遮天蔽日(blot,1s,512MB)

【算法分析】

前置知识：

1.求任意 n 边形的面积：

设 n 边形的顶点按逆时针顺序分别为 $P_0,P_1,\ldots,P_{n-1}$，那么面积为 $|\frac{1}{2}\sum_{i=0}^{n-1}P_i\times P_{i+1}|$。

2.求任意 n 边形的重心：

考虑按照面积公式将 n 边形剖分成 n 个有向三角形，记第 i 个三角形的重心为 (x_i,y_i)，有向面积为 s_i，n 边形的重心为 G，面积为 S。那么 $X_G=\frac{\sum_{i=1}^{n}s_i\times x_i}{S},Y_G=\frac{\sum_{i=1}^{n}s_i\times y_i}{S}$。

3.(x,y) 绕原点逆时针转 θ 度（弧度），得到的坐标为 $(x\cos\theta-y\cos\theta,x\sin\theta+y\cos\theta)$。

4.判断线段 AB,CD 是否相交：

如果满足以下条件之一则 AB,CD 不相交：

(1).$max(X_A,X_B)<min(X_C,X_D)$

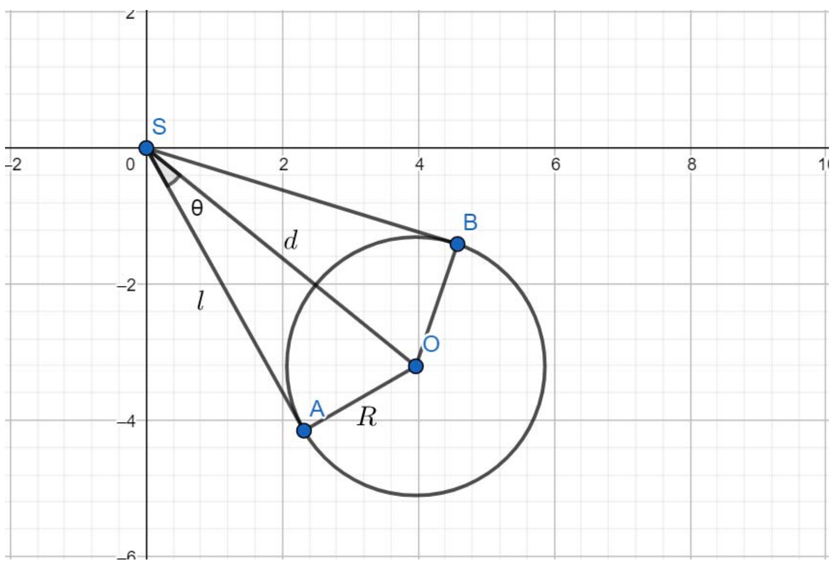
(2).$max(X_C,X_D)<min(X_A,X_B)$

(3).$max(Y_A,Y_B)<min(Y_C,Y_D)$

(4).$max(Y_C,Y_D)<min(Y_A,Y_B)$

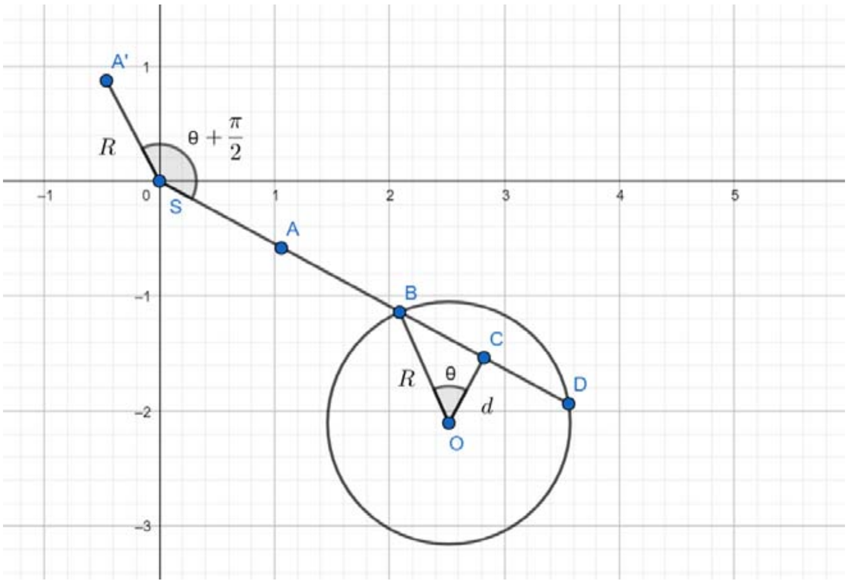
否则，判断 A,B 是否在 CD 两侧，以及 C,D 是否在 AB 两侧，即如果 $(\overrightarrow{CD}\times\overrightarrow{CA})\times(\overrightarrow{CD}\times\overrightarrow{CB})\le 0$ 且 $(\overrightarrow{AB}\times\overrightarrow{AC})\times(\overrightarrow{AB}\times\overrightarrow{AD})\le 0$ 则相交。

5.如图，S 为原点，圆 O 的半径为 R， 求过 S 的切线和圆 O 的交点 A,B：



令 $d=OS$，则 $\theta=arcsin(\frac{R}{d})$，$AS=BS=d\times sin\,\theta$。那么把 O 绕 S 分别往顺时针和逆时针旋转 θ 度，再把横纵坐标分别乘上 $cos\,\theta$，就得到了点 A,B。

6.如图，已知线段 AS 和圆 O 无交点，求直线 AS 和圆 O 的交点 B：



先求出点 O 到直线 AS 的距离 $d=|\frac{\overrightarrow{SO}\times\overrightarrow{SA}}{|\overrightarrow{SA}|}|$。

当 $\overrightarrow{SO}\times\overrightarrow{SA}\geq 0$ 时（$\overrightarrow{SO}\times\overrightarrow{SA}<0$ 同理）：

$\theta=arccos(\frac{d}{R})$，将点 A 逆时针旋转 $\frac{\pi}{2}+\theta$ 度得到点 A'，再把 A' 的横纵坐标分别乘上 $\frac{R}{|\overrightarrow{SA'}|}$。此时易得 $SA'\perp OB,SA'=OB$，这样就能得到 B 的坐标了。

回到本题。为了方便，我们令太阳为原点，先按照题意求出 n 边形（干扰器）的重心并旋转。

接着，求出过原点的切线与圆（地球）的两个交点 B_1,B_2。对于 n 边形上的第 i 个点 P_i，若直线 OP_i 与圆有交点且线段 OP_i 与圆无交点，那么求出 C_i 为 OP_i 与圆的交点。

把 B_1,B_2 以及所有 C_i 按极角排序。显然极角相邻的两条直线间的弧，要么都被照到，要么都不被照到。弧 MN 被照到的条件：不存在多边形的任何一条边 EF 使得 OM 和 EF 相交且 ON 和 EF 相交。把能照到的弧长全部相加就是答案。

时间复杂度 $O(n^2)$。

【参考程序】

```
// 陈予菲
#include <bits/stdc++.h>

using namespace std;

#define ld long double
```

```cpp
template <class t>
inline void read(t & res)
{
    char ch;
    bool fl = 0; res = 0;
    while (ch = getchar(), ch != '-' && !isdigit(ch));
    if (ch == '-')
        fl = 1;
    else
        res = ch ^ 48;
    while (ch = getchar(), isdigit(ch))
        res = res * 10 + (ch ^ 48);
    if (fl)
        res = ~res + 1;
}

const int N = 105;
const ld pi = 3.141592653589, eps = 1e-11;

int n, m;
ld a1, a2, ans, t1, t2, T;

struct point
{
    ld x, y;
    point(){}
    point(ld _x, ld _y) :
        x(_x), y(_y) {}

    inline void scanf()
    {
        int _x, _y;
        read(_x); read(_y);
        x = _x; y = _y;
    }

    inline point rotate(ld a) // 求绕(0,0)逆时针旋转(弧度为 a)后的位置
    {
        ld cosA = cos(a), sinA = sin(a);
        return point(x * cosA - y * sinA, x * sinA + y * cosA);
    }

    inline ld len() // 求到原点的距离
    {
        return sqrt(x * x + y * y);
    }
}S, G, b[N];

inline point operator + (point a, point b)
{
    return point(a.x + b.x, a.y + b.y);
}
```

```
inline point operator - (point a, point b)
{
    return point(a.x - b.x, a.y - b.y);
}

inline point operator * (point a, ld v)
{
    return point(a.x * v, a.y * v);
}

inline point operator / (point a, ld v)
{
    return point(a.x / v, a.y / v);
}

inline ld mul(point a, point b) // 求 ab 的点积
{
    return a.x * b.x + a.y * b.y;
}

inline ld operator * (point a, point b) // 求 ab 的叉积
{
    return a.x * b.y - b.x * a.y;
}

inline bool is_cross(point a, point b, point c, point d) // 判断线段 ab 和 cd
是否相交
{
    if (min(a.x, b.x) - max(c.x, d.x) > eps || min(c.x, d.x) - max(a.x, b.x) >
eps)
        return 0;
    if (min(a.y, b.y) - max(c.y, d.y) > eps || min(c.y, d.y) - max(a.y, b.y) >
eps)
        return 0;
    if (((d - a) * (b - a)) * ((c - a) * (b - a)) > eps)
        return 0;
    if (((d - c) * (a - c)) * ((d - c) * (b - c)) > eps)
        return 0;
    return 1;
}

inline int nxt(int x)
{
    return x == n ? 1 : x + 1;
}

struct polygon
{
    point p[N];

    inline ld area() // 求多边形的面积
```

```
    {
        ld res = 0;
        for (int i = 1; i <= n; i++)
            res += p[i] * p[nxt(i)];
        return res / 2.0;
    }

    inline void init()
    {
        for (int i = 1; i <= n; i++)
            p[i].scanf();
        if (area() < 0)
            reverse(p + 1, p + n + 1);
    }

    inline void getG() // 求多边形的重心
    {
        ld sum = area(), gx = 0, gy = 0; // gx,gy 分别为 x,y 的带权平均数
        for (int i = 1; i <= n; i++)
        {
            point a = p[i], b = p[nxt(i)];
            ld s = a * b / 2.0;
            gx += s * (a.x + b.x) / 3.0;
            gy += s * (a.y + b.y) / 3.0;
        }
        gx /= sum; gy /= sum;
        G = point(gx, gy);
    }

    inline bool check(point a, point b) // 判断圆弧 ab 是否被照射到
    {
        point o = point(0, 0);
        for (int i = 1; i <= n; i++)
        {
            point c = p[i], d = p[nxt(i)];
            if (is_cross(o, a, c, d) && is_cross(o, b, c, d))
                return 0;
            // 判断 ab 是否被线段 cd 挡住
        }
        return 1;
    }
}A;

struct circle
{
    point O;
    ld R;

    inline void scanf()
    {
        int x, y, r;
        read(x); read(y); read(r);
```

```
        O = point(x, y);
        R = r;
    }

    inline void getcut() // 将太阳放在原点，求过原点的切线和圆的交点
    {
        ld d = O.len(), a = asin(R / d);
        b[1] = O.rotate(+a) * cos(a);
        b[2] = O.rotate(-a) * cos(a);
        m = 2;
    }

    inline point getcross(point A) // 求直线 OA 和圆的交点
    {
        ld d = fabs(O * A) / A.len(), a = acos(d / R);
        if (O * A >= 0)
            return (A.rotate(+pi / 2.0)).rotate(+a) * R / A.len() + O;
        else
            return (A.rotate(-pi / 2.0)).rotate(-a) * R / A.len() + O;
    }

    inline ld calc(point b, point c) // 求圆弧 bc 的长度
    {
        point ub = b - O, uc = c - O;
        ld a = mul(ub, uc) / ub.len() / uc.len();
        return fabs(acos(a) * R);
    }
}C;

inline bool cmp(const point &a, const point &b) // 极角排序
{
    return a * b > 0;
}

int main()
{
    freopen("blot.in", "r", stdin);
    freopen("blot.out", "w", stdout);
    S.scanf(); C.scanf();
    int _t1, _t2, _T;
    read(n); read(_t1); read(_t2); read(_T);
    t1 = _t1; t2 = _t2; T = _T;
    A.init(); A.getG();

    int i;
    for (i = 1; i <= n; i++) // 把太阳放在原点
        A.p[i] = A.p[i] - S;
    C.O = C.O - S;
    G = G - S;
    a1 = T / t1 * 2.0 * pi;
    a2 = T / t2 * 2.0 * pi;
```

```
    for (i = 1; i <= n; i++) // 按题意旋转
        A.p[i] = A.p[i] - G, A.p[i] = A.p[i].rotate(a2);
    G = G - C.O; G = G.rotate(a1); G = G + C.O;
    for (i = 1; i <= n; i++)
        A.p[i] = A.p[i] + G;

    C.getcut();
    for (i = 1; i <= n; i++)
        if  ((A.p[i]  *  b[1])  *  (A.p[i]  *  b[2])  <=  eps  &&  A.p[i].len()  -
b[1].len() <= eps)
            b[++m] = C.getcross(A.p[i]);
        // 如果直线 OA.p[i]和圆有交点并且线段 OA.p[i]和圆无交点
    sort(b + 1, b + m + 1, cmp);

    for (i = 1; i < m; i++)
        if (A.check(b[i], b[i + 1]))
            ans += C.calc(b[i], b[i + 1]);

    double fans = ans;
    printf("%.2lf\n", fans);
    fclose(stdin);
    fclose(stdout);
    return 0;
}
```

通技进阶（gentech，1s，512MB）

【算法分析】

三维空间内的直线比较难处理，我们可以将三维空间的**直线**转化为二维平面内的**动点**。具体地，考虑到没有直线垂直于 x 轴，我们用动平面 $x=t$ 来截取直线，不难发现，无论平面怎么移动，每个直线和平面都有且仅有一个交点。如果将动平面的位置看成时间，那么每个交点都是一个动点，并且这个动点的移动速度是不变的。容易算出每个点在 $0s$ 时所在的坐标和每秒的移动速度。

那么**直线相交的位置**就相当于**平面上的动点相交的位置**。判断两个动点是否相交就比较容易了。

若一个集合中的所有动点两两相交，那么一定有两种情况之一：

1．在某一时刻全部交于一点；

2．对应的三维直线共面，且不存在两个直线平行。

第一种情况我们枚举其中一个点，然后将其他能与它相交的点按照相交时间排序，计算出对应时间最多能有几个点交于一点即可。

第二种情况的处理比较巧妙，我们考虑对应的这些三维直线所在的同一个平面，这个平面和动平面 $x=t$ 的公共部分一定是一条直线，并且随着动平面 $x=t$ 的移动，两个平面相交部分的直线也随之**平移**。那么对应到平面上，就是这些动点必须时时刻刻在一条直线上，而且这条直线的斜率不变。

于是我们仍然枚举其中一个点作为原点，并且将其他能和这个点相交，按照**相对于原点的速度**极角排序。注意，如果两个动点能相交，那么它们的连线斜率是不变的（可以考虑对应的两条三维直线所在的公共平面）。因此相对于原点的速度的极角序相同的点一定是共线的。

注意到可能会有相对于原点的速度方向**相反**的情况，但是我们注意到，如果取最左端或者最右端的点作为原点，又限制了都与原点相交，因此一个合法的点集一定存在一个点作为原点，其他点的速度方向相同，答案会统计完整。因此直接按照极角序排是没有问题的。

这里要注意排除两个动点始终平行的情况。我们只要在极角序相同的时候另外按照 x 坐标为第一关键字、y 坐标为第二关键字排序即可，运动速度相同的动点始终是平行的。

时间复杂度为 $O(n^2 \log n)$。具体实现可以看参考程序。

【参考程序】

```
//陈煜翔
#include <cmath>
#include <cstdio>
#include <cctype>
#include <iostream>
#include <algorithm>

template <class T>
inline void read(T &x)
{
    static char ch;
    static bool opt;
    while (!isdigit(ch = getchar()) && ch != '-');
    x = (opt = ch == '-') ? 0 : ch - '0';
    while (isdigit(ch = getchar()))
        x = x * 10 + ch - '0';
    if (opt)
        x = ~x + 1;
}

template <class T>
inline void putint(T x)
{
    static char buf[45], *tail = buf;
    if (!x)
        putchar('0');
    else
    {
```

```
        if (x < 0)
            putchar('-'), x = ~x + 1;
        for (; x; x /= 10) *++tail = x % 10 + '0';
        for (; tail != buf; --tail) putchar(*tail);
    }
}

template <class T>
inline bool tense(T &x, const T &y)
{
    return y < x ? x = y, true : false;
}

template <class T>
inline bool relax(T &x, const T &y)
{
    return x < y ? x = y, true : false;
}

typedef long long s64;
typedef long double ld;
typedef std::pair<int, int> pii;
#define mp(x, y) std::make_pair(x, y)

const ld eps = 1e-10;
const int MaxN = 1e3 + 5;

struct point
{
    ld x, y;
    point(){}
    point(ld _x, ld _y):
        x(_x), y(_y) {}
    inline point operator + (point rhs) const
    {
```

```
        return point(x + rhs.x, y + rhs.y);
    }
    inline point operator - (point rhs) const
    {
        return point(x - rhs.x, y - rhs.y);
    }
    inline point operator * (ld rhs) const
    {
        return point(x * rhs, y * rhs);
    }
    inline ld operator * (point rhs) const
    {
        return x * rhs.y - y * rhs.x;
    }

    inline bool operator == (point rhs) const
    {
        return fabs(x - rhs.x) <= eps && fabs(y - rhs.y) <= eps;
    }
    inline bool operator < (point rhs) const
    {
        ld d = atan2(y, x) - atan2(rhs.y, rhs.x);
        if (fabs(d) > eps)
            return d < 0; //按照极角排序
        else
        {
            if (fabs(x - rhs.x) > eps)
                return x < rhs.x; //相同的时候按照坐标排序
            else
                return y < rhs.y;
        }
    }
}p[MaxN], v[MaxN];

int n;
```

```
int ans;

inline ld calc(int a, int b) //求出两个动点其中一维相交的时间
{
    point dp = p[a] - p[b];
    point dv = v[a] - v[b];
    if (fabs(dv.x) > eps)
        return -dp.x / dv.x;
    else if (fabs(dv.y) > eps)
        return -dp.y / dv.y;
    else
        return 0;
}

int main()
{
    freopen("gentech.in", "r", stdin);
    freopen("gentech.out", "w", stdout);

    read(n);
    for (int i = 1; i <= n; ++i)
    {
        int t1, ax, ay, t2, bx, by;
        read(t1), read(ax), read(ay), read(t2), read(bx), read(by);
        v[i] = point((ld)(bx - ax)/(t2 - t1),(ld)(by - ay)/(t2 - t1));
        p[i] = point(ax, ay) - v[i] * t1; //计算 t=0 的位置和速度
    }

    for (int st = 1; st <= n; ++st)
    {
        int tot = 0;

        static ld tim[MaxN];
        static point cur[MaxN];
        for (int i = 1; i <= n; ++i)
```

```
        if (i != st)
        {
            ld t = calc(st, i);
            if (p[st] + v[st] * t == p[i] + v[i] * t)
            {
                ++tot; //把能相交的点取出来
                tim[tot] = t;
                cur[tot] = v[i] - v[st];
            }
        }

    if (!tot)
        continue;

    std::sort(tim + 1, tim + tot + 1);
    for (int l = 1, r = 0; l <= tot; l = r + 1)
    {
        while (r + 1 <= tot && fabs(tim[r + 1] - tim[l]) <= eps)
            ++r;
        relax(ans, r - l + 1); //判断同一时刻交于一点
    }

    std::sort(cur + 1, cur + tot + 1);
    for (int l = 1, r = 0; l <= tot; l = r + 1)
    {
        while (r + 1 <= tot && fabs(cur[r + 1] * cur[l]) <= eps)
            ++r;

        int cnt = 0;
        for (int i = l + 1; i <= r; ++i)
            cnt += cur[i] == cur[i - 1];
        relax(ans, r - l + 1 - cnt); //判断共线的情况
    }
}
```

```
    std::cout << ans + 1 << '\n';

    return 0;
}
```

鼹鼠(mole,1s,512MB)

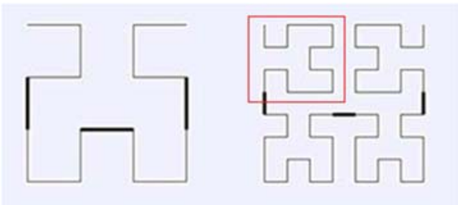
【算法分析】

注意到 n 阶 $Hilbert$ 曲线所在的正方形网格的边长为 2^n-1，我们可以直接用一个二维 $bool$ 数组存储。具体地，我们只需要知道哪些方格是土壤，哪些方格是空气。

观察 $Hilbert$ 曲线的性质，我们可以得到一种较为简便的构造 $bool$ 数组的方法：

首先将 $n-1$ 阶 $Hilbert$ 曲线的 $bool$ 数组复制四份，下面两份显然不用再进行改变，中间增加的四条过道除了最下面的那一条以外其它都是空气。

考虑上面的两份，以左边那一份为例，右边那一份本质相同。显然所有方格都至少与正方形网格的某一边相通。如下图所示，原先只有向上的那一边面向空气，旋转后向上的那一边变为面向土壤，而其它边都恰好面向空气。也就是说，现在左边这一份的空气和土壤的分布情况恰好和原来的相反，旋转后直接取反即可。

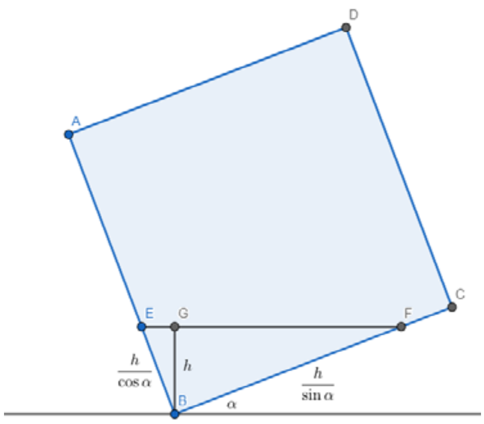


把空气方格看作点，相邻的空气方格连边，原图构成了一个森林结构。

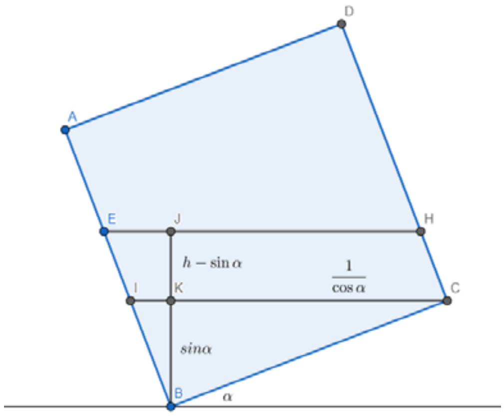
注意到如果我们已知一个方格的水位 h，我们可以计算出这个方格的水量 S 。

具体地，我们以 $\alpha \le 45$ 为例，$\alpha > 45$ 的情况可以改为取 α 的余角计算，本质相同。

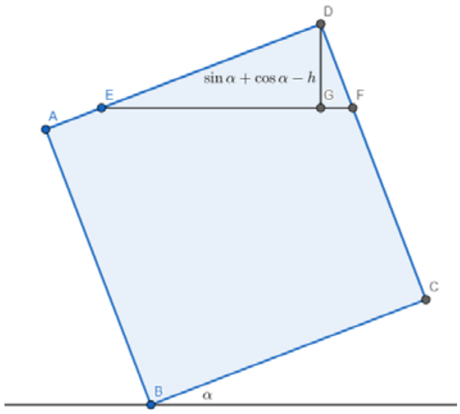
1. 若 $0 \le h \le \sin\alpha$，如下图所示，$S=\frac{h^2}{2\sin\alpha\cos\alpha}$；



2. 若 $\sin\alpha < h < \cos\alpha$，如下图所示，$S=\frac{\sin^2\alpha}{2\sin\alpha\cos\alpha}+\frac{h-\sin\alpha}{\cos\alpha}=\frac{2\sin\alpha h-\sin^2\alpha}{2\sin\alpha\cos\alpha}$;

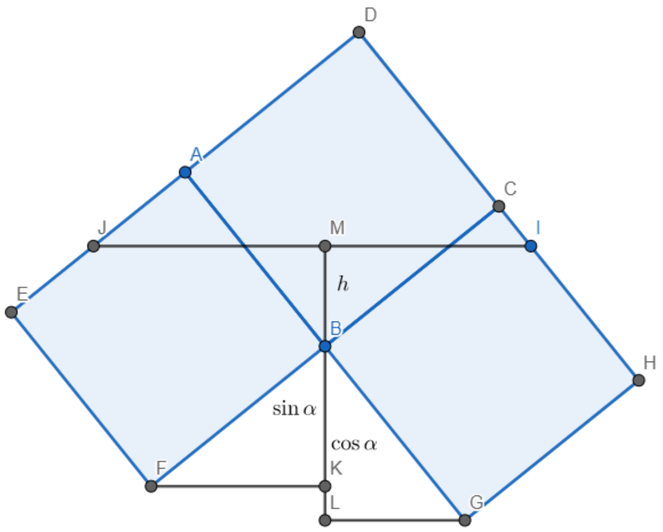


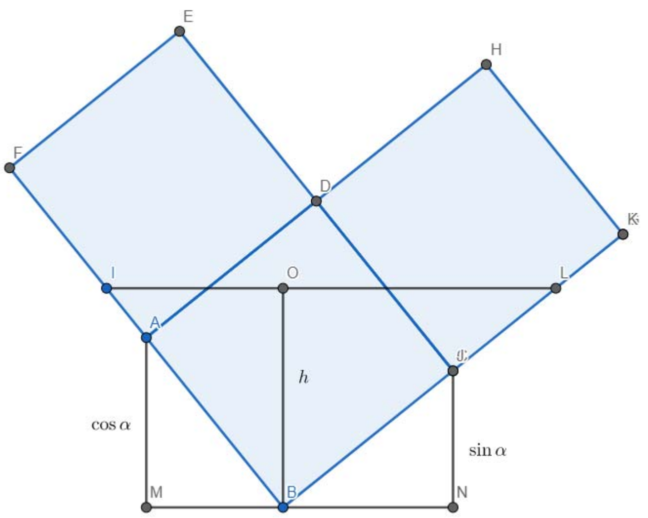
3. 若 $\cos\alpha \le h \le \sin\alpha+\cos\alpha$，如下图所示，$S=1-\frac{(\sin\alpha+\cos\alpha-h)^2}{2\sin\alpha\cos\alpha}=\frac{2\sin\alpha\, h+2\cos\alpha\, h-h^2-1}{2\sin\alpha\cos\alpha}$。



考虑 DFS 整片森林，由当前方格的水位可以推出周围未遍历的方格的水位：

具体地，若当前的方格水位为 $h(h>0, h=0$ 直接停止扩展)，上方的方格水位要下降 $\cos\alpha$，下方的方格水位要上升 $\cos\alpha$，左边的方格水位要上升 $\sin\alpha$，右边的方格水位要下降 $\sin\alpha$。若扩展后的水位小于 0 则按 0 计算，大于$\sin\alpha+\cos\alpha$则按$\sin\alpha+\cos\alpha$计算，如下图所示。





实现时可以把 $\frac{1}{2\sin\alpha\cos\alpha}$ 的部分提到最后计算，精度会高一些，时间复杂度 $O(4^n)$。

注意 $\alpha = 0$ 时需要特判，此时每个方格的水位只有 0 和 1 两种，并且限制条件就是不能遍历向上的方格，很容易处理。

【参考程序】

```
//陈贤
#include <bits/stdc++.h>

const double pi = acos(-1);
const double eps = 1e-8;
const int N = 4100;
bool vis[N][N], stp[N][N];
long double alpha, tS, tC, S, C, H, ans;
```

```
int n, m;

template <class T>
inline T Max(T x, T y) {return x > y ? x : y;}
template <class T>
inline T Min(T x, T y) {return x < y ? x : y;}

inline double askS(double h)
{ // 给定一个方格的水位，求出水量
    if (h <= tS + eps)
        return h * h;
    else if (h <= tC + eps)
        return tS * (2.0 * h - tS);
    else
        return (2.0 * H - h) * h - 1;
}

inline void dfs1(int x, int y, long double h)
{ // DFS 扩展出每个方格的水位
    if (!vis[x][y])
        return ;
    h = Max(Min(h, H), (long double)0.0);
    if (h <= eps)
        return ;
    ans += askS(h);
    vis[x][y] = false;

    dfs1(x, y - 1, h + S);
    dfs1(x + 1, y, h + C);
    dfs1(x, y + 1, h - S);
    dfs1(x - 1, y, h - C);
}

inline void dfs2(int x, int y)
{
    if (!vis[x][y])
        return ;
    ans += 1.0;
    vis[x][y] = false;

    dfs2(x, y - 1);
    dfs2(x + 1, y);
    dfs2(x, y + 1);
}

int main()
{
    freopen("mole.in", "r", stdin);
    freopen("mole.out", "w", stdout);

    scanf("%d%Lf", &n, &alpha);
    alpha = alpha / 180.0 * pi;
```

```
    m = 1;
    vis[1][1] = true;
    for (int i = 2, half; i <= n; ++i)
    { //构造 n 阶 Hilbert 曲线
        half = m + 1;
        m = m << 1 | 1;
        for (int j = 1; j <= m; ++j)
            vis[half][j] = true;
        for (int j = 1; j <= half; ++j)
            vis[j][half] = true;
        for (int j = 1; j < half; ++j)
            for (int k = 1; k < half; ++k)
                vis[j + half][k] = vis[j + half][k + half] = stp[j][k] =
vis[j][k];
        for (int j = 1; j < half; ++j)
            for (int k = 1; k < half; ++k)
            {
                vis[k][half - j + half] = !stp[j][k];
                vis[half - k][j] = !stp[j][k];
            }
    }

    if (alpha <= eps) // 特判 alpha = 0
    {
        for (int i = 1; i <= m; ++i)
            if (vis[1][i] && !stp[1][i])
                dfs2(1, i);
        printf("%.6Lf\n", ans);
    }
    else
    {
        S = sin(alpha);
        C = cos(alpha);
        H = S + C;
        if (alpha > pi / 4.0)
            alpha = pi / 2.0 - alpha;
        tS = sin(alpha);
        tC = cos(alpha);
        for (int i = 1; i <= m; ++i)
            if (vis[1][i] && !stp[1][i])
                dfs1(1, i, C);
        printf("%.6Lf\n", ans / (2.0 * tS * tC));
    }

    fclose(stdin); fclose(stdout);
    return 0;
}
```