



ACM/ICPC/CCPC Template

浙江工商大学 ZJSU

ZYF

November 14, 2019

Contents

0	prime	1
0.1	素数筛选 (埃拉托色尼筛选)	1
0.2	判断素数	1
0.3	米勒罗宾素数	1
0.4	区间素数筛选	2
1	逆元	3
1.1	扩展 gcd 求逆元 (模数不一定为质数)	3
1.2	费马小定理求逆元 (模数需为质数)	3
2	线性同余方程	4
2.1	一次线性同余方程组	4
2.2	二次线性同余方程组	4
2.3	EXCRT	6
2.4	CRT	6
3	BSGS	8
3.1	bsgs	8
3.2	exbsgs	8
4	欧拉函数	11
4.1	欧拉函数	11
5	组合数学	13
5.1	lucas	13
5.2	exlucas	13
5.3	Big Combination	15
5.4	组合数学初始化 (杨辉三角)	15
5.5	组合数学初始化 (阶乘的逆元)	16
5.6	第一斯特林数	16
5.7	卡特兰数	17
6	异或	19
6.1	线性基	19
7	矩阵	21
7.1	斐波那切数列十进制	21
8	others	23
8.1	$ax+by=c$ exgcd	23
8.2	formula	23
8.3	万进制	23
8.4	定积分	24
8.5	用树状数组求逆序对	24
8.6	线性基	25
8.7	大数板子 (易)	26
8.8	大数板子 (完整)	29
8.9	快速乘	33
8.10	母函数	33
8.11	合数分解	34
9	未学会	36
9.1	莫比乌斯反演	36

10 几何	37
10.1 fcy	37
10.2 圆	42
10.3 体积	47
10.4 二维凸包	48
10.5 二维几何模板	49
10.6 最小三角形	51
10.7 最大三角形	52
10.8 三维凸包	54
10.9 三维几何模板	55

0 prime

0.1 素数筛选 (埃拉托色尼筛选)

```

1  /*
2  * 埃拉托色尼筛选法
3  * 素数筛选, 判断小于 MAXN 的数是不是素数。
4  * notprime 是一张表, 为 false 表示是素数, true 表示不是素数
5  */
6  const int MAXN = 1000010;
7  bool notprime[MAXN]; // 值为 false 表示素数, 值为 true 表示非素数
8  void init()
9  {
10     memset(notprime, false, sizeof(notprime));
11     notprime[0] = notprime[1] = true;
12     for (int i = 2; i < MAXN; i++)
13         if (!notprime[i])
14         {
15             if (i > MAXN / i)
16                 continue; // 防止后面 i*i 溢出 (或者 i,j 用 long long)
17             // 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
18             for (int j = i * i; j < MAXN; j += i)
19                 notprime[j] = true;
20         }
21 }

```

0.2 判断素数

```

1  bool judge(int x){
2      float n_sqrt;
3      if(x == 2 || x == 3){
4          return true;
5      }
6      if(x % 6 != 1 && x % 6 != 5){
7          return false;
8      }
9      n_sqrt = floor(sqrt(float(x)));
10     for(int i = 5; i <= n_sqrt; i += 6){
11         if(x % i == 0 || x % (i + 2) == 0){
12             return false;
13         }
14     }
15     return true;
16 }

```

0.3 米勒罗宾素数

```

1  //注意需要
2  //srand(time(NULL));
3  int modular_exp(int a, int m, int n) {
4      if (m == 0) {
5          return 1;
6      }
7      if (m == 1) {
8          return (a % n);
9      }
10     ll w = modular_exp(a, m / 2, n);

```

```
11  w = w * w % n;
12  if (m & 1) {
13      w = w * a % n;
14  }
15  return w;
16 }
17
18 bool Miller_Rabin(int n) {
19     if (n == 2) {
20         return true;
21     }
22     for (int i = 0; i < maxn; i++) {
23         int a = rand() % (n - 2) + 2;
24         if (modular_exp(a, n, n) != a) {
25             return false;
26         }
27     }
28     return true;
29 }
```

0.4 区间素数筛选

```
1  //对区间[a, b)内的整数执行筛法
2  //函数返回区间内素数个数
3  //is_prime[i - a] = true表示i是素数
4  1 < a < b <= 1e12, b - a <= 1e6;
5  const int maxn = "Edit";
6  bool is_prime_small[maxn], is_prime[maxn];
7  ll prime[maxn];
8
9  int segment_sieve(ll a, ll b){
10     int tot = 0;
11     for(ll i = 0; i * i < b; i++) is_prime_small[i] = true;
12     for(ll i = 0; i * i < b; i++) is_prime[i] = true;
13     for(ll i = 2; i * i < b; i++){
14         if(is_prime[i]){
15             for(ll j = 2 * i; j * j < b; j += i)
16                 is_prime_small[j] = false;
17             for(ll j = max(2LL, (a + i - 1) / i) * i; j < b; j += i)
18                 is_prime[j - a] = false;
19         }
20     }
21     for(ll i = 0; i < b - a; i++){
22         if(is_prime_small[i]) prime[tot++] = i + a;
23     }
24     return tot;
25 }
```

1 逆元

1.1 扩展 gcd 求逆元 (模数不一定为质数)

```
1 LL exgcd(LL a,LL b,LL &x,LL &y)//扩展欧几里得算法
2 {
3     if(b==0)
4     {
5         x=1,y=0;
6         return a;
7     }
8     LL ret=exgcd(b,a%b,y,x);
9     y-=a/b*x;
10    return ret;
11 }
12 LL getInv(int a,int mod)//求a在mod下的逆元,不存在逆元返回-1
13 {
14     LL x,y;
15     LL d=exgcd(a,mod,x,y);
16     return d==1?(x%mod+mod)%mod:-1;
17 }
```

1.2 费马小定理求逆元 (模数需为质数)

```
1 /*
2  * 费马小定理求逆元
3  * 仅当 mod 为素数时可用
4  * 调用 inv 函数获取逆元
5  */
6
7 const int mod = 1000000009;
8
9 long long quickpow(long long a, long long b)
10 {
11     if (b < 0)
12         return 0;
13     long long ret = 1;
14     a %= mod;
15     while (b)
16     {
17         if (b & 1)
18             ret = (ret * a) % mod;
19         b >>= 1;
20         a = (a * a) % mod;
21     }
22     return ret;
23 }
24 long long inv(long long a)
25 {
26     return quickpow(a, mod - 2);
27 }
```

2 线性同余方程

2.1 一次线性同余方程组

```
1 ll x, m, M, r, y, z;
2 int n;
3
4 ll gcd(ll a, ll b) {
5     return b == 0 ? a : gcd(b, a % b);
6 }
7
8 void inv(ll a, ll b) {
9     if (a % b == 0) {
10         z = 0;
11         y = 1;
12         return;
13     }
14     inv(b, a % b);
15     ll r = z;
16     z = y;
17     y = r - a / b * y;
18 }
19
20 void solve() {
21     x = 0;
22     m = 1;
23     for (int i = 0; i < n; i++) {
24         cin >> M >> r;
25         ll b = r - x, d = gcd(m, M);
26         if (b % d) {
27             cout << "-1" << endl; //不存在
28             return;
29         }
30         inv(m / d, M / d);
31         ll t = b / d * z % (M / d);
32         x += m * t;
33         m *= M / d;
34     }
35     x = x > 0 ? x : x + m;
36     cout << x << endl;
37 }
```

2.2 二次线性同余方程组

```
1 //p为模数
2 struct node{
3     ll p, d;
4 };
5
6 ll w;
7
8 ll quickMod(ll a, ll b, ll m){
9     ll ans = 1;
10    a %= m;
11    while(b){
12        if(b & 1){
13            ans = ans * a % m;
14        }
15    }
```

```
15     b >>= 1;
16     a = a * a % m;
17 }
18 return ans;
19 }
20
21 //二次域乘法
22 node multi_er(node a, node b, ll m){
23     node ans;
24     ans.p = (a.p * b.p % m + a.d * b.d % m * w % m) % m;
25     ans.d = (a.p * b.d % m + a.d * b.p % m) % m;
26     return ans;
27 }
28
29 //二次域上快速幂
30 node power(node a, ll b, ll m){
31     node ans;
32     ans.p = 1;
33     ans.d = 0;
34     while(b){
35         if(b & 1){
36             ans = multi_er(ans, a, m);
37         }
38         b >>= 1;
39         a = multi_er(a, a, m);
40     }
41     return ans;
42 }
43
44 //求勒让德符号
45 ll legendre(ll a, ll p){
46     return quickMod(a, (p - 1) >> 1, p);
47 }
48
49 ll mod(ll a, ll m){
50     a %= m;
51     return a < 0 ? a + m : a;
52 }
53
54 ll solve(ll n, ll p){
55     if(p == 2){
56         return 1;
57     }
58     if(legendre(n, p) + 1 == p){
59         return -1;
60     }
61     ll a = -1, t;
62     while(1){
63         a = rand() % p;
64         t = a * a - n;
65         w = mod(t, p);
66         if(legendre(w, p) + 1 == p){
67             break;
68         }
69     }
70
71     node tmp;
72     tmp.p = a;
73     tmp.d = 1;
```



```
74     node ans = power(tmp, (p + 1) >> 1, p);
75     return ans.p;
76 }
77
78 int main(void) {
79     int T;
80     cin >> T;
81     while(T--){
82         int n, p;
83         cin >> n >> p;
84         n %= p;
85         int a = solve(n, p);
86         if(a == -1){
87             cout << "No root" << endl;
88             continue;
89         }
90         int b = p - a;
91         if(a > n){
92             swap(a, b);
93         }
94         if(a == b){
95             cout << a << endl;
96         }else{
97             cout << a << ' ' << b << endl;
98         }
99     }
100 }
```

2.3 EXCRT

```
1  //x mod m[i] = r[i]; m[i] 可以两两不互质
2  //引用返回通解x = re + k * mo;函数返回是否有解
3
4  bool excrt(ll r[], ll m[], ll n, ll &re, ll &mo){
5      ll x, y;
6      mo = m[0], re = r[0];
7      for(int i = 1; i < n; i++){
8          ll d = exgcd(mo, m[i], x, y);
9          if((r[i] - re) % d != 0) return 0;
10         x = (r[i] - re) / d * x % (m[i] / d);
11         re += x * mo;
12         mo = mo / d * m[i];
13         re %= mo;
14     }
15     re = (re + mo) % mo;
16     return 1;
17 }
```

2.4 CRT

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

```
8    ll d = exgcd(tm, m[i], x, y);
9    re = (re + tm * x * r[i]) % mo;
10   }
11   re = (re + mo) % mo;
12 }
```

3 BSGS

3.1 bsgs

```

1 void bsgs(ll y,ll z,ll p)// $y^x \equiv z \pmod p$  gcd(y, p) = 1
2 {
3     if(y==0 && z==0){puts("1");return ;} //几句特判
4     if(y==0 && z!=0){return;} //不存在
5
6     mp.clear();
7     ll m=ceil(sqrt(p));
8     ll tmp=z%p; mp[tmp]=0; //右边  $z \cdot A^j$  , 当j=0时为z
9     for(ll i=1; i<=m; i++)
10    {
11        tmp=tmp*y%p;
12        mp[tmp]=i;
13    }
14
15    ll t=power(y,m,p);
16    tmp=1; //左边  $y^{i \cdot m}$  , 当i=0时为1
17    for(ll i=1; i<=m; i++)
18    {
19        tmp=tmp*t%p; //i每加1, 多乘  $y^{i \cdot m}$ 
20        if(mp[tmp])
21        {
22            ll ans=i*m-mp[tmp];
23            printf("%lld\n", (ans%p+p)%p);
24            return ;
25        }
26    }
27    return ; //不存在
28 }

```

3.2 exbsgs

```

1 //解决  $x^y \pmod z == k$  的最小y
2
3 struct Hash {
4     int i;
5     ll xi;
6     Hash(int a, ll b) : i(a), xi(b) {}
7 };
8
9 vector<Hash> has[maxn];
10
11 ll powMod(ll a, ll b, ll mod) {
12     ll res = 1;
13     while (b) {
14         if (b & 1) {
15             res = res * a % mod;
16         }
17         b >>= 1;
18         a = a * a % mod;
19     }
20     return res;
21 }
22
23 ll gcd(ll a, ll b) {

```

```
24     return b == 0 ? a : gcd(b, a % b);
25 }
26
27 ll exgcd(ll a, ll b, ll &x, ll &y) {
28     if (b == 0) {
29         x = 1;
30         y = 0;
31         return a;
32     }
33     ll res = exgcd(b, a % b, x, y);
34     ll t = x;
35     x = y;
36     y = t - a / b * y;
37     return res;
38 }
39
40 ll BSGS(ll x, ll z, ll p) {
41     z %= p;
42     ll val = 1;
43     for (int i = 0; i <= 100; i++, val = (val * x) % p) {
44         if (val == z) {
45             return i;
46         }
47     }
48     ll q = 1, cnt = 0;
49     while ((val = gcd(x, p)) != 1) {
50         if (z % val) {
51             return -1;
52         }
53         p /= val;
54         z /= val;
55         q = q * x / val % p;
56         cnt++;
57     }
58
59     ll m = (ll) sqrt((double) p);
60     for (int i = 0; i < maxn; i++) {
61         has[i].clear();
62     }
63     val = 1;
64     for (int i = 0; i <= m; i++) {
65         int vv = val % maxn;
66         has[vv].push_back(Hash(i, val));
67         val = val * x % p;
68     }
69
70     ll xm = powMod(x, m, p), a, b;
71     for (int i = 0; i <= m; i++) {
72         exgcd(q, p, a, b);
73         val = ((z * a) % p + p) % p;
74         ll vv = val % maxn;
75         for (int j = 0; j < has[vv].size(); j++) {
76             if (has[vv][j].xi == val) {
77                 return i * m + has[vv][j].i + cnt;
78             }
79         }
80         q = q * xm % p;
81     }
82     return -1;
```

83 }

4 欧拉函数

4.1 欧拉函数

```

1 //O(logn)的时间求一个数的phi
2 ll euler(ll n){
3     ll rt = n;
4     for(int i = 2; i * i <= n; i++){
5         if(n % i == 0){
6             rt -= rt / i;
7             while(n % i == 0){
8                 n /= i;
9             }
10        }
11    }
12    if(n > 1){
13        rt -= rt / n;
14    }
15    return rt;
16 }

17 //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
18
19 //用一个数组求欧拉值
20 ll phi[maxn];
21 void ZyfPhi() {
22     phi[1] = 0;
23     for(int i = 2; i < maxn; i++){
24         phi[i] = i;
25     }
26     for(int i = 2; i < maxn; i++){
27         if(phi[i] == i){
28             for(int j = i; j < maxn; j += i){
29                 phi[j] = phi[j] / i * (i - 1);
30             }
31         }
32     }
33 }
34 for (int i = 2; i < maxn; i++) {
35     phi[i] = phi[i] + phi[i - 1];
36 }
37 }
38
39 //O(n)得到欧拉函数phi[], 素数表prime[], 素数个数tot
40 bool vis[maxn];
41 int tot, phi[maxn], prime[maxn];
42 void Phi(){
43     phi[1] = 0;
44     for(int i = 2; i < maxn; i++){
45         if(!vis[i]){
46             prime[tot++] = i, phi[i] = i - 1;
47             for(int j = 0; j < tot; j++){
48                 if(i * prime[j] > maxn){
49                     break;
50                 }
51                 vis[i * prime[j]] = 1;
52                 if(i % prime[j] == 0){
53                     phi[i * prime[j]] = phi[i] * prime[j];
54                     break;
55                 }else{

```

```
56         phi[i * prime[j]] = phi[i] * (prime[j] - 1);
57     }
58 }
59 }
60 }
61 }
```

5 组合数学

5.1 lucas

```

1 //求解组合数取模p , 其中p为质数
2 //求解逆元
3 ll powMod(ll a, ll b, ll p){
4     ll ret = 1;
5     while(b){
6         if(b & 1){
7             ret = ret * a % mod;
8         }
9         b >>= 1;
10        a = a * a % p;
11    }
12    return ret;
13 }
14
15 //求组合数, 因为分解得较小了, 所以可以用暴力
16 ll C(ll n, ll m, ll p){
17     if(m > n){
18         return 0;
19     }
20     ll c1 = 1, c2 = 1;
21     for(int i = n + m - 1; i <= n; i++){
22         c1 = c1 * i % p;
23     }
24     for(int i = 2; i <= m; i++){
25         c2 = c2 * i % p;
26     }
27
28     return c1 * powMod(c2, p - 2, p) % p;
29 }
30
31 ll lucas(ll n, ll m, ll p){
32     if(!m) return 1;
33     return C(n % p, m % p, p) * lucas(n / p, m / p, p) * p;
34 }

```

5.2 exlucas

```

1 //求解组合数求模p , 且p不一定是质数
2 ll c[1000006], a[1000005];
3 ll powMod(ll a, ll b, ll p){
4     ll ret = 1;
5     while(b){
6         if(b & 1) ret = ret * a % p;
7         b >>= 1;
8         a = a * a % p;
9     }
10    return ret;
11 }
12
13 //求阶乘
14 ll fac(ll n, ll p, ll pk){
15     if(!n) return 1;
16     ll ans = 1;
17     for(int i = 1; i < pk; i++){

```



```

18     if(i % p) ans = ans * i % pk;//同余部分
19 }
20 ans = powMod(ans, n / pk, pk);
21 for(int i = 1; i <= n % pk; i++){//剩余无法凑同余的部分
22     if(i % p) ans = ans * i % pk;
23 }
24 return ans * fac(n / p, p, pk) % pk;
25 }
26
27 ll exgcd(ll a, ll b, ll &x, ll &y){
28     if(!b){
29         x = 1, y = 0;
30         return a;
31     }
32     ll xx, yy, g = exgcd(b, a % b, xx, yy);
33     x = yy;
34     y = xx - a / b * yy;
35     return g;
36 }
37
38 ll inv(ll a, ll b){ //求逆元
39     ll x, y;
40     exgcd(a, b, x, y);
41     return (x % b + b) % b;
42 }
43
44 //求组合数
45 ll C(ll n, ll m, ll p, ll pk){
46     if(m > n) return 0;
47     ll f1 = fac(n, p, pk), f2 = fac(m, p, pk), f3 = fac(n - m, p, pk), cnt = 0;
48     for(ll i = n; i; i /= p){
49         cnt += i / p;
50     }
51     for(ll i = m; i; i /= p){
52         cnt -= i / p;
53     }
54     for(ll i = n - m; i; i /= p){
55         cnt -= i / p;
56     }
57     return f1 * inv(f2, pk) % pk * inv(f3, pk) % pk * powMod(p, cnt, pk) % pk;
58 }
59
60 ll CRT(ll cnt){
61     ll M = 1, ans = 0;
62     for(int i = 1; i <= cnt; i++){
63         M *= c[i]; //p的值发生变化, 所以要重新计算
64     }
65     for(int i = 1; i <= cnt; i++){
66         ans = (ans + a[i] * (M / c[i]) % M * inv(M / c[i], c[i]) % M) % M;
67     }
68     return ans;
69 }
70
71 ll exlucas(ll n, ll m, ll p){
72     ll temp, cnt = 0;
73     for(int i = 2; p > 1 && i <= p / i; i++){
74         ll tmp = 1;
75         while(p % i == 0){
76             p /= i, tmp *= i;

```

```

77     }
78     if(tmp > 1){
79         a[++cnt] = C(n, m, i, tmp);
80         c[cnt] = tmp;
81     }
82 }
83 if(p > 1){
84     c[++cnt] = p, a[cnt] = C(n, m, p, p);
85 }
86 return CRT(cnt);
87 }
88
89 int main(){
90     ll m, n, p;
91     cin >> n >> m >> p;
92     cout << exlucas(n, m, p);
93 }

```

5.3 Big Combination

```

1  // 0 <= n <= 1e9, 0 <= m <= 1e4, 1 <= k <= 1e9 + 7
2  //利用逆元求解
3  vector<int> v;
4  int dp[110];
5  ll Cal(int l, int r, int k, int dis){
6      ll res = 1;
7      for(int i = l; i <= r; i++){
8          int t = i;
9          for(int j = 0; j < v.size(); j++){
10             int y = v[j];
11             while(t % y == 0) dp[j] += dis, t /= y;
12         }
13         res = res * (ll) t % k;
14     }
15     return res;
16 }
17
18 ll Comb(int n, int m, int k){
19     memset(dp, 0, sizeof(dp));
20     v.clear();
21     int tmp = k;
22     for(int i = 2; i * i <= tmp; i++){
23         if(tmp % i == 0){
24             int num = 0;
25             while(tmp % i == 0) tmp /= i, num++;
26             v.push_back(i);
27         }
28     }
29     if(tmp != 1) v.push_back(tmp);
30     ll ans = Cal(n - m + 1, n, k, 1);
31     for(int j = 0; j < v.size(); j++) ans = ans * powMod(v[j], dp[j], k) % k;
32     ans = ans * inv(Cal(2, m, k, -1), k) % k; //inv是求逆元函数
33     return ans;
34 }

```

5.4 组合数学初始化 (杨辉三角)

```

1 // 0 <= m <= n <= 1000
2
3 const int maxn = 1010;
4 ll C[maxn][maxn];
5 void CalComb(){
6     C[0][0] = 1;
7     for(int i = 1; i < maxn; i++){
8         C[i][0] = 1;
9         for(int j = 1; j <= i; j++){
10             C[i][j] = (C[i - 1][j - 1] + C[i - 1][j]) % mod;
11         }
12     }
13 }

```

5.5 组合数学初始化 (阶乘的逆元)

```

1 // 0 <= m <= n <= 1e5, 模p为质数
2
3 const int maxn = 100010;
4 ll f[maxn];
5 ll inv[maxn]; //阶乘的逆元
6 void CalFact(){
7     f[0] = 1;
8     for(int i = 1; i < maxn; i++){
9         f[i] = (f[i - 1] * i) % p;
10    }
11    inv[maxn - 1] = powMod(f[maxn - 1], p - 2, p);
12    for(int i = maxn - 2; ~i; i--){
13        inv[i] = inv[i + 1] * (i + 1) % p;
14    }
15 }
16
17 ll C(int n, int m){
18     return f[n] * inv[m] % p * inv[n - m] % p;
19 }

```

5.6 第一斯特林数

```

1 //s(n, m) = s(n - 1, m - 1) + (n - 1) * s(n - 1, m)
2 //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。
3 typedef long long ll;
4 const int N = 101;
5 ll s[N][N];
6
7 void init(){
8     memset(s, 0, sizeof(s));
9     s[1][1] = 1;
10    for(int i = 2; i < N; i++){
11        for(int j = 1; j <= i; j++){
12            s[i][j] = (s[i - 1][j - 1] + (i - 1) * s[i - 1][j]);
13        }
14    }
15 }
16
17 int main(void){
18     init();
19     int T;

```

```
20  cin >> T;
21  while(T--){
22      int n, k;
23      cin >> n >> k;
24      ll sum = 1;
25      for(ll i = 2; i <= n; i++){
26          sum *= i;
27      }
28      ll fz = 0;
29      for(int i = 1; i <= k; i++){
30          fz += (s[n][i] - s[n-1][i-1]);
31      }
32      double ans = fz * 1.0 / sum;
33      printf("%.4lf\n", ans);
34  }
35 }
```

5.7 卡特兰数

```
1  int a[105][100];
2
3  void catalan(){
4      a[2][0] = 1;
5      a[2][1] = 2;
6      a[1][0] = 1;
7      a[1][1] = 1;
8      int len = 1, yu;
9      for(int i = 3; i < 101; i++){
10         yu = 0;
11         for(int j = 1; j <= len; j++){
12             int t = (a[i-1][j]) * (4 * i - 2) + yu;
13             yu = t/10;
14             a[i][j] = t % 10;
15         }
16         while(yu){
17             a[i][++len] = yu % 10;
18             yu /= 10;
19         }
20         for(int j = len; j >= 1; j--){
21             int t = a[i][j] + yu * 10;
22             a[i][j] = t / (i + 1);
23             yu = t % (i + 1);
24         }
25         while(!a[i][len]){
26             len--;
27         }
28         a[i][0] = len;
29     }
30 }
31
32 int main(void){
33     catalan();
34     int n;
35     while(cin >> n){
36         if(n == -1){
37             break;
38         }
39         for(int i = a[n][0]; i > 0; i--){
```

```
40     cout << a[n][i];
41     }
42     cout << endl;
43 }
44 }
```

45 //解决的问题

46 //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。

47 //2.Cn可以表示有 n 个结点的不同构的二叉树的种类数。

48 //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。

49 //4.Cn可以表示在 $n \times n$ 的格点中从左下角沿格线走到右上角且始终不超过对角线的方案数。

50 //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。

51 //6.Cn表示有 n 个元素的出栈顺序的种类数。

52 //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。

53 //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

```

1  //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。
2  //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
3  const int maxn = (int) 5e5 + 100;
4  int p[maxn][31], pos[maxn][31];
5  int n, q;
6  int lastans = 0;
7
8  typedef long long ll;
9
10 void push_back(int x, int i) {
11     for (int j = 0; j <= 30; j++) {
12         p[i][j] = p[i - 1][j];
13         pos[i][j] = pos[i - 1][j];
14     }
15     int ti = i;
16     for (int j = 30; j >= 0; j--) {
17         if (x & (1 << j)) {
18             if (!p[i][j]) {
19                 p[i][j] = x;
20                 pos[i][j] = ti;
21                 break;
22             }
23             if (pos[i][j] < ti) {
24                 swap(p[i][j], x);
25                 swap(pos[i][j], ti);
26             }
27             x ^= p[i][j];
28         }
29     }
30 }
31
32 int main(void) {
33     ios_base::sync_with_stdio(false);
34     cin.tie(0);
35     cout.tie(0);
36     int T;
37     cin >> T;
38     while (T--) {
39         lastans = 0;
40         // memset(p, 0, sizeof(p));
41         // memset(pos, 0, sizeof(pos));
42         cin >> n >> q;
43         int x;
44         for (int i = 1; i <= n; i++) {
45             cin >> x;
46             push_back(x, i);
47         }
48         while (q--) {
49             int e, l, r;
50             cin >> e;
51             if (e == 1) {
52                 cin >> l;
53                 l ^= lastans;

```

```
54     push_back(l, n + 1);
55     n++;
56 } else {
57     cin >> l >> r;
58     l = (l ^ lastans) % n + 1;
59     r = (r ^ lastans) % n + 1;
60     if (l > r) {
61         swap(l, r);
62     }
63     int ret = 0;
64     for (int i = 30; i >= 0; i--) {
65         if (pos[r][i] >= l && (ret ^ p[r][i]) > ret) {
66             ret ^= p[r][i];
67         }
68     }
69     cout << ret << endl;
70     lastans = ret;
71 }
72 }
73 }
74 }
```

7 矩阵

7.1 斐波那切数列十进制

```

1  typedef long long ll;
2  const int maxn = (int) 1e6 + 100;
3  char str[maxn];
4  int num[maxn];
5  ll mod;
6
7  struct mat {
8      ll m[3][3];
9
10     mat() {
11         m[1][1] = m[1][2] = m[2][1] = m[2][2] = 0;
12     }
13
14     mat friend operator*(mat a, mat b) {
15         mat res;
16         for (int k = 1; k <= 2; k++) {
17             for (int i = 1; i <= 2; i++) {
18                 for (int j = 1; j <= 2; j++) {
19                     res.m[i][j] += a.m[i][k] * b.m[k][j];
20                 }
21             }
22         }
23
24         for (int i = 1; i <= 2; i++) {
25             for (int j = 1; j <= 2; j++) {
26                 res.m[i][j] %= mod;
27             }
28         }
29         return res;
30     }
31
32     mat friend operator^(mat a, int b) {
33         mat res;
34         res.m[1][1] = res.m[2][2] = 1LL;
35         while (b) {
36             if (b & 1) {
37                 res = res * a;
38             }
39             a = a * a;
40             b >>= 1;
41         }
42         return res;
43     }
44 };
45
46 int main(void) {
47     #ifndef ACM_LOCAL
48         freopen("in.txt", "r", stdin);
49         freopen("out.txt", "w", stdout);
50     #endif
51     ios::sync_with_stdio(false);
52     cin.tie(0);
53     cout.tie(0);
54
55     int a, b, x1, x2;

```



```
56  cin >> x1 >> x2 >> a >> b;
57  cin >> str >> mod;;
58
59  int len = strlen(str);
60  for (int i = 0; i < len; i++) {
61      num[i] = str[i] - '0';
62  }
63
64  num[len - 1]--;
65  for (int i = len - 1; i >= 0 && num[i] < 0; i--) {
66      num[i] += 10;
67      num[i - 1]--;
68  }
69  mat base, A, ans;
70  ans.m[1][1] = ans.m[2][2] = 1;
71  base.m[1][1] = a % mod;
72  base.m[1][2] = b % mod;
73  base.m[2][1] = 1LL;
74  A.m[1][1] = x2 % mod;
75  A.m[2][1] = x1 % mod;
76  for (int i = len - 1; i >= 0; i--) {
77      if (num[i]) {
78          ans = ans * (base ^ num[i]);
79      }
80      base = base ^ 10;
81  }
82  ans = ans * A;
83  cout << ans.m[1][1] % mod << endl;
84 }
```

8 others

8.1 $ax+by=c$ exgcd

```

1 //引用返回通解:  $X = x + k * dx, Y = y - k * dy$ ;
2 //引用返回的x是最小非负整数解, 方程无解函数返回0
3
4 #define Mod(a, b) (((a) % (b)) + (b)) % (b))
5
6 ll exgcd(ll a, ll b, ll &x, ll &y){
7     ll d = a;
8     if(b){
9         d = exgcd(b, a % b, y, x);
10        y -= x * (a / b);
11    }else{
12        x = 1;
13        y = 0;
14    }
15    return d;
16 }
17
18 bool solve(ll a, ll b, ll c, ll &x, ll &y, ll &dx, ll &dy){
19     if(a == 0 & b == 0){
20         return 0;
21     }
22     ll x0, y0;
23     ll d = exgcd(a, b, x0, y0);
24     if(c % d != 0) return 0;
25     dx = b / d, dy = a / d;
26     x = Mod(x0 * c / d, dx);
27     y = (c - a * x) / b;
28     return 1;
29 }

```

8.2 formula

8.3 万进制

```

1 #include <iomanip>
2 void factorial(int n)
3 {
4     int a[10001];
5     int places, carry, i, j;
6
7     a[0] = 1;
8     places = 0; //当前数的总位数
9     for (i = 1; i <= n; i++)
10    {
11        carry = 0;
12        for (j = 0; j <= places; j++)
13        {
14            a[j] = a[j] * i + carry; //如果是多次幂函数, 将i改成数字即可
15            carry = a[j] / 10000;
16            a[j] %= 10000;
17        }
18        if (carry > 0)

```

```

19     {
20         places++;
21         a[places] = carry;
22     }
23 }
24 /*
25  * 输出
26  * 最高位原样输出
27  * 其他位小于1000的，高位补0
28  * 需要头文件<iomanip>
29  */
30 cout << a[places];
31 for (i = places - 1; i >= 0; i--)
32 {
33     cout << setw(4) << setfill('0') << a[i];
34 }
35 cout << endl;
36 }

```

8.4 定积分

```

1 double F(double x)
2 {
3     //Simpson公式用到的函数
4 }
5 double simpson(double a, double b)//三点Simpson法，这里要求F是一个全局函数
6 {
7     double c = a + (b - a) / 2;
8     return (F(a) + 4 * F(c) + F(b)) * (b - a) / 6;
9 }
10 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
11 {
12     double c = a + (b - a) / 2;
13     double L = simpson(a, c), R = simpson(c, b);
14     if (fabs(L + R - A) <= 15 * eps) return L + R + (L + R - A) / 15.0;
15     return asr(a, c, eps / 2, L) + asr(c, b, eps / 2, R);
16 }
17 double asr(double a, double b, double eps)//自适应Simpson公式（主过程）
18 {
19     return asr(a, b, eps, simpson(a, b));
20 }

```

8.5 用树状数组求逆序对

```

1 //用树状数组的方法求逆序对
2 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
3 ll ans;
4 int n;
5 int lowbit(int x){
6     return x & (-x);
7 }
8
9 void add(ll a[], ll pos, ll val){
10     while(pos <= maxn){
11         a[pos] += val;
12         pos += lowbit(pos);

```

```

13 }
14 }
15
16 ll sum(ll a[], ll pos){//求逆序对的个数
17     ll tmp = 0;
18     while(pos > 0){
19         tmp += a[pos];
20         pos -= lowbit(pos);
21     }
22     return tmp;
23 }
24
25 int main(void) {
26     while(cin >> n){
27         memset(c, 0, sizeof(ll) * (n + 5));
28         memset(b, 0, sizeof(ll) * (n + 5));
29         ans = 0;
30         //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
31         for(int i = 1; i <= n; i++){
32             cin >> num[i];
33             add(c, num[i], 1);
34             //逆序对的个数
35             //ans += sum(c, num[i]);
36             ans += num[i] * (i - sum(c, num[i]));
37         }
38         for(int i = n; i >= 1; i--){
39             ans += num[i] * (sum(b, num[i]));
40             add(b, num[i], 1);
41         }
42         cout << ans << endl;
43     }
44 }

```

8.6 线性基

```

1  const int MAXN = 31; //如果为 long long 则改成 63 即可
2  int d[MAXN + 5];
3  void init()
4  {
5      memset(d, 0, sizeof(d));
6  }
7  void add(int x)
8  {
9      for (int i = MAXN; i >= 0; i--)
10     {
11         if (x & (1LL << i))
12         {
13             if (d[i])
14                 x ^= d[i];
15             else
16             {
17                 d[i] = x;
18                 break;
19             }
20         }
21     }
22 }
23 // 如何求异或后第k小的值

```

```

24 void work() //处理线性基
25 {
26     for (int i = 1; i <= 60; i++)
27         for (int j = 1; j <= i; j++)
28             if (d[i] & (1 << (j - 1)))
29                 d[i] ^= d[j - 1];
30 }
31 ll k_th(ll k)
32 {
33     if (k == 1 && tot < n)
34         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
35     if (tot < n)
36         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
37     work();
38     ll ans = 0;
39     for (int i = 0; i <= 60; i++)
40         if (d[i] != 0)
41         {
42             if (k % 2 == 1)
43                 ans ^= d[i];
44             k /= 2;
45         }
46 }

```

8.7 大数板子 (易)

```

1  int flag = 1;
2  //初始化
3  void initial(string &a, string &b){
4      while (a.size() < b.size()) a = '0' + a;
5      while (b.size() < a.size()) b = '0' + b;
6  }
7  //打印
8  void print(string &a, string &b){
9      cout << a << endl;
10     cout << b << endl;
11 }
12 //找出最大的字符串
13 void findMax(string &a, string &b){
14     string tmp;
15     if (a < b){
16         tmp = b;
17         b = a;
18         a = tmp;
19     }
20 }
21 //删除第一个字符'0'
22 bool del(string &a){
23     if (a[0] == '0'){
24         a.erase(0, 1);
25         return true;
26     }
27     else
28         return false;
29 }
30 //删除前面所有的0
31 void delAllZroe(string &a){

```

```
32 while (del(a)){
33     del(a);
34 };
35 }
36 //大数加法
37 string bigltergeAdd(string a, string b){
38     initial(a, b);
39     a = '0' + a;
40     b = '0' + b;
41     for (int i = a.size() - 1; i >= 0; i--){
42         int num1 = a[i] - '0';
43         int num2 = b[i] - '0';
44         if (num1 + num2 > 9){
45             a[i - 1] = a[i - 1] - '0' + 1 + '0';
46             a[i] = (num1 + num2) - 10 + '0';
47         }
48         else{
49             a[i] = (num1 + num2) + '0';
50         }
51     }
52     del(a);
53     // cout<<a<<endl;
54     return a;
55 }
56 //大数减法
57 string bigltergeSub(string a, string b){
58     initial(a, b);
59     string tmp = a;
60     findMax(a, b);
61     if (a != tmp){
62         flag = -1;
63     }
64     for (int i = a.size() - 1; i >= 0; i--){
65         int num1 = a[i] - '0';
66         int num2 = b[i] - '0';
67         if (num1 < num2){
68             a[i - 1] = a[i - 1] - '0' - 1 + '0';
69             a[i] = (num1 + 10 - num2) + '0';
70         }
71         else{
72             a[i] = (num1 - num2) + '0';
73         }
74     }
75     del(a);
76     // cout<<a<<endl;
77     return a;
78 }
79 //大数乘法(大数加法实现)
80 void bigltergeMul(string a, string b){
81     delAllZroe(a);
82     delAllZroe(b);
83     if (a == "" || b == ""){
84         printf("0\n"); return;
85     }
86     initial(a, b);
87     findMax(a, b);
88     string res = "0";
89     int count = 0;
90     delAllZroe(b);
```

```
91  for (int i = b.size() - 1; i >= 0; i--){
92      int num1 = b[i] - '0';
93      if (i != b.size() - 1)    a = a + '0';
94      for (int j = 1; j <= num1; j++){
95          res = bigltergeAdd(res, a);
96      }
97  }
98  delAllZroe(res);
99  cout << res << endl;
100 }
101 //大数除法
102 void bigltergeDiv(string a, string b){
103     initial(a, b);
104     if (a<b){ cout << "0" << endl; return; }
105     delAllZroe(b);
106     string res = "0";
107     string restmp = "1";
108     string tmp = b;
109     for (int i = 1; i<(a.size() - b.size()); i++){
110         tmp += '0';
111         restmp += '0';
112     }
113     initial(a, b);
114     while (a >= b){
115         initial(a, tmp);
116         if (a >= tmp){
117             a = bigltergeSub(a, tmp);
118             res = bigltergeAdd(res, restmp);
119         }
120         else{
121             tmp.erase(tmp.size() - 1);
122             restmp.erase(restmp.size() - 1);
123             initial(a, tmp);
124             if (a >= tmp){
125                 a = bigltergeSub(a, tmp);
126                 res=bigltergeAdd(res, restmp);
127             }
128         }
129         initial(a, b);
130     }
131     cout << res << endl;
132 }
133
134 int main(void) {
135     string a, b;
136     while (cin >> a >> b){
137         string ans;
138         //ans = bigltergeAdd(a,b);
139         ans = bigltergeSub(a,b);
140         // bigltergeMul(a,b);
141         // bigltergeDiv(a,b);
142         if(flag == -1){
143             cout << '-';
144         }
145         cout << ans << endl;
146     }
147 }
```

8.8 大数板子 (完整)

```

1  const int maxn = 10005; /*精度位数,自行调整*/
2  //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
3  //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)
4  //3.还可以加上qmul, 取模的过程也就是str(), c_str()再搞一次
5  class bign {
6      //io*2 bign*5*2 bool*6
7      friend istream &operator>>(istream &, bign &);
8
9      friend ostream &operator<<(ostream &, const bign &);
10
11     friend bign operator+(const bign &, const bign &);
12
13     friend bign operator+(const bign &, int &);
14
15     friend bign operator*(const bign &, const bign &);
16
17     friend bign operator*(const bign &, int &);
18
19     friend bign operator-(const bign &, const bign &);
20
21     friend bign operator-(const bign &, int &);
22
23     friend bign operator/(const bign &, const bign &);
24
25     friend bign operator/(const bign &, int &);
26
27     friend bign operator%(const bign &, const bign &);
28
29     friend bign operator%(const bign &, int &);
30
31     friend bool operator<(const bign &, const bign &);
32
33     friend bool operator>(const bign &, const bign &);
34
35     friend bool operator<=(const bign &, const bign &);
36
37     friend bool operator>=(const bign &, const bign &);
38
39     friend bool operator==(const bign &, const bign &);
40
41     friend bool operator!=(const bign &, const bign &);
42
43 private://如果想访问len,改成public
44     int len, s[maxn];
45 public:
46     bign() {
47         memset(s, 0, sizeof(s));
48         len = 1;
49     }
50
51     bign operator=(const char *num) {
52         int i = 0, ol;
53         ol = len = strlen(num);
54         while (num[i++] == '0' && len > 1)
55             len--;
56         memset(s, 0, sizeof(s));
57         for (i = 0; i < len; i++)

```



```
58     s[i] = num[ol - i - 1] - '0';
59     return *this;
60 }
61
62 bign operator=(int num) {
63     char s[maxn];
64     sprintf(s, "%d", num);
65     *this = s;
66     return *this;
67 }
68
69 bign(int num) {
70     *this = num;
71 }
72
73 bign(const char *num) {
74     *this = num;
75 }
76
77 string str() const {
78     string res = "";
79     for (int i = 0; i < len; i++)
80         res = char(s[i] + '0') + res;
81     if (res == "")
82         res = "0";
83     return res;
84 }
85 };
86
87 bool operator<(const bign &a, const bign &b) {
88     int i;
89     if (a.len != b.len)
90         return a.len < b.len;
91     for (i = a.len - 1; i >= 0; i--)
92         if (a.s[i] != b.s[i])
93             return a.s[i] < b.s[i];
94     return false;
95 }
96
97 bool operator>(const bign &a, const bign &b) {
98     return b < a;
99 }
100
101 bool operator<=(const bign &a, const bign &b) {
102     return !(a > b);
103 }
104
105 bool operator>=(const bign &a, const bign &b) {
106     return !(a < b);
107 }
108
109 bool operator!=(const bign &a, const bign &b) {
110     return a < b || a > b;
111 }
112
113 bool operator==(const bign &a, const bign &b) {
114     return !(a < b || a > b);
115 }
116
```

```
117 bign operator+(const bign &a, const bign &b) {
118     int up = max(a.len, b.len);
119     bign sum;
120     sum.len = 0;
121     for (int i = 0, t = 0; t || i < up; i++) {
122         if (i < a.len)
123             t += a.s[i];
124         if (i < b.len)
125             t += b.s[i];
126         sum.s[sum.len++] = t % 10;
127         t /= 10;
128     }
129     return sum;
130 }
131
132 bign operator+(const bign &a, int &b) {
133     bign c = b;
134     return a + c;
135 }
136
137 bign operator*(const bign &a, const bign &b) {
138     bign res;
139     for (int i = 0; i < a.len; i++) {
140         for (int j = 0; j < b.len; j++) {
141             res.s[i + j] += (a.s[i] * b.s[j]);
142             res.s[i + j + 1] += res.s[i + j] / 10;
143             res.s[i + j] %= 10;
144         }
145     }
146     res.len = a.len + b.len;
147     while (res.s[res.len - 1] == 0 && res.len > 1)
148         res.len--;
149     if (res.s[res.len])
150         res.len++;
151     return res;
152 }
153
154 bign operator*(const bign &a, int &b) {
155     bign c = b;
156     return a * c;
157 }
158
159 //只支持大数减小数
160 bign operator-(const bign &a, const bign &b) {
161     bign res;
162     int len = a.len;
163     for (int i = 0; i < len; i++) {
164         res.s[i] += a.s[i] - b.s[i];
165         if (res.s[i] < 0) {
166             res.s[i] += 10;
167             res.s[i + 1]--;
168         }
169     }
170     while (res.s[len - 1] == 0 && len > 1)
171         len--;
172     res.len = len;
173     return res;
174 }
175
```

```
176 bign operator-(const bign &a, int &b) {
177     bign c = b;
178     return a - c;
179 }
180
181 bign operator/(const bign &a, const bign &b) {
182     int i, len = a.len;
183     bign res, f;
184     for (i = len - 1; i >= 0; i--) {
185         f = f * 10;
186         f.s[0] = a.s[i];
187         while (f >= b) {
188             f = f - b;
189             res.s[i]++;
190         }
191     }
192     while (res.s[len - 1] == 0 && len > 1)
193         len--;
194     res.len = len;
195     return res;
196 }
197
198 bign operator/(const bign &a, int &b) {
199     bign c = b;
200     return a / c;
201 }
202
203 bign operator%(const bign &a, const bign &b) {
204     int len = a.len;
205     bign f;
206     for (int i = len - 1; i >= 0; i--) {
207         f = f * 10;
208         f.s[0] = a.s[i];
209         while (f >= b)
210             f = f - b;
211     }
212     return f;
213 }
214
215 bign operator%(const bign &a, int &b) {
216     bign c = b;
217     return a % c;
218 }
219
220 bign &operator+=(bign &a, const bign &b) {
221     a = a + b;
222     return a;
223 }
224
225 bign &operator-=(bign &a, const bign &b) {
226     a = a - b;
227     return a;
228 }
229
230 bign &operator*=(bign &a, const bign &b) {
231     a = a * b;
232     return a;
233 }
234
```

```
235 bign &operator/=(bign &a, const bign &b) {
236     a = a / b;
237     return a;
238 }
239
240 bign &operator++(bign &a) {
241     a = a + 1;
242     return a;
243 }
244
245 bign &operator++(bign &a, int) {
246     bign t = a;
247     a = a + 1;
248     return t;
249 }
250
251 bign &operator--(bign &a) {
252     a = a - 1;
253     return a;
254 }
255
256 bign &operator--(bign &a, int) {
257     bign t = a;
258     a = a - 1;
259     return t;
260 }
261
262 istream &operator>>(istream &in, bign &x) {
263     string s;
264     in >> s;
265     x = s.c_str();
266     return in;
267 }
268
269 ostream &operator<<(ostream &out, const bign &x) {
270     out << x.str();
271     return out;
272 }
273
274 int main(void) {
275     bign a;
276     bign b;
277     cin >> a >> b;
278     cout << a / b << endl;
279 }
```

8.9 快速乘

```
1 inline ll ksc(ll x, ll y, ll p){ //p是mod
2     ll z = (long double) x / p * y;
3     ll res = (unsigned long long)x * y - (unsigned long long)z * p;
4     return (res + p) % p;
5 }
```

8.10 母函数

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

```

2  const int number=3;
3  int main()
4  {
5      int sum;//sum是指数
6      int c1[33005], c2[33005];
7      while(scanf("%d", &sum), sum)
8      {
9          for(int i = 0; i <= sum; ++i)
10         {
11             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
12             c2[i] = 0;
13         }
14         //复杂度O(number*sum*sum),也可通过打表后O(1)访问
15         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
16         {
17             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
18             {
19                 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
20                 {
21                     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
                        边括号系数都为1，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
22                 }
23             }
24             for(int j = 0; j <= sum; ++j)
25             {
26                 c1[j] = c2[j];
27                 c2[j] = 0; // 记得每次合并一个括号后要把 c2 清零
28             }
29         }
30         printf("%d\n", c1[sum]);
31     }
32     return 0;
33 }

```

8.11 合数分解

```

1  //素数筛选
2  const int MAXN=100000;
3  int prime[MAXN+1];//得到小于等于MAXN的所有素数
4  void getPrime()
5  {
6      memset(prime,0,sizeof(prime));
7      for(int i=2;i<=MAXN;i++)
8      {
9          if(!prime[i])prime[++prime[0]]=i;
10         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
11         {
12             prime[prime[j]*i]=1;
13             if(i%prime[j]==0)break;
14         }
15     }
16 }
17 //合数分解（前面需要先素数筛选）
18 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
19 int fatCnt;//fatCnt的值表示整除数种类
20 int getFactors(long long x)
21 {

```

```
22 fatCnt=0;
23 long long tmp=x;
24 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
25 {
26     factor[fatCnt][1]=0;
27     if(tmp%prime[i]==0)
28     {
29         factor[fatCnt][0]=prime[i];
30         while(tmp%prime[i]==0)
31         {
32             factor[fatCnt][1]++;
33             tmp/=prime[i];
34         }
35         fatCnt++;
36     }
37 }
38 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。
39 {
40     factor[fatCnt][0]=tmp;
41     factor[fatCnt++][1]=1;
42 }
43 return fatCnt;
44 }
```

9 未学会

9.1 莫比乌斯反演

```

1 //F(n) = Σ d|n f(d) ⇔ f(n) = Σ d|n μ(d)F(n/d)
2 //F(n) = Σ n|d f(d) ⇔ f(n) = Σ n|d μ(d/n)F(d)
3
4 const int maxn = "Edit";
5 int prime[maxn], tot, mu[maxn];
6 bool check[maxn];
7 void CalMu(){
8     mu[1] = 1;
9     for(int i = 2; i < maxn; i++){
10         if(!check[i]) prime[tot++] = i, mu[i] = -1;
11         for(int j = 0; j < tot; j++){
12             if(i * prime[j] >= maxn) break;
13             check[i * prime[j]] = true;
14             if(i % prime[j] == 0){
15                 mu[i * prime[j]] = 0;
16                 break;
17             }else{
18                 mu[i * prime[j]] = - mu[i];
19             }
20         }
21     }
22 }
23
24 //Examples
25 //有n个数(n < 100000, 1 < ai < 1e6), 问这n个数中互质的数的对数
26
27 const int maxn = "Edit";
28 int b[maxn];
29
30 ll solve(int n){
31     ll ans = 0;
32     for(int i = 0, x; i < n; i++){
33         cin >> x;
34         b[x]++;
35     }
36     for(int i = 1; i < maxn; i++){
37         int cnt = 0;
38         for(int j = i; j < maxn; j += i){
39             cnt += b[j];
40         }
41         ans += 1LL * mu[i] * cnt * cnt;
42     }
43     return (ans - b[1]) / 2;
44 }
45
46 //gcd(x, y) == 1的对数, x <= n, y <= m;
47
48 ll solve(int n, int m){
49     if(n > m) swap(n, m);
50     ll ans = 0;
51     for(int i = 1; i <= n; i++){
52         ans += (ll)mu[i] * (n / i) * (m / i);
53     }
54     return ans;
55 }

```

10 几何

10.1 fcy

```

1  #include<bits/stdc++.h>
2  #define mp make_pair
3  #define rep(i,a,b) for(int i=a;i<=b;i++)
4  using namespace std;
5  typedef long long ll;
6  const double inf=1e200;
7  const double eps=1e-12;
8  const double pi=acos(-1.0);
9  const int maxn=1000010;
10 struct point{
11     double x,y;
12     point(){}
13     point(double xx,double yy):x(xx),y(yy){}
14 };
15 struct line{
16     point a;//起点
17     point p;//起点到终点的向量
18     double angle;
19 };
20 struct Circle{
21     point c; double r;
22 };
23 int dcmp(double x){ return fabs(x)<eps?0:(x<0?-1:1);}
24 point operator +(point A,point B) { return point(A.x+B.x,A.y+B.y);}
25 point operator -(point A,point B) { return point(A.x-B.x,A.y-B.y);}
26 point operator *(point A,double p){ return point(A.x*p,A.y*p);}
27 point operator /(point A,double p){ return point(A.x/p,A.y/p);}
28 point rotate(point A,double rad){ //向量的旋转
29     return point(A.x*cos(rad)-A.y*sin(rad), A.x*sin(rad)+A.y*cos(rad));
30 }
31 bool operator ==(const point& a,const point& b) {
32     return dcmp(a.x-b.x)==0&& dcmp(a.y-b.y)==0;
33 }
34 double dot(point A,point B){ return A.x*B.x+A.y*B.y;}
35 double det(point A,point B){ return A.x*B.y-A.y*B.x;}
36 double dot(point O,point A,point B){ return dot(A-O,B-O);}
37 double det(point O,point A,point B){ return det(A-O,B-O);}
38 double length(point A){ return sqrt(dot(A,A));}
39 double dist(point A,point B){ return length(A-B);}
40 double angle(point A,point B){ return acos(dot(A,B)/length(A)/length(B));} //夹角
41 point jiaopoint(point p,point v,point q,point w)
42 { //p+tv q+tw, 点加向量表示直线, 求直线交点
43     //如果是线段, 还应该实现判定是否相离; 必要时判定是否平行
44     point u=p-q;
45     double t=det(w,u)/det(v,w);//如果平行, 除0会有问题
46     return p+v*t;
47 }
48 point lintersect(line A,line B) //直线交点, 同上
49 {
50     point C=A.a-B.a;
51     double t=det(C,B.p)/det(B.p,A.p);
52     return A.a+A.p*t;
53 }
54 point GetCirPoint(point a,point b,point c)
55 {

```



```

56 point p=(a+b)/2; //ab中点
57 point q=(a+c)/2; //ac中点
58 point v=rotate(b-a,pi/2.0),w=rotate(c-a,pi/2.0); //中垂线的方向向量
59 if (dcmp(det(v,w))==0) //平行
60 {
61     if(dcmp(length(a-b)+length(b-c)-length(a-c))==0) return (a+c)/2;
62     if(dcmp(length(b-a)+length(a-c)-length(b-c))==0) return (b+c)/2;
63     if(dcmp(length(a-c)+length(c-b)-length(a-b))==0) return (a+b)/2;
64 }
65 return jiaopoint(p,v,q,w);
66 }
67 point w[maxn];
68 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。
69 {
70     point C=w[1]; double r=0;
71     rep(i,1,N){
72         if(dist(C,w[i])>r) {
73             C=w[i]; r=0;
74             rep(j,1,i-1) {
75                 if(dist(C,w[j])>r) {
76                     C=(w[i]+w[j])/2; r=dist(w[i],w[j])/2;
77                     rep(k,1,j-1) {
78                         if(dist(C,w[k])>r){
79                             C=GetCirPoint(w[i],w[j],w[k]);
80                             r=dist(C,w[i]);
81                         }
82                     }
83                 }
84             }
85         }
86     }
87     printf("%.2lf %.2lf %.2lf\n",C.x,C.y,r);
88 }
89 double area(vector<point>p){ //求面积
90     double ans=0; int sz=p.size();
91     for(int i=1;i<sz-1;i++) ans+=det(p[i]-p[0],p[i+1]-p[0]);
92     return ans/2.0;
93 }
94 double seg(point O,point A,point B){ //返回比例
95     if(dcmp(B.x-A.x)==0) return (O.y-A.y)/(B.y-A.y);
96     return (O.x-A.x)/(B.x-A.x);
97 }
98 vector<point>pp[110];
99 pair<double,int>s[110*60];
100 double polyunion(vector<point>*p,int N){ //需要这些点是顺时针，N个多边形。
101     double res=0;
102     for(int i=0;i<N;i++){
103         int sz=p[i].size();
104         for(int j=0;j<sz;j++){
105             int m=0;
106             s[m++]=mp(0,0);
107             s[m++]=mp(1,0);
108             point a=p[i][j],b=p[i][(j+1)%sz];
109             for(int k=0;k<N;k++){
110                 if(i!=k){
111                     int sz2=p[k].size();
112                     for(int ii=0;ii<sz2;ii++){
113                         point c=p[k][ii],d=p[k][(ii+1)%sz2];
114                     }
115                 }
116             }
117         }
118     }
119 }

```

```

115         int c2=dcmp(det(b-a,d-a));
116         if(c1==0&& c2==0){
117             if(dcmp(dot(b-a,d-c)){
118                 s[m++]=mp(seg(c,a,b),1);
119                 s[m++]=mp(seg(c,a,b),-1);
120             }
121         }
122         else{
123             double s1=det(d-c,a-c);
124             double s2=det(d-c,b-c);
125             if(c1>=0&& c2<0) s[m++]=mp(s1/(s1-s2),1);
126             else if(c1<0&& c2>=0) s[m++]=mp(s1/(s1-s2),-1);
127         }
128     }
129 }
130 }
131 sort(s,s+m);
132 double pre=min(max(s[0].first,0.0),1.0),now,sum=0;
133 int cov=s[0].second;
134 for(int j=1;j<m;j++){
135     now=min(max(s[j].first,0.0),1.0);
136     if(!cov) sum+=now-pre;
137     cov+=s[j].second;
138     pre=now;
139 }
140 res+=det(a,b)*sum;
141 }
142 }
143 return res/2;
144 }
145 void CirinterCir(point a,double r1,point b,double r2) //求圆和圆的交点
146 {
147     //记得提前判断是否相交，如果相交就利用角度就可以求了
148     double L=dist(a,b);
149     if(L>r1+r2) return ;//相离
150     if(L+r1<r2||L+r2<r1) return ;//包含
151     double t=acos((r1*r1+L*L-r2*r2)/(2.0*r1*L));
152     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
153     double ang1=base+t,ang2=base-t;
154     point A=a+point{r1*cos(ang1),r1*sin(ang1)};
155     point B=a+point{r1*cos(ang2),r1*sin(ang2)};
156 }
157 bool cmp1(point a,point b){ return a.x==b.x?a.y<b.y:a.x<b.x; } //x排序
158 void convexhull(point *a,int n,point *ch,int &top) //凸包
159 {
160     sort(a+1,a+n+1,cmp1);//x排序
161     top=0;
162     for(int i=1;i<=n;i++){ //下凸包
163         while(top>=2&&det(ch[top-1],ch[top],a[i])<=0) top--;
164         ch[++top]=a[i];
165     }
166     int ttop=top;
167     for(int i=n-1;i>=1;i--){ //上凸包
168         while(top>ttop&&det(ch[top-1],ch[top],a[i])<=0) top--;
169         ch[++top]=a[i];
170     }
171 }
172 double rotating_calipers(point p[],int top) //求最远的点距离
173 {

```

```

174 double ans=0; int now=2;
175 rep(i,1,top-1){ //先求出凸包，然后凸包上旋转卡壳
176     while(det(p[i],p[i+1],p[now])<det(p[i],p[i+1],p[now+1])){
177         now++; //最远距离对应了最大面积。
178         if(now==top) now=1;
179     }
180     ans=max(ans,dist(p[now],p[i]));
181 }
182 return ans;
183 }
184 point ch[2000000],p[2000000];
185 double getangle(point a){ return atan2(a.y,a.x);}
186 double getangle(line a){ return getangle(a.p);}
187 point ss[maxn]; line t[maxn],q[maxn]; int head,tail;
188 bool cmp2(line a,line b){ //方向的极角排序
189     double A=getangle(a),B=getangle(b);
190     point t=(b.a+b.p)-a.a;
191     if(fabs(A-B)<eps) return det(a.p,t)>=0.0;
192     return A<B;
193 }
194 bool onright(line P,line a,line b)
195 {
196     point o=llintersect(a,b);
197     point Q=o-P.a;
198     return det(Q,P.p)>0; //如果同一直线上不能相互看到，则>=0
199 }
200 bool halfplaneintersect(int N)
201 {
202     ss[N+1]=ss[1];
203     rep(i,1,N) t[i].a=ss[i], t[i].p=ss[i+1]-ss[i];
204     sort(t+1,t+N+1,cmp2);
205     int tot=0;
206     rep(i,1,N-1) {
207         if(fabs(getangle(t[i])-getangle(t[i+1]))>eps)
208             t[++tot]=t[i];
209     }
210     t[++tot]=t[N]; head=tail=0;
211     rep(i,1,tot){
212         while(tail>head+1&&onright(t[i],q[tail],q[tail-1])) tail--;
213         while(tail>head+1&&onright(t[i],q[head+1],q[head+2])) head++;
214         q[++tail]=t[i];
215     }
216     while(tail>head+1&&onright(t[head+1],q[tail],q[tail-1])) tail--;return tail-head>2;
217 }
218 double TriAngleCircleInsection(Circle C, point A, point B) //圆与多边形面积交
219 {
220     //a[N+1]=a[1]; ans=0; 拆成多个三角形，求向量面积核
221     //rep(i,1,N) ans+=TriAngleCircleInsection(C,a[i],a[i+1]);
222     point OA=A-C.c,OB=B-C.c;
223     point BA=A-B, BC=C.c-B;
224     point AB=B-A, AC=C.c-A;
225     double DOA=length(OA),DOB=length(OB),DAB=length(AB),r=C.r;
226     if(dcmp(det(OA,OB))==0) return 0; //，三点一线，不构成三角形
227     if(dcmp(DOA-C.r)<0&&dcmp(DOB-C.r)<0) return det(OA,OB)*0.5; //内部
228     else if(DOB<r&&DOA>=r) //一内一外
229     {
230         double x=(dot(BA,BC)+sqrt(r*r*DAB*DAB-det(BA,BC)*det(BA,BC)))/DAB;
231         double TS=det(OA,OB)*0.5;
232         return asin(TS*(1-x/DAB)*2/r/DOA)*r*r*0.5+TS*x/DAB;

```

```

233 }
234 else if(DOB>=r&&DOA<r)// 一外一内
235 {
236     double y=(dot(AB,AC)+sqrt(r*r*DAB*DAB-det(AB,AC)*det(AB,AC)))/DAB;
237     double TS=det(OA,OB)*0.5;
238     return asin(TS*(1-y/DAB)*2/r/DOB)*r*r*0.5+TS*y/DAB;
239 }
240 else if(fabs(det(OA,OB))>=r*DAB||dot(AB,AC)<=0||dot(BA,BC)<=0)//弧
241 {
242     if(dot(OA,OB)<0){
243         if(det(OA,OB)<0) return (-acos(-1.0)-asin(det(OA,OB)/DOA/DOB))*r*r*0.5;
244         else return (acos(-1.0)-asin(det(OA,OB)/DOA/DOB))*r*r*0.5;
245     }
246     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]
247 }
248 else //弧+三角形
249 {
250     double x=(dot(BA,BC)+sqrt(r*r*DAB*DAB-det(BA,BC)*det(BA,BC)))/DAB;
251     double y=(dot(AB,AC)+sqrt(r*r*DAB*DAB-det(AB,AC)*det(AB,AC)))/DAB;
252     double TS=det(OA,OB)*0.5;
253     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);
254 }
255 }
256 double ltoseg(point p,point a,point b){
257     point t=p-a;
258     if(dot(t,b-a)<=0) return dist(p,a);
259     else if(dot(p-b,a-b)<=0) return dist(p,b);
260     return fabs(det(t,b-a))/dist(a,b);
261 }
262 bool isinside(point a) //O(N)判定是否在（任意多边形）内
263 {
264     //算法描述：首先，对于多边形的水平边不做考虑，其次，
265     //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，
266     //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。
267     int ncross=0; int N;
268     rep(i,0,N-1) {
269         point p1=p[i],p2=p[i+1];
270         if(ltoseg(a,p[i],p[i+1])==0) return true; //在线段上
271         if(p1.y==p2.y) continue; //默认做水平x轴的线，所以水平线不考虑
272         if(a.y<min(p1.y,p2.y)) continue; //相离不考虑
273         if(a.y>max(p1.y,p2.y)) continue;
274         double t=det(a-p[i],a-p[i+1]);
275         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++;
276     }
277     return (ncross&1);
278 }
279 bool check(point A,int top) //二分点在（凸多边形）内
280 {
281     int L=2,R=top-2,Mid;
282     while(L<=R){
283         Mid=(L+R)>>1;
284         if(det(ch[Mid]-ch[1],A-ch[1])<0) R=Mid-1;
285         else {
286             if(det(ch[Mid+1]-ch[1],A-ch[1])<=0&&det(ch[Mid+1]-ch[Mid],A-ch[Mid])>=0)
287                 return true;
288             L=Mid+1;
289         }
290     }
291     return false;

```

```

292 }
293 int main()
294 {
295
296 }

```

10.2 圆

```

1 struct point
2 {
3     double x, y;
4     point();
5     point(double _x, double _y);
6     bool operator<(Point b) const;    // 点左右判断 - line 中使用
7     bool operator==(point b) const;    // 点相等判断
8     point operator+(const point &b) const; // 向量相加
9     point operator-(const point &b) const; // 向量相减
10    point operator*(const double &k) const; // 向量乘法
11    point operator/(const double &k) const; // 向量除法
12    point trunc(double r) const;        // 向量模转换
13    double operator*(const point &b) const; // 向量点积
14    double operator^(const Point &b) const; // 向量叉积
15    double len(bool isSqrt = true);    // 向量模长度, 参数为是否开平方
16    double distance(const point &other); // 点距离
17    point rotleft();                   // 向量绕原点逆时针旋转 90 度
18    point rotright();                  // 向量绕原点顺时针旋转 90 度
19    point rotate(point p, double angle); // 绕 p 点逆时针旋转 angle 度
20    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
21 };
22
23 struct line
24 {
25     point s, e;
26     line();
27     line(point _s, point _e);
28     bool operator==(line v);          // 判断射线重合
29     line(point p, double angle);      // 根据一个点和倾斜角 angle 确定直线, 0 <= angle < pi
30     line(double a, double b, double c); // 根据 ax + by + c = 0 确定直线
31     void adjust();                     // 调整线段
32     double lenth();                    // 求线段长度
33     double angle();                    // 返回线段的倾斜角
34     int relation(point p);             // 点和直线关系
35     bool pointonseg(point p);          // 点在线段上的判断
36     bool parallel(line v);             // 两向量平行 (对应直线平行或重合)
37     int segcrossseg(line v);           // 两线段相交判断
38     int linecrossseg(line v);          // 直线和线段相交判断
39     int linecrossline(line v);         // 两直线关系
40     point crosspoint(line v);          // 求两直线的交点
41     double dispointtoline(point p);    // 点到直线的距离
42     double dispointtoseg(point p);     // 点到线段的距离
43     double dissegtoseg(line v);        // 返回线段到线段的距离
44     Point lineprog(Point p);           // 返回点 p 在直线上的投影
45     Point symmetrypoint(Point p);      // 返回点 p 关于直线的对称点
46 }
47
48 const double eps = 1e-8;
49 const double pi = acos(-1.0);
50

```

```

51 int sgn(double x)
52 {
53     if (fabs(x) < eps)
54         return 0;
55     if (x < 0)
56         return -1;
57     return 1;
58 }
59
60 struct circle
61 {
62     point p; // 圆心
63     double r; // 半径
64     circle() {}
65     circle(point _p, double _r) : p(_p), r(_r) {}
66     // 三点求算外接圆、内切圆
67     // 需要 point 的 + / rotate() 以及 line 的 crosspoint()
68     // bool 变量表示是否为外接圆
69     circle(point a, point b, point c, bool isCircumcircle)
70     {
71         if (isCircumcircle)
72         {
73             line u = line((a + b) / 2, ((a + b) / 2) + ((b - a).rotleft()));
74             line v = line((b + c) / 2, ((a + c) / 2) + ((c - b).rotleft()));
75             p = u.crosspoint(v);
76             r = p.distance(a);
77         }
78         else
79         {
80             line u, v;
81             double m = atan2(b.y - a.y, b.x - a.x), n = atan2(c.y - a.y, c.x - a.x);
82             u.s = a;
83             u.e = u.s + point(cos((n + m) / 2), sin((n + m) / 2));
84             v.s = b;
85             m = atan2(a.y - b.y, a.x - b.x);
86             n = atan2(c.y - b.y, c.x - b.x);
87             p = u.crosspoint(v);
88             r = line(a, b).dispointtoseg(p);
89         }
90     }
91     bool operator==(circle v)
92     {
93         return (p == v.p) && sgn(r - v.r) == 0;
94     }
95     bool operator<(circle v) const
96     {
97         return ((p < v.p) || ((p == v.p) && sgn(r - v.r) < 0));
98     }
99     // 面积
100    double area()
101    {
102        return pi * r * r;
103    }
104    // 周长
105    double circumference()
106    {
107        return 2 * pi * r;
108    }
109    // 点圆关系

```

```
110 // 0 圆外
111 // 1 圆上
112 // 2 圆内
113 int relation(point b)
114 {
115     double dst = b.distance(p);
116     if (sgn(dst - r) < 0)
117         return 2;
118     else if (sgn(dst - r) == 0)
119         return 1;
120     return 0;
121 }
122 // 线段和圆的关系
123 // // 需要 line 的 dispointtoseg
124 int relationseg(line v)
125 {
126     double dst = v.dispointtoseg(p);
127     if (sgn(dst - r) < 0)
128         return 2;
129     else if (sgn(dst - r) == 0)
130         return 1;
131     return 0;
132 }
133 // 直线和圆的关系
134 // // 需要 line 的 dispointtoline
135 int relationline(line v)
136 {
137     double dst = v.dispointtoline(p);
138     if (sgn(dst - r) < 0)
139         return 2;
140     else if (sgn(dst - r) == 0)
141         return 1;
142     return 0;
143 }
144 // 两圆的关系
145 // 5 相离
146 // 4 外切
147 // 3 相交
148 // 2 内切
149 // 1 内含
150 // 需要 point 的 distance
151 int relationcircle(circle v)
152 {
153     double d = p.distance(v.p);
154     if (sgn(d - r - v.r) > 0)
155         return 5;
156     if (sgn(d - r - v.r) == 0)
157         return 4;
158     double l = fabs(r - v.r);
159     if (sgn(d - r - v.r) < 0 && sgn(d - l) > 0)
160         return 3;
161     if (sgn(d - l) == 0)
162         return 2;
163     if (sgn(d - l) < 0)
164         return 1;
165 }
166 // 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
167 // 需要 relationcircle
168 int pointcrosscircle(circle v, point &p1, point &p2)
```

```

169 {
170     int rel = relationcircle(v);
171     if (rel == 1 || rel == 5)
172         return 0;
173     double d = p.distance(v.p);
174     double l = (d * d + r * r - v.r * v.r) / (2 * d);
175     double h = sqrt(r * r - l * l);
176     point tmp = p + (v.p - p).trunc(l);
177     p1 = tmp + ((v.p - p).rotleft().trunc(h));
178     p2 = tmp + ((v.p - p).rotright().trunc(h));
179     if (rel == 2 || rel == 4)
180         return 1;
181     return 2;
182 }
183 // 求直线和圆的交点，返回交点个数
184 int pointcrossline(line v, point &p1, point &p2)
185 {
186     if (!(*this).relationline(v))
187         return 0;
188     point a = v.lineprog(p);
189     double d = v.dispointtoline(p);
190     d = sqrt(r * r - d * d);
191     if (sgn(d) == 0)
192     {
193         p1 = a;
194         p2 = a;
195         return 1;
196     }
197     p1 = a + (v.e - v.s).trunc(d);
198     p2 = a - (v.e - v.s).trunc(d);
199     return 2;
200 }
201 // 得到过 a,b 两点，半径为 r1 的两个圆
202 // 需要 pointcrosscircle
203 int gercircle(point a, point b, double r1, circle &c1, circle &c2)
204 {
205     circle x(a, r1), y(b, r1);
206     int t = x.pointcrosscircle(y, c1.p, c2.p);
207     if (!t)
208         return 0;
209     c1.r = c2.r = r;
210     return t;
211 }
212 // 得到与直线 u 相切，过点 q, 半径为 r1 的圆
213 int getcircle(line u, point q, double r1, circle &c1, circle &c2)
214 {
215     double dis = u.dispointtoline(q);
216     if (sgn(dis - r1 * 2) > 0)
217         return 0;
218     if (sgn(dis) == 0)
219     {
220         c1.p = q + ((u.e - u.s).rotleft().trunc(r1));
221         c2.p = q + ((u.e - u.s).rotright().trunc(r1));
222         c1.r = c2.r = r1;
223         return 2;
224     }
225     line u1 = line((u.s + (u.e - u.s).rotleft().trunc(r1)), (u.e + (u.e - u.s).rotleft().trunc(r1)));
226     line u2 = line((u.s + (u.e - u.s).rotright().trunc(r1)), (u.e + (u.e - u.s).rotright().trunc(r1)));
227     circle cc = circle(q, r1);

```



```

228     point p1, p2;
229     if (!cc.pointcrossline(u1, p1, p2))
230         cc.pointcrossline(u2, p1, p2);
231     c1 = circle(p1, r1);
232     if (p1 == p2)
233     {
234         c2 = c1;
235         return 1;
236     }
237     c2 = circle(p2, r1);
238     return 2;
239 }
240 // 同时与直线 u,v 相切, 半径为 r1 的圆
241 int getcircle(line u, line v, double r1, circle &c1, circle &c2, circle &c3, circle &c4)
242 {
243     if (u.parallel(v))
244         return 0;
245     line u1 = line(u.s + (u.e - u.s).rotleft().trunc(r1), u.e + (u.e - u.s).rotleft().trunc(r1));
246     line u2 = line(u.s + (u.e - u.s).rotright().trunc(r1), u.e + (u.e - u.s).rotright().trunc(r1));
247     line u3 = line(v.s + (v.e - v.s).rotleft().trunc(r1), v.e + (v.e - v.s).rotleft().trunc(r1));
248     line u4 = line(v.s + (v.e - v.s).rotright().trunc(r1), v.e + (v.e - v.s).rotright().trunc(r1));
249     c1.r = c2.r = c3.r = c4.r = r1;
250     c1.p = u1.crosspoint(v1);
251     c2.p = u1.pointonseg(v2);
252     c3.p = u2.pointonseg(v1);
253     c4.p = u2.pointonseg(v2);
254     return 4;
255 }
256 // 同时与不相交圆 cx,cy 相切, 半径为 r1 的圆
257 int getcircle(circle cx, circle cy, double r1, circle &c1, circle &c2)
258 {
259     circle x(cx.p, r1 + cx.r), y(cy.p, r1 + cy.r);
260     int t = x.pointcrosscircle(y, c1.p, c2.p);
261     if (!t)
262         return 0;
263     c1.r = c2.r = r1;
264     return t;
265 }
266 int tangentline(point q, line &u, line &v)
267 {
268     int x = relation(q);
269     if (x == 2)
270         return 0;
271     if (x == 1)
272     {
273         u = line(q, q + (q - p).rotleft());
274         v = u;
275         return 1;
276     }
277     double d = p.distance(q);
278     double l = r * r / d;
279     double h = sqrt(r * r - l * l);
280     u = line(q, p + ((q - p).trunc(l) + (q - p).rotleft().trunc(h)));
281     v = line(q, p + ((q - p).trunc(l) + (q - p).rotright().trunc(h)));
282     return 2;
283 }
284 // 求两圆相交的面积
285 double areacircle(circle v)
286 {

```

```

287     int rel = relationcircle(v);
288     if (rel >= 4)
289         return 0.0;
290     if (rel <= 2)
291         return min(area(), v.area());
292     double d = p.distance(v.p);
293     double hf = (r + v.r + d) / 2.0;
294     double ss = 2 * sqrt(hf * (hf - r) * (hf - v.f) * (hf - d));
295     double a1 = acos((r * r + d * d - v.r * v.r) / (2.0 * r * d));
296     a1 = a1 * r * r;
297     double a2 = acos((v.r * v.r + d * d - r * r) / (2.0 * v.r * d));
298     a2 = a2 * v.r * v.r;
299     return a1 + a2 - ss;
300 }
301 // 求圆和三角形 pab 的相交面积
302 double areatriangle(Point a, Point b)
303 {
304     if (sgn((p - a) ^ (p - b)) == 0)
305         return 0.0;
306     point q[5];
307     int len = 0;
308     q[len++] = a;
309     line l(a, b);
310     point p1, p2;
311     if (pointcrossline(l, q[1], q[2]) == 2)
312     {
313         if (sgn((a - q[1]) * (b - q[1])) < 0)
314             q[len++] = q[1];
315         if (sgn((a - q[2]) * (b - q[2])) < 0)
316             q[len++] = q[2];
317     }
318     q[len++] = b;
319     if (len == 4 && sgn((q[0] - q[1]) * (q[2] - q[1])) > 0)
320         swap(q[1], q[2]);
321     double res = 0;
322     for (int i = 0; i < len - 1; i++)
323     {
324         if (relation(q[i]) == 0 || relation(q[i + 1]) == 0)
325         {
326             double arg = p.rad(q[i], q[i + 1]);
327             res += r * r * arg / 2.0;
328         }
329         else
330             res += fabs((q[i] - p) ^ (q[i + 1] - p)) / 2.0;
331     }
332     return res;
333 }
334 };

```

10.3 体积

```

1  #include <iostream>
2  #include <cmath>
3
4  using namespace std;
5
6  const double pi = acos(-1.0);
7

```

```

8 //圆锥体体积公式  $V = 1/3 * S * h$ , S是底面积, h是高
9 double volumn_Cone(double r, double h){
10     return 1.0 / 3 * r * r * pi * h;
11 }
12
13 //三棱锥体积公式
14 //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).
15 //O为原点, 则三棱锥O-ABC体积为
16 // $V = 1.0 / 6 * \text{abs}(a1 * b2 * c3 + b1 * c2 * a3 + c1 * a2 * b3 - a1 * c2 * b3 - b1 * a2 * c3 - c1 * b2 * a3)$ ;
17 struct Point{
18     double x;
19     double y;
20     double z;
21 };
22
23 double volumn_a(Point a, Point b, Point c){
24     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);
25 }
26
27 //椭圆在xyz-笛卡尔坐标系中的标准方程是: $(x - x0)^2 / a^2 + (y - y0)^2 / b^2 + (z - z0)^2 / c^2 = 1$ 
28 //体积是 $V = 4 / 3 * \pi * a * b * c$ ;
29 double volumn_Ellipse(double a, double b, double c){
30     return 4.0 / 3 * pi * a * b * c;
31 }
32
33 //圆台的体积
34 double volumn_RoundTable(double R, double r, double h){
35     return pi * h / 3.0 * (R * R + r * r + R * r);
36 }
37
38 //球缺
39 double volumn_MissingBall(double h, double r){
40     return pi * h * h * (r - h / 3.0);
41 }
42
43 //交叉圆柱体的体积
44 double volumn_CrossCylinder(double h1, double h2, double r){
45     return pi * r * r * (h1 + h2 - 2.0 / 3 * r);
46 }
47
48 //梯形体的体积
49 double volumn_TrapezoidalBody(double a, double b, double h1, double a1, double b1){
50     return h / 6 * ((2 * a + a1) * b + (2 * a1 + a) * b1);
51     //或者return h / 6 * (a * b + (a + a1) * (b + b1) + a1 * b1);
52 }

```

10.4 二维凸包

```

1 //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
2 //输入不能有重复节点
3 //如果精度要求搞需要用dcmp判断
4 //如果不希望在边上右点, 需要将 <= 改为 <
5 int ConvexHull(Point *p, int n, Point *ch)
6 {
7     sort(p, p+n);
8     int m = 0;
9     for(int i = 0; i < n; ++i)
10     {

```

```

11     while(m>1&& Cross(ch[m-1]-ch[m-2],p[i]-ch[m-2])<=0) m--;
12     ch[m++] = p[i];
13
14 }
15 int k = m;
16 for(int i = n-2; i >= 0; --i)
17 {
18     while(m > k&& Cross(ch[m-1]-ch[m-2],p[i]-ch[m-2]) <= 0) m--;
19     ch[m++] = p[i];
20 }
21 if(n > 1) m--;
22 return m;
23 }

```

10.5 二维几何模板

```

1  #include <bits/stdc++.h>
2  #define mem(ar,num) memset(ar,num,sizeof(ar))
3  #define me(ar) memset(ar,0,sizeof(ar))
4  #define lowbit(x) (x&(-x))
5  #define forn(i,n) for(int i = 0; i < n; ++i)
6  using namespace std;
7  typedef long long LL;
8  typedef unsigned long long ULL;
9  const int prime = 999983;
10 const int INF = 0x7FFFFFFF;
11 const LL INFF = 0x7FFFFFFFFFFFFFFF;
12 const double pi = acos(-1.0);
13 const double inf = 1e18;
14 const double eps = 1e-10;
15 const LL mod = 1e9 + 7;
16 struct Point
17 {
18     double x,y;
19
20     Point(double x = 0,double y = 0):x(x),y(y) {}
21
22 };
23 typedef Point Vector;
24 Vector operator + (Vector A,Vector B)
25 {
26     return Vector(A.x + B.x,A.y + B.y);
27 }
28 Vector operator - (Vector A,Vector B)
29 {
30     return Vector(A.x-B.x,A.y-B.y);
31 }
32 Vector operator / (Vector A,double p)
33 {
34     return Vector(A.x/p,A.y/p);
35 }
36 Vector operator * (Vector A,double p)
37 {
38     return Vector(A.x*p,A.y*p);
39 }
40 double angle(Vector v)//求向量的角度从0到2*pi
41 {
42     return atan2(v.y,v.x);

```

```
43 }
44 int dcmp(double x)
45 {
46     if(fabs(x)<eps)
47         return 0;
48     else
49         return x < 0?-1:1;
50 }
51 bool operator < (const Point &a,const Point &b)
52 {
53     if(dcmp(a.x-b.x)==0)
54         return a.y<b.y;
55     else
56         return a.x<b.x;
57 }
58
59
60 bool operator == (const Point &a,const Point &b)
61 {
62     return !dcmp(a.x-b.x)&&!dcmp(a.y-b.y);
63 }
64 double Dot(Vector A,Vector B)
65 {
66     return A.x*B.x+A.y*B.y;
67 }
68 double Length(Vector A)
69 {
70     return sqrt(A.x*A.x+A.y*A.y);
71 }
72 double Angle(Vector A,Vector B)
73 {
74     return acos(Dot(A,B)/Length(A)/Length(B));
75 }
76 double Cross(Vector A,Vector B)
77 {
78     return A.x*B.y - A.y*B.x;
79 }
80 double Area2(Point A,Point B,Point C)
81 {
82     return Cross(B-A,C-A);
83 }
84 Vector Rotate(Vector A,double rad)
85 {
86     return Vector (A.x*cos(rad)-A.y*sin(rad),A.x*sin(rad)+A.y*cos(rad));
87 }
88 Vector Normal(Vector A)//单位法线
89 {
90     double L = Length(A);
91     return Vector(-A.y/L,A.x/L);
92 }
93 //调用前确保直线有唯一交点，当且仅当Cross(v,w)非0
94 Point Get_Line_Intersection(Point P,Vector v,Point Q,Vector w)
95 {
96     Vector u = P - Q;
97     double t = Cross(w,u)/Cross(v,w);
98     return P+v*t;
99 }
100 double Distance_To_Line(Point P,Point A,Point B)//点到直线的距离
101 {
```

```

102   Vector v1 = B-A,v2 = P-A;
103   return fabs(Cross(v1,v2)/Length(v1));
104 }
105 double Distance_To_Segment(Point P,Point A,Point B)
106 {
107     if(A==B)
108         return Length(P-A);
109     Vector v1 = B-A,v2 = P-A,v3 = P-B;
110     if(dcmp(Dot(v1,v2))<0)
111         return Length(v1);
112     else if(dcmp(Dot(v1,v3))>0)
113         return Length(v3);
114     else
115         return fabs(Cross(v1,v2))/Length(v1);
116 }
117 Point Get_Line_Projection(Point P,Point A,Point B)//求投影点
118 {
119     Vector v = B- A;
120     return A + v*(Dot(v,P-A)/Dot(v,v));
121 }
122 //线段相交判定 相交不在线段的端点
123 bool Segment_Proper_Intersection(Point a1,Point a2,Point b1,Point b2)
124 {
125     double c1 = Cross(a2-a1,b1-a1),c2 = Cross(a2-a1,b2-a1),
126     c3 = Cross(b2-b1,a2-b1),c4 = Cross(b2-b1,a1-b1);
127     return dcmp(c1)*dcmp(c2)<0&& dcmp(c3)*dcmp(c4)<0;
128 }
129 //判断点是否在线段上(不包括端点)
130 bool Onsegment(Point p,Point a1,Point a2)
131 {
132     return dcmp(Cross(a1-p,a2-p))==0&& dcmp(Dot(a1-p,a2-p))<0;
133 }

```

10.6 最小三角形

```

1 // 最小三角形，注意longlong
2 const int maxn = 5000+10;
3 struct Point{
4
5     long long x,y;
6     Point(long long xx = 0,long long yy = 0):x(xx),y(yy){}
7 };
8 typedef Point Vector ;
9
10 Point operator - (const Point A,const Point B){
11     return Point(A.x-B.x,A.y-B.y);
12 }
13 long long Cross(Vector A,Vector B){
14     return A.x*B.y-A.y*B.x;
15 }
16 Point A[maxn],B[maxn];
17 bool operator <(const Point &A,const Point &B){
18
19     return A.y*B.x < A.x*B.y;
20 }
21 int main(void)
22 {
23     int N;cin>>N;

```

```
24  for(int i = 0;i < N; ++i){
25      cin>>A[i].x>>A[i].y;
26  }
27  double ans = inf;
28  for(int i = 0;i < N; ++i){
29      int t = 0;
30      for(int j = 0;j < N; ++j)
31          if(i != j)
32              B[t++] = A[j]-A[i];
33      sort(B,B+t);
34      // assert(t == N-1);
35      for(int j = 0;j < t-1; ++j){
36          ans = min(ans,fabs(Cross(B[j],B[j+1]))/2.0);
37      }
38  }
39  printf("%.3F",ans);
40
41  return 0;
42 }
```

10.7 最大三角形

```
1  // 最大三角形
2  const int N=50005;
3
4  struct Point
5  {
6      double x,y;
7  };
8
9  Point stack[N];
10 Point p[N];
11 Point MinA;
12
13 int top;
14
15 double dist(Point A,Point B)
16 {
17     return sqrt((A.x-B.x)*(A.x-B.x)+(A.y-B.y)*(A.y-B.y));
18 }
19
20 double cross(Point A,Point B,Point C)
21 {
22     return (B.x-A.x)*(C.y-A.y)-(B.y-A.y)*(C.x-A.x);
23 }
24
25 bool cmp(Point a,Point b)
26 {
27     double k=cross(MinA,a,b);
28     if(k>0) return 1;
29     if(k<0) return 0;
30     return dist(MinA,a)<dist(MinA,b);
31 }
32
33 void Graham(int n)
34 {
35     int i;
36     for(i=1; i<n; i++)
```

```
37     if(p[i].y<p[0].y||(p[i].y==p[0].y&& p[i].x<p[0].x))
38         swap(p[i],p[0]);
39     MinA=p[0];
40     sort(p+1,p+n,cmp);
41     stack[0]=p[0];
42     stack[1]=p[1];
43     top=1;
44     for(i=2; i<n; i++)
45     {
46         while(cross(stack[top-1],stack[top],p[i])<=0&& top>=1) --top;
47         stack[++top]=p[i];
48     }
49 }
50
51 double rotating_calipers(int n)
52 {
53     int j=1,k=0;
54     double ans=0;
55     for(int i=0;i<n;i++)
56     {
57         j=(i+1)%n;
58         k=(j+1)%n;
59         while(fabs(cross(stack[i],stack[j],stack[k]))<fabs(cross(stack[i],stack[j],stack[(k+1)%n])))
60             k=(k+1)%n;
61         while(j!=i&& k!=i)
62         {
63             ans=max(ans,fabs(cross(stack[i],stack[j],stack[k])));
64             while(fabs(cross(stack[i],stack[j],stack[k]))<fabs(cross(stack[i],stack[j],stack[(k+1)%n])))
65                 k=(k+1)%n;
66             j=(j+1)%n;
67         }
68     }
69     return ans*0.5;
70 }
71
72 int main()
73 {
74     int n;
75     while(~scanf("%d",&n))
76     {
77         if(n==1) break;
78         for(int i=0;i<n;i++)
79             scanf("%lf%lf",&p[i].x,&p[i].y);
80         if(n<3)
81         {
82             puts("0.00");
83             continue;
84         }
85         Graham(n);
86         top++;
87         if(top<3)
88         {
89             puts("0.00");
90             continue;
91         }
92         if(top==3)
93         {
94             printf("%.2lf\n",fabs(cross(stack[0],stack[1],stack[2]))/2);
95             continue;
96         }
97     }
```



```

96     }
97     printf("%.2lf\n",rotating_calipers(top));
98 }
99 return 0;
100 }

```

10.8 三维凸包

```

1  struct Face{
2      int v[3];
3      Vector3 normal(Vector *P)
4      {
5          return Cross(P[v[1]]-P[v[0]],P[v[2]]-P[v[0]]);
6      }
7      int cansee(Point *P,int i)const
8      {
9          return Dot(P[i]-P[v[0]],normal(P)) > 0?1 : 0;
10     }
11 };
12 vector <Face> CH3D(Point3* P,int n)
13 {
14     vector <Face> cur;
15     cur.push_back((Face){0,1,2});
16     cur.push_back((Face){2,1,0});
17     for(int i = 3;i < n; ++i)
18     {
19         vector<Face> next;
20         //计算每条边 左面 的可见性
21         for(int j= 0;j < cur.size(); ++j)
22         {
23             Face &f = cur[j];
24             int res = f.cansee(P,i);
25             if(!res) next.push_back(f);
26             for(int k = 0;k < 3; ++k)
27                 vis[f.v[k]][f.v[(k+1)%3]] = res;
28         }
29         for(int j = 0;j < cur.size(); ++j)
30         {
31             for(int k = 0;k < 3; ++k)
32             {
33                 int a = cur[j].v[k],b = cur[j].v[(k+1)%3];
34                 if(vis[a][b] != vis[b][a]&&vis[a][b])//(a,b) 是分界线 , 左边对P[i] 可见
35                     next.push_back((Face){a,b,i});
36             }
37         }
38         cnr = next;
39     }
40     return cur;
41 }
42 double rand01() {return rand() / (double) RAND_MAX;}//0-1 的随机数
43 double randeps() {return (rand01()-0.5) * eps;}
44 Point3 add_noise(Point3 p)
45 {
46     return Point3(p.x + randeps(),p.y+randeps(),p.z+randeps());
47 }
48
49 //.....
50 struct Face{

```

```

51  int v[3];
52  Vector3 normal(Vector *P)
53  {
54      return Cross(P[v[1]]-P[v[0]],P[v[2]]-P[v[0]]);
55  }
56  int cansee(Point *P,int i)const
57  {
58      return Dot(P[i]-P[v[0]],normal(P)) > 0?1 : 0;
59  }
60 };
61 vector <Face> CH3D(Point3* P,int n)
62 {
63     vector <Face> cur;
64     cur.push_back((Face){0,1,2});
65     cur.push_back((Face){2,1,0});
66     for(int i = 3;i < n; ++i)
67     {
68         vector<Face> next;
69         //计算每条边 左面 的可见性
70         for(int j= 0;j < cur.size(); ++j)
71         {
72             Face &f = cur[j];
73             int res = f.cansee(P,i);
74             if(!res) next.push_back(f);
75             for(int k = 0;k < 3; ++k)
76                 vis[f.v[k]][f.v[(k+1)%3]] = res;
77         }
78         for(int j = 0;j < cur.size(); ++j)
79         {
80             for(int k = 0;k < 3; ++k)
81             {
82                 int a = cur[j].v[k],b = cur[j].v[(k+1)%3];
83                 if(vis[a][b] != vis[b][a]&&vis[a][b])//(a,b) 是分界线 , 左边对P[i] 可见
84                     next.push_back((Face){a,b,i});
85             }
86         }
87         cur = next;
88     }
89     return cur;
90 }
91 double rand01() {return rand() / (double) RAND_MAX;}//0-1 的随机数
92 double randeps() {return (rand01()-0.5) * eps;}
93 Point3 add_noise(Point3 p)
94 {
95     return Point3(p.x + randeps(),p.y+randeps(),p.z+randeps());
96 }

```

10.9 三维几何模板

```

1  #include <bits/stdc++.h>
2  const double eps = 1e-6;
3  using namespace std;
4
5  struct Point3
6  {
7      double x,y,z;
8      Point3(double x = 0,double y = 0,double z = 0):x(x),y(y),z(z) {}
9  };

```

```
10 typedef Point3 Vector3;
11 int dcmp(double d)
12 {
13     if(fabs(d)< eps)
14         return 0;
15     else
16         return d < 0?-1:1;
17 }
18 Vector3 operator +(Vector3 v1,Vector3 v2)
19 {
20     return Vector3(v1.x+v2.x,v1.y+v2.y,v1.z+v2.z);
21 }
22 Vector3 operator -(Vector3 v1,Vector3 v2)
23 {
24     return Vector3(v1.x-v2.x,v1.y-v2.y,v1.z-v2.z);
25 }
26 Vector3 operator *(Vector3 v,double c)
27 {
28     return Vector3(v.x*c,v.y*c,v.z*c);
29 }
30 Vector3 operator /(Vector3 v,double c)
31 {
32     return Vector3(v.x/c,v.y/c,v.z/c);
33 }
34 bool operator ==(Point3 A,Point3 B)
35 {
36     return !dcmp(A.x-B.x)&&!dcmp(A.y-B.y)&&!dcmp(A.z-B.z);
37 }
38 double Dot(Vector3 A,Vector3 B)
39 {
40     return A.x*B.x+A.y*B.y+A.z*B.z;
41 }
42 double Length(Vector3 A)
43 {
44     return sqrt(Dot(A,A));
45 }
46 double Angle(Vector3 A,Vector3 B)//求两向量的夹角
47 {
48     return acos(Dot(A,B)/(2*Length(A)*Length(B)));
49 }
50 double DistanceToPlane(const Point3 &p,const Point3 &p0,const Vector3&n)//
51 {
52     return fabs(Dot(p-p0,n))/Length(n);
53 }
54 Point3 GetPlaneProjection(const Point3&p,const Point3&p0,const Vector3&n)
55 {
56     return p-n*Dot(p-p0,n);
57 }
58 //直线p1-p2 到平面p0-n的交点。 假定交点唯一存在
59 Point3 LinePlaneIntersection(Point3 p1,Point3 p2,Point3 p0,Vector3 n)
60 {
61     Vector3 v= p2 - p1;
62     // /*if(dcmp(Dot(v,n))==0)
63     // {
64     //     if(dcmp(Dot(p1-p0,n))==0)
65     //         直线在平面上
66     //     else
67     //         直线与平面平行
68     // }
```

```

69 // */
70 double t = Dot(n,p0-p1)/Dot(n,p2-p1);
71 return p1 + v*t;
72 }
73 Point3 LinePlaneIntetsection(Point3 p1,Point3 p2,double A,double B,double C,double D)
74 {
75     Vector3 v = p2-p1;
76     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));
77     return p1 + v*t;
78 }
79 Vector3 Cross(Vector3 A,Vector3 B)
80 {
81     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);
82 }
83 double Area2(Point3 A,Point3 B,Point3 C)
84 {
85     return Length(Cross(B-A,C-A));
86 }
87 ////已知平面的三点,求出点法式
88 //Vector3 Solven(Point3 A,Point3 B,Point3 C)
89 //{
90 //    return Cross(B-A,C-A);
91 //}
92 //判断一个点是否在三角形内, 可以用面积法
93 bool PointInTri(Point3 P,Point3 A,Point3 B,Point3 C)
94 {
95     double area1 = Area2(P,A,B);
96     double area2 = Area2(P,A,C);
97     double area3 = Area2(P,B,C);
98     double area4 = Area2(A,B,C);
99     return dcmp(area1+area2+area3-area4)==0;
100 }
101 //判断线段是否与三角形相交
102 bool TriSegIntersection(Point3 P0,Point3 P1,Point3 P2,Point3 A,Point3 B,Point3 &P)
103 {
104     Vector3 n = Cross(P1-P0,P2-P0);
105
106     if(dcmp(Dot(n,B-A))==0)
107         return false;
108
109     double t = Dot(n,P0-A)/Dot(n,B-A);
110     if(dcmp(t) < 0 || dcmp(t-1) > 0)
111         return false;
112     P = A + (B-A) * t;
113     return PointInTri(P,P0,P1,P2);
114 }
115 double DitantceToLine(Point3 P,Point3 A,Point3 B)
116 {
117     return Length(Cross(A-P,B-P))/Length(A-B);
118 }
119 double DistanceToSegment(Point3 P,Point3 A,Point3 B)
120 {
121     if(A==B) return Length(P-A);
122     Vector3 v1 = B - A, v2 = P - A,v3 = P-B;
123     if(dcmp(Dot(v1,v2)) == 0) return Length(v2);
124     if(dcmp(Dot(v1,v3)) > 0) return Length(v3);
125     return Length(Cross(v1,v2))/Length(v1);
126 }
127 double Volume6(Point3 A,Point3 B,Point3 C,Point3 D)

```

```
128 {
129     return Dot(D-A,Cross(B-A,C-A));
130 }
131 //
132 int main(void)
133 {
134
135     Point3 A(0,0,0),B(0,100,0),C(100,0,0),D(25,25,0);
136     cout<<PointInTri(D,A,B,C)<<endl;
137     return 0;
138 }
```