

矩阵树定理

算法简介

一种生成树的计数定理，时间复杂度 $O(n^3)$

算法实现

无向图

定义生成树的权值为所有该生成树中所有边权的乘积，则有如下结论：

邻接矩阵 D 中 $d_{i,i} = \sum_{j \neq i} w_{i,j}$ 所有与节点 i 相连的边的权值和

邻接矩阵 L 中 $d_{i,j} = \text{edge}[i][j].w$ (注意无向图中 $d_{i,j} = d_{j,i}$)

记基尔霍夫矩阵 $K = D - L$ 为 K 去掉第 i 行与第 i 列得到的余子式(i 可以任取)。

则有 $\det(K') = \sum_{T \text{ 是生成树}} \prod_{e \in T} w_e$ 所有生成树的权值和。特别地，当所有边权为 1 时所有生成树的权值和等于生成树个数。

有向图

邻接矩阵 L 定义不变(但要注意边的有向性)。

如果邻接矩阵 D 中 $d_{i,i} = \sum_{j} w_{j,i}$ 节点 i 的所有入边的权值和。

记 K' 为 K 去掉第 i 行与第 i 列得到的余子式，则 $\det(K') = \sum_{T \text{ 是外向树}} \prod_{e \in T} w_e$ 所有以节点 i 为根的外向树(边从根指向叶子节点)的权值和。

如果邻接矩阵 D 中 $d_{i,i} = \sum_{j} w_{i,j}$ 节点 i 的所有出边的权值和。

记 K' 为 K 去掉第 i 行与第 i 列得到的余子式，则 $\det(K') = \sum_{T \text{ 是内向树}} \prod_{e \in T} w_e$ 所有以节点 i 为根的内向树(边从叶子节点指向根)的权值和。

代码模板

洛谷p6178

给定一个 n 个节点 m 条带权边的图，输入 t 表示图是否为有向图。

求其所有不同生成树的权值之和(如果是有向图，则求以 1 为根的外向树)，对 10^9+7 取模。

```
const int MAX_size=305,mod=1e9+7;
struct Matrix{
    int ele[MAX_size][MAX_size];
```

```
};
int Inv(int x,int p){
    int ans=1,base=x,k=p-2;
    while(k){
        if(k&1)
            ans=1LL*ans*base%p;
        base=1LL*base*base%p;
        k>>=1;
    }
    return ans;
}
int det(Matrix a,int n,int mod){
    int ans=1;
    _rep(i,2,n){
        int pos=i;
        _rep(j,i,n)if(a.ele[j][i]){pos=j;break;}
        if(!a.ele[pos][i])return 0;
        if(pos!=i){_rep(j,i,n) swap(a.ele[i][j],a.ele[pos][j]);ans=mod-ans;}
        ans=1LL*ans*a.ele[i][i]%mod;
        int k=Inv(a.ele[i][i],mod);
        _rep(j,i,n)a.ele[i][j]=1LL*a.ele[i][j]*k%mod;
        _rep(j,i+1,n)for(int k=n;k>=i;k--
            a.ele[j][k]=(a.ele[j][k]-1LL*a.ele[j][i]*a.ele[i][k])%mod;
        }
        return (ans+mod)%mod;
    }
}
int main()
{
    int n=read_int(),m=read_int(),t=read_int(),u,v,w;
    Matrix x;
    mem(x.ele,0);
    if(t==0){
        while(m--){
            u=read_int(),v=read_int(),w=read_int();
            if(u==v)continue;
            x.ele[u][u]=(x.ele[u][u]+w)%mod,x.ele[v][v]=(x.ele[v][v]+w)%mod;
            x.ele[u][v]=(x.ele[u][v]-w)%mod,x.ele[v][u]=(x.ele[v][u]-w)%mod;
        }
    }
    else{
        while(m--){
            u=read_int(),v=read_int(),w=read_int();
            if(u==v)continue;
            x.ele[u][v]=(x.ele[u][v]-w)%mod,x.ele[v][v]=(x.ele[v][v]+w)%mod;
        }
    }
    enter(det(x,n,mod));
}
```

```

return 0;
}

```

算法练习

习题一

[洛谷p3317](#)

题意

给定一个 n 个点的完全图，表示 n 个城市，该地区经过了一场洪水，城市之间的道路受损。

输入一个 $n \times n$ 矩阵，矩阵元素 $p_{i,j}$ 表示城市 i,j 之间道路依然连通的概率。

问经过洪水后该地区所有道路恰好构成一棵树的概率。

输入保证 $p_{i,j}=p_{j,i}, p_{i,i}=0$

题解

若将 $p_{i,j}$ 作为边 i,j 的权值套用矩阵树定理，设 E 为总边集 T 为生成树边集，则有

$$P = \sum_T \prod_{e \in E-T} (1-p_e) \prod_{e \in T} p_e = \prod_{e \in E} (1-p_e) \sum_T \prod_{e \in T} \frac{p_e}{1-p_e}$$

最后关于 $p_e=1$ 的情况，可以考虑缩点，或者令 $p_e=1-\varepsilon$

```

const int MAX_size=55;
const double eps=1e-8;
struct Matrix{
    double ele[MAX_size][MAX_size];
};
double det(Matrix a,int n){
    double ans=1.0;
    _rep(i,2,n){
        int pos=i;
        _rep(j,i+1,n)if(fabs(a.ele[j][i])>fabs(a.ele[pos][i]))pos=j;
        if(fabs(a.ele[pos][i])<eps)return 0.0;
        if(pos!=i){_rep(j,i,n)swap(a.ele[i][j],a.ele[pos][j]);ans=-ans;}
        ans*=a.ele[i][i];
        for(int j=n;j>=i;j--)a.ele[i][j]/=a.ele[i][i];
        _rep(j,i+1,n)for(int k=n;k>=i;k--)
            a.ele[j][k]=a.ele[j][k]-a.ele[j][i]*a.ele[i][k];
    }
    return ans;
}

```

```
}  
double a[MAX_size][MAX_size];  
int main()  
{  
    int n=read_int();double ans=1.0;  
    Matrix x;  
    mem(x.ele,0);  
    _rep(i,1,n)  
    _rep(j,1,n){  
        scanf("%lf",&a[i][j]);  
        if(fabs(1.0-a[i][j])<eps)  
            a[i][j]=1.0-eps;  
        if(i>j)  
            ans*=1.0-a[i][j];  
            a[i][j]/=1.0-a[i][j];  
    }  
    _rep(i,1,n)  
    _rep(j,1,n)  
    x.ele[i][j]-=a[i][j],x.ele[j][j]+=a[i][j];  
    printf("%lf",ans*det(x,n));  
    return 0;  
}
```

习题二

[洛谷p4208](#)

题意

现在给出了一个简单无向加权图，求这个图中有多少个不同的最小生成树，结果对 31011 的模。

性质

1. 如果 A, B 都是 G 的最小生成树，则将 A, B 中所有边权从小到大排序，将得到相同结果。
2. 如果 A, B 都是 G 的最小生成树，则删去 A, B 中权值超过 w 的边后 A, B 图连通性完全相同 (w 取值任意)

证明

对性质一，将 A, B 中所有边按边权从小到大排序，得 a_1, a_2, \dots, a_n 和 b_1, b_2, \dots, b_n

考虑 A, B 第一次出现边不相同的位置，有 $a_i \neq b_i$ 不妨设 $w(a_i) \geq w(b_i)$

情况一：存在 $a_j = b_i$ 则有 $j > i, w(b_i) = w(a_j) \geq w(a_i) \geq w(b_i)$

所以有 $w(a_i) = w(b_i) = w(a_j)$ 交换 a_i, a_j 序列 $\{w(a)\}$ 不改变。

情况二：不存在 $a_j = b_i$ 考虑把 b_i 加入 A 得到一个环，由于 A 是最小生成树，故环上所有边权不超过 $w(b_i)$

同时环上一定有某条边不属于 B 否则 B 上存在环，不妨记这条边为 a_j

则有 $w(b_i) \geq w(a_j) \geq w(a_i) \geq w(b_i)$ 所以有 $w(a_i) = w(b_i) = w(a_j)$

考虑把边 a_j 换成 b_i 然后交换 a_i, a_j 的在序列中位置，序列 $\{w(a)\}$ 不改变。

最后总有序列 $\{a\}, \{b\}$ 完全相同，于是有初始的 $\{w(a)\}$ 与 $\{w(b)\}$ 完全相同，证毕。

对性质二，只能给出不太严谨的证明。考虑 Kruskal 算法过程。

由于 Kruskal 中权值相同的边排序任意，说明按任意顺序考虑权值相同的边，都不会影响后续结果。

所有相同权值的边选取结束后，图的连通性必然相同，证毕。

题解

从小到大考虑每个不同的边权。考虑某个边权 w 记所有边权相同的边为边集 E_w

根据性质二，所有边权小于 w 的边选择完成后图的连通性是确定的。于是对选择完成后的图进行缩点，然后计算从边集 E_w 中选边的合法方案。

计算合法方案时发现即使选择完边集 E_w 中的边后也不能保证图是树，所以如果直接使用矩阵树定理将返回 0 。

于是考虑添加一些虚边保证合法选取 E_w 中的边后图形一定是树。

由于合法选取 E_w 中的边后图的连通性也是确定的，所以可以考虑在合法选取 E_w 中的边后图中加入一些虚边使得图恰好连通。

发现将任意某个最小生成树中所有权值大于 w 作为虚边加入图中恰好能满足条件，然后使用矩阵树定理即可计算方案数。

最终答案即为所有不同的边权的选择方案的乘积。

但题目给定的 31011 并不是素数，所以计算行列式的消元过程中不能直接计算逆元。

考虑在消元过程中模拟辗转相除法。但辗转相除法只对两个数操作，而消元需要对两行所有数操作。计算行列式复杂度变为 $O(n^3 \log v)$

考虑在辗转相除过程中维护系数，即维护 $i' = ai + bj, j' = ci + dj$ 可以将计算行列式复杂度降为 $O(n^3)$

最后发现每次消元中 $n =$ 连通块个数 $=$ 最小生成树 T 中边权为 w 的边的个数 cnt_w 于是总时间复杂度为

$$O\left(\sum_{w \in T} \text{cnt}_w^3\right) = O(|T|^3) = O(n^3)$$

```
const int MAX_size=105,MAXN=105,MAXM=1e3+5,mod=31011;
struct Matrix{
    int ele[MAX_size][MAX_size];
```

```
};  
int sp_gcd(int a,int b,pair<int,int> &x,pair<int,int> &y){  
    int sign=1;  
    x.first=1,x.second=0,y.first=0,y.second=1;  
    while(b){  
        x.first=(x.first-a/b*y.first)%mod;  
        x.second=(x.second-a/b*y.second)%mod;  
        a%=b;  
        swap(x,y);  
        swap(a,b);  
        sign=-sign;  
    }  
    return sign;  
}  
int det(Matrix a,int n,int mod){  
    int ans=1;  
    _rep(i,2,n){  
        int pos=i;  
        _rep(j,i,n)if(a.ele[j][i]){pos=j;break;}  
        if(!a.ele[pos][i])return 0;  
        if(pos!=i){_rep(j,i,n) swap(a.ele[i][j],a.ele[pos][j]);ans=mod-  
ans;}  
        pair<int,int> x,y;  
        int t1,t2;  
        _rep(j,i+1,n){  
            ans*=sp_gcd(a.ele[i][i],a.ele[j][i],x,y);  
            _rep(k,i,n){  
                t1=a.ele[i][k],t2=a.ele[j][k];  
                a.ele[i][k]=(t1*x.first+t2*x.second)%mod;  
                a.ele[j][k]=(t1*y.first+t2*y.second)%mod;  
            }  
        }  
        ans=ans*a.ele[i][i]%mod;  
    }  
    return (ans+mod)%mod;  
}  
struct Edge{  
    int u,v,w;  
    bool operator < (const Edge &b)const{  
        return w<b.w;  
    }  
}edge[MAXM];  
bool vis[MAXM];  
int p[MAXN],block_id[MAXN],block_cnt;  
int Find(int x){return x==p[x]?x:p[x]=Find(p[x]);}  
int main()  
{  
    int n=read_int(),m=read_int(),u,v,w;  
    Matrix X;  
    _rep(i,1,m)
```

```

edge[i].u=read_int(),edge[i].v=read_int(),edge[i].w=read_int();
sort(edge+1,edge+1+m);
_rep(i,1,n)
p[i]=i;
block_cnt=n;
for(int i=1;i<=m&&block_cnt>1;i++){
    int x=Find(edge[i].u),y=Find(edge[i].v);
    if(x!=y)
        p[x]=y,vis[i]=true,block_cnt--;
}
if(block_cnt>1){
    puts("0");
    return 0;
}
int ans=1;
for(int pos1=1,pos2=1;pos1<=m;pos1=pos2){
    _rep(i,1,n)p[i]=i,block_id[i]=0;
    _rep(i,1,m){
        if(vis[i]&&edge[i].w!=edge[pos1].w)
            p[Find(edge[i].u)]=Find(edge[i].v);
    }
    block_cnt=0;
    _rep(i,1,n){
        if(!block_id[Find(i)])block_id[Find(i)]=++block_cnt;
        block_id[i]=block_id[Find(i)];
    }
    _rep(i,1,block_cnt)_rep(j,1,block_cnt)X.ele[i][j]=0;
    while(edge[pos2].w==edge[pos1].w){
        u=block_id[edge[pos2].u],v=block_id[edge[pos2].v];
        X.ele[u][v]--;X.ele[v][u]--;
        X.ele[u][u]++;X.ele[v][v]++;
        pos2++;
    }
    ans=ans*det(X,block_cnt,mod)%mod;
}
enter(ans);
return 0;
}

```

From: <https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link: https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string;jxm2001:%E7%9F%A9%E9%98%B5%E6%A0%91%E5%AE%9A%E7%90%86&rev=1595751465

Last update: 2020/07/26 16:17