1 a hdu 5073 Galaxy 瞎搞数学题
2 b hdu 5531 Rebuild 计算几何
3 c hdu 5950 矩阵快速幂
4 d poj 1830 开关问题 高斯消元
5 e hdu 2824 欧拉函数 基础题
6 f hdu 1071 水题
7 g hdu 5114 扩展欧几里德 / 同余方程

A.Galaxy

题意：

需要去加快一个星系的旋转速度，我们需要减小

$$I = \sum_{i=1}^{n} w_i * d_i^2$$

的值。其中所有的星球的重量都是一个单位，n个星球最多可以移动k个星球。

注意：重心是会随着每一个星球的移动而移动 。

分析：取出k个星球放到剩下的星球的新重心上的I肯定比放在其他地方更优。因为放在新重心上则相当于去掉了这些星球它们到重心的距离平方和等于零。

然后考虑选取哪些星球，也就是去掉哪些星球。经过简单的推到我们就可以得到一个结论：最后剩下的星球一定在原来的序列中连续。提示；方差最小。

那么只要枚举k个n-k长度的区间所对应的I的值求出最小值就好了。但是这里的枚举直接暴力枚举的话是会超时的。然后注意到这个n-k长度的区间在从前往后每移动一格过程中I的变化是可以用公式算出来的而不是再次暴力的枚举该区间的每一个数来算。

设当前区间是[i, n - k + i)当前的平均值（重心所在的位置）为x。当前区间的所有位置的下标和是sum则区间向右移动一格后，sum = sum - d[i] + d[n - k + i]重心向右移动dx = sum / (n - k) - x;则I = I + (n - k) * dx * dx + (d[i + n - k] - x) * (d[i + n - k] - x) - (d[i] - x) * (d[i] - x);

代码：

```
#include <cstdio>
#include <algorithm>

using namespace std;

const double eps = 1e-9;

int main(void)
{
    int t, n, k;
    double num[50010];
    scanf("%d", &t);
    while(t--)
    {
        double sum = 0.0, x, dx, ans = 0.0, minn;
        scanf("%d%d", &n, &k);
        for(int i = 0; i < n; i++)
            scanf("%lf", &num[i]);
        sort(num, num + n);
        for(int i = 0; i < n - k; i++)
            sum += num[i];
        x = sum / (n - k);
        for(int i = 0; i < n - k; i++)
            ans += (num[i] - x) * (num[i] - x);
        minn = ans;
        for(int i = 0; i < k; i++)
        {
            sum = sum + num[i + n - k] - num[i];
            dx = sum / (n - k) - x;
            x += dx;
            ans = ans + (n - k) * dx * dx + (num[i + n
- k] - x) * (num[i + n - k] - x) - (num[i] - x) *
(num[i] - x);
            if(ans < minn + eps)
                minn = ans;
        }
        printf("%.1f\n", minn);
    }
    return 0;
}
```

B.Rebuild

题意：这道题目按顺序给出一些点的坐标，点依次相连，并且最后一个点和第一个点相连形成一个环。问在换上以每个点为圆心作一系列圆，要求相邻两点的圆相互外切。如不存在这样的点则输出“IMPOSSIBLE”否则输出这些圆总面积的最小值，并依次输出此时所有圆的半径（所有输出保留两位小数）

分析：我们可以假设第一个圆的半径为x那么能够一次表示出所有的圆的半径，并且可以把每个圆的半径依次表示为f[i] + x,或者f[i] - x;（第i个点）i为奇数是是+，偶数是-；因为要求所有的半径大于等于0，那么我们在可以一次求出所有的f[i]先，和x的范围[minn, maxn]。最后可以用x表示出最后一个圆的半径。

如果是奇数个点：那么x是一个可以求的定值，只要判断在不在[minn, maxn]里就可以，如果在则有解，那么可以通过f[i]求出所有圆的半径，面积也容易求。

如果是偶数个点：那么最后的半径和第一个圆的半径是同号码的，那么不能直接求出x，我们可以用x来表示每一个圆的半径，那么总面积一定能表示称一个x的二次函数，然后结合x在区间[minn, maxn]内，可以求出最大的总面积，并求出此时的x。然后可以求出每个圆的半径。

代码：

```
//
//  main.cpp
//  Rebuild
//
//  Created by  绿色健康文艺小清新 on 2.25
//  Copyright 2017年  绿色健康文艺小清新. All rights
reserved.
//

#include <cstdio>
#include <cmath>

const double PI = acos(-1);
const double eps = 1e-7;

int n;
double len[10005], f[10005], x;

struct point{
    double x, y;
} p[10005];

double cal_len(point a, point b)
{
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y)
```

```
* (a.y - b.y));
}

void cal_all_len()
{
    for(int i = 0; i < n - 1; i++)
        len[i] = cal_len(p[i], p[i + 1]);
    len[n - 1] = cal_len(p[0], p[n - 1]);
}

double abs(double x)
{
    if(x > 0)
        return x;
    return -x;
}

void print(double ans)
{
    printf("%.2f\n", ans * PI);
    for(int i = 0; i < n; i++)
    {
        if(i & 1)
            printf("%.2f\n", f[i] - x);
        else
            printf("%.2f\n", f[i] + x);
    }
}

int main()
{
    int T;
    scanf("%d", &T);
    while (T--)
    {
        double maxn = 0x3f3f3f, minn = 0;

        scanf("%d", &n);
        for(int i = 0; i < n; i++)
            scanf("%lf%lf", &p[i].x, &p[i].y);

           cal_all_len();

        f[0] = 0;
        for(int i = 1; i < n; i++)
        {
            f[i] = len[i - 1] - f[i - 1];
```

```
            if((i & 1) && f[i] < maxn)
                maxn = f[i];
            if(!(i & 1) && (-f[i]) > minn)
                minn = -f[i];
        }
        if(minn >= maxn + eps)
        {
            printf("IMPOSSIBLE\n");
            continue;
        }

        if(n & 1)
        {
            x = 1.0 * (len[n - 1] - f[n -1]) / 2;
            if(x <= minn - eps || x >= maxn + eps)
            {
                printf("IMPOSSIBLE\n");
                continue;
            }

            double ans = 0.0;
            for(int i = 0; i < n; i++)
            {
                if(i & 1)
                    ans += (f[i] - x) * (f[i] - x);
                else
                    ans += (f[i] + x) * (f[i] + x);
            }

            print(ans);
        }
        else
        {
            if(abs(f[n - 1] - len[n - 1]) > eps ||
(minn - maxn) > eps)
            {
                printf("IMPOSSIBLE\n");
                continue;
            }

            double A = 0, B = 0, C = 0;
            for(int i = 0; i < n; i++)
            {
                A = A + 1;
                C += f[i] * f[i];
                if(i & 1)
                    B -= 2 * f[i];
```

```
                else
                    B += 2 * f[i];
            }

            double l = (-B / 2) / A;

            if(l < minn + eps)
                x = minn;
            else if(maxn < l + eps)
                x = maxn;
            else
                x = l;

            print(A * x * x + B * x + C);
        }
    }
    return 0;
}
```

C.**Recursive sequence**

题意：求一个递推式

分析：直接递归时间复杂度不够，这里用矩阵快速幂

代码：

```
#include <cstdio>
#include <cstring>

const long long int mod = (long long int)2147493647;

struct node
{
    long long int mat[7][7];
};
long long int ans[7], n, a, b;
node m;

node mul(node a, node b)
{
    node ans;
    memset(ans.mat, 0, sizeof(ans.mat));
    for(int i = 0; i < 7; i++)
    for(int j = 0; j < 7; j++)
    for(int k = 0; k < 7; k++)
```

```
            ans.mat[i][j] = (ans.mat[i][j] + a.mat[i][k] *
b.mat[k][j] % mod) % mod;
    return ans;
}

long long int solve()
{
    if(n == 1)
        return a;
    if(n == 2)
        return b;
    node c;
    for(int i = 0; i < 7; i++)
        for(int j = 0; j < 7; j++)
            c.mat[i][j] = (i == j);

    for(int k = n - 2; k; k >>= 1)
    {
        if(k & 1)
            c = mul(c, m);
        m = mul(m, m);
    }
    long long int ANS = 0;
    for(int i = 0; i < 7; i++)
        ANS =  (ANS + c.mat[0][i] * ans[i] % mod) %
mod;
    return ANS;
}

int main(void)
{
    int t;
    scanf("%d", &t);
    while(t--)
    {
        scanf("%lld%lld%lld", &n, &a, &b);
        ans[0] = b % mod;
        ans[1] = a % mod;
        ans[2] = 81;
        ans[3] = 27;
        ans[4] = 9;
        ans[5] = 3;
        ans[6] = 1;
        int mgj[49] =
        {1, 2, 1, 0, 0, 0, 0,
         1, 0, 0, 0, 0, 0, 0,
         0, 0, 1, 4, 6, 4, 1,
```

```
            0, 0, 0, 1, 3, 3, 1,
            0, 0, 0, 0, 1, 2, 1,
            0, 0, 0, 0, 0, 1, 1,
            0, 0, 0, 0, 0, 0, 1};
        for(int i = 0; i < 7; i++)
        for(int j = 0; j < 7; j++)
            m.mat[i][j] = mgj[i * 7 + j];
        printf("%lld\n", solve());
    }
    return 0;
}
```

D.开关问题

某些开关的动作可能影响另一些开关的状态,因此以开关为节点，如果存在这种关系就加入一条有向边（开始我想成对称的了，浪费了很多时间--），这样就构成了一个图，可以用邻接矩阵表示（但是要转置一下，后面细说）。当某个开关按下时，其自身状态改变，受其影响的开关的状态也改变。

用两个N维向量表示初始状态和结束状态，两者逐个元素异或，就得到了开关状态的变化。

以第一个样例输入为例分析，3个开关，两两相连，初始状态000，最终状态111，将对角线的0全部换成1，得矩阵A=

将矩阵每一列想象为一个开关按下后产生的效果（1表示状态翻转，0表示不变），比如，第二列就表示按下第二个开关，则第二个开关的本身状态要改变（这就是把对角线0换成1的原因），受第二个开关影响的开关j状态也要改变，恰好对应邻接矩阵中A[j, 2]=1

把A写成分块矩阵的形式，每一列作为一个子矩阵，则有A=[a1, a2, a3],此处ai均为列向量，设第i个开关按下次数为xi，xi=0或1（开关按两下和没按是等效的，0/1就够了）

记初始状态b0=[0,0,0],最终状态b1=[1,1,1],则状态变化b=b0^b1=[1,1,1],这里b也是列向量。目标就是求x1a1 + x2a2 +x3a3 = b的解的个数（此处的加是模2加，也就是异或，下同）

这个方程可以写成 

下面就是解这个线性方程组

对增广矩阵[A b]做初等行变换，化成阶梯形（高斯消元法），如果存在[0,0,...,0,1]的行，就是无解；如果存在r行[0,0,...,0,0]，就意味着有r个自由变量，因为这里的变量只取0/1，所以有2r个解；如果不存在[0,0,...,0,*]，即把最后一行去掉后不存在全0行，则A为满秩矩阵，则方程组有唯一解。

代码：

```
//
//  main.cpp
//  开关问题
//
//  Created by  绿色健康文艺小清新 on 2.25
//  Copyright 2017年  绿色健康文艺小清新. All rights
reserved.
//
#include <cmath>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <algorithm>

using namespace std;

const int mod = 2;

int a[30][31];

int gcd(int a, int b){
    int t;
    while(b){
        t = b;
        b = a % b;
        a = t;
    }
    return a;
}

int lcm(int a, int b){
    return a / gcd(a, b) * b;
}

int Gauss(int equ, int var){
    int k, col;
    for (k = 0, col = 0; k < equ && col < var; ++k,
++col){
        int max_r = k;
        for (int i = k + 1; i < equ; ++i){
            if (abs(a[i][col]) > abs(a[max_r][col]))
max_r = i;
        }
        if (max_r != k){
```

```
            for (int i = k; i <= var; ++i)
                swap(a[k][i], a[max_r][i]);
        }
        if (a[k][col] == 0){
            --k;
            continue;
        }
        for (int i = k + 1; i < equ; ++i){
            if (a[i][col] != 0) {
                int LCM = lcm(abs(a[i][col]), abs(a[k]
[col]));
                int ta = LCM / abs(a[i][col]);
                int tb = LCM / abs(a[k][col]);
                if (a[i][col] * a[k][col] < 0) tb = -
tb;
                for (int j = col; j <= var; ++j) {
                    a[i][j] = ((a[i][j] * ta - a[k][j]
* tb) % mod + mod) % mod;
                }
            }
        }
    }
    for (int i = k; i < equ; ++i){
        if (a[i][col] != 0) return -1;
    }
    return var - k;
}

int s[30], e[30];
long long int b[30];

void build(int n)
{
    int x, y;
    while(scanf("%d%d", &x, &y), x || y)
        a[y - 1][x - 1] = 1;
    for(int i = 0; i < n; i++)
        a[i][n] = s[i] ^ e[i];
}

void init()
{
    b[0] = 1;
    for(int i = 1; i < 30; i++)
        b[i] = b[i - 1] * 2;
}
```

```
int main(void)
{
    int t, n;
    scanf("%d", &t);
    init();
    while(t--)
    {
        scanf("%d", &n);
        for(int i = 0; i < n; i++)
        for(int j = 0; j <= n; j++)
            a[i][j] = 0;
        for(int i = 0; i < n; i++)
        {
            scanf("%d", &s[i]);
            a[i][i] = 1;
        }
        for(int i = 0; i < n; i++)
            scanf("%d", &e[i]);
        build(n);
        int ans = Gauss(n, n);
        if(ans == -1)
            printf("Oh,it's impossible~!!\n");
        else
            printf("%lld\n", b[ans]);
    }
    return 0;
}
```

E.**The Euler function**

题意：求phi(a) + phi(a+1) +...+phi(b);

分析：紫书原题 p322

代码：

```cpp
#include <cstdio>
#include <cstring>

const int MAXN = 3 * 1e6 + 10;

long long int ans[MAXN];

void init(int n)
{
    memset(ans, 0, sizeof(ans));
    ans[1] = 1;
    for(int i = 2; i <= n; i++)
    {
        if(!ans[i])
            for(int j = i; j <= n; j += i)
            {
                if(!ans[j])
                    ans[j] = j;
                ans[j] = ans[j] / i * (i - 1);
            }
            ans[i] += ans[i - 1];
    }
}

int main(void)
{
    int a, b;
    init(MAXN - 5);
    while(scanf("%d%d", &a, &b) != EOF)
        printf("%lld\n", ans[b] - ans[a - 1]);
    return 0;
}
```

F.**The area**

签到题。求一个积分。

代码:

```
#include <stdio.h>
double a, b, c, d, k;
double fun(double x)
{
    return a * x * x * x / 3 - (2 * a * b + k) * x * x
/ 2 + (a * b * b + c - d) * x;
}
int main()
{
    int t;
    double x1, y1, x2, y2, x3, y3;
    scanf("%d", &t);
    while (t--)
    {
        scanf("%lf%lf%lf%lf%lf%lf", &x1, &y1, &x2, &y2,
&x3, &y3);
        b = x1;
        c = y1;
        a = (y2 - y1) / (x2 - x1) / (x2 - x1);
        k = (y3 - y2) / (x3 - x2);
        d = y2 - k * x2;
        printf("%.2lf\n", fun(x3) - fun(x2));
    }
    return 0;
}
```

G.**Collision**

题意：在一个长x宽y的矩形内部有两个小球，以相同的速度运动，碰到矩形的边，小球无能量损失反弹，问两小球是否会相撞，若会相撞，求撞击的那一点。

分析：分位三种情况，所有数据乘2就能保证在运算过程中只有整数。

代码：

```
//
//  main.cpp
//  Collision
//
//  Created by  绿色健康文艺小清新 on 11.21
//  Copyright 2016年  绿色健康文艺小清新. All rights
reserved.
//

#include <cstdio>
```

```
typedef long long int LL;

struct point{
    LL x, y;
    point(){}
    point(LL _x, LL _y):x(_x),y(_y){}
}a, b, s;

point format(point g)
{
    g.x %= (2 * s.x);
    g.y %= (2 * s.y);
    if(0 <= g.x && g.x <= s.x && 0 <= g.y && g.y <=
s.y)
        return g;
    if(0 <= g.x && g.x <= s.x && s.y <= g.y && g.y <= 2
* s.y)
        return point(g.x, 2 * s.y - g.y);
    if(s.x <= g.x && g.x <= 2 * s.x && 0 <= g.y && g.y
<= s.y)
        return point(2 * s.x - g.x, g.y);
    //if(s.x <= g.x && g.x <= 2 * s.x && s.y <= g.y &&
g.y <= 2 * s.y)
        return point(2 * s.x - g.x, 2 * s.y - g.y);
    //return g;
}

LL e_gcd(LL a, LL b, LL &x, LL &y){
    if(!b){
        x=1;
        y=0;
        return a;
    }
    LL ans,tep;
    ans = e_gcd(b, a % b, x, y);
    tep = x;
    x = y;
    y = tep - a / b * y;
    return ans;
}

LL gcd(LL a, LL b)
{
    if(!b)
        return a;
    return gcd(b, a % b);
```

```
}


LL inv(LL a, LL n){
    LL x, y;
    LL d = e_gcd(a, n, x, y);
    if(d == 1) return (x % n + n) % n;
    else return -1;
}

//将两个方程合并为一个
bool merge(LL a1, LL n1, LL a2, LL n2, LL& a3, LL& n3)
{
    LL d = gcd(n1,n2);
    LL c = a2-a1;
    if(c % d)
        return false;
    c = (c % n2 + n2) % n2;
    c /= d;
    n1 /= d;
    n2 /= d;
    c *= inv(n1, n2);
    c %= n2;
    c *= n1 * d;
    c += a1;
    n3 = n1 * n2 * d;
    a3 = (c % n3 + n3) % n3;
    return true;
}

//求模线性方程组x=ai(mod ni),ni可以不互质
LL China_Reminder2(LL len, LL* a,LL* n)
{
    LL a1 = a[0], n1 = n[0];
    LL a2, n2;
    for(int i = 1; i < len; i++)
    {
        LL aa,nn;
        a2 = a[i], n2 = n[i];
        if(!merge(a1,n1,a2,n2,aa,nn))
            return -1;
        a1 = aa;
        n1 = nn;
    }
    return (a1 % n1 + n1) % n1;
}
```

```
void solve()
{
    LL t;
    point ans;
    if(a.x == b.x)
    {
        t = s.y - (a.y + b.y) / 2;
        ans = format(point(a.x + t, a.y + t));
        printf("%.1f %.1f\n", ans.x / 2.0, ans.y /
2.0);
        return;
    }
    if(a.y == b.y)
    {
        t = s.x - (a.x + b.x) / 2;
        ans = format(point(a.x + t, a.y + t));
        printf("%.1f %.1f\n", ans.x / 2.0, ans.y /
2.0);
        return;
    }
    else
    {
        point m = point((a.x + b.x) / 2, (a.y + b.y) /
2);
        long long int h[5], r[5];
        h[0] = -m.x, h[1] = -m.y;
        r[0] = s.x, r[1] = s.y;
        t = China_Reminder2(2, h, r);
        if(t == -1)
            printf("Collision will not happen.\n");
        else
        {
            ans = format(point(a.x + t, a.y + t));
            printf("%.1f %.1f\n", ans.x / 2.0, ans.y /
2.0);
        }
        return;
    }
    return;
}

int main(void)
{
    int t;
    LL  x, y;
    scanf("%d", &t);
```

```
    for(int cas = 1; cas <= t; cas++)
    {
        scanf("%lld%lld", &x, &y);
        s = point(2 * x, 2 * y);
        scanf("%lld%lld", &x, &y);
        a = point(2 * x, 2 * y);
        scanf("%lld%lld", &x, &y);
        b = point(2 * x, 2 * y);
        printf("Case #%d:\n", cas);
        solve();
    }
    return 0;
}
```