离散对数所有解 / 离散对数非0最小解

```
#include <iostream>
#include <string.h>
#include <algorithm>
#include <stdio.h>
#include <math.h>
#include <bitset>
#include <map>

using namespace std;
typedef long long LL;

const int N = 1000005;

/**以下为求原根部分 */
bitset<N> prime;
int p[N],pri[N];
int k,cnt;

void isprime()
{
    prime.set();
    for(int i=2; i<N; i++)
    {
        if(prime[i])
        {
            p[k++] = i;
            for(int j=i+i; j<N; j+=i)
                prime[j] = false;
        }
    }
}

void Divide(LL n)
{
    cnt = 0;
    LL t = (LL)sqrt(1.0*n);
    for(int i=0; p[i]<=t; i++)
    {
        if(n%p[i]==0)
        {
            pri[cnt++] = p[i];
            while(n%p[i]==0) n /= p[i];
        }
    }
    if(n > 1)
        pri[cnt++] = n;
}

LL multi(LL a,LL b,LL m)
{
    LL ans = 0;
    a %= m;
    while(b)
```

```
    {
        if(b & 1)
        {
            ans = (ans + a) % m;
            b--;
        }
        b >>= 1;
        a = (a + a) % m;
    }
    return ans;
}

LL quick_mod(LL a,LL b,LL m)
{
    LL ans = 1;
    a %= m;
    while(b)
    {
        if(b&1)
        {
            ans = multi(ans,a,m);
            b--;
        }
        b >>= 1;
        a = multi(a,a,m);
    }
    return ans;
}

LL broot(LL p)
{
    Divide(p-1);
    for(int g=2 ;; g++)
    {
        bool flag = true;
        for(int i=0; i<cnt; i++)
        {
            LL t = (p - 1) / pri[i];
            if(quick_mod(g,t,p) == 1)
            {
                flag = false;
                break;
            }
        }
        if(flag) return g;
    }
}
/**以上为求原根部分 */

/** 以下为Baby_Step */

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

```
void extend_Euclid(LL a,LL b,LL &x,LL &y)
{
    if(b == 0)
    {
        x = 1;
        y = 0;
        return;
    }
    extend_Euclid(b,a%b,x,y);
    LL tmp = x;
    x = y;
    y = tmp - (a / b) * y;
}

LL Inv(LL a,LL p)
{
    return quick_mod(a,p-2,p);
}

LL Baby_Step(LL A,LL B,LL C)
{
    map<LL,int> mp;
    LL M = ceil(sqrt(1.0*C));
    LL t = Inv(quick_mod(A,M,C),C);
    LL ans = 1;
    for(int i=0; i<M; i++)
    {
        if(!mp.count(ans))
            mp[ans] = i;
        ans = multi(ans,A,C);
    }
    for(int i=0; i<M; i++)
    {
        if(mp.count(B))
            return i * M + mp[B];
        B = multi(B,t,C);
    }
    return -1;
}
/** 以上为Baby_Step */

LL ans[1005];

void Work(LL A,LL B,LL C)
{
    LL root = broot(C);
    LL t1 = Baby_Step(root,B,C);
    LL t2 = C - 1;
    LL d = gcd(A,t2);
    if(t1 % d)
    {
        puts("-1");
        return;
    }
```

```
    LL x,y;
    extend_Euclid(A,t2,x,y);
    t2 /= d;
    t1 /= d;
    ans[0] = (x * t1 % t2 + t2) % t2;
    for(int i=1; i<d; i++)
        ans[i] = ans[i-1] + t2;
    for(int i=0; i<d; i++)
        ans[i] = quick_mod(root,ans[i],C);
    sort(ans,ans+d);
    for(int i=0; i<d; i++)
        cout<<ans[i]<<endl;
}

int main()
{
    int T = 1;
    LL A,B,C;
    isprime();
    while(cin>>A>>C>>B)
    {
        printf("case%d:\n",T++);
        Work(A,B,C);
    }
    return 0;
}

A^y = B (%C)


#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<cmath>
#define LL long long
#define N 1000000
using namespace std;
struct Node{
    int idx;
    int val;
}baby[N];
bool cmp(Node n1,Node n2){
    return n1.val!=n2.val?n1.val<n2.val:n1.idx<n2.idx;
}
int gcd(int a,int b){
    return b==0?a:gcd(b,a%b);
}
int extend_gcd(int a,int b,int &x,int &y){
    if(b==0){
        x=1;
        y=0;
        return a;
    }
    int gcd=extend_gcd(b,a%b,x,y);
```

```
    int t=x;
    x=y;
    y=t-a/b*y;
    return gcd;
}
int inval(int a,int b,int n){
    int e,x,y;
    extend_gcd(a,n,x,y);
    e=((LL)x*b)%n;
    return e<0?e+n:e;
}
int PowMod(int a,int b,int MOD){
    LL ret=1%MOD,t=a%MOD;
    while(b){
        if(b&1)
            ret=((LL)ret*t)%MOD;
        t=((LL)t*t)%MOD;
        b>>=1;
    }
    return (int)ret;
}
int BinSearch(int num,int m){
    int low=0,high=m-1,mid;
    while(low<=high){
        mid=(low+high)>>1;
        if(baby[mid].val==num)
            return baby[mid].idx;
        if(baby[mid].val<num)
            low=mid+1;
        else
            high=mid-1;
    }
    return -1;
}
int BabyStep(int A,int B,int C){
    LL tmp,D=1%C;
    int temp;
    for(int i=0,tmp=1%C;i<100;i++,tmp=((LL)tmp*A)%C)
        if(tmp==B)  // 返回非0变成 tmp == B && i
            return i;
    int d=0;
    while((temp=gcd(A,C))!=1){
        if(B%temp) return -1;
        d++;
        C/=temp;
        B/=temp;
        D=((A/temp)*D)%C;
    }
    int m=(int)ceil(sqrt((double)C));
    for(int i=0,tmp=1%C;i<=m;i++,tmp=((LL)tmp*A)%C){
        baby[i].idx=i;
        baby[i].val=tmp;
    }
    sort(baby,baby+m+1,cmp);
    int cnt=1;
```

```
    for(int i=1;i<=m;i++)
        if(baby[i].val!=baby[cnt-1].val)
            baby[cnt++]=baby[i];
    int am=PowMod(A,m,C);
    for(int i=0;i<=m;i++,D=((LL)(D*am))%C){
        int tmp=inval(D,B,C);
        if(tmp>=0){
            int pos=BinSearch(tmp,cnt);
            if(pos!=-1)     // 返回非0变成pos != -1 && i * m + pos + d
                return i*m+pos+d;
        }
    }
    return -1;
}
int main(){
    int A,B,C;
    while(scanf("%d%d%d",&A,&C,&B)!=EOF){
        if(B>=C){
            puts("Orz,I can’t find D!");
            continue;
        }
        int ans=BabyStep(A,B,C);
        if(ans==-1)
            puts("Orz,I can’t find D!");
        else
            printf("%d\n",ans);
    }
    return 0;
}
```

三维凸包（求一个面作为底面时的最高点和投影面积）

```
#include<stdio.h>
#include<iostream>
#include<string.h>
#include<string>
#include<ctype.h>
#include<math.h>
#include<set>
#include<map>
#include<vector>
#include<queue>
#include<bitset>
#include<algorithm>
#include<time.h>
using namespace std;
void fre() { freopen("c://test//input.in", "r", stdin); freopen("c://
test//output.out", "w", stdout); }
#define MS(x, y) memset(x, y, sizeof(x))
#define ls o<<1
#define rs o<<1|1
typedef long long LL;
typedef unsigned long long UL;
```

```
typedef unsigned int UI;
template <class T1, class T2>inline void gmax(T1 &a, T2 b) { if (b > a)a
= b; }
template <class T1, class T2>inline void gmin(T1 &a, T2 b) { if (b < a)a
= b; }
const int N = 1e3 + 10, M = 0, Z = 1e9 + 7, inf = 0x3f3f3f3f;
template <class T1, class T2>inline void gadd(T1 &a, T2 b) { a = (a + b)
% Z; }
int casenum, casei;
const LL mod = 365 * 24 * 60 * 60;
const double eps = 1e-8;
int n;


inline int sgn(double x)
{
    if(fabs(x) < eps) return 0;
    return x > 0 ? 1 : -1;
}
double sqr(double x)
{
    return x * x;
}

class point3
{
public:
    double x, y, z;
    point3(){}
    point3(double x, double y, double z) : x(x), y(y), z(z) {}
    double length() const{
        return sqrt(sqr(x) + sqr(y) + sqr(z));
    }
    double len2() const{
        return sqr(x) + sqr(y) + sqr(z);
    }
};
bool operator == (const point3 &a, const point3 &b)
{
    if(sgn(a.x - b.x) == 0 && sgn(a.y - b.y) == 0 && sgn(a.z - b.z) ==
0) return 1;
    return 0;
}
point3 operator - (const point3 &a, const point3 &b)
{
    return point3(a.x - b.x, a.y - b.y, a.z - b.z);
}
point3 operator + (const point3 &a, const point3 &b)
{
    return point3(a.x + b.x, a.y + b.y, a.z + b.z);
}

point3 operator / (const point3 &a, double b)
{
    return point3(a.x / b, a.y / b, a.z / b);
```

```
}
point3 operator * (const point3 &a, double b)
{
    return point3(a.x * b, a.y * b, a.z * b);
}
point3 det(const point3 &a, const point3 &b)
{
    return point3(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z, a.x *
b.y - a.y * b.x);
}
double dot(const point3 &a, const point3 &b)
{
    return a.x * b.x + a.y * b.y + a.z * b.z;
}
double mix(const point3 &a, const point3 &b, const point3 &c)
{
    return dot(a, det(b, c));
}
double vlen(point3 p)
{
    return p.length();
}
double dist(const point3 &a, const point3 &b)
{
    return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y) + sqr(a.z - b.z));
}

class line3
{
public:
    point3 a, b;
    line3(){}
    line3(point3 a, point3 b) : a(a), b(b) {}
};
struct plane3
{
public:
    point3 a, b, c;
    plane3(){}
    plane3(point3 a, point3 b, point3 c) : a(a), b(b), c(c) {}
};
point3 pvec(point3 s1, point3 s2, point3 s3)
{
    return det((s1 - s2), (s2 - s3));
}
point3 pvec(plane3 s)
{
    return det((s.a - s.b), (s.b - s.c));
}
double ptoline(point3 p, line3 l)
{
    return vlen(det(p - l.a, l.b - l.a)) / dist(l.a, l.b);
}
int parallel(line3 u, line3 v)
{
```

```
    return vlen(det(u.a - u.b, v.a - v.b)) < eps;
}
int parallel(plane3 u, plane3 v)
{
    return vlen(det(pvec(u), pvec(v))) < eps;
}
bool zero(double x)
{
    return fabs(x) < eps;
}
int parallel(line3 l, plane3 v)
{
    return zero(dot(l.a - l.b, pvec(v)));
}
point3 intersection(line3 l, plane3 s)
{
    point3 ret = pvec(s);
    double t = (ret.x * (s.a.x - l.a.x) + ret.y * (s.a.y - l.a.y) +
ret.z * (s.a.z - l.a.z)) / (ret.x * (l.b.x - l.a.x) + ret.y * (l.b.y -
l.a.y) + ret.z * (l.b.z - l.a.z));
    ret = l.a + (l.b - l.a) * t;
    return ret;
}
bool intersection(plane3 pl1, plane3 pl2, line3 &li)
{
    if(parallel(pl1, pl2)) return false;
    li.a = parallel(line3(pl2.a, pl2.b), pl1) ?
intersection(line3(pl2.b, pl2.c), pl1) : intersection(line3(pl2.a,
pl2.b), pl1);
    point3 fa;
    fa = det(pvec(pl1), pvec(pl2));
    li.b = li.a + fa;
    return true;
}
double ptoplane(point3 p, plane3 s)
{
    return fabs(dot(pvec(s), p - s.a)) / vlen(pvec(s));
}

#define SIZE(X) (int(X.size()))
int mark[1005][1005];
point3 info[1005];
int cnt;

double area(int a, int b, int c)
{
    return (det(info[b] - info[a], info[c] - info[a])).length();
}
double volume(int a, int b, int c, int d)
{
    return mix(info[b] - info[a], info[c] - info[a], info[d] - info[a]);
}
struct Face
{
    int a, b, c;
```

```
    Face(){}
    Face(int a, int b, int c) : a(a), b(b), c(c) {}
    int &operator [] (int k){
        if(k == 0) return a;
        if(k == 1) return b;
        return c;
    }
};
vector<Face> face;
inline void insert(int a, int b, int c)
{
    face.push_back(Face(a, b, c));
}
void add(int v)
{
    vector<Face> tmp;
    int a, b, c;
    cnt ++;
    for(int i = 0; i < SIZE(face); i ++){
        a = face[i][0];
        b = face[i][1];
        c = face[i][2];
        if(sgn(volume(v, a, b, c)) < 0){
            mark[a][b] = mark[b][a] = mark[b][c] = mark[c][b] = mark[c]
[a] = mark[a][c] = cnt;
        }
        else tmp.push_back(face[i]);
    }
    face = tmp;
    for(int i = 0; i < SIZE(tmp); i ++){
        a = face[i][0];
        b = face[i][1];
        c = face[i][2];
        if(mark[a][b] == cnt) insert(b, a, v);
        if(mark[b][c] == cnt) insert(c, b, v);
        if(mark[c][a] == cnt) insert(a, c, v);
    }
}

int Find()
{
    for(int i = 2; i < n; i ++){
        point3 ndir = det(info[0] - info[i], info[1] - info[i]);
        if(ndir == point3()) continue;
        swap(info[i], info[2]);
        for(int j = i + 1; j < n; j ++){
            if(sgn(volume(0, 1, 2, j)) != 0){
                swap(info[j], info[3]);
                insert(0, 1, 2);
                insert(0, 2, 1);
                return 1;
            }
        }
    }
    return 0;
```

```
}

bool operator < (const point3 &a, const point3 &b)
{
    return sgn(a.x - b.x) < 0 || sgn(a.x - b.x) == 0 && sgn(a.y - b.y) <
0 || sgn(a.x - b.x) == 0 && sgn(a.y - b.y) == 0 && sgn(a.z - b.z) < 0;
}
point3 proj(point3 p1, point3 p2, point3 q)
{
    return p1 + ((p2 - p1) * (dot((p2 - p1), (q - p1)) / (p2 -
p1).len2()));
}

const double PI = acos(-1.0);

struct point
{
    double x, y;
    point(){}
    point(double a, double b) : x(a), y(b) {}
    friend bool operator == (const point &a, const point &b){
        return sgn(a.x - b.x) == 0 && sgn(a.y - b.y) == 0;

    }
    friend point operator - (const point &a, const point &b){
        return point(a.x - b.x, a.y - b.y);
    }
    friend point operator + (const point &a, const point &b){
        return point(a.x + b.x, a.y + b.y);
    }

    friend point operator / (const point &a, double b){
        return point(a.x / b, a.y / b);
    }
    friend point operator * (const point &a, double b){
        return point(a.x * b, a.y * b);
    }
    double length(){
        return sqrt(sqr(x) + sqr(y));
    }
};
double det(const point &a, const point &b)
{
    return a.x * b.y - a.y * b.x;
}
double dot(const point &a, const point &b)
{
    return a.x * b.x + a.y * b.y;
}
point rotate_point(const point &p, double A)
{
    double tx = p.x, ty = p.y;
    return point(tx * cos(A) - ty * sin(A), tx * sin(A) + ty * cos(A));
}
point move_d(point t, line3 a, double len)
```

```
{
    point d = point(a.b.x - a.a.x, a.b.y - a.a.y);
    d = d / d.length();
    d = rotate_point(d, PI / 2);
    return t + d * len;
}
struct polygon_convex
{
    vector<point> p;
    polygon_convex(int size = 0){
        p.resize(size);
    }
}con;

bool comp_less(const point &a, const point &b)
{
    return sgn(a.x - b.x) < 0 || sgn(a.x - b.x) == 0 && sgn(a.y - b.y) <
0;
}
polygon_convex convex_hull(vector<point> a)
{
    polygon_convex res(2 * a.size() + 5);
    sort(a.begin(), a.end(), comp_less);
    a.erase(unique(a.begin(), a.end()), a.end());
    int m = 0;
    for(int i = 0; i < a.size(); i ++){
        while(m > 1 && sgn(det(res.p[m - 1] - res.p[m - 2], a[i] -
res.p[m - 2])) <= 0) -- m;
        res.p[m ++] = a[i];
    }
    int k = m;
    for(int i = int(a.size()) - 2; i >= 0; i --){
        while(m > k && sgn(det(res.p[m - 1] - res.p[m - 2], a[i] -
res.p[m - 2])) <= 0) -- m;
        res.p[m ++] = a[i];
    }
    res.p.resize(m);
    if(a.size() > 1) res.p.resize(m - 1);
    return res;
}

void D3_convex()
{
    sort(info, info + n);
    n = unique(info, info + n) - info;
    face.clear();
    random_shuffle(info, info + n);
    if(Find()){
        MS(mark, 0);
        cnt = 0;
        for(int i = 3; i < n; i ++) add(i);
        double ans = 0;
        double H = 0, AREA = 0;
        for(int i = 0; i < SIZE(face); i ++){   // 每个面
            double h = 0;
```

```
            plane3 cur = plane3(info[face[i][0]], info[face[i][1]],
info[face[i][2]]);
            for(int j = 0; j < n; j ++){
                gmax(h, ptoplane(info[j], cur));
            }
            point3 fa = pvec(cur);  // 法向量

            con.p.clear();
            for(int j = 0; j < n; j ++){
                line3 tmpl = line3(info[j], info[j] + fa);  //
                point3 tmp = intersection(tmpl, cur);   // 点在该平面上的映
射点
                line3 inter;    // 平面交线
                if(intersection(cur, plane3(point3(1.0, 1.0, 0),
point3(1.0, 0, 0), point3(0, 1.0, 0)), inter)){
                    double dis = ptoline(tmp, inter);   // 交点到直线距离,
移动距离
                    point3 interp = proj(inter.a, inter.b, tmp);    // 垂
足
                    point after;
                    if(tmp.z >= 0 ) after = move_d(point(interp.x,
interp.y), inter, dis);   //
                    else after = move_d(point(interp.x, interp.y),
inter, -dis);
                    con.p.push_back(after);
                }
                else{
                    con.p.push_back(point(tmp.x, tmp.y));
                }
            }
            con = convex_hull(con.p);
            con.p.push_back(con.p[0]);
            double area = 0;
            for(int i = 0; i < con.p.size() - 1; i ++){
                area += det(con.p[i], con.p[i + 1]);
            }
            area /= 2;

            if(h > H){
                H = h; AREA = area;
            }
            else if(h == H && area < AREA){
                AREA = area;
            }
        }
        printf("%.3f %.3f\n", H, AREA);
    }
}

int main()
{
    while(~scanf("%d", &n), n){
        for(int i = 0; i < n; i ++){
            scanf("%lf%lf%lf", &info[i].x, &info[i].y, &info[i].z);
```

```
        }
        D3_convex();
      }
      return 0;
}
/*
【trick&&吐槽】


【题意】


【分析】


【时间复杂度&&优化】

1.155
1.732

my:
1.155
1.443

*/
```

三维凸包（求任意光线角度下的最大投影面积，随机化光线角度）

```
#include<stdio.h>
#include<iostream>
#include<string.h>
#include<string>
#include<ctype.h>
#include<math.h>
#include<set>
#include<map>
#include<vector>
#include<queue>
#include<bitset>
#include<algorithm>
#include<time.h>
using namespace std;
void fre() { freopen("c://test//input.in", "r", stdin); freopen("c://
test//output.out", "w", stdout); }
#define MS(x, y) memset(x, y, sizeof(x))
#define ls o<<1
#define rs o<<1|1
typedef long long LL;
typedef unsigned long long UL;
typedef unsigned int UI;
template <class T1, class T2>inline void gmax(T1 &a, T2 b) { if (b > a)a
= b; }
```

```
template <class T1, class T2>inline void gmin(T1 &a, T2 b) { if (b < a)a
= b; }
const int N = 1e3 + 10, M = 0, Z = 1e9 + 7, inf = 0x3f3f3f3f;
template <class T1, class T2>inline void gadd(T1 &a, T2 b) { a = (a + b)
% Z; }
int casenum, casei;
const LL mod = 365 * 24 * 60 * 60;
const double eps = 1e-8;
int n;


inline int sgn(double x)
{
    if(fabs(x) < eps) return 0;
    return x > 0 ? 1 : -1;
}
double sqr(double x)
{
    return x * x;
}

class point3
{
public:
    double x, y, z;
    point3(){}
    point3(double x, double y, double z) : x(x), y(y), z(z) {}
    double length() const{
        return sqrt(sqr(x) + sqr(y) + sqr(z));
    }
    double len2() const{
        return sqr(x) + sqr(y) + sqr(z);
    }
};
bool operator == (const point3 &a, const point3 &b)
{
    if(sgn(a.x - b.x) == 0 && sgn(a.y - b.y) == 0 && sgn(a.z - b.z) ==
0) return 1;
    return 0;
}
point3 operator - (const point3 &a, const point3 &b)
{
    return point3(a.x - b.x, a.y - b.y, a.z - b.z);
}
point3 operator + (const point3 &a, const point3 &b)
{
    return point3(a.x + b.x, a.y + b.y, a.z + b.z);
}

point3 operator / (const point3 &a, double b)
{
    return point3(a.x / b, a.y / b, a.z / b);
}
point3 operator * (const point3 &a, double b)
{
```

```
    return point3(a.x * b, a.y * b, a.z * b);
}
point3 det(const point3 &a, const point3 &b)
{
    return point3(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z, a.x *
b.y - a.y * b.x);
}
double dot(const point3 &a, const point3 &b)
{
    return a.x * b.x + a.y * b.y + a.z * b.z;
}
double mix(const point3 &a, const point3 &b, const point3 &c)
{
    return dot(a, det(b, c));
}
double vlen(point3 p)
{
    return p.length();
}
double dist(const point3 &a, const point3 &b)
{
    return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y) + sqr(a.z - b.z));
}

class line3
{
public:
    point3 a, b;
    line3(){}
    line3(point3 a, point3 b) : a(a), b(b) {}
};
struct plane3
{
public:
    point3 a, b, c;
    plane3(){}
    plane3(point3 a, point3 b, point3 c) : a(a), b(b), c(c) {}
};
point3 pvec(point3 s1, point3 s2, point3 s3)
{
    return det((s1 - s2), (s2 - s3));
}
point3 pvec(plane3 s)
{
    return det((s.a - s.b), (s.b - s.c));
}
double ptoline(point3 p, line3 l)
{
    return vlen(det(p - l.a, l.b - l.a)) / dist(l.a, l.b);
}
int parallel(line3 u, line3 v)
{
    return vlen(det(u.a - u.b, v.a - v.b)) < eps;
}
int parallel(plane3 u, plane3 v)
```

```
{
    return vlen(det(pvec(u), pvec(v))) < eps;
}
bool zero(double x)
{
    return fabs(x) < eps;
}
int parallel(line3 l, plane3 v)
{
    return zero(dot(l.a - l.b, pvec(v)));
}
point3 intersection(line3 l, plane3 s)
{
    point3 ret = pvec(s);
    double t = (ret.x * (s.a.x - l.a.x) + ret.y * (s.a.y - l.a.y) +
ret.z * (s.a.z - l.a.z)) / (ret.x * (l.b.x - l.a.x) + ret.y * (l.b.y -
l.a.y) + ret.z * (l.b.z - l.a.z));
    ret = l.a + (l.b - l.a) * t;
    return ret;
}
bool intersection(plane3 pl1, plane3 pl2, line3 &li)
{
    if(parallel(pl1, pl2)) return false;
    li.a = parallel(line3(pl2.a, pl2.b), pl1) ?
intersection(line3(pl2.b, pl2.c), pl1) : intersection(line3(pl2.a,
pl2.b), pl1);
    point3 fa;
    fa = det(pvec(pl1), pvec(pl2));
    li.b = li.a + fa;
    return true;
}
double ptoplane(point3 p, plane3 s)
{
    return fabs(dot(pvec(s), p - s.a)) / vlen(pvec(s));
}

bool operator < (const point3 &a, const point3 &b)
{
    return sgn(a.x - b.x) < 0 || sgn(a.x - b.x) == 0 && sgn(a.y - b.y) <
0 || sgn(a.x - b.x) == 0 && sgn(a.y - b.y) == 0 && sgn(a.z - b.z) < 0;
}
point3 proj(point3 p1, point3 p2, point3 q)
{
    return p1 + ((p2 - p1) * (dot((p2 - p1), (q - p1)) / (p2 -
p1).len2()));
}
bool dots_inlline(point3 p1, point3 p2, point3 p3)
{
    return vlen(det(p1 - p2, p2 - p3)) == 0;
}

const double PI = acos(-1.0);

struct point
{
```

```
    double x, y;
    point(){}
    point(double a, double b) : x(a), y(b) {}
    friend bool operator == (const point &a, const point &b){
        return sgn(a.x - b.x) == 0 && sgn(a.y - b.y) == 0;

    }
    friend point operator - (const point &a, const point &b){
        return point(a.x - b.x, a.y - b.y);
    }
    friend point operator + (const point &a, const point &b){
        return point(a.x + b.x, a.y + b.y);
    }

    friend point operator / (const point &a, double b){
        return point(a.x / b, a.y / b);
    }
    friend point operator * (const point &a, double b){
        return point(a.x * b, a.y * b);
    }
    double length(){
        return sqrt(sqr(x) + sqr(y));
    }
};
double det(const point &a, const point &b)
{
    return a.x * b.y - a.y * b.x;
}
double dot(const point &a, const point &b)
{
    return a.x * b.x + a.y * b.y;
}
point rotate_point(const point &p, double A)
{
    double tx = p.x, ty = p.y;
    return point(tx * cos(A) - ty * sin(A), tx * sin(A) + ty * cos(A));
}
point move_d(point t, line3 a, double len)
{
    point d = point(a.b.x - a.a.x, a.b.y - a.a.y);
    if(d.length() == 0) return t;
    d = d / d.length();
    d = rotate_point(d, PI / 2);
    return t + d * len;
}
struct polygon_convex
{
    vector<point> p;
    polygon_convex(int size = 0){
        p.resize(size);
    }
}con;

bool comp_less(const point &a, const point &b)
{
```

```
    return sgn(a.x - b.x) < 0 || sgn(a.x - b.x) == 0 && sgn(a.y - b.y) <
0;
}
polygon_convex convex_hull(vector<point> a)
{
    polygon_convex res(2 * a.size() + 5);
    sort(a.begin(), a.end(), comp_less);
    a.erase(unique(a.begin(), a.end()), a.end());
    int m = 0;
    for(int i = 0; i < a.size(); i ++){
        while(m > 1 && sgn(det(res.p[m - 1] - res.p[m - 2], a[i] -
res.p[m - 2])) <= 0) -- m;
        res.p[m ++] = a[i];
    }
    int k = m;
    for(int i = int(a.size()) - 2; i >= 0; i --){
        while(m > k && sgn(det(res.p[m - 1] - res.p[m - 2], a[i] -
res.p[m - 2])) <= 0) -- m;
        res.p[m ++] = a[i];
    }
    res.p.resize(m);
    if(a.size() > 1) res.p.resize(m - 1);
    return res;
}
point3 info[N];

#define SIZE(X) (int(X.size()))
int mark[1005][1005];
int cnt;

double area(int a, int b, int c)
{
    return (det(info[b] - info[a], info[c] - info[a])).length();
}
double volume(int a, int b, int c, int d)
{
    return mix(info[b] - info[a], info[c] - info[a], info[d] - info[a]);
}
struct Face
{
    int a, b, c;
    Face(){}
    Face(int a, int b, int c) : a(a), b(b), c(c) {}
    int &operator [] (int k){
        if(k == 0) return a;
        if(k == 1) return b;
        return c;
    }
};
vector<Face> face;
inline void insert(int a, int b, int c)
{
    face.push_back(Face(a, b, c));
}
void add(int v)
```

```
{
    vector<Face> tmp;
    int a, b, c;
    cnt ++;
    for(int i = 0; i < SIZE(face); i ++){
        a = face[i][0];
        b = face[i][1];
        c = face[i][2];
        if(sgn(volume(v, a, b, c)) < 0){
            mark[a][b] = mark[b][a] = mark[b][c] = mark[c][b] = mark[c]
[a] = mark[a][c] = cnt;
        }
        else tmp.push_back(face[i]);
    }
    face = tmp;
    for(int i = 0; i < SIZE(tmp); i ++){
        a = face[i][0];
        b = face[i][1];
        c = face[i][2];
        if(mark[a][b] == cnt) insert(b, a, v);
        if(mark[b][c] == cnt) insert(c, b, v);
        if(mark[c][a] == cnt) insert(a, c, v);
    }
}

int Find()
{
    for(int i = 2; i < n; i ++){
        point3 ndir = det(info[0] - info[i], info[1] - info[i]);
        if(ndir == point3()) continue;
        swap(info[i], info[2]);
        for(int j = i + 1; j < n; j ++){
            if(sgn(volume(0, 1, 2, j)) != 0){
                swap(info[j], info[3]);
                insert(0, 1, 2);
                insert(0, 2, 1);
                return 1;
            }
        }
    }
    return 0;
}

void D3_convex()
{

    sort(info, info + n);
    n = unique(info, info + n) - info;
    face.clear();
    random_shuffle(info, info + n);
    if(Find()){
        MS(mark, 0);
        cnt = 0;
        for(int i = 3; i < n; i ++) add(i);
    }
```

```
}
point3 fa(point3 a, point3 b, point3 c)
{
    return det((b - a), (c - a));
}
double check(point3 mydi)
{
    point3 ret = point3(0.0, 0.0, 0.0);
    for(int i = 0; i < SIZE(face); i ++){
        point3 cur = fa(info[face[i][0]], info[face[i][1]], info[face[i]
[2]]);  // 枚举每个面， 求出平面的方向向量?
        if((dot(mydi, cur)) >= 0) ret = ret + cur;  //  如果方向向量和投影
面?

    }
    return vlen(ret);
}

int getrand()
{
    int ret = rand() % 200;
    if(rand() % 2)ret = -ret;
    return ret;
}

void solve()
{
    double ans = 0;
    for(int i = 0; i < 10000; i ++){
        point3 mydi;
        mydi.x = getrand();
        mydi.y = getrand();
        mydi.z = getrand();
        ans = max(ans, check(mydi));    // 随机投影面
    }
    printf("%.12f\n", ans / 2);
}
int main()
{
    scanf("%d", &casenum);
    for(casei = 1; casei <= casenum; casei ++){
        scanf("%d", &n);
        for(int i = 0; i < n; i ++){
            scanf("%lf%lf%lf", &info[i].x, &info[i].y, &info[i].z);
        }
        printf("Case #%d: ", casei);
        D3_convex();
        solve();
    }
    return 0;
}
```