ACM 模板库

xiaoyu1_1

2014-10-16

BITSS

目录

数论 2
数论基本定理 2
斯特林公式 2
质数打表 2
模线性方程 3
扩展欧几里得 3
求 n！的最高位数字 3
筛法欧拉函数 4
容斥原理 4
求逆元 5
抛针问题 5
欧拉函数—单值 5
母函数 5
莫比乌斯反演 6
快速幂 9
矩阵快速幂 9
高斯消元—01 矩阵 9
高斯消元—小数 10
斐波那契额数列通式 11
Miller_Rabin 算法进行素数测试 11
gcd—lcm 13
a^b%c(b 为大整数) 13
组合数学 13
基本知识 13
组合 C（）模板 14
拓扑排序 14
第一类 Stirling 数 15
字符串 15
KMP 15
（Manacher）算法----O(n)回文子串 16
网络流 17
网络流 17
费用流 17
图论 18
最短路 18
最小生成树 19
数据结构 19
单调队列 19
树链剖分-点值 20
树链剖分-段值 22
树状数组模板 23
拓扑排序 24
经典题 25
nlog(n)最长上升子序列 25
TSP 25
技巧 26
C++高精度模板 26
java 大整数模板 27
一些函数 28
二分模板 28
三分模板 28
输入加速 29
随机生成数据 29
手动扩栈 29
计算出二进制数中有多少个 1 29
计算几何 29
求多边形重心 29
求最小面积外接矩阵和最小周长外接矩阵 30
凸包 31
动态规划 dp 32
背包 32
数位 dp 33
博弈 34
NIM 游戏 34
SG 函数 34
一些经典博弈问题 35
STL 用法 35
Bitset 35
String 36
优先队列 37

数论

数论基本定理

剩余类　　：%4，{0,4,8,12……}，{1,5,9,13……}，……
简化剩余类 ：{1,5,9,13……}，{3,7,11,15……}（欧拉函数==简化剩余类个数）
完全剩余系 ：剩余类 每个挑一个 的集合
简化剩余系 ：简化剩余类 每个挑一个 的集合

欧拉定理
欧拉定理：设 m 是正整数，且 gcd(a,m)=1，则 a ψ(m)≡1 mod m

推论：若 m=pq 且 p 和 q 为素数，a∈Z，如果 gcd(a,m)=p 或 q，则同样有 a ψ(m)+1≡a mod m。

Fermat（费尔马）小定理
设 m 是素数，a 是与 m 互素的整数，则 am-1≡1 mod m，从而对任意整数 a 有 am≡a mod m

（1）
1. 整数的唯一分解定理：
任意正整数都有且只有一种方式写出其素因子的乘积表达式。
A=(p1^k1)*(p2^k2)*(p3^k3)*....*(pn^kn)　其中 pi 均为素数
2. 约数和公式：
对于已经分解的整数 A=(p1^k1)*(p2^k2)*(p3^k3)*....*(pn^kn)
有 A 的所有因子之和为
S = (1+p1+p1^2+p1^3+...p1^k1) * (1+p2+p2^2+p2^3+….p2^k2) * (1+p3+ p3^3+…+ p3^k3) * .... * (1+pn+pn^2+pn^3+...pn^kn)

（2）欧拉函数是少于或等于 n 的数中与 n 互质的数的数目；
（3）小于 n 的数中与 n 互质的数的和
题意：
给一个 n，求
少于或等于 n 的数中与 n 不互质的数的和
我们先求
少于或等于 n 的数中与 n 互质的数的和
对于 i 与 n 互素
gcd(n,i)=1
必有 gcd(n,n-i)=1
设 n 的欧拉函数值为 f[n]
则有 f[n]个数与 n 互素，这些数两两相加必等于 n
于是有答案为 f[n]*n/2

（4）不知道什么定理
A^x = A^(x % Phi(C) + Phi(C)) (mod C)　（x>=Phi(C)）

斯特林公式

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n.$$

有斯特林公式

lgn!=log10(sqrt(2*PI*n*1.0))+n*log10(n*1.0/e)
算出 lgn!后，比如对于
n=5
lg5!=2.07918
由于整数部分与最高位无关
剪掉变成 0.07918
10^0.07918=1.19999
于是最高位就是 1
由于斯特林公式在 n 比较小的时候误差较大，所以在 n 较小时可以用 O（n）的方法算 lgn!

质数打表

```
#define N 50000   //质数范围
int                                          prime[1000000];
//prime[0]=2,prime[1]=3,prime[2]=5,……

void init_prime()
{
    int i, j;
    for(i = 2;i <= sqrt(N*1.0); ++i)
    {
        if(!prime[i])
            for(j = i * i; j < N; j += i)
                prime[j] = 1;
    }
    j = 0;
    for(i = 2;i <= N; ++i)
        if(!prime[i])
            prime[j++] = i;
}
```

模线性方程

模线性方程：ax + by = c （1）
给定整数 a,b,c,求解整数 x,y?
浓缩：ax + by = gcd(a,b)
如果 c 是 gcd(a,b)的倍数，则方程（1）有解。有解就意味着有无数解。否则，无解！

假设 x1,y1 是 a*x+b*y=gcd(a,b)的一个解
通解：
x = x1 + k * b/gcd(a,b)
y = y1 - k * a/gcd(a,b)
其中，k 为任意整数

新理解：
ax+by=1;
x1,y1 为一组解；
通解：x=x1 + k*b;
y=y1 - k*a;

扩展欧几里得

ax+by=c.
x,y 满足 ax+by=gcd(a,b);
x*=c/gcd(a,b);
y*=c/gcd(a,b);后
x,y 满足 ax+by=c;
通解：
x0=x+k*b/gcd(a,b);
y0=y-k*a/gcd(a,b);

代码：

```
ll x,y;
ll exgcd(ll a,ll b)
{
    if(b==0)
    {
        x=1,y=0;
        return a;
    }
    ll gcd=exgcd(b,a%b);
    ll t=x;
    x=y;
    y=t-a/b*y;
    return gcd;
}
ll mod(ll a,ll b)
{
    if(a>=0)
        return a%b;        //a%b,结果正负，只和 a 的正负有关
    else
        return a%b+abs(b); //b 可能为负数。
}
```

求 n！的最高位数字

题意，给一个 n(<10^7)
比如 3！=6 最高位就是 6
解法：
lgn!=lg1+lg2+....+lgn=temp
这里如果用 O（n）的累加算 lgn!就会因为样例数可能达到 1000 个

10^7*10^3>10^8 而超时

有斯特林公式
n!== sqrt(2*PI*n)*(n/e)^n;
lgn!=log10(sqrt(2*PI*n*1.0))+n*log10(n*1.0/e)
算出 lgn!后，比如对于
n=5
lg5!=2.07918
由于整数部分与最高位无关
剪掉变成 0.07918
10^0.07918=1.19999
于是最高位就是 1
由于斯特林公式在 n 比较小的时候误差较大，所以在 n 较小时可以用 O（n）的方法算 lgn!

```
#define pi acos(-1.0)
#define e exp(1.0)
int main()
{
    int t;
    scanf("%d",&t);
    while(t--)
    {
        int i,j,k;
        double temp=0,n;
        scanf("%lf",&n);
        if(n>10)
        {
            temp=log10(sqrt(2*pi*n))+n*log10(n/e);
        }
        else
        {
```

```
                for(i=1;i<=(int)n;i++)
                {
                        temp+=log10(i*1.0);
                }
        }
        temp-=(int)temp;
        double te=pow(10.0,temp);
        printf("%d\n",(int)te);
    }
}
```

筛法欧拉函数

φ函数的值　通式：φ(x)=x(1-1/p1)(1-1/p2)(1-1/p3)(1-1/p4)…..(1-1/pn),其中 p1, p2……pn 为 x 的所有质因数，x 是不为 0 的整数。φ(1)=1（唯一和 1 互质的数(小于等于 1)就是 1 本身）。 (注意：每种质因数只一个。比如 12=2*2*3 那么φ（12）=12*（1-1/2）*(1-1/3)=4

```
#define N 1000005
#define ll long long
int prime[N];
ll phi[N];
ll sum[N];

void dabiao()
{
    int i,j,k;
    memset(prime,0,sizeof(prime));
    for(i=2;i*i<N;i++)
        if(!prime[i])
            for(j=i*i;j<N;j+=i)
                prime[j]=1;
    for(i=1;i<N;i++)
        phi[i]=i;
    for(i=2;i<N;i++)
        if(!prime[i])
            for(j=i;j<N;j+=i)
                phi[j]=phi[j]/i*(i-1);
    sum[1]=1;
    for(i=2;i<N;i++)
        sum[i]=sum[i-1]+phi[i];
}
int main()
{
    dabiao();
    int i,j,k;
    int a,b;
    while(~scanf("%d%d",&a,&b))
        printf("%lld\n",sum[b]-sum[a-1]);
}
```

容斥原理

```
//求挑 4 个数互质的个数
#define ll long long
#define N 10010
int p[N],g[N];
int t[20];
ll h[N];
void solve(int x)
{
    int i,j,k,cnt=0;
    for(i=2;i*i<=x;i++)
    {
        if(x%i==0)
        {
            t[cnt++]=i;
            while(x%i==0)
                x/=i;
        }
    }
    if(x!=1)
        t[cnt++]=x;
    int tt=(1<<cnt);
    for(i=1;i<tt;i++)
    {
        int tsum=1,tflag=0;
        for(j=0;j<cnt;j++)
        {
            if((i>>j)&1)
            {
                tsum*=t[j];
                tflag++;
            }
        }
        p[tsum]++;
        g[tsum]=tflag;
    }
}
int main()
{
    int n;
    ll i,j,k;
    h[0]=h[1]=h[2]=h[3]=0;
```

```
    for(i=4;i<=10000;i++)
    {
        h[i]=i*(i-1)*(i-2)*(i-3)/24;
    }
    while(scanf("%d",&n)!=EOF)
    {
        memset(p,0,sizeof(p));
        memset(g,0,sizeof(g));
        for(i=0;i<n;i++)
        {
            scanf("%d",&k);
            solve(k);
        }
        ll res=h[n];
        for(i=1;i<=10000;i++)
        {
            if(p[i]!=0)
            {
                if(g[i]&1)
                    res-=h[p[i]];
                else
                    res+=h[p[i]];
            }
        }
        cout<<res<<endl;
    }
}
```

求逆元

```
//a 对 mod 的逆元
ll inv(ll a)
{
    return a == 1 ? 1 : (long long)(mod - mod / a) * inv(mod % a) % mod;
}
```

抛针问题

针长 L，平行直线距离 D
针与直线相交概率：2L/piD

欧拉函数—单值

```
//求 1..n-1 中与 n 互质的数的个数
int eular(int n){
    int ret=1,i;
    for (i=2;i*i<=n;i++)
        if (n%i==0){
            n/=i,ret*=i-1;
            while (n%i==0)
                n/=i,ret*=i;
        }
    if (n>1)
        ret*=n-1;
    return ret;
}
```

母函数

母函数：
要买由 M 个水果组成的水果拼盘，对于每种水果，个数上我有限制，既不能少于某个特定值，也不能大于某个特定值。能组出多少种不同的方案？

```
#include<iostream>
using namespace std;
int Min[110];
int Max[110];
int c1[110];
int c2[110];
int main()
{
    int n,m,i,k,j;

    while(scanf("%d%d",&n,&m)!=EOF)
    {
        for(i=0;i<n;i++)
            scanf("%d%d",&Min[i],&Max[i]);
        memset(c1,0,sizeof(c1));
        memset(c2,0,sizeof(c2));
        for(i=Min[0];i<=Max[0];i++)
            c1[i]=1;
        for(i=1;i<n;i++)
        {
            for(j=0;j<=m;j++)
                for(k=Min[i];k+j<=m&&k<=Max[i];k++)
                    c2[k+j]+=c1[j];
            for(j=0;j<=m;j++)
            {
                c1[j]=c2[j];
                c2[j]=0;
            }
        }

        printf("%d/n",c1[m]);
```

```
    }
    return 0;
}
```

指数型母函数：
有 n 种物品，并且知道每种物品的数量。要求从中选出 m 件物品的排列数。例如有两种物品 A,B，并且数量都是 1，从中选 2 件物品，则排列有"AB","BA"两种。

```
#include<algorithm>
#include <iomanip>
#include<vector>
using namespace std;
#define ll long long
#define inf 0x3f3f3f3f
#define N 12

double c1[N],c2[N],g[N];
int a[N];
void init()
{
    g[0]=1;
    for(int i=1;i<=10;i++)
        g[i]=g[i-1]*i;
}
int main()
{
    int n,m,i,j,k;
    init();
    while(scanf("%d%d",&n,&m)!=EOF)
    {
        memset(c1,0,sizeof(c1));
        memset(c2,0,sizeof(c2));
        for(i=0;i<n;i++)
            scanf("%d",&a[i]);
        for(i=0;i<=a[0];i++)
            c1[i]=1.0/g[i];
        for(i=1;i<n;i++)
        {
            for(j=0;j<=m;j++)
                for(k=0;k<=a[i]&&k+j<=m;k++)
                    c2[j+k]+=c1[j]/g[k];
            for(j=0;j<=m;j++)
            {
                c1[j]=c2[j];
                c2[j]=0;
            }
        }
        printf("%.0lf\n",c1[m]*g[m]);
    }
}
```

莫比乌斯反演

$$F(n)=\sum_{d|n} f(d)$$

$$f(n)=\sum_{d|n} \mu(d)F(n/d)$$

设 A(d)：gcd(a, b)=d 的有多少种
设 B(j): gcd(a, b)是 j 的倍数的有多少种，易知 B(j) = (n/j)*(m/j)
B(j)=A(j)+A(2*j)+A(3*j)+……
由莫比乌斯反演得：
A(j)=μ(1)*B(j)+μ(2)*B(2*j)+μ(3)*B(3*j)+……

分块加速思想
你可以再纸上模拟一下：设 d 在[i, n/(n/i)]的区间上，则该区间内所有的 n/d 都是一样的。

hdu 2841 Visible Trees
求两个数段内互质对的个数。
[cpp] view plaincopy

```
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
#define ll __int64
using namespace std;
#define N 100000
int mib[N+10];
int f[N+10];
bool vis[N+10];
void mobi(){
    int i,j;
    for(i=1;i<=N;i++) mib[i]=1;
    for(i=2;i<=N;i++){
        if(vis[i]) continue;
        for(j=i;j<=N;j+=i){
            vis[j]=1;
            if((j/i)%i==0){
                mib[j]=0;
                continue;
            }
            if(mib[j]==0) continue;
            mib[j]=-mib[j];
        }
```

```
    }
    f[0]=mib[0];
    for(i=1;i<=N;i++)
        f[i]=f[i-1]+mib[i];
}
int main()
{
    int t,b,d,i,j;
    scanf("%d",&t);
    mobi();
    while(t--)
    {
        scanf("%d%d",&b,&d);
        if(b<d)
            swap(b,d);
        ll ans=0;
    //hdu 2841 Visible Trees
    //求两个数段内互质对的个数。
        for(i=1;i<=d;i=j+1)
        {
            j=min(b/(b/i),d/(d/i));
            ans+=(f[j]-f[i-1])*((ll)(b/i)*(d/i));
        }
    //hdu 1695 GCD
    //和上道题，有些小差别，gcd(a,b) 和 gcd(b,a)属于一种。
    for(i=1;i<=d;i=j+1)
        {
            j=min(b/(b/i),d/(d/i));
            ans+=(ll)(f[j]-f[i-1])*((ll)(2*(ll)(b/i)*(d/i))+d/i-(ll)(d/i)*(d/i))/2;
        }
        printf("%I64d\n",ans);
    }
}
```

题意：给 n 个数，n 和 每个数的范围都是 1---222222，求 n 个数中互质的对数。

题解：

f(d) 表示 gcd 为 d 的倍数的数的对数

g(d)表示 gcd 恰好为 d 的数的对数

f(d) = ∑ g(n) (n % d == 0)

g(d) = ∑ mu[n / d] * f(n) (n %d == 0)

上面两个式子就是套路，没啥难的

所以 g(1) = ∑mu[n] * f(n)

求一下 mu【】，枚举一下 n 就可以 AC 了

筛法求下每个数的倍数的个数。在莫比乌斯下就好了

代码：

```
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
#include<iostream>
#define ll long long
using namespace std;
#define N 222222

int mib[N+10];
int f[N+10];
bool vis[N+10];
void mobius()
{
    int i,j;
    for(i=1;i<=N;i++) mib[i]=1,vis[i]=false;
    for(i=2;i<=N;i++)
    {
        if(vis[i]) continue;
        for(j=i;j<=N;j+=i)
        {
            vis[j]=true;
            if((j/i)%i==0)
            {
                mib[j]=0;
                continue;
            }
            mib[j]=-mib[j];
        }
    }
    f[0]=mib[0];
    for(i=1;i<=N;i++)
        f[i]=f[i-1]+mib[i];
}
int p[N+10];
int cnt[N+10],num[N+10];
int main()
{
    mobius();
    int i,j,k;
    int n;
    while(scanf("%d",&n)!=EOF)
```

```
    {
        memset(num,0,sizeof(num));
        memset(cnt,0,sizeof(cnt));
        int mx=0;
        for(i=1;i<=n;i++)
        {
            scanf("%d",&p[i]);
            num[p[i]]++;
            mx=max(mx,p[i]);
        }
        for(i=1;i<=N;i++)
            for(j=i;j<=N;j+=i)
                cnt[i]+=num[j];
        ll ans=0;
        for(i=1;i<=mx;i++)
        {
            if(!cnt[i]) continue;
            ans+=(ll)mib[i]*((ll)(cnt[i])*(cnt[i]-1)/2);
        }
        cout<<ans<<endl;
    }
}
```

给出 N,M

执行如下程序：

```
long long    ans = 0,ansx = 0,ansy = 0;
for(int i = 1; i <= N; i ++)
    for(int j = 1; j <= M; j ++)
        if(gcd(i,j) == 1) ans ++,ansx += i,ansy += j;
cout << ans << " " << ansx << " " << ansy << endl;
```

求 ans，比较简单

求 ansx，（ansy 同理）

定义 A(i),gcd(a,b)=i 时，加到 ansx 上的和

定义 B(i),gcd(a,b)=i，2*i，3*i……时，加到 ansx 上的总和

B(i)=A(i)+A(2*i)+A(3*i)+……

莫比乌斯反演得：

A(i)=μ(1)*B(i)+μ(2)*B(2*i)+μ(3)*B(3*i)+……

A(1)=μ(1)*B(1)+μ(2)*B(2)+μ(3)*B(3)+……

B(i)=(n/i+1)*(n/i)/2*i * (m/i);

即可用莫比乌斯求解该题，再加上分块加速思想的优化。

```
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
#include<iostream>
#define ll long long
using namespace std;
#define N 100000

int mib[N+10];
int f[N+10];
ll ff[N+10];
bool vis[N+10];
void mobius()
{
    int i,j;
    for(i=1;i<=N;i++) mib[i]=1,vis[i]=false;
    for(i=2;i<=N;i++)
    {
        if(vis[i]) continue;
        for(j=i;j<=N;j+=i)
        {
            vis[j]=true;
            if((j/i)%i==0)
            {
                mib[j]=0;
                continue;
            }
            mib[j]=-mib[j];
        }
    }
    f[0]=ff[0]=0;
    for(i=1;i<=N;i++)
    {
        f[i]=f[i-1]+mib[i];
        ff[i]=ff[i-1]+(ll)mib[i]*i;
    }
}

int main()
{
    int n,m,i,j,k;
    mobius();
    while(scanf("%d%d",&n,&m)!=EOF)
    {
        ll ans=0,ansx=0,ansy=0;
        int t=min(n,m);
        for(i=1;i<=t;i=j+1)
        {
            j=min(n/(n/i),m/(m/i));
            ans+=(ll)(f[j]-f[i-1])*(m/i)*(n/i);
            ansx+=(ll)(ff[j]-ff[i-1])*(n/i+1)*(n/i)/2*(m/i);
            ansy+=(ll)(ff[j]-ff[i-1])*(m/i+1)*(m/i)/2*(n/i);
        }
        printf("%lld %lld %lld\n",ans,ansx,ansy);
```

```
    }
}
```

快速幂

```
m^n%k
m 的 n 次方模 k
ll quickpow(ll m,ll n,ll k)
{
    ll b =1;
    while (n >0)
    {
        if (n &1)
            b = (b*m)%k;
        n = n >>1 ;
        m = (m*m)%k;
    }
    return b;
}
```

矩阵快速幂

```
struct node
{
    int a[32][32];
}T,I;
node multi(node A,node B)
{
    int i,j,k;
    node C;
    memset(C.a,0,sizeof(C.a));
    for(i=0;i<cnt;i++)
        for(j=0;j<cnt;j++)
        {
            for(k=0;k<cnt;k++)
            {
                C.a[i][j]+=A.a[i][k]*B.a[k][j];
                C.a[i][j]%=mod;
            }
            C.a[i][j]=(C.a[i][j]+p)%mod;
        }
    return C;
}
```

高斯消元—01 矩阵

题意：n 盏灯，m 个开关，每个开关控制一些灯。给出 n 个灯的状态，询问开关的方案数，使得达到给出的状态。

灯的状态是由控制它的所有开关的状态异或而来，所以每个开关不是 0，就是 1。

高斯消元后，等式左边系数全为 0，等式右边系数不为 0，显然无解。否则答案就是 2^(自由元的个数)。

```
#define N 55
#define ll __int64

int n,m;
int g[N][N],a[N][N];
ll gauss()
{
    int i,j,k,h=0,l=0;
    for(l=0;l<m;l++)
    {
        for(i=h;i<n;i++)
            if(g[i][l])
                break;
        if(i<n)
        {
            if(i!=h)
                for(j=l;j<=m;j++)
                    swap(g[i][j],g[h][j]);
            for(j=h+1;j<n;j++)
                if(g[j][l])
                    for(k=l;k<=m;k++)
                        g[j][k]^=g[h][k];
            h++;
        }
    }
    for(i=n-1;i>=0;i--)
    {
        for(j=i+1;j<n;j++)
            g[i][m]^=(g[j][m]&&g[i][j]);
    }
    for(i=0;i<n;i++)
    {
        int t=0;
        for(j=0;j<m;j++)
            t|=g[i][j];
        if(!t && g[i][m])
            return 0;
    }
    return 1ll<<(m-h);
}
int main()
{
    int t,i,j,k;
```

```
    scanf("%d",&t);
    int res=0;
    while(t--)
    {
        res++;
        scanf("%d%d",&n,&m);
        memset(a,0,sizeof(a));
        for(i=0;i<m;i++)
        {
            scanf("%d",&k);
            while(k--)
            {
                scanf("%d",&j);
                a[j-1][i]=1;
            }
        }
        int q;
        scanf("%d",&q);
        printf("Case %d:\n",res);
        while(q--)
        {
            memcpy(g,a,sizeof(a));
            for(i=0;i<n;i++)
                scanf("%d",&g[i][m]);
            printf("%I64d\n",gauss());
        }
    }
}
```

高斯消元—小数

```
#include<stdio.h>
#include<string.h>
#include<math.h>
#include<iostream>
#include<algorithm>
using namespace std;
#define N 110
#define ll __int64
#define inf 0x3f3f3f3f

int n,m;
double g[N][N],p[N][N];
int nm;
void gauss()
{
    int i,j,k,r;
    for(i=0;i<nm;i++)
    {
        r=i;
        double big=0;
        for(j=i;j<nm;j++)
            if(fabs(g[j][i])>big)
            {
                r=j;
                big=fabs(g[j][i]);
            }
        if(r!=i)
            for(j=0;j<=nm;j++)
                swap(g[r][j],g[i][j]);

        for(k=i+1;k<nm;k++)
        {
            if(g[k][i]==0) continue;
            double f=g[k][i]/g[i][i];
            for(j=i;j<=nm;j++)
                g[k][j]-=f*g[i][j];
        }
    }
    for(i=nm-1;i>=0;i--)
    {
        for(j=i+1;j<nm;j++)
            g[i][nm]-=g[j][nm]*g[i][j];
        g[i][nm]/=g[i][i];
    }
}
int calc(int x,int y)
{
    return (x-1)*m+y-1;
}
int main()
{
    int l,i,j,k,t;
    int res=0;
    while(scanf("%d%d%d",&m,&n,&l))
    {
        if(!n&&!m&&!l) break;
        if(res!=0)
            printf("\n");
        res++;
        memset(g,0,sizeof(g));
        nm=n*m;
        for(i=1;i<=n;i++)
            for(j=1;j<=m;j++)
                scanf("%lf",&p[i][j]);
        for(i=1;i<=n;i++)
            for(j=1;j<=m;j++)
```

```
            {
                int hang=(i-1)*m+j-1;
                g[hang][nm]=p[i][j];
                int sum=0;
                for(k=1;k<=n;k++)
                    for(t=1;t<=m;t++)
                    {
                        if(abs(i-k)+abs(j-t)<=l)
                        {
                            sum++;
                            g[hang][calc(k,t)]=1;
                        }
                    }
                g[hang][nm]*=sum;
            }
        gauss();
        for(i=0;i<nm;i++)
        {
            if((i+1)%m==0)
                printf("%8.2lf\n",g[i][nm]);
            else
                printf("%8.2lf",g[i][nm]);
        }
    }
    //system("pause");
}
```

斐波那契额数列通式

F(n)=(1/ √ 5)*{[(1+ √ 5)/2]^n - [(1- √ 5)/2]^n}

Miller_Rabin 算法进行素数测试

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<time.h>
#include<iostream>
#include<algorithm>
using namespace std;


//*********************************************
*****************
// Miller_Rabin 算法进行素数测试
//速度快，而且可以判断 <2^63 的数
//*********************************************
*****************
const int S=20;//随机算法判定次数，S 越大，判错概率越小


//计算 (a*b)%c.    a,b 都是 long long 的数，直接相乘可能溢出的
//  a,b,c <2^63
long long mult_mod(long long a,long long b,long long c)
{
    a%=c;
    b%=c;
    long long ret=0;
    while(b)
    {
        if(b&1){ret+=a;ret%=c;}
        a<<=1;
        if(a>=c)a%=c;
        b>>=1;
    }
    return ret;
}



//计算  x^n %c
long long pow_mod(long long x,long long n,long long mod)//x^n%c
{
    if(n==1)return x%mod;
    x%=mod;
    long long tmp=x;
    long long ret=1;
    while(n)
    {
        if(n&1) ret=mult_mod(ret,tmp,mod);
        tmp=mult_mod(tmp,tmp,mod);
        n>>=1;
    }
    return ret;
}



//以 a 为基,n-1=x*2^t          a^(n-1)=1(mod n)   验证 n 是不是合数
//一定是合数返回 true,不一定返回 false
```

```
bool check(long long a,long long n,long long x,long long t)
{
    long long ret=pow_mod(a,x,n);
    long long last=ret;
    for(int i=1;i<=t;i++)
    {
        ret=mult_mod(ret,ret,n);
        if(ret==1&&last!=1&&last!=n-1) return true;//合数
        last=ret;
    }
    if(ret!=1) return true;
    return false;
}

// Miller_Rabin()算法素数判定
//是素数返回 true.(可能是伪素数，但概率极小)
//合数返回 false;

bool Miller_Rabin(long long n)
{
    if(n<2)return false;
    if(n==2)return true;
    if((n&1)==0) return false;//偶数
    long long x=n-1;
    long long t=0;
    while((x&1)==0){x>>=1;t++;}
    for(int i=0;i<S;i++)
    {
        long long a=rand()%(n-1)+1;//rand()需要 stdlib.h 头文件
        if(check(a,n,x,t))
            return false;//合数
    }
    return true;
}



//***********************************************
//pollard_rho 算法进行质因数分解
//***********************************************
long long factor[100];//质因数分解结果（刚返回时是无序的）
int tol;//质因数的个数。数组小标从 0 开始

long long gcd(long long a,long long b)
{
    if(a==0)return 1;//???????
    if(a<0) return gcd(-a,b);
    while(b)
    {
        long long t=a%b;
        a=b;
        b=t;
    }
    return a;
}

long long Pollard_rho(long long x,long long c)
{
    long long i=1,k=2;
    long long x0=rand()%x;
    long long y=x0;
    while(1)
    {
        i++;
        x0=(mult_mod(x0,x0,x)+c)%x;
        long long d=gcd(y-x0,x);
        if(d!=1&&d!=x) return d;
        if(y==x0) return x;
        if(i==k){y=x0;k+=k;}
    }
}
//对 n 进行素因子分解
void findfac(long long n)
{
    if(Miller_Rabin(n))//素数
    {
        factor[tol++]=n;
        return;
    }
    long long p=n;
    while(p>=n)p=Pollard_rho(p,rand()%(n-1)+1);
    findfac(p);
    findfac(n/p);
}

int main()
{
    //srand(time(NULL));//需要 time.h 头文件//POJ 上 G++不能加这句话
    long long n;
    while(scanf("%I64d",&n)!=EOF)
    {
        if(n==1)//n==1 需要特判
            continue;
```

```
            tol=0;
            findfac(n);
            for(int i=0;i<tol;i++)printf("%I64d ",factor[i]);
            printf("\n");
            if(Miller_Rabin(n))printf("Yes\n");
            else printf("No\n");
        }
        return 0;
}
```

gcd—lcm

```
int gcd(int a,int b){
        return b?gcd(b,a%b):a;
}

inline int lcm(int a,int b){
        return a/gcd(a,b)*b;
}
```

a^b%c(b 为大整数)

$$A^x = A^{(x \bmod Phi(C) + Phi(C))} \bmod C\ (x \geq Phi(C))$$

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<algorithm>
using namespace std;

long long quickpow(long long n,long long m,long long p)
{
        long long t=1;
        while(m)
        {
                if(m&1) t=t*n%p;
                m=m>>1;
                n=n*n%p;
        }
        return t;
}
long long eular(long long n){
        long long ret=1,i;
        for (i=2;i*i<=n;i++)
                if (n%i==0){
                        n/=i,ret*=i-1;
                        while (n%i==0)
                                n/=i,ret*=i;
                }
        if (n>1)
                ret*=n-1;
        return ret;
}
char b[1000005];
int main()
{
        long long a,c;
        while(~scanf("%lld%s%lld",&a,b,&c))
        {
                a%=c;
                long long tb=0;
                long long phic=eular(c);
                int flag=0;
                int len=strlen(b);
                for(int i=0;i<len;i++)
                {
                        tb=tb*10+b[i]-'0';
                        if(tb>=phic) tb%=phic,flag=1;
                }
                if(flag==1) tb+=phic;
                printf("%lld\n",quickpow(a,tb,c));
        }
}
```

组合数学

基本知识

排列：A(n,k)
圆排列：A(n,k)/k
重排列：n^k

组合：C(n,k)
重组合：C(n+k-1,k) //k 个相同求，放入 n 个不同的盒子
        C(k-1,n-1) // k 个相同求，放入 n 个不同的盒子，k>=n，盒子不能为空

等差数列求和公式：Sn=na1+n(n-1)d/2 或 Sn=n(a1+an)/2
等比数列求和公式：Sn=a1(1-q^n)/(1-q)

$$1^3+2^3+\cdots+n^3=\left[\frac{n(n+1)}{2}\right]^2=(1+2+\cdots+n)^2$$

$$1^2+2^2+3^2+\cdots+n^2=\frac{n(n+1)(2n+1)}{6}$$

错排公式：
D(n) = (n-1) [D(n-2) + D(n-1)]
D(n) = n! [(-1)^2/2! + … + (-1)^(n-1)/(n-1)! + (-1)^n/n!].

$$\lim_{n\to\infty} D_n/n! = e^{-1}$$

卡塔兰数： 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670
Cn = C(2n,n)-C(2n,n+1) = (1/(n+1))*C(2n,n)
Cn+1 = (4n+2)/(n+2)*Cn

- C_n 表示长度 *2n* 的 dyck word 的个数。Dyck word 是一个有 *n* 个 X 和 *n* 个 Y 组成的字串，且所有的前缀字串皆满足 X 的个数大于等于 Y 的个数。以下为长度为 6 的 dyck words:

  XXXYYY XYXXYY XYXYXY XXYYXY XXYXYY

- 将上例的 X 换成左括号，Y 换成右括号，C_n 表示所有包含 *n* 组括号的合法运算式的个数：

  ((())) ()(()) ()()() (())() (()())

  - C_n 表示有 *n* 个节点组成不同构二叉树的方案数。
  - C_n 表示有 *2n+1* 个节点组成不同构满二叉树（full binary tree）的方案数。下图中，*n* 等于 *3*，圆形表示内部节点，月牙形表示外部节点。本质同上。
  - C_n 表示所有在 *n* × *n* 格点中不越过对角线的**单调路径**的个数。一个单调路径从格点左下角出发，在格点右上角结束，每一步均为向上或向右。计算这种路径的个数等价于计算 Dyck word 的个数：X 代表“向右”，Y 代表“向上”.
  - C_n 表示通过连结顶点而将 *n* + 2 边的凸多边形分成三角形的方法个数。
  - 设在圆上选择 2n 个（等间隔的）点。请问将这些点成对的连接起来使得所得到的 n 条线段不相交的方法数是多少？

调和级数：1+1/2+1/3+1/4+...1/n = ln(n+1) + r

C(n,k)(k<=n)的奇偶性取决于(n-k)与 k 的二进制表达式是否存在同一位上的两个数码均为 1，若存在，则为偶数，反之为奇数

2	3	5	7	11	13	17	19
23	29	31	37	41	43	47	53
59	61	67	71	73	79	83	89
97	101	103	107	109	113	127	131
137	139	149	151	157	163	167	173
179	181	191	193	197	199	211	223
227	229	233	239	241	251	257	263
269	271	277	281	283	293	307	311
313	317	331	337	347	349	353	359
367	373	379	383	389	397	401	409
419	421	431	433	439	443	449	457
461	463	467	479	487	491	499	

组合 C（）模板

```
#define ll __int64
ll c[N+M][N];  //数组一定不要开小，开小 wa，不是 re，写的记忆话搜索
ll mod=1e9+7;
memset(c,-1,sizeof(c));//不要忘记初始化
ll C(ll n,ll k)
{
    if(c[n][k]!=-1) return c[n][k];
    if(k==1) return c[n][k]=n;
    if(k==0) return c[n][k]=1;
    if(n==k) return c[n][k]=1;
    if(k > (n>>1)) return c[n][k]=C(n,n-k);
    return c[n][k]=(C(n-1,k-1)+C(n-1,k))%mod;
}
```

m 个相同的小球放入 n 个盒子里，有
c（n-1+m，m） 种放法

拓扑排序

a 做要先做 b，b 到 a 连边建图
d[]记录每个点的入度，
每次删除一个度为 0 的点，并删除其连的边，最后能删完，则图中无环。

```
bool check()
{
    int i,j,k;
    while(!q.empty())
        q.pop();
    for(i=0;i<=n;i++)
    {
        d[i]=0;
        f[i].clear();
    }
    for(i=0;i<m;i++)
    {
        f[b[i]].push_back(a[i]);
        d[a[i]]++;
    }
    for(i=1;i<=n;i++)
        if(d[i]==0)
            q.push(i);
    int sum=0;
    while(!q.empty())
    {
        int t=q.front();
        q.pop();
        sum++;
        int len=f[t].size();
        for(j=0;j<len;j++)
        {
            int v=f[t][j];
            d[v]--;
            if(d[v]==0)
                q.push(v);
        }
    }
    if(sum==n)
        return true;
    else
        return false;
}
```

第一类 Stirling 数

第一类 Stirling 数（n 个人分成 k 组，每组内再按特定顺序围圈的分组方法的数目）

第一类 Stirling 数是有正负的，其绝对值是 n 个元素的项目分作 k 个环排列的方法数目

例如 s(4,2)：

{A,B},{C,D}
{A,C},{B,D}
{A,D},{B,C}
{A},{B,C,D}
{A},{B,D,C}
{B},{A,C,D}
{B},{A,D,C}
{C},{A,B,D}
{C},{A,D,B}
{D},{A,B,C}
{D},{A,C,B}

代码：

```
memset(s,-1,sizeof(s));
ll s[2020][2020];
ll S(ll n,ll k)
{
    if(s[n][k]!=-1) return s[n][k];
    if(k==0 || n<k) return 0;
    if(n==1 && k==1) return s[n][k]=1;
    return s[n][k]=(S(n-1,k-1)+(n-1)*S(n-1,k))%mod;
}
```

字符串

KMP

首先弄明白 next[]数组的意义，next[i]表示 第 i 个数的前 L 个字符串等于 后 L 个数的字符串，且字符串长度最长，即 L 最大；

例如：字符串　abcabcda

next[0]=-1;
next[1]=0;　a
next[2]=0;　ab
next[3]=0;　abc
next[4]=1;　abca，
next[5]=2;　abcab，
next[6]=3;　abcabc，
next[7]=0;　abcabcd，
next[8]=1;　abcabcda，

next 数组表示当 j 失配时，下次匹配 next[j]=j;

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
```

```
char a[10010],b[1000010];
int nex[10010];
void getnext(int n)//求子串 nex 数组
{
	int i=0,j=-1;
	nex[0]=-1;
	while(i<n)
	{
		if(j==-1 || a[i]==a[j])
			i++,j++,nex[i]=j;
		else
			j=nex[j];
	}
}
int calc()//求子串在母串中出现的次数
{
	int n=strlen(a);
	int h=strlen(b);
	int i=0,j=0,sum=0;
	while(i<h)
	{
		if(j==-1 || a[j]==b[i])
			i++,j++;
		else
			j=nex[j];
		if(j==n)
			sum++,j=nex[j];
	}
	return sum;
}
int main()
{
	int t;
	scanf("%d",&t);
	while(t--)
	{
		scanf("%s%s",a,b);
		int n=strlen(a);
		getnext(n);
		printf("%d\n",calc());
	}
}
```

（Manacher）算法----O(n)回文子串

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

void findBMstr(string str)
{
	int *p = new int[str.size() + 1];
	memset(p, 0, sizeof(p));

	int mx = 0, id = 0;
	for(int i = 1; i <=    str.size(); i++)
	{
		if(mx > i)
			p[i] = (p[2*id - i] < (mx - i) ? p[2*id - i] : (mx -
i));
		else
			p[i] = 1;
		while(str[i - p[i]] == str[i + p[i]])
			p[i]++;
		if(i + p[i] > mx)
		{
			mx = i + p[i];
			id = i;
		}
	}
	int max = 0, ii;
	for(int i = 1; i < str.size(); i++)
	{
		if(p[i] > max)
		{
			ii = i;
			max = p[i];
		}
	}
	max--;
	int start = ii - max ;
	int end = ii + max;
	for(int i = start; i <= end; i++)
		if(str[i] != '#')
			cout << str[i];
	cout << endl;
	delete    p;
}

int main()
{
	string str = "12212321";
	string str0;
	str0 += "$#";
	for(int i = 0; i < str.size(); i++)
	{
```

```
            str0 += str[i];
            str0 += "#";
        }
        cout << str0 << endl;
        findBMstr(str0);
        return 0;
}
```

网络流

网络流

```
const int maxnode = 1000 + 5;
const int maxedge = 2 * 1000 + 5;//开边数 x2.。
const int oo = 1000000000;
int node, src, dest, nedge;
int head[maxnode], point[maxedge], next1[maxedge], flow[maxedge], capa[maxedge];//point[x]==y 表示第 x 条边连接 y，head，next 为邻接表，flow[x]表示 x 边的动态值，capa[x]表示 x 边的初始值
int dist[maxnode], Q[maxnode], work[maxnode];//dist[i]表示 i 点的等级
void init(int _node, int _src, int _dest){//初始化，node 表示点的个数，src 表示起点，dest 表示终点
        node = _node;
        src = _src;
        dest = _dest;
        for (int i = 0; i < node; i++) head[i] = -1;
        nedge = 0;
}
void addedge(int u, int v, int c1, int c2){//增加一条 u 到 v 流量为 c1，v 到 u 流量为 c2 的两条边
        point[nedge] = v, capa[nedge] = c1, flow[nedge] = 0, next1[nedge] = head[u], head[u] = (nedge++);
        point[nedge] = u, capa[nedge] = c2, flow[nedge] = 0, next1[nedge] = head[v], head[v] = (nedge++);
}
bool dinic_bfs(){
        memset(dist, 255, sizeof (dist));
        dist[src] = 0;
        int sizeQ = 0;
        Q[sizeQ++] = src;
        for (int cl = 0; cl < sizeQ; cl++)
                for (int k = Q[cl], i = head[k]; i >= 0; i = next1[i])
                        if (flow[i] < capa[i] && dist[point[i]] < 0){
                                dist[point[i]] = dist[k] + 1;
                                Q[sizeQ++] = point[i];
                        }
        return dist[dest] >= 0;
}
int dinic_dfs(int x, int exp){
        if (x == dest) return exp;
        for (int &i = work[x]; i >= 0; i = next1[i]){
                int v = point[i], tmp;
                if (flow[i] < capa[i] && dist[v] == dist[x] + 1 && (tmp = dinic_dfs(v, min(exp, capa[i] - flow[i]))) > 0){
                        flow[i] += tmp;
                        flow[i^1] -= tmp;
                        return tmp;
                }
        }
        return 0;
}
int dinic_flow(){
        int result = 0;
        while (dinic_bfs()){
                for (int i = 0; i < node; i++) work[i] = head[i];
                while (1){
                        int delta = dinic_dfs(src, oo);
                        if (delta == 0) break;
                        result += delta;
                }
        }
        return result;
}
//建图前,运行一遍 init()；
//加边时，运行 addedge(a,b,c,0),表示点 a 到 b 流量为 c 的边建成（注意点序号要从 0 开始）
//求解最大流运行 dinic_flow(),返回值即为答案
```

费用流

```
const int N = 1010;//点
const int M = 2 * 10010;//边
const int inf = 1000000000;
struct Node{//边，点 f 到点 t，流量为 c，费用为 w
        int f, t, c, w;
}e[M];
int next1[M], point[N], dis[N], q[N], pre[N], ne;//ne 为已添加的边数，next，point 为邻接表,dis 为花费，pre 为父亲节点
bool u[N];
void init(){
        memset(point, -1, sizeof(point));
```

```
    ne = 0;
}
void add_edge(int f, int t, int d1, int d2, int w){//f 到 t 的一条边，流量为 d1,反向流量 d2,花费 w,反向边花费-w（可以反悔）
    e[ne].f = f, e[ne].t = t, e[ne].c = d1, e[ne].w = w;
    next1[ne] = point[f], point[f] = ne++;
    e[ne].f = t, e[ne].t = f, e[ne].c = d2, e[ne].w = -w;
    next1[ne] = point[t], point[t] = ne++;
}
bool spfa(int s, int t, int n){
    int i, tmp, l, r;
    memset(pre, -1, sizeof(pre));
    for(i = 0; i < n; ++i)
        dis[i] = inf;
    dis[s] = 0;
    q[0] = s;
    l = 0, r = 1;
    u[s] = true;
    while(l != r) {
        tmp = q[l];
        l = (l + 1) % (n + 1);
        u[tmp] = false;
        for(i = point[tmp]; i != -1; i = next1[i]) {
            if(e[i].c && dis[e[i].t] > dis[tmp] + e[i].w) {
                dis[e[i].t] = dis[tmp] + e[i].w;
                pre[e[i].t] = i;
                if(!u[e[i].t]) {
                    u[e[i].t] = true;
                    q[r] = e[i].t;
                    r = (r + 1) % (n + 1);
                }
            }
        }
    }
    if(pre[t] == -1)
        return false;
    return true;
}
void MCMF(int s, int t, int n, int &flow, int &cost){//起点 s，终点 t，点数 n，最大流 flow，最小花费 cost
    int tmp, arg;
    flow = cost = 0;
    while(spfa(s, t, n)) {
        arg = inf, tmp = t;
        while(tmp != s) {
            arg = min(arg, e[pre[tmp]].c);
            tmp = e[pre[tmp]].f;
        }
        tmp = t;
        while(tmp != s) {
            e[pre[tmp]].c -= arg;
            e[pre[tmp] ^ 1].c += arg;
            tmp = e[pre[tmp]].f;
        }
        flow += arg;
        cost += arg * dis[t];
    }
}
//建图前运行 init()
//节点下标从 0 开始
//加边时运行 add_edge(a,b,c,0,d)表示加一条 a 到 b 的流量为 c 花费为 d 的边（注意花费为单位流量花费）
//特别注意双向边，运行 add_edge(a,b,c,0,d),add_edge(b,a,c,0,d)较好，不要只运行一次 add_edge(a,b,c,c,d),费用会不对。
//求解时代入 MCMF(s,t,n,v1,v2)，表示起点为 s，终点为 t，点数为 n 的图中，最大流为 v1，最大花费为 v2
```

图论

最短路

（1）floyd：

```
for(int k=1;k<=n;k++)
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                if(map[i][j]>map[i][k]+map[k][j])

map[i][j]=map[i][k]+map[k][j];
```

（2）dijkstra：

```
void dijkstra(int a)
{
    int i,j,k;
    memset(f,0,sizeof(f));
    for(i=1;i<=N;i++)
        d[i]=map[a][i];
    f[a]=1;
    for(i=1;i<N;i++)
    {
        int min=0x3f3f3f3f;
        for(j=1;j<=N;j++)
```

```
            if(d[j]<min && f[j]==0){
                min=d[j],k=j;
            }
        f[k]=1;
        for(j=1;j<=N;j++)
            if(d[k]+map[k][j]<d[j] && f[j]==0)
                d[j]=d[k]+map[k][j];
    }
}
```

（3）spfa

```
#define N 100010
struct node
{
    int b,len;
};
vector<node> f[N];
int dis[N];
int flag[N];
void spfa(int x)
{
    int i;
    queue<int> q;
    for(i=0;i<=n;i++)
    {
        dis[i]=inf;
        flag[i]=0;
    }
    dis[x]=0;
    flag[x]=1;
    q.push(x);
    while(!q.empty())
    {
        int t=q.front();
        q.pop();
        flag[t]=0;
        int l=f[t].size();
        for(i=0;i<l;i++)
        {
            int en=f[t][i].len,tb=f[t][i].b;
            if(dis[t]+en<dis[tb])
            {
                dis[tb]=dis[t]+en;
                if(flag[tb]==0)
                {
                    flag[tb]=1;
                    q.push(tb);
                }
            }
        }
    }
}
```

最小生成树

```
#define V 110 //点的个数
#define E 5100   //边的个数

int parent[V];
memset(parent,-1,sizoef(parent));
int root(int p)
{
    if(parent[p]==-1) return p;
    else return parent[p]=root(parent[p]);
}
void merge(int a,int b)
{
    a=root(a);
    b=root(b);
    parent[a]=b;
}
```

数据结构

单调队列

复习下单调队列~
单调队列正如其名，是一个排序（这道题是从大到小）的队列，而且能够保证所有的元素入队列一次出队列一次，所以平摊到每个元素的复杂度就是 O(1)。

1）插入操作：从队尾开始查找，把队尾小于待插入元素的元素全部删除，再加入待插入的元素。这个操作最坏的情况下是 O(n)，但是我们采用聚集分析的方法，知道每个元素最多删除一次，那么 N 个元素删除 N 次，平摊到每一操作的复杂度就是 O(1)了。

2）删除队首元素：比如本文给的那个题，窗口一直往后移动，每一次移动都会删除一个元素，所以很可能队首会是要删除的元素，那么每次移动窗口的元素要进行一次检查，如果队首元素失效的话，就删掉队首元素。

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int p[1001000];
```

```
int a[1001000],b[1001000];
int ta[1001000],tb[1001000]; //ta 区间最小值，tb 区间最大值
int main()
{
    int n,k;
    int i,j;
    scanf("%d%d",&n,&k);
    for(i=1;i<=n;i++)
        scanf("%d",&p[i]);
    int l1=1,l2=1,r1=1,r2=1;
    int t1=1,t2=1;
    for(i=1;i<=n;i++)
    {
        while(r1>l1 && p[i]<p[a[r1-1]])
            r1--;
        a[r1++]=i;
        while(i-a[l1]>=k)
            l1++;
        if(i>=k)
            ta[t1++]=p[a[l1]];

        while(r2>l2 && p[i]>p[b[r2-1]])
            r2--;
        b[r2++]=i;
        while(i-b[l2]>=k)
            l2++;
        if(i>=k)
            tb[t2++]=p[b[l2]];
    }
    printf("%d",ta[1]);
    for(i=2;i<t1;i++)
        printf(" %d",ta[i]);
    printf("\n%d",tb[1]);
    for(i=2;i<t2;i++)
        printf(" %d",tb[i]);
    printf("\n");
    //system("pause");
}
```

树链剖分-点值

```
#include<stdio.h>
#include<string.h>
#include<algorithm>
#include<stdlib.h>
#include<iostream>
#include<vector>
using namespace std;
#define ll long long
const int N=50010;

struct Edge
{
    int to,next;
}edge[N*2];
int head[N],tot;
int top[N];//top[v]表示 v 所在的重链的顶端节点
int fa[N]; //父亲节点
int deep[N];//深度
int num[N];//num[v]表示以 v 为根的子树的节点数
int p[N];//p[v]表示 v 与其父亲节点的连边在线段树中的位置
int fp[N];//和 p 数组相反
int son[N];//重儿子
int pos;
void init()
{
    tot=0;
    memset(head,-1,sizeof(head));
    pos=1;
    memset(son,-1,sizeof(son));
}
void addedge(int u,int v)
{
    edge[tot].to=v;                edge[tot].next=head[u];
head[u]=tot++;
}
void dfs1(int u,int pre,int d)//第一遍 dfs 求出 fa，deep，num，son
{
    deep[u]=d;
    fa[u]=pre;
    num[u]=1;
    for(int i=head[u];i!=-1;i=edge[i].next)
    {
        int v=edge[i].to;
        if(v!=pre)
        {
            dfs1(v,u,d+1);
            num[u]+=num[v];
            if(son[u]==-1 || num[v]>num[son[u]])
                son[u]=v;
        }
    }
}
void getpos(int u,int sp)//第二遍 dfs 求出 top 和 p
```

```
{
    top[u]=sp;
    p[u]=pos++;
    fp[p[u]]=u;
    if(son[u]==-1) return ;
    getpos(son[u],sp);
    for(int i=head[u];i!=-1;i=edge[i].next)
    {
        int v=edge[i].to;
        if(v!=son[u] && v!=fa[u])
            getpos(v,v);
    }
}

int e[N];
int in[N];
int n;
int Lowbit(int t)
{
    return t&(-t);
}
int Sum(int p)
{
    int sum=0;
    while(p>0)
    {
        sum+=in[p];
        p-=Lowbit(p);
    }
    return sum;
}
void add(int p,int num)
{
    while(p<=n)
    {
        in[p]+=num;
        p+=Lowbit(p);
    }
}
void change(int u,int v,int val)//查询 u->v 边的最大值
{
    int f1=top[u],f2=top[v];
    while(f1!=f2)
    {
        if(deep[f1]<deep[f2])
        {
            swap(f1,f2);
            swap(u,v);
        }
        add(p[f1],val);//p[f1],p[u]
        add(p[u]+1,-val);
        u=fa[f1]; f1=top[u];
    }
    if(deep[u]>deep[v]) swap(u,v);//p[u],p[v]
    add(p[u],val);
    add(p[v]+1,-val);
}
int main()
{
    int m,q;
    while(scanf("%d%d%d",&n,&m,&q)!=EOF)
    {
        init();
        int i,j,k;
        for(i=1;i<=n;i++)
            scanf("%d",&e[i]);
        while(m--)
        {
            int v,u;
            scanf("%d%d",&u,&v);
            addedge(u,v);
            addedge(v,u);
        }
        dfs1(1,0,0);
        getpos(1,1);
        memset(in,0,sizeof(in));
        for(i=1;i<=n;i++)
        {
            add(p[i],e[i]);
            add(p[i]+1,-e[i]);
        }
        char op[2];
        while(q--)
        {
            scanf("%s",op);
            if(op[0]=='Q')
            {
                int v;
                scanf("%d",&v);
                printf("%d\n",Sum(p[v]));
            }
            else
            {
                int v,u,val;
                scanf("%d%d%d",&v,&u,&val);
                if(op[0]=='D')
                    val=-val;
                change(u,v,val);
```

```
            }
        }
    }
}
```

树链剖分-段值

```
#include<stdio.h>
#include<string.h>
#include<algorithm>
#include<stdlib.h>
#include<iostream>
#include<vector>
using namespace std;
#define ll long long
const int N=10010;

struct Edge
{
    int to,next;
}edge[N*2];
int head[N],tot;
int top[N];//top[v]表示 v 所在的重链的顶端节点
int fa[N]; //父亲节点
int deep[N];//深度
int num[N];//num[v]表示以 v 为根的子树的节点数
int p[N];//p[v]表示 v 与其父亲节点的连边在线段树中的位置
int fp[N];//和 p 数组相反
int son[N];//重儿子
int pos;
void init()
{
    tot=0;
    memset(head,-1,sizeof(head));
    pos=1;
    memset(son,-1,sizeof(son));
}
void addedge(int u,int v)
{
    edge[tot].to=v;                edge[tot].next=head[u];
head[u]=tot++;
}
void dfs1(int u,int pre,int d)//第一遍 dfs 求出 fa，deep，num，son
{
    deep[u]=d;
    fa[u]=pre;
    num[u]=1;
    for(int i=head[u];i!=-1;i=edge[i].next)
    {
        int v=edge[i].to;
        if(v!=pre)
        {
            dfs1(v,u,d+1);
            num[u]+=num[v];
            if(son[u]==-1 || num[v]>num[son[u]])
                son[u]=v;
        }
    }
}
void getpos(int u,int sp)//第二遍 dfs 求出 top 和 p
{
    top[u]=sp;
    p[u]=pos++;
    fp[p[u]]=u;
    if(son[u]==-1) return ;
    getpos(son[u],sp);
    for(int i=head[u];i!=-1;i=edge[i].next)
    {
        int v=edge[i].to;
        if(v!=son[u] && v!=fa[u])
            getpos(v,v);
    }
}
struct node
{
    int l,r,mx;
}tree[4*N];
void build(int id,int x,int y)
{
    tree[id].l=x;
    tree[id].r=y;
    if(x==y) return ;
    int mid=(x+y)>>1;
    build(id<<1,x,mid);
    build(id<<1|1,mid+1,y);
}
void pushup(int id)
{
    tree[id].mx=max(tree[id<<1].mx,tree[id<<1|1].mx);
}
void update(int id,int x,int y)
{
    if(tree[id].l==x && tree[id].r==x)
    {
        tree[id].mx=y;
```

```
        return ;
    }
    int mid=(tree[id].l+tree[id].r)>>1;
    if(x<=mid)
        update(id<<1,x,y);
    else
        update(id<<1|1,x,y);
    pushup(id);
}
int query(int id,int x,int y)
{
    if(tree[id].l==x && tree[id].r==y)
        return tree[id].mx;
    int mid=(tree[id].l+tree[id].r)>>1;
    if(y<=mid)
        return query(id<<1,x,y);
    else if(mid+1<=x)
        return query(id<<1|1,x,y);
    else
        return
max(query(id<<1,x,mid),query(id<<1|1,mid+1,y));
}
int find(int u,int v)//查询 u->v 边的最大值
{
    int f1=top[u],f2=top[v];
    int tmp=0;
    while(f1!=f2)
    {
        if(deep[f1]<deep[f2])
        {
            swap(f1,f2);
            swap(u,v);
        }
        tmp=max(tmp, query(1,p[f1],p[u]));//p[f1],p[u]
        u=fa[f1]; f1=top[u];
    }
    if(u==v) return tmp;
    if(deep[u]>deep[v]) swap(u,v);//p[son[u]],p[v]
    return max(tmp, query(1,p[son[u]],p[v]));
}
int e[N][3];
int main()
{
    int t;
    scanf("%d",&t);
    while(t--)
    {
        int n,i,j,k;
        scanf("%d",&n);
        init();
        for(i=1;i<n;i++)
        {
            scanf("%d%d%d",&e[i][0],&e[i][1],&e[i][2]);
            addedge(e[i][0],e[i][1]);
            addedge(e[i][1],e[i][0]);
        }
        dfs1(1,0,0);
        getpos(1,1);
        build(1,1,pos-1);
        for(i=1;i<n;i++)
        {
            if(deep[e[i][0]]>deep[e[i][1]])
                swap(e[i][0],e[i][1]);
            update(1,p[e[i][1]],e[i][2]);
        }
        char op[10];
        while(scanf(" %s",op))
        {
            if(op[0]=='D') break;
            if(op[0]=='C')
            {
                int a,b;
                scanf("%d%d",&a,&b);
                update(1,p[e[a][1]],b);
            }
            else
            {
                int a,b;
                scanf("%d%d",&a,&b);
                printf("%d\n",find(a,b));
            }
        }
    }
}
```

树状数组模板

```
//一维树状数组
#include<stdio.h>
#include<string.h>
#define N 50050
int n;
int in[N];

int Lowbit(int t)
{
    return t&(-t);
```

```
}
int Sum(int p)
{
	int sum=0;
	while(p>0)
	{
		sum+=in[p];
		p-=Lowbit(p);
	}
	return sum;
}
void plus(int p,int num)
{
	while(p<=n)
	{
		in[p]+=num;
		p+=Lowbit(p);
	}
}



//二维树状数组
#define N 1010
int n=1001;
int c[N][N];
int map[N][N];
int lowbit(int x)
{
	return x&(-x);
}
void update(int x,int y,int val)
{
	int i,j;
	for(i=x;i<=n;i+=lowbit(i))
		for(j=y;j<=n;j+=lowbit(j))
			c[i][j]+=val;
}
int getsum(int x,int y)
{
	int i,j,sum=0;
	for(i=x;i>0;i-=lowbit(i))
		for(j=y;j>0;j-=lowbit(j))
			sum+=c[i][j];
	return sum;
}
```

拓扑排序

a 做要先做 b，b 到 a 连边建图
d[]记录每个点的入度，
每次删除一个度为 0 的点，并删除其连的边，最后能删完，则图中无环。

```
queue<int> q;
bool check(int mid)
{
	int i,j,k;
	while(!q.empty())
		q.pop();
	for(i=0;i<=n;i++)
	{
		d[i]=0;
		f[i].clear();
	}
	for(i=0;i<m;i++)
	{
		if(c[i]>mid)
		{
			f[b[i]].push_back(a[i]);
			d[a[i]]++;
		}
	}
	for(i=1;i<=n;i++)
		if(d[i]==0)
			q.push(i);
	int sum=0;
	while(!q.empty())
	{
		int t=q.front();
		q.pop();
		sum++;
		int len=f[t].size();
		for(j=0;j<len;j++)
		{
			int v=f[t][j];
			d[v]--;
			if(d[v]==0)
				q.push(v);
		}
	}
	if(sum==n)
		return true;
	else
		return false;
```

```
}
```

经典题

nlog(n)最长上升子序列

```
#include<iostream>
#include<vector>
#define MAX 1010
using namespace std;
vector<int> len;

// 这里我返回的满足 len[k]>=val[i]且 k 最小的位置

// 和上文红色部分的描述是等价的，只是变成了更新 len[k]，而不是 len[k+1]
int bisearch(int val){
	int left=0, right=len.size()-1;
	while(left<=right){
		int mid = (left+right)>>1;
		if(len[mid] < val) left = mid+1;
		else right = mid-1;
	}
	return left;
}

int main(){
	int n, val;
	cin>>n;
	for(int i=0; i<n; ++i){
		cin>>val;
		int k = bisearch(val);
		if(len.size()==k) len.push_back(val);
		else len[k] = val;
	}
	cout<<len.size()<<endl;
	return 0;
}
```

二分查找用函数：

```
#include<iostream>
#include<algorithm>
#define MAX 1010
using namespace std;

int p[1010];
int main(){
	int n, val;
	cin>>n;
	int cnt=0;
	for(int i=0; i<n; ++i){
		cin>>val;
		int k = lower_bound(p,p+cnt,val)-p;
		if(cnt==k) p[cnt++]=val;
		else p[k] = val;
	}
	cout<<cnt<<endl;
	return 0;
}
```

TSP

```
TSP：
  for(i=0;i<end;i++)    //end=(1<<k);
                for(j=0;j<k;j++)
                        if((i>>j)&1){
                                for(b=0;b<k;b++){
                                        if((i>>b)&1)

dp[i][j]=min(dp[i][j],dp[i-(1<<j)][b]+dist[b][j]);
                                }
                        }
```

双调 TSP

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<iostream>
#include<cstdlib>
#include<vector>
#include<cmath>
using namespace std;
#define inf 0x3f3f3f3f

int n;
double x[202],y[202];
double d[202][202];
double dp[202][202];
double dis(double a,double b,double c,double d)
{
	return sqrt((a-c)*(a-c)+(b-d)*(b-d));
```

```
}
int main()
{
	int i,j,k;
	while(cin>>n)
	{
		for(i=1;i<=n;i++)
		{
			cin>>x[i]>>y[i];
			for(j=1;j<i;j++)
				d[i][j]=dis(x[i],y[i],x[j],y[j]);
		}
		dp[2][1]=d[2][1];
		for(i=3;i<=n;i++)
		{
			for(j=1;j<=i-2;j++)
				dp[i][j]=dp[i-1][j]+d[i][i-1];
			dp[i][i-1]=inf;
			for(j=1;j<=i-2;j++)
				if(dp[i-1][j]+d[i][j]<dp[i][i-1])
					dp[i][i-1]=dp[i-1][j]+d[i][j];
		}
		printf("%.2lf\n",dp[n][n-1]+d[n][n-1]);
	}
}
```

技巧

C++高精度模板

```
#include<stdio.h>
#include<string.h>
#include<math.h>
#include<iostream>
#include<algorithm>
#include<string>
using namespace std;

struct Bigint {
	// representations and structures
	string a; // to store the digits
	int sign; // sign = -1 for negative numbers, sign = 1 otherwise
	// constructors
	Bigint() {} // default constructor
	Bigint( string b ) { (*this) = b; } // constructor for string

	// some helpful methods
	int size() { // returns number of digits
		return (int)a.size();
	}
	Bigint inverseSign() { // changes the sign
		sign *= -1;
		return (*this);
	}
	Bigint normalize( int newSign ) { // removes leading 0, fixes sign
		for( int i = (int)a.size() - 1; i > 0 && a[i] == '0'; i-- )
			a.erase(a.begin() + i);
		sign = ( a.size() == 1 && a[0] == '0' ) ? 1 : newSign;
		return (*this);
	}

	// assignment operator
	void operator = ( string b ) { // assigns a string to Bigint
		a = b[0] == '-' ? b.substr(1) : b;
		reverse( a.begin(), a.end() );
		this->normalize( b[0] == '-' ? -1 : 1 );
	}

	// conditional operators
	bool operator < ( const Bigint &b ) const { // less than operator
		if( sign != b.sign ) return sign < b.sign;
		if( a.size() != b.a.size() )
			return sign == 1 ? a.size() < b.a.size() : a.size() > b.a.size();
		for( int i = (int)a.size() - 1; i >= 0; i-- ) if( a[i] != b.a[i] )
			return sign == 1 ? a[i] < b.a[i] : a[i] > b.a[i];
		return false;
	}
	bool operator == ( const Bigint &b ) const { // operator for equality
		return a == b.a && sign == b.sign;
	}
	// mathematical operators
	Bigint operator + ( Bigint b ) { // addition operator overloading
		if( sign != b.sign ) return (*this) - b.inverseSign();
		Bigint c;
		for(int i = 0, carry = 0; i<a.size() || i<b.size() || carry; i++ ) {
			carry+=(i<a.size() ? a[i]-48 : 0)+(i<b.a.size() ? b.a[i]-48 : 0);
			c.a += (carry % 10 + 48);
```

```
            carry /= 10;
        }
        return c.normalize(sign);
    }
    Bigint operator - ( Bigint b ) { // subtraction operator overloading
        if( sign != b.sign ) return (*this) + b.inverseSign();
        int s = sign; sign = b.sign = 1;
        if( (*this) < b ) return ((b - (*this)).inverseSign()).normalize(-s);
        Bigint c;
        for( int i = 0, borrow = 0; i < a.size(); i++ ) {
            borrow = a[i] - borrow - (i < b.size() ? b.a[i] : 48);
            c.a += borrow >= 0 ? borrow + 48 : borrow + 58;
            borrow = borrow >= 0 ? 0 : 1;
        }
        return c.normalize(s);
    }
    Bigint operator * ( Bigint b ) { // multiplication operator overloading
        Bigint c("0");
        for( int i = 0, k = a[i] - 48; i < a.size(); i++, k = a[i] - 48 ) {
            while(k--) c = c + b; // ith digit is k, so, we add k times
            b.a.insert(b.a.begin(), '0'); // multiplied by 10
        }
        return c.normalize(sign * b.sign);
    }
    Bigint operator / ( Bigint b ) { // division operator overloading
        if( b.size() == 1 && b.a[0] == '0' ) b.a[0] /= ( b.a[0] - 48 );
        Bigint c("0"), d;
        for( int j = 0; j < a.size(); j++ ) d.a += "0";
        int dSign = sign * b.sign; b.sign = 1;
        for( int i = (int)a.size() - 1; i >= 0; i-- ) {
            c.a.insert( c.a.begin(), '0');
            c = c + a.substr( i, 1 );
            while( !( c < b ) ) c = c - b, d.a[i]++;
        }
        return d.normalize(dSign);
    }
    Bigint operator % ( Bigint b ) { // modulo operator overloading
        if( b.size() == 1 && b.a[0] == '0' ) b.a[0] /= ( b.a[0] - 48 );
        Bigint c("0");
        b.sign = 1;
        for( int i = (int)a.size() - 1; i >= 0; i-- ) {
            c.a.insert( c.a.begin(), '0');
            c = c + a.substr( i, 1 );
            while( !( c < b ) ) c = c - b;
        }
        return c.normalize(sign);
    }
    // output method
    void print() {
        if( sign == -1 ) putchar('-');
        for( int i = (int)a.size() - 1; i >= 0; i-- ) putchar(a[i]);
        printf("\n");
    }
};
```

Bigint 功能：

```
Bigint one=string("1");     常数赋值
Bigint zero=string("0");

string n;
Bigint nn=n; //赋值
==
+
-
*
、
%
nn.print();     输出
```

java 大整数模板

```
import java.util.*;
import java.io.*;
import java.math.*;
public class Main
{
    public static void main(String args[])
    {
        Scanner in = new Scanner(System.in);
        BigInteger N,D,x,y,four,n18,one,gg;
        while(in.hasNext()){
            int n, d;
            n = in.nextInt();
            d = in.nextInt();
            if(n == 1 && d == 1){
```

```
                System.out.println(0);
                continue;
                }
                N = BigInteger.valueOf(n);
                D = BigInteger.valueOf(d);
                four = BigInteger.valueOf(4);
                n18 = BigInteger.valueOf(18);
                one = BigInteger.valueOf(1);
                x =
(N.add(four)).multiply(N.add(four)).multiply(D).multiply(D.
add(one.negate()));
                y = n18.multiply(N.pow(d));
                if(x.equals(y))System.out.println("1");
                else{
                gg = x.gcd(y);
                System.out.print(x.divide(gg));
                System.out.print("/");
                System.out.println(y.divide(gg));
                }
            }
        }
}
```

一些函数

int k = lower_bound(p,p+cnt,val)-p;

iterator lower_bound(const key_type &key): 返回一个迭代器，指向键值>= key 的第一个元素。

iterator upper_bound(const key_type &key):返回一个迭代器，指向键值> key 的第一个元素。

对有序的数组去重

```
sort(p,p+n);
n=unique(p,p+n)-p;
```

二分模板

（3）二分模板

符和（某种情况）的最大值

```
while(l<r)
{
    int m=(l+r+1)/2;
    if(check(m))
        l=m;
    else
        r=m-1;
}
return l;
```

符合 (某种情况) 的最小值

```
while(l<r)
{
    int m=(l+r)/2;
    if(check(m))
        r=m;
    else
        l=m+1;
}
return r;
```

三分模板

1.小数三分：

```
#define inf (1e6)
#define EPS (1e-8)
double Calc(double a)
{
    ……
}
double Solve()
{
    double Left, Right;
    double mid, midmid;
    double mid_area, midmid_area;
    Left = -inf; Right = inf;
    while (Left + EPS < Right)
    {
        mid = (Left + Right) / 2;
        midmid = (mid + Right) / 2;
        double mid_area = Calc(mid);
        double midmid_area = Calc(midmid);
        // 假设求解最小极值.最大极值改为
(mid_area >= midmid_area)
        if (mid_area <= midmid_area) Right = midmid;
        else Left = mid;
    }
    return Left;
}
```

2.整数三分

```
#define ll long long

    ll l=0,r=1e9+1;
    ll m1,m2;
    ll ans;
```

```
    ll t,tt;
    while(l<=r)
    {
        m1 = l+(r-l)/3;
        m2 = r-(r-l)/3;
        t = f(m1);
        tt = f(m2);//计算值
        if(t>tt)    //“>”时求最小值，“<”时求最大值
        {
            l = m1+1;
            ans = m2;
        }
        else
        {
            r = m2-1;
            ans = m1;
        }
    }
    printf("%I64d\n",f(ans));
```

输入加速

```
int read()
{
    char c;
    ll ret=0;
    while((c=getchar())<'0'||c>'9');
    while(c>='0'&&c<='9') ret=ret*10+(c-'0'),c=getchar();
    return ret;
}

int m=read();
```

随机生成数据

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>
#include<math.h>
#include<algorithm>
#include<iostream>
using namespace std;
int main()
{
    freopen("in.txt","w",stdout);
    srand((unsigned)time(NULL));
    for(int i=1;i<=10;i++)
    {
        int n = rand()%50+1;
        int m = rand()%100;
        printf("%d %d\n",n,m);
        for(int j=1;j<=m;j++)
        {
            int k = rand()%n;
            int t = rand()%n;
            printf("%d %d\n",k,t);
        }
        printf("\n");
    }
    return 0;
}
```

手动扩栈

```
#pragma comment(linker, "/STACK:102400000,102400000")
```

计算出二进制数中有多少个 1

```
int get_one(int x)
{
    int sum=0;
    while(x){
        x&=(x-1);
        sum++;
    }
    return sum;
}
```

计算几何

求多边形重心

计算几何中：
三角形的重心： x = (xa+xb+xc)/3, y = (ya+yb+yc)/3;
四边形的重心：作一对角线，将它分成两个三角形分别求出重心与面积 (x1,y1) ,s1 ; (x2, y2), s2 则该四边形的重心为： x = (x1*s1+x2*s2)/(s1+s2), y = (y1*s1+y2*s2)/(s1+s2);
五边形则分为一个三角形与一个四边形……

任意多边形中直接取任一点（一般为原点）把多边形分为 n-2 个三角形 分别求重心

x=∑si*xi/∑si

y=∑si*yi/∑si

si 为每块三角形的有向面积

代码：

```
#include<stdio.h>
#include<string.h>

struct node
{
	double x,y;
}p[100010];
double area(node a,node b,node c)
{
	return (a.x*b.y+b.x*c.y+c.x*a.y-a.y*b.x-b.y*c.x-c.y*a.x)/2;
}
int main()
{
	int n;
	int t;
	scanf("%d",&t);
	while(t--)
	//while(scanf("%d",&n)!=EOF)
	{
		scanf("%d",&n);
		int i,j,k;
		for(i=0;i<n;i++)
			scanf("%lf%lf",&p[i].x,&p[i].y);
		double sum=0,sumx=0,sumy=0,mian;
		for(i=1;i<n-1;i++)
		{
			mian=area(p[0],p[i],p[i+1]);
			sum+=mian;
			sumx+=(p[0].x+p[i].x+p[i+1].x)/3*mian;
			sumy+=(p[0].y+p[i].y+p[i+1].y)/3*mian;
		}

printf("%.2lf %.2lf\n",sumx/sum+0.00000001,sumy/sum+0.00000001);
	}
}
```

求最小面积外接矩阵和最小周长外接矩阵

```
#include <iostream>
#include <cstdio>
#include <cmath>
#include <algorithm>
#include <vector>
using namespace std;
const double eps=1e-6;
const double inf=1e10;
int dcmp(double x)
{
	if(fabs(x)<eps) return 0;
	else return x<0?-1:1;
}
struct point
{
	double x,y;
	point(double x=0,double y=0):x(x),y(y){}
};
point operator-(point a,point b){return point(a.x-b.x,a.y-b.y);}
point operator+(point a,point b){return point(a.x+b.x,a.y+b.y);}
point operator*(point a,double p){return point(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 dcmp(a.x-b.x)==0&&dcmp(a.y-b.y)==0;
}
double cross(point a,point b){return a.x*b.y-a.y*b.x;}
double dot(point a,point b){return a.x*b.x+a.y*b.y;}
double length(point a){return sqrt(dot(a,a));}
point normal(point a)
{
	double l=length(a);
	return point(a.x/l,a.y/l);
}
double distoline(point p,point a,point b)
{
	point v1=b-a,v2=p-a;
	return fabs(cross(v1,v2))/length(v1);
```

```
}
vector<point> p;
double mina,minp;
vector<point> convex(vector<point>& p)
{
	sort(p.begin(),p.end());
	int n=p.size();
	vector<point> ch(n+1);
	int m=0;
	for(int i=0;i<n;i++)
	{
		while(m>1&&cross(ch[m-1]-ch[m-2],p[i]-ch[m-2])<=0) m--;
		ch[m++]=p[i];
	}
	int k=m;
	for(int i=n-2;i>=0;i--)
	{
		while(m>k&&cross(ch[m-1]-ch[m-2],p[i]-ch[m-2])<=0) m--;
		ch[m++]=p[i];
	}
	if(n>1) m--;
	ch.resize(m);
	return ch;
}
void rotating_calipers(vector<point>& points)
{
	vector<point> p=convex(points);
	int n=p.size();
	p.push_back(p[0]);
	mina=inf;minp=inf;
	int l=1,r=1,u=1;
	for(int i=0;i<n;i++)
	{
		point edge=normal(p[(i+1)%n]-p[i]);
		while(dot(edge,p[r%n]-p[i])<dot(edge,p[(r+1)%n]-p[i])) r++;
		while(u<r||cross(edge,p[u%n]-p[i])<cross(edge,p[(u+1)%n]-p[i])) u++;
		while(l<u||dot(edge,p[l%n]-p[i])>dot(edge,p[(l+1)%n]-p[i])) l++;
		double w=dot(edge,p[r%n]-p[i])-dot(edge,p[l%n]-p[i]);
		double h=distoline(p[u%n],p[i],p[(i+1)%n]);
		mina=min(mina,w*h);
		minp=min(minp,2*(w+h));
	}
}
int main()
{
	int n;
	while(~scanf("%d",&n))
	{
		if(!n) break;
		p.clear();
		for(int i=0;i<n;i++)
		{
			double x,y;
			scanf("%lf%lf",&x,&y);
			p.push_back(point(x,y));
		}
		rotating_calipers(p);
		printf("%.2f %.2f\n",mina,minp);
	}
	return 0;
}
```

凸包

```
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
#include <iostream>
#include <stdlib.h>
#include <ctime>
using namespace std;
#define eps 1e-8
#define pi acos(-1.0)
#define N 55

int n,t,d;

int dcmp(double x) {return x < -eps ? -1 : x > eps; }

struct point {
	double x, y;
	point() {}
	point(double _x, double _y) : x(_x), y(_y) {}
	point operator - (const point &p) {return point(x - p.x, y - p.y);}
	point operator + (const point &p) {return point(x + p.x, y + p.y);}
	point operator / (double d){return point(x / d, y / d);}
	point operator * (double d){return point(x * d, y * d);}
	double arc() {return atan2(y, x);}
```

```
    point rotate(double phi) {
        return point(x * cos(phi) - y * sin(phi), x * sin(phi) - y * cos(phi));
    }
    bool operator < (const point& p) const {
        return dcmp(x - p.x) < 0 || (dcmp(x - p.x) == 0 && dcmp(y - p.y) < 0);
    }
    bool operator == (const point& p) {
        return dcmp(x - p.x) == 0 && dcmp(y - p.y) == 0;
    }
    double operator * (const point &p) {return x * p.y - y * p.x;} //2??y
    double operator & (const point &p) {return x * p.x + y * p.y;} //μ??y
    double dis(const point &p) {return sqrt((x - p.x) * (x - p.x) + (y - p.y) * (y - p.y));}
    void input() {scanf("%lf%lf", &x, &y);}
    void output() {printf("%.2lf %.2lf\n", x, y);}
} p[N], q[N];

//边上有点，更改<= 为 < 去掉
int ConvexHull(point p[], int n, point q[]) {
    sort(p, p + n);
    int m = 0;
    for (int i = 0; i < n; i ++ ) {
        while (m > 1 && dcmp((q[m - 1] - q[m - 2]) * (p[i] - q[m - 2])) <= 0) m -- ;
        q[m ++] = p[i];
    }

    if (n < 3) return n;
    int k = m;
    for (int i = n - 2; i >= 0; i -- ) {
        while (m > k && dcmp((q[m - 1] - q[m - 2]) * (p[i] - q[m - 2])) <= 0) m -- ;
        q[m ++ ] = p[i];
    }
    if (n > 1) m -- ;
    return m;
}

int main()
{
    scanf("%d",&t);
    int i,j,k,res=0;
    while(t--)
    {
        res++;
        scanf("%d%d",&n,&d);
        for(i=0;i<n;i++)
            p[i].input();
        int m=ConvexHull(p,n,q);
        double ans=q[0].dis(q[m-1]);
        for(i=1;i<m;i++)
            ans+=q[i].dis(q[i-1]);
        printf("Case #%d: %.4lf\n",res,ans/pi/(double)d);
    }
}
```

动态规划 dp

背包

（1）01 背包：

```
for(i=1;i<=n;i++)
        for(j=V;j>=v[i];j--)
            f[j]=max(f[j],f[j-v[i]]+w[i]);
```

（2）完全背包：

```
for(i=1;i<=n;i++)                   //i：1--n
        for(j=w[i];j<=V;j++) //j：w[i]---V    正序的
            f[j]=min(f[j],f[j-w[i]]+v[i]);
```

（3）多重背包：

用 2 禁止转换，可以将 多重背包 化为 01 背包

例如：有 13 个 125，的背包，13=1+2+4+6. 所以有 125，250，500，625, 4 种背包。。将 13 化为 4 了，省时间

```
    int count=0,t=1;
        for(i=0;i<n;i++){
            t=1;
            while(t<num[i]){
                num[i]-=t;
                p[count++]=v[i]*t;
                t*=2;
            }
            p[count++]=v[i]*(num[i]);
        }
for(i=0;i<count;i++)
        for(j=V;j>=p[i];j--)
            f[j]=max(f[j],f[j-p[i]]+p[i]);
```

（4）分组背包：

```
for(i=1;i<=n;i++)                    //分 1---n 组
    for(j=m;j>=0;j--)                //时间
        for(k=0;k<=j;k++)            //每组分配的时间
            f[j]=max(f[j],f[j-k]+t[i][k]);
```

数位 dp

```
//求个数数位 dp
int n,k,d;
int bit[20];
int dp[20][20];
int dfs(int pos,int num,int flag)
{
    if(num<0) return 0;
    if(pos==-1) return num==0;
    if(flag==0 && dp[pos][num]!=-1) return dp[pos][num];
    int end=flag?bit[pos]:9;
    int i,j,k,ans=0;
    for(i=0;i<=end;i++)
        ans+=dfs(pos-1,(i==d)?(num-1):num,flag&&i==end);
    if(flag==0) dp[pos][num]=ans;
    return ans;
}
int calc(int x)
{
    int i,j,len=0;
    while(x)
    {
        bit[len++]=x%10;
        x/=10;
    }
    return dfs(len-1,k,1);
}
int main()
{
    int i,j;
    while(scanf("%d%d%d",&n,&k,&d)!=EOF)
    {
    memset(dp,-1,sizeof(dp));
        printf("%d\n",calc(n));
    }
}

//求和数位 dp
typedef pair<int, int> PII;
typedef pair<PII, int> PIII;
typedef pair<LL, LL> PLL;
typedef pair<LL, int> PLI;
typedef pair<LD, LD> PDD;
#define MP make_pair
#define PB push_back
#define sz(x) ((int)(x).size())
#define clr(ar,val) memset(ar, val, sizeof(ar))
#define FOR(i,n) for(int i=0;i<(n);++i)
#define forIt(mp,it) for(__typeof(mp.begin()) it = mp.begin();it!=mp.end();it++)
const double PI = acos(-1.0);
#define MOD 1000000007

PLL dp[25][205][2][2];
int digit[25];
LL l,r,k,p10[25];

PLL dfs(int cur,int sum,int x,int lead,int limit){
    if(cur<0){
        if(sum==k) return MP(0,1);
        return MP(0,0);
    }
    if(!limit&&~dp[cur][sum][x][lead].first) return dp[cur][sum][x][lead];
    if(!lead) x = 1;
    LL ans = 0,cnt = 0;
    int ed = limit?digit[cur]:9;
    int p = 1;
    if(x==0) p = -1;
    for(int i = 0;i<=ed;i++){
        PLL ret = dfs(cur-1,sum+i*p,!x,lead||i,limit&&i==ed);

    ans+=i*p10[cur]%MOD*ret.second%MOD+ret.first;
        ans%=MOD;
        cnt+=ret.second;
        cnt%=MOD;
    }
    if(!limit) dp[cur][sum][x][lead] = MP(ans,cnt);
    return MP(ans,cnt);
}

PLL solve(LL n){
    if(n<0) return MP(0,0);
    int cnt = 0;
    while(n){
        digit[cnt++] = n%10;
        n/=10;
```

```
    }
    return dfs(cnt-1,100,1,0,1);
}

int main(void){
#ifndef ONLINE_JUDGE
    //freopen("/Users/mac/Desktop/data.in","r",stdin);
#endif
    p10[0] = 1;
    for(int i = 1;i<=20;i++) p10[i] = 1ll*p10[i-1]*10%MOD;
    memset(dp,-1,sizeof(dp));
    scanf("%lld %lld %lld",&l,&r,&k);
    k+=100;
    LL ret = solve(r).first-solve(l-1).first;
    if(ret<0) ret+=MOD;
    printf("%lld\n",ret);
    return 0;
}
```

博弈

NIM 游戏

结论：异或和为 0，必输；否则赢。
解释：对于某个局面(a1,a2,...,an)，若 a1^a2^...^an!=0，一定存在某个合法的移动，将 ai 改变成 ai'后满足 a1^a2^...^ai'^...^an=0。不妨设 a1^a2^...^an=k，则一定存在某个 ai，它的二进制表示在 k 的最高位上是 1（否则 k 的最高位那个 1 是怎么得到的）。这时 ai^k<ai 一定成立。则我们可以将 ai 改变成 ai'=ai^k，此时 a1^a2^...^ai'^...^an=a1^a2^...^an^k=0。

题：桌子上有 M 堆扑克牌；每堆牌的数量分别为 Ni(i=1…M)；两人轮流进行；每走一步可以任意选择一堆并取走其中的任意张牌；桌子上的扑克全部取光，则游戏结束；最后一次取牌的人为胜者。，，“先手的人如果想赢，第一步有几种选择呢？”

```
int p[110];
int main()
{
    int n;
    while(scanf("%d",&n),n)
    {
        int i,j,k;
        for(i=0;i<n;i++)
            scanf("%d",&p[i]);
        int t=0;
        for(i=0;i<n;i++)
            t^=p[i];
        //printf("%d\n",t);
        int ans=0;
        for(i=0;i<n;i++)
        {
            if((t^p[i])<p[i])
                ans++;
        }
        printf("%d\n",ans);
    }
}
```

SG 函数

题：3 堆石子，每次只能取斐波那契数列中的数。问输赢

```
int s[20];   //斐波那契额数列个数
int g[1010];   //石子数的 sg 函数
int G(int x)
{
    int mex[18]={0};
    if(g[x]!=-1) return g[x];
    if(x-s[0]<0) return g[x]=0;
    for(int i=0;i<20&&(x-s[i]>=0);i++)
    {
        mex[G(x-s[i])]=1;
    }
    for(int i=0;;i++)
        if(mex[i]==0)
            return g[x]=i;
}
int main()
{
    int n,m,p;
    int i,j,k;
    s[0]=s[1]=1;
    for(i=2;i<=16;i++)
        s[i]=s[i-1]+s[i-2];
    memset(g,-1,sizeof(g));
    while(scanf("%d%d%d",&n,&m,&p))
    {
        if(n==0&&m==0&&p==0) break;
        int ans=G(n)^G(m)^G(p);
        if(ans==0) printf("Nacci\n");
```

```
            else printf("Fibo\n");
        }
}

//每次取完，变成两种情况
int sg[200];
int G(int x)
{
    if(sg[x]!=-1) return sg[x];
    if(x==0||x==1) return sg[x]=0;
    int mex[1000]={0};
    for(int i=0;i<=x-2;i++)
    {
        mex[G(i)^G(x-2-i)]=1;
    }
    for(int i=0;;i++)
        if(!mex[i])
            return sg[x]=i;
}
```

一些经典博弈问题

威佐夫博弈：有两堆各若干个物品，两个人轮流从某一堆或同时从两堆中取同样多的物品，规定每次至少取一个，多者不限，最后取光者得胜。
必输：（0，0）、（1，2）、（3，5）、（4，7）、（6，10）、（8，13）、（9，15）、（11，18）、（12，20）……

```
int main()
{
    int n,m;
    double t=(1+sqrt(5.0))/2;
    while(scanf("%d%d",&n,&m)!=EOF)
    {
        if(n>m) swap(n,m);
        int k=m-n;
        if((int)(t*k)==n)
            printf("0\n");
        else
            printf("1\n");
    }
}
```

①、从一堆石子中取走任意多个，②、将一堆数量不少于 2 的石子分成都不为空的两堆。
sg 函数值：1 2 4 3 5 6 8 7 ……

```
int main()
{
    int t;
    scanf("%d",&t);
    while(t--)
    {
        int n,i,j,k;
        scanf("%d",&n);
        int ans=0;
        while(n--)
        {
            scanf("%d",&k);
            j=k%4;
            if(j==3) k++;
            else if(j==0) k--;
            ans^=k;
        }
        if(ans)
            printf("Alice\n");
        else
            printf("Bob\n");
    }
}
```

STL 用法

Bitset

std::bitset 是 STL 的一部分，准确地说，std::bitset 是一个模板类，它的模板参数不是类型，而整形的数值（这一特性是 ISO C++2003 的新特性），有了它我们可以像使用数组一样使用位。下面看一个例子：

```
#include<bitset>
std::bitset<8> bs;//它是一个模板，传递的参数告诉编译器 bs 有 8 个位。
```

我们接着看上面的代码，通过上面两行的代码我们得到一个 bitset 的对象 bs，bs 可以装入 8 个位，我们可以通过数组的下标运算符来存取：

```
bs[0]=1;//把第 0 位设置为 1
bs[3]=true;//把第 3 位设置为 1,因为 true 可以转换为 1
bs[7]=0;//这个大家都明白了
```

bitset 被设计为开放的，也就是说一个 bitset 对象可以转换为其它类型的值，典型的，我们想把一个整数设置成具有特定的位模式，我们可以简单地把一个 bitset 转换为一个整数：

```
unsigned long value=bs.to_ulong();
std::bitset<32> bs32(value);
```

```
bs32[15]=1;
value=bs32.to_ulong();
```
此外 bitset 还可以也字符串互换，这样我们就可以更直观对 bitset 进行操作了，我只是简单地把我们想要的”01“字符串就可以了：
```
std::bitset<32> bs("011010101001");
std::string str=bs.to_string();

//=====================================
bitset<n> b;
```
b 有 n 位，每位都为 0.参数 n 可以为一个表达式.
如 bitset<5> b0;则"b0"为"00000";

```
bitset<n> b(unsigned long u);
```
b 有 n 位,并用 u 赋值;如果 u 超过 n 位,则顶端被截除
如:bitset<5>b0(5);则"b0"为"00101";

```
bitset<n> b(string s);
```
b 是 string 对象 s 中含有的位串的副本
```
string bitval("10011");
bitset<5> b0(bitval4);
```
则"b0"为"10011";

```
bitset<n> b(s, pos);
```
b 是 s 中从位置 pos 开始位的副本,前面的多余位自动填充 0;
```
string bitval("01011010");
bitset<10> b0(bitval5, 3);
```
则"b0" 为 "0000011010";

```
bitset<n> b(s, pos, num);
```
b 是 s 中从位置 pos 开始的 num 个位的副本,如果 num<n,则前面的空位自动填充 0;
```
string bitval("11110011011");
bitset<6> b0(bitval5, 3, 6);
```
则"b0" 为 "100110";

```
// 流
os << b
```
把 b 中的位集输出到 os 流
```
os >>b
```
输入到 b 中,如"cin>>b",如果输入的不是 0 或 1 的字符,只取该字符前面的二进制位.

```
// 属性方法
bool any()
```
是否存在置为 1 的二进制位?和 none()相反

```
bool none()
```
是否不存在置为 1 的二进制位,即全部为 0?和 any()相反.

```
size_t count()
```
二进制位为 1 的个数.

```
size_t size()
```
二进制位的个数

```
flip()
```
把所有二进制位逐位取反

```
flip(size_t pos)
```
把在 pos 处的二进制位取反

```
bool operator[](size_type _Pos)
```
获取在 pos 处的二进制位

```
set()
```
把所有二进制位都置为 1

```
set(pos)
```
把在 pos 处的二进制位置为 1

```
reset()
```
把所有二进制位都置为 0

```
reset(pos)
```
把在 pos 处的二进制位置为 0

```
test(size_t pos)
```
在 pos 处的二进制位是否为 1?

```
unsigned long to_ulong()
```
用同样的二进制位返回一个 unsigned long 值

```
string to_string()
```
返回对应的字符串.

String

```
#include<string>
string s;
s=s+'a'; //在尾部添加字符
s=s+"abc"; //在尾部添加字符串

s[3] //访问
s.length(); //返回字符串的长度
```

printf(s.c_str); //换行

string::iterator it;
it=s.begin();
s.insert(it+i,'a'); //在第 i 个后面插入字符，即‘a’变为第 i+1 个字符
s.erase(it+i); //删除第 i+1 个字符

s.replace(i,j,"good"); //从第 i+1 个开始，连续 j 个字符替换为“good”
1，大于
s.compare("cat");//比较函数，返回值　　0，等于
　　-1，小于
reverse(s.begin(),s.end());//反向排序

vector<string> q;//作为 vector 的元素，类似于字符串数组

优先队列

（1）优先队列：

priority_queue<int> q;　//大到小

priority_queue<int, vector<int>, greater<int> > q;　//小到大

```
struct node
{
	int value,zhi;
	bool operator < (const node &a) const
	{
		return a.value < value;//value 由小到大
	}
};
priority_queue<node> q;
```