 & 0 \\
3 & 0 & 0 & 1 & 0 \\
4 & 0 & 0 & 0 & 2 \\ \hline
\end{array}
\;-\;
\begin{array}{c|cccc|}
A & 1 & 2 & 3 & 4 \\ \hline
1 & 0 & 1 & 0 & 1 \\
2 & 1 & 0 & 1 & 1 \\
3 & 0 & 1 & 0 & 0 \\
4 & 1 & 1 & 0 & 0 \\ \hline
\end{array}
\;=\;
\begin{array}{c|cccc|}
K & 1 & 2 & 3 & 4 \\ \hline
1 & 2 & -1 & 0 & -1 \\
2 & -1 & 3 & -1 & -1 \\
3 & 0 & -1 & 1 & 0 \\
4 & -1 & -1 & 0 & 2 \\ \hline
\end{array}
$$

算法思想

利用图的Kirchhoff矩阵，可以在$O(n^3)$时间内求出生成树的个数。

行列式计算（模板）

通过行列式的变换，复杂度可以优化到$O(n^3)$

```
    for(int i=1;i<cnt;i++){  //下面是把矩阵转化为上三角的形式
        int j;
        for(j=i;j<cnt;j++){
            if(sq[j][i]) break;
        }
        if(j==cnt) continue; //这一行全为0
        if(j!=i){
            for(int k=0;k<=cnt;k++) swap(sq[i][k],sq[j][k]); //对两行进行交换
            ans*=-1; //行列式的值取相反数
        }
        for(j=i+1;j<cnt;j++){ //上三角（对同一列其他行的元素化0处理）
            while(sq[j][i]){
                int t=sq[j][i]/sq[i][i];
                for(int k=i;k<cnt;k++){
                    sq[j][k]=(sq[j][k]-sq[i][k]*t%mod+mod)%mod; //更新该行
                }
                if(!sq[j][i]) break; //对角化0
                for(int k=i;k<cnt;k++){
                    swap(sq[i][k],sq[j][k]); //交换两行
                }
                ans*=-1; //取反
            }
        }
    }
    for(int i=1;i<cnt;i++){
        ans*=sq[i][i]; //行列式计算：乘对角元素
        ans%=mod;
    }
```

建议去看高斯-约旦消元的模板。

普通做法题解

我们需要注意到最小生成树的两个性质：

（1）不同的最小生成树中，每种权值的边出现的个数是确定的；

（2）不同的生成树中，某一种权值的边连接完成后，形成的联通块状态是一样的。

于是可以把每种权值的处理看成是分开的好几步，将结果逐一相乘。

1.做一遍最小生成树，统计每种权值的边使用的情况；

2.对每种边权的边搜索，得出每一种权值的边选择情况；

3.乘法原理

```
#include <bits/stdc++.h>
using namespace std;
const int N=1e3+7;
```

```
struct node{
    int x,y,z;
}a[N],b[N];

int cmp(node a, node b){
    return a.z<b.z;   //边从小到大排序
}

int n,m,t;
int f[N],d[N],c[N];
int cnt,ans,xx,yy;

int find(int x){ //并查集的查找操作
    if(f[x]==x){
        return x;
    }
    return f[x]=find(f[x]);
}

void dfs(int now,int k,int x){ //now表当前位置，k表示加入边数，x表权值种类在d数组中位置
    if(now>b[x].y){ //如果搜过右端点
        if(k==d[x]){
            cnt++;    //符合情况则+1
        }
        return;
    }
    int p[101];
    for(int i=1;i<=n;i++){
        p[i]=f[i];    //存储集合情况
    }
    xx=find(a[now].x);
    yy=find(a[now].y);
    if(xx!=yy){ //如果可以加入最小生成树
        f[xx]=yy;
        dfs(now+1,k+1,x); //继续搜索下一个
    }
    for(int i=1;i<=n;i++){
        f[i]=p[i];    //还原集合情况（回溯）
    }
    dfs(now+1,k,x); //找下一个位置
}

signed main(){
    scanf("%d%d",&n,&m);
    for (int i=1;i<=m;i++) { //输入
        scanf("%d%d%d",&a[i].x,&a[i].y,&a[i].z);
    }
    sort(a+1,a+1+m,cmp); //对边进行排序
    for (int i = 1; i <= n; i++)  f[i]=i;
    a[0].z=-INT_MAX;
    t=0;
    for(int i=1;i<=m;i++){
        if(a[i].z==a[i-1].z){ //搜索同一权值的左右位置，方便搜索
            b[t].y++;
            c[i]=t;
        }else{
            t++;
```

```
                b[t].x=i;
                b[t].y=i;
                c[i]=t; //x表示左端点，t表示权值种数
            }
        }
        cnt=0;
        for (int i=1;i<=m;i++){
            xx=find(a[i].x);
            yy=find(a[i].y); //寻找最小生成树
            if(xx!=yy){
                f[xx]=yy;
                d[c[i]]++; //d存储该权值需要的边数
                cnt++;
            }
            if(cnt==n-1) break;
        }
        if(cnt!=n-1){
            printf("0");     //没找到最小生成树
            return 0;
        }
        for (int i=1;i<=n;i++) f[i]=i;
        ans=1;
        for(int i=1;i<=t;i++)
            if(d[i]>0){ //表示存在这个权值
                cnt=0;
                dfs(b[i].x,0,i); //搜索
                ans=(ans*cnt)%31011;
                for(int j=b[i].x;j<=b[i].y;j++){
                    xx=find(a[j].x);
                    yy=find(a[j].y);
                    if (xx!=yy){
                        f[xx]=yy; //合并
                    }
                }
            }
        printf("%d\n", ans);
        return 0;
}
```

其他待更新

Kruskal重构树

最小瓶颈路

瓶颈生成树

Boruvka算法

可能会再出一章，因为目前还不是很必要学。

参考资料

https://blog.csdn.net/yeruby/article/details/38615045

https://blog.csdn.net/clover_hxy/article/details/69397184

https://www.cnblogs.com/zj75211/p/8039443.html

https://www.cnblogs.com/yangsongyi/p/10697176.html

https://www.cnblogs.com/zj75211/p/8039443.html

OI-wiki

百度百科