

ACM/ICPC/CCPC Template

浙江工商大学 ZJSU

ZYF

November 14, 2019

Contents

U	prime	1
).1 素数筛选 (埃拉托色尼筛选)	
).2 判断素数	
).3 米勒罗宾素数	
).4 区间素数筛选	
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	
	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	
	3.2 exbsgs	8
	7-11-7 WL	
4	欧拉函数	11
	4.1 欧拉函数	11
E	组合数学	13
J	5.1 lucas	
	5.2 exlucas	
	5.3 Big Combination	
	5.4 组合数学初始化 (杨辉三角)	
	5.5 组合数学初始化 (阶乘的逆元)	
	5.6 第一斯特林数	
	5.7 卡特兰数	17
c	异或	19
o		
	5.1 线性基	19
7	矩阵	21
•	~~~~ 7.1 斐波那切数列十进制	
	.1	21
8	others	23
	8.1 ax+by=c exgcd	23
	8.2 formula	
	3.3 万进制	
	8.4 定积分	
	8.5 用树状数组求逆序对	
	8.6 线性基	
	3.7 大数板子 (易)	
	3.8 大数板子 (完整)	
	3.9 快速乘	
	8.10 母函数	
	3.11 合数分解	34
o	未学会	36
ð	↑子云).1 莫比乌斯反演	
	/-1 大山一州区/宍	

ACM/ICPC/CCPC Template, ZYF

几何																										37
10.1	fcy																					 				37
10.2																										
10.3	体积	l																				 				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
    * 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。
    * notprime 是一张表,为 false 表示是素数, true 表示不是素数
5
   const int MAXN = 1000010;
6
   bool notprime[MAXN]; // 值为 false 表示素数,值为 true 表示非素数
7
8
   void init()
9
10
     memset(notprime, false, sizeof(notprime));
     notprime[0] = notprime[1] = true;
11
     for (int i = 2; i < MAXN; i++)
12
       if (!notprime[i])
13
       {
14
15
         if(i > MAXN/i)
16
           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
17
         for (int j = i * i; j < MAXN; j += i)
18
           notprime[j] = true;
19
       }
20
21 }
```

0.2 判断素数

```
bool judge(int x){
 1
       float n_sqrt;
 2
       if(x == 2 || x == 3){}
 3
         return true;
 4
 5
       if(x % 6!= 1 && x % 6!= 1){
 6
 7
         return false:
 8
       n_sqrt = floor(sqrt(float(x)));
 9
10
       for(int i = 5; i <= n_sqrt; i += 6){</pre>
         if(x \% i == 0 || x \% (i + 2) == 0){}
11
            return false;
12
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);
```

```
w = w * w % n;
11
       if (m & 1) {
12
13
         w = w * a % n;
14
15
       return w;
16 }
17
    bool Miller_Rabin(int n) {
18
19
       if (n == 2) {
20
         return true;
21
22
       for (int i = 0; i < maxn; i++) {
23
         int a = rand() \% (n - 2) + 2;
         if (modular_exp(a, n, n) != a) {
24
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
   //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;
    const int maxn = "Edit";
5
    bool is_prime_small[maxn], is_prime[maxn];
6
    ll prime[maxn];
7
8
    int segment_sieve(ll a, ll b){
9
10
      int tot = 0;
       for(ll i = 0; i * i < b; i++) is_prime_small[i] = true;</pre>
11
      for(ll i = 0; i * i < b; i++) is_prime[i] = true;</pre>
12
      for(ll i = 2; i * i < b; i++){}
13
14
         if(is_prime[i]){
15
           for(ll j = 2 * i; j * j < b; j += i)
             is_prime_small[j] = false;
16
17
           for(ll j = max(2LL, (a + i - 1) / i) * i; j < b; j += i)
             is_prime[j - a] = false;
18
        }
19
20
21
      for(ll i = 0; i < b - a; i++){
22
         if(is_prime[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)

```
LL exgcd(LL a,LL b,LL &x,LL &y)//扩展欧几里得算法
1
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;
      return ret;
10
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 {
     LL x,y;
14
     LL d=exgcd(a,mod,x,y);
15
     return d==1?(x%mod+mod)%mod:-1;
16
17 }
```

1.2 费马小定理求逆元 (模数需为质数)

```
/*
 1
    * 费马小定理求逆元
 2
    * 仅当 mod 为素数时可用
    * 调用 inv 函数获取逆元
    */
 5
 6
    const int mod = 1000000009;
 7
 8
    long long quickpow(long long a, long long b)
 9
10
11
      if (b < 0)
12
        return 0;
13
      long long ret = 1;
      a %= mod;
14
      while (b)
15
16
17
        if (b & 1)
          ret = (ret * a) % mod;
18
19
        b >>= 1;
20
        a = (a * a) \% mod;
      }
21
22
      return ret;
23
24
    long long inv(long long a)
25
      return quickpow(a, mod - 2);
26
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) {
       return b == 0 ? a : gcd(b, a % b);
 5
 6
 7
     void inv(ll a, ll b) {
 8
       if (a % b == 0) {
 9
10
         z = 0;
         y = 1;
11
12
         return;
13
14
       inv(b, a % b);
       ll r = z;
15
16
       z = y;
       y = r - a / b * y;
17
18
19
20
    void solve() {
21
     x = 0;
22
       m = 1;
       for (int i = 0; i < n; i++) {
23
         cin >> M >> r;
24
         ll b = r - x, d = gcd(m, M);
25
26
         if (b % d) {
           cout << "-1" << endl;//不存在
27
28
           return;
29
         inv(m/d, M/d);
30
         ll t = b / d * z % (M / d);
31
32
         x += m * t;
33
         m *= M/d;
34
35
       x = x > 0 ? x : x + m;
       cout << x << endl;
36
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
    ll w;
 6
 7
 8
    ll quickMod(ll a, ll b, ll m){
 9
      llans = 1;
      a %= m;
10
      while(b){
11
12
        if(b & 1){
           ans = ans * a % m;
13
14
        }
```

```
b >>= 1;
15
16
         a = a * a % m;
17
18
      return ans;
19
20
    //二次域乘法
21
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
    //二次域上快速幂
29
    node power(node a, ll b, ll m){
30
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
    //求勒让德符号
44
    ll legendre(ll a, ll p){
45
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){
      if(p == 2){
55
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
      node tmp;
71
72
      tmp.p = a;
73
      tmp.d = 1;
```

```
74
       node ans = power(tmp, (p + 1) >> 1, p);
 75
       return ans.p;
    }
 76
 77
     int main(void) {
 78
       int T;
 79
       cin >> T;
80
       while(T--){
81
82
         int n, p;
 83
         cin >> n >> p;
 84
         n \% = p;
 85
         int a = solve(n, p);
         if(a == -1){
 86
            cout << "No root" << endl;
87
            continue;
 88
 89
         int b = p - a;
 90
 91
         if(a > n){
92
           swap(a, b);
93
         if(a == b){
 94
            cout << a << endl;
 95
         }else{
 96
 97
            cout << a << ' ' << b << endl;
 98
99
       }
    }
100
           EXCRT
     2.3
    //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
  3
     bool excrt(ll r[], ll m[], ll n, ll &re, ll &mo){
  4
  5
       ll x, y;
       mo = m[0], re = r[0];
  6
       for(int i = 1; i < n; i++){}
  7
         ll d = exgcd(mo, m[i], x, y);
  8
         if((r[i] - re) % d!= 0) return 0;
  9
         x = (r[i] - re) / d * x % (m[i] / d);
 10
         re += x * mo;
 11
         mo = mo / d * m[i];
 12
         re %= mo;
 13
 14
       }
 15
       re = (re + mo) %mo;
       return 1;
 16
 17 }
     2.4 CRT
  1 //x mod m[i] = r[i];要求m[i]两两互质
```

//引用返回通解x = re + k * mo;

for(int i = 0; i < n; i++){

ll x, y, tm = mo / m[i];

mo = 1, re = 0;

3

4

5

6

void crt(ll r[], ll m[], ll n, ll &re, ll &mo){

for(int i = 0; i < n; i++) mo *= m[i];

3 BSGS

3.1 bsgs

```
void bsgs(ll y,ll z,ll p)//y^x=z \pmod{p} gcd(y, p) = 1
 1
 2
 3
      if(y==0 && z==0){puts("1");return;}//几句特判
      if(y==0 && z!=0){return;}//不存在
 4
 5
      mp.clear();
 6
      ll m=ceil(sqrt(p));
 7
 8
      ll tmp=z%p;mp[tmp]=0;//右边z*A^j, 当j=0时为z
 9
      for(ll i=1;i<=m;i++)
10
11
        tmp=tmp*y%p;
        mp[tmp]=i;
12
      }
13
14
15
      ll t=power(y,m,p);
16
      tmp=1;//左边y^(i*m), 当i=0时为1
      for(ll i=1;i<=m;i++)
17
18
        tmp=tmp*t%p;//i每加1,多乘y^(i*m)
19
        if(mp[tmp])
20
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
    //解决x ^ y mod z == k的最小y
 1
 2
 3
    struct Hash {
 4
      int i;
 5
      ll xi;
 6
      Hash(int a, ll b) : i(a), xi(b) {}
    };
 7
 8
    vector<Hash> has[maxn];
 9
10
    ll powMod(ll a, ll b, ll mod) {
11
      ll res = 1;
12
      while (b) {
13
        if (b & 1) {
14
           res = res * a % mod;
15
        }
16
17
        b >>= 1;
18
        a = a * a % mod;
      }
19
20
      return res;
21
22
    ll gcd(ll a, ll b) {
```

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

83 }

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){
 3
       ll rt = n;
       for(int i = 2; i * i <= n; i++){
 4
         if(n \% i == 0){
 5
 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
22
    void ZyfPhi() {
23
       phi[1] = 0;
24
       for(int i = 2; i < maxn; i++){</pre>
25
         phi[i] = i;
26
27
       for(int i = 2; i < maxn; i++){
         if(phi[i] == i){
28
29
           for(int j = i; j < maxn; j += i){
30
             phi[j] = phi[j] / i * (i - 1);
31
           }
         }
32
33
       for (int i = 2; i < maxn; i++) {
34
35
         phi[i] = phi[i] + phi[i - 1];
36
37
    }
38
    //O(n)得到欧拉函数phi[], 素数表prime[], 素数个数tot
39
    bool vis[maxn];
    int tot, phi[maxn], prime[maxn];
41
    void Phi(){
42
43
       phi[1] = 0;
       for(int i = 2; i < maxn; i++){</pre>
44
45
         if(!vis[i]){
           prime[tot++] = i, phi[i] = i - 1;
46
           for(int j = 0; j < tot; j++){
47
              if(i * prime[j] > maxn){
48
49
                break;
50
             }
             vis[i * prime[j]] = 1;
51
             if(i % prime[j] == 0){
52
                phi[i * prime[j]] = phi[i] * prime[j];
53
                break;
54
55
             }else{
```

5 组合数学

5.1 lucas

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

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

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

5.3 Big Combination

```
1 // 0 \le n \le 1e9, 0 \le m \le 1e4, 1 \le k \le 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;
         for(int j = 0; j < v.size(); j++){</pre>
 9
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
     ll Comb(int n, int m, int k){
18
19
       memset(dp, 0, sizeof(dp));
20
       v.clear();
       int tmp = k;
21
       for(int i = 2; i * i <= tmp; i++){</pre>
22
23
         if(tmp \% i == 0){
24
            int num = 0;
            while(tmp % i == 0) tmp /= i, num++;
25
26
            v.push_back(i);
27
         }
28
29
       if(tmp != 1) v.push_back(tmp);
       ll \, ans = Cal(n - m + 1, n, k, 1);
30
       for(int j = 0; j < v.size(); j++)ans = ans * powMod(v[j], dp[j], k) % k;
31
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)

```
// 0 <= m <= n <= 1000
2
    const int maxn = 1010;
3
    ll C[maxn][maxn];
 4
    void CalComb(){
5
      C[0][0] = 1;
6
      for(int i = 1; i < maxn; i++){</pre>
7
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
           组合数学初始化 (阶乘的逆元)
   // 0 <= m <= n <= 1e5, 模p为质数
3
    const int maxn = 100010;
    ll f[maxn];
4
    ll inv[maxn]; //阶乘的逆元
5
    void CalFact(){
6
      f[0] = 1;
7
      for(int i = 1; i < maxn; i++){</pre>
8
        f[i] = (f[i-1] * i) % p;
9
10
      inv[maxn - 1] = powMod(f[maxn - 1], p - 2, p);
11
      for(int i = maxn - 2; \sim i; i--){
12
        inv[i] = inv[i+1] * (i+1) % p;
13
14
      }
15
    }
16
    ll C(int n, int m){
17
      return f[n] * inv[m] % p * inv[n - m] % p;
18
19
    }
    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;
3
    const int N = 101;
 4
    ll s[N][N];
5
6
 7
    void init(){
      memset(s, 0, sizeof(s));
8
9
      s[1][1] = 1;
10
      for(int i = 2; i < N; i++){
        for(int j = 1; j <= i; j ++){
11
12
           s[i][j] = (s[i-1][j-1] + (i-1) * s[i-1][j]);
13
14
      }
    }
15
16
    int main(void){
17
      init();
18
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;
         for(int i = 1; i \le k; i++){
29
30
            fz += (s[n][i] - s[n-1][i-1]);
31
32
         double ans = fz * 1.0 / sum;
         printf("%.4lf\n", ans);
33
34
    }
35
             卡特兰数
     5.7
     int a[105][100];
 1
 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;
         for(int j = 1; j \le len; j++){
11
12
            int t = (a[i-1][j]) * (4 * i - 2) + yu;
13
           yu = t/10;
14
           a[i][j] = t \% 10;
15
         while(yu){
16
17
            a[i][++len] = yu % 10;
           yu /= 10;
18
19
20
         for(int j = len; j >= 1; j--){
            int t = a[i][j] + yu * 10;
21
22
            a[i][j] = t/(i+1);
23
           yu = t \% (i + 1);
24
         while(!a[i][len]){
25
           len--;
26
27
28
         a[i][0] = len;
29
       }
30
    }
31
     int main(void){
32
33
       catalan();
34
       int n;
       while(cin >> n){
35
36
         if(n == -1){
37
            break;
38
         for(int i = a[n][0]; i > 0; i-){
39
```

- 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*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。
- 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a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

L //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
 2
     const int maxn = (int) 5e5 + 100;
 3
    int p[maxn][31], pos[maxn][31];
 4
 5
    int n, q;
     int lastans = 0;
 6
 7
 8
     typedef long long ll;
 9
     void push_back(int x, int i) {
10
       for (int j = 0; j \le 30; j++) {
11
         p[i][j] = p[i - 1][j];
12
         pos[i][j] = pos[i - 1][j];
13
14
15
       int ti = i;
       for (int j = 30; j >= 0; j--) {
16
         if (x & (1 << j)) {
17
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);
              swap(pos[i][j], ti);
25
26
27
             ^= 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;
       cin >> T;
37
       while (T--) {
38
39
         lastans = 0;
           memset(p, 0, sizeof(p));
40
    //
     //
           memset(pos, 0, sizeof(pos));
41
42
         cin >> n >> q;
43
         int x;
         for (int i = 1; i <= n; i++) {
44
           cin >> x;
45
           push_back(x, i);
46
         }
47
         while (q--) {
48
49
           int e, l, r;
           cin >> e;
50
           if (e == 1) {
51
52
              cin >> l;
53
              l ^= lastans;
```

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

7 矩阵

7.1 斐波那切数列十进制

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

```
56
       cin >> x1 >> x2 >> a >> b;
57
       cin >> str >> mod;;
58
       int len = strlen(str);
59
       for (int i = 0; i < len; i++) {
60
61
         num[i] = str[i] - '0';
62
63
64
       num[len - 1]--;
       for (int i = len - 1; i >= 0 && num[i] < 0; i--) {
65
66
         num[i] += 10;
67
         num[i - 1]--;
       }
68
       mat base, A, ans;
69
       ans.m[1][1] = ans.m[2][2] = 1;
70
       base.m[1][1] = a % mod;
71
       base.m[1][2] = b \% \mod;
72
       base.m[2][1] = 1LL;
73
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

```
//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
 3
    #define Mod(a, b) (((a) % (b) + (b)) % (b))
 4
 5
    ll exgcd(ll a, ll b, ll &x, ll &y){
 6
 7
      lld = a;
      if(b){
 8
         d = exgcd(b, a \% b, y, x);
 9
10
        y = x * (a / b);
11
      }else{
12
        x = 1;
13
        y = 0;
14
15
      return d;
16 }
17
    bool solve(ll a, ll b, ll c, ll &x, ll &y, ll &dx, ll &dy){
18
      if(a == 0 \& b == 0){
19
         return 0;
20
21
22
      ll x0, y0;
23
      ll d = exgcd(a, b, x0, y0);
      if(c % d != 0) return 0;
24
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

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

```
19
          places++;
20
          a[places] = carry;
21
22
23
     }
24
      *输出
25
      * 最高位原样输出
26
27
      * 其他位小于1000的, 高位补0
      *需要头文件<iomanip>
28
29
30
      cout << a[places];
      for (i = places - 1; i >= 0; i--)
31
32
        cout << setw(4) << setfill('0') << a[i];
33
34
35
      cout << endl;
36
    8.4 定积分
    double F(double x)
2
    {
     //Simpson公式用到的函数
 3
4
    double simpson(double a, double b)//三点Simpson法,这里要求F是一个全局函数
5
6
    {
7
      double c = a + (b - a) / 2;
      return (F(a) + 4 * F(c) + F(b))*(b - a) / 6;
8
9
    }
    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
10
        点Simpson值A
11
    {
      double c = a + (b - a) / 2;
12
      double L = simpson(a, c), R = simpson(c, b);
13
     if (fabs(L + R - A) \le 15 * eps) return L + R + (L + R - A) / 15.0;
14
15
      return asr(a, c, eps / 2, L) + asr(c, b, eps / 2, R);
16 }
   double asr(double a, double b, double eps)//自适应Simpson公式(主过程)
17
18
19
      return asr(a, b, eps, simpson(a, b));
20
   }
          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;
    int n;
 4
    int lowbit(int x){
5
     return x & (-x);
6
7
8
    void add(ll a[], ll pos, ll val){
9
10
      while(pos <= maxn){</pre>
        a[pos] += val;
11
12
        pos += lowbit(pos);
```

```
13
   }
14
15
    ll sum(ll a[], ll pos){//求逆序对的个数
16
17
      ll tmp = 0;
18
      while(pos > 0){
19
        tmp += a[pos];
20
        pos -= lowbit(pos);
21
      }
22
      return tmp;
23
    }
24
    int main(void) {
25
      while(cin >> n){
26
27
        memset(c, 0, sizeof(ll) * (n +5));
        memset(b, 0, sizeof(ll) * (n + 5));
28
29
        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
30
        for(int i = 1; i <= n; i++){
31
          cin >> num[i];
32
33
          add(c, num[i], 1);
          //逆序对的个数
34
35
          //ans += sum(c, num[i]);
36
          ans += num[i] * (i - sum(c, num[i]));
37
        for(int i = n; i >= 1; i--){
38
          ans += num[i] * (sum(b, num[i]));
39
          add(b, num[i], 1);
40
        }
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
    int d[MAXN + 5];
 2
 3
    void init()
 4
    {
 5
      memset(d, 0, sizeof(d));
    }
 6
    void add(int x)
 7
 8
    {
 9
      for (int i = MAXN; i \ge 0; i--)
10
        if (x & (1LL << i))
11
12
        {
          if (d[i])
13
            x ^= d[i];
14
          else
15
16
17
            d[i] = x;
18
            break;
19
20
21
      }
22
    // 如何求异或后第k小的值
```

```
void work() //处理线性基
24
25
    {
26
      for (int i = 1; i <= 60; i++)
27
        for (int j = 1; j \le i; j++)
28
          if (d[i] & (1 << (j - 1)))
29
            d[i] ^= d[j - 1];
30
    ll k_th(ll k)
31
    {
32
      if (k == 1 \&\& tot < n)
33
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)
        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
36
37
      work();
      ll ans = 0;
38
      for (int i = 0; i \le 60; i++)
39
40
        if(d[i]!=0)
41
        {
          if (k % 2 == 1)
42
43
            ans = d[i];
44
          k /= 2;
        }
45
46 }
          大数板子 (易)
 1 int flag = 1;
    //初始化
 3
    void initial(string &a, string &b){
      while (a.size() < b.size())a = '0' + a;
 5
      while (b.size() < a.size())b = '0' + b;
 6 }
 7
    //打印
    void print(string &a, string &b){
 8
 9
      cout << a << endl;
10
      cout << b << endl;
11 }
12 //找出最大的字符串
    void findMax(string &a, string &b){
13
14
      string tmp;
15
      if (a<b){
16
        tmp = b;
17
        b = a;
18
        a = tmp;
19
      }
20 }
    //删除第一个字符'0'
21
    bool del(string &a){
22
23
      if (a[0] == '0'){
24
        a.erase(0, 1);
25
        return true;
26
      }
27
      else
28
        return false;
29
    //删除前面所有的 0
    void delAllZroe(string &a){
```

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

```
91
        for (int i = b.size() - 1; i >= 0; i--){
          int num1 = b[i] - '0';
92
          if (i != b.size() - 1) a = a + '0';
93
          for (int j = 1; j \le num1; j++){
94
 95
            res = bigItergeAdd(res, a);
96
          }
97
        }
98
        delAllZroe(res);
        cout << res << endl;
99
100
    }
101
     //大数除法
102
     void bigItergeDiv(string a, string b){
        initial(a, b);
103
        if (a<b){ cout << "0" << endl; return; }</pre>
104
        delAllZroe(b);
105
        string res = "0";
106
        string restmp = "1";
107
108
        string tmp = b;
109
        for (int i = 1; i<(a.size() - b.size()); i++){</pre>
          tmp += '0';
110
          restmp += '0';
111
112
113
        initial(a, b);
114
        while (a >= b){
115
          initial(a, tmp);
          if (a >= tmp){
116
            a = bigItergeSub(a, tmp);
117
            res = bigItergeAdd(res, restmp);
118
          }
119
          else{
120
            tmp.erase(tmp.size() - 1);
121
122
            restmp.erase(restmp.size() - 1);
            initial(a, tmp);
123
124
            if (a >= tmp){
125
               a = bigItergeSub(a, tmp);
126
               res=bigItergeAdd(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;
          //ans = bigItergeAdd(a,b);
138
139
          ans = bigItergeSub(a,b);
140 //
          bigItergeMul(a,b);
141
     //
           bigItergeDiv(a,b);
142
          if(flag == -1){}
            cout << '-';
143
          }
144
145
          cout << ans << endl;
146
147
```

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 {
5
      //io*2 bign*5*2 bool*6
6
 7
      friend istream & operator >> (istream &, bign &);
 8
9
      friend ostream & operator << (ostream &, const bign &);
10
      friend bign operator+(const bign &, const bign &);
11
12
13
      friend bign operator+(const bign &, int &);
14
      friend bign operator*(const bign &, const bign &);
15
16
      friend bign operator*(const bign &, int &);
17
18
19
      friend bign operator-(const bign &, const bign &);
20
21
      friend bign operator-(const bign &, int &);
22
      friend bign operator/(const bign &, const bign &);
23
24
25
      friend bign operator/(const bign &, int &);
26
27
      friend bign operator%(const bign &, const bign &);
28
      friend bign operator%(const bign &, int &);
29
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
      friend bool operator!=(const bign &, const bign &);
41
42
    private://如果想访问len,改成public
43
44
      int len, s[maxn];
    public:
45
      bign() {
46
        memset(s, 0, sizeof(s));
47
48
        len = 1;
      }
49
50
      bign operator=(const char *num) {
51
52
        int i = 0, ol;
53
        ol = len = strlen(num);
        while (num[i++] == '0' && len > 1)
54
55
          len--;
        memset(s, 0, sizeof(s));
56
        for (i = 0; i < len; i++)
57
```

```
s[i] = num[ol - i - 1] - '0';
58
          return *this;
59
        }
60
 61
        bign operator=(int num) {
62
          char s[maxn];
63
          sprintf(s, "%d", num);
64
          *this = s;
65
          return *this;
66
        }
 67
68
69
        bign(int num) {
 70
          *this = num;
 71
 72
        bign(const char *num) {
 73
          *this = num;
 74
 75
        }
 76
        string str() const {
 77
          string res = "";
 78
 79
          for (int i = 0; i < len; i++)
 80
            res = char(s[i] + '0') + res;
          if (res == "")
 81
            res = "0";
 82
 83
          return res;
        }
 84
     };
 85
 86
     bool operator<(const bign &a, const bign &b) {</pre>
 87
        int i;
 88
 89
        if (a.len != b.len)
          return a.len < b.len;
90
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) {
        return a < b || a > b;
110
111 }
112
     bool operator==(const bign &a, const bign &b) {
113
114
        return !(a < b || a > b);
115
116
```

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

```
bign operator-(const bign &a, int &b) {
176
        bign c = b;
177
178
        return a - c;
     }
179
180
     bign operator/(const bign &a, const bign &b) {
181
        int i, len = a.len;
182
183
        bign res, f;
        for (i = len - 1; i >= 0; i--) {
184
          f = f * 10;
185
186
          f.s[0] = a.s[i];
187
          while (f >= b) {
188
            f = f - b;
189
            res.s[i]++;
          }
190
        }
191
        while (res.s[len - 1] == 0 && len > 1)
192
193
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
     bign operator%(const bign &a, const bign &b) {
203
        int len = a.len;
204
205
        bign f;
        for (int i = len - 1; i >= 0; i--) {
206
          f = f * 10;
207
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
```

```
bign &operator/=(bign &a, const bign &b) {
235
236
       a = a/b;
237
       return a;
238 }
239
     bign & operator++(bign & a) {
240
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
     bign & operator -- (bign & a) {
251
252
       a = a - 1;
253
       return a;
254 }
255
     bign & operator -- (bign & a, int) {
256
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;
        cin >> a >> b;
277
278
        cout << a / b << endl;
279 }
     8.9
             快速乘
     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;a<=b?i<=b:i>=b;a<=b?++i:-i)</pre>
```

```
const int number=3;
3
   int main()
   {
4
     int sum;//sum是指数
5
     int c1[33005], c2[33005];
6
     while(scanf("%d", &sum), sum)
7
8
    {
9
      for(int i = 0; i <= sum; ++i)</pre>
10
        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
11
12
        c2[i] = 0;
13
      }
      //复杂度O(number*sum*sum),也可通过打表后O(1)访问
14
      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
15
       为公比的等比数列,若无此数列则删除此i即可
16
        for(int j = 0; j \le 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
17
18
         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
19
20
           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
21
       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
22
         }
23
        }
24
        for(int j = 0; j \le sum; ++j)
25
         c1[j] = c2[j];
26
27
          c2[j] = 0; // 记得每次合并一个括号后要把 c2 清零
28
29
      printf("%d\n", c1[sum]);
30
31
32
    return 0;
33
```

8.11 合数分解

```
//素数筛选
   const int MAXN=100000:
3
   int prime[MAXN+1];//得到小于等于MAXN的所有素数
   void getPrime()
4
5
   {
     memset(prime,0,sizeof(prime));
6
     for(int i=2;i<=MAXN;i++)</pre>
7
8
     {
9
       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
10
       {
11
         prime[prime[j]*i]=1;
12
         if(i%prime[j]==0)break;
13
14
     }
15
16
17
    //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)
20
21
   {
```

```
fatCnt=0;
22
23
     long long tmp=x;
     for(int i=1;prime[i]*prime[i]<=tmp;i++)</pre>
24
                                                //除法改为乘法提速,改为除法防止爆范围
25
       factor[fatCnt][1]=0;
26
27
       if(tmp%prime[i]==0)
28
29
         factor[fatCnt][0]=prime[i];
         while(tmp%prime[i]==0)
30
31
           factor[fatCnt][1]++;
32
           tmp/=prime[i];
33
34
35
         fatCnt++;
       }
36
37
     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。
38
39
       factor[fatCnt][0]=tmp;
40
41
       factor[fatCnt++][1]=1;
42
     }
43
      return fatCnt;
44 }
```

9 未学会

9.1 莫比乌斯反演

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

10 几何

10.1 fcy

```
#include<bits/stdc++.h>
 1
    #define mp make pair
 3
    #define rep(i,a,b) for(int i=a;i<=b;i++)</pre>
   using namespace std;
 4
 5 typedef long long ll;
    const double inf=1e200;
 6
 7
    const double eps=1e-12;
    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
      double angle;
18
    };
19
20
    struct Circle{
      point c; double r;
21
22
    }:
23
    int dcmp(double x){ return fabs(x)<eps?0:(x<0?-1:1);}</pre>
24
    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);}
27
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) {
       return dcmp(a.x-b.x)==0\&dcmp(a.y-b.y)==0;
32
33 }
    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);}
39
    double angle(point A,point B){ return acos(dot(A,B)/length(A)/length(B));} //夹角
40
    point jiaopoint(point p,point v,point q,point w)
41
    { //p+tv q+tw,点加向量表示直线,求直线交点
42
      //如果是线段,还应该实现判定是否相离;必要时判是否平行
43
      point u=p-q;
44
      double t=det(w,u)/det(v,w);//如果平行,除0会有问题
45
      return p+v*t;
46
47
    point llintersect(line A,line B) //直线交点,同上
48
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
    point GetCirPoint(point a,point b,point c)
54
55
```

```
point p=(a+b)/2; //ab中点
 56
 57
        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
 58
       if (dcmp(det(v,w))==0) //平行
 59
 60
         if(dcmp(length(a-b)+length(b-c)-length(a-c))==0) return (a+c)/2;
 61
         if(dcmp(length(b-a)+length(a-c)-length(b-c))==0) return (b+c)/2;
 62
         if(dcmp(length(a-c)+length(c-b)-length(a-b))==0) return (a+b)/2;
 63
 64
       }
 65
       return jiaopoint(p,v,q,w);
 66
     }
 67
     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。
 68
 69
       point C=w[1]; double r=0;
 70
 71
       rep(i,1,N){
 72
         if(dist(C,w[i])>r) {
 73
            C=w[i]; r=0;
            rep(j,1,i-1) {
 74
 75
              if(dist(C,w[j])>r) {
                C=(w[i]+w[j])/2; r=dist(w[i],w[j])/2;
 76
                rep(k,1,j-1) {
 77
                  if(dist(C,w[k])>r){
 78
 79
                    C=GetCirPoint(w[i],w[j],w[k]);
 80
                    r=dist(C,w[i]);
 81
                  }
               }
 82
 83
 84
           }
 85
 86
       printf("%.2lf %.2lf %.2lf\n",C.x,C.y,r);
 87
 88
     double area(vector<point>p){//求面积
 89
        double ans=0; int sz=p.size();
 90
       for(int i=1;i < sz-1;i++) ans+=det(p[i]-p[0],p[i+1]-p[0]);
 91
 92
       return ans/2.0;
 93
     }
     double seg(point O,point A,point B){//返回比例
 94
       if(dcmp(B.x-A.x)==0) return (O.y-A.y)/(B.y-A.y);
 95
 96
       return (O.x-A.x)/(B.x-A.x);
 97
     vector<point>pp[110];
 98
     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。
100
        double res=0;
101
       for(int i=0;i<N;i++){</pre>
102
         int sz=p[i].size();
103
         for(int j=0;j<sz;j++){</pre>
104
105
            int m=0;
106
            s[m++]=mp(0,0);
107
            s[m++]=mp(1,0);
            point a=p[i][j], b=p[i][(j+1)%sz];
108
            for(int k=0;k<N;k++){
109
              if(i!=k){
110
                int sz2=p[k].size();
111
112
                for(int ii=0;ii<sz2;ii++){</pre>
113
                  point c=p[k][ii],d=p[k][(ii+1)%sz2];
                  int c1=dcmp(det(b-a,c-a));
114
```

```
int c2=dcmp(det(b-a,d-a));
115
                 if(c1==0&&c2==0){
116
                   if(dcmp(dot(b-a,d-c))){
117
118
                     s[m++]=mp(seg(c,a,b),1);
119
                     s[m++]=mp(seg(c,a,b),-1);
120
                 }
121
122
                 else{
                   double s1=det(d-c,a-c);
123
                   double s2=det(d-c,b-c);
124
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;
           int cov=s[0].second;
133
           for(int j=1;j<m;j++){</pre>
134
             now=min(max(s[j].first,0.0),1.0);
135
             if(!cov) sum+=now-pre;
136
             cov+=s[j].second;
137
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
       //记得提前判断是否相交,如果相交就利用角度就可以求了
147
       double L=dist(a,b);
148
       if(L>r1+r2) return;//相离
149
       if(L+r1<r2||L+r2<r1) return;//包含
150
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
       double ang1=base+t,ang2=base-t;
153
154
       point A=a+point{r1*cos(ang1),r1*sin(ang1)};
155
       point B=a+point{r1*cos(ang2),r1*sin(ang2)};
156
     bool cmp1(point a,point b){ return a.x==b.x?a.y<b.y:a.x<b.x; } //x排序
157
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
       top=0;
161
       for(int i=1;i<=n;i++){ //下凸包
162
         while(top>=2&&det(ch[top-1],ch[top],a[i])<=0) top--;
163
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;
       rep(i,1,top-1){//先求出凸包,然后凸包上旋转卡壳
175
176
         while(det(p[i],p[i+1],p[now])<det(p[i],p[i+1],p[now+1])){</pre>
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
       }
       return ans;
182
183
     }
     point ch[2000000],p[2000000];
184
185
     double getangle(point a){ return atan2(a.v,a.x);}
     double getangle(line a){ return getangle(a.p);}
186
     point ss[maxn]; line t[maxn],q[maxn]; int head,tail;
187
     bool cmp2(line a, line b){//方向的极角排序
188
       double A=getangle(a),B=getangle(b);
189
       point t=(b.a+b.p)-a.a;
190
191
       if(fabs(A-B)<eps) return det(a.p,t)>=0.0;
192
       return A<B;
193 }
    bool onright(line P,line a,line b)
194
195
       point o=llintersect(a,b);
196
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];
       rep(i,1,N) t[i].a=ss[i], t[i].p=ss[i+1]-ss[i];
203
204
       sort(t+1,t+N+1,cmp2);
205
       int tot=0;
206
       rep(i,1,N-1) {
         if(fabs(getangle(t[i])-getangle(t[i+1]))>eps)
207
208
          t[++tot]=t[i];
209
210
       t[++tot]=t[N]; head=tail=0;
211
       rep(i,1,tot){
         while(tail>head+1&&onright(t[i],q[tail],q[tail-1])) tail--;
212
213
         while(tail>head+1&&onright(t[i],q[head+1],q[head+2])) head++;
214
         q[++tail]=t[i];
215
       while(tail>head+1&&onright(t[head+1],q[tail],q[tail-1])) tail--;return tail-head>2;
216
217
     double TriAngleCircleInsection(Circle C, point A, point B) //圆与多边形面积交
218
219
     {
220
       //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]);
221
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;
       if(dcmp(det(OA,OB))==0) return 0; //, 三点一线,不构成三角形
226
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
         double y=(dot(AB,AC)+sqrt(r*r*DAB*DAB-det(AB,AC)*det(AB,AC)))/DAB;
236
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
       else if(fabs(det(OA,OB))>=r*DAB||dot(AB,AC)<=0||dot(BA,BC)<=0)//
240
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
        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]
246
247
      }
       else //弧+三角形
248
249
250
        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;
251
        double TS=det(OA,OB)*0.5;
252
        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);
253
254
      }
   }
255
256
    double ltoseg(point p,point a,point b){
257
       point t=p-a;
      if(dot(t,b-a)<=0) return dist(p,a);</pre>
258
259
       else if(dot(p-b,a-b)<=0) return dist(p,b);
       return fabs(det(t,b-a))/dist(a,b);
260
261
    bool isinside(point a) //O(N)判定是否在(任意多边形)内
262
263
    //算法描述:首先,对于多边形的水平边不做考虑,其次,
264
    //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,
265
    //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。
266
267
      int ncross=0; int N;
      rep(i,0,N-1) {
268
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
        if(p1.y==p2.y) continue; //默认做水平x轴的线,所以水平线不考虑
271
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))((t<=0\&\&p[i+1].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
   {
      int L=2,R=top-2,Mid;
281
282
      while(L<=R){
283
        Mid=(L+R)>>1;
284
        if(det(ch[Mid]-ch[1],A-ch[1])<0) R=Mid-1;
285
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;
          L=Mid+1;
288
289
290
291
      return false;
```

```
292 }
    int main()
293
294
    {
295
    }
296
     10.2
             圆
    struct point
 1
 2
    {
 3
       double x, y;
       point();
 4
 5
      point(double _x, double _y);
       bool operator<(Point b) const;
 6
                                      // 点左右判断 - line 中使用
 7
      bool operator==(point b) const;
                                      // 点相等判断
      point operator+(const point &b) const; // 向量相加
 8
 9
      point operator-(const point &b) const; // 向量相减
       point operator*(const double &k) const; // 向量乘法
 10
       point operator/(const double &k) const; // 向量除法
 11
12
       point trunc(double r) const;
                                    // 向量模转换
13
       double operator*(const point &b) const; // 向量点积
       double operator^(const Point &b) const; // 向量叉积
14
       double len(bool isSqrt = true);
                                     // 向量模长度,参数为是否开平方
15
       double distance(const point &other); // 点距离
16
       point rotleft();
                              // 向量绕原点逆时针旋转 90 度
17
18
      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
19
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;
      line();
26
       line(point _s, point _e);
27
28
      bool operator==(line v);
                                 // 判断射线重合
                                 // 根据一个点和倾斜角 angle 确定直线,0 <= angle < pi
29
      line(point p, double angle);
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
                              // 返回线段的倾斜角
33
       double angle();
34
      int relation(point p);
                               // 点和直线关系
35
       bool pointonseg(point p);
                                  // 点在线段上的判断
                              // 两向量平行 (对应直线平行或重合)
      bool parallel(line v);
36
37
      int segcrossseg(line v);
                                // 两线段相交判断
      int linecrossseg(line v);
                                // 直线和线段相交判断
38
      int linecrossline(line v);
39
                                // 两直线关系
      point crosspoint(line v);
40
                                // 求两直线的交点
      double dispointtoline(point p); // 点到直线的距离
41
       double dispointtoseg(point p);
                                   // 点到线段的距离
42
       double dissegtoseg(line v);
43
                                  // 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
44
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
```

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

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

```
169
          int rel = relationcircle(v);
170
          if (rel == 1 || rel == 5)
171
            return 0;
172
173
          double d = p.distance(v.p);
          double l = (d * d + r * r - v.r * v.r) / (2 * d);
174
          double h = sqrt(r * r - l * l);
175
          point tmp = p + (v.p - p).trunc(l);
176
          p1 = tmp + ((v.p - p).rotleft().trunc(h));
177
178
          p2 = tmp + ((v.p - p).rotright().trunc(h));
179
          if (rel == 2 || rel == 4)
180
            return 1;
181
          return 2;
182
        }
        // 求直线和圆的交点,返回交点个数
183
        int pointcrossline(line v, point &p1, point &p2)
184
185
186
          if (!(*this).relationline(v))
187
            return 0;
          point a = v.lineprog(p);
188
          double d = v.dispointtoline(p);
189
          d = sqrt(r * r - d * d);
190
          if(sgn(d) == 0)
191
192
193
            p1 = a;
194
            p2 = a;
195
            return 1;
196
          p1 = a + (v.e - v.s).trunc(d);
197
          p2 = a - (v.e - v.s).trunc(d);
198
199
          return 2;
200
        // 得到过 a,b 两点,半径为 r1 的两个圆
201
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);
          if (!t)
207
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);
          if (sgn(dis - r1 * 2) > 0)
216
217
            return 0;
218
          if(sgn(dis) == 0)
219
220
            c1.p = q + ((u.e - u.s).rotleft().trunc(r1));
            c2.p = q + ((u.e - u.s).rotright().trunc(r1));
221
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
        }
        // 同时与直线 u,v 相切, 半径为 r1 的圆
240
241
        int getcircle(line u, line v, double r1, circle &c1, circle &c2, circle &c3, circle &c4)
242
          if (u.parallel(v))
243
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));
          line u2 = line(u.s + (u.e - u.s).rotright().trunc(r1), u.e + (u.e - u.s).rotright().trunc(r1));
246
          line u3 = line(v.s + (v.e - v.s).rotleft().trunc(r1), v.e + (v.e - v.s).rotleft().trunc(r1));
247
          line u4 = line(v.s + (v.e - v.s).rotright().trunc(r1), v.e + (v.e - v.s).rotright().trunc(r1));
248
          c1.r = c2.r = c3.r = c4.r = r1;
249
          c1.p = u1.crosspoint(v1);
250
251
          c2.p = u1.pointonseq(v2);
252
          c3.p = u2.pointonseq(v1);
          c4.p = u2.pointonseg(v2);
253
254
          return 4;
255
        // 同时与不相交圆 cx,cy 相切, 半径为 r1 的圆
256
        int getcircle(circle cx, circle cy, double r1, circle &c1, circle &c2)
257
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);
          if (!t)
261
262
             return 0;
263
          c1.r = c2.r = r1;
264
          return t;
265
        }
        int tangentline(point q, line &u, line &v)
266
267
          int x = relation(q);
268
269
          if(x == 2)
             return 0;
270
271
          if(x == 1)
272
273
            u = line(q, q + (q - p).rotleft());
274
            v = u;
            return 1;
275
          }
276
277
          double d = p.distance(q);
278
          double l = r * r / d;
279
          double h = \operatorname{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);
          if (rel >= 4)
288
             return 0.0;
289
          if (rel <= 2)
290
             return min(area(), v.area());
291
292
           double d = p.distance(v.p);
           double hf = (r + v.r + d) / 2.0;
293
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));
           a1 = a1 * r * r;
296
297
           double a2 = acos((v.r * v.r + d * d - r * r) / (2.0 * v.r * d));
298
          a2 = a2 * v.r * v.r;
          return a1 + a2 - ss;
299
300
        }
        // 求圆和三角形 pab 的相交面积
301
302
        double areatriangle(Point a, Point b)
303
304
          if (sgn((p - a) ^ (p - b)) == 0)
305
             return 0.0;
306
           point q[5];
          int len = 0;
307
           q[len++] = a;
308
          line l(a, b);
309
310
          point p1, p2;
311
          if (pointcrossline(l, q[1], q[2]) == 2)
312
313
             if (sgn((a - q[1]) * (b - q[i])) < 0)
               q[len++] = q[1];
314
315
             if (sgn((a - q[2]) * (b - q[2])) < 0)
               q[len++] = q[2];
316
317
318
           q[len++] = b;
          if (len == 4 \&\& sgn((q[0] - q[1]) * (q[2] - q[1])) > 0)
319
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
             else
329
330
               res += fabs((q[i] - p) ^ (q[i + 1] - p)) / 2.0;
331
332
          return res;
333
        }
334
     };
                体积
      10.3
      #include <iostream>
  1
  2
      #include <cmath>
  3
  4
      using namespace std;
  5
      const double pi = acos(-1.0);
  6
```

```
//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){
9
      return 1.0 / 3 * r * r * pi * h;
10
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 \text{ //V} = 1.0 \text{ / } 6 \text{ * abs(a1 * b2 * c3 + b1 * c2 * a3 + c1 * a2 * b3 - a1 * c2 * b3 - b1 * a2 * c3 - c1 * b2 * a3);
   struct Point{
      double x;
18
19
      double v:
      double z;
20
21 };
22
   double volumn_a(Point a, Point b, Point c){
23
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
28 //体积是V=4/3*pi*a*b*c;
29 double volumn_Ellipse(double a, double b, double c){
      return 4.0 / 3 * pi * a * b * c;
30
31 }
32
33 //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);
35
   }
36
37
38
   //球缺
    double volumn_MissingBall(double h, double r){
39
      return pi * h * h * (r - h / 3.0);
40
41 }
42
   //交叉圆柱体的体积
43
    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
    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);
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 <
   int ConvexHull(Point *p,int n ,Point *ch)
5
6
   {
7
      sort(p,p+n);
8
      int m = 0;
9
      for(int i = 0; i < n; ++i)
10
      {
```

```
while(m>1&& Cross(ch[m-1]-ch[m-2],p[i]-ch[m-2])<=0) m--;
11
12
        ch[m++] = p[i];
13
      }
14
      int k = m;
15
      for(int i = n-2; i >= 0; --i)
16
17
      {
        while(m > k&& Cross(ch[m-1]-ch[m-2],p[i]-ch[m-2]) <= 0) m--;
18
19
        ch[m++] = p[i];
20
21
      if(n > 1) m--;
22
      return m;
    }
23
             二维几何模板
    10.5
    #include <bits/stdc++.h>
    #define mem(ar,num) memset(ar,num,sizeof(ar))
3
    #define me(ar) memset(ar,0,sizeof(ar))
4
    #define lowbit(x) (x&(-x))
    #define forn(i,n) for(int i = 0;i < n; ++i)
5
    using namespace std;
6
7
    typedef long long LL;
    typedef unsigned long long ULL;
    const int prime = 999983;
10 const int INF = 0x7FFFFFFF;
11
   const LL INFF =0x7FFFFFFFFFFF;
12 const double pi = acos(-1.0);
   const double inf = 1e18;
13
    const double eps = 1e-10;
    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
    {
      return Vector(A.x/p,A.y/p);
34
35
    }
    Vector operator * (Vector A, double p)
36
37
38
      return Vector(A.x*p,A.y*p);
   }
39
    double angle(Vector v)//求向量的角度从0到2*pi
40
41
42
      return atan2(v.y,v.x);
```

```
43
     int dcmp(double x)
 44
 45
     {
       if(fabs(x)<eps)
 46
 47
         return 0;
 48
       else
         return x < 0?-1:1;
 49
 50
     bool operator < (const Point &a,const Point &b)
 51
 52
     {
 53
       if(dcmp(a.x-b.x)==0)
 54
         return a.y<b.y;
 55
       else
         return a.x<b.x;
 56
     }
 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 }
     double Dot(Vector A, Vector B)
 64
     {
 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
     double Angle(Vector A, Vector B)
 72
 73
     {
       return acos(Dot(A,B)/Length(A)/Length(B));
 74
 75
     }
 76
     double Cross(Vector A, Vector B)
 77
     {
       return A.x*B.y - A.y*B.x;
 78
 79
    }
 80
     double Area2(Point A, Point B, Point C)
 81
 82
       return Cross(B-A,C-A);
 83
     Vector Rotate(Vector A, double rad)
 84
 85
       return Vector (A.x*cos(rad)-A.y*sin(rad),A.x*sin(rad)+A.y*cos(rad));
 86
 87
     Vector Normal(Vector A)//单位法线
 88
 89
    {
       double L = Length(A);
 90
       return Vector(-A.y/L,A.x/L);
 91
 92 }
     //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 }
     double Distance_To_Segment(Point P,Point A,Point B)
105
106
     {
       if(A==B)
107
108
         return Length(P-A);
       Vector v1 = B-A, v2 = P-A, v3 = P-B;
109
       if(dcmp(Dot(v1,v2))<0)</pre>
110
         return Length(v1);
111
112
       else if(dcmp(Dot(v1,v3))>0)
113
         return Length(v3);
114
       else
         return fabs(Cross(v1,v2))/Length(v1);
115
116
     Point Get_Line_Projection(Point P,Point A,Point B)//求投影点
117
118
119
       Vector v = B- A;
       return A + v*(Dot(v,P-A)/Dot(v,v));
120
121 }
     //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)
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);
       return dcmp(c1)*dcmp(c2)<0&dcmp(c3)*dcmp(c4)<0;
127
128 }
     //判断点是否在线段上(不包括端点)
129
     bool Onsegment(Point p,Point a1,Point a2)
130
131
       return dcmp(Cross(a1-p,a2-p))==0&&dcmp(Dot(a1-p,a2-p))<0;
132
133
             最小三角形
     10.6
    // 最小三角形,注意longlong
     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){
       return Point(A.x-B.x,A.y-B.y);
 11
 12 }
 13
     long long Cross(Vector A, Vector B){
       return A.x*B.y-A.y*B.x;
 14
 15
     Point A[maxn],B[maxn];
     bool operator <(const Point &A,const Point &B){
 17
 18
 19
       return A.y*B.x < A.x*B.y;
    }
 20
 21
     int main(void)
 22
 23
       int N;cin>>N;
```

```
for(int i = 0; i < N; ++i){
24
25
         cin>>A[i].x>>A[i].y;
26
27
       double ans = inf;
      for(int i = 0; i < N; ++i){
28
29
         int t = 0;
         for(int j = 0; j < N; ++j)
30
31
           if(i != j)
32
             B[t++] = A[j]-A[i];
         sort(B,B+t);
33
34
         // assert(t == N-1);
35
         for(int j = 0; j < t-1; ++j){
           ans = min(ans,fabs(Cross(B[j],B[j+1]))/2.0);
36
        }
37
38
      printf("%.3f",ans);
39
40
41
      return 0;
42 }
              最大三角形
     10.7
    // 最大三角形
    const int N=50005;
 3
 4
    struct Point
 5
    {
 6
      double x,y;
 7
    };
 8
 9
    Point stack[N];
    Point p[N];
10
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
    double cross(Point A,Point B,Point C)
20
21
    {
      return (B.x-A.x)*(C.y-A.y)-(B.y-A.y)*(C.x-A.x);
22
23
24
    bool cmp(Point a,Point b)
25
26
    {
       double k=cross(MinA,a,b);
27
      if(k>0) return 1;
28
      if(k<0) return 0;
29
30
      return dist(MinA,a)<dist(MinA,b);</pre>
31
    }
32
33
    void Graham(int n)
34
    {
35
      for(i=1; i<n; i++)
36
```

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

```
96
         printf("%.2lf\n",rotating_calipers(top));
97
98
 99
       return 0;
100
      10.8 三维凸包
     struct Face{
  1
  2
       int v[3];
       Vector3 normal(Vector *P)
  3
  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
     {
        vector <Face> cur;
 14
        cur.push_back((Face){{0,1,2}});
 15
        cur.push_back((Face){{2,1,0}});
 16
 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
         for(int j= 0;j < cur.size(); ++j)</pre>
 21
 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)
              vis[f.v[k]][f.v[(k+1)%3]] = res;
 27
 28
 29
         for(int j = 0;j < cur.size(); ++j)</pre>
 30
 31
            for(int k = 0; k < 3; ++k)
 32
              int a = cur[j].v[k],b = cur[j].v[(k+1)%3];
 33
              if(vis[a][b] != vis[b][a]&&vis[a][b])//(a,b) 是分界线, 左边对P[i] 可见
 34
 35
              next.push_back((Face){{a,b,i}});
 36
           }
 37
 38
         cnr = next;
 39
 40
       return cur;
 41
     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)
 44
 45
       return Point3(p.x + randeps(),p.y+randeps(),p.z+randeps());
 46
 47
 48
 49
     //.....
     struct Face{
```

```
int v[3];
51
       Vector3 normal(Vector *P)
52
53
      {
         return Cross(P[v[1]]-P[v[0]],P[v[2]]-P[v[0]]);
54
55
      int cansee(Point *P,int i)const
56
57
      {
         return Dot(P[i]-P[v[0]],normal(P)) > 0?1:0;
58
59
      }
60
    };
    vector <Face> CH3D(Point3* P,int n)
61
62
       vector <Face> cur;
63
64
       cur.push_back((Face){{0,1,2}});
       cur.push_back((Face){{2,1,0}});
65
       for(int i = 3; i < n; ++i)
66
67
68
         vector<Face> next;
         //计算每条边 🛮 左面 🖟 的可见性
69
         for(int j= 0;j < cur.size(); ++j)</pre>
70
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
         for(int j = 0;j < cur.size(); ++j)</pre>
78
79
         {
           for(int k = 0; k < 3; ++k)
80
81
82
             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
83
              next.push_back((Face){{a,b,i}});
84
85
           }
        }
86
87
         cnr = next;
88
      }
89
      return cur;
90
    double rand01() {return rand() / (double) RAND_MAX;}//0-1 的随机数
91
    double randeps() {return (rand01()-0.5) * eps;}
    Point3 add_noise(Point3 p)
93
94
95
      return Point3(p.x + randeps(),p.y+randeps(),p.z+randeps());
96
    }
              三维几何模板
     10.9
    #include <bits/stdc++.h>
    const double eps = 1e-6;
 3
    using namespace std;
 4
    struct Point3
 5
    {
 6
       Point3(double x = 0,double y = 0,double z = 0):x(x),y(y),z(z) {}
 8
 9
    };
```

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

```
69 // */
       double t = Dot(n,p0-p1)/Dot(n,p2-p1);
 70
       return p1 + v*t;
 71
 72
 73
     Point3 LinePlaneIntetsection(Point3 p1,Point3 p2,double A,double B,double C,double D)
 74
    {
 75
       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));
 76
       return p1 + v*t;
 77
 78
    }
     Vector3 Cross(Vector3 A, Vector3 B)
 79
 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
     double Area2(Point3 A,Point3 B,Point3 C)
 83
 84
 85
       return Length(Cross(B-A,C-A));
 86
    }
     ////已知平面的三点,求出点法式
 87
    //Vector3 Solven(Point3 A,Point3 B,Point3 C)
    //{
 89
 90 // return Cross(B-A,C-A);
 91 //}
     //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)
 94
     {
 95
       double area1 = Area2(P,A,B);
       double area2 = Area2(P,A,C);
 96
       double area3 = Area2(P,B,C);
 97
       double area4 = Area2(A,B,C);
 98
       return dcmp(area1+area2+area3-area4)==0;
 99
100 }
101
     //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)
102
103
104
       Vector3 n = Cross(P1-P0,P2-P0);
105
106
       if(dcmp(Dot(n,B-A))==0)
         return false;
107
108
109
       double t = Dot(n,P0-A)/Dot(n,B-A);
       if(dcmp(t) < 0 \mid | dcmp(t-1) > 0)
110
         return false:
111
112
       P = A + (B-A) * t;
       return PointInTri(P,P0,P1,P2);
113
114 }
double DitantceToLine(Point3 P,Point3 A,Point3 B)
116 {
       return Length(Cross(A-P,B-P))/Length(A-B);
117
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;
       if(dcmp(Dot(v1,v2)) == 0) return Length(v2);
123
124
      if(dcmp(Dot(v1,v3)) > 0) return Length(v3);
125
       return Length(Cross(v1,v2))/Length(v1);
126
     double Volume6(Point3 A,Point3 B,Point3 C,Point3 D)
127
```