数学 & 计算几何

判断两条线段是否相交

```
//Attention: 线段和线段的交点在线段的端点上

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cmath>
#include <algorithm>
using namespace std;

struct NodeTp {
    int x, y;
    void Read() {
        scanf ("%d%d", &x, &y);
    }
    NodeTp operator- (NodeTp A) {
        return (NodeTp) {
            (x - A.x), (y - A.y)
        };
    }
};

inline int Cross (NodeTp A, NodeTp B) {
    return A.x * B.y - A.y * B.x;
}

bool CheckRec (NodeTp A1, NodeTp A2, NodeTp B1, NodeTp B2) { // 快速跨立实验
    int x1, x2, x3, x4;
    x1 = min (A1.x, A2.x);
    x2 = max (A1.x, A2.x);
    x3 = min (B1.x, B2.x);
    x4 = max (B1.x, B2.x);
    if (x2 < x3 || x4 < x1) return false;
    x1 = min (A1.y, A2.y);
    x2 = max (A1.y, A2.y);
    x3 = min (B1.y, B2.y);
    x4 = max (B1.y, B2.y);
    if (x2 < x3 || x4 < x1) return false;
    return true;
```

```
}

bool Check (NodeTp A, NodeTp B, NodeTp C) { // 向量A、B是否在C的两
侧
    if (A.x + B.x == 0 && A.y + B.y == 0) return false;
    int tmp1 = Cross (A, C);
    int tmp2 = Cross (B, C);
    if (tmp1 > 0 && tmp2 < 0) return true;
    else if (tmp1 < 0 && tmp2 > 0) return true;
    return false;
}

int main() {
    NodeTp A1, A2, B1, B2;
    A1.Read();
    A2.Read();
    B1.Read();
    B2.Read();
    printf ("Line1: (%d,%d) (%d,%d)\n", A1.x, A1.y, A2.x, A2.y);
    printf ("Line2: (%d,%d) (%d,%d)\n", B1.x, B1.y, B2.x, B2.y);
    if (!CheckRec (A1, A2, B1, B2) ) printf ("Not Cross!\n");
    else if (Check (B1 - A1, B2 - A2, A2 - A1) && Check (A1 - B1,
A2 - B1, B2 - B1) ) printf ("Cross!\n");
    else printf ("Not Cross!\n");
    return 0;
}
```

圆面积并

```
//work函数可计算出被覆盖i次的圆面积并，储存于ans[i]中
//复杂度O(N^2*log(N))
//实现方式为有向面积并
//弓形切割+内部有向多边形面积
//包含函数：
//弓形面积sector
//点积dot 叉积xet
//向量旋转Rotate
//两圆求交点getcross,返回值void，结果存于d1、d2中，flag表示各种情况
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <algorithm>
```

```
#define LL long long
using namespace std;
const double pi = 3.14159265358979323846264338327950;
int N, len;
double ans1, ans2, ans[100010];
class poi {
  public:
    double x, y;
} d1, d2;
class cir {
  public:
    poi p;
    double r;
} P[100010];
class rec {
  public:
    poi p;
    int t;
    double ta;
} q[100010];
bool cmp (const rec &a, const rec &b) {
    return a.ta < b.ta || (a.ta == b.t && a.t < b.t);
}
double sector (cir a, double alpha) {
    double res1 = a.r * a.r * alpha / 2;
    double res2 = a.r * a.r * sin (alpha) / 2;
    return res1 - res2;
}
double dot (poi a, poi b) {
    return a.x * b.x + a.y * b.y;
}
double xet (poi a, poi b) {
    return a.x * b.y - b.x * a.y;
}
void Rotate (poi a, poi &b, double alpha) {
    b.x = a.x * cos (alpha) - a.y * sin (alpha);
    b.y = a.x * sin (alpha) + a.y * cos (alpha);
}
void getcross (cir a, cir b, poi &d1, poi &d2, int &flag) {
    poi d;
    double dist, co, alpha, timi;
    d.x = b.p.x - a.p.x;
    d.y = b.p.y - a.p.y;
    dist = sqrt (dot (d, d) );
```

```
    if (dist >= a.r + b.r) {
        flag = 0;
        return;
    }
    if (dist + b.r <= a.r) {
        flag = 0;
        return;
    }
    if (dist + a.r <= b.r) {
        flag = 1;
        return;
    }
    co = (a.r * a.r + dist * dist - b.r * b.r) / a.r / dist / 2;
    alpha = acos (co);
    timi = a.r / dist;
    Rotate (d, d1, -alpha);
    d1.x *= timi;
    d1.y *= timi;
    d1.x += a.p.x;
    d1.y += a.p.y;
    Rotate (d, d2, alpha);
    d2.x *= timi;
    d2.y *= timi;
    d2.x += a.p.x;
    d2.y += a.p.y;
    flag = 2;
}
void work() {
    int r, delta, flag, now;
    for (int i = 1; i <= N; i++) {
        delta = now = r = 0;
        q[++r].p.x = P[i].p.x - P[i].r;
        q[r].p.y = P[i].p.y;
        q[r].t = 1;
        q[r].ta = -pi;
        q[++r].p.x = P[i].p.x - P[i].r;
        q[r].p.y = P[i].p.y;
        q[r].t = -1;
        q[r].ta = pi;
        for (int j = 1; j <= N; j++)
            if (i != j) {
                getcross (P[i], P[j], d1, d2, flag);
                if (flag == 0) continue; //相离或P[j]被P[i]包含
                if (flag == 1) {
```

```
                delta++;    //P[i]被P[j]包含
                continue;
            }
            q[++r].p = d1;
            q[r].t = 1;
            q[r].ta  =  atan2  (q[r].p.y  -  P[i].p.y,  q[r].p.x  -
P[i].p.x);
            q[++r].p = d2;
            q[r].t = -1;
            q[r].ta  =  atan2  (q[r].p.y  -  P[i].p.y,  q[r].p.x  -
P[i].p.x);
            if (q[r - 1].ta > q[r].ta) now++;
        }
        sort (q + 1, q + r + 1, cmp);
        for (int j = 1; j <= r; j++) {
            now += q[j].t;
            int step = now + delta;
            double alpha = q[j % r + 1].ta - q[j].ta;
            if (alpha < 0) alpha += 2 * pi;
            ans[step] += sector (P[i], alpha);
            ans[step] += xet (q[j].p, q[j % r + 1].p) / 2;
        }
    }
}
int main() {
    freopen ("B.in", "r", stdin);
    freopen ("B.out", "w", stdout);
    scanf ("%d", &N);
    for  (int  i  =  1;  i  <=  N;  i++)  scanf  ("%lf%lf%lf",  &P[i].p.x,
&P[i].p.y, &P[i].r);
    work();
    for (int i = 1; i <= N; i++)
        if (i & 1) ans1 += ans[i] - ans[i + 1];
        else ans2 += ans[i] - ans[i + 1];
//  printf("%.10lf %.10lf\n",ans1,ans2);
    printf ("%.5lf %.5lf\n", ans1, ans2);
}
```

圆与多边形面积交

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

```
using namespace std;

const int MaxN = 100;
const double eps = 1e-7;
const double PI = acos (-1.0);

struct NodeTp {
    double x, y;
    NodeTp() {};
    NodeTp (double x, double y) : x (x), y (y) {}
    void Print() {
        printf ("%.2lf %.2lf\n", x, y);
    }
};

int n, m, x, y, x1, y11, x2, y2;
double r, sumarea, dis[MaxN];
NodeTp O, nod[MaxN], area[MaxN];

inline double Distance (NodeTp A, NodeTp B) {
    return sqrt ( (A.x - B.x) * (A.x - B.x) + (A.y - B.y) * (A.y -
B.y) );
}

inline double Cross (NodeTp A, NodeTp B) {
    //printf("Triangle: \n"); A.Print(); B.Print(); printf("\n");
    return A.x * B.y - B.x * A.y;
}

inline double Sector (NodeTp A, NodeTp B) {
    //printf("Sector: \n"); A.Print(); B.Print(); printf("\n");
    double phiA = atan2 (A.y, A.x), phiB = atan2 (B.y, B.x);
    double phi = phiB - phiA;
    if (phi > PI) phi -= PI * 2;
    if (phi < -PI) phi += PI * 2;
    return phi * r * r * .5;
}

void Calc (NodeTp A, NodeTp B) {
    //printf("Calc :  \n"); A.Print(); B.Print(); printf("\n");
    double a = (A.x - B.x) * (A.x - B.x) + (A.y - B.y) * (A.y -
B.y);
    double b = 2 * ( (B.x - A.x) * A.x + (B.y - A.y) * A.y);
    double c = A.x * A.x + A.y * A.y - r * r, d = b * b - 4 * a *
```

```
c;
    area[++m] = A;
    if (d < -eps) return;
    d = sqrt (d);
    double t1 = (-b + d) / (a + a);
    double t2 = - (b + d) / (a + a);
    if (t1 > t2) swap (t1, t2);
    // printf("(%.3lf, %.3lf)\n", t1, t2);
    if (t1 > eps && t1 < 1 - eps) area[++m] = NodeTp (A.x + (B.x -
A.x) * t1, A.y + (B.y - A.y) * t1);
    if (t2 > eps && t2 < 1 - eps) area[++m] = NodeTp (A.x + (B.x -
A.x) * t2, A.y + (B.y - A.y) * t2);
    return;
}

int main() {
    freopen ("mammoth.in", "r", stdin);
    freopen ("mammoth.out", "w", stdout);
    scanf ("%d%d%lf", &x, &y, &r);
    scanf ("%d%d%d%d", &x1, &y11, &x2, &y2);
    O.x = 0.0;
    O.y = 0.0;
    nod[1].x = x1 - x, nod[1].y = y11 - y;
    nod[2].x = x2 - x, nod[2].y = y11 - y;
    nod[3].x = x2 - x, nod[3].y = y2 - y;
    nod[4].x = x1 - x, nod[4].y = y2 - y;
    n = 4;
    for (int i = 1; i < n; i++) Calc (nod[i], nod[i + 1]);
    Calc (nod[n], nod[1]);
    area[++m] = area[1];
    //for(int i=1;i<=m;i++) area[i].Print(); printf("\n");
    for (int i = 1; i < m; i++)
        if (hypot ( (area[i].x + area[i + 1].x) *.5, (area[i].y +
area[i + 1].y) *.5) > r - eps)
            sumarea += Sector (area[i], area[i + 1]);
        else sumarea += Cross (area[i], area[i + 1]) * .5;
    printf ("%.12lf\n", fabs (sumarea) );
    return 0;
}
```

自适应辛普森

```
double eps = 1e-6;

double f (double x) {
    //数学函数
}

double Simpson (double a, double b) {
    double c = (a + b) / 2;
    return (f (a) + 4 * f (c) + f (b) ) / 6 * (b - a);
}
double AdaptiveSimpson (double a, double b, double eps) {
    double c = (a + b) / 2;
    double S = Simpson (a, b);
    double S1 = Simpson (a, c);
    double S2 = Simpson (c, b);
    if (fabs (S1 + S2 - S) < 15 * eps) return S;
    else return AdaptiveSimpson (a, c, eps / 2) + AdaptiveSimpson
(c, b, eps / 2);
}
```

CRT 中国剩余定理

```
//有同余方程组 x=ai(%mi)
//这个板子 mi 不互质也可以用
//设 M=m1*m2*…*mn, Mi=M/mi
//ti=Mi^(-1) (%mi 意义下的逆元)
//通解 x=sigma(ai*ti*Mi)+k*M
//在模 M 意义下仅有一个解
void Gcd (intl a, intl b) {
    if (!b) {
        x = 1LL;
        y = 0;
        return;
    }
    Gcd (b, a % b);
    swap (x, y);
    y = y - (a / b) * x;
}
intl gcd (intl a, intl b) {
    return (b ? gcd (b, a % b) : a);
}
void upt (intl &x, intl a) {
```

```
    x = (x % a + a) % a;
}
intl C (intl a, intl b, intl Mo) {
    intl ans = 0;
    while (b) {
        if (b & 1LL) (ans += a) %= Mo;
        (a += a) %= Mo;
        b >>= 1;
    }
    return ans;
}
int main() {
    while (scanf ("%d", &n) != EOF) {
        for (i = 1; i <= n; i++) scanf ("%lld %lld", &A[i], &R[i]);
        for (i = 2; i <= n; i++) {
            if (R[i] < R[1]) swap (R[i], R[1]), swap (A[i], A[1]);
            js = gcd (A[1], A[i]);
            if ( (R[i] - R[1]) % js) break;
            Gcd (A[1] / js, A[i] / js);
            upt (x, A[i] / js);
            x *= (R[i] - R[1]) / js;
            a = A[1] * A[i] / js;
            b = (C (x, A[1], a) + R[1]) % a;
            A[1] = a;
            R[1] = b;
        }
        if (i <= n) printf ("-1\n");
        else printf ("%lld\n", (R[1] % A[1] + A[1]) % A[1]);
    }
    return 0;
}
```

EXGCD:

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

```
    return r;
}
```

exGSBS：

```
    A^B≡C (mod P)
→   A/t * A^(B-1)≡C/t (mod P/t)
    使 t=gcd(A,P) until t=1;

    若 C mod t≠0 则判无解

    根据欧拉定理 A ^ phi(P)≡1 (mod P)(当 gcd(A,P)=1 时)
    所以能求得逆元

    粗略的证明:
    设 D=A^B
    若 D≡C (mod P)
    ∴D=k*P+C (k∈Z)
    ∴D/t=k*(P/t)+C/t
    即 D/t≡C/t (mod P/t)
    反推也一样

    ·但这样还是会有反例
    设我们消了 k 次 t，那么我们得到的式子为
    设 L=t1*t2*t3...tk
    ((A ^ k) / L) * (A ^ (B - k)) ≡ C / L (mod P/L)
    用普通 BSGS 求出（B-k），但这时求出的 B 一定≥k，会漏解
    解决方法显然，k≤log P，只要先暴力验证较小范围内的数就行了
```

FFT——GY：

```
//FFT 优化高精度乘法

#include <cstring>
#include <cstdlib>
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <ctime>
#include <cmath>
using namespace std;

const int con = 100000;
#define pi 3.14159265358979323846264338327
```

```
int N, lena, lenb, test;
int rev[600010];
char sa[600010], sb[600010];
class complex {
  public:
    long double re, im;
    void getdata (long double x, long double y) {
        re = x;
        im = y;
    }
    complex operator + (const complex &X) {
        complex c;
        c.re = re + X.re;
        c.im = im + X.im;
        return c;
    }
    complex operator - (const complex &X) {
        complex c;
        c.re = re - X.re;
        c.im = im - X.im;
        return c;
    }
    complex operator * (const complex &X) {
        complex c;
        c.re = re * X.re - im * X.im;
        c.im = re * X.im + im * X.re;
        return c;
    }
} a[600010], b[600010], W[2][600010];

void FFT (complex *a, int f) {
    complex x, y;
   for (int i = 0; i < N; i++) if (i < rev[i]) swap (a[i], a[rev[i]]);
    for (int i = 1; i < N; i <<= 1) {
        for (int j = 0, t = N / (i << 1); j < N; j += i << 1)
            for (int k = 0, l = 0; k < i; l += t, k++) {
                x = W[f][l] * a[j + k + i];
                y = a[j + k];
                a[j + k] = y + x;
                a[j + k + i] = y - x;
            }
    }
    if (f) for (int i = 0; i < N; i++) a[i].re /= N;
}
```

```
int main() {
    freopen ("2179.in", "r", stdin);
    freopen ("2179.out", "w", stdout);
    scanf ("%d", &lena);
    scanf ("%s%s", sa, sb);
    lena = strlen (sa);
    lenb = strlen (sb);
    for (int i = 0; i <= (lena - 1) / 5; i++)
        for (int j = 4; j >= 0; j--)
            if (lena - i * 5 - j > 0)
                a[i].re = a[i].re * 10 + sa[lena - 1 - i * 5 - j] -
'0';
    lena = (lena - 1) / 5 + 1;
    for (int i = 0; i <= (lenb - 1) / 5; i++)
        for (int j = 4; j >= 0; j--)
            if (lenb - i * 5 - j > 0)
                b[i].re = b[i].re * 10 + sb[lenb - 1 - i * 5 - j] -
'0';
    lenb = (lenb - 1) / 5 + 1;


    //预处理部分
    for (N = 1; N < lena || N < lenb; N <<= 1);
    N <<= 1;
    for (int i = 0; i < N; i++) {
        int x = i, y = 0;
        for (int k = 1; k < N; x >>= 1, k <<= 1) (y <<= 1) |= x & 1;
        rev[i] = y;
    }
    for (int i = 0; i < N; i++) {
        W[0][i].getdata (cos (2 * pi * i / N), sin (2 * pi * i /
N) );
        W[1][i].im = -W[0][i].im;
        W[1][i].re = W[0][i].re;
    }

    //FFT 主代码
    FFT (a, 0);
    FFT (b, 0);
    for (int i = 0; i < N; i++)
        a[i] = a[i] * b[i];
    FFT (a, 1);
```

```
    for (int i = 0; i < N; i++) {
        long double t = (a[i].re + 0.5) / con;
        t = floor (t);
        a[i].re = a[i].re - t * con;
        if (abs (a[i].re) < 1e-3) a[i].re = 0;
        a[i + 1].re = a[i + 1].re + t;
    }
    while (N && abs (a[N].re) < 1e-3) N--;
    printf ("%.0lf", double (a[N].re) );
    for (int i = N - 1; i >= 0; i--) {
//      if (fabs(a[i].re)<1e-3)a[i].re=0;
        printf ("%05.0lf", double (a[i].re) );
    }
}
```

NTT:

```
#include <bits/stdc++.h>
using namespace std;
#define LL long long
#define intl long long
const LL P = 998244353;
const LL G = 3;
int N, M, Test;
LL W[2][400010];
LL A[400010], B[400010], fac[400010], ni[400010], po[400010];
int rev[400010];
int a[400010];

LL Pow (LL a, LL b) {
    LL c = 1;
    for (; b; b >>= 1, a = a * a % P)
        if (b & 1) c = c * a % P;
    return c;
}

void FFT (LL *a, int f) {
    for (int i = 0; i < M; i++)
        if (i < rev[i]) swap (a[i], a[rev[i]]);
    for (int i = 1; i < M; i <<= 1)
        for (int j = 0, t = M / (i << 1); j < M; j += i << 1)
            for (int k = 0, l = 0; k < i; k++, l += t) {
                LL x, y;
```

```
                x = W[f][l] * a[j + k + i] % P;
                y = a[j + k];
                a[j + k] = (y + x) % P;
                a[j + k + i] = (y - x + P) % P;
            }
    if (f)
        for (int i = 0, x = Pow (M, P - 2); i < M; i++)
            a[i] = a[i] * x % P;
}

void work() {
    for (int i = 0; i < M; i++) {
        int x = i, y = 0;
        for (int k = 1; k < M; x >>= 1, k <<= 1) (y <<= 1) |= x & 1;
        rev[i] = y;
    }
    W[0][0] = W[1][0] = 1;
    LL x = Pow (G, (P - 1) / M), y = Pow (x, P - 2);
    for (int i = 1; i < M; i++) {
        W[0][i] = x * W[0][i - 1] % P;
        W[1][i] = y * W[1][i - 1] % P;
    }
}

void Init() {
    memset (A, 0, sizeof (A) );
    memset (B, 0, sizeof (B) );
    for (int i = 0; i <= N - 1; i++) B[i] = ni[i];
    for (int i = 0; i <= N - 1; i++) {
        A[i] = a[N - i];
        A[i] *= fac[N - i - 1];
        A[i] %= P;
        A[i] *= po[i];
        A[i] %= P;
    }
}

void pri() {
    for (int i = 0; i < N; i++) printf ("%I64d ", A[i]);
    putchar ('\n');
}

int main() {
    freopen ("I.in", "r", stdin);
```

```
    freopen ("I.out", "w", stdout);
    scanf ("%d", &Test);
    fac[0] = 1LL;
    po[0] = 1LL;

    for (int i = 1; i <= 100000; i++) po[i] = po[i - 1] * 2LL % P;
    for (int i = 1; i <= 100000; i++) fac[i] = fac[i - 1] * (intl)
i % P;

    ni[100000] = Pow (fac[100000], P - 2LL);

    for (int i = 99999; i >= 0; i--)
        ni[i] = (ni[i + 1] * (intl) (i + 1) ) % P;

    for (int tt = 1; tt <= Test; tt++) {
        scanf ("%d", &N);
        for (M = 1; M <= N; M <<= 1);
        M <<= 1;
        for (int i = 1; i <= N; i++) scanf ("%d", &a[i]);
        sort (a + 1, a + N + 1);
        reverse (a + 1, a + N + 1);
        Init();

        work();

        FFT (A, 0);
        FFT (B, 0);
        for (int i = 0; i < M; i++) A[i] = A[i] * B[i] % P;
        FFT (A, 1);

        for (int i = 0; i < N; i++) A[i] = (A[i] * ni[N - i - 1]) %
P;
        reverse (A, A + N);
        for (int i = 1; i < N; i++) A[i] += A[i - 1], A[i] %= P;
        pri();
    }
    return 0;
}
```

GAUSS:

```
const int MaxN = 307;

int n, m;
int A[MaxN][MaxN], B[MaxN], X[MaxN];

inline void Swap (int &x, int &y) {
    int t = x;
    x = y;
    y = t;
    return;
}

void Gauss() {
    int i, j, k, p, m1, m2;
    bool flag, fl;
    flag = false;
    for (i = 1, p = 1; i <= m; p++) { //´¦ÀíµÚi¸ö·½³Ì ÏûÔªµÚp¸öÎ´ÖªÊý
        for (j = i + 1, k = i; j <= m; j++)
            if (A[j][p] > A[k][p]) k = j;
        if (A[k][p] != 0) {
            if (k != i) {
                Swap (B[i], B[k]);
                for (j = p; j <= n; j++) Swap (A[i][j], A[k][j]);
            }
            for (j = i + 1; j <= m; j++) {
                m1 = A[i][p];
                m2 = A[j][p];
                fl = true;
                for (k = p; k <= n; k++) {
                    A[j][k] = (A[j][k] * m1 - A[i][k] * m2) % 7;
                    if (A[j][k] < 0) A[j][k] += 7;
                    fl = fl & (A[j][k] == 0);
                }
                B[j] = (B[j] * m1 - B[i] * m2) % 7;
                if (B[j] < 0) B[j] += 7;
                if (fl)
                    if (B[j] != 0) {
                        printf ("Inconsistent data.\n");
                        return;
                    } else flag = true;
            }
            i++;
```

```
        }
    }
    if (p < n) {
        printf ("Multiple solutions.\n");
        return;
    }
    if (A[n][n] == 0 && B[n] == 0) {
        printf ("Multiple solutions.\n");
        return;
    }
    X[n] = Calc (A[n][n], B[n]);
    for (i = n - 1; i >= 1; i--) {
        for (j = i + 1, k = 0; j <= n; j++)
            k += A[i][j] * X[j];
        k = B[i] - k;
        k = ( (k % 7) + 7) % 7;
        if (A[i][i] == 0)
            if (k) {
                printf ("Inconsistent data.\n");
                return;
            } else {
                printf ("Multiple solutions.\n");
                return;
            }
        X[i] = Calc (A[i][i], k);
    }
    for (i = 1; i <= n; i++) printf ("%d ", X[i]);
    printf ("\n");
    return;
}
```

Geometry

向量旋转／距离

```
struct point {
    double x, y;
    point (double x = 0, double y = 0) : x (x), y (y) {}
};
int sig (double a) {
    return (a < -eps ? -1 : (a > eps) );
}
//默认 vector 就是 point ,就不 typedef 了
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);
}
double operator * (point a, point b) {
    return a.x * b.x + a.y * b.y;    // 点积
}
double operator / (point a, point b) {
    return a.x * b.y - a.y * b.x;    // 叉积 b 在 a 左边叉积
}
> 0
bool operator == (const point &a, const point &b) {
    return !sig (a.x - b.x) && !sig (a.y - b.y);
}
double length (point a) {
    return sqrt (a * a);    // 求向量长度
}
double angle (point a, point b) {
    return acos (a * b / length (a) / length (b) );    // 两向量夹角
}
(弧度)
//向量旋转|| 旋转坐标系
point rotate (point a, double rad) {
    return point (a.x * cos (rad) - a.y * sin (rad), a.x * sin (rad)
+ a.y * cos (rad) );
} // cos - sin, sin + cos.... 好记
//----------------直线相关----------------
// 直线的参数法表示:
struct line {
    point p, v; // 直线上任意一点 x 满足 x = p + v * t;
};
// 直线求交
point cross_line (line a, line b) {
    point u = a.p - b.p;
    if (!sig (a.v / b.v) ) return point (INF, INF);
    double t = (b.v / u) / (a.v / b.v);
    return a.p + a.v * t;
}
```

```
// 点到直线的距离 -> 叉积得到的有向面积除以底
double dis_dot_line (point p, line A) {
    point a = A.p, b = A.p + A.v;
    point v1 = b - a, v2 = p - a;
    return fabs ( (v1 / v2) / length (v1) );
}
// 点到线段的距离 分三种情况考虑
double dis_dot_seg (point p, point a, point b) {
    if (a == b) return length (p - a);
    point v1 = b - a, v2 = p - a, v3 = p - b;
    if (sig (v1 * v2) < 0) return length (v2);
    else if (sig (v1 * v2) > 0) return length (v3);
    else return fabs ( (v1 / v2) / length (v1) );
}
// 点在直线上的投影点 列方程求解
point dot_projection (point a, line b) {
    return b.p + b.v * ( (b.v * (a - b.p) ) / (b.v * b.v) );
}
//计算多边形的有向面积-- 虽替!
double count_s (point *p, int n) {
    double ans = 0;
    for (int i = 1; i < n; i++) ans += p[i] / p[i + 1];
    ans += p[n] / p[1];
    return fabs (ans / 2.0);
}

//以下来自 GY
//poi3 为三维向量
//绕轴极角方向旋转 selta
poi3 rotatex (double selta) {
    poi3 c;
    c.x = x;
    c.y = y * cos (selta) - z * sin (selta);
    c.z = y * sin (selta) + z * cos (selta);
    return c;
}
poi3 rotatey (double selta) {
    poi3 c;
    c.y = y;
    c.z = z * cos (selta) - x * sin (selta);
    c.x = z * sin (selta) + x * cos (selta);
    return c;
}
poi3 rotatez (double selta) {
```

```
    poi3 c;
    c.z = z;
    c.x = x * cos (selta) - y * sin (selta);
    c.y = x * sin (selta) + y * cos (selta);
    return c;
}
```

MillerRabin & Pollard‘s rho:

```
const LL pp = 1000000007;
LL pri[10] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};

LL gcd (LL x, LL y) {
    while (y) {
        LL t = x % y;
        x = y;
        y = t;
    }
    return x;
}

LL product_mod (LL x, LL y, LL n) {
    LL s = 0, t = x;
    for (; y; y >>= 1) {
        if (y & 1) s = (s + t) % n;
        t = t * 2 % n;
    }
    return s;
}

LL power_mod (LL x, LL y, LL n) {
    x %= n;
    LL s = 1, t = x;
    for (; y; y >>= 1) {
        if (y & 1) s = product_mod (s, t, n);
        t = product_mod (t, t, n);
    }
    return s;
}

//判断点N是否为质数
bool Miller_Rabin (LL N) {
    if (N < 2) return false;
    if (N == 2) return true;
```

```
    if (! (N & 1) ) return false;
    LL m, k = 0, a;
    m = N - 1;
    while (! (m & 1) ) k++, m = m / 2;
    for (LL i = 0; i < 10; i++) {
        if (pri[i] >= N) break;
        a = power_mod (pri[i], m, N);
        if (a == 1) continue;
        LL j;
        for (j = 0; j < k; j++) {
            if (a == N - 1) break;
            a = product_mod (a, a, N);
        }
        if (j == k) return false;
    }
    return true;
}

LL pollard_rho (LL C, LL N) {
    LL i, k, x1, x2, d;
    i = 1;
    k = 2;
    x1 = x2 = rand() % N;
    do {
        i++;
        d = gcd (x2 - x1 + N, N);
        if (d > 1 && d < N) return d;
        if (i == k) k = k << 1, x2 = x1;
        x1 = (product_mod (x1, x1, N) + N - C) % N;
    } while (x1 != x2);
    return N;
}

//对N分解质因数 结果存储于q[]
void rho (LL N) {
    if (N < 2) return;
    if (Miller_Rabin (N) ) {
        q[++last] = N;
        return;
    }
    LL T;
    do {
        T = pollard_rho (rand() % (N - 1) + 1, N);
    } while (T == N);
```

```
    rho (N / T);
    rho (T);
}
```

第二类斯特林数通项:S(n,m)=sigma((-1)^k*C(m,k)*(m-k)^n)/(m!) 0<=k<=m

任意四面体体积:

已知任意四面体（三棱锥）六条棱的棱长，求其体积。
不妨记同一顶点引出的三条棱棱长的平方分别为 a,b,c,它们的对棱棱长的平方分别为 d，e，f，则四面体的体积 V 满足:
V=sqrt[ad(b+c+e+f-a-d)+be(a+c+d+f-b-e)+cf(a+b+d+e-c-f)-abf-bcd-cae-def)]/12
证明的话，有空再发。
补充一些特殊四面体的体积公式:
①直角四面体（三条侧棱两两互相垂直，记其长分别为 a,b,c）: V=abc/6
②正四面体：棱长为 a，则 V=a^3*sqrt(2)/12
③等腰四面体（三组对棱都相等，记每组对棱的长分别为 a,b,c，p=(a^2+b^2+c^2)/2）
V=sqrt[(p-a)(p-b)(p-c)]

1 楼给出的体积公式，看似繁杂而冗长，其实非常有规律，十分方便记忆。首先记根号外的系数 1/12；其次记根号里任何字母的次数都是 2。然后，根号内由两部分组成，前面三个括号分别为【对棱平方积】乘以【其他四边平方和减去这两边的平方和】。最后四个，分别为四面体四个面三角形【三边平方积】。搞定!
准确的说，根号里字母的次数是 1（因为已经设定了字母是棱长的平方）。

组合数:

```
//都挺显然的
sigma(C(i,m)) = C(n+1,m+1)  m<=i<=n
sigma(C(i,k)) = C(n+m+1,k+1)-C(n,k+1)    n<=i<=n+m
sigma(C(n,i)*C(m,p-i)) = C(n+m,p) 0<=i<=p
sigma(k*C(n,k)) = n*2^(n-1) 1<=k<=n

lucas 定理:
C(n,m) = C(n/p,m/p)*C(n%p,m%p) (%P)


二项式反演:
f(n) = ∑(C(n,k)*g(k))  0<=k<=n
g(n) = ∑((-1)^(n-k)*C(n,k)*g(k))  0<=k<=n
```

NIM_K 问题:

Moore's Nim_k 问题，是每个人每次可以从最多 k 堆中拿取任意多的石子，不同的堆可以不一样。
经典 Nim 问题实际上是 Moore's Nim 1 问题。

有如下结论:
对于每一堆，把它石子的个数用二进制表示；对所有的石子堆，如果在任何一个二进制位上 1 的个数总是 k+1 的整数倍，则是必败状态，反之则是必胜状态。

(1)
终止状态必然为 P 状态：如果不是 P 状态，则说明在某一位上至少有一堆石子不是空，则显然它不会是结束状态。

(2)
任何一个 P 状态，经过一次操作以后必然会到达 N 状态：在某一次移动中，至少有一堆被改变，也就是说至少有一个二进制位被改变。由于最多只能改变 k 堆石子，所以对于任何一个二进制位，1 的个数至多改变 k。而由于原先的总数为 k+1 的整数倍，所以改变之后必然不可能是 k+1 的整数倍。故在 P 状态下一次操作的结果必然是 N 状态。

(3)
任何一个 N 状态，总有一种操作可以使其变为 P 状态：从高位到低位考虑所有的二进制位。假设用了某种方法，改变了 m 堆，使得 X 位之前的所有位都可以回归到 k+1 的整数倍。现在要证明总有一种方法让第 X 位也恢复到 k+1 的整数倍。

我们会发现这样一个性质：对于那些已经改变的 m 堆，它们的第 X 位，我们既可以设置成 0 也可以设置成 1，这个结论在后面证明。这样这些堆的 X 位是可以自由选择的，我们不去考虑它。除去这 m 堆之后，剩下堆在 X 位上 1 的总和 sum 又有两种情况，分别讨论:

1. sum <= k-m。此时可以将这些堆上的 1 全部拿掉，然后将那 m 堆的 X 位全设置成 0 。由于之前改变了 m 堆，现在新改变了 sum 堆，所以总共改变了 sum+m <= k 堆，满足题目要求，所以是可以达到的。

2. sum > k-m。此时我们在之前改变的 m 堆中选择 k+1-sum 堆，将它们的 X 位设置成 1，剩下的设置成 0。由于 k+1-sum < k+1-(k-m) < m+1，也就是说 k+1-sum <= m，故这也是可以达到的。

这样我们会发现，总有一种方法可以在满足题目要求的情况下让第 X 位 1 的总和也回归到 k+1 的整数倍，所以从高到低考察每一位，必然会有一种方法可以让所有的位回归到 k+1 的整数倍，也就是达到 P 状态。

以上三条即可以证明，上文所定义的 N 和 P 状态确实是满足条件的，故 Moore's Nim k 的结论成立。

下面证明一下上文中那个未证的问题，即在高位取走了一些石子之后，我们就可以随意指定低位上的数字。在第 X 位时，高位取走的石子数必然至少为 2^(X+1)个，这个数是大于 2^X 的。如果当前第 X 位是 1，那么我们可以通过取(结果为 0)或不取(结果为 1)来任意指定；如果当前第 X 位是 0,那么我们可以通过不取(结果为 0)或放回 2^X 个(结果为 1)来调整。所以我们可以随意指定第 X 位上的数字。

misereversion of Nim_K 如果无路可走则获胜

分三种情况讨论：

(1)

所有的堆（非零堆，下同）全是 1。此时就相当于一个经典的 Nim 游戏，每次最多可以取走 k 个石子。

按照经典 misere version of Nim 的结论可以判断出此时的胜负方法。

此时如果 1 堆个数模 k+1 的结果是 1 则必败，否则必胜。

(2)

有最少 1 个堆，最多 k 个堆的个数大于 1。这时是必胜态。

注意到状态 1 和胜负相关的唯一因素是 1 堆个数模 k+1 的结果。我们可以通过拿走 0 到 k 个堆来随意调整当前状态模的结果，然后再将所有大于 1 的堆降到 1 就行了。所以总有一种方法可以达到状态 1 中的必胜态。

(3) 有多于 k 个堆的个数大于 1。

这时可以先按照普通 Nim_k 的方法去走。

由于每次大于 1 的堆最多减少 k 个，所以最后必然会走到状态 2。

此时应用状态 2 的必胜策略即可获胜。

注意到状态 2 必然是原始 Nim_k 下的 N 状态，也就是至少有一个二进制位不是 k+1 的整数倍。

则可知此时原始 Nim_k 下的 N 状态仍然是 misere version 的 N 状态。

特别说明的是，在上面的讨论中如果将 k=1 代入，和经典 misere version of Nim 的解决方法完全相同。

Polya 定理：

设 G 是 n 个对象的一个置换群

用 m 种颜色给这 n 个对象染色

则不同的染色方案数为：

L=(m ^ c(P1) + m ^ c(P2) + … + m ^ c(Pg)) / |G|

其中：

|G|为置换个数

c(Pi)为 Pi 置换的循环节数

Burnside 引理:

设 G 是 n 个对象的一个置换群

每个置换都写成不相交循环的乘积　c(Pi)是在置换 Pi 的作用下不动点的个数，也就是长度为 1 的循环的个数。

通过上述置换的变换操作后可以相等的元素属于同一个等价类

若 G 将[1,n]划分成 L 个等价类

则:

等价类个数为: L = (c(P1) + c(P2) + … + c(Pg)) / |G|

牛顿迭代法

（1）选择一个接近零点的 X0

（2）Xi+1 = Xi - f(Xi) / f'(Xi)

（3）平方收敛，每迭代一次，有效数字将增加一倍

x^k 与 Cxk: x^k=sigma(i=0~k)(i!*C(x,i)*S(k,i))