西南科技大学 *ACM*

ACMICPC 竞赛

常用模板程序集

By islands

目录

一、数学部分

1、常用数学知识汇总

1.1 组合公式

(1) 求和公式 k=1,2,……,n;

1） sum(k)=n(n+1)/2;

2） sum(2k-1)=n^2;

3） sum(k^2)=n (n+1)(2n+1)/6;

4） sum((2k-1)^2)=n(4n^2-1)/3;

5） sum(k^3)=(n(n+1)/2)^2;

6） sum((2k-1)^3)=n^2(2n^2-1);

7） sum(k^4)=n(n+1)(2n+1)(3n^2+3n-1)/30;

8） sum(k^5)=n^2(n+1)^2(2n^2+2n-1)/12;

9） sum(k(k+1))=n(n+1)(n+2)/3;

10）sum(k(k+1)(k+2))=n(n+1)(n+2)(n+3)/4;

11）sum(k(k+1)(k+2)(k+3))=n(n+1)(n+2)(n+3)(n+4)/5;

(2) C(m,n)=C(m-1,n)+C(m-1,n-1);

1.2 知识点总结

（1）2 个极限

$$\lim_{n\to\infty}\frac{\sqrt{2\pi n}(n/e)^n}{n!}=1$$

$$\lim_{n\to\infty}T(n)=\frac{n}{\log_{10}^n}$$

（2）递推逆元公式

$$inv[i]=(M-M/i)*inv[M\%i]\%M$$

（3）错排公式

$$D(n)=(n-1)\left[D(n-2)+D(n-1)\right]$$

（4）斐波拉契数列性质（F[0]=0,F[1]=1）

1）$[f(n)]^2-f(n-1)f(n+1)=(-1)^{(n-1)}$

2）斐波那契数列的第 n+2 项同时也代表了集合{1,2,…,n}中所有不包含相邻正整数的子集个数。

3） $f(0)+f(1)+f(2)+....+f(n)=f(n+2)-1$

4） $f(1)+f(3)+.....+f(2n-1)=f(2n)$

5） $f(2)+f(4)+....+f(2n)=f(2n+1)-1$

6） $f(1)^2+f(2)^2+...f(n)^2=f(n)*f(n+1)$

7）隔项公式

$f(2n-2m-2)[f(2n)+f(2n+2)]=f(2m+2)+f(4n-2m)$ $(n>m>=-1,n>=1)$

8） $f(2n)/f(n)=f(n-1)+f(n+1)$

（5）求 n 前面和 n 互质的数的乘积对 n 取模

结论：对于 1，2，4，P^n, 2*P^n,答案为 N-1，其余情况都是 1。也就是说，1，2，4，以及只有一个质因子(奇数)或者它的 1/2 只有一个

（6）多项式的差分性质：

对于任意满足多项式 P(n) = aD.n^D+aD-1.n^D-1+...+a1.n+a0 的 D 阶多项式，取一阶差分得：tmp = P(n) - (n - 1)肯定是个 D-1 阶多项式，以此类推，取 n-1 阶差分，就只剩下一个数 d。

（7） lusca 定理 求 C(x,y)%p;

lusca (x,y,p)=(c(x%p,y%p)%p)*lusca(x/p,y/p,p);

lusca(x,0,p)=1;

2、生成格雷码

每个 2 进制数据对应一个格雷码

每次调用 gray()取得下一个码。

0…000 是第一个码，10000…00 是最后一个码。

code 是用 2 进制储存的 2 进制。

```
int code[10];
void gray(int n,int *code)
{
    int t=0;
    for(int i=0;i<n;t+=code[i++]);
    if(t&1)
        for(n--;!code[n];n--);
    code[n-1]=1-code[n-1];
}
```


3、ex_gcd

求解线性同余方程:满足方程 ax+by=gcd(a,b)的一组解 d 为最大公约数

x%=b;if(ax<0) x+=b 可以求得 a 关于 b 的逆元

```
void ex_gcd(int a,int b,int &d,int &x,int &y)
{
    if(b==0)
    {
        d=a;
        x=1;
        y=0;
        return ;
    }
    ex_gcd(b,a%b,d,x,y);
    int t=x;
    x=y;
    y=t-a/b*y;
}
```

4、欧拉函数

欧拉函数满足 a^phi[m]%m=1;(其中 a,m 为互质)

对于单个欧拉函数可以直接求:

phi[m*n]=phi[m]*phi[n](n,m是素数)

phi[p^n]=p^n-p^(n-1);

```
int phi(int a)
{
    int ans=1;
    for(int i=2; i*i<=a; i++)
    {
        int pp=1;
        int cnt=0;
        while(a%i==0)
        {
            a/=i;
            cnt++;
```


```
            pp*=i;
        }
        if(pp>=i)
            ans*=(pp-pp/i);
    }
    if(a>1)
        ans*=(a-1);
    return ans;
}
```

2）区间预处理

```
LL phi[mmax],ans[mmax];
void pre()
{
    int i,j;
    for(i=1; i<mmax; i++)
        phi[i]=i;
    for(i=2; i<mmax; i++)
    {
        if(phi[i]==i)
        {
            for(j=i; j<mmax; j+=i)
                phi[j]=phi[j]/i*(i-1);
        }
    }
}
```

5、高斯消去

```
double a[mmax][mmax];
double ans[mmax];
void gauss(int n,int m)
{
    int max_r;
    memset(ans,0,sizeof ans);
    for(int r=0; r<n; r++)
    {
        max_r=r;
```


```
        for(int k=r+1; k<n; k++)
        {
            if(fabs(a[k][r])>fabs(a[max_r][r]))
                max_r=k;
        }
        if(max_r!=r)
        {
            for(int k=r; k<m; k++)
            {
                swap(a[r][k],a[max_r][k]);
            }
        }
        if(fabs(a[r][r])<eps)
            continue;
        for(int j=r+1; j<n; j++)
        {
            double ak=a[j][r]/a[r][r];
            a[j][r]=0.0;
            for(int c=r+1; c<m; c++)
            {
                a[j][c]=a[j][c]-a[r][c]*ak;
            }
        }
    }
    for(int i=n-1; i>=0; i--)
    {
        ans[i]=a[i][m-1];
        for(int j=i+1; j<n; j++)
            ans[i]-=ans[j]*a[i][j];
        ans[i]/=a[i][i];
    }
}
```

6、辛普森积分

辛普森积分是一种可以根据精度调整迭代次数的积分：

//F(x)为函数原型

```
double F(double x)
```


```
{
    return x*x;
}
double sim(double a,double b)
{
    double c=(a+b)/2;
    return (F(a)+4*F(c)+F(b))*(b-a)/6;
}
double sac(double a,double b,double eps,double A)
{
    double c=(a+b)/2;
    double L=sim(a,c);
    double R=sim(c,b);
    if((L+R-A)<15*eps)
        return L+R+(L+R-A)/15;
    return sac(a,c,eps/2,L)+sac(c,b,eps/2,R);
}
double sc(double x,double y,double eps)
{
    return sac(x,y,eps ,sim(x,y));
}
int main()
{
    double a,b;
    scanf("%lf %lf",&a,&b);
    cout<<sc(a,b,1e-3)<<endl;
}
```

7、一些关于整除的性质

1）能被 3 或 9 整除的数的特征是它的各个数位上的数字之和能被 3 或 9 整除。

2）能被 11 整除的数的特征是它的奇位上的数字和减去偶位上的数字和，所得的差能被 11 整除。

3）一个整数割去末位数字后的数，再减去末位数字（前面所说的那个数）的 n 倍，如果余数为 10n+1 的倍数，则原数也是 10n+1 的倍数。

4）一个整数割去末位数字后的数，再减去末位数字（前面所说的那个数）的 n 倍，如果余数为 10n－1 的倍数，则原数也是 10n－1 的倍数。

5）一个整数割去末位数字后的数，再减去末位数字（前面所说的那个数）的 3n＋1 倍，如果余数为 10n+3 的倍数，则原数也是 10n+3 的倍数。

6）一个整数割去末位数字后的数，再减去末位数字（前面所说的那个数）的 3n＋1 倍，如果余数为 10n+7 的倍数，则原数也是 10n+7 的倍数。

7）9 的余数：把一个数的各个数字相加，得到若不是一位数再第 2 次相加直到出现一位数。该一位数则是除 9 后的余数。

8、几类特殊计数

8.1 卡特兰数

递推公式：令 h(0)=1,h(1)=1

1）h(n)= h(0)*h(n-1)+h(1)*h(n-2) + ... + h(n-1)h(0) (n>=2)

2）h(n)=h(n-1)*(4*n-2)/(n+1);

3）h(n)=C(2n,n)/(n+1) (n=0,1,2,...)

4）h(n)=c(2n,n)-c(2n,n-1)(n=0,1,2,...)

几种情况用处：

1） 出栈次序：一个栈(无穷大)的进栈序列为 1，2，3，…，n，不同的出栈序列有 h(n)种 。

2） 凸多边形三角划分：在一个凸多边形中，通过若干条互不相交的对角线，把这个多边形划分成了若干个三角形。对于凸 n 变形，有 h(n)种不同划分方案数。

3） 给定节点组成二叉树: 给定 N 个节点，能构成 h(n)种不同的二叉树。

8.2 Stirling 数

1）第一类 Stirling 数 s(p,k)表示：将 p 个物体排成 k 个非空循环排列的方法数。

递推公式： s(p,k)=(p-1)*s(p-1,k)+s(p-1,k-1) ,1<=k<=p-1

边界条件： s(p,0)=0 ,p>=1 s(p,p)=1 ,p>=0

2) 第二类 Stirling 数 s(p,k)表示: 将 p 个物体划分成 k 个非空的不可辨别的（可以理解为盒子没有编号）集合的方法数。k!S(p,k)是把 p 个人分进 k 间有差别(如：被标有房号）的房间(无空房）的方法数。

第二类斯特林数的展开式：

$$S_2(n,m)=\frac{1}{m!}N=\frac{1}{m!}\sum_{k=0}^{m}(-1)^k\binom{m}{k}(m-k)^n$$

9、大整数的 poLLard_rho 分解

//初始化 flag, Factor 储存质因子。

```
typedef __int64  LL;
#define testnum 10
LL pri[]= {2,3,5,7,11,13,17,19,23,29,31,37,41,43,47};
LL Factor[100005],flag;
LL gcd(LL a,LL b)
{
    while(b)
    {
        LL c=a%b;
        a=b;
        b=c;
    }
    return a;
}
LL multi(LL a,LL b,LL n)
{
    LL tmp=0;
    while(b)
    {
        if(b&1)
        {
            tmp+=a;
            if(tmp>=n)tmp-=n;
        }
        a<<=1;
        if(a>=n)a-=n;
        b>>=1;
    }
    return tmp;
}
LL multimod(LL a,LL m,LL n)
{
    LL tmp=1;
    a%=n;
    while(m)
    {
        if(m&1)tmp=multi(tmp,a,n);
        a=multi(a,a,n);
```


```
        m>>=1;
    }
    return tmp;
}
bool MiLLer_Rabin(LL n)
{
    if(n<2)return 0;
    if(n==2)return 1;
    if(!(n&1))return 0;
    LL k=0,i,j,m,a;
    m=n-1;
    while(!(m&1))m>>=1,k++;
    for(i=0; i<testnum; i++)
    {
        if(pri[i]>=n)return 1;
        a=multimod(pri[i],m,n);
        if(a==1)continue;
        for(j=0; j<k; j++)
        {
            if(a==n-1)break;
            a=multi(a,a,n);
        }
        if(j==k)return 0;
    }
    return 1;
}
LL poLLard_rho(LL c,LL n)
{
    LL i,x,y,k,d;
    i=1;
    x=y=rand()%n;
    k=2;
    do
    {
        i++;
        d=gcd(n+y-x,n);
        if(d>1&&d<n)return d;
        if(i==k)y=x,k<<=1;
```


```
        x=(multi(x,x,n)+n-c)%n;
    }
    while(y!=x);
    return n;
}
void rho(LL n)
{
    if(MiLLer_Rabin(n))
    {
        Factor[flag]=n;
        flag++;
        return;
    }
    LL t=n;
    while(t>=n)t=poLLard_rho(rand()%(n-1)+1,n);
    rho(t);
    rho(n/t);
    return ;
}
```

10、FBI 对 n 求模的循环节

原理：对于一个正整数 n，我们求 Fib 数模 n 的循环节的长度的方法如下：

1）把 n 素因子分解，即

2）分别计算 Fib 数模每个 的循环节长度，假设长度分别是

3）那么 Fib 模 n 的循环节长度

从上面三个步骤看来，貌似最困难的是第二步，那么我们如何求 Fib 模 的循环节长度呢？

这里有一个优美的定理：Fib 数模 的最小循环节长度等于 ，其中 表示 Fib 数模素数 的最小循环节长度。可以看出我们现在最重要的就是求

对于求 我们利用如下定理：

如果 5 是模 的二次剩余，那么循环节的的长度是 的因子，否则，循环节的长度是 的因子。（顺便说一句，对于小于等于 5 的素数，我们直接特殊判断，loop(2)=3,loop(3)=8,loop(5)=20。）那么我们可以先求出所有的因子，然后用矩阵快速幂来一个一个判断，这样时间复杂度不会很大。

A 是模 p（p 是质数）的 2 次剩余当且仅当 `a^((p-1)/2)=1mod(p)`

代码：Acdream 1124 求 F[F[F[n]]] % P

```
#include<cstdio>
#include<vector>
#include<iostream>
using namespace std;
typedef __int64 LL;
struct Factor
{
    LL p;
    int cnt;
};
struct mat
{
    LL A[2][2];
};
vector<Factor>fac;
void get_factor(LL x)
{
    fac.clear();
    for(LL i=2; i*i<=x; i++)
    {
        int cnt=0;
        while(x%i==0)
        {
            x/=i;
            cnt++;
        }
        if(cnt)
        {
            Factor tp= {i,cnt};
            fac.push_back(tp);
        }
    }
    if(x>1)
    {
        Factor tp= {x,1};
        fac.push_back(tp);
    }
}
```


```
void unit(mat &x)
{
    x.A[0][0]=x.A[1][1]=1;
    x.A[0][1]=x.A[1][0]=0;
}
mat mat_mul(mat A,mat B,LL mod)
{
    mat ret;
    for(int i=0; i<2; i++)
    {
        for(int j=0; j<2; j++)
        {
            ret.A[i][j]=0;
            for(int k=0; k<2; k++)
            {
                ret.A[i][j]+=A.A[i][k]*B.A[k][j];
                ret.A[i][j]%=mod;
            }
        }
    }
    return ret;
}
mat mat_pow_mod(LL n,LL mod)
{
    mat ret,temp;
    unit(ret),unit(temp);
    temp.A[0][1]=temp.A[1][0]=1;
    temp.A[1][1]=0;
    for(; n; n/=2)
    {
        if(n&1)
            ret=mat_mul(ret,temp,mod);
        temp=mat_mul(temp,temp,mod);
    }
    return ret;
}
LL gcd(LL a,LL b)
{
```


```
    if(b==0)
        return a;
    return gcd(b,a%b);
}
LL lcm(LL a,LL b)
{
    return a/gcd(a,b)*b;
}
LL f(LL n,LL mod)
{
    if(n==0)
        return 0;
    if(n==1)
        return 1;
    mat ret=mat_pow_mod(n-1,mod);
    return ret.A[0][0];
}
LL pow_mod(LL a,LL b,LL mod)
{
    LL res=1,tmp=a;
    for(; b; b/=2)
    {
        if(b&1)
            res=(res*tmp)%mod;
        tmp=(tmp*tmp)%mod;
    }
    return res;
}
LL G(LL p)
{
    if(p==2)
        return 3;
    if(p==3)
        return 8;
    if(p==5)
        return 20;
    LL i,max_n;
    if(pow_mod(5,(p-1)/2,p)==1)
```


```
        max_n=p-1;
    else
        max_n=2*(p+1);
    for(i=1; i*i<=max_n; i++)
    {
        if(max_n%i==0)
        {
            LL x=f(i,p);
            LL y=f(i+1,p);
            if(i>1&&x==0&&y==1)
                return i;
        }
    }
    i--;
    for(; i>=1; i--)
    {
        if(max_n%i==0)
        {
            LL tt=max_n/i;
            LL x=f(tt,p);
            LL y=f(tt+1,p);
            if(x==0&&y==1)
                return tt;
        }
    }
    return 0;
}
LL loop(int id)
{
    LL ans=G(fac[id].p);
    for(int i=1; i<fac[id].cnt; i++)
        ans*=fac[id].p;
    return ans;
}
LL work(LL x)//主函数
{
    get_factor(x);
    LL ans=1;
```


```
    int Sz=fac.size();
    for(int i=0; i<Sz; i++)
    {
        LL xi=loop(i);
        ans=lcm(ans,xi);
    }
    return ans;
}
LL tmp[4];
int main()
{
    int t;
    LL n,p;
    cin>>t;
    while(t--)
    {
        cin>>n>>p;
        tmp[0]=p;
        for(int i=1; i<3; i++)
            tmp[i]=work(tmp[i-1]);
        n++;
        n=(f(n,tmp[2])+1)%tmp[2];
        n=(f(n,tmp[1])+1)%tmp[1];
        n=f(n,tmp[0]);
        printf("%lld\n",n);
    }
    return 0;
}
```

11、莫比乌斯反演

$$f(n)=\sum_{d|n}\mu(d)g(\frac{n}{d})$$

$$\mu(n)=\begin{cases}1 & n=1\\ (-1)^k & n=p_1p_2\cdots p_k\\ 0 & \text{其他}\end{cases}$$

几种常见的反演：

1）g(n)=n,f(n)=phi(n)(欧拉函数)；

2） g(n)=n*n f(n)=[1-n]与[1-n]上互质的对数；

3） $f(d)$ 为满足 $gcd(x,y)=d$ 且 $1\le x\le n$ 和 $1\le y\le m$ 的 (x,y) 的对数

$F(d)$ 为满足 $d\mid gcd(x,y)$ 且 $1\le x\le n$ 和 $1\le y\le m$ 的 (x,y) 的对数

那么，很显然 $F(x)=\frac{n}{x}\bullet\frac{m}{x}$，反演后得到 $f(x)=\sum_{x\mid d}\mu(\frac{d}{x})F(d)=\sum_{x\mid d}\mu(\frac{d}{x})\frac{n}{d}\bullet\frac{m}{d}$

线性预处理莫比乌斯函数：

```
int u[mmax],[mmax];
bool vis[mmax];
void init()    //此处为线性求u函数
{
    memset(vis,0,sizeof(vis));
    u[1] = 1;
    int cnt = 0;
    for(int i=2; i<mmax; i++)
    {
        if(!vis[i])
        {
            prime[cnt++] = i;
            u[i] = -1;
        }
        for(int j=0; j<cnt&&i*prime[j]<mmax; j++)
        {
            vis[i*prime[j]] = 1;
            if(i%prime[j])
                u[i*prime[j]] = -u[i];
            else
            {
                u[i*prime[j]] = 0;
                break;
            }
        }
    }
}
```


12、FFT & NTT

12.1 FFT

y1=a[0]+a[1]x^1+a[2]x^2.....a[n]x^n

y2=b[0]+b[1]x^1+b[2]x^2.....b[m]x^m

y=y1*y2; 在O(nlgn)的复杂度求出y的系数

//hdu 1402 高精度乘法 （特别注意开 2 倍内存）

```
#include<cstdio>
#include<string>
#include<cmath>
#include<cstring>
#include<iostream>
using namespace std;
const int mmax = 100010;
const double pi = acos(-1);
struct complex
{
    double r,i;
    complex(double real=0.0,double imag=0.0)
    {
        r=real,i=imag;
    }
    complex operator+(const complex o)
    {
        return complex(r+o.r,i+o.i);
    }
    complex operator-(const complex o)
    {
        return complex(r-o.r,i-o.i);
    }
    complex operator*(const complex o)
    {
        return complex(r*o.r-i*o.i,r*o.i+i*o.r);
    }
```


```
};   //  复数
int rev(int x,int r)  //蝴蝶操作
{
    int ans=0;
    for(int i=0; i<r; i++)
    {
        if(x&(1<<i))
        {
            ans+=1<<(r-i-1);
        }
    }
    return ans;
}
void FFT(int n,complex A[],int on)  //on==1 正变换 on==-1 逆变换
{
    int r=0;
    for(;; r++)
    {
        if((1<<r)==n)
            break;
    }
    for(int i=0; i<n; i++)
    {
        int tmp=rev(i,r);
        if(i<tmp)
            swap(A[i],A[tmp]);
    }
    for(int s=1; s<=r; s++)
    {
        int m=1<<s;
        complex wn(cos(on*2.0*pi/m),sin(on*2.0*pi/m));
        for(int k=0; k<n; k+=m)
        {
            complex w(1.0,0.0);
            for(int j=0; j<m/2; j++)
            {
                complex t,u;
                t=w*A[k+j+m/2];
```


```
                u=A[k+j];
                A[k+j]=u+t;
                A[k+j+m/2]=u-t;
                w=w*wn;
            }
        }
    }
}

complex A[2*mmax],B[2*mmax];
int ans[2*mmax];
int main()
{
    //freopen("in.txt","r",stdin);
    int n,m;
    string s1;
    string s2;
    while(cin>>s1>>s2)
    {
        n=s1.size();
        m=s2.size();
        memset(A,0,sizeof A);
        memset(B,0,sizeof B);
        for(int i=n-1; i>=0 ; i--)
            A[i]=complex(s1[n-i-1]-'0',0);
        for(int i=m-1; i>=0; i--)
            B[i]=complex(s2[m-i-1]-'0',0);
        int tmp=1;
        while(tmp<max(n,m))
            tmp*=2;
        n=tmp;
        FFT(2*n,A,1);
        FFT(2*n,B,1);
        for(int i=0; i<2*n; i++)
            A[i]=A[i]*B[i];
        FFT(2*n,A,-1);
        memset(ans,0,sizeof ans);
        for(int i=0; i<2*n; i++)
```


```
            A[i].r/=2*n;
        for(int i=0; i<2*n; i++)
        {
            int tmp=A[i].r+0.5;
            ans[i]+=tmp;
            if(ans[i]>=10)
            {
                ans[i+1]+=ans[i]/10;
                ans[i]%=10;
            }
        }
        int e=0;
        for(int i=2*n-1; i>=0; i--)
        {
            if(ans[i])
            {
                e=i;
                break;
            }
        }
        for(int i=e; i>=0; i--)
        {
            printf("%d",ans[i]);
        }
        printf("\n");
    }
    return 0;
}
```

12.2 NTT & 分治

NTT 是对于在模 p 域下面的 FFT。这里的 p 必须是费马素数(p=a*2^n),需要求出 p 的一个原根 g.

//ZOJ 3874

```
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<iostream>
```


```
using namespace std;
typedef long long LL;
const int mmax  = 1<<17;
const int mod = 786433;
const int g = 11;  //原根
LL quick_mod(LL a,LL b)
{
    LL ans=1;
    for(; b; b/=2)
    {
        if(b&1)
            ans=ans*a%mod;
        a=a*a%mod;
    }
    return ans;
}
int rev(int x,int r)  //蝴蝶操作
{
    int ans=0;
    for(int i=0; i<r; i++)
    {
        if(x&(1<<i))
        {
            ans+=1<<(r-i-1);
        }
    }
    return ans;
}
void NTT(int n,LL A[],int on) // 长度为N (2的次数)
{
    int r=0;
    for(;; r++)
    {
        if((1<<r)==n)
            break;
    }
    for(int i=0; i<n; i++)
    {
```


```
        int tmp=rev(i,r);
        if(i<tmp)
            swap(A[i],A[tmp]);
    }
    for(int s=1; s<=r; s++)
    {
        int m=1<<s;
        LL wn=quick_mod(g,(mod-1)/m);
        for(int k=0; k<n; k+=m)
        {
            LL  w=1;
            for(int j=0; j<m/2; j++)
            {
                LL t,u;
                t=w*(A[k+j+m/2]%mod)%mod;
                u=A[k+j]%mod;
                A[k+j]=(u+t)%mod;
                A[k+j+m/2]=((u-t)%mod+mod)%mod;
                w=w*wn%mod;
            }
        }
    }
    if(on==-1)
    {
        for(int i=1; i<n/2; i++)
            swap(A[i],A[n-i]);
        LL inv=quick_mod(n,mod-2);
        for(int i=0; i<n; i++)
            A[i]=A[i]%mod*inv%mod;
    }
}
LL A[mmax+10],B[mmax+10];
LL dp[mmax+10];
LL jie[mmax+10];
void cdq(int l,int r)
{
    if(l==r)
    {
```


```
        dp[l]=jie[l]-dp[l];
        dp[l]=(dp[l]%mod+mod)%mod;
        return ;
    }
    int mid=(l+r)>>1;
    cdq(l,mid);
    int n=r-l+1;
    for(int i=0,j=l; i<n; i++,j++)
    {
        if(j<=mid)
            A[i]=dp[l+i];
        else
            A[i]=0;
    }
    for(int i=0; i<n; i++)
        B[i]=jie[i+1];
    NTT(n,A,1);
    NTT(n,B,1);
    for(int i=0; i<n; i++)
        A[i]=A[i]*B[i]%mod;
    NTT(n,A,-1);
    for(int i=r,j=n-2; i>mid; i--,j--)
    {
        LL tmp=A[j];
        dp[i]+=tmp;
        dp[i]=(dp[i]%mod+mod)%mod;
    }
    cdq(mid+1,r);
}
void pre()
{
    jie[0]=1;
    for(int i=1; i<mmax; i++)
        jie[i]=jie[i-1]*i%mod;
    int n=1;
    while(n<=mmax) n<<=1;
    n/=2;
    memset(dp,0,sizeof dp);
```


```
    cdq(1,n);
}
int c[mmax+10];
int main()
{
    pre();
    int T;
    cin>>T;
    while(T--)
    {
        int n,m;
        scanf("%d %d",&n,&m);
        LL ans=1;
        while(m--)
        {
            int cnt;
            scanf("%d",&cnt);
            for(int i=0; i<cnt; i++)
                scanf("%d",&c[i]);
            sort(c,c+cnt);
            bool fg=1;
            for(int i=1; i<cnt; i++)
                if(c[i]!=c[i-1]+1)
                {
                    fg=0;
                    break;
                }
            if(!fg)
                ans=0;
            else
            {
                ans*=dp[cnt];
                ans%=mod;
            }
        }
        printf("%lld\n",ans);
    }
    return 0;
```


```
}
```

13、分拆数

关于分拆数的几种结论：

1） n 的分拆数中 全部是奇数的个数 =每个数字都不同的方案数。

2） n 的分拆数中 最大部分为 m 的个数 = 把 n 分拆成 m 部分的个数。

3） n 的分拆数中 每部分的数都相等的个数 = n 的因子个数。

4） n 的分拆数中 每部分都是 1 或 2(或者把 n 分拆成 1 或 2 部分)的个数=floor(n/2+1);

5） n 的分拆数中 每部分都是 1 或 2 或 3(或者把 n 分拆成 1 或 2 或 3 部分)的个数=(n+3)^2/12;

第一种情况：直接拆分

```
int dp[100010];
void init()
{
    memset(dp,0,sizeof(dp));
    dp[0] = 1;
    for(int i = 1; i <= mmax; i++)
    {
        for(int j = 1, r = 1; i - (3 * j * j - j) / 2 >= 0; j++, r *= -1)
        {
            dp[i] += dp[i -(3 * j * j - j) / 2] * r;
            dp[i] %= mod;
            dp[i] = (dp[i]+mod)%mod;
            if( i - (3 * j * j + j) / 2 >= 0 )
            {
                dp[i] += dp[i - (3 * j * j + j) / 2] * r;
                dp[i] %= mod;
                dp[i] = (dp[i]+mod)%mod;
            }
        }
    }
}
```

第 2 钟情况：要求拆分的数中每个数出现的次数不能大于等于 k 次

```
int dp[mmax];
void init()
```


```
{
    memset(dp,0,sizeof(dp));
    dp[0] = 1;
    for(int i = 1; i<mmax; i++)
    {
        for(int j = 1, r = 1; i - (3 * j * j - j) / 2 >= 0; j++, r *= -1)
        {
            dp[i] += dp[i -(3 * j * j - j) / 2] * r;
            dp[i] %= mod;
            dp[i] = (dp[i]+mod)%mod;
            if( i - (3 * j * j + j) / 2 >= 0 )
            {
                dp[i] += dp[i - (3 * j * j + j) / 2] * r;
                dp[i] %= mod;
                dp[i] = (dp[i]+mod)%mod;
            }
        }
    }
}
int solve(int n,int k)
{
    int ans = dp[n];
    for(int j = 1, r = -1; n - k*(3 * j * j - j) / 2 >= 0; j++, r *= -1)
    {
        ans += dp[n -k*(3 * j * j - j) / 2] * r;
        ans %= mod;
        ans = (ans+mod)%mod;
        if( n - k*(3 * j * j + j) / 2 >= 0 )
        {
            ans += dp[n - k*(3 * j * j + j) / 2] * r;
            ans %= mod;
            ans = (ans+mod)%mod;
        }
    }
    return ans;
}
```


14 、 baby_step giant_step

```
(POJ 2417,3243)
// a^x = b (mod n) n是素数和不是素数都可以
// 求解上式 0<=x < n的解

#define MOD 76543
int hs[MOD],head[MOD],next[MOD],id[MOD],top;
void insert(int x,int y)
{
    int k = x%MOD;
    hs[top] = x, id[top] = y, next[top] = head[k], head[k] = top++;
}
int find(int x)
{
    int k = x%MOD;
    for(int i = head[k]; i != -1; i = next[i])
        if(hs[i] == x)
            return id[i];
    return -1;
}
int BSGS(int a,int b,int n)
{
    memset(head,-1,sizeof(head));
    top = 1;
    if(b == 1)return 0;
    int m = sqrt(n*1.0), j;
    long long x = 1, p = 1;
    for(int i = 0; i < m; ++i, p = p*a%n) insert(p*b%n,i);
    for(long long i = m; ; i += m)
    {
        if( (j = find(x = x*p%n)) != -1 )return i-j;
        if(i > n)break;
    }
    return -1;
}
```


二、图论部分

1、two_set

```
struct node
{
    int en;
    int next;
}E[2000000];
int p[mmax],num;
void init()
{
    memset(p,-1,sizeof p);
    num=0;
}
void add(int st,int en)
{
    E[num].en=en;
    E[num].next=p[st];
    p[st]=num++;
}
int mark[mmax],S[mmax];
int n,m,c;
bool dfs(int u)
{
    if(mark[u^1])
        return 0;
    if(mark[u])
        return 1;
    mark[u]=1;
    S[c++]=u;
    for(int i=p[u]; i+1; i=E[i].next)
    {
        int v=E[i].en;
```


```
        if(!dfs(v))
            return 0;
    }
    return 1;
}
bool two_set()
{
    memset(mark,0,sizeof mark);
    for(int i=0; i<2*n; i+=2)
    {
        if(!mark[i]&&!mark[i+1])
        {
            c=0;
            if(!dfs(i))
            {
                while(c>0) mark[S[--c]]=0;
                if(!dfs(i+1)) return 0;
            }
        }
    }
    return 1;
}
int main()
{
    while(scanf("%d %d",&n,&m)!=EOF)
    {
        init();
        int id1,id2,c1,c2;
        for(int i=0; i<m; i++)
        {
            scanf("%d %d %d %d",&id1,&id2,&c1,&c2);
            add(id1*2+c1,id2*2+(c2^1));
            add(id2*2+c2,id1*2+(c1^1));
        }
        if(two_set())
            puts("YES");
        else
            puts("NO");
```


```
    }
    return 0;
}
```

2、连通性问题

2.1 桥、割点、边双联通

```
struct node
{
    int en,next;
    bool vis,fg;
} E[2*mmax];
int p[mmax],num;
void add(int st,int en)
{
    E[num].vis=0;
    E[num].en=en;
    E[num].fg=0;
    E[num].next=p[st];
    p[st]=num++;
}
int dfn[mmax],low[mmax];
bool iscut[mmax];
int times;
void init()
{
    memset(p,-1,sizeof p);
    num=times=0;
}
void dfs(int u) //求桥 割点
{
    int child=0;
    dfn[u]=low[u]=++times;
    for(int i=p[u]; i+1; i=E[i].next)
    {
        if(!E[i].vis)
        {
```


```
            E[i].vis=E[i^1].vis=1;
            int v=E[i].en;
            if(!dfn[v])
            {
                child++;
                dfs(v);
                low[u]=min(low[v],low[u]);
                if(low[v]>dfn[u])
                {
                    E[i].fg=1;
                    E[i^1].fg=1;// 标记为桥
                }
                if(low[v]>=dnf[u])
                    iscut[u]=1;  //标记为割点
            }
            else
                low[u]=min(low[u],dfn[v]);
        }
    }
}
```

2.2 点双连通

```
struct node
{
    int en,next;
} E[2*mmax];
int num,p[mmax];
void add(int st,int en)
{
    E[num].en=en;
    E[num].next=p[st];
    p[st]=num++;
}
void init()
{
    memset(p,-1,sizeof p);
    num=0;
```


```
}
int dfn[mmax],low[mmax];
int times,bcc_cnt;  // 时间戳 bcc个数
stack<node>S;
int bcc[mmax];
set<int>Q[mmax];
void dfs(int u,int fa)
{
    dfn[u]=low[u]=++times;
    for(int i=p[u]; i+1; i=E[i].next)
    {
        int v=E[i].en;
        if(!dfn[v])
        {
            S.push(E[i]);
            dfs(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>=dfn[u])
            {
                bcc_cnt++;
                Q[bcc_cnt].clear();
                while(1)
                {
                    node x=S.top();
                    S.pop();
                    bcc[bcc_cnt]++;
                    Q[bcc_cnt].insert(x.st);
                    Q[bcc_cnt].insert(x.en);
                    if(x.st==u&&x.en==v)
                        break;
                }
            }
        }
        else if(dfn[v]<dfn[u]&&v!=fa)
        {
            S.push(E[i]);
            low[u]=min(low[u],dfn[v]);
        }
```


```
    }
}
```

2.3 强联通

```
struct node
{
    int en;
    int next;
}E[50010];
int p[mmax];
int num;
void add(int st,int en)
{
    E[num].en=en;
    E[num].next=p[st];
    p[st]=num++;
}
void init()
{
    memset(p,-1,sizeof p);
    num=0;
}
int fa[mmax],dfn[mmax],low[mmax];
int Q[mmax];
int pp,times;
bool instack[mmax];
int find(int x)
{
    return fa[x]==x?x:fa[x]=find(fa[x]);
}
void tarjin(int u)
{

    dfn[u]=low[u]=++times;
    Q[++pp]=u;
    instack[u]=1;
    for(int i=p[u];i+1;i=E[i].next)
```


```
    {
        int v=E[i].en;
        if(!dfn[v])
        {
            tarjin(v);
            if(low[u]>low[v])
                low[u]=low[v];
        }
        else if(instack[v])
            low[u]=min(low[u],dfn[v]);
    }
    if(dfn[u]==low[u])
    {
        while(pp)
        {
            int x=Q[pp--];
            instack[x]=0;
            if(x==u)
                break;
            int xx=find(x);
            fa[xx]=u;
        }
    }
}
```

3、混合欧拉回路

要判断一个混合图 G（V,E）（既有有向边又有无向边）是欧拉图，方法如下：

假设有一张图有向图 G'，在不论方向的情况下它与 G 同构。并且 G'包含了 G 的所有有向边。那么如果存在一个图 G'使得 G'存在欧拉回路，那么 G 就存在欧拉回路。

其思路就将混合图转换成有向图判断。实现的时候，我们使用网络流的模型。现任意构造一个 G'。用 Ii 表示第 i 个点的入度，Oi 表示第 i 个点的出度。如果存在一个点 k，|Ok-Ik|mod 2=1，那么 G 不存在欧拉回路。接下来则对于所有 Ii>Oi 的点从源点连到 i 一条容量为(Ii-Oi)/2 的边，对于所有 Ii<Oi 的点从 i 连到汇点一条容量为(Oi-Ii)/2 的边。如果对于节点 U 和 V，无向边（U，V）∈E，那么 U 和 V 之间互相建立容量为无限大的边。如果此网络的最大流等于∑|Ii-Oi|/2，那么就存在欧拉回路。

4、生成树问题

4.1 最小 k 度生成树

1. 先求出最小 m 度限制生成树：原图中去掉和 V0 相连的所有边，得到 m 个连通分量，则这 m 个连通分量必须通过 v0 来连接，所以，在图 G 的所有生成树中 dT(v0)≥m 。也就是说，当 k<m 时，问题无解。对每个连通分量求一次最小生成树，对于每个连通分量 V′ ，用一条与 V0 直接连接的最小的边把它与 V0 点连接起来，使其整体成为一个生成树。于是，我们就得到了一个 m 度限制生成树，这就是最小 m 度限制生成树。

2. 由最小 m 度限制生成树得到最小 m+1 度限制生成树：连接和 V0 相邻的点 v （即是添加边），则可以知道一定会有一个环出现（因为原来是一个生成树），所以只要找到这个环上的最大权边（不能与 v0 点直接相连）并删除（即是删除边），就可以得到一个 m+1 度限制生成树。枚举所有和 V0 相邻点 v，找到替换后，增加权值最小的一次替换 (当然，找不到这样的边时，就说明已经求出，可以直接退出了)，这样就可以求得了最小 m+1 度限制生成树。

```
typedef __int64 LL;
map<string,int>q;
struct node
{
    int st,en,len;
    bool fg;
} E[mmax*mmax];
struct edge
{
    int st,en,len;
    int next;
    bool fg;
} mst[mmax];
int p[mmax],num;
void init()
{
    memset(p,-1,sizeof p);
    num=0;
}
void add(int st,int en,int len,bool fg)
{
    mst[num].st=st;
```


```
    mst[num].en=en;
    mst[num].len=len;
    mst[num].fg=fg;
    mst[num].next=p[st];
    p[st]=num++;
}
char name1[30],name2[30];
int n,m,k,v0,numk;
int fa[mmax];
int find(int x)
{
    if(x==fa[x])
        return fa[x];
    return fa[x]=find(fa[x]);
}
bool cmp(node x,node y)
{
    return x.len<y.len;
}
int kru()
{
    init();
    int sum=0;
    for(int i=1; i<=n; i++)
        fa[i]=i;
    for(int i=0; i<m; i++)
    {
        if(E[i].st==v0||E[i].en==v0)
            continue;
        int x=find(E[i].st);
        int y=find(E[i].en);
        if(x!=y)
        {
            fa[y]=x;
            sum+=E[i].len;
            add(E[i].st,E[i].en,E[i].len,1);
            add(E[i].en,E[i].st,E[i].len,1);
        }
```


```
    }
    numk=0;
    for(int i=0; i<m; i++)
    {
        int x;
        if(E[i].st==v0||E[i].en==v0)
        {
            if(E[i].st==v0)
                x=find(E[i].en);
            if(E[i].en==v0)
                x=find(E[i].st);
            bool fg=0;
            if(x!=v0)
            {
                fa[x]=v0;
                sum+=E[i].len;
                fg=1;
                numk++;
            }
            add(E[i].st,E[i].en,E[i].len,fg);
            add(E[i].en,E[i].st,E[i].len,fg);
        }
    }
    return sum;
}
int maxlen,id;
int dfs(int x,int fa)
{
    int fg=0,i,ok;
    for(i=p[x]; i+1; i=mst[i].next)
    {
        if(mst[i].fg&&mst[i].en==v0)
        {
            return 1;
        }
    }
    for(int i=p[x]; i+1; i=mst[i].next)
    {
```


```
        if(mst[i].fg&&mst[i].en!=fa)
        {
            ok=dfs(mst[i].en,x);
            fg+=ok;
            if(ok&&maxlen<mst[i].len)
                maxlen=mst[id=i].len;
        }
    }
    return fg;
}
void solve(int sum)
{
    int t,ida,idb,ans=sum;
    for(; numk<k; numk++)
    {
        int minadd=0;
        for(int i=p[v0]; i+1; i=mst[i].next)
        {
            if(!mst[i].fg)
            {
                t=mst[i].en;
                maxlen=0,dfs(t,t);
                if(maxlen&&minadd>mst[i].len-maxlen)
                {
                    minadd=mst[i].len-maxlen;
                    ida=i,idb=id;
                }
            }
        }
        if(minadd==0)
            break;
        if(ans>sum+minadd)
            ans=sum+minadd;
        sum=ans;
        mst[ida].fg=mst[ida^1].fg=1;
        mst[idb].fg=mst[idb^1].fg=0;
    }
    printf("Total miles driven: %d\n",ans);
```


```
}
int  main()
{
    while(cin>>m)
    {
        q.clear();
        strcpy(name1,"Park");
        q[name1]=1;
        v0=1,n=1;
        int val;
        for(int i=0; i<m; i++)
        {
            scanf("%s %s %d",name1,name2,&val);
            if(!q[name1])
                q[name1]=++n;
            if(!q[name2])
                q[name2]=++n;
            E[i].st=q[name1];
            E[i].en=q[name2];
            E[i].len=val;
        }
        scanf("%d",&k);
        sort(E,E+m,cmp);
        int sum=kru();
        if(numk>k)
        {
            puts("no answer");
            continue;
        }
        solve(sum);
    }
    return 0;
}
```

4.2 最小树形图的 朱—刘 算法

```
struct node
{
```


```
    int st,en;
    double len;
    int next;
} E[mmax];
int n,m;
struct point
{
    double x,y;
} P[110];
double get_dis(point x,point y)
{
    return sqrt((x.x-y.x)*(x.x-y.x)+(x.y-y.y)*(x.y-y.y));
}
double minin[110];
int fa[110],id[110];
int pre[110];
double zhuliu(int st)
{
    int t;
    int numn=0;
    double sum=0.0;
    while(1)
    {
        ///ONE 找每个点的最小入边及所对应的前驱节点
        for(int i=1; i<=n; i++)
            minin[i]=inf;
        for(int i=0; i<m; i++)
        {
            int u=E[i].st,v=E[i].en;
            if(u!=v&&minin[v]>E[i].len)
            {
                minin[v]=E[i].len;
                pre[v]=u;
            }
        }
        minin[st]=0;
        for(int i=1; i<=n; i++)
            if(minin[i]>=inf)
```


```
        return -1;
///TWO 查找和处理有向环
numn = minin[st] = 0;
memset(id, -1, sizeof id);
memset(fa, -1, sizeof fa);
for(int i=1; i<=n; i++)
{
    sum+=minin[i];
    for(t=i; t!=st&&fa[t]!=i&&id[t]==-1; t=pre[t])
        fa[t]=i;
    if(t!=st&&id[t]==-1)
    {
        id[t]=++numn;
        for(int ta=pre[t]; ta-t; ta=pre[ta])
            id[ta]=numn;
    }
}
if(!numn)
    break;
///THREE 处理及合并顶点
for(int i=1; i<=n; i++)
    if(id[i]==-1)
        id[i]=++numn;
int numm=0;
for(int i=0; i<m; i++)
{
    t=E[i].en;
    E[i].st=id[E[i].st];
    E[i].en=id[E[i].en];
    if(E[i].st!=E[i].en)///THREE 处理及合并顶点
    {
        E[numm].st=E[i].st;
        E[numm].en=E[i].en;
        E[numm].len=E[i].len-minin[t];
        numm++;
    }
}
n=numn,m=numm,st=id[st];
```


```
    }
    return sum;
}
int main()
{
    while(scanf("%d %d",&n,&m)!=EOF)
    {
        for(int i=1; i<=n; i++)
            scanf("%lf %lf",&P[i].x,&P[i].y);
        int x,y;
        for(int i=0; i<m; i++)
        {
            scanf("%d %d",&x,&y);
            double dis=get_dis(P[x],P[y]);
            E[i].st=x;
            E[i].en=y;
            if(x==y)
                E[i].len=inf;
            else
                E[i].len=dis;
        }
        double ans=zhuliu(1);
        if(ans<0)
            printf("poor snoopy\n");
        else
            printf("%.2lf\n",ans);
    }
    return  0;
}
```

4.3 最优比例生成树 （2 分+最小生成树）

```
struct point
{
    int x,y,z;
} P[mmax];
int n;
double cost[mmax][mmax];
```


```
double dis[mmax][mmax];
double get_dis(point x,point y)
{
    return sqrt(1.0*(x.x-y.x)*(x.x-y.x)+1.0*(x.y-y.y)*(x.y-y.y));
}

double Dis[mmax];
bool vis[mmax];
int pre[mmax];
double prim(double mid)
{
    for(int i=1; i<=n; i++)
    {
        Dis[i]=cost[1][i]-mid*dis[1][i];
        pre[i]=1;
    }
    double Cost=0,Len=0;
    memset(vis,0,sizeof vis);
    vis[1]=1;
    Dis[1]=0;
    for(int i=1; i<n; i++)
    {
        double mink=inf;
        int e;
        for(int j=2; j<=n; j++)
        {
            if(!vis[j]&&Dis[j]<=mink)
            {
                mink=Dis[j];
                e=j;
            }
        }
        vis[e]=1;
        Cost+=1.0*abs(P[e].z-P[pre[e]].z);
        Len+=dis[e][pre[e]];
        for(int j=1; j<=n; j++)
        {
            if(!vis[j]&&Dis[j]>cost[e][j]-mid*dis[e][j])
```


```
            {
                Dis[j]=cost[e][j]-mid*dis[e][j];
                pre[j]=e;
            }
        }
    }
    return Cost/Len;
}
void Dink()
{
    double l=0,ans;
    while(1)
    {
        ans=prim(l);
        if(fabs(ans-l)<=eps)
            break;
        l=ans;
    }
    printf("%.3lf\n",ans);
}
int main()
{
    while(cin>>n&&n)
    {
        for(int i=1; i<=n; i++)
        {
            scanf("%d %d %d",&P[i].x,&P[i].y,&P[i].z);
        }
        for(int i=1; i<=n; i++)
        {
            for(int j=1; j<=n; j++)
            {
                cost[i][j]=1.0*abs(P[i].z-P[j].z);
                dis[i][j]=get_dis(P[i],P[j]);
            }
        }
        Dink();
    }
```


```
    return 0;
}
```

5、带权匹配（KM）

```
int w[mmax][mmax];
int lx[mmax],ly[mmax];
int link[mmax];
bool S[mmax],T[mmax];
int dx[mmax];
bool match(int u,int n)
{
    S[u]=1;
    for(int i=1;i<=n;i++)
    {
        if(T[i])
            continue;
        if(lx[u]+ly[i]==w[u][i])
        {
            T[i]=1;
            if(link[i]==-1||match(link[i],n))
            {
                link[i]=u;
                return 1;
            }
        }
        else
            dx[i]=min(dx[i],lx[u]+ly[i]-w[u][i]);

    }
    return 0;
}

void updata(int n)
{

    int min_n=inf;
    for(int i=1;i<=n;i++)
```


```
    {
        if(!T[i])
            min_n=min(min_n,dx[i]);
    }
    for(int i=1;i<=n;i++)
    {
        if(S[i]) lx[i]-=min_n;
        if(T[i]) ly[i]+=min_n;
        else dx[i]-=min_n;
    }
}
void km(int n)
{
    memset(link,-1,sizeof link);
    memset(lx,0,sizeof lx);
    memset(ly,0,sizeof ly);
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            lx[i]=max(lx[i],w[i][j]);
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
            dx[j]=inf;
        while(1)
        {
            memset(S,0,sizeof S);
            memset(T,0,sizeof T);
            if(match(i,n))
                break;
            else
                updata(n);
        }
    }
}
int main()
{
    int n;
    while(cin>>n)
```


```
    {
        for(int i=1;i<=n;i++)
        {
            for(int j=1;j<=n;j++)
            {
                scanf("%d",&w[i][j]);
            }
        }
        km(n);
        int ans=0;
        for(int i=1;i<=n;i++)
            ans+=lx[i]+ly[i];
        printf("%d\n",ans);
    }
    return 0;
}
```

6、网络流问题

6.1 网络最大流

6.1.1 dinic 算法

```
struct node
{
    int flow;
    int en;
    int next;
} E[40100];
int p[mmax];
int num;
void init()
{
    memset(p,-1,sizeof p);
    num=0;
}
void add(int st,int en,int flow)
{
```


```
    E[num].en=en;
    E[num].flow=flow;
    E[num].next=p[st];
    p[st]=num++;
    E[num].en=st;
    E[num].flow=0;
    E[num].next=p[en];
    p[en]=num++;
}

int d[mmax];
bool vis[mmax];
int qq[mmax];
int cur[mmax];
bool bfs(int st,int en)
{
    memset(vis,0,sizeof vis);
    int qcnt=0;
    qq[++qcnt]=st;
    d[st]=0;
    vis[st]=1;
    while(qcnt)
    {
        int x=qq[qcnt];
        qcnt--;
        for(int i=p[x]; i+1; i=E[i].next)
        {
            int v=E[i].en;
            if(!vis[v]&&E[i].flow)
            {
                vis[v]=1;
                qq[++qcnt]=v;
                d[v]=d[x]+1;
            }
        }
    }
    return vis[en];
}
```


```
int dfs(int st,int en,int  flow)
{
    if(st==en||flow==0)
        return flow;
    int f=0,dd;
    for(int &i=cur[st]; i+1; i=E[i].next)
    {
        int v=E[i].en;
        if(d[st]+1==d[v]&&(dd=dfs(v,en,min(flow,E[i].flow)))>0)
        {
            E[i].flow-=dd;
            E[i^1].flow+=dd;
            flow-=dd;
            f+=dd;
            if(flow==0)
                break;
        }
    }
    return f;
}
int dinic(int st,int en,int n)
{
    int flow=0;
    while(bfs(st,en))
    {
        for(int i=0; i<=n; i++)
            cur[i]=p[i];
        flow+=dfs(st,en,inf);
    }
    return flow;
}
```

6.1.2 isap 算法

```
struct node
{
    int st;
    int en;
```


```
    int  flow;
    int next;
} E[510000];
int num;
int p[5100];
void init()
{
    memset(p,-1,sizeof p);
    num=0;
}
void add(int st,int en,int flow)
{
    E[num].st=st;
    E[num].en=en;
    E[num].flow=flow;
    E[num].next=p[st];
    p[st]=num++;
    E[num].st=en;
    E[num].en=st;
    E[num].flow=0;
    E[num].next=p[en];
    p[en]=num++;
}
int T;
int vh[5100];
int h[5100];

int find(int u,int st,int en,int f)
{
    if(u==en)
        return f;
    int left=f;
    int tmp=T;
    for(int i=p[u]; i+1; i=E[i].next)
    {
        int v=E[i].en;
        int flow=E[i].flow;
        if(flow)
```


```
        {
            if(h[v]+1==h[u])
            {
                int d=min(left,flow);
                d=find(v,st,en,d);
                left-=d;
                E[i].flow-=d;
                E[i^1].flow+=d;
                if(h[st]>=T+1)
                    return f-left;
                if(!left)
                    break;
            }
            if(h[v]<tmp)
                tmp=h[v];
        }
    }
    if(left==f)
    {
        vh[h[u]]--;
        if(vh[h[u]]==0)
            h[st]=T+1;
        h[u]=tmp+1;
        vh[h[u]]++;
    }
    return f-left;
}
int isap(int st,int en,int flow)
{
    memset(vh,0,sizeof vh);
    memset(h,0,sizeof h);
    vh[0]=T+1;
    int ans=0;
    while(h[st]<=T)
        ans+=find(st,st,en,flow);
    return ans;
}
```


6.2 费用流

6.2.1 spfa 写法（使用稀疏图）

```
struct node
{
    int st;
    int en;
    int flow,cost;
    int next;
} E[101000];
int p[mmax],num;
void init()
{
    memset(p,-1,sizeof p);
    num=0;
}
void add(int st,int en,int flow,int cost)
{
    E[num].st=st;
    E[num].en=en;
    E[num].flow=flow;
    E[num].cost=cost;
    E[num].next=p[st];
    p[st]=num++;
    E[num].st=en;
    E[num].en=st;
    E[num].flow=0;
    E[num].cost=-cost;
    E[num].next=p[en];
    p[en]=num++;
}
int pre[mmax],dis[mmax];
bool fg[mmax];
bool spfa(int st,int en)
{
    for(int i=0; i<=en; i++)
        fg[i]=0,dis[i]=inf,pre[i]=-1;
```


```
    queue<int>q;
    q.push(st);
    fg[st]=1;
    dis[st]=0;
    while(!q.empty())
    {
        int u=q.front();
        q.pop();
        fg[u]=0;
        for(int i=p[u]; i+1; i=E[i].next)
        {
            int v=E[i].en;
            if(E[i].flow&&dis[v]>dis[u]+E[i].cost)
            {
                dis[v]=dis[u]+E[i].cost;
                pre[v]=i;
                if(!fg[v])
                {
                    fg[v]=1;
                    q.push(v);
                }
            }
        }
    }
    if(dis[en]<inf)
        return 1;
    return 0;
}
int solve(int st,int en)
{
    int ans=0;
    while(spfa(st,en))
    {
        int d=inf;
        for(int i=pre[en]; i+1; i=pre[E[i].st])
            d=min(d,E[i].flow);
        for(int i=pre[en]; i+1; i=pre[E[i].st])
        {
```


```
            E[i].flow-=d;
            E[i^1].flow+=d;
            ans+=d*E[i].cost;
        }
    }
    return ans;
}
int main()
{
    while(cin>>n&&n)
    {
        init();
        ///建图
    }
    return 0;
}
```

6.2.2 zkw 费用流 （稠密图）

//注意，不能使用与任何负边权的图上。

```
struct node
{
    int st;
    int en;
    int flow,cost;
    int next;
} E[51000];
int num;
int p[mmax];
void init()
{
    memset(p,-1,sizeof p);
    num=0;
}
void add(int st,int en,int flow,int cost)
{
    E[num].st=st;
```


```
    E[num].en=en;
    E[num].flow=flow;
    E[num].cost=cost;
    E[num].next=p[st];
    p[st]=num++;
    E[num].st=en;
    E[num].en=st;
    E[num].flow=0;
    E[num].cost=-cost;
    E[num].next=p[en];
    p[en]=num++;
}
int dis[mmax],p_l[mmax];;
bool vis[mmax];
int T;
int cost_flow;
int augment(int u,int flow,int st,int en)
{
    if(u==en)
    {
        cost_flow+=flow*dis[st];
        return flow;
    }
    vis[u]=1;
    for(int i=p_l[u]; i+1; i=E[i].next)
    {
        int v=E[i].en;
        if(E[i].flow&&(!vis[v])&&dis[u]==dis[v]+E[i].cost)
        {
            int delta=augment(v,min(flow,E[i].flow),st,en);
            if(delta)
            {
                E[i].flow-=delta;
                E[i^1].flow+=delta;
                p_l[u]=i;
                return delta;
            }
        }
```


```
    }
    p_l[u]=0;
    return 0;
}
bool modifydist(int st,int en)
{
    int delta=inf,i;
    for(i=0; i<=T; i++)
    {
        if(vis[i])
        {
            for(int j=p[i]; j+1; j=E[j].next)
            {
                int v=E[j].en;
                if(E[j].flow&&(!vis[v])&&dis[v]+E[j].cost-dis[i]<delta)
                    delta=dis[v]+E[j].cost-dis[i];
            }
        }
    }
    if(delta==inf) return 0;
    for(i=0; i<=T; i++)
    {
        if(vis[i])
        {
            dis[i]+=delta;
            vis[i]=0;
        }
        p_l[i]=p[i];
    }
    return 1;
}
void zkwflow(int st,int en)
{
    cost_flow=0;
    for(int i=0; i<=T; i++)
        p_l[i]=p[i];
    do
    {
```


```
        while(augment(st,inf,st,en));
    }
    while(modifydist(st,en));
}
```

6.3 最大闭合权图

定义 W[I]代表顶点 I 的权值，新增源点 S，汇点 T。

1、若 W[I]>0，则 S 向 I 连一条容量为 W[I]的边。

2、若 W[I]<0，则 I 向 T 连一条容量为 -W[I] 的边。

3、原图中的边，容量设置为正无穷。

这样，最小割就对应了最大权闭合图，而 总盈利-最大流 就是最大权和。

6.4 最大密度子图

6.4.1 最小割模型

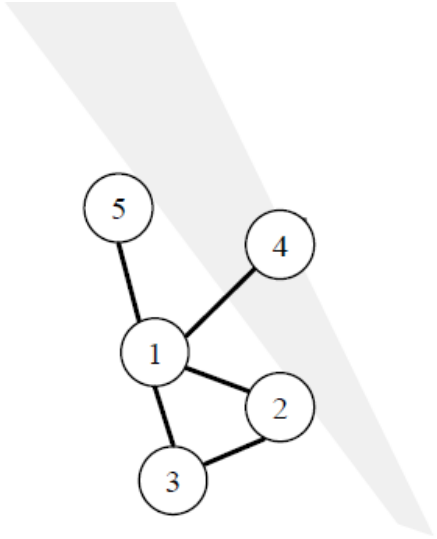
将上面的思路整理一下，即是将原图 $G(V,E)$ 转化为网络 $N=(V_N,E_N)$ 的过程：在原图点集 V 的基础上增加源 s 和汇 t；将每条原无向边 (u,v) 替换为两条容量为 1 的有向边 $\langle u,v\rangle$ 和 $\langle v,u\rangle$；增加连接源 s 到原图每个点 v 的有向边 $\langle s,v\rangle$，容量为 U；增加连接原图每个点 v 到汇 t 的有向边 $\langle v,t\rangle$，容量为 $(U+2g-d_v)$。更形式化地，有：|

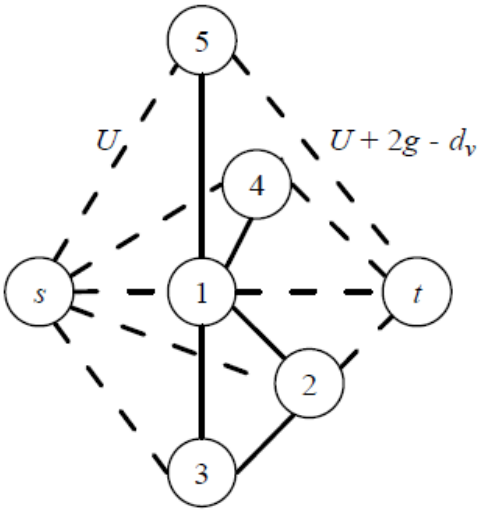
$$V_N=V\bigcup\{s,t\}$$

$$E_N=\{\langle u,v\rangle\,|\,(u,v)\in E\}\bigcup\{\langle s,v\rangle\,|\,v\in V\}\bigcup\{\langle v,t\rangle\,|\,v\in V\}$$

$$\begin{cases}c(u,v)=1 & (u,v)\in E\\ c(s,v)=U & v\in V\\ c(v,t)=U+2g-d_v & v\in V\end{cases}$$

事实上，因为对于 $\forall v\in V$，度数 d_v 不会超过总边数 m，且由于猜测值 g 是非负的，所以令足够大的数 U 取 m，即令 $U=m$，便可保证所有的容量都是非负的。






```
double eps;
struct node
{
    double flow;
    int en;
    int next;
} E[40100];
int p[mmax];
int num;
void init()
{
    memset(p,-1,sizeof p);
    num=0;
}
void add(int st,int en,double flow)
{
    E[num].en=en;
    E[num].flow=flow;
    E[num].next=p[st];
    p[st]=num++;
    E[num].en=st;
    E[num].flow=0;
    E[num].next=p[en];
    p[en]=num++;
}
```


```
int T;//最大点数
int vh[mmax];
int h[mmax];
double find(int u,int st,int en,double f)
{
    if(u==en)
        return f;
    double left=f;
    int tmp=T;
    for(int i=p[u]; i+1; i=E[i].next)
    {
        int v=E[i].en;
        double flow=E[i].flow;
        if(flow>=eps)
        {
            if(h[v]+1==h[u])
            {
                double d=min(left,flow);
                d=find(v,st,en,d);
                left-=d;
                E[i].flow-=d;
                E[i^1].flow+=d;
                if(h[st]>=T+1)
                    return f-left;
                if(fabs(left)<eps)
                    break;
            }
            if(h[v]<tmp)
                tmp=h[v];
        }
    }
    if(fabs(f-left)<eps)
    {
        vh[h[u]]--;
        if(vh[h[u]]==0)
            h[st]=T+1;
        h[u]=tmp+1;
        vh[h[u]]++;
```


```
    }
    return f-left;
}
double isap(int st,int en)
{
    memset(vh,0,sizeof vh);
    memset(h,0,sizeof h);
    vh[0]=T+1;
    double ans=0;
    while(h[st]<=T)
        ans+=find(st,st,en,inf);
    return ans;
}
int n,m;
struct edge
{
    int st,en;
} G[1010];
int D[mmax];
void build(double k)
{
    double U=m;
    init();
    for(int i=1; i<=n; i++)
    {
        add(0,i,U);
        add(i,n+1,U+2*k-D[i]);
    }
    for(int i=1; i<=m; i++)
    {
        add(G[i].st,G[i].en,1);
        add(G[i].en,G[i].st,1);
    }
}
bool fg[mmax];
void Dfs(int u)
{
    fg[u]=1;
```


```
    for(int i=p[u]; i+1; i=E[i].next)
    {
        int v=E[i].en;
        if(!fg[v]&&E[i].flow>0)
            Dfs(v);
    }
}
int main()
{
    while(cin>>n>>m)
    {
        memset(D,0,sizeof D);
        for(int i=1; i<=m; i++)
        {
            scanf("%d %d",&G[i].st,&G[i].en);
            D[G[i].st]++;
            D[G[i].en]++;
        }
        if(m==0)
        {
            printf("1\n1\n\n");
            continue;
        }
        double l=0;
        double r=m;
        eps=1.0/n/n;
        T=n+1;
        while(r-l>=eps)
        {
            double mid=(l+r)/2;
            build(mid);
            double ans=isap(0,n+1);
            ans=(1.0*m*n-ans)/2;
            if(ans>=eps)
                l=mid;
            else
                r=mid;
        }
```


```
        build(l);
        isap(0,n+1);
        memset(fg,0,sizeof fg);
        Dfs(0);
        int ans=0;
        for(int i=1; i<=n; i++)
            if(fg[i])
                ans++;
        cout<<ans<<endl;
        for(int i=1; i<=n; i++)
        {
            if(fg[i])
                printf("%d\n",i);
        }
        puts("");
    }
    return 0;
}
```

6.4.2 最大闭合权图模型

//网络流部分见上面最小割模型

```
int n,m;
struct edge
{
    int st,en;
} G[1010];
void build(double k)
{
    init();
    for(int i=1; i<=m; i++)
    {
        add(i+n,G[i].st,inf);
        add(i+n,G[i].en,inf);
        add(0,i+n,1);
    }
    for(int i=1; i<=n; i++)
```


```
        add(i,m+n+1,k);
}
bool fg[mmax];
void DFS(int u)
{
    fg[u]=1;
    for(int i=p[u]; i+1; i=E[i].next)
    {
        int v=E[i].en;
        if(!fg[v]&&E[i].flow>0)
            DFS(v);
    }
}
int main()
{
    while(cin>>n>>m)
    {
        for(int i=1; i<=m; i++)
            scanf("%d %d",&G[i].st,&G[i].en);
        if(m==0)
        {
            printf("1\n1\n\n");
            continue;
        }
        double S=0;
        double TT=1.0*m;
        eps=1.0/n/n;
        T=n+m+1;
        while(TT-S>=eps)
        {
            double mid=(TT+S)/2;
            build(mid);
            double ans=isap(0,m+n+1);
            ans=1.0*m-ans;
            if(ans>=eps)
                S=mid;
            else
                TT=mid;
```


```
        }
        build(S);
        isap(0,m+n+1);
        memset(fg,0,sizeof fg);
        DFS(0);
        int ans=0;
        for(int i=1; i<=n; i++)
        {
            if(fg[i])
                ans++;
        }
        printf("%d\n",ans);
        for(int i=1; i<=n; i++)
        {
            if(fg[i])
                printf("%d\n",i);
        }
        puts("");
    }
    return 0;
}
```

7、生成树计数

Matrix-Tree 定理(Kirchhoff 矩阵-树定理)。Matrix-Tree 定理是解决生成树计数问题最有力的武器之一。它首先于 1847 年被 Kirchhoff 证明。

G 的度数矩阵 D[G]是一个 n*n 的矩阵，并且满足：当 i≠j 时,dij=0；当 i=j 时，dij 等于 vi 的度数。

G 的邻接矩阵 A[G]也是一个 n*n 的矩阵， 并且满足：如果 vi、vj 之间有边直接相连，则 aij=1，否则为 0。

我们定义 G 的 Kirchhoff 矩阵(也称为拉普拉斯算子)C[G]为 C[G]=D[G]-A[G]，则 Matrix-Tree 定理可以描述为：G 的所有不同的生成树的个数等于其 Kirchhoff 矩阵 C[G]任何一个 n-1 阶主子式的行列式的绝对值。所谓 n-1 阶主子式，就是对于 r(1≤r≤n)，将 C[G]的第 r 行、第 r 列同时去掉后得到的新矩阵，用 Cr[G]表示。

8、最大团搜索

```
#include<cstdio>
#include<string>
#include<queue>
```


```
#include<map>
#include<algorithm>
using namespace std;
const int inf = 0x3fffffff;
const int mmax = 60;
int G[mmax][mmax];
int list[mmax][mmax];
int mc[mmax],len[mmax];
int ans;
bool found;
int n;
void dfs(int size)
{
    int i,j,k;
    if(len[size]==0)
    {
        if(size>ans)
        {
            ans=size;
            found=1;
        }
        return ;
    }
    for(k=0; k<len[size] && !found; k++)
    {
        if(size+len[size]-k<=ans)
            break;
        i=list[size][k];
        if(size+mc[i]<=ans)
            break;
        for(j=k+1,len[size+1]=0; j<len[size]; j++)
        {
            if(G[i][list[size][j]] )
                list[size+1][len[size+1]++]=list[size][j];
        }
        dfs(size+1);
    }
}
```


```
void max_cluster()
{
    int i,j;
    mc[n]=ans=1;
    for(i=n-1; i; i--)
    {
        found=0;
        len[1]=0;
        for(j=i+1; j<=n; j++)
        {
            if(G[i][j])
                list[1][len[1]++]=j;
        }
        dfs(1);
        mc[i]=ans;
    }
}
int main()
{
    while(~scanf("%d",&n) && n)
    {
        for(int i=1; i<=n; i++)
            for(int j=1; j<=n; j++)
                scanf("%d",&G[i][j]);
        max_cluster();
        printf("%d\n",ans);
    }
    return 0;
}
```


三、数据结构

1、kmp

//O(n+m)的复杂度求出模式串和母串的匹配

```
int next[mmax];
void get_next(char *p)
{
    int m=strlen(p);
    next[0]=next[1]=0;
    for(int i=1; i<m; i++)
    {
        int j=next[i];
        while(j && p[i]!=p[j]) j=next[j];
        next[i+1]= p[i]==p[j]?j+1:0;
    }
}
bool fg[mmax];
void kmp(char *T,char *s)
{
    memset(fg,0,sizeof fg);
    int n=strlen(T),m=strlen(s);
    get_next(s);
    int j=0;
    for(int i=0; i<n; i++)
    {
        while(j && T[i] != s[j]) j=next[j];
        if(s[j] == T[i]) j++;
        if(j==m)
            fg[i-m+2]=1;
    }
}
```


2、扩展 kmp

//扩展 kmp 求一个字符串 s 的 所有 s[i-n] 与 s[1-n]的 Lcp

```
const int mmax=1100;
int next[mmax],ret[mmax];

void get_next(char *s)
{
    int n=strlen(s);
    int i,j,k;
    for(j=0; 1+j<n && s[j]==s[1+j]; j++);
    next[1]=j;
    k=1;
    for(i=2; i<n; i++)
    {
        int len=k+next[k],L=next[i-k];
        if(L<len-i)
            next[i]=L;
        else
        {
            for(j=max(0,len-i); i+j<n && s[j]==s[i+j]; j++);
            next[i]=j;
            k=i;
        }
    }
    next[0]=n;
}
void ex_kmp(char *T,char *s)
{
    int n=strlen(T),m=strlen(s);
    int i,j,k;
    for(j=0; j<n && j<m &&T[j]==s[j]; j++);
    ret[0]=j;
    k=0;
    for(i=1; i<n; i++)
    {
        int len=k+ret[k],L=next[i-k];
```


```
        if(L<len-i)
            ret[i]=L;
        else
        {
            for(j=max(0,len-i); j<m && i+j<n && s[j]==T[i+j]; j++);
            ret[i]=j;
            k=i;
        }
    }
}
```

3、Aho-Corasick 自动机

//多模式匹配

```
struct AC_tire
{
    int ch[mmax][26];
    int cnt[mmax],Fail[mmax],last[mmax],id[mmax];
    int ans[200];
    int num;
    char patter[200][200];
    char str[1000010];
    void init()
    {
        memset(ch[0],0,sizeof ch[0]);
        cnt[0]=0;
        num=0;
        memset(ans,0,sizeof ans);
    }
    void read(int n)
    {
        for(int i=0; i<n; i++)
        {
            scanf("%s",patter[i]);
            Insert(i,patter[i]);
        }
    }
```


```
int newnode()
{
    num++;
    memset(ch[num],0,sizeof ch[num]);
    cnt[num]=0;
    return num;
}
void insert(int id_s,char *s)
{
    int u=0;
    for(int i=0; s[i]; i++)
    {
        if(!ch[u][s[i]-'a'])
            ch[u][s[i]-'a']=newnode();
        u=ch[u][s[i]-'a'];
    }
    cnt[u]++;
    id[u]=id_s;
}
void get_Fail()
{
    queue<int>q;
    Fail[0]=0;
    for(int i=0; i<26; i++)
    {
        int u=ch[0][i];
        if(u)
        {
            q.push(u);
            Fail[u]=0;
            last[u]=0;
        }
    }
    while(!q.empty())
    {
        int x=q.front();
        q.pop();
        for(int i=0; i<26; i++)
```


```
            {
                int u=ch[x][i];
                if(!u)
                {
                    ch[x][i]=ch[Fail[x]][i];
                    continue;
                }
                q.push(u);
                int v=Fail[x];
                while(v && !ch[v][i]) v=Fail[v];
                Fail[u]=ch[v][i];
                last[u]=cnt[Fail[u]]? Fail[u]:last[Fail[u]];
            }
        }
    }
    void get__(int x)
    {
        if(x)
        {
            ans[id[x]]+=cnt[x];
            get__(last[x]);
        }
    }
    void query(char *T)
    {
        get_Fail();
        int n=strlen(T);
        int j=0;
        for(int i=0; i<n; i++)
        {
            j=ch[j][T[i]-'a'];
            if(cnt[j])
                get__(j);
            else if(last[j])
                get__(last[j]);
        }
    }
};
```


4、后缀数组

```
const int mmax=210000;
char s[mmax];
int sa[mmax],t[mmax],t2[mmax],c[mmax],n;
void build_sa(int m)
{
    int i,*x=t,*y=t2;
    for(i=0; i<m; i++) c[i]=0;
    for(i=0; i<n; i++) c[ x[i]=s[i]]++;
    for(i=1; i<m; i++) c[i]+=c[i-1];
    for(i=n-1; i>=0; i--) sa[--c[x[i]]]=i;
    for(int k=1; k<=n; k<<=1)
    {
        int p=0;
        for(i=n-k; i<n; i++) y[p++]=i;
        for(i=0; i<n; i++) if(sa[i]>=k) y[p++]=sa[i]-k;

        for(i=0; i<m; i++) c[i]=0;
        for(i=0; i<n; i++) c[x[y[i]]]++;
        for(i=1; i<m; i++) c[i]+=c[i-1];
        for(i=n-1; i>=0; i--) sa[--c[x[y[i]]]]=y[i];
        swap(x,y);
        p=1;
        x[sa[0]]=0;
        for(i=1; i<n; i++)
            x[sa[i]] = y[sa[i-1]]==y[sa[i]] && y[sa[i-1]+k]==y[sa[i]+k]?p-1:p++;
        if(p>=n) break;
        m=p;
    }
}
int m;
int cmp_suffix(char *patter ,int p)
{
    return strncmp(patter,s+sa[p],m);
}
int find(char *P) //查找是否出现子串P
{
```


```
    m=strlen(P);
    if(cmp_suffix(P,0)<0) return -1;
    if(cmp_suffix(P,n-1)>0) return -1;
    int l=0,r=n;
    while(l<r)
    {
        int mid=(l+r)>>1;
        int res=cmp_suffix(P,m);
        if(!res) return m;
        if(res<0)
            r=mid;
        else
            l=mid+1;
    }
    return -1;

}
int rank[mmax],h[mmax];
void get_h()
{
    int i,j,k=0;
    for(i=0; i<n; i++) rank[sa[i]]=i;
    for(i=0; i<n; i++)
    {
        if(k) k--;
        if(rank[i])
        {
            j=sa[rank[i]-1];
            while(s[i+k]==s[j+k]) k++;
            h[rank[i]]=k;
        }
    }
}
```

5、Manacher 求最长回文子串（O（n））

```
//通过插入字符使的回文串一定为奇数
//hihocoder 1032
```


```
const int mmax = 1000010;
const int inf=0x3fffffff;
char str[mmax];
int s[2*mmax];
int p[2*mmax];
int Manacher(int n)
{
    int ans=0;
    memset(p,0,sizeof p);
    int id=0;
    p[0]=1;
    for(int i=1; i<n; i++)
    {
        if(id+p[id]-1<i)
            p[i]=1;
        else
            p[i]=min(p[2*id-i],id+p[id]-i);
        while(i+p[i]<n && i-p[i]>=0&&s[i+p[i]]==s[i-p[i]]  )
            p[i]++;
        if(i+p[i]>id+p[id])
            id=i;
        ans=max(ans,p[i]-1);
    }
    return ans;
}
int main()
{
    int t;
    scanf("%d",&t);
    while(t--)
    {
        scanf("%s",str);
        int len=strlen(str);
        int cnt=0;
        for(int i=0; i<len; i++)
        {
            s[cnt++]='#';
```


```
            s[cnt++]=str[i];
        }
        s[cnt++]='#';
        printf("%d\n",Manacher(cnt));
    }
    return 0;
}
```

6、树链剖分

//此处为点权，若应用于边权，需要将边权向下映射成点权。并且 solve()函数的部分也需要注意。

//hdu 5274 求路径上出现奇数次的点

```
#include<stdio.h>
#include<string.h>
#include<algorithm>
using namespace std;
const int mmax = 100010;
const int inf=0x3fffffff;
struct edge
{
    int st,en;
    int next;
} E[2*mmax];
int p[mmax],num;
int fa[mmax],son[mmax],top[mmax];
int deep[mmax],f_node[mmax],Newid[mmax],w[mmax];
bool vis[mmax];
void add(int st,int en)
{
    E[num].st=st;
    E[num].en=en;
    E[num].next=p[st];
    p[st]=num++;
}
void init()
{
```


```
    memset(p,-1,sizeof p);
    num=0;
}
struct tree
{
    int l,r;
    int sum;
    int mid()
    {
        return (l+r)>>1;
    }
} T[4*mmax];
void build(int id,int l,int r)
{
    T[id].l=l,T[id].r=r;
    if(l==r)
    {
        T[id].sum=w[f_node[l]];
        return ;
    }
    int mid=T[id].mid();
    build(id<<1,l,mid);
    build(id<<1|1,mid+1,r);
    T[id].sum=T[id<<1].sum^T[id<<1|1].sum;
}
void updata(int id,int pos,int val)
{
    if(T[id].l==T[id].r)
    {
        T[id].sum=val;
        return ;
    }
    int mid=T[id].mid();
    if(mid>=pos)
        updata(id<<1,pos,val);
    else
        updata(id<<1|1,pos,val);
    T[id].sum=T[id<<1].sum^T[id<<1|1].sum;
```


```
}
int query(int id,int l,int r)
{
    if(l<=T[id].l&&T[id].r<=r)
        return T[id].sum;
    int mid=T[id].mid();
    int ans=0;
    if(mid>=l)
        ans^=query(id<<1,l,r);
    if(mid<r)
        ans^=query(id<<1|1,l,r);
    return ans;
}
int dfs(int u)
{
    vis[u]=1;
    int cnt=1,tmp=0,e=0;
    for(int i=p[u]; i+1; i=E[i].next)
    {
        int v=E[i].en;
        if(!vis[v])
        {
            fa[v]=u;
            deep[v]=deep[u]+1;
            int tt=dfs(v);
            cnt+=tt;
            if(tmp<tt)
            {
                tmp=tt;
                e=v;
            }
        }
    }
    son[u]=e;
    return cnt;
}
int now_cnt;
void new_id(int u)//重新编号
```


```
{
    f_node[now_cnt]=u;//编号我now_cnt的点是u
    Newid[u]=now_cnt;//点u的编号为now_cnt
    now_cnt++;
    vis[u]=1;
    if(son[u])
    {
        top[son[u]]=top[u];
        new_id(son[u]);
    }
    for(int i=p[u]; i+1; i=E[i].next)
    {
        int v=E[i].en;
        if(!vis[v])
            new_id(v);
    }
}
int solve(int x,int y)//查询x到路径上的点权和
{
    int ans=0;
    while(top[x]!=top[y])
    {
        if(deep[top[x]]<deep[top[y]])
            swap(x,y);
        ans^=query(1,Newid[top[x]],Newid[x]);
        x=fa[top[x]];
    }
    if(deep[x]>deep[y])
        swap(x,y);
    ans^=query(1,Newid[x],Newid[y]);
    return ans;
}
int main()
{
    int n,q;
    int t;
    scanf("%d",&t);
    while(t--)
```


```
{
    scanf("%d %d",&n,&q);
    init();
    for(int i=0; i<n-1; i++)
    {
        int u,v;
        scanf("%d %d",&u,&v);
        add(u,v);
        add(v,u);
    }
    for(int i=1; i<=n; i++)
    {
        scanf("%d",&w[i]);
        if(w[i]==0)
            w[i]=mmax;
    }
    fa[1]=1;
    deep[1]=0;
    memset(vis,0,sizeof vis);
    for(int i=1; i<=n; i++)
        top[i]=i;
    dfs(1);
    memset(vis,0,sizeof vis);
    now_cnt=1;
    new_id(1);
    build(1,1,n);
    while(q--)
    {
        int d,x,y;
        scanf("%d %d %d",&d,&x,&y);
        if(d==0)
        {
            if(y==0)
                y=mmax;
            updata(1,Newid[x],y);
        }
        else
        {
```


```
                if(x>y)
                    swap(x,y);
                int tmp=solve(x,y);
                if(tmp==0)
                    puts("-1");
                else
                {
                    if(tmp==mmax)
                        tmp=0;
                    printf("%d\n",tmp);
                }
            }
        }
    }
    return 0;
}
```

7、dfs 序+lca+树状数组

//hdu 5274

```
#include<stdio.h>
#include<cmath>
#include<string.h>
#include<algorithm>
using namespace std;
const int mmax = 500010;
const int inf=0x3fffffff;
struct edge
{
    int st,en;
    int next;
} E[2*mmax];
int p[mmax],num,w[mmax];
void add(int st,int en)
{
    E[num].st=st;
    E[num].en=en;
```


```
    E[num].next=p[st];
    p[st]=num++;
}
void init()
{
    memset(p,-1,sizeof p);
    num=0;
}
int Times;
int deep[mmax],First[mmax],Last[mmax];
int C[mmax];
int fa[mmax][20];
int low_bit(int x)
{
    return x&(-x);
}
int n;
void update(int x)
{
    for(int i=First[x]; i<=n; i+=low_bit(i))
        C[i]^=w[x];
    for(int i=Last[x]; i<=n; i+=low_bit(i))
        C[i]^=w[x];
}
int get_sum(int x)
{
    int fg=0;
    for(int i=First[x]; i>0; i-=low_bit(i))
        fg^=C[i];
    return fg;
}
void dfs(int u,int Deep)
{
    Times++;
    First[u]=Times;
    deep[u]=Deep;
    for(int i=1; (1<<i)<=deep[u]; i++)
        fa[u][i]=fa[ fa[u][i-1]][i-1];
```


```
    for(int i=p[u]; i+1; i=E[i].next)
    {
        int v=E[i].en;
        if(deep[v]==-1)
        {
            fa[v][0]=u;
            dfs(v,Deep+1);
        }
    }
    Last[u]=Times+1;
}
int lca(int x,int y)
{
    if(deep[x]<deep[y])
        swap(x,y);
    for(int i=19; i>=0; i--)
    {
        if(fa[x][i]!=-1 && deep[fa[x][i]]>=deep[y])
            x=fa[x][i];
        if(deep[x]==deep[y])
            break;
    }
    if(x==y)
        return x;
    for(int i=19; i>=0; i--)
    {
        if(fa[x][i]!=fa[y][i])
            x=fa[x][i],y=fa[y][i];
    }
    return fa[x][0];
}
int main()
{
    int q;
    while(~scanf("%d",&n))
    {
        init();
```


```
    for(int i=1; i<=n; i++)
        scanf("%d",&w[i]);
    for(int i=0; i<n-1; i++)
    {
        int u,v;
        scanf("%d %d",&u,&v);
        add(u,v);
        add(v,u);
    }
    Times=0;
    memset(deep,-1,sizeof deep);
    memset(fa,-1,sizeof fa);
    dfs(1,0);
    memset(C,0,sizeof C);
    for(int i=1; i<=n; i++)
        update(i);
    scanf("%d",&q);
    while(q--)
    {
        char ss[2];
        int x,y;
        scanf("%s %d %d",ss,&x,&y);
        if(ss[0]=='C')
        {
            update(x);
            w[x]=y;
            update(x);
        }
        else
        {
            int tmp=get_sum(x)^get_sum(y)^w[lca(x,y)];
            if(tmp)
                puts("Yes");
            else
                puts("No");
        }
    }
}
```


```
    return 0;
}
```

8、LCT（动态树问题）

```
#include<cstdio>
#include<algorithm>
#include<cstring>
using namespace std;
const int mmax  = 200010;
int ch[mmax][2];
int Pnt[mmax];
int Data[mmax];
int Sum[mmax];
int Rev[mmax];
int List[mmax];
int Total;
inline bool isRoot(int t)
{
    return !Pnt[t] || (ch[Pnt[t]][0]!=t && ch[Pnt[t]][1]!=t);
}
void Reverse(int cur)
{
    if(!Rev[cur])
        return ;
    swap(ch[cur][0],ch[cur][1]);
    Rev[ch[cur][0]]^=1;
    Rev[ch[cur][1]]^=1;
    Rev[cur]=0;
}
inline  void Update(int cur)
{
    Sum[cur]=Sum[ch[cur][0]]+Sum[ch[cur][1]]+Data[cur];
}
void Rotate(int son,int d)
{
    if(isRoot(son))
        return ;
```


```
    int fax=Pnt[son];
    Pnt[son]=Pnt[fax];
    if(!isRoot(fax))
        ch[Pnt[fax]][fax==ch[Pnt[fax]][1]]=son;
    ch[fax][d^1]=ch[son][d];
    if(ch[son][d])
        Pnt[ch[son][d]]=fax;
    ch[son][d]=fax;
    Pnt[fax]=son;
    Update(fax);
    Update(son);
}
void Splay(int cur)
{
    int pnt,anc;
    List[++Total]=cur;
    for(int i=cur; !isRoot(i); i=Pnt[i])
        List[++Total]=Pnt[i];
    for(; Total; Total--)
        if(Rev[List[Total]])
            Reverse(List[Total]);
    while(!isRoot(cur))
    {
        pnt=Pnt[cur];
        if(isRoot(pnt))
            Rotate(cur,cur==ch[pnt][0]);
        else
        {
            anc=Pnt[pnt];
            int d=(pnt==ch[anc][0]);
            if(cur==ch[pnt][d])
                Rotate(cur,d^1),Rotate(cur,d);
            else
                Rotate(pnt,d),Rotate(cur,d);
        }
    }
}
int Access(int u) //打通u到根节点的一条路径 返回根节点
```


```
{
    int v=0;
    for(; u; u=Pnt[u])
    {
        Splay(u);
        ch[u][1]=v;
        v=u;
        Update(u);
    }
    for(; ch[v][0]; v=ch[v][0]);
    return v;
}
void Modify(int x,int d) //修改x点的点权为d
{
    Splay(x);
    Data[x]=d;
    Update(x);
}
int Query(int x,int y) //询问x到y路径上信息
{
    int rx=Access(x),ry=Access(y);
    if(rx!=ry)
        return -1;
    for(int u=x,v=0; u; u=Pnt[u])
    {
        Splay(u);
        if(!Pnt[u])
            return Sum[ch[u][1]]+Data[u]+Sum[v];
        ch[u][1]=v;
        Update(u);
        v=u;
    }
}
void Join(int x,int y) //加入点边x-y,y是x的父亲
{
    int rx=Access(x),ry=Access(y);
    if(rx!=ry)
    {
```


```
        Splay(x);
        ch[x][1]=0;
        Rev[x]=1;
        Pnt[x]=y;
        Update(x);
    }
}
void cut(int x) //切断点x的其父亲的边
{
    Access(x);
    Splay(x);
    Pnt[ch[x][0]]=0;
    ch[x][0]=0;
    Update(x);
}
int k[mmax];
int main()
{
    int n;
    scanf("%d",&n);
    memset(Sum,0,sizeof Sum);
    memset(Rev,0,sizeof Rev);
    memset(Pnt,0,sizeof Pnt);
    memset(ch,0,sizeof ch);
    memset(Data,0,sizeof Data);
    Total=0;
    for(int i=1; i<=n; i++)
        Data[i]=1;
    for(int i=1; i<=n; i++)
    {
        scanf("%d",&k[i]);
        if(i+k[i]<=n)
            Join(i,i+k[i]);
        else
            Join(i,n+1);
    }
    int q;
    scanf("%d",&q);
```


```
    while(q--)
    {
        int d,x,y;
        scanf("%d",&d);
        if(d==1)
        {
            scanf("%d",&x);
            x++;
            printf("%d\n",Query(x,n+1));
        }
        else
        {
            scanf("%d %d",&x,&y);
            x++;
            cut(x);
            k[x]=y;
            if(x+k[x]<=n)
                Join(x,x+k[x]);
            else
                Join(x,n+1);
        }
    }
    return 0;
}
```

9、后缀自动机（sam）

// hdu4622 求字符串 l-r 不同的字符串的个数

```
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int mmax = 4010;
int sum[mmax];
struct Sam
{
    char s[mmax];
```


```
int son[mmax][26],pre[mmax],step[mmax],last,total;
int cnt[mmax];
int ans;//不同的字符串的个数
void init()
{
    total=last=ans=0;
    memset(son[0],0,sizeof son[0]);
    step[0]=0;
    pre[0]=0;
    cnt[0]=1;
}
inline void push_back(int v)
{
    step[++total]=v;
    pre[total]=0;
    memset(son[total],0,sizeof son[total]);
    cnt[total]=0;
}
void Extend(int ch)
{
    push_back(step[last]+1);
    int p=last,np=total;
    for(; !son[p][ch]; p=pre[p])
    {
        son[p][ch]=np;
        cnt[np]+=cnt[p];
    }
    if(son[p][ch]==np) pre[np]=0;
    else
    {
        int q=son[p][ch];
        if(step[q]!=step[p]+1)
        {
            push_back(step[p]+1);
            int nq=total;
            memcpy(son[nq],son[q],sizeof(son[q]));
            pre[nq]=pre[q];
            pre[q]=pre[np]=nq;
```


```
                for(; son[p][ch]==q; p=pre[p])
                {
                    cnt[son[p][ch]]-=cnt[p];//这一步很重要
                    son[p][ch]=nq;
                    cnt[nq]+=cnt[p];
                }
            }
            else
                pre[np]=q;
        }
        ans+=cnt[np];
        last=np;
    }

} sam;
int ans[2010][2010];
int main()
{
    int t;
    scanf("%d",&t);
    while(t--)
    {
        scanf("%s",sam.s);
        int len=strlen(sam.s);
        for(int i=0; i<len; i++)
        {
            sam.init();
            for(int j=i; j<len; j++)
            {
                sam.Extend(sam.s[j]-'a');
                ans[i][j]=sam.ans;
            }
        }
        int q;
        scanf("%d",&q);
        while(q--)
        {
            int l,r;
```


```
            scanf("%d %d",&l,&r);
            l--,r--;
            printf("%d\n",ans[l][r]);
        }
    }
    return  0;
}
```

10、2d 线段树模板

```
#include<iostream>
#include<cstring>
#include<algorithm>
using namespace std;
const int maxn =(400+10)*4;
struct IntervalTree2D
{
    int sum[maxn][maxn],n,m;
    int xo,xleaf,x1,y1,x2,y2,x,y,v,sum;
    void query1D(int o,int L,int R)
    {
        if(y1<=L&&R<=y2)
        {
            sum^=sum[xo][o];
        }
        else
        {
            int M=L+(R-L)/2;
            if(y1<=M) query1D(o*2,L,M);
            if(M<y2) query1D(o*2+1,M+1,R);
        }
    }
    void query2D(int o,int L,int R)
    {
        if(x1<=L&&R<=x2)
        {
            xo=o;
            query1D(1,1,m);
```


```
        }
        else
        {
            int M=L+(R-L)/2;
            if(x1<=M) query2D(o*2,L,M);
            if(M<x2) query2D(o*2+1,M+1,R);
        }
    }
    void modify1D(int o,int L,int R)
    {
        if(L==R)
        {
            if(xleaf)
            {
                sum[xo][o]=v;
                return;
            }
            sum[xo][o]=sum[xo*2][o]^sum[xo*2+1][o];
        }
        else
        {
            int M=L+(R-L)/2;
            if(y<=M) modify1D(o*2,L,M);
            else modify1D(o*2+1,M+1,R);
            sum[xo][o]=sum[xo][o*2]^sum[xo][o*2+1];
        }
    }
    void modify2D(int o,int L,int R)
    {
        if(L==R)
        {
            xo=o;
            xleaf=1;
            modify1D(1,1,m);
        }
        else
        {
            int M=L+(R-L)/2;
```


```
            if(x<=M) modify2D(o*2,L,M);
            else modify2D(o*2+1,M+1,R);
            xo=o;
            xleaf=0;
            modify1D(1,1,m);
        }
    }
    void query()
    {
        sum=0;
        query2D(1,1,n);
    }
    void modify()
    {
        modify2D(1,1,n);
    }
};
IntervalTree2D t;
int main()
{
    int T,ca,q;
    scanf("%d",&T);
    for(ca=1; ca<=T; ca++)
    {
        scanf("%d %d",&n,&m,&q);
        t.n=n;
        t.m=m;
        for(i=1; i<=n; i++)
            for(j=1; j<=m; j++)
            {
                scanf("%d",&t.v);
                t.x=i;
                t.y=j;
                t.modify();
            }
        while(q--)
        {
            int X1,X2,Y1,Y2;
```


```
            int op;
            scanf("%d",&op);
            if(op==1)
            {
                scanf("%d %d %d %d",&t.x1,&t.y1,&t.x2,&t.y2);
                t.query();
                if(t.sum)
                    puts("Yes");
                else
                    puts("No");
            }
            else
            {
                scanf("%d %d %d",&t.x,&t.y,&t.v);
                t.modify();
            }
        }
    }
    return 0;
}
```

11、平衡树

11.1 treap

```
struct Treap
{
    int key,sz,r;//关键字，大小，随机数
    int cnt;
    Treap * ch[2];
    Treap(int key0)
    {
        key=key0;
        sz=cnt=1;
        r=rand();
        ch[0]=ch[1]=0;
    }
```


```
    void up()
    {
        sz=cnt;
        sz+=(ch[0]?ch[0]->sz:0);
        sz+=(ch[1]?ch[1]->sz:0);
    }
};
typedef Treap *  Treap_Point;
void Treap_Clear(Treap_Point &p) //内存清空
{
    if(!p)
        return ;
    Treap_Clear(p->ch[0]);
    Treap_Clear(p->ch[1]);
    delete p,p=0;
}
void Treap_Rotate(Treap_Point &fa,int d) //d=0 左旋 d=1 右旋
{
    Treap_Point son=fa->ch[d^1];
    fa->ch[d^1]=son->ch[d];
    son->ch[d]=fa;
    fa->up();
    son->up();
    fa=son;
}
void Treap_Insert(Treap_Point &p,int key) // 添加一个关键字
{
    if(!p)
    {
        p=new Treap(key);
        return ;
    }
    if(p->key==key)
    {
        p->cnt++;
        p->up();
        return ;
    }
```


```
    int d=p->key < key;
    Treap_Insert(p->ch[d],key);
    if(p->r < p->ch[d]->r)
        Treap_Rotate(p,d^1);
    p->up();
}
Treap_Point Treap_Serach(Treap_Point root,int x) //查找关键字为x的节点
{
    Treap_Point  p=root;
    while(p && p->key!=x)
        p=p->ch[p->key <x];
    if(p && p->key==x)
        return p;
    return 0;
}
int Treap_Kth(Treap_Point root,int k) //查找第k大的关键字
{
    if(!root || !(k>=1 && k<=root->sz))
        return -1;
    Treap_Point p=root;
    while(1)
    {
        int num=p->ch[0]?p->ch[0]->sz:0;
        if(num+1<=k && k<=num+p->cnt)
            return p->key;
        else if(num+p->cnt >k)
            p=p->ch[0];
        else
        {
            k-=num+p->cnt;
            p=p->ch[1];
        }
    }
}
int Treap_Pre(Treap_Point root,int &key) //前驱
{
    int ans=0,fg=0;
```


```
    Treap_Point p=root;
    while(p)
    {
        if(p->key < key)
            fg=1,ans=p->key;
        p=p->ch[p->key < key];
    }
    if(!fg)
        return -1;
    return ans;
}
int Treap_Suc(Treap_Point root,int &key) //后继
{
    int ans,fg=0;
    Treap_Point p=root;
    while(p)
    {
        if(p->key > key)
            fg=1, ans=p->key;
        p = p->ch[p->key <= key];
    }
    if(!fg)
        return -1;
    else
        return ans;
}
void Treap_Delete(Treap_Point &p,int &key) //删除关键字key
{
    if(key == p->key)
    {
        if(p->cnt >1)
        {
            p->cnt--;
            p->sz--;
            return ;
        }
        if(p->ch[0] && p->ch[1])
        {
```


```
            bool d = (p->ch[0]->r > p->ch[1]->r);
            Treap_Rotate(p, d);
            Treap_Delete(p->ch[d], key);
        }
        else
        {
            Treap_Point del=p;
            p = p->ch[p->ch[0] ? 0:1];
            delete del,del=0;
        }
    }
    else
        Treap_Delete(p->ch[p->key < key], key);
    if(p)
        p->up();
}
int Treap_Rank(Treap_Point root ,int &key) //关键字key的排名
{
    int ans=0;
    Treap_Point p=root;
    while(1)
    {
        if(key == p->key)
        {
            ans += 1+(p->ch[0] ? p->ch[0]->sz : 0);
            break;
        }
        else if(key < p->key)
            p = p->ch[0];
        else
        {
            ans += p->cnt+(p->ch[0] ? p->ch[0]->sz : 0);
            p = p->ch[1];
        }
    }
    return ans;
}
int main()
```


```
{
    Treap_Point root=NULL;
    int n;
    while(cin>>n)
    {
        Treap_Clear(root);
        while(n--)
        {
            int d,x;
            scanf("%d %d",&d,&x);
            if(d==1)
                Treap_Insert(root,x);
            if(d==2)
                Treap_Delete(root,x);
            if(d==3)
                printf("%d\n",Treap_Rank(root,x));
            if(d==4)
                printf("%d\n",Treap_Kth(root,x));
            if(d==5)
                printf("%d\n",Treap_Pre(root,x));
            if(d==6)
                printf("%d\n",Treap_Suc(root,x));
        }
    }
    return 0;
}
```

11.2 splay

```
const int mmax= 50010;
const int mod=1000000007;
struct Splay
{
    int key,val;
    int fg,num;
    Splay *ch[2],*fa;
    Splay(int key=0,int val=0,Splay *fa=0):key(key),val(val),fa(fa)
    {
```


```
        fg=1;
        num=0;
        ch[0]=ch[1]=0;
    }
    void push_down()
    {
        if(num)
        {
            if(ch[0])
            {
                ch[0]->key=num+fg*ch[0]->key;
                ch[0]->num=num+fg*ch[0]->num;
                ch[0]->fg=fg*ch[0]->fg;
            }
            if(ch[1])
            {
                ch[1]->key=num+fg*ch[1]->key;
                ch[1]->num=num+fg*ch[1]->num;
                ch[1]->fg=fg*ch[1]->fg;
            }
            if(fg<0)
                swap(ch[0],ch[1]);
            num=0;
            fg=1;
        }
    }
};
typedef Splay* Splay_Point;
void  Splay_Clear(Splay_Point &p)  //清空
{
    if(!p)
        return ;
    Splay_Clear(p->ch[0]);
    Splay_Clear(p->ch[1]);
    delete p,p=0;
}
void Splay_Rotate( Splay_Point &son , bool d )
{
```


```
    Splay_Point fax = son->fa;
    son->push_down();
    fax->push_down();
    son->fa = fax->fa;
    if( fax->fa )
        fax->fa->ch[fax==fax->fa->ch[1]] = son;
    fax->ch[d^1] = son->ch[d];
    if( son->ch[d] )
        son->ch[d]->fa = fax;
    son->ch[d] = fax;
    fax->fa = son;
}
void Splay_Splay( Splay_Point &root,Splay_Point &p , Splay_Point goal  )//p节点旋转到goal下面
若为空 则旋转到根节点
{
    while(p->fa != goal)
    {
        p->push_down();
        if(p->fa->fa == goal)
            Splay_Rotate(p, p==p->fa->ch[0]);
        else
        {
            Splay_Point fax=p->fa;
            bool d = (fax == fax->fa->ch[0]);
            if(p == p->fa->ch[d])
                Splay_Rotate(p, d^1), Splay_Rotate(p, d);
            else
                Splay_Rotate(fax, d), Splay_Rotate(p, d);
        }
    }
    if(!goal)
        root = p;
}
void  Splay_Insert(Splay_Point &root,int key,int val)  //添加节点
{
    Splay_Point p=root,fax=0;
    while(p)
    {
```


```
        fax=p;
        p=p->ch[p->key < key];
    }
    p= new Splay(key,val,fax);
    if(fax)
        fax->ch[fax->key < key]=p;
    Splay_Splay(root,p,0);
}
Splay_Point Splay_Serach(Splay_Point root,int key)  //查找关键字key
{
    Splay_Point p=root;
    while(p)
    {
        if(p->key == key)
            return p;
        p->push_down();
        p=p->ch[p->key < key];
    }
    return 0;
}
void Splay_Delete(Splay_Point &root, Splay_Point &del)  //删除节点
{
    Splay_Splay(root,del, 0);
    if(del->ch[0] == 0)
    {
        if(del->ch[1] == 0)
            delete root,root=0;
        else
        {
            del->ch[1]->fa = 0;
            root = del->ch[1];
            delete del,del=0;
        }
    }
    else
    {
        Splay_Point tt=del->ch[0];
        while(tt->ch[1])
```


```
        {
            tt->push_down();
            tt = tt->ch[1];
        }
        Splay_Splay(root,tt, del);
        tt->fa = 0;
        tt->ch[1] = del->ch[1];
        if(del->ch[1])
            del->ch[1]->fa = tt;
        root = tt;
        delete del,del=0;
    }
}
void updata(Splay_Point &root ,int l ,int r)
{
    Splay_Point p1=Splay_Serach(root,l-1);
    Splay_Splay(root,p1,0);
    Splay_Point p2=Splay_Serach(root,r+1);
    Splay_Splay(root,p2,p1);
    p2=p2->ch[0];
    p2->key=l+r-p2->key;
    p2->num=l+r-p2->num;
    p2->fg=-p2->fg;
}
int main()
{
    int n,m;
    Splay_Point root=0;
    while(cin>>n>>m)
    {
        Splay_Clear(root);
        for(int i=0; i<=n+1; i++)
            Splay_Insert(root,i,i);
        while(m--)
        {
            int l,r;
            scanf("%d %d",&l,&r);
            updata(root,l,r);
```


```
        }
        for(int i=1; i<=n; i++)
        {
            Splay_Point p=Splay_Serach(root,i);
            printf("%d ",p->val);
        }
    }
    return 0;
}
```


四、计算几何

1、基础几何

---

double Dot(Vector A,Vector B) 点积

double Length(Vector A) 取模

double Angle(Vector A,Vector B) 夹角

double Cross(Vector A,Vector B) 叉积

double Area2(Point A,Point B,Point C) 有向面积

Vector Ratate(Vector A,double rad) 向量旋转

Vector Normal(Vector A) 法向量

Point GetLineIntersection(Point P,Point Q,Vector v,Vector w) 直线相交 line1 P+tv line2 Q+tw

double Dis_Point_Line(Point P,Point A,Point B) 点到直线距离

Point GetLineProjection(Point P,Point A,Point B) 点在直线上投影

bool SegmentProperIntersection(Point a1,Point a2,Point b1,Point b2) 线段规范相交

bool OnSegment(Point p,Point a1,Point a2) 点在直线上

double PolygonArea(Point *p,int n) 多边形面积

```
int sgn(double x)// 符号函数
{
    if(fabs(x)<eps)
        return 0;
    return x<0?-1:1;
}
struct Point
{
    double x,y;
    Point (double x=0.0,double y=0.0):x(x),y(y) {}
    void read()
    {
```


```
        scanf("%lf %lf",&x,&y);
    }
};//点集
typedef Point Vector;
struct Line
{
    Point p;
    Vector v;
    Line(Point p=Point(0,0),Vector v=0):p(p),v(v) {}
    Point point(double t)
    {
        return Point(p.x+v.x*t,p.y+v.y*t);
    }
};
//向量+向量=向量，向量+点=点
Vector operator + (Vector A,Vector B)
{
    return Vector(A.x+B.x,A.y+B.y);
}
//点-点=向量
Vector operator - (Point A,Point B)
{
    return Vector(A.x-B.x,A.y-B.y);
}
//向量*数=向量
Vector operator * (Vector A,double p)
{
    return Vector(A.x*p,A.y*p);
}
//向量/数=向量
Vector operator / (Vector A,double p)
{
    return Vector(A.x/p,A.y/p);
}
bool operator < (const Point &a,const Point &b)
{
    return a.x<b.x || (a.x==b.x&&a.y<b.y);
}
```


```
bool operator == (const Point &a,const Point &b)
{
    return sgn(a.x-b.x)==0&&sgn(a.y-b.y)==0;
}
//运算部分
double Dot(Vector A,Vector B)
{
    return A.x*B.x+A.y*B.y;
}
double Length(Vector A)
{
    return sqrt(Dot(A,A));
}
double Angle(Vector A,Vector B)
{
    return acos(Dot(A,B)/Length(A)/Length(B));
}
double Cross(Vector A,Vector B)
{
    return A.x*B.y-A.y*B.x;
}
double Area2(Point A,Point B,Point C)
{
    return Cross(B-A,C-A);
}
//向量逆时针旋转
Vector Rotate(Vector A,double rad)
{
    return Vector(A.x*cos(rad)-A.y*sin(rad),A.x*sin(rad)+A.y*cos(rad));
}
Vector Normal(Vector A) //单位法线
{
    double L=Length(A);
    return Vector(-A.y/L,A.x/L);
}
//直线相交  line1 P+tv  line2 Q+tw
Point GetLineIntersection(Point P,Point Q,Vector v,Vector w)
{
```


```
    Vector u=P-Q;
    double t=Cross(w,u) / Cross(v,w);
    return P+v*t;
}
```

//点到直线距离

```
double Dis_Point_Line(Point P,Point A,Point B)
{
    Vector v1=B-A,v2=P-A;
    return fabs(Cross(v1,v2)/Length(v1));
}
double Dis_Point_Line(Point P,Line L)
{
    Vector v1=L.v,v2=P-L.p;
    return fabs(Cross(v1,v2)/Length(v1));
}
double DistanceToSegment(Point p,Point a,Point b)
{
    if(a==b)
        return Length(p-a);
    Vector v1=b-a,v2=p-a,v3=p-b;
    if(sgn(Dot(v1,v2))<0) return Length(v2);
    if(sgn(Dot(v1,v3))>0) return Length(v3);
    return fabs(Cross(v1,v2)) / Length(v1);
}
```

//点在直线上的投影

```
Point GetLineProjection(Point P,Point A,Point B)
{
    Vector v=B-A;
    return A+v*(Dot(v,P-A) / Dot(v,v) );
}
```

//线段规范相交

```
bool SegmentProperIntersection(Point a1,Point a2,Point b1,Point b2)
{
    double c1=Cross(a2-a1,b1-a1),c2=Cross(a2-a1,b2-a1);
    double c3=Cross(b2-b1,a1-b1),c4=Cross(b2-b1,a2-b1);
    return sgn(c1)*sgn(c2)<0&&sgn(c3)*sgn(c4)<0;
}
```


```
//点是否在线段上
bool OnSegment(Point p,Point a1,Point a2)
{
    return sgn(Cross(a1-p,a2-p))==0 && sgn(Dot(a1-p,a2-p))<0;
}

//多边形有向面积
double PolygonArea(Point *p,int n)
{
    double area=0.0;
    for(int i=1; i<n-1; i++)
        area+=Cross(p[i]-p[0],p[i+1]-p[0]);
    return area/2;
}
```

2、圆和球相关

```
struct Circle
{
    Point c;
    double r;
    Circle(Point c=Point(0,0),double r=0): c(c),r(r) {}
    Point point(double a)
    {
        return Point(c.x+cos(a)*r,c.y+sin(a)*r);
    }
    void read()
    {
        scanf("%lf %lf %lf",&c.x,&c.y,&r);
    }
};
double angle(Vector v)
{
    return atan2(v.y,v.x);
}
//圆和直线相交
int getLineCircleIntersection(Line L,Circle C,vector<Point>& sol)
{
```


```
    double a=L.v.x,b=L.p.x-C.c.x,c=L.v.y,d=L.p.y-C.c.y;
    double e=a*a+c*c,f=2.0*(a*b+c*d),g=b*b+d*d-C.r*C.r;
    double delta=f*f-4.0*e*g;
    double t1,t2;
    if(sgn(delta)<0) return 0;
    if(sgn(delta)==0)
    {
        t1=t2=-f/(2*e);
        sol.push_back(L.point(t1));
        return 1;
    }
    t1=(-f-sqrt(delta))/(2.0*e),t2=(-f+sqrt(delta))/(2.0*e);
    sol.push_back(L.point(t1)),sol.push_back(L.point(t2));
    return 2;
}
```

//圆和圆相交

```
int getCircleCircleIntersection(Circle C1,Circle C2,vector<Point>& sol)
{
    double d=Length(C1.c-C2.c);
    if(sgn(d)==0)
    {
        if(sgn(C1.r-C2.r)==0)
            return -1; //重合
        return 0;
    }
    if(sgn(C1.r+C2.r-d)<0)
        return 0;
    if(sgn(fabs(C1.r-C2.r)-d )>0)
        return 0;

    double a=angle(C2.c-C1.c);
    double da=acos( (C1.r*C1.r+d*d-C2.r*C2.r)/ (2 * C1.r*d) );
    Point p1=C1.point(a-da),p2=C1.point(a+da);
    sol.push_back(p1);
    if(p1==p2)
        return 1;
    sol.push_back(p2);
    return 2;
```


```
}
//过点 p 到圆 c 的切线，v[i]是第 i 条切线的向量
int getTangents(Point p,Circle C, Vector* v)
{
    Vector u=C.c-p;
    double dist=Length(u);
    if(dist<C.r) return 0;
    else if(sgn(dist-C.r)==0)
    {
        v[0]=Rotate(u,pi/2);
        return 1;
    }
    else
    {
        double ang=asin(C.r/dist);
        v[0]=Rotate(u,ang);
        v[1]=Rotate(u,-ang);
        return 2;
    }
}
//圆的公切线 返回切条条数 返回-1 表示无数条切线
//a[i] 和 b[i]分别是第条切线在圆 A,圆 B 上的切点
int getTangents(Circle A,Circle B,Point *a, Point *b)
{
    int cnt=0;
    if(A.r<B.r)
    {
        swap(A,B);
        swap(a,b);
    }
    double d2=(A.c.x-B.c.x)*(A.c.x-B.c.x)+(A.c.y-B.c.y)*(A.c.y-B.c.y);
    double rdiff=A.r-B.r;
    double rsum=A.r+B.r;
    if(sgn(d2-rdiff*rdiff)<0)
        return 0;

    double base=atan2(B.c.y-A.c.y,B.c.x-A.c.x);
    if(sgn(d2)==0&&sgn(A.r-B.r)==0)
```


```
        return -1;
    if(sgn(d2-rdiff*rdiff)==0)
    {
        a[cnt]=A.point(base),b[cnt]=B.point(base),cnt++;
        return 1;
    }

    double ang=acos((A.r-B.r) / sqrt(d2));
    a[cnt]=A.point(base+ang);
    b[cnt]=B.point(base+ang);
    cnt++;
    a[cnt]=A.point(base-ang);
    b[cnt]=B.point(base-ang);
    cnt++;
    if(sgn(d2-rsum*rsum)==0)
    {
        a[cnt]=A.point(base);
        b[cnt]=B.point(base+pi);
        cnt++;
    }
    else if(sgn(d2-rsum*rsum)>0)
    {
        double ang=acos( (A.r+B.r)/sqrt(d2) );
        a[cnt]=A.point(base+ang);
        b[cnt]=B.point(base+ang+pi);
        cnt++;
        a[cnt]=A.point(base-ang);
        b[cnt]=B.point(base-ang+pi);
        cnt++;
    }
    return cnt;
}
```

3、凸包相关

//判断点是否在多边形内部

```
int isPointInPolygon(Point p,Point *Poly,int n)
{
```


```
    int wn=0;
    for(int i=0; i<n; i++)
    {
        if(OnSegment(p,Poly[i],Poly[(i+1)%n]))
            return -1;//在边界上
        int k=sgn(Cross(Poly[(i+1)%n]-Poly[i],p-Poly[i]));
        int d1=sgn(Poly[i].y-p.y);
        int d2=sgn(Poly[(i+1)%n].y-p.y);
        if(k>0&&d1<=0&&d2>0) wn++;
        if(k<0&&d2<=0&&d1>0) wn--;
    }
    if(wn!=0) return 1;
    return 0;
}
//二维凸包扫描
int ConvexHull(Point *p,int n,Point *Poly)
{
    sort(p,p+n);
    int m=0;
    for(int i=0; i<n; i++)
    {
        while(m>1 && Cross(Poly[m-1]-Poly[m-2],p[i]-Poly[m-2] ) <= 0 ) m--;
        Poly[m++]=p[i];
    }
    int k=m;
    for(int i=n-2; i>=0; i--)
    {
        while(m>k &&  Cross(Poly[m-1]-Poly[m-2],p[i]-Poly[m-2]) <=0) m--;
        Poly[m++]=p[i];
    }
    if(n > 1) m--;
    return m;
}
```

4、半平面交

```
//判断点在有向直线左侧
bool OnLeft(Line L,Point p)
```


```
{
    return Cross(L.v,p-L.p) > 0;
}
//半平面交
int HalfplaneIntersection(Line * L, int n, Point * Poly)
{
    sort(L,L+n);
    int first,last;
    Point *p=new Point[n];
    Line *q=new Line[n];
    q[first=last=0]=L[0];
    for(int i=1; i<n ; i++)
    {
        while(first<last &&  !OnLeft(L[i], p[last-1])) last--;
        while(first<last &&  !OnLeft(L[i], p[first])) first++;
        q[++last]=L[i];
        if(fabs(Cross(q[last].v, q[last-1].v) ) <eps)
        {
            last--;
            if(OnLeft(q[last],L[i].p)) q[last]=L[i];
        }
        if(first<last) p[last-1]=GetLineIntersection(q[last-1].p,q[last].p,
                                    q[last-1].v,q[last].v);
    }
    while(first<last &&  !OnLeft(q[first],p[last-1])) last--;
    if(last-first <=1) return 0;
    p[last] =GetLineIntersection(q[last].p,q[first].p,
                                 q[last].v,q[first].v);
    int m=0;
    for(int i=first; i<=last; i++) Poly[m++]=p[i];
    return m;
}
```

5、旋转卡壳

旋转卡壳算法原理

如下是一个更直观的算法：

1.计算多边形 y 方向上的端点。 我们称之为 ymin 和 ymax 。

2.通过 ymin 和 ymax 构造两条水平切线。 由于他们已经是一对对踵点， 计算他们之间的距离并维护为一个当前最大值。

3.同时旋转两条线直到其中一条与多边形的一条边重合。

4.一个新的对踵点对此时产生。 计算新的距离， 并和当前最大值比较， 大于当前最大值则更新。

5.重复步骤 3 和步骤 4 的过程直到再次产生对踵点对 (ymin,ymax) 。

6.输出确定最大直径的对踵点对。

//旋转卡壳 求直径

```
double RotateStuck(int m,Point * Poly)
{
    int i=0,j=0;
    double angi=0.0,angj=pi;
    for(int k=0; k<m; k++)
    {
        if(Poly[k].y<Poly[i].y)
            i=k;
        if(Poly[k].y>Poly[j].y)
            j=k;
    }
    double m_dis=Length(Poly[i]-Poly[j]);
    int cnt=0;
    while(sgn(angi-pi)<=0)
    {

        double dangi=Angle(Poly[(i+1)%m]-Poly[i])-angi;
        double dangj=Angle(Poly[(j+1)%m]-Poly[j])-angj;
        if(dangi<0)
            dangi+=2.0*pi;
        if(dangj<0)
            dangj+=2.0*pi;
        if(sgn(dangi-dangj)==0)
        {
            m_dis=max(m_dis,Length(Poly[(i+1)%m]-Poly[(j+1)%m]));
            m_dis=max(m_dis,Length(Poly[(i+1)%m]-Poly[j]));
            m_dis=max(m_dis,Length(Poly[i]-Poly[(j+1)%m]));
            i++,j++;
            i%=m,j%=m;
```


```
            angi+=dangi;
            angj+=dangi;
        }
        else if(sgn(dangi-dangj)<0)
        {
            i++;
            i%=m;
            m_dis=max(m_dis,Length(Poly[i]-Poly[j]));
            angi+=dangi;
            angj+=dangi;
        }
        else if(sgn(dangi-dangj)>0)
        {
            j++;
            j%=m;
            m_dis=max(m_dis,Length(Poly[i]-Poly[j]));
            angi+=dangj;
            angj+=dangj;
        }
    }
    return m_dis;
}
```

6、3 维几何

```
struct Point3
{
    double x,y,z;
    Point3(double x=0,double y=0,double z=0):x(x),y(y),z(z) {}
};
typedef Point3 Vector3;
Vector3 operator + (Vector3 A,Vector3 B)
{
    return Vector3(A.x+B.x,A.y+B.y,A.z+B.z);
}
Vector3 operator - (Vector3 A,Vector3 B)
{
    return Vector3(A.x-B.x,A.y-B.y,A.z-B.z);
```


```
}
Vector3 operator * (Vector3 A,double k)
{
    return Vector3(A.x*k,A.y*k,A.z*k);
}
Vector3 operator / (Vector3 A,double k)
{
    return Vector3(A.x/k,A.y/k,A.z/k);
}
bool operator < (const Point3 &a,const Point3 &b)
{
    return a.x<b.x ||(a.x==b.x&&a.y<b.y) ||( a.x==b.x&&a.y==b.y&&a.z<b.z );
}
bool operator == (const Point3 &a,const Point3 &b)
{
    return sgn(a.x-b.x)==0 && sgn(a.y-b.y)==0  && sgn(a.z-b.z)==0;
}
double Dot(Vector3 A,Vector3 B)
{
    return A.x*B.x+A.y*B.y+A.z*B.z;
}
double Length(Vector3 A)
{
    return sqrt(Dot(A,A));
}
double Angle(Vector3 A,Vector3 B)
{
    return acos( Dot(A,B)/Length(A)/Length(B) );
}
double DistanceToPlane(Point3 p,Point3 p0,Vector3 n)
{
    return fabs(Dot(p-p0,n)) / Length(n);
}
Point3 GetPlaneProjection(Point3 p,Point3 p0,Vector3 n)
{
    n=n/Length(n);
    double d=Dot(p-p0,n)/Length(n);
    return p-n*d;
```


```
}
Point3 LinePlaneIntersection(Point3 p1,Point3 p2,Point3 p0,Vector3 n)
{
    Vector3 v=p2-p1;
    if( sgn(Dot(n,p2-p1)) ==0)
        return Point3(1e10,1e10);
    double t=(Dot(n,p0-p1)/ Dot(n,p2-p1) );
    return p1+v*t;
}
Vector3 Cross(Vector3 A,Vector3 B)
{
    return Vector3(A.y*B.z-A.z*B.y,A.z*B.x-A.x*B.z,A.x*B.y-A.y*B.x);
}
double Area2(Point3 A,Point3 B,Point3 C)
{
    return Length(Cross(B-A,C-A));
}
bool PointInTri(Point3 p,Point3 p0,Point3 p1,Point3 p2)
{
    double area1=Area2(p,p0,p1);
    double area2=Area2(p,p1,p2);
    double area3=Area2(p,p2,p0);
    if(sgn(area1+area2+area3-Area2(p0,p1,p2) ) ==0)
        return 1;
    return 0;
}
double DistanceToLine(Point3 p,Point3 A,Point3 B)
{
    Vector3 v1=B-A,v2=p-A;
    return Length(Cross(v1,v2))/Length(v1);
}
double DistanceToSegment(Point3 p,Point3 A,Point3 B)
{
    if(A==B)
        return Length(p-A);
    Vector3 v1=B-A,v2=p-A,v3=p-B;
    if(sgn(Dot(v1,v2)) < 0) return Length(v2);
    else if(sgn(Dot(v1,v3) ) > 0) return Length(v3);
```


```
    else return Length(Cross(v1,v2))/Length(v1);
}
double Volume(Point3 A,Point3 B,Point3 C,Point3 D)
{
    return Dot(D-A,Cross(B-A,C-A));
}
```

//3 维凸包

```
struct Face //3维平面
{
    int v[3];
    Vector3 normal(Point3 *p) const
    {
        return Cross(p[v[1]]-p[v[0]],p[v[2]]-p[v[0]]);
    }
    int cansee(Point3 *p,int i) const
    {
        return Dot(p[i]-p[v[0]],normal(p)) >0?1:0;
    }
};
```

//没有考虑各种特殊情况（如 4 点共面），实际中对输入点进行微小扰动

```
vector<Face> Ch3D(Point3 *p,int n)
{
    vector<Face> cur;
    cur.PB((Face)
    {
        0,1,2
    });
    cur.PB((Face)
    {
        2,1,0
    });
    for(int i=3; i<n; i++)
    {
        vector<Face> next;
        bool vis[4][4];
        memset(vis,0,sizeof vis);
        for(int j=0; j<cur.size(); j++)
```


```
        {
            Face& f=cur[j];
            int res=f.cansee(p,i);
            if(!res)
                next.PB(f);
            for(int k=0; k<3; k++)
                vis[f.v[k] ][f.v[(k+1)%3] ]=res;
        }
        for(int j=0; j<cur.size(); j++)
            for(int k=0; k<3; k++)
            {
                int a=cur[j].v[k],b=cur[j].v[(k+1)%3];
                if(vis[a][b]!=vis[b][a] && vis[a][b])
                    next.PB((Face)
                {
                    a,b,i
                });
            }
        cur=next;
    }
    return cur;
}
```

7、最小矩形覆盖

```
double Rote(int n)
{
    double minsqr=1e12;
    int left=0,right=0,up=0;
    for(int i=0; i<n; i++)
    {
        while(sgn( Cross(Poly[i]-Poly[up+1],Poly[i+1]-Poly[up+1]) -
Cross(Poly[i]-Poly[up],Poly[i+1]-Poly[up]) ) >=0 ) up++,up%=n;
        while(sgn(Dot(Poly[i+1]-Poly[right+1],Poly[i]-Poly[i+1]) -
Dot(Poly[i+1]-Poly[right],Poly[i]-Poly[i+1])) >=0 ) right++,right%=n;
        if(i==0) left=right;
        while(sgn(Dot(Poly[i]-Poly[left+1],Poly[i+1]-Poly[i]) - Dot(Poly[i]-
Poly[left],Poly[i+1]-Poly[i])) >=0 ) left++,left%=n;
```


```
        double L=Length(Poly[i]-Poly[i+1]);
        double H=Cross(Poly[i]-Poly[up],Poly[i+1]-Poly[up])/L;
        double bottom = Dot(Poly[i]-Poly[left],Poly[i+1]-Poly[i])/L +
Dot(Poly[i+1]-Poly[right],Poly[i]-Poly[i+1])/L+L;
        double newsqr=bottom*H;

        if(newsqr<minsqr)
        {
            minsqr=newsqr;
            ans[0]=Poly[i]+ (Poly[i]-Poly[i+1])*Dot(Poly[i]-
Poly[left],Poly[i+1]-Poly[i])/L/L;
            Vector A=(Poly[i+1]-Poly[i]);
            A=Normal(A);
            ans[1]=ans[0]+A*bottom;
            A=Rotate(A,pi/2);
            ans[2]=ans[1]+A*H;
            A=Rotate(A,pi/2);
            ans[3]=ans[2]+A*bottom;
        }
    }
    return minsqr;
}
```


五、其他部分

1、输入输出外挂

//用于整数的输入和输出

```
inline int read()
{
    char ch;
    bool flag=false;
    int re=0;
    while(!(((ch=getchar())>='0'&&(ch<='9'))||(ch=='-')));
    if(ch!='-')
    {
        re*=10;
        re+=ch-'0';
    }
    else
        flag=true;
    while((ch=getchar())>='0'&&(ch<='9'))
    {
        re*=10;
        re+=ch-'0';
    }
    if(flag)
        re=-re;
    return re;
}
inline void write(int x)
{
    if(x<0)
    {
        putchar('-');
        x=-x;
```


```
    }
    if(x>=10)
    {
        write(x/10);
    }
    putchar(x%10+'0');
}
```

2、高精度

重载了加，减，乘。

构造函数：Bignum()，Bignum(int sz,char *a),Bignum(int sz,int *a),Bignum(int x)

```
const int mmax = 510;
const int inf = 0x3fffffff;
struct Bignum
{
    int Sz;
    int num[mmax];
    void print()
    {
        for(int i=Sz-1; i>=0; i--)
            printf("%d",num[i]);
        puts("");
    }
    Bignum()
    {
        memset(num,0,sizeof num);
    }
    Bignum(int sz,char *a)
    {
        memset(num,0,sizeof num);
        for(int i=0; i<sz; i++)
        {
            num[i]=a[sz-1-i]-'0';
        }
        Sz=1;
```


```
        for(int i=sz-1; i>=0; i--)
        {
            if(num[i])
            {
                Sz=i+1;
                break;
            }
        }
    }
    Bignum(int sz,int *a)
    {
        memset(num,0,sizeof num);
        for(int i=0; i<sz; i++)
            num[i]=a[i];
        Sz=sz;
    }
    Bignum(int x)
    {
        memset(num,0,sizeof num);
        if(x==0)
            Sz=1;
        Sz=0;
        while(x)
        {
            num[Sz++]=x%10;
            x/=10;
        }
    }
    Bignum operator + (const Bignum &a)
    {
        int tmp[mmax];
        memset(tmp,0,sizeof tmp);
        int len=max(Sz,a.Sz);
        for(int i=0; i<len; i++)
        {
            tmp[i]+=num[i]+a.num[i];
            tmp[i+1]+=tmp[i]/10;
            tmp[i]%=10;
```


```
    }
    return Bignum(len+(tmp[len]?1:0),tmp);
}
Bignum operator - (const Bignum &a)
{
    int tmp[mmax];
    memset(tmp,0,sizeof tmp);
    int len=max(Sz,a.Sz);
    for(int i=0; i<len; i++)
    {
        tmp[i]+=num[i]-a.num[i];
        if(tmp[i]<0)
        {
            tmp[i+1]-=1;
            tmp[i]+=10;
        }
    }
    for(int i=len-1; i>=0; i--)
    {
        if(tmp[i])
            return Bignum(i+1,tmp);
    }
    return Bignum(1,tmp);
}
Bignum operator * (const Bignum &a)
{
    int tmp[mmax];
    memset(tmp,0,sizeof tmp);
    for(int i=0; i<Sz; i++)
        for(int j=0; j<a.Sz; j++)
        {
            tmp[i+j]+=num[i]*a.num[j];
        }
    for(int i=0; i<Sz+a.Sz; i++)
    {
        tmp[i+1]+=tmp[i]/10;
        tmp[i]%=10;
    }
```


```
        for(int i=Sz+a.Sz-1; i>=0; i--)
        {
            if(tmp[i])
                return Bignum(i+1,tmp);
        }
        return Bignum(1,tmp);
    }
    bool operator <(const Bignum &a) const
    {
        for(int i=mmax-1; i>=0; i--)
        {
            if(num[i]>a.num[i])
                return 0;
            if(num[i]<a.num[i])
                return 1;
        }
        return 0;
    }
    bool operator ==(const Bignum &a) const
    {
        for(int i=mmax-1; i>=0; i--)
        {
            if(num[i]!=a.num[i])
                return 0;
        }
        return 1;
    }
};
```

3、精确覆盖（DLX）

3.1 不可重复覆盖

```
#include <iostream>
#include<stdio.h>
#include<cmath>
#include<string.h>
#include<algorithm>
```


```
#include<string>
#include<vector>
using namespace std;
const int maxn = 2010;
const int maxnode = 20010;
const int maxr = 5010;
struct DLX
{
    int n,m,sz; // 行数，列数，节点总数
    int row[maxnode],col[maxnode];   //各节点行列标号
    int L[maxnode],R[maxnode],U[maxnode],D[maxnode];  // 十字链表 记录节点ID

    int ansd,ans[maxr];   //解
    int H[maxn],S[maxn];
    void init(int n,int m)
    {
        this->n=n;
        this->m=m;
        for(int i = 0; i <= m; i++)
        {
            S[i] = 0;
            U[i] = D[i] = i;
            L[i] = i-1;
            R[i] = i+1;
        }
        R[m] = 0;
        L[0] = m;
        sz =m;
        for(int i = 1; i <= n; i++)H[i] = -1;
    }
    void Link(int r,int c)
    {
        ++S[col[++sz]=c];
        row[sz] = r;
        U[sz] = U[c];
        D[U[c]] = sz;
        D[sz] = c;
        U[c] = sz;
```


```
        if(H[r] < 0)H[r] = L[sz] = R[sz] = sz;
        else
        {
            L[sz] = L[H[r]];
            R[L[H[r]]] = sz;
            R[sz] = H[r];
            L[H[r]] = sz;
        }
    }
#define FOR(i,A,s) for(int i=A[s];i!=s;i=A[i])
    void remove(int c)
    {
        L[R[c]]=L[c];
        R[L[c]]=R[c];
        FOR(i,D,c)
        FOR(j,R,i)
        {
            U[D[j]]=U[j];
            D[U[j]]=D[j];
            --S[col[j]];
        }
    }
    void restore(int c)
    {
        FOR(i,U,c)
        FOR(j,L,i)
        {
            ++S[col[j]];
            U[D[j]]=j;
            D[U[j]]=j;
        }
        L[R[c]]=c;
        R[L[c]]=c;
    }

    bool dfs(int d)
    {
        if(R[0]==0)
```


```
        {
            ansd=d;
            return  1;
        }
        //找到S最小的列c
        int c=R[0];                 //第一个为删除的列
        FOR(i,R,0)
        if(S[i]<S[c])
            c=i;

        remove(c);
        FOR(i,D,c)
        {
            ans[d]=row[i];
            FOR(j,R,i) remove(col[j]);
            if(dfs(d+1))  return 1;
            FOR(j,L,i) restore(col[j]);
        }
        restore(c);
        return 0;
    }
    bool solve()
    {
        return dfs(0);
    }
};

int main()
{
    int n,m;
    while(~scanf("%d %d",&n,&m))
    {
        DLX X;
        X.init(n,m);
        for(int i=1; i<=n; i++)
        {
            for(int j=1; j<=m; j++)
            {
```


```
                int x;
                scanf("%d",&x);
                if(x)
                    X.Link(i,j);
            }
        }
        if(X.solve())
            puts("Yes, I found it");
        else
            puts("It is impossible");
    }
    return 0;
}
```

3.2 可以重复覆盖

```
const int inf = 0x3fffffff;
const int maxn = 15*15+10;
const int maxnode = maxn*maxn;
struct DLX
{
    int n,m,sz;
    int U[maxnode],D[maxnode],R[maxnode],L[maxnode];
    int row[maxnode],col[maxnode];
    int H[maxn],S[maxn];
    int ans;
    void init(int n,int m)
    {
        this->n=n;
        this->m=m;
        for(int i = 0; i <= m; i++)
        {
            S[i] = 0;
            U[i] = D[i] = i;
            L[i] = i-1;
            R[i] = i+1;
        }
        R[m] = 0;
```


```
    L[0] = m;
    sz =m;
    for(int i = 1; i <= n; i++)H[i] = -1;
}
void Link(int r,int c)
{
    ++S[col[++sz]=c];
    row[sz] = r;
    U[sz] = U[c];
    D[U[c]] = sz;
    D[sz] = c;
    U[c] = sz;
    if(H[r] < 0)H[r] = L[sz] = R[sz] = sz;
    else
    {
        L[sz] = L[H[r]];
        R[L[H[r]]] = sz;
        R[sz] = H[r];
        L[H[r]] = sz;
    }
}
#define FOR(i,A,s) for(int i=A[s];i!=s;i=A[i])
void remove(int c)
{
    FOR(i,D,c)
    {
        L[R[i]]=L[i];
        R[L[i]]=R[i];
    }
}
void restore(int c)
{
    FOR(i,U,c)
    {
        L[R[i]]=i;
        R[L[i]]=i;
    }
}
```


```
bool v[maxnode];
int f()
{
    int ret = 0;
    for(int c = R[0]; c != 0; c = R[c]) v[c] = true;
    for(int c = R[0]; c != 0; c = R[c])
        if(v[c])
        {
            ret++;
            v[c] = false;
            for(int i = D[c]; i != c; i = D[i])
                for(int j = R[i]; j != i; j = R[j])
                    v[col[j]] = false;
        }
    return ret;
}
void dfs(int d)
{
    if(d+f()>=ans)
        return ;
    if(R[0]==0)
    {
        ans=min(ans,d);          //找到了解
        return ;
    }
    //找到S最小的列c
    int c=R[0];                  //第一个为删除的列
    FOR(i,R,0)
    if(S[i]<S[c])
        c=i;

    FOR(i,D,c)
    {
        remove(i);
        FOR(j,R,i) remove(j);
        dfs(d+1);
        FOR(j,L,i) restore(j);
        restore(i);
```


```
        }
        return;
    }
    int solve()
    {
        ans=inf;
        dfs(0);
        return ans;
    }
};
```

4、线扫描

4.1 矩形面积并

```
#include <iostream>
#include<string>
#include<cstdio>
#include<cmath>
#include<algorithm>
using namespace std;
const int mmax = 210;
const double eps = 1e-8;
int sgn(double x)
{
    if(fabs(x)<eps)
        return 0;
    return x<0?-1:1;
}
struct Rect
{
    double x1,x2;
    double y1,y2;
    void read()
    {
        scanf("%lf %lf %lf %lf",&x1,&y1,&x2,&y2);
        if(x1>x2)
            swap(x1,x2);
```


```
        if(y1>y2)
            swap(y1,y2);
    }
} R[110];
struct node
{
    int l,r;
    int cov;
    double len;
    int mid()
    {
        return (l+r)>>1;
    }
} T[4*mmax];
struct Num
{
    double x;
    int id;
    bool operator < (const Num &a) const
    {
        return x<a.x;
    }
} X[mmax];
struct Line
{
    int l,r;
    double y;
    int fg;
    bool operator < (const Line &a) const
    {
        return y<a.y;
    }
} L[mmax];
void build(int id,int l,int r)
{
    T[id].l=l,T[id].r=r;
    T[id].cov=0,T[id].len=0;
    if(l==r)
```


```
        return ;
    int mid=T[id].mid();
    build(id<<1,l,mid);
    build(id<<1|1,mid+1,r);
}
double pos[mmax];
void updata(int id,int l,int r,int fg)
{
    if(l<=T[id].l && T[id].r<=r)
    {
        T[id].cov+=fg;
        if(T[id].cov)
            T[id].len=pos[T[id].r+1]-pos[T[id].l];
        else
        {
            if(T[id].l==T[id].r)
                T[id].len=0;
            else
                T[id].len=T[id<<1].len+T[id<<1|1].len;
        }
        return ;
    }
    int mid=T[id].mid();
    if(mid>=l)
        updata(id<<1,l,r,fg);
    if(mid<r)
        updata(id<<1|1,l,r,fg);
    if(T[id].cov==0)
    {
        T[id].len=T[id<<1].len+T[id<<1|1].len;
    }
}
int id[mmax];
int main()
{
    int n,ca=0;
    while(~scanf("%d",&n)&&n)
    {
```


```
int cnt=0;
for(int i=0; i<n; i++)
{
    R[i].read();
    X[cnt].x=R[i].x1;
    X[cnt].id=cnt;
    cnt++;
    X[cnt].x=R[i].x2;
    X[cnt].id=cnt;
    cnt++;
}

sort(X,X+cnt);
id[X[0].id]=1;
pos[1]=X[0].x;
for(int i=1; i<cnt; i++)
{
    if(sgn(X[i].x-X[i-1].x)==0)
        id[X[i].id]=id[X[i-1].id];
    else
        id[X[i].id]=id[X[i-1].id]+1;
    pos[id[X[i].id]]=X[i].x;
}
for(int i=0; i<n; i++)
{
    L[i].l=id[2*i];
    L[i].r=id[2*i+1];
    L[i].y=R[i].y1;
    L[i].fg=1;
    L[i+n].l=id[2*i];
    L[i+n].r=id[2*i+1];
    L[i+n].y=R[i].y2;
    L[i+n].fg=-1;
}
sort(L,L+2*n);
build(1,1,id[X[cnt-1].id]-1);
updata(1,L[0].l,L[0].r-1,L[0].fg);
double ans=0.0;
```


```
        for(int i=1; i<2*n; i++)
        {
            ans+=(L[i].y-L[i-1].y)*T[1].len;
            updata(1,L[i].l,L[i].r-1,L[i].fg);
        }
        printf("Test case #%d\nTotal explored area: %.2lf\n\n",++ca,ans);
    }
    return 0;
}
```

4.2 矩形周长并

```
#include <iostream>
#include<string>
#include<cstdio>
#include<cmath>
#include<algorithm>
using namespace std;
const int mmax = 20000;
const double eps = 1e-8;
int sgn(int x)
{
    return x<0?-1:1;
}
struct Rect
{
    int x1,x2;
    int y1,y2;
    void read()
    {
        scanf("%d %d %d %d",&x1,&y1,&x2,&y2);
        if(x1>x2)
            swap(x1,x2);
        if(y1>y2)
            swap(y1,y2);
    }
} R[5010];
struct node
```


```
{
    int l,r;
    int cov;
    int cnt;
    int len;
    bool lcov,rcov;
    int mid()
    {
        return (l+r)>>1;
    }
} T[4*mmax];
struct Num
{
    int x;
    int id;
    bool operator < (const Num &a) const
    {
        return x<a.x;
    }
} X[mmax];
struct Line
{
    int l,r;
    int y;
    int fg;
    bool operator < (const Line &a) const
    {
        return y<a.y;
    }
} L[mmax];
void build(int id,int l,int r)
{
    T[id].l=l,T[id].r=r;
    T[id].cov=T[id].len=0;
    T[id].lcov=T[id].rcov=0;
    T[id].cnt=0;
    if(l==r)
        return ;
```


```
    int mid=T[id].mid();
    build(id<<1,l,mid);
    build(id<<1|1,mid+1,r);
}
double pos[mmax];
void updata(int id,int l,int r,int fg)
{
    if(l<=T[id].l && T[id].r<=r)
    {
        T[id].cov+=fg;
        if(T[id].cov)
        {
            T[id].len=pos[T[id].r+1]-pos[T[id].l];
            T[id].cnt=2;
            T[id].lcov=T[id].rcov=1;
        }
        else
        {
            if(T[id].l==T[id].r)
            {
                T[id].cnt=T[id].len=0;
                T[id].lcov=T[id].rcov=0;
            }
            else
            {
                T[id].len=T[id<<1].len+T[id<<1|1].len;
                T[id].cnt=T[id<<1].cnt+T[id<<1|1].cnt;
                T[id].lcov=T[id<<1].lcov;
                T[id].rcov=T[id<<1|1].rcov;
                if(T[id<<1].rcov && T[id<<1|1].lcov)
                    T[id].cnt-=2;
            }
        }
        return ;
    }
    int mid=T[id].mid();
    if(mid>=l)
        updata(id<<1,l,r,fg);
```


```
    if(mid<r)
        updata(id<<1|1,l,r,fg);
    if(T[id].cov==0)
    {
        T[id].len=T[id<<1].len+T[id<<1|1].len;
        T[id].cnt=T[id<<1].cnt+T[id<<1|1].cnt;
        T[id].lcov=T[id<<1].lcov;
        T[id].rcov=T[id<<1|1].rcov;
        if(T[id<<1].rcov && T[id<<1|1].lcov)
            T[id].cnt-=2;
    }
}
int id[mmax];
int main()
{
    int n;
    while(~scanf("%d",&n))
    {
        int cnt=0;
        for(int i=0; i<n; i++)
        {
            R[i].read();
            X[cnt].x=R[i].x1;
            X[cnt].id=cnt;
            cnt++;
            X[cnt].x=R[i].x2;
            X[cnt].id=cnt;
            cnt++;
        }
        sort(X,X+cnt);
        id[X[0].id]=1;
        pos[1]=X[0].x;
        for(int i=1; i<cnt; i++)
        {
            if(sgn(X[i].x-X[i-1].x)==0)
                id[X[i].id]=id[X[i-1].id];
            else
                id[X[i].id]=id[X[i-1].id]+1;
```


```
            pos[id[X[i].id]]=X[i].x;
        }
        for(int i=0; i<n; i++)
        {
            L[i].l=id[2*i];
            L[i].r=id[2*i+1];
            L[i].y=R[i].y1;
            L[i].fg=1;
            L[i+n].l=id[2*i];
            L[i+n].r=id[2*i+1];
            L[i+n].y=R[i].y2;
            L[i+n].fg=-1;
        }
        sort(L,L+2*n);
        build(1,1,id[X[cnt-1].id]-1);
        updata(1,L[0].l,L[0].r-1,L[0].fg);
        int ans=T[1].len,last=T[1].len;
        for(int i=1; i<2*n; i++)
        {
            ans+=T[1].cnt*(L[i].y-L[i-1].y);
            updata(1,L[i].l,L[i].r-1,L[i].fg);
            ans+=abs(T[1].len-last);
            last=T[1].len;
        }
        printf("%d\n",ans);
    }
    return 0;
}
```