线性代数

线代基本知识

余子式为去除某些行和列后剩余的矩阵部分的行列式M_{ij}

代数余子式为$A_{ij} = (-1)^{i+j} M_{ij}$

伴随矩阵为$A^* = \begin{vmatrix} A_{11} \dots A_{1m} \\ \vdots \qquad \vdots \\ A_{n1} \dots A_{nm} \end{vmatrix}$

（1） A 可逆当且仅当 A^* 可逆；

（2）如果 A 可逆，则 $A^* = |A| A^{-1}$ ；

（3）对于 A^* 的秩有：

$rank(A^*) = n, rank(A) = n$

$rank(A^*) = 1, rank(A) = n-1$

$rank(A^*) = 0, rank(A) < n-1$

（4） $|A^*| = |A|^{n-1}$ ；

（5） $(kA)^* = k^{n-1} A^*$ ；

（6）若 A 可逆，则 $(A^{-1})^* = (A^*)^{-1}$ ；

（7） $(A^T)^* = (A^*)^T$ ；

（8） $(AB)^* = B^* A^*$ 。

（9）AA* = A*A = |A|E

引理 一个n阶行列式，如果其中第i行所有元素除 a_{ij} 外都为零，那么这行列式等于 a_{ij} 与它的代数余子式的乘积。

$$D=\begin{vmatrix} a_{11} & \cdots & a_{1j} & \cdots & a_{1n} \\ \vdots & & \vdots & & \vdots \\ 0 & \cdots & a_{ij} & \cdots & 0 \\ \vdots & & \vdots & & \vdots \\ a_{n1} & \cdots & a_{nj} & \cdots & a_{nn} \end{vmatrix}=a_{ij}A_{ij}$$

定理3.1 行列式等于它的任一行(列)的各元素与其对应的代数余子式乘积之和。即

$$D=a_{i1}A_{i1}+a_{i2}A_{i2}+\cdots+a_{in}A_{in}\quad(i=1,2,\cdots,n)\ ,$$

或

$$D=a_{1j}A_{1j}+a_{2j}A_{2j}+\cdots+a_{nj}A_{nj}\quad(j=1,2,\cdots,n)$$

推论 行列式某一行(列)的元素与另一行(列)的对应元素的代数余子式乘积之和等于零，即

$$a_{i1}A_{j1}+a_{i2}A_{j2}+\cdots+a_{in}A_{jn}=0\quad(i\neq j)\ ,$$

或

$$a_{1i}A_{1j}+a_{2i}A_{2j}+\cdots+a_{ni}A_{nj}=0\quad(i\neq j)$$

范德蒙德(Vandermonde)行列式

$$D_n=\begin{vmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & \vdots & & \vdots \\ x_1^{n-1} & x_2^{n-1} & \cdots & x_n^{n-1} \end{vmatrix}=\prod_{n\geq i\geq j\geq 1}(x_i-x_j)$$

高斯消元

求多元一次方程

```
const double eps=1e-7;
double a[N][N],b[N];
void Gauss(int n){//b为增广矩阵，a为系数矩阵，得出b为结果
    int r;
    for(int i=0;i<n;++i){
        //数据稳定性优化
        r=i;
        for(int j=i+1;j<n;++j)
            if(fabs(a[j][i])>fabs(a[r][i]))
                r=j;
        if(fabs(a[r][i])<eps)return;//无解
        if(r!=i){
            for(int j=0;j<n;++j)swap(a[r][j],a[i][j]);
            swap(b[r],b[i]);
        }
        //消元
        for(int j=n;j>=i;j--)
            for(int k=i+1;k<n;++k)
                if(j==n) b[k]-=a[k][i]/a[i][i]*b[i];
                else a[k][j]-=a[k][i]/a[i][i]*a[i][j];
    }
    //回代
    for(int i=n-1;i>=0;--i){
        if(fabs(a[i][i])<eps)continue;
        for(int j=i+1;j<n;++j)b[i]-=a[i][j]*b[j];
        b[i]/=a[i][i];//最后b有0则无穷解，行列式答案就是b[i]之积
    }
}
```

行列式

```
double det(int n){
    double eps=1e-8;
    double ans=1;
    for(int i=1;i<=n;i++){
        int mx=i;
        for(int j=i+1;j<=n;j++){
            if(fabs(a[j][i])>fabs(a[mx][i]))mx=j;
        }
        if(mx!=i)for(int j=1;j<n;j++)swap(a[i][j],a[mx][j]);
        for(int k=i+1;k<=n;k++){
            double mul=a[k][i]/a[i][i];
            for(int j=i;j<=n;j++){
                a[k][j]-=a[i][j]*mul;
            }
        }
        if(fabs(a[i][i])<eps){
            return 0;
        }
    }
    for(int i=1;i<=n;i++)ans*=a[i][i];
    return fabs(ans);
}
```

```
int det(int n){
    int res=1;
    for(int i=1;i<=n;i++){//枚举主对角线上第i个元素
        for(int j=i+1;j<=n;j++){//枚举剩下的行
            while(A[j][i]){//辗转相除
                int t=A[i][i]/A[j][i];
                for(int k=i;k<=n;k++)//转为倒三角
                    A[i][k]=(A[i][k]-t*A[j][k]+mod)%mod;
                swap(A[i],A[j]);//交换i、j两行
                res=-res;//取负
            }
        }
        res=(res*A[i][i])%mod;
    }
    return (res+mod)%mod;
}
```

矩阵求逆

求a的逆矩阵

```
int A[N][N*2];
int qpow(int x,int y){
    int ans=1;
    while (y){
        if (y&1)ans=ans*x%mod;
        y>>=1;x=x*x%mod;
    }return ans;
}
void S(){
    m=2*n;//矩阵的宽
```

```
    for (int i=1;i<=n;i++){
        A[i][i+n]=1;//后面要跟上一个n阶单位矩阵
    }
    for (int i=1;i<=n;i++){//高斯-若尔当消元的板子
        int place=i;
        for (int j=i+1;j<=n;j++){//找到绝对值最大的元素开始消元
            if(abs(A[j][i])>abs(A[place][i]))place=j;
        }
        if (i!=place)swap(A[i],A[place]);
        if(!A[i][i]){//如果某行没有主元则A无法化为单位矩阵，无解
            printf("No Solution");return;
        }
        long long inv=qpow(A[i][i],mod-2);//本题加入的逆元特色
        for (int j=1;j<=n;j++){
            if(j!=i){
                long long multiple=A[j][i]*inv%mod;//等价于除以A[i][i]，消去其他行在第i列上的数，使之变成简化阶梯形矩阵
                for (int k=i;k<=m;k++){
                    A[j][k]=((A[j][k]-A[i][k]*multiple)%mod+mod)%mod;
                }
            }
        }
        for (int j=1;j<=m;j++)A[i][j]=(A[i][j]*inv%mod);//由于此处需要简化阶梯型矩阵，要把原矩阵化为简化矩阵的必须操作。
        //“在使用高斯-若尔当消元的时候，计算机计算的时候通常采用回带法，而人操作的时候建议采用此法。”——《线性代数及其应用》
    }
    for (int i=1;i<=n;i++){
        for (int j=n+1;j<=2*n;j++){
            printf("%d ",A[i][j]);
        }puts("");
    }
}
```

例题：给定$p[i][j]$概率数组，代表从i走到j的概率，当从i走到i时，人会停止。求$dp[i][j]$数组，代表从i开始走，j停止的概率。

$i \neq j$时$, dp[i][j] = \sum_{k\neq i}(p[i][k]dp[k][j])$

$i = j$时$, dp[i][j] = \sum_{k\neq i}(p[i][k]dp[k][j]) + p[i][j]$

令dp数组为A，则$A = p_1A + p_2$，p_1是p数组去除对角线，p_2是p数组对角线数组

则$A = (1 - p_1)^{-1}p_2$

异或高斯消元

```
#include<bits/stdc++.h>
using namespace std;
const int N=200;
int A[N][N],B[N];//B记录第i个变量的行号，无则=0
void Guass(int n,int m){//n变量 m条方程 n+1的位置为方程右边的值
    int i = 1, j = 1, k, r, c;
    while (i <= m && j <= n){
        r = i;
        for (k = i + 1; k <= m; k++)if (A[k][j] > A[r][j])r = k;
        if (A[r][j]){
            for (c = 1; c <= n + 1; c++)swap(A[i][c], A[r][c]);
```

```
            for (k = i + 1; k <= m; k++){
                if (A[k][j]){
                    for (c=j;c<=n+1;c++)A[k][c]^=A[i][c];
                }
            }
            B[j]=i;
            i++;
        }
        j++;
        B[j]=0;
    }
    //i-1为矩阵的秩
    /* //满秩时下式可以求出答案，否则要枚举自由变量确定值
    for (;i>=1;i--){
        for (j=1;j<i;j++){
            if (A[j][i]==1){
                for (k=i;k<=m;k++){
                    A[j][k]^=A[i][k];
                }
            }
        }
    }*/
}
int n,m;
int ans=0x3f3f3f3f;
int l[N];
void dfs(int x,int num){//x为当前的变量下标
    if(num>=ans)return;//剪枝
    if(x==0){ans=min(ans,num);return;}
    int t=B[x];
    if(t){//不是自由元
        l[x]=A[t][n+1];
        for(int i=x+1;i<=n;++i)if(A[t][i])l[x]^=l[i];
        dfs(x-1,num+l[x]);
    }
    else{//枚举自由变量
        l[x]=0;dfs(x-1,num+l[x]);
        l[x]=1;dfs(x-1,num+l[x]);
    }
}
signed main(){
    scanf("%d%d",&n,&m);
    for (int i=1;i<=n;i++)A[i][i]=A[i][n+1]=1;
    for (int i=1;i<=m;i++){
        int x,y;
        scanf("%d%d",&x,&y);
        A[x][y]=A[y][x]=1;
    }
    Guass(n,n);
    dfs(n,0);
    printf("%d\n",ans);
    return 0;
}
```

矩阵树定理

有prefer序列和矩阵树定理得，完全图的生成树个数为n^{n-2}

图的生成树个数（允许重边，不允许自环）

无向图：度数对角矩阵-邻接矩阵，求行列式

有向图：已知根i，入度对角矩阵-邻接矩阵，高斯消元，对角矩阵除去根所在位置的积

如下代码，无向图也等价把n当为根，去除其所在位置的值

```
int A[N][N];
int det(int n){//求行列式
    int res=1;
    for(int i=1;i<=n-1;i++){//枚举主对角线上第i个元素
        for(int j=i+1;j<=n-1;j++){//枚举剩下的行
            while(A[j][i]){//辗转相除
                int t=A[i][i]/A[j][i];
                for(int k=i;k<=n-1;k++)//转为倒三角
                    A[i][k]=(A[i][k]-t*A[j][k]+mod)%mod;
                swap(A[i],A[j]);//交换i、j两行
                res=-res;//取负
            }
        }
        res=(res*A[i][i])%mod;
    }
    return (res+mod)%mod;
}
void add(int x,int y){//无向图要建两次边
    A[x][y]--;A[y][y]++;
}
```

$\sum_{Tree}\prod_{edge\in Tree} weight_{edge}$

生成树的边权积的和：$A[i][j]-=边权，A[j][j]+=连着的边权$

$\sum_{Tree} 1$

生成树的边权积的和：$A[i][j]-=1$，$A[j][j]+=1$

生成树的边权和的和：每条边的贡献变为$wx+1$,高斯消元出的x的一次项系数

最小生成树个数

定理：同一个图的每个最小生成树中，边权相等的边数量相等

假设最小生成树上不同边权总数为k，最小生成树上每种边权的条数为c_k,每次考虑一种边权，每次对最小生成树上其它点并查集缩点，自环不考虑，求行列式,最后乘法得出,m为所有边权和最小生成树相等的边

复杂度为$O(kn+\sum_{i=1}^{k} c_k^3+m)\le O(n^2+n^3+m)=O(n^3+m),\because \sum_{i=1}^{k} c_k^3\le(\sum_{i=1}^{k} c_k)^3$

考虑辗转相除的行列式求法，复杂度为$O(n^3logn+m)$

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int N=100+10,M=1000+10,mod=31011;
int n,m;
```

```
int A[N][N];
int det(int n){//求行列式
    int res=1;
    for(int i=1;i<=n-1;i++){//枚举主对角线上第i个元素
        for(int j=i+1;j<=n-1;j++){//枚举剩下的行
            while(A[j][i]){//辗转相除
                int t=A[i][i]/A[j][i];
                for(int k=i;k<=n-1;k++)//转为倒三角
                    A[i][k]=(A[i][k]-t*A[j][k]+mod)%mod;
                swap(A[i],A[j]);//交换i、j两行
                res=-res;//取负
            }
        }
        res=(res*A[i][i])%mod;
    }
    return (res+mod)%mod;
}
void add(int x,int y){//无向图要建两次边
    A[x][y]--;A[y][y]++;
}
int f[N],is[N],cnt=0;
int ff(int x){
    if (f[x]==x)return x;
    return f[x]=ff(f[x]);
}
bool uni(int x,int y){
    int xx=ff(x),yy=ff(y);
    if (xx!=yy){
        f[xx]=yy;return 1;
    }return 0;
}
struct node{
    int x,y,z;
}e[M];
bool cmp(node a,node b){
    return a.z<b.z;
}
vector<node>et;
signed main(){
    cin>>n>>m;
    for (int i=1;i<=m;i++)scanf("%lld%lld%lld",&e[i].x,&e[i].y,&e[i].z);
    sort(e+1,e+1+m,cmp);
    for (int i=1;i<=n;i++)f[i]=i;
    et.clear();
    for (int i=1;i<=m;i++){
        if (uni(e[i].x,e[i].y)){
            et.push_back(e[i]);
        }
    }
    int ans=1,pre=0,sz=et.size();
    if (sz<n-1)return puts("0"),0;
    for (int i=1;i<=m;i++){
        if (e[i].z>et[sz-1].z)break;
        if (e[i].z!=pre){
            if (pre)ans=ans*det(cnt)%mod;
            pre=e[i].z;
            for (int j=0;j<=n;j++)for (int k=0;k<=n;k++)A[j][k]=0;
            cnt=0;for (int j=1;j<=n;j++)f[j]=j,is[j]=0;
```

```
            for (node j:et)if (j.z!=e[i].z)uni(j.x,j.y);
        }
        int xx=ff(e[i].x),yy=ff(e[i].y);
        if (xx!=yy){
            if (!is[xx])is[xx]=++cnt;
            if (!is[yy])is[yy]=++cnt;
            add(is[xx],is[yy]);add(is[yy],is[xx]);
        }
    }
    ans=ans*det(cnt)%mod;
    cout<<ans<<endl;
    return 0;
}
```

LGV引理

$w(P)$表示P这条路径上所有边的边权之积。(路径计数时，可以将边权都设为1)(事实上，边权可以为生成函数)

$e(u,v)$表示u到v的每一条路径的$w(P)$之和，即$\sum_{P:u\to v} w(P)$。

起点集合，A是有向无环图点集的一个子集，大小为n。

终点集合，B也是有向无环图点集的一个子集，大小也为n。

一组$A\to B$的不相交路径S：S_i是一条从A_i到$B_{\sigma(S)_i}$的路径($\sigma(S)$是一个排列)，对于任何$i\neq j$，S_i和S_j没有公共顶点。

$N(\sigma)$表示排列σ的逆序对个数。

$$M=\begin{bmatrix} e(A_1,B_1)\dots e(A_1,B_n) \\ \vdots \qquad\qquad \vdots \\ e(A_n,B_1)\dots e(A_n,B_n) \end{bmatrix}$$

A到B的不相交路径权值和:$det(A)=\sum_{S:A\to B}(-1)^{N(\sigma(S))}\prod_{i=1}^{n} w(S_i)$

$\sum_{S:A\to B}$表示满足上文要求的$A\to B$的每一组不相交路径S。

线性基

```
const int maxn=63;//或62
int p[maxn+1],d[maxn+1];
int cnt=0;
//普通线性基
void get_lb(int x){
    for (int i=maxn;i>=0;i--){
        if (!(x>>i))continue;
        if (!p[i]){
            p[i]=x;
            break;
        }
        x^=p[i];
    }
}

//求最大异或和
int getsum(){
    int ans=0;
    for (int i=maxn;i>=0;i--)
```

```
        if((ans^p[i])>ans)
            ans^=p[i];
    return ans;
}
//求第k小的值
void rebuild(){
    for (int i=maxn;i>=0;i--){
        if (p[i]){//优化，可有可无
            for (int j=i+1;j<=maxn;j++){
                if ((p[j]>>i)&1) p[j]^=p[i];
            }
        }
    }
    for(int i=0;i<=maxn;i++) if (p[i]) d[cnt++]=p[i];
    return;
}
int kth(int k){
    if(cnt!=n)--k;
    if (k>=(1LL<<cnt)) return -1;
    int ans=0;
    for(int i=0;i<=cnt;i++){
        if ((k>>i)&1){
            ans^=d[i];
        }
    }
    return ans;
}
```

矩阵运算和快速幂

```
 Const int M=2;
struct Mat{
    int m[M][M];
    Mat(){
        for (int i=0;i<M;i++){
            for (int j=0;j<M;j++){
                m[i][j]=0;
            }
        }
    }
};
Mat operator *(Mat &a, Mat &b){
    Mat ans;
    for(int i=0;i<M;i++){
        for(int j=0;j<M;j++){
            for(int k=0;k<M;k++){
                ans.m[i][j]=(ans.m[i][j]+a.m[i][k]*b.m[k][j]%mod)%mod;
            }
        }
    }
    return ans;
}
Mat qpow(Mat t,int p){
    Mat ans;
    for(int i=0;i<M;i++){
        ans.m[i][i]=1;
    }
```

```
    while(p) {
        if (p&1)ans=ans*t;
        p>>=1;t=t*t;
    }
    return ans;
}
```

奇异值分解(SVD)

$$\text{拉伸}(x,y):\begin{bmatrix}2&0\\0&1\end{bmatrix}\begin{bmatrix}x_1&x_2\\y_1&y_2\end{bmatrix}=\begin{bmatrix}2x_1&2x_2\\y_1&y_2\end{bmatrix}$$
$$\text{逆时针旋转}(x,y):\begin{bmatrix}\cos(\theta)&-\sin(\theta)\\\sin(\theta)&\cos(\theta)\end{bmatrix}\begin{bmatrix}x_1&x_2\\y_1&y_2\end{bmatrix}$$
$$M=U\Sigma V^T,m*n\text{的}M\text{矩阵分解为}m\text{阶正交矩阵}U\text{，}m*n\text{的对角矩阵}\Sigma\text{，}n\text{阶正交矩阵}V$$
$$\text{正交矩阵的转置矩阵为其逆矩阵，}UV\text{为旋转，}\Sigma\text{为拉伸}$$
$$MV=U\Sigma$$
$$\text{分解方法：}$$
$$M^TM=(U\Sigma V^T)^TU\Sigma V^T=V\Sigma\Sigma V^T=VLV^T(L=\Sigma\Sigma)$$
$$M^TMV=VL,M^TM\vec{v}=\lambda\vec{v}$$
$$\text{得}V\text{是}M^TM\text{的特征向量，}L\text{为特征值的对角矩阵}$$
$$\text{同理},MM^TU=UL,U\text{是}MM^T\text{的特征向量，}L\text{为特征值的对角矩阵}$$
$$\text{应用：}$$
$$M^{-1}=V\Sigma^{-1}U^T$$

主成分分析(PCA)

投影过后方差最大作为主成分(坐标轴)

1.使得数据集更易使用。

2.降低算法的计算开销。

3.去除噪声。

4.使得结果容易理解。

白数据：xy方向都是标准正态分布，xy不相关

$$\text{白数据}D\leftarrow\text{拉伸}S^{-1}R^{-1}D\leftarrow\text{旋转}R^{-1}D\leftarrow\text{我们手头的数据}D',S\text{为拉伸对角矩阵，}R\text{为旋转正交矩阵}$$
$$\text{协方差矩阵}cov(x,y)=\frac{\sum_{i=1}^{n}(x_i-\overline{x})(y_i-\overline{y})}{n-1}\text{的特征向量就是}R$$
$$\text{协方差表示的是数据变化时是同向变化的还是反向变化的。}x\uparrow,y\uparrow,\text{则}cov(x,y)>0$$
$$\text{协方差矩阵}C=\begin{bmatrix}cov(x,x)&cov(x,y)\\cov(x,y)&cov(y,y)\end{bmatrix}$$
$$\text{白数据}C=\begin{bmatrix}1&0\\0&1\end{bmatrix}$$
$$\text{中心化后，}C=\frac{1}{n-1}DD^T,D=\begin{bmatrix}x_1&x_2\\y_1&y_2\end{bmatrix}$$
$$C'=\frac{1}{n-1}D'D'^T=\frac{1}{n-1}RSD(RSD)^T=RS(\frac{1}{n-1}DD^T)S^TR^T=RSS^TR^T=RLR^{-1},L=SS^T$$
$$R\text{就是}C'\text{的特征向量，}L\text{为特征值的对角矩阵}$$

1.去中心化，将原点调整到数据中心

2.