ACM/ICPC/CCPC Template

浙江工商大学 ZJSU

ZYF

November 14, 2019

Contents

0 prime

0.1 素数筛选 (埃拉托色尼筛选)

```
/*
 * 埃拉托色尼筛选法
 * 素数筛选，判断小于 MAXN 的数是不是素数。
 * notprime 是一张表，为 false 表示是素数，true 表示不是素数
 */
const int MAXN = 1000010;
bool notprime[MAXN]; // 值为 false 表示素数，值为 true 表示非素数
void init()
{
    memset(notprime, false, sizeof(notprime));
    notprime[0] = notprime[1] = true;
    for (int i = 2; i < MAXN; i++)
        if (!notprime[i])
        {
            if (i > MAXN / i)
                continue; // 防止后面 i*i 溢出 (或者 i,j 用 long long)
            // 直接从 i*i 开始就可以，小于 i 倍的已经筛选过了, 注意是 j += i
            for (int j = i * i; j < MAXN; j += i)
                notprime[j] = true;
        }
}
```

0.2 判断素数

```
bool judge(int x){
    float n_sqrt;
    if(x == 2 || x == 3){
        return true;
    }
    if(x % 6 != 1 && x % 6 != 1){
        return false;
    }
    n_sqrt = floor(sqrt(float(x)));
    for(int i = 5; i <= n_sqrt; i += 6){
        if(x % i == 0 || x % (i + 2) == 0){
            return false;
        }
    }
    return true;
}
```

0.3 米勒罗宾素数

```
//注意需要
//srand(time(NULL));
int modular_exp(int a, int m, int n) {
    if (m == 0) {
        return 1;
    }
    if (m == 1) {
        return (a % n);
    }
    ll w = modular_exp(a, m / 2, n);
    w = w * w % n;
    if (m & 1) {
        w = w * a % n;
    }
    return w;
}

bool Miller_Rabin(int n) {
    if (n == 2) {
        return true;
    }
    for (int i = 0; i < maxn; i++) {
        int a = rand() % (n - 2) + 2;
        if (modular_exp(a, n, n) != a) {
            return false;
        }
    }
    return true;
}
```

0.4 区间素数筛选

```
//对区间[a, b)内的整数执行筛法
//函数返回回区间内素数个数
//is_prime[i - a] = true表示i是素数
1 < a < b <= 1e12, b - a <= 1e6;
const int maxn = "Edit";
bool is_prime_small[maxn], is_prime[maxn];
ll prime[maxn];

int segment_sieve(ll a, ll b){
    int tot = 0;
    for(ll i = 0; i * i < b; i++)   is_prime_small[i] = true;
    for(ll i = 0; i * i < b; i++)   is_prime[i] = true;
    for(ll i = 2; i * i < b; i++){
        if(is_prime[i]){
            for(ll j = 2 * i; j * j < b; j += i)
                is_prime_small[j] = false;
            for(ll j = max(2LL, (a + i - 1) / i) * i; j < b; j += i)
                is_prime[j - a] = false;
        }
    }
    for(ll i = 0; i < b - a; i++){
        if(is_prime[i]) prime[tot++] = i + a;
    }
    return tot;
}
```

1 逆元

1.1 扩展 gcd 求逆元 (模数不一定为质数)

```
LL exgcd(LL a,LL b,LL &x,LL &y)//扩展欧几里得算法
{
    if(b==0)
    {
        x=1,y=0;
        return a;
    }
    LL ret=exgcd(b,a%b,y,x);
    y-=a/b*x;
    return ret;
}
LL getInv(int a,int mod)//求a在mod下的逆元，不存在逆元返回-1
{
    LL x,y;
    LL d=exgcd(a,mod,x,y);
    return d==1?(x%mod+mod)%mod:-1;
}
```

1.2 费马小定理求逆元 (模数需为质数)

```
/*
 * 费马小定理求逆元
 * 仅当 mod 为素数时可用
 * 调用 inv 函数获取逆元
 */

const int mod = 1000000009;

long long quickpow(long long a, long long b)
{
    if (b < 0)
        return 0;
    long long ret = 1;
    a %= mod;
    while (b)
    {
        if (b & 1)
            ret = (ret * a) % mod;
        b >>= 1;
        a = (a * a) % mod;
    }
    return ret;
}
long long inv(long long a)
{
    return quickpow(a, mod - 2);
}
```

2 线性同余方程

2.1 一次线性同余方程组

```
ll x, m, M, r, y, z;
int n;

ll gcd(ll a, ll b) {
    return b == 0 ? a : gcd(b, a % b);
}

void inv(ll a, ll b) {
    if (a % b == 0) {
        z = 0;
        y = 1;
        return;
    }
    inv(b, a % b);
    ll r = z;
    z = y;
    y = r - a / b * y;
}

void solve() {
    x = 0;
    m = 1;
    for (int i = 0; i < n; i++) {
        cin >> M >> r;
        ll b = r - x, d = gcd(m, M);
        if (b % d) {
            cout << "-1" << endl;//不存在
            return;
        }
        inv(m / d, M / d);
        ll t = b / d * z % (M / d);
        x += m * t;
        m *= M / d;
    }
    x = x > 0 ? x : x + m;
    cout << x << endl;
}
```

2.2 二次线性同余方程组

```
//p为模数
struct node{
    ll p, d;
};

ll w;

ll quickMod(ll a, ll b, ll m){
    ll ans = 1;
    a %= m;
    while(b){
        if(b & 1){
            ans = ans * a % m;
        }
        b >>= 1;
        a = a * a % m;
    }
    return ans;
}

//二次域乘法
node multi_er(node a, node b, ll m){
    node ans;
    ans.p = (a.p * b.p % m + a.d * b.d % m * w % m) % m;
    ans.d = (a.p * b.d % m + a.d * b.p % m) % m;
    return ans;
}

//二次域上快速幂
node power(node a, ll b, ll m){
    node ans;
    ans.p = 1;
    ans.d = 0;
    while(b){
        if(b & 1){
            ans = multi_er(ans, a, m);
        }
        b >>= 1;
        a = multi_er(a, a, m);
    }
    return ans;
}

//求勒让德符号
ll legendre(ll a, ll p){
    return quickMod(a, (p - 1) >> 1, p);
}

ll mod(ll a, ll m){
    a %= m;
    return a < 0 ? a + m : a;
}

ll solve(ll n, ll p){
    if(p == 2){
        return 1;
    }
    if(legendre(n, p) + 1 == p){
        return -1;
    }
    ll a = -1, t;
    while(1){
        a = rand() % p;
        t = a * a - n;
        w = mod(t, p);
        if(legendre(w, p) + 1 == p){
            break;
        }
    }

    node tmp;
    tmp.p = a;
    tmp.d = 1;
    node ans = power(tmp, (p + 1) >> 1, p);
    return ans.p;
}

int main(void) {
    int T;
    cin >> T;
    while(T--){
        int n, p;
        cin >> n >> p;
        n %= p;
        int a = solve(n, p);
        if(a == -1){
            cout << "No root" << endl;
            continue;
        }
        int b = p - a;
        if(a > n){
            swap(a, b);
        }
        if(a == b){
            cout << a << endl;
        }else{
            cout << a << ' ' << b << endl;
        }
    }
}
```

2.3 EXCRT

```
//x mod m[i] = r[i]; m[i] 可以两两不互质
//引用返回通解x = re + k * mo;函数返回是否有解

bool excrt(ll r[], ll m[], ll n, ll &re, ll &mo){
    ll x, y;
    mo = m[0], re = r[0];
    for(int i = 1; i < n; i++){
        ll d = exgcd(mo, m[i], x, y);
        if((r[i] - re) % d != 0)    return 0;
        x = (r[i] - re) / d * x % (m[i] / d);
        re += x * mo;
        mo = mo / d * m[i];
        re %= mo;
    }
    re = (re + mo) %mo;
    return 1;
}
```

2.4 CRT

```
//x mod m[i] = r[i];要求m[i]两两互质
//引用返回通解x = re + k * mo;
void crt(ll r[], ll m[], ll n, ll &re, ll &mo){
    mo = 1, re = 0;
    for(int i = 0; i < n; i++)  mo *= m[i];
    for(int i = 0; i < n; i++){
        ll x, y, tm = mo / m[i];
        ll d = exgcd(tm, m[i], x, y);
        re = (re + tm * x *r[i]) % mo;
    }
    re = (re + mo) % mo;
}
```

3 BSGS

3.1 bsgs

```
void bsgs(ll y,ll z,ll p)//y^x=z(mod p) gcd(y, p) = 1
{
   if(y==0 && z==0){puts("1");return ;}//几句特判
   if(y==0 && z!=0){return;}//不存在

   mp.clear();
   ll m=ceil(sqrt(p));
   ll tmp=z%p;mp[tmp]=0;//右边z*A^j，当j=0时为z
   for(ll i=1;i<=m;i++)
   {
      tmp=tmp*y%p;
      mp[tmp]=i;
   }

   ll t=power(y,m,p);
   tmp=1;//左边y^(i*m)，当i=0时为1
   for(ll i=1;i<=m;i++)
   {
      tmp=tmp*t%p;//i每加1，多乘y^(i*m)
      if(mp[tmp])
      {
         ll ans=i*m-mp[tmp];
         printf("%lld\n",(ans%p+p)%p);
         return ;
      }
   }
   return;//不存在
}
```

3.2 exbsgs

```
//解决x ^ y mod z == k的最小y

struct Hash {
   int i;
   ll xi;
   Hash(int a, ll b) : i(a), xi(b) {}
};

vector<Hash> has[maxn];

ll powMod(ll a, ll b, ll mod) {
   ll res = 1;
   while (b) {
      if (b & 1) {
         res = res * a % mod;
      }
      b >>= 1;
      a = a * a % mod;
   }
   return res;
}

ll gcd(ll a, ll b) {
    return b == 0 ? a : gcd(b, a % b);
}

ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    ll res = exgcd(b, a % b, x, y);
    ll t = x;
    x = y;
    y = t - a / b * y;
    return res;
}

ll BSGS(ll x, ll z, ll p) {
    z %= p;
    ll val = 1;
    for (int i = 0; i <= 100; i++, val = (val * x) % p) {
        if (val == z) {
            return i;
        }
    }
    ll q = 1, cnt = 0;
    while ((val = gcd(x, p)) != 1) {
        if (z % val) {
            return -1;
        }
        p /= val;
        z /= val;
        q = q * x / val % p;
        cnt++;
    }

    ll m = (ll) sqrt((double) p);
    for (int i = 0; i < maxn; i++) {
        has[i].clear();
    }
    val = 1;
    for (int i = 0; i <= m; i++) {
        int vv = val % maxn;
        has[vv].push_back(Hash(i, val));
        val = val * x % p;
    }

    ll xm = powMod(x, m, p), a, b;
    for (int i = 0; i <= m; i++) {
        exgcd(q, p, a, b);
        val = ((z * a) % p + p) % p;
        ll vv = val % maxn;
        for (int j = 0; j < has[vv].size(); j++) {
            if (has[vv][j].xi == val) {
                return i * m + has[vv][j].i + cnt;
            }
        }
        q = q * xm % p;
    }
    return -1;
```

```
}
```

4 欧拉函数

4.1 欧拉函数

```
//O(logn)的时间求一个数的phi
ll euler(ll n){
    ll rt = n;
    for(int i = 2; i * i <= n; i++){
        if(n % i == 0){
            rt -= rt / i;
            while(n % i == 0){
                n /= i;
            }
        }
    }
    if(n > 1){
        rt -= rt / n;
    }
    return rt;
}

//给出一个N，求[1,n]中与N互质的数的和就是这个公式: n*phi[n]/2

//用一个数组求欧拉值
ll phi[maxn];
void ZyfPhi() {
    phi[1] = 0;
    for(int i = 2; i < maxn; i++){
        phi[i] = i;
    }
    for(int i = 2; i < maxn; i++){
        if(phi[i] == i){
            for(int j = i; j < maxn; j += i){
                phi[j] = phi[j] / i * (i - 1);
            }
        }
    }
    for (int i = 2; i < maxn; i++) {
        phi[i] = phi[i] + phi[i - 1];
    }
}

//O(n)得到欧拉函数phi[], 素数表prime[], 素数个数tot
bool vis[maxn];
int tot, phi[maxn], prime[maxn];
void Phi(){
    phi[1] = 0;
    for(int i = 2; i < maxn; i++){
        if(!vis[i]){
            prime[tot++] = i, phi[i] = i - 1;
            for(int j = 0; j < tot; j++){
                if(i * prime[j] > maxn){
                    break;
                }
                vis[i * prime[j]] = 1;
                if(i % prime[j] == 0){
                    phi[i * prime[j]] = phi[i] * prime[j];
                    break;
                }else{
                    phi[i * prime[j]] = phi[i] * (prime[j] - 1);
                }
            }
        }
    }
}
```

5 组合数学

5.1 lucas

```
//求解组合数取模p，其中p为质数
//求解逆元
ll powMod(ll a, ll b, ll p){
    ll ret = 1;
    while(b){
        if(b & 1){
            ret = ret * a % mod;
        }
        b >>= 1;
        a = a * a % p;
    }
    return ret;
}

//求组合数，因为分解得较小了，所以可以用暴力
ll C(ll n, ll m, ll p){
    if(m > n){
        return 0;
    }
    ll c1 = 1, c2 = 1;
    for(int i = n + m - 1; i <= n; i++){
        c1 = c1 * i % p;
    }
    for(int i = 2; i <= m; i++){
        c2 = c2 * i % p;
    }

    return c1 * powMod(c2, p - 2, p) % p;
}

ll lucas(ll n, ll m, ll p){
    if(!m)  return 1;
    return C(n % p, m %p, p) * lucas(n / p, m / p, p) * p;
}
```

5.2 exlucas

```
//求解组合数求模p，且p不一定是质数
ll c[1000006], a[1000005];
ll powMod(ll a, ll b, ll p){
    ll ret = 1;
    while(b){
        if(b & 1)   ret =ret * a % p;
        b >>= 1
        a = a * a % p;
    }
    return ret;
}

//求阶乘
ll fac(ll n, ll p, ll pk){
    if(!n)  return 1;
    ll ans = 1;
    for(int i = 1; i < pk; i++){
        if(i % p)  ans = ans * i % pk;//同余部分
    }
    ans = powMod(ans, n / pk, pk);
    for(int i = 1; i <= n % pk; i++){//剩余无法凑同余的部分
        if(i % p)  ans = ans * i % pk;
    }
    return ans * fac(n / p, p, pk) % pk;
}

ll exgcd(ll a, ll b, ll &x, ll &y){
    if(!b){
        x = 1, y = 0;
        return a;
    }
    ll xx, yy, g = exgcd(b, a % b, xx, yy);
    x = yy;
    y = xx - a / b * yy;
    return g;
}

ll inv(ll a, ll b){ //求逆元
    ll x, y;
    exgcd(a, p, x, y);
    return (x % p + p) % p;
}

//求组合数
ll C(ll n, ll m, ll p, ll pk){
    if(m > n)  return 0;
    ll f1 = fac(n, p, pk), f2 = fac(m, p, pk), f3 = fac(n - m, p, pk), cnt = 0;
    for(ll i = n; i; i /= p){
        cnt += i / p;
    }
    for(ll i = m; i; i /= p){
        cnt -= i / p;
    }
    for(ll i = n - m; i; i /= p){
        cnt -= i / p;
    }
    return f1 * inv(f2, pk) % pk * inv(f3, pk) % pk *powMod(p, cnt, pk) % pk;
}

ll CRT(ll cnt){
    ll M = 1, ans = 0;
    for(int i = 1; i <= cnt; i++){
        M *= c[i]; //p的值发生变化，所以要重新计算
    }
    for(int i = 1; i <= cnt; i++){
        ans = (ans + a[i] * (M / c[i]) % M * inv(M / c[i], c[i]) % M) % M;
    }
    return ans;
}

ll exlucas(ll n, ll m, ll p){
    ll temp, cnt = 0;
    for(int i = 2; p > 1 && i <= p / i; i++){
        ll tmp = 1;
        while(p % i == 0){
            p /= i, tmp *= i;
        }
        if(tmp > 1){
            a[++cnt] = C(n, m, i, tmp);
            c[cnt] = tmp;
        }
    }
    if(p > 1){
        c[++cnt] = p, a[cnt] = C(n, m, p, p);
    }
    return CRT(cnt);
}

int main(){
    ll m, n, p;
    cin >> n >> m >> p;
    cout << exlucas(n, m, p);
}
```

5.3 Big Combination

```
// 0 <= n <= 1e9, 0 <= m <= 1e4, 1 <= k <= 1e9 + 7
//利用逆元求解
vector<int> v;
int dp[110];
ll Cal(int l, int r, int k, int dis){
    ll res = 1;
    for(int i = l; i <= r; i++){
        int t = i;
        for(int j = 0; j < v.size(); j++){
            int y = v[j];
            while(t % y == 0) dp[j] += dis, t /= y;
        }
        res = res * (ll) t % k;
    }
    return res;
}

ll Comb(int n, int m, int k){
    memset(dp, 0, sizeof(dp));
    v.clear();
    int tmp = k;
    for(int i = 2; i * i <= tmp; i++){
        if(tmp % i == 0){
            int num = 0;
            while(tmp % i == 0) tmp /= i, num++;
            v.push_back(i);
        }
    }
    if(tmp != 1)   v.push_back(tmp);
    ll ans = Cal(n - m + 1, n, k, 1);
    for(int j = 0; j < v.size(); j++)ans = ans * powMod(v[j], dp[j], k) % k;
    ans = ans * inv(Cal(2, m, k, -1), k) % k; //inv是求逆元函数
    return ans;
}
```

5.4 组合数学初始化（杨辉三角）

```
// 0 <= m <= n <= 1000

const int maxn = 1010;
ll C[maxn][maxn];
void CalComb(){
    C[0][0] = 1;
    for(int i = 1; i < maxn; i++){
        C[i][0] = 1;
        for(int j = 1; j <= i; j++){
            C[i][j] = (C[i - 1][j - 1] + C[i - 1][j]) % mod;
        }
    }
}
```

5.5 组合数学初始化 (阶乘的逆元)

```
// 0 <= m <= n <= 1e5, 模p为质数

const int maxn = 100010;
ll f[maxn];
ll inv[maxn]; //阶乘的逆元
void CalFact(){
    f[0] = 1;
    for(int i = 1; i < maxn; i++){
        f[i] = (f[i - 1] * i) % p;
    }
    inv[maxn - 1] = powMod(f[maxn - 1], p - 2, p);
    for(int i = maxn - 2; ~i; i--){
        inv[i] = inv[i +1] * (i + 1) % p;
    }
}

ll C(int n, int m){
    return f[n] * inv[m] % p * inv[n - m] % p;
}
```

5.6 第一斯特林数

```
//s(n, m) = s(n - 1, m - 1) + (n - 1) * s(n - 1, m)
//给定正整数n(1<=n<=20),计算出n个元素的集合{1,2, □ ,n} 可以化为多少个不同的非空子集。
typedef long long ll;
const int N = 101;
ll s[N][N];

void init(){
    memset(s, 0, sizeof(s));
    s[1][1] = 1;
    for(int i = 2; i < N; i++){
        for(int j = 1; j <= i; j ++){
            s[i][j] = (s[i - 1][j - 1] + (i - 1) * s[i - 1][j]);
        }
    }
}

int main(void){
    init();
    int T;
    cin >> T;
    while(T--){
        int n, k;
        cin >> n >> k;
        ll sum = 1;
        for(ll i = 2; i <= n; i++){
            sum *= i;
        }
        ll fz = 0;
        for(int i = 1; i <= k; i++){
            fz += (s[n][i] - s[n-1][i-1]);
        }
        double ans = fz * 1.0 / sum;
        printf("%.4lf\n", ans);
    }
}
```

5.7 卡特兰数

```
int a[105][100];

void catalan(){
    a[2][0] = 1;
    a[2][1] = 2;
    a[1][0] = 1;
    a[1][1] = 1;
    int len = 1, yu;
    for(int i = 3; i < 101; i++){
        yu = 0;
        for(int j = 1; j <= len; j++){
            int t = (a[i-1][j]) * (4 * i - 2) + yu;
            yu = t/10;
            a[i][j] = t % 10;
        }
        while(yu){
            a[i][++len] = yu % 10;
            yu /= 10;
        }
        for(int j = len; j >= 1; j--){
            int t = a[i][j] + yu * 10;
            a[i][j] = t / (i + 1);
            yu = t % (i + 1);
        }
        while(!a[i][len]){
            len--;
        }
        a[i][0] = len;
    }
}

int main(void){
    catalan();
    int n;
    while(cin >> n){
        if(n == -1){
            break;
        }
        for(int i = a[n][0]; i > 0; i--){
            cout << a[n][i];
        }
        cout << endl;
    }
}
//解决的问题
//1.Cn可以表示长度为2n的Dyck Words的种类数，Dyck Words由n个A字符与n个B字符组成，且满足在任意位置上，前缀中A的数量不小于B的数量，如果A用(代替，把B用)代替，就是一个典型的括号表达式，因此也可以用来表示合法的表达式个数。
//2.Cn可以表示有n个结点的不同构的二叉树的种类数。
//3.Cn可以表示有2n + 1个结点的不同构的满二叉树的种类数。
//4.Cn可以表示在n*n的格点中从左下角延格线走到右上角且始终不超过对角线的方案数。
//5.Cn可以表示通过连接顶点将n + 2个顶点的凸多边形划分成三角形的方案数。
//6.Cn表示有n个元素的出栈顺序的种类数。
//7.Cn可以用在买票找零钱问题上，对于2n的观众，收银台初始没有钱，无法找零，接下来有n个人拿a元买票，n个人拿2a元买票（需要找零钱数为a），能够保证每一个需要找零的观众来的时候都能够得到找零的合法方案数。
//8.Cn可以表示在二维直角坐标系中，从(0, 0)走到(2n, 0)点，每个相邻整数点的纵坐标差值的绝对值为1，且点始终不会落到x轴下方的方案数。
```

6 异或

6.1 线性基

```
//所谓线性基，就是线性代数里面的概念。一组线性无关的向量便可以作为一组基底，张起一个线性的向量空间，这个基地又称之为线性基。这个线性基的基底进行线性运算，可以表示向量空间内的所有向量，也即所有向量可以拆成基底的线性组合。
//这篇用了前缀和能快速计算多次询问中的不同区间的最大异或和
const int maxn = (int) 5e5 + 100;
int p[maxn][31], pos[maxn][31];
int n, q;
int lastans = 0;

typedef long long ll;

void push_back(int x, int i) {
    for (int j = 0; j <= 30; j++) {
        p[i][j] = p[i - 1][j];
        pos[i][j] = pos[i - 1][j];
    }
    int ti = i;
    for (int j = 30; j >= 0; j--) {
        if (x & (1 << j)) {
            if (!p[i][j]) {
                p[i][j] = x;
                pos[i][j] = ti;
                break;
            }
            if (pos[i][j] < ti) {
                swap(p[i][j], x);
                swap(pos[i][j], ti);
            }
            x ^= p[i][j];
        }
    }
}

int main(void) {
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    int T;
    cin >> T;
    while (T--) {
        lastans = 0;
//      memset(p, 0, sizeof(p));
//      memset(pos, 0, sizeof(pos));
        cin >> n >> q;
        int x;
        for (int i = 1; i <= n; i++) {
            cin >> x;
            push_back(x, i);
        }
        while (q--) {
            int e, l, r;
            cin >> e;
            if (e == 1) {
                cin >> l;
                l ^= lastans;
                push_back(l, n + 1);
                n++;
            } else {
                cin >> l >> r;
                l = (l ^ lastans) % n + 1;
                r = (r ^ lastans) % n + 1;
                if (l > r) {
                    swap(l, r);
                }
                int ret = 0;
                for (int i = 30; i >= 0; i--) {
                    if (pos[r][i] >= l && (ret ^ p[r][i]) > ret) {
                        ret ^= p[r][i];
                    }
                }
                cout << ret << endl;
                lastans = ret;
            }
        }
    }
}
```

7 矩阵

7.1 斐波那切数列十进制

```cpp
typedef long long ll;
const int maxn = (int) 1e6 + 100;
char str[maxn];
int num[maxn];
ll mod;

struct mat {
    ll m[3][3];

    mat() {
        m[1][1] = m[1][2] = m[2][1] = m[2][2] = 0;
    }

    mat friend operator*(mat a, mat b) {
        mat res;
        for (int k = 1; k <= 2; k++) {
            for (int i = 1; i <= 2; i++) {
                for (int j = 1; j <= 2; j++) {
                    res.m[i][j] += a.m[i][k] * b.m[k][j];
                }
            }
        }

        for (int i = 1; i <= 2; i++) {
            for (int j = 1; j <= 2; j++) {
                res.m[i][j] %= mod;
            }
        }
        return res;
    }

    mat friend operator^(mat a, int b) {
        mat res;
        res.m[1][1] = res.m[2][2] = 1LL;
        while (b) {
            if (b & 1) {
                res = res * a;
            }
            a = a * a;
            b >>= 1;
        }
        return res;
    }
};

int main(void) {
#ifdef ACM_LOCAL
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
#endif
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);

    int a, b, x1, x2;
    cin >> x1 >> x2 >> a >> b;
    cin >> str >> mod;;

    int len = strlen(str);
    for (int i = 0; i < len; i++) {
        num[i] = str[i] - '0';
    }

    num[len - 1]--;
    for (int i = len - 1; i >= 0 && num[i] < 0; i--) {
        num[i] += 10;
        num[i - 1]--;
    }
    mat base, A, ans;
    ans.m[1][1] = ans.m[2][2] = 1;
    base.m[1][1] = a % mod;
    base.m[1][2] = b % mod;
    base.m[2][1] = 1LL;
    A.m[1][1] = x2 % mod;
    A.m[2][1] = x1 % mod;
    for (int i = len - 1; i >= 0; i--) {
        if (num[i]) {
            ans = ans * (base ^ num[i]);
        }
        base = base ^ 10;
    }
    ans = ans * A;
    cout << ans.m[1][1] % mod << endl;
}
```

8 others

8.1 ax+by=c exgcd

```
//引用返回通解：X = x + k * dx, Y = y - k * dy;
//引用返回的x是最小非负整数解，方程无解函数返回0

#define Mod(a, b) (((a) % (b) + (b)) % (b))

ll exgcd(ll a, ll b, ll &x, ll &y){
   ll d = a;
   if(b){
      d = exgcd(b, a % b, y, x);
      y -= x * (a / b);
   }else{
      x = 1;
      y = 0;
   }
   return d;
}

bool solve(ll a, ll b, ll c, ll &x, ll &y, ll &dx, ll &dy){
   if(a == 0 & b == 0){
      return 0;
   }
   ll x0, y0;
   ll d = exgcd(a, b, x0, y0);
   if(c % d != 0)  return 0;
   dx = b / d, dy = a / d;
   x = Mod(x0 * c / d, dx);
   y = (c - a * x) / b;
   return 1;
}
```

8.2 formula

8.3 万进制

```
#include <iomanip>
void factorial(int n)
{
   int a[10001];
   int places, carry, i, j;

   a[0] = 1;
   places = 0; //当前数的总位数
   for (i = 1; i <= n; i++)
   {
      carry = 0;
      for (j = 0; j <= places; j++)
      {
         a[j] = a[j] * i + carry; //如果是多次幂函数，将i改成数字即可
         carry = a[j] / 10000;
         a[j] %= 10000;
      }
      if (carry > 0)
```

```
        {
            places++;
            a[places] = carry;
        }
    }
    /*
     * 输出
     * 最高位原样输出
     * 其他位小于1000的，高位补0
     * 需要头文件<iomanip>
     */
    cout << a[places];
    for (i = places - 1; i >= 0; i--)
    {
        cout << setw(4) << setfill('0') << a[i];
    }
    cout << endl;
}
```

8.4 定积分

```
double F(double x)
{
    //Simpson公式用到的函数
}
double simpson(double a, double b)//三点Simpson法，这里要求F是一个全局函数
{
    double c = a + (b - a) / 2;
    return (F(a) + 4 * F(c) + F(b))*(b - a) / 6;
}
double asr(double a, double b, double eps, double A)//自适应Simpson公式（递归过程）。已知整个区间[a,b]上的三点Simpson值A
{
    double c = a + (b - a) / 2;
    double L = simpson(a, c), R = simpson(c, b);
    if (fabs(L + R - A) <= 15 * eps)return L + R + (L + R - A) / 15.0;
    return asr(a, c, eps / 2, L) + asr(c, b, eps / 2, R);
}
double asr(double a, double b, double eps)//自适应Simpson公式（主过程）
{
    return asr(a, b, eps, simpson(a, b));
}
```

8.5 用树状数组求逆序对

```
//用树状数组的方法求逆序对
ll num[maxn], b[maxn], c[maxn]; //b用于正序的中间数组，c用于反序的中间数组
ll ans;
int n;
int lowbit(int x){
    return x & (-x);
}

void add(ll a[], ll pos, ll val){
    while(pos <= maxn){
        a[pos] += val;
        pos += lowbit(pos);
    }
}

ll sum(ll a[], ll pos){//求逆序对的个数
    ll tmp = 0;
    while(pos > 0){
        tmp += a[pos];
        pos -= lowbit(pos);
    }
    return tmp;
}

int main(void) {
    while(cin >> n){
        memset(c, 0, sizeof(ll) * (n +5));
        memset(b, 0, sizeof(ll) * (n + 5));
        ans = 0;
        //这是求逆序对的和，若求逆序对的个数只要一个for循环就可
        for(int i = 1; i <= n; i++){
            cin >> num[i];
            add(c, num[i], 1);
            //逆序对的个数
            //ans += sum(c, num[i]);
            ans += num[i] * (i - sum(c, num[i]));
        }
        for(int i = n; i >= 1; i--){
            ans += num[i] * (sum(b, num[i]));
            add(b, num[i], 1);
        }
        cout << ans << endl;
    }
}
```

8.6 线性基

```
const int MAXN = 31; //如果为 long long 则改成 63 即可
int d[MAXN + 5];
void init()
{
    memset(d, 0, sizeof(d));
}
void add(int x)
{
    for (int i = MAXN; i >= 0; i--)
    {
        if (x & (1LL << i))
        {
            if (d[i])
                x ^= d[i];
            else
            {
                d[i] = x;
                break;
            }
        }
    }
}
// 如何求异或后第k小的值
```

```
void work() //处理线性基
{
    for (int i = 1; i <= 60; i++)
        for (int j = 1; j <= i; j++)
            if (d[i] & (1 << (j - 1)))
                d[i] ^= d[j - 1];
}
ll k_th(ll k)
{
    if (k == 1 && tot < n)
        return 0; //特判一下，假如k=1，并且原来的序列可以异或出0，就要返回0，tot表示线性基中的元素个数，n表示序列长度
    if (tot < n)
        k--; //类似上面，去掉0的情况，因为线性基中只能异或出不为0的解
    work();
    ll ans = 0;
    for (int i = 0; i <= 60; i++)
        if (d[i] != 0)
        {
            if (k % 2 == 1)
                ans ^= d[i];
            k /= 2;
        }
}
```

8.7 大数板子 (易)

```
int flag = 1;
//初始化
void initial(string &a, string &b){
    while (a.size()<b.size())a = '0' + a;
    while (b.size()<a.size())b = '0' + b;
}
//打印
void print(string &a, string &b){
    cout << a << endl;
    cout << b << endl;
}
//找出最大的字符串
void findMax(string &a, string &b){
    string tmp;
    if (a<b){
        tmp = b;
        b = a;
        a = tmp;
    }
}
//删除第一个字符'0'
bool del(string &a){
    if (a[0] == '0'){
        a.erase(0, 1);
        return true;
    }
    else
        return false;
}
//删除前面所有的 0
void delAllZroe(string &a){
    while (del(a)){
        del(a);
    };
}
//大数加法
string bigItergeAdd(string a, string b){
    initial(a, b);
    a = '0' + a;
    b = '0' + b;
    for (int i = a.size() - 1; i >= 0; i--){
        int num1 = a[i] - '0';
        int num2 = b[i] - '0';
        if (num1 + num2>9){
            a[i - 1] = a[i - 1] - '0' + 1 + '0';
            a[i] = (num1 + num2) - 10 + '0';
        }
        else{
            a[i] = (num1 + num2) + '0';
        }
    }
    del(a);
    // cout<<a<<endl;
    return a;
}
//大数减法
string bigItergeSub(string a, string b){
    initial(a, b);
    string tmp = a;
    findMax(a, b);
    if(a != tmp){
        flag = -1;
    }
    for (int i = a.size() - 1; i >= 0; i--){
        int num1 = a[i] - '0';
        int num2 = b[i] - '0';
        if (num1<num2){
            a[i - 1] = a[i - 1] - '0' - 1 + '0';
            a[i] = (num1 + 10 - num2) + '0';
        }
        else{
            a[i] = (num1 - num2) + '0';
        }
    }
    del(a);
    // cout<<a<<endl;
    return a;
}
//大数乘法(大数加法实现)
void bigItergeMul(string a, string b){
    delAllZroe(a);
    delAllZroe(b);
    if (a == "" || b == ""){
        printf("0\n"); return;
    }
    initial(a, b);
    findMax(a, b);
    string res = "0";
    int count = 0;
    delAllZroe(b);
    for (int i = b.size() - 1; i >= 0; i--){
        int num1 = b[i] - '0';
        if (i != b.size() - 1)      a = a + '0';
        for (int j = 1; j <= num1; j++){
            res = bigItergeAdd(res, a);
        }
    }
    delAllZroe(res);
    cout << res << endl;
}
//大数除法
void bigItergeDiv(string a, string b){
    initial(a, b);
    if (a<b){ cout << "0" << endl;  return; }
    delAllZroe(b);
    string res = "0";
    string restmp = "1";
    string tmp = b;
    for (int i = 1; i<(a.size() - b.size()); i++){
        tmp += '0';
        restmp += '0';
    }
    initial(a, b);
    while (a >= b){
        initial(a, tmp);
        if (a >= tmp){
            a = bigItergeSub(a, tmp);
            res = bigItergeAdd(res, restmp);
        }
        else{
            tmp.erase(tmp.size() - 1);
            restmp.erase(restmp.size() - 1);
            initial(a, tmp);
            if (a >= tmp){
                a = bigItergeSub(a, tmp);
                res=bigItergeAdd(res, restmp);
            }
        }
        initial(a, b);
    }
    cout << res << endl;
}

int main(void) {
    string a, b;
    while (cin >> a >> b){
        string ans;
        //ans = bigItergeAdd(a,b);
        ans = bigItergeSub(a,b);
//      bigItergeMul(a,b);
//      bigItergeDiv(a,b);
        if(flag == -1){
            cout << '-';
        }
        cout << ans << endl;
    }
}
```

8.8 大数板子 (完整)

```
const int maxn = 10005;/*精度位数,自行调整*/
//1.如果需要控制输出位数的话，在str()里面把len调成需要的位数
//2.很大的位数是会re的，所以如果是幂运算的话，如 计算x^p的位数n, n=p*log(10)x+1;(注意要加一）
//3.还可以加上qmul，取模的过程也就是str()，c_str()再搞一次
class bign {
  //io*2 bign*5*2 bool*6
  friend istream &operator>>(istream &, bign &);

  friend ostream &operator<<(ostream &, const bign &);

  friend bign operator+(const bign &, const bign &);

  friend bign operator+(const bign &, int &);

  friend bign operator*(const bign &, const bign &);

  friend bign operator*(const bign &, int &);

  friend bign operator-(const bign &, const bign &);

  friend bign operator-(const bign &, int &);

  friend bign operator/(const bign &, const bign &);

  friend bign operator/(const bign &, int &);

  friend bign operator%(const bign &, const bign &);

  friend bign operator%(const bign &, int &);

  friend bool operator<(const bign &, const bign &);

  friend bool operator>(const bign &, const bign &);

  friend bool operator<=(const bign &, const bign &);

  friend bool operator>=(const bign &, const bign &);

  friend bool operator==(const bign &, const bign &);

  friend bool operator!=(const bign &, const bign &);

private://如果想访问len,改成public
  int len, s[maxn];
public:
  bign() {
    memset(s, 0, sizeof(s));
    len = 1;
  }

  bign operator=(const char *num) {
    int i = 0, ol;
    ol = len = strlen(num);
    while (num[i++] == '0' && len > 1)
      len--;
    memset(s, 0, sizeof(s));
    for (i = 0; i < len; i++)
```

```
            s[i] = num[ol - i - 1] - '0';
        return *this;
    }

    bign operator=(int num) {
        char s[maxn];
        sprintf(s, "%d", num);
        *this = s;
        return *this;
    }

    bign(int num) {
        *this = num;
    }

    bign(const char *num) {
        *this = num;
    }

    string str() const {
        string res = "";
        for (int i = 0; i < len; i++)
            res = char(s[i] + '0') + res;
        if (res == "")
            res = "0";
        return res;
    }
};

bool operator<(const bign &a, const bign &b) {
    int i;
    if (a.len != b.len)
        return a.len < b.len;
    for (i = a.len - 1; i >= 0; i--)
        if (a.s[i] != b.s[i])
            return a.s[i] < b.s[i];
    return false;
}

bool operator>(const bign &a, const bign &b) {
    return b < a;
}

bool operator<=(const bign &a, const bign &b) {
    return !(a > b);
}

bool operator>=(const bign &a, const bign &b) {
    return !(a < b);
}

bool operator!=(const bign &a, const bign &b) {
    return a < b || a > b;
}

bool operator==(const bign &a, const bign &b) {
    return !(a < b || a > b);
}

```

```
bign operator+(const bign &a, const bign &b) {
    int up = max(a.len, b.len);
    bign sum;
    sum.len = 0;
    for (int i = 0, t = 0; t || i < up; i++) {
        if (i < a.len)
            t += a.s[i];
        if (i < b.len)
            t += b.s[i];
        sum.s[sum.len++] = t % 10;
        t /= 10;
    }
    return sum;
}

bign operator+(const bign &a, int &b) {
    bign c = b;
    return a + c;
}

bign operator*(const bign &a, const bign &b) {
    bign res;
    for (int i = 0; i < a.len; i++) {
        for (int j = 0; j < b.len; j++) {
            res.s[i + j] += (a.s[i] * b.s[j]);
            res.s[i + j + 1] += res.s[i + j] / 10;
            res.s[i + j] %= 10;
        }
    }
    res.len = a.len + b.len;
    while (res.s[res.len - 1] == 0 && res.len > 1)
        res.len--;
    if (res.s[res.len])
        res.len++;
    return res;
}

bign operator*(const bign &a, int &b) {
    bign c = b;
    return a * c;
}

//只支持大数减小数
bign operator-(const bign &a, const bign &b) {
    bign res;
    int len = a.len;
    for (int i = 0; i < len; i++) {
        res.s[i] += a.s[i] - b.s[i];
        if (res.s[i] < 0) {
            res.s[i] += 10;
            res.s[i + 1]--;
        }
    }
    while (res.s[len - 1] == 0 && len > 1)
        len--;
    res.len = len;
    return res;
}

```

```
bign operator-(const bign &a, int &b) {
  bign c = b;
  return a - c;
}

bign operator/(const bign &a, const bign &b) {
  int i, len = a.len;
  bign res, f;
  for (i = len - 1; i >= 0; i--) {
    f = f * 10;
    f.s[0] = a.s[i];
    while (f >= b) {
      f = f - b;
      res.s[i]++;
    }
  }
  while (res.s[len - 1] == 0 && len > 1)
    len--;
  res.len = len;
  return res;
}

bign operator/(const bign &a, int &b) {
  bign c = b;
  return a / c;
}

bign operator%(const bign &a, const bign &b) {
  int len = a.len;
  bign f;
  for (int i = len - 1; i >= 0; i--) {
    f = f * 10;
    f.s[0] = a.s[i];
    while (f >= b)
      f = f - b;
  }
  return f;
}

bign operator%(const bign &a, int &b) {
  bign c = b;
  return a % c;
}

bign &operator+=(bign &a, const bign &b) {
  a = a + b;
  return a;
}

bign &operator-=(bign &a, const bign &b) {
  a = a - b;
  return a;
}

bign &operator*=(bign &a, const bign &b) {
  a = a * b;
  return a;
}

```

```
bign &operator/=(bign &a, const bign &b) {
  a = a / b;
  return a;
}

bign &operator++(bign &a) {
  a = a + 1;
  return a;
}

bign &operator++(bign &a, int) {
  bign t = a;
  a = a + 1;
  return t;
}

bign &operator--(bign &a) {
  a = a - 1;
  return a;
}

bign &operator--(bign &a, int) {
  bign t = a;
  a = a - 1;
  return t;
}

istream &operator>>(istream &in, bign &x) {
  string s;
  in >> s;
  x = s.c_str();
  return in;
}

ostream &operator<<(ostream &out, const bign &x) {
  out << x.str();
  return out;
}

int main(void) {
  bign a;
  bign b;
  cin >> a >> b;
  cout << a / b << endl;
}
```

8.9 快速乘

```
inline ll ksc(ll x, ll y, ll p){   //p是mod
  ll z = (long double) x / p * y;
  ll res = (unsigned long long)x * y - (unsigned long long)z * p;
  return (res + p) % p;
}
```

8.10 母函数

```
#define myfor(a,b,c) for(int i=a;a<=b?i<=b:i>=b;a<=b?++i:--i)
```

```
const int number=3;
int main()
{
  int sum;//sum是指数
  int c1[33005], c2[33005];
  while(scanf("%d", &sum), sum)
  {
    for(int i = 0; i <= sum; ++i)
    {
      c1[i] = 1;//初始化为第一个括号各项的系数，之后再依次与后边的合并更新
      c2[i] = 0;
    }
    //复杂度O(number*sum*sum),也可通过打表后O(1)访问
    for(int i = 2; i <= number; ++i)// 共有number个大括号相乘 ，直接从第二个括号开始合并，并且第i个括号内是以x^i为公比的等比数列，若无此数列则删除此i即可
    {
      for(int j = 0; j <= sum; ++j)// 每次都合并到第一个括号中，这里 j 代表第一个括号中的各项系数
      {
        for(int k = 0; k+j <= sum; k += i) //虽然括号之间是相乘关系，但是指数之间是相加关系
        {
          c2[k+j] += c1[j]; // c2 数组可以理解为每次存放的中间结果,因为每次都是后边的括号与第一个括号可并，而后边的括号系数都为一，所以只有第一个括号中的系数对合并后相应的系数有贡献
        }
      }
      for(int j = 0; j <= sum; ++j)
      {
        c1[j] = c2[j];
        c2[j] = 0;      // 记得每次合并一个括号后要把  c2 清零
      }
    }
    printf("%d\n", c1[sum]);
  }
  return 0;
}
```

8.11 合数分解

```
//素数筛选
const int MAXN=100000;
int prime[MAXN+1];//得到小于等于MAXN的所有素数
void getPrime()
{
  memset(prime,0,sizeof(prime));
  for(int i=2;i<=MAXN;i++)
  {
    if(!prime[i])prime[++prime[0]]=i;
    for(int j=1;j<=prime[0]&&prime[j]*i<=MAXN;j++)     //除法改为乘法提速， 改为除法防止爆范围
    {
      prime[prime[j]*i]=1;
      if(i%prime[j]==0)break;
    }
  }
}
//合数分解（前面需要先素数筛选）
long long factor[100][2];//factor[fatCnt][0]记录目前最小整除数,factor[fatCnt][1]记录 □ factor[fatCnt][0] □ 个数
int fatCnt;//fatCnt的值表示整除数种类
int getFactors(long long x)
{
```

```cpp
    fatCnt=0;
    long long tmp=x;
    for(int i=1;prime[i]*prime[i]<=tmp;i++)            //除法改为乘法提速，改为除法防止爆范围
    {
        factor[fatCnt][1]=0;
        if(tmp%prime[i]==0)
        {
            factor[fatCnt][0]=prime[i];
            while(tmp%prime[i]==0)
            {
                factor[fatCnt][1]++;
                tmp/=prime[i];
            }
            fatCnt++;
        }
    }
    if(tmp!=1)//最后如果tmp不为1表示还有一个未取到的素数约数。
    {
        factor[fatCnt][0]=tmp;
        factor[fatCnt++][1]=1;
    }
    return fatCnt;
}
```

9 未学会

9.1 莫比乌斯反演

```
//F(n) =∑d|n f(d) □ f(n) =∑d|n μ(d)F(n d
//F(n) =∑n|d f(d) □ f(n) =∑n|d μ( d n)F(d)

const int maxn = "Edit";
int prime[maxn], tot, mu[maxn];
bool check[maxn];
void CalMu(){
    mu[1] = 1;
    for(int i = 2; i < maxn; i++){
        if(!check[i])   prime[tot++] = i, mu[i] = -1;
        for(int j = 0; j < tot; j++){
            if(i * prime[j] >= maxn)    break;
            check[i * prime[j]] = true;
            if(i % prime[j] == 0){
                mu[i * prime] = 0;
                break;
            }else{
                mu[i * prime[j]] = - mu[i];
            }
        }
    }
}

//Examples
//有n个数(n < 100000, 1 < ai < 1e6)，问这n个数中互质的数的对数

const int maxn = "Edit";
int b[maxn];

ll solve(int n){
    ll ans = 0;
    for(int i = 0, x; i < n; i++){
        cin >> x;
        b[x]++;
    }
    for(int i = 1; i < maxn; i++){
        int cnt = 0;
        for(int j = i; j < maxn; j += i){
            cnt += b[j];
        }
        ans += 1LL * mu[i] * cnt * cnt;
    }
    return (ans - b[1]) / 2;
}

//gcd(x, y) == 1的对数, x <= n, y <= m;

ll solve(int n, int m){
    if(n > m)   swap(n, m);
    ll ans = 0;
    for(int i = 1; i <= n; i++){
        ans += (ll)mu[i] * (n / i) * (m / i);
    }
    return ans;
}
```

10 几何

10.1 fcy

```
#include<bits/stdc++.h>
#define mp make_pair
#define rep(i,a,b) for(int i=a;i<=b;i++)
using namespace std;
typedef long long ll;
const double inf=1e200;
const double eps=1e-12;
const double pi=acos(-1.0);
const int maxn=1000010;
struct point{
  double x,y;
  point(){}
  point(double xx,double yy):x(xx),y(yy){}
};
struct line{
  point a;//起点
  point p;//起点到终点的向量
  double angle;
};
struct Circle{
  point c; double r;
};
int dcmp(double x){ return fabs(x)<eps?0:(x<0?-1:1);}
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);}
point operator /(point A,double p){ return point(A.x/p,A.y/p);}
point rotate(point A,double rad){ //向量的旋转
  return point(A.x*cos(rad)-A.y*sin(rad), A.x*sin(rad)+A.y*cos(rad));
}
bool operator ==(const point& a,const point& b) {
   return dcmp(a.x-b.x)==0&&dcmp(a.y-b.y)==0;
}
double dot(point A,point B){ return A.x*B.x+A.y*B.y;}
double det(point A,point B){ return A.x*B.y-A.y*B.x;}
double dot(point O,point A,point B){ return dot(A-O,B-O);}
double det(point O,point A,point B){ return det(A-O,B-O);}
double length(point A){ return sqrt(dot(A,A));}
double dist(point A,point B){ return length(A-B);}
double angle(point A,point B){ return acos(dot(A,B)/length(A)/length(B));} //夹角
point jiaopoint(point p,point v,point q,point w)
{  //p+tv q+tw，点加向量表示直线，求直线交点
  //如果是线段，还应该实现判定是否相离； 必要时判是否平行
  point u=p-q;
  double t=det(w,u)/det(v,w);//如果平行，除0会有问题
  return p+v*t;
}
point llintersect(line A,line B) //直线交点，同上
{
  point C=A.a-B.a;
  double t=det(C,B.p)/det(B.p,A.p);
  return A.a+A.p*t;
}
point GetCirPoint(point a,point b,point c)
{
  point p=(a+b)/2;   //ab中点
  point q=(a+c)/2;   //ac中点
  point v=rotate(b-a,pi/2.0),w=rotate(c-a,pi/2.0);  //中垂线的方向向量
  if (dcmp(det(v,w))==0)   //平行
  {
      if(dcmp(length(a-b)+length(b-c)-length(a-c))==0) return (a+c)/2;
      if(dcmp(length(b-a)+length(a-c)-length(b-c))==0) return (b+c)/2;
      if(dcmp(length(a-c)+length(c-b)-length(a-b))==0) return (a+b)/2;
  }
  return jiaopoint(p,v,q,w);
}
point w[maxn];
void MinCir(int N) //增量法，玄学复杂度O(N)；如果精度要求不高，可以三分套三分。
{
  point C=w[1]; double r=0;
  rep(i,1,N){
      if(dist(C,w[i])>r) {
          C=w[i]; r=0;
          rep(j,1,i-1) {
              if(dist(C,w[j])>r) {
                  C=(w[i]+w[j])/2; r=dist(w[i],w[j])/2;
                  rep(k,1,j-1) {
                      if(dist(C,w[k])>r){
                          C=GetCirPoint(w[i],w[j],w[k]);
                          r=dist(C,w[i]);
                      }
                  }
              }
          }
      }
  }
  printf("%.2lf %.2lf %.2lf\n",C.x,C.y,r);
}
double area(vector<point>p){ //求面积
  double ans=0; int sz=p.size();
  for(int i=1;i<sz-1;i++) ans+=det(p[i]-p[0],p[i+1]-p[0]);
  return ans/2.0;
}
double seg(point O,point A,point B){ //返回比例
  if(dcmp(B.x-A.x)==0) return (O.y-A.y)/(B.y-A.y);
  return (O.x-A.x)/(B.x-A.x);
}
vector<point>pp[110];
pair<double,int>s[110*60];
double polyunion(vector<point>*p,int N){ //需要这些点是顺时针，N个多边形。
  double res=0;
  for(int i=0;i<N;i++){
      int sz=p[i].size();
      for(int j=0;j<sz;j++){
          int m=0;
          s[m++]=mp(0,0);
          s[m++]=mp(1,0);
          point a=p[i][j],b=p[i][(j+1)%sz];
          for(int k=0;k<N;k++){
              if(i!=k){
                  int sz2=p[k].size();
                  for(int ii=0;ii<sz2;ii++){
                      point c=p[k][ii],d=p[k][(ii+1)%sz2];
                      int c1=dcmp(det(b-a,c-a));
            int c2=dcmp(det(b-a,d-a));
            if(c1==0&&c2==0){
              if(dcmp(dot(b-a,d-c))){
                s[m++]=mp(seg(c,a,b),1);
                s[m++]=mp(seg(c,a,b),-1);
              }
            }
            else{
              double s1=det(d-c,a-c);
              double s2=det(d-c,b-c);
              if(c1>=0&&c2<0) s[m++]=mp(s1/(s1-s2),1);
              else if(c1<0&&c2>=0) s[m++]=mp(s1/(s1-s2),-1);
            }
          }
        }
      }
      sort(s,s+m);
      double pre=min(max(s[0].first,0.0),1.0),now,sum=0;
      int cov=s[0].second;
      for(int j=1;j<m;j++){
        now=min(max(s[j].first,0.0),1.0);
        if(!cov) sum+=now-pre;
        cov+=s[j].second;
        pre=now;
      }
      res+=det(a,b)*sum;
    }
  }
  return res/2;
}
void CirinterCir(point a,double r1,point b,double r2) //求圆和圆的交点
{
  //记得提前判断是否相交，如果相交就利用角度就可以求了
  double L=dist(a,b);
  if(L>r1+r2) return ;//相离
  if(L+r1<r2||L+r2<r1) return ;//包含
  double t=acos((r1*r1+L*L-r2*r2)/(2.0*r1*L));
  double base=atan2(b.y-a.y,b.x-a.x);  //atan2的范围是(-pi,pi]，这个很多时候用起来更直观，但是缺点是慢
  double ang1=base+t,ang2=base-t;
  point A=a+point{r1*cos(ang1),r1*sin(ang1)};
  point B=a+point{r1*cos(ang2),r1*sin(ang2)};
}
bool cmp1(point a,point b){ return a.x==b.x?a.y<b.y:a.x<b.x; } //x排序
void convexhull(point *a,int n,point *ch,int &top) //凸包
{
  sort(a+1,a+n+1,cmp1);//x排序
  top=0;
  for(int i=1;i<=n;i++){ //下凸包
    while(top>=2&&det(ch[top-1],ch[top],a[i])<=0) top--;
    ch[++top]=a[i];
  }
  int ttop=top;
  for(int i=n-1;i>=1;i--){ //上凸包
    while(top>ttop&&det(ch[top-1],ch[top],a[i])<=0) top--;
    ch[++top]=a[i];
  }
}
double rotating_calipers(point p[],int top) //求最远的点距离
{
  double ans=0; int now=2;
  rep(i,1,top-1){ //先求出凸包，然后凸包上旋转卡壳
    while(det(p[i],p[i+1],p[now])<det(p[i],p[i+1],p[now+1])){
      now++; //最远距离对应了最大面积。
      if(now==top) now=1;
    }
    ans=max(ans,dist(p[now],p[i]));
  }
  return ans;
}
point ch[2000000],p[2000000];
double getangle(point a){ return atan2(a.y,a.x);}
double getangle(line a){ return getangle(a.p);}
point ss[maxn]; line t[maxn],q[maxn]; int head,tail;
bool cmp2(line a,line b){ //方向的极角排序
  double A=getangle(a),B=getangle(b);
  point t=(b.a+b.p)-a.a;
  if(fabs(A-B)<eps) return det(a.p,t)>=0.0;
  return A<B;
}
bool onright(line P,line a,line b)
{
  point o=llintersect(a,b);
  point Q=o-P.a;
  return det(Q,P.p)>0; //如果同一直线上不能相互看到，则>=0
}
bool halfplaneintersect(int N)
{
  ss[N+1]=ss[1];
  rep(i,1,N) t[i].a=ss[i], t[i].p=ss[i+1]-ss[i];
  sort(t+1,t+N+1,cmp2);
  int tot=0;
  rep(i,1,N-1) {
    if(fabs(getangle(t[i])-getangle(t[i+1]))>eps)
     t[++tot]=t[i];
  }
  t[++tot]=t[N]; head=tail=0;
  rep(i,1,tot){
    while(tail>head+1&&onright(t[i],q[tail],q[tail-1])) tail--;
    while(tail>head+1&&onright(t[i],q[head+1],q[head+2])) head++;
    q[++tail]=t[i];
  }
  while(tail>head+1&&onright(t[head+1],q[tail],q[tail-1])) tail--;return tail-head>2;
}
double TriAngleCircleInsection(Circle C, point A, point B) //圆与多边形面积交
{
  //a[N+1]=a[1]; ans=0;。 拆成多个三角形，求矢量面积核
  //rep(i,1,N) ans+=TriAngleCircleInsection(C,a[i],a[i+1]);
  point OA=A-C.c,OB=B-C.c;
  point BA=A-B, BC=C.c-B;
  point AB=B-A, AC=C.c-A;
  double DOA=length(OA),DOB=length(OB),DAB=length(AB),r=C.r;
  if(dcmp(det(OA,OB))==0) return 0; // ，三点一线，不构成三角形
  if(dcmp(DOA-C.r)<0&&dcmp(DOB-C.r)<0) return det(OA,OB)*0.5; //内部
  else if(DOB<r&&DOA>=r) //一内一外
  {
    double x=(dot(BA,BC)+sqrt(r*r*DAB*DAB-det(BA,BC)*det(BA,BC)))/DAB;
    double TS=det(OA,OB)*0.5;
    return asin(TS*(1-x/DAB)*2/r/DOA)*r*r*0.5+TS*x/DAB;
  }
  else if(DOB>=r&&DOA<r)// 一外一内
  {
    double y=(dot(AB,AC)+sqrt(r*r*DAB*DAB-det(AB,AC)*det(AB,AC)))/DAB;
    double TS=det(OA,OB)*0.5;
    return asin(TS*(1-y/DAB)*2/r/DOB)*r*r*0.5+TS*y/DAB;
  }
  else if(fabs(det(OA,OB))>=r*DAB||dot(AB,AC)<=0||dot(BA,BC)<=0)//弧
  {
    if(dot(OA,OB)<0){
      if(det(OA,OB)<0) return (-acos(-1.0)-asin(det(OA,OB)/DOA/DOB))*r*r*0.5;
      else  return ( acos(-1.0)-asin(det(OA,OB)/DOA/DOB))*r*r*0.5;
    }
    else     return asin(det(OA,OB)/DOA/DOB)*r*r*0.5; //小于90度，以为asin对应的区间是[-90度,90度]
  }
  else //弧+三角形
  {
    double x=(dot(BA,BC)+sqrt(r*r*DAB*DAB-det(BA,BC)*det(BA,BC)))/DAB;
    double y=(dot(AB,AC)+sqrt(r*r*DAB*DAB-det(AB,AC)*det(AB,AC)))/DAB;
    double TS=det(OA,OB)*0.5;
    return (asin(TS*(1-x/DAB)*2/r/DOA)+asin(TS*(1-y/DAB)*2/r/DOB))*r*r*0.5 + TS*((x+y)/DAB-1);
  }
}
double ltoseg(point p,point a,point b){
  point t=p-a;
  if(dot(t,b-a)<=0) return dist(p,a);
  else if(dot(p-b,a-b)<=0) return dist(p,b);
  return fabs(det(t,b-a))/dist(a,b);
}
bool isinside(point a) //O(N)判定是否在（任意多边形）内
{
//算法描述：首先，对于多边形的水平边不做考虑，其次，
//对于多边形的顶点和射线相交的情况，如果该顶点时其所属的边上纵坐标较大的顶点，则计数，否则忽略该点，
//最后，对于Q在多边形上的情形，直接判断Q是否属于多边形。
  int ncross=0; int N;
  rep(i,0,N-1) {
    point p1=p[i],p2=p[i+1];
    if(ltoseg(a,p[i],p[i+1])==0) return true; //在线段上
    if(p1.y==p2.y) continue; //默认做水平x轴的线，所以水平线不考虑
    if(a.y<min(p1.y,p2.y)) continue; //相离不考虑
    if(a.y>max(p1.y,p2.y)) continue;
    double t=det(a-p[i],a-p[i+1]);
    if((t>=0&&p[i].y<a.y&&p[i+1].y>=a.y)||(t<=0&&p[i+1].y<a.y&&p[i].y>=a.y)) ncross++;
  }
  return (ncross&1);
}
bool check(point A,int top) //二分点在（凸多边形）内
{
  int L=2,R=top-2,Mid;
  while(L<=R){
    Mid=(L+R)>>1;
    if(det(ch[Mid]-ch[1],A-ch[1])<0) R=Mid-1;
    else {
      if(det(ch[Mid+1]-ch[1],A-ch[1])<=0&&det(ch[Mid+1]-ch[Mid],A-ch[Mid])>=0)
        return true;
      L=Mid+1;
    }
  }
  return false;
```

```
}
int main()
{

}
```

10.2 圆

```
struct point
{
  double x, y;
  point();
  point(double _x, double _y);
  bool operator<(Point b) const;      // 点左右判断 - line 中使用
  bool operator==(point b) const;     // 点相等判断
  point operator+(const point &b) const;  // 向量相加
  point operator-(const point &b) const;  // 向量相减
  point operator*(const double &k) const; // 向量乘法
  point operator/(const double &k) const; // 向量除法
  point trunc(double r) const;        // 向量模转换
  double operator*(const point &b) const; // 向量点积
  double operator^(const Point &b) const; // 向量叉积
  double len(bool isSqrt = true);      // 向量模长度，参数为是否开平方
  double distance(const point &other);  // 点距离
  point rotleft();                // 向量绕原点逆时针旋转 90 度
  point rotright();               // 向量绕原点顺时针旋转 90 度
  point rotate(point p, double angle);  // 绕 p 点逆时针旋转 angle 度
  double rad(point a, point b);        // 计算点 this、a、b 组成的角的角度，角的两条射线为 this-a、this-b
};

struct line
{
  point s, e;
  line();
  line(point _s, point _e);
  bool operator==(line v);          // 判断射线重合
  line(point p, double angle);      // 根据一个点和倾斜角 angle 确定直线,0 <= angle < pi
  line(double a, double b, double c); // 根据 ax + by + c = 0 确定直线
  void adjust();                 // 调整线段
  double lenth();                // 求线段长度
  double angle();                // 返回线段的倾斜角
  int relation(point p);           // 点和直线关系
  bool pointonseg(point p);         // 点在线段上的判断
  bool parallel(line v);           // 两向量平行 (对应直线平行或重合)
  int segcrossseg(line v);          // 两线段相交判断
  int linecrossseg(line v);         // 直线和线段相交判断
  int linecrossline(line v);        // 两直线关系
  point crosspoint(line v);         // 求两直线的交点
  double dispointtoline(point p);    // 点到直线的距离
  double dispointtoseg(point p);     // 点到线段的距离
  double dissegtoseg(line v);       // 返回线段到线段的距离
  Point lineprog(Point p);          // 返回点 p 在直线上的投影
  Point symmetrypoint(Point p);      // 返回点 p 关于直线的对称点
}

const double eps = 1e-8;
const double pi = acos(−1.0);

```

```
int sgn(double x)
{
    if (fabs(x) < eps)
        return 0;
    if (x < 0)
        return -1;
    return 1;
}

struct circle
{
    point p;  // 圆心
    double r; // 半径
    circle() {}
    circle(point _p, double _r) : p(_p), r(_r) {}
    // 三点求算外接圆、内切圆
    // 需要 point 的 + / rotate() 以及 line 的 crosspoint()
    // bool 变量表示是否为外接圆
    circle(point a, point b, point c, bool isCircumcircle)
    {
        if (isCircumcircle)
        {
            line u = line((a + b) / 2, ((a + b) / 2) + ((b - a).rotleft()));
            line v = line((b + c) / 2, ((a + c) / 2) + ((c - b).rotleft()));
            p = u.crosspoint(v);
            r = p.distance(a);
        }
        else
        {
            line u, v;
            double m = atan2(b.y - a.y, b.x - a.x), n = atan2(c.y - a.y.c.x - a.x);
            u.s = a;
            u.e = u.s + point(cos((n + m) / 2), sin((n + m) / 2));
            v.s = b;
            m = atan2(a.y - b.y, a.x - b.x);
            n = atan2(c.y - b.y, c.x - b.x);
            p = u.crosspoint(v);
            r = line(a, b).dispointtoseg(p);
        }
    }
    bool operator==(circle v)
    {
        return (p == v.p) && sgn(r - v.r) == 0;
    }
    bool operator<(circle v) const
    {
        return ((p < v.p) || ((p == v.p) && sgn(r - v.r) < 0));
    }
    // 面积
    double area()
    {
        return pi * r * r;
    }
    // 周长
    double circumference()
    {
        return 2 * pi * r;
    }
    // 点圆关系
// 0 圆外
// 1 圆上
// 2 圆内
int relation(point b)
{
  double dst = b.distance(p);
  if (sgn(dst - r) < 0)
    return 2;
  else if (sgn(dst - r) == 0)
    return 1;
  return 0;
}
// 线段和圆的关系
// // 需要 line 的 dispointtoseg
int relationseg(line v)
{
  double dst = v.dispointtoseg(p);
  if (sgn(dst - r) < 0)
    return 2;
  else if (sgn(dst - r) == 0)
    return 1;
  return 0;
}
// 直线和圆的关系
// // 需要 line 的 dispointtoline
int relationline(line v)
{
  double dst = v.dispointtoline(p);
  if (sgn(dst - r) < 0)
    return 2;
  else if (sgn(dst - r) == 0)
    return 1;
  return 0;
}
// 两圆的关系
// 5 相离
// 4 外切
// 3 相交
// 2 内切
// 1 内含
// 需要 point 的 distance
int relationcircle(circle v)
{
  double d = p.distance(v.p);
  if (sgn(d - r - v.r) > 0)
    return 5;
  if (sgn(d - r - v.r) == 0)
    return 4;
  double l = fabs(r - v.r);
  if (sgn(d - r - v.r) < 0 && sgn(d - l) > 0)
    return 3;
  if (sgn(d - l) == 0)
    return 2;
  if (sgn(d - l) < 0)
    return 1;
}
// 求两个圆的交点，返回 0 表示没有交点，返回 1 是一个交点，2 是两个交点
// 需要 relationcircle
int pointcrosscircle(circle v, point &p1, point &p2)
```

```
{
    int rel = relationcircle(v);
    if (rel == 1 || rel == 5)
        return 0;
    double d = p.distance(v.p);
    double l = (d * d + r * r−v.r * v.r) / (2 * d);
    double h = sqrt(r * r - l * l);
    point tmp = p + (v.p - p).trunc(l);
    p1 = tmp + ((v.p - p).rotleft().trunc(h));
    p2 = tmp + ((v.p - p).rotright().trunc(h));
    if (rel == 2 || rel == 4)
        return 1;
    return 2;
}
// 求直线和圆的交点，返回交点个数
int pointcrossline(line v, point &p1, point &p2)
{
    if (!(*this).relationline(v))
        return 0;
    point a = v.lineprog(p);
    double d = v.dispointtoline(p);
    d = sqrt(r * r - d * d);
    if (sgn(d) == 0)
    {
        p1 = a;
        p2 = a;
        return 1;
    }
    p1 = a + (v.e - v.s).trunc(d);
    p2 = a - (v.e - v.s).trunc(d);
    return 2;
}
// 得到过 a,b 两点，半径为 r1 的两个圆
// 需要 pointcrosscircle
int gercircle(point a, point b, double r1, circle &c1, circle &c2)
{
    circle x(a, r1), y(b, r1);
    int t = x.pointcrosscircle(y, c1.p, c2.p);
    if (!t)
        return 0;
    c1.r = c2.r = r;
    return t;
}
// 得到与直线 u 相切，过点 q, 半径为 r1 的圆
int getcircle(line u, point q, double r1, circle &c1, circle &c2)
{
    double dis = u.dispointtoline(q);
    if (sgn(dis - r1 * 2) > 0)
        return 0;
    if (sgn(dis) == 0)
    {
        c1.p = q + ((u.e - u.s).rotleft().trunc(r1));
        c2.p = q + ((u.e - u.s).rotright().trunc(r1));
        c1.r = c2.r = r1;
        return 2;
    }
    line u1 = line((u.s + (u.e - u.s).rotleft().trunc(r1)), (u.e + (u.e - u.s).rotleft().trunc(r1)));
    line u2 = line((u.s + (u.e - u.s).rotright().trunc(r1)), (u.e + (u.e - u.s).rotright().trunc(r1)));
    circle cc = circle(q, r1);
```

```
        point p1, p2;
        if (!cc.pointcrossline(u1, p1, p2))
            cc.pointcrossline(u2, p1, p2);
        c1 = circle(p1, r1);
        if (p1 == p2)
        {
            c2 = c1;
            return 1;
        }
        c2 = circle(p2, r1);
        return 2;
    }
    // 同时与直线u,v相切，半径为r1的圆
    int getcircle(line u, line v, double r1, circle &c1, circle &c2, circle &c3, circle &c4)
    {
        if (u.parallel(v))
            return 0;
        line u1 = line(u.s + (u.e - u.s).rotleft().trunc(r1), u.e + (u.e - u.s).rotleft().trunc(r1));
        line u2 = line(u.s + (u.e - u.s).rotright().trunc(r1), u.e + (u.e - u.s).rotright().trunc(r1));
        line u3 = line(v.s + (v.e - v.s).rotleft().trunc(r1), v.e + (v.e - v.s).rotleft().trunc(r1));
        line u4 = line(v.s + (v.e - v.s).rotright().trunc(r1), v.e + (v.e - v.s).rotright().trunc(r1));
        c1.r = c2.r = c3.r = c4.r = r1;
        c1.p = u1.crosspoint(v1);
        c2.p = u1.pointonseg(v2);
        c3.p = u2.pointonseg(v1);
        c4.p = u2.pointonseg(v2);
        return 4;
    }
    // 同时与不相交圆cx,cy相切，半径为r1的圆
    int getcircle(circle cx, circle cy, double r1, circle &c1, circle &c2)
    {
        circle x(cx.p, r1 + cx.r), y(cy.p, r1 + cy.r);
        int t = x.pointcrosscircle(y, c1.p, c2.p);
        if (!t)
            return 0;
        c1.r = c2.r = r1;
        return t;
    }
    int tangentline(point q, line &u, line &v)
    {
        int x = relation(q);
        if (x == 2)
            return 0;
        if (x == 1)
        {
            u = line(q, q + (q - p).rotleft());
            v = u;
            return 1;
        }
        double d = p.distance(q);
        double l = r * r / d;
        double h = sqrt(r * r - l * l);
        u = line(q, p + ((q - p).trunc(l) + (q - p).rotleft().trunc(h)));
        v = line(q, p + ((q - p).trunc(l) + (q - p).rotright().trunc(h)));
        return 2;
    }
    // 求两圆相交的面积
    double areacircle(circle v)
    {
        int rel = relationcircle(v);
        if (rel >= 4)
            return 0.0;
        if (rel <= 2)
            return min(area(), v.area());
        double d = p.distance(v.p);
        double hf = (r + v.r + d) / 2.0;
        double ss = 2 * sqrt(hf * (hf - r) * (hf - v.f) * (hf - d));
        double a1 = acos((r * r + d * d - v.r * v.r) / (2.0 * r * d));
        a1 = a1 * r * r;
        double a2 = acos((v.r * v.r + d * d - r * r) / (2.0 * v.r * d));
        a2 = a2 * v.r * v.r;
        return a1 + a2 - ss;
    }
    // 求圆和三角形 pab 的相交面积
    double areatriangle(Point a, Point b)
    {
        if (sgn((p - a) ^ (p - b)) == 0)
            return 0.0;
        point q[5];
        int len = 0;
        q[len++] = a;
        line l(a, b);
        point p1, p2;
        if (pointcrossline(l, q[1], q[2]) == 2)
        {
            if (sgn((a - q[1]) * (b - q[i])) < 0)
                q[len++] = q[1];
            if (sgn((a - q[2]) * (b - q[2])) < 0)
                q[len++] = q[2];
        }
        q[len++] = b;
        if (len == 4 && sgn((q[0] - q[1]) * (q[2] - q[1])) > 0)
            swap(q[1], q[2]);
        double res = 0;
        for (int i = 0; i < len - 1; i++)
        {
            if (relation(q[i]) == 0 || relation(q[i + 1]) == 0)
            {
                double arg = p.rad(q[i], q[i + 1]);
                res += r * r * arg / 2.0;
            }
            else
                res += fabs((q[i] - p) ^ (q[i + 1] - p)) / 2.0;
        }
        return res;
    }
};
```

10.3 体积

```
#include <iostream>
#include <cmath>

using namespace std;

const double pi = acos(-1.0);

```

```
//圆锥体体积公式 V = 1 / 3 * S * h, S是底面积, h是高
double volumn_Cone(double r, double h){
    return 1.0 / 3 * r * r * pi * h;
}

//三棱锥体积公式
//已知空间内三角形三顶点坐标A(a1, a2, a3), B(b1, b2, b3), C(c1, c2, c3).
//O为原点，则三棱锥O-ABC体积为
//V = 1.0 / 6 * abs(a1 * b2 * c3 + b1 * c2 * a3 + c1 * a2 * b3 - a1 * c2 * b3 - b1 * a2 * c3 - c1 * b2 * a3);
struct Point{
    double x;
    double y;
    double z;
};

double volumn_a(Point a, Point b, Point c){
    return 1.0 / 6 * abs(a.x * b.y * c.z + b.x * c.y * a.z + c.x * z.y * b.z - a.x * c.y * b.z - b.x * a.y * c.z - c.x * b.y * a.z);
}

//椭球在xyz-笛卡尔坐标系中的标准方程是:(x - x0) ^ 2 / a ^ 2 + (y - y0) ^ 2 / b ^ 2 + (z - z0) ^ 2 / c ^ 2 = 1
//体积是V = 4 / 3 * pi * a * b * c;
double volumn_Ellipse(double a, double b, double c){
    return 4.0 / 3 * pi * a * b * c;
}

//圆台的体积
double volumn_RoundTable(double R, double r, double h){
    return pi * h / 3.0 * (R * R + r * r + R * r);
}

//球缺
double volumn_MissingBall(double h, double r){
    return pi * h * h * (r - h / 3.0);
}

//交叉圆柱体的体积
double volumn_CrossCylinder(double h1, double h2, double r){
    return pi * r * r * (h1 + h2 - 2.0 / 3 * r);
}

//梯形体的体积
double volumn_TrapezoidalBody(double a, double b, double h1, double a1, double b1){
    return h / 6 * ((2 * a + a1) * b + (2 * a1 + a) * b1);
    //或者return h / 6 * (a * b + (a + a1) * (b + b1) + a1 * b1);
}
```

10.4 二维凸包

```
//计算凸包，输入点数组p,个数为p，输出点数组为ch。函数返回凸包顶点数
//输入不能有重复节点
//如果精度要求搞需要用dcmp判断
//如果不希望在边上右点，需要将 <= 改为 <
int ConvexHull(Point *p,int n ,Point *ch)
{
    sort(p,p+n);
    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--;
    return m;
}
```

10.5 二维几何模板

```
#include <bits/stdc++.h>
#define mem(ar,num) memset(ar,num,sizeof(ar))
#define me(ar) memset(ar,0,sizeof(ar))
#define lowbit(x) (x&(-x))
#define forn(i,n) for(int i = 0;i < n; ++i)
using namespace std;
typedef long long LL;
typedef unsigned long long ULL;
const int    prime = 999983;
const int    INF = 0x7FFFFFFF;
const LL     INFF =0x7FFFFFFFFFFFFFFF;
const double pi = acos(-1.0);
const double inf = 1e18;
const double eps = 1e-10;
const LL     mod = 1e9 + 7;
struct Point
{
    double x,y;

    Point(double x = 0,double y = 0):x(x),y(y) {}

};
typedef Point Vector;
Vector operator + (Vector A,Vector B)
{
    return Vector(A.x + B.x,A.y + B.y);
}
Vector operator - (Vector A,Vector B)
{
    return Vector(A.x-B.x,A.y-B.y);
}
Vector operator / (Vector A,double p)
{
    return Vector(A.x/p,A.y/p);
}
Vector operator * (Vector A,double p)
{
    return Vector(A.x*p,A.y*p);
}
double angle(Vector v)//求向量的角度从0到2*pi
{
    return atan2(v.y,v.x);
```

```
}
int dcmp(double x)
{
    if(fabs(x)<eps)
        return 0;
    else
        return x < 0?-1:1;
}
bool operator < (const Point &a,const Point &b)
{
    if(dcmp(a.x-b.x)==0)
        return a.y<b.y;
    else
        return a.x<b.x;
}


bool operator == (const Point &a,const Point &b)
{
    return !dcmp(a.x-b.x)&&!dcmp(a.y-b.y);
}
double Dot(Vector A,Vector B)
{
    return A.x*B.x+A.y*B.y;
}
double Length(Vector A)
{
    return sqrt(A.x*A.x+A.y*A.y);
}
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);
}
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);
}
//调用前确保直线有唯一交点，当且仅当Cross(v,w)非0
Point Get_Line_Intersection(Point P,Vector v,Point Q,Vector w)
{
    Vector u = P - Q;
    double t = Cross(w,u)/Cross(v,w);
    return P+v*t;
}
double Distance_To_Line(Point P,Point A,Point B)//点到直线的距离
{
    Vector v1 = B-A,v2 = P-A;
    return fabs(Cross(v1,v2)/Length(v1));
}
double Distance_To_Segment(Point P,Point A,Point B)
{
    if(A==B)
      return Length(P-A);
    Vector v1 = B-A,v2 = P-A,v3 = P-B;
    if(dcmp(Dot(v1,v2))<0)
      return Length(v1);
    else if(dcmp(Dot(v1,v3))>0)
      return Length(v3);
    else
      return fabs(Cross(v1,v2))/Length(v1);
}
Point Get_Line_Projection(Point P,Point A,Point B)//求投影点
{
    Vector v = B- A;
    return A + v*(Dot(v,P-A)/Dot(v,v));
}
//线段相交判定 相交不在线段的端点
bool Segment_Proper_Intersection(Point a1,Point a2,Point b1,Point b2)
{
    double c1 =  Cross(a2-a1,b1-a1),c2 = Cross(a2-a1,b2-a1),
        c3 =  Cross(b2-b1,a2-b1),c4 = Cross(b2-b1,a1-b1);
    return dcmp(c1)*dcmp(c2)<0&&dcmp(c3)*dcmp(c4)<0;
}
//判断点是否在线段上(不包括端点 )
bool Onsegment(Point p,Point a1,Point a2)
{
    return dcmp(Cross(a1-p,a2-p))==0&&dcmp(Dot(a1-p,a2-p))<0;
}
```

10.6 最小三角形

```
// 最小三角形，注意longlong
const int maxn = 5000+10;
struct  Point{

  long long  x,y;
  Point(long long  xx = 0,long long  yy = 0):x(xx),y(yy){}
};
typedef Point Vector ;

Point operator - (const Point A,const Point B){
  return Point(A.x-B.x,A.y-B.y);
}
long long  Cross(Vector A,Vector B){
  return A.x*B.y-A.y*B.x;
}
Point A[maxn],B[maxn];
bool operator <(const Point &A,const Point &B){

  return A.y*B.x  < A.x*B.y;
}
int main(void)
{
  int N;cin>>N;
```

```
    for(int i = 0;i < N; ++i){
        cin>>A[i].x>>A[i].y;
    }
    double   ans = inf;
    for(int i = 0;i < N; ++i){
        int t = 0;
        for(int j = 0;j < N; ++j)
            if(i != j)
                B[t++] = A[j]-A[i];
        sort(B,B+t);
        // assert(t == N-1);
        for(int j = 0;j < t-1; ++j){
            ans = min(ans,fabs(Cross(B[j],B[j+1]))/2.0);
        }
    }
    printf("%.3f",ans);

    return 0;
}
```

10.7 最大三角形

```
// 最大三角形
const int N=50005;

struct Point
{
    double x,y;
};

Point stack[N];
Point p[N];
Point MinA;

int top;

double dist(Point A,Point B)
{
    return sqrt((A.x-B.x)*(A.x-B.x)+(A.y-B.y)*(A.y-B.y));
}

double cross(Point A,Point B,Point C)
{
    return (B.x-A.x)*(C.y-A.y)-(B.y-A.y)*(C.x-A.x);
}

bool cmp(Point a,Point b)
{
    double k=cross(MinA,a,b);
    if(k>0) return 1;
    if(k<0) return 0;
    return dist(MinA,a)<dist(MinA,b);
}

void Graham(int n)
{
    int i;
    for(i=1; i<n; i++)
```

```
        if(p[i].y<p[0].y||(p[i].y==p[0].y&&p[i].x<p[0].x))
            swap(p[i],p[0]);
    MinA=p[0];
    sort(p+1,p+n,cmp);
    stack[0]=p[0];
    stack[1]=p[1];
    top=1;
    for(i=2; i<n; i++)
    {
        while(cross(stack[top-1],stack[top],p[i])<=0&&top>=1) --top;
        stack[++top]=p[i];
    }
}

double rotating_calipers(int n)
{
    int j=1,k=0;
    double ans=0;
    for(int i=0;i<n;i++)
    {
        j=(i+1)%n;
        k=(j+1)%n;
        while(fabs(cross(stack[i],stack[j],stack[k]))<fabs(cross(stack[i],stack[j],stack[(k+1)%n])))
             k=(k+1)%n;
        while(j!=i&&k!=i)
        {
            ans=max(ans,fabs(cross(stack[i],stack[j],stack[k])));
            while(fabs(cross(stack[i],stack[j],stack[k]))<fabs(cross(stack[i],stack[j],stack[(k+1)%n])))
                 k=(k+1)%n;
            j=(j+1)%n;
        }
    }
    return ans*0.5;
}

int main()
{
    int n;
    while(~scanf("%d",&n))
    {
        if(n==-1) break;
        for(int i=0;i<n;i++)
           scanf("%lf%lf",&p[i].x,&p[i].y);
        if(n<3)
        {
            puts("0.00");
            continue;
        }
        Graham(n);
        top++;
        if(top<3)
        {
            puts("0.00");
            continue;
        }
        if(top==3)
        {
            printf("%.2lf\n",fabs(cross(stack[0],stack[1],stack[2]))/2);
            continue;
    }
    printf("%.2lf\n",rotating_calipers(top));
  }
  return 0;
}
```

10.8 三维凸包

```
struct Face{
  int v[3];
  Vector3 normal(Vector *P)
  {
    return Cross(P[v[1]]-P[v[0]],P[v[2]]-P[v[0]]);
  }
  int cansee(Point *P,int i)const
  {
    return Dot(P[i]-P[v[0]],normal(P)) > 0?1 : 0;
  }
};
vector <Face> CH3D(Point3* P,int n)
{
  vector <Face> cur;
  cur.push_back((Face){{0,1,2}});
  cur.push_back((Face){{2,1,0}});
  for(int i = 3;i < n; ++i)
  {
    vector<Face> next;
    //计算每条边 □ 左面 □ 的可见性
    for(int j= 0;j < cur.size(); ++j)
    {
      Face &f = cur[j];
      int res = f.cansee(P,i);
      if(!res) next.push_back(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])//(a,b) 是分界线，左边对P[i] 可见
         next.push_back((Face){{a,b,i}});
      }
    }
    cnr = next;
  }
  return cur;
}
double rand01() {return rand() / (double) RAND_MAX;}//0-1 的随机数
double randeps() {return (rand01()-0.5) * eps;}
Point3 add_noise(Point3 p)
{
  return Point3(p.x + randeps(),p.y+randeps(),p.z+randeps());
}

//.....................................................................
struct Face{
  int v[3];
  Vector3 normal(Vector *P)
  {
    return Cross(P[v[1]]-P[v[0]],P[v[2]]-P[v[0]]);
  }
  int cansee(Point *P,int i)const
  {
    return Dot(P[i]-P[v[0]],normal(P)) > 0?1 : 0;
  }
};
vector <Face> CH3D(Point3* P,int n)
{
  vector <Face> cur;
  cur.push_back((Face){{0,1,2}});
  cur.push_back((Face){{2,1,0}});
  for(int i = 3;i < n; ++i)
  {
    vector<Face> next;
    //计算每条边 □ 左面 □ 的可见性
    for(int j= 0;j < cur.size(); ++j)
    {
      Face &f = cur[j];
      int res = f.cansee(P,i);
      if(!res) next.push_back(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])//(a,b) 是分界线，左边对P[i] 可见
         next.push_back((Face){{a,b,i}});
      }
    }
    cnr = next;
  }
  return cur;
}
double rand01() {return rand() / (double) RAND_MAX;}//0-1 的随机数
double randeps() {return (rand01()-0.5) * eps;}
Point3 add_noise(Point3 p)
{
  return Point3(p.x + randeps(),p.y+randeps(),p.z+randeps());
}
```

10.9 三维几何模板

```
#include <bits/stdc++.h>
const double eps = 1e-6;
using namespace std;

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;
int dcmp(double d)
{
  if(fabs(d)< eps)
    return 0;
  else
    return d < 0?-1:1;
}
Vector3 operator +(Vector3 v1,Vector3 v2)
{
  return Vector3(v1.x+v2.x,v1.y+v2.y,v1.z+v2.z);
}
Vector3 operator -(Vector3 v1,Vector3 v2)
{
  return Vector3(v1.x-v2.x,v1.y-v2.y,v1.z-v2.z);
}
Vector3 operator *(Vector3 v,double c)
{
  return Vector3(v.x*c,v.y*c,v.z*c);
}
Vector3 operator /(Vector3 v,double c)
{
  return Vector3(v.x/c,v.y/c,v.z/c);
}
bool operator ==(Point3 A,Point3 B)
{
  return !dcmp(A.x-B.x)&&!dcmp(A.y-B.y)&&!dcmp(A.z-B.z);
}
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)//求两向量的夹角
{
  return acos(Dot(A,B)/(2*Length(A)*Length(B)));
}
double DistanceToplane(const Point3 &p,const Point3 &p0,const Vector3& n)//
{
  return fabs(Dot(p-p0,n))/Length(n);
}
Point3  GetPlaneProjection(const Point3&p,const Point3&p0,const Vector3&n)
{
  return p-n*Dot(p-p0,n);
}
//直线p1-p2 到平面p0-n的交点。 假定交点唯一存在
Point3 LinePlaneIntetsection(Point3 p1,Point3 p2,Point3 p0,Vector3 n)
{
  Vector3 v= p2 - p1;
//   /*if(dcmp(Dot(v,n))==0)
//   {
//       if(dcmp(Dot(p1-p0,n))==0)
//           直线在平面上
//       else
//           直线与平面平行
//   }
//   */
    double t  = Dot(n,p0-p1)/Dot(n,p2-p1);
    return p1 + v*t;
}
Point3 LinePlaneIntetsection(Point3 p1,Point3 p2,double A,double B,double C,double D)
{
    Vector3 v = p2-p1;
    double t = (A*p1.x+B*p1.y+C*p1.z+D)/(A*(p1.x-p2.x)+B*(p1.y-p2.y)+C*(p1.z-p2.z));
    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));
}
////已知平面的三点,求出点法式
//Vector3 Solven(Point3 A,Point3 B,Point3 C)
//{
//   return Cross(B-A,C-A);
//}
//判断一个点是否在三角形内，可以用面积法
bool PointInTri(Point3 P,Point3 A,Point3 B,Point3 C)
{
    double area1 = Area2(P,A,B);
    double area2 = Area2(P,A,C);
    double area3 = Area2(P,B,C);
    double area4 = Area2(A,B,C);
    return dcmp(area1+area2+area3-area4)==0;
}
//判断线段是否与三角形相交
bool TriSegIntersection(Point3 P0,Point3 P1,Point3 P2,Point3 A,Point3 B,Point3 &P)
{
    Vector3 n = Cross(P1-P0,P2-P0);

    if(dcmp(Dot(n,B-A))==0)
        return false;

    double t = Dot(n,P0-A)/Dot(n,B-A);
    if(dcmp(t) < 0 || dcmp(t-1) > 0)
        return false;
    P = A + (B-A) * t;
    return PointInTri(P,P0,P1,P2);
}
double DitantceToLine(Point3 P,Point3 A,Point3 B)
{
    return Length(Cross(A-P,B-P))/Length(A-B);
}
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(dcmp(Dot(v1,v2)) == 0) return Length(v2);
   if(dcmp(Dot(v1,v3)) >  0) return Length(v3);
   return Length(Cross(v1,v2))/Length(v1);
}
double Volume6(Point3 A,Point3 B,Point3 C,Point3 D)
```

```
{
    return Dot(D-A,Cross(B-A,C-A));
}
//
int  main(void)
{

    Point3 A(0,0,0),B(0,100,0),C(100,0,0),D(25,25,0);
    cout<<PointInTri(D,A,B,C)<<endl;
    return 0;
}
```