Capybara dreaming

Contents

1 First things first

1.1 Includes

```
#include <bits/stdc++.h>
using namespace std;

#define ll long long
#define pb push_back
#define D(x) cout << #x " = " << (x) << endl

typedef vector<int> vi;
typedef vector<vi> vvi;

typedef pair<int, int> ii;
typedef vector<ii> vii;
```

2 Matemática

2.1 Transformada rápida de Fourrier

```
// Resolve:
// - De quantas maneiras conseguimos atingir Y com X tentativas
// - Dado X tentativas, conseguimos atingir Y?
// Complexidade:
// X * Ymax * Ymax(log Ymax)

// TEOREMA DA CONVOLUcÃO:
// Podemos fazer a convolucão de 2 polinomios utilizando a FFT
// Reduzindo a complexidade de n^2 para n log n
// Definimos a convolucão como h[i] = sum(a[j] * b[j-i]) para todo j de 0 a i.
// Exemplo: h[5] = a[5] * b[0] + a[4] * b[1] + a[3] * b[2]...
// Segundo o teorema da convolucão
// h(f . g) = transformada inversa de (transformada (f) * transformada (g))
// onde . é o operador de convolucão.
// e * é o operador de multiplicacão termo a termo.

#include <bits/stdc++.h>

using namespace std;

// primeira potência de 2 maior que o limite de H
#define MAX_DIST (262144 * 2)

typedef complex<double> cpx;
const double pi = acos(-1.0);

int p[MAX_DIST];
int maxDist;

// in:     vector de entrada
// out:    vector de saida
// n:      Tamanho do input/output {DEVE SER DA ORDEM DE 2}
// type:   1 = Transformada, -1 = Transformada inversa
void FFT(vector<cpx> &v, vector<cpx> &ans, int n, int type)
{
        assert(!(n & (n - 1)));
        int i, sz, o;
        p[0] = 0;
        for (i = 1; i < n; i++)
                p[i] = (p[i >> 1] >> 1) | ((i & 1) ? (n >> 1) : 0);
        for (i = 0; i < n; i++)
                ans[i] = v[p[i]];
        for (sz = 1; sz < n; sz <<= 1)
        {
                const cpx wn(cos(type * pi / sz), sin(type * pi / sz));
                for (o = 0; o < n; o += (sz << 1))
                {
                        cpx w = 1;
                        for (i = 0; i < sz; i++)
                        {
                                const cpx u = ans[o + i], t = w * ans[o + sz + i];
                                ans[o + i] = u + t;
                                ans[o + i + sz] = u - t;
                                w *= wn;
                        }
                }
        }

        if (type == -1)
                for (i = 0; i < n; i++)
                        ans[i] /= n;
}

// Exemplo:
// Há um robo que pode disparar bolas em N distâncias diferentes.
// Queremos saber se ele alcanca uma distância M com 1 ou 2 tacadas.
// Resolucão:
// Podemos definir um vetor distances[MAX_DIST],
// onde a distances[i] = 1 se ele pode tacar até a distancia i
```

```
// e distances[i] = 0 caso contrario
// Para ver se o robo acerta com 1 tacada, é trivial.
// Para ver se o robo acerta com 2 tacadas, podemos fazer a convolucão de distances com distances.
// Ex: Acertar a Pode[10] é igual a: Pode[10] || Pode[9] * Pode[1] || Pode[8] * Pode[2]...
// Ou seja, H = FFTi(FFT(distances) ** 2);

// Complexidade:
// 2 * 200k * log(200k) = 8m

int main()
{
        int N, d;
        vector<cpx> distances, fftOut;

        while (cin >> N)
        {
                maxDist = 0;

                distances = vector<cpx>(MAX_DIST);
                fftOut = vector<cpx>(MAX_DIST);

                // Distancia 0 é uma posicão de "possível"
                distances[0] = cpx(1, 0);

                for (int i = 0; i < N; i++)
                {
                        cin >> d;
                        if (d > maxDist)
                                maxDist = d;
                        distances[d] = cpx(1, 0);
                }

                int shiftAmount;

                for (shiftAmount = 0; (maxDist >> shiftAmount) != 0; shiftAmount++)
                        ;

                maxDist = 1 << (shiftAmount + 1);

                // fftOut <= transformada de distances
                FFT(distances, fftOut, maxDist, 1);

                // Multiplicacão termo a termo de f e g, no caso, f = g = fftOut
                // fftOut *= fftOut
                for (int i = 0; i < maxDist; i++)
                        fftOut[i] = fftOut[i] * fftOut[i];

                // transformada inversa da multiplcacão termo a termo.
                FFT(fftOut, distances, maxDist, -1);

                cin >> N;
                int total = 0;

                for (int i = 0; i < N; i++)
                {
                        cin >> d;

                        // Entra a distancia d
                        // e verifica se a parte real da distância[d] é positiva
                        // distância[d] guarda de quantas maneiras conseguimos atingir D
                        if (distances[d].real() > 0.01)
                                total++;
                }

                cout << total << endl;
        }
        return 0;
}
```

2.2 Matrizes

```
#include <bits/stdc++.h>
using namespace std;
```

```cpp
#define ll long long

typedef vector<ll> vl;
typedef vector<vl> vvl;
const int mod = 1000000;

// Retorna a matriz I_n
vvl matrixUnit(int n) {
    vvl res(n, vl(n));
    for (int i = 0; i < n; i++)
        res[i][i] = 1;
    return res;
}

// Retorna a+b
vvl matrixAdd(const vvl &a, const vvl &b) {
    int n = a.size();
    int m = a[0].size();
    vvl res(n, vl(m));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            res[i][j] = (a[i][j] + b[i][j]) % mod;
    return res;
}

// Retorna a*b
vvl matrixMul(const vvl &a, const vvl &b) {
    int n = a.size();
    int m = a[0].size();
    int k = b[0].size();
    vvl res(n, vl(k));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < k; j++)
            for (int p = 0; p < m; p++)
               res[i][j] = (res[i][j] + (long long) ((a[i][p] % mod) * (b[p][j] % mod) % mod)) % mod;
    return res;
}

// Retorna a matriz a^p
vvl matrixPow(const vvl &a, long long p) {
    if (p == 0)
        return matrixUnit(a.size());
    if (p & 1)
        return matrixMul(a, matrixPow(a, p - 1));
    return matrixPow(matrixMul(a, a), p / 2);
}

// Retorna sum^p_i=0 (a^i)
vvl matrixPowSum(const vvl &a, long long p) {
    long long n = a.size();
    if (p == 0)
        return vvl(n, vl(n));
    if (p % 2 == 0)
        return matrixMul(matrixPowSum(a, p / 2), matrixAdd(matrixUnit(n),  matrixPow(a, p / 2)));
    return matrixAdd(a, matrixMul(matrixPowSum(a, p - 1), a));
}

int main() {
        long long n, l, k, i;

        while(scanf("%lld %lld %lld", &n, &l, &k) > 0) {
                vvl matriz = vvl(2, vl(2));

                matriz[0][0] = l;
                matriz[0][1] = k;
                matriz[1][0] = 1;
                matriz[1][1] = 0;

                matriz = matrixPow(matriz, n / 5);

                printf("%06lld\n", matriz[0][0]);
        }
}
```

3 Geometria

3.1 Linha de eventos radial

```cpp
//  - Radial sweep in Q2 quadrant in nlogn.
//  - Sorts events using cross product to avoid dealing with
//    numeric problems.
#include <bits/stdc++.h>
using namespace std;

struct Point {
        Point(int x = 0, int y = 0) : x(x), y(y) {}

        bool operator<(const Point& o) const {
                // Order points in a quadrant by angle with origin:
                // Uses anti-clockwise order by returning true when the
                // cross product between the points is positive.
                return (x*o.y - y*o.x) > 0;
        }

        /*
        bool operator<=(const Point& o) const {
        return (x*o.y - y*o.x) >= 0;
        }
        */
        int x, y;
};

pair<int, int> solve(const vector<Point>& points) {
        map<Point, pair<int, int> > events;

        Point begin(0, 1);
        Point end(-1, 0);

        // Add events on the borders to guarantee that we consider them.
        events[begin];
        events[end];

        int superior = 0;  // Number of points in Q1 quadrant.
        int same = 0;      // Number of points in origin.
        int active = 0;    // Number of current points in Q2 and Q4 quadrant better
                                        // than origin.

        int best_pos = points.size();
        int worst_pos = 0;

        for (const auto& p : points) {
                if (p.x < 0 && p.y < 0) {}
                else if (p.x > 0 && p.y > 0) superior++;
                else if (p.x == 0 && p.y == 0) same++;
                else if (p.x <= 0 && p.y >= 0) {
                        // assert(begin <= Point(p.x, p.y));
                        //assert(Point(p.x, p.y) <= end);
                        events[Point(p.x, p.y)].first++;
                }
                else if (p.x >= 0 && p.y <= 0) {
                        //assert(begin <= Point(-p.x, -p.y));
                        //assert(Point(-p.x, -p.y) <= end);
                        active++;
                        events[Point(-p.x, -p.y)].second++;
                }
                else assert(false);
        }

        for (const auto& e : events) {
                int tie_best_pos = superior + active - e.second.second;
                int tie_worst_pos = superior + active + e.second.first + same;
                active += e.second.first - e.second.second;

                best_pos = min(best_pos, tie_best_pos);
                worst_pos = max(worst_pos, tie_worst_pos);
        }

        return make_pair(best_pos + 1, worst_pos + 1);
```

```
}

// Reads the set of points and centers them around Maria's product.
vector<Point> read() {
        int n, cx, cy;
        cin >> n >> cx >> cy;
        vector<Point> points(n - 1);
        for (Point& p : points) {
                cin >> p.x >> p.y;
                p.x -= cx;
                p.y -= cy;
        }
        return points;
}

int main() {
        auto input = read();
        auto solution = solve(input);

        for (auto& i : input)
                swap(i.x, i.y);

        assert(solution == solve(input));

        cout << solution.first << " " << solution.second << endl;
        return 0;
}
```

4 Estruturas de dados

4.1 Grafos

```cpp
#include <bits/stdc++.h>
using namespace std;

#define ll long long
#define pb push_back

typedef vector<int> vi;

struct Vertice
{
    int id, pai;
    ll dist;

    Vertice(int id, ll dist = 1, int pai = -1) : id(id), dist(dist), pai(pai) {}

    bool operator<(Vertice a) const
    {
        return a.dist < dist;
    }
};

typedef vector<Vertice> vv;
typedef vector<vv> vvv;

struct Grafo
{
    vvv g;
    vi pais;
    int n;

    Grafo(int n) : n(n)
    {
        g = vvv(n, vv());
        pais = vi(n);
    }

    void operator=(Grafo const &a)
    {
        g = a.g;
        pais = a.pais;
        n = a.n;
    }

    void addAresta(int a, int b, ll d = 0)
    {
        g[a].pb(Vertice(b, d));
    }

    void removeAresta(int a, int b)
    {
        g[a].erase(remove_if(g[a].begin(), g[a].end(), [b](Vertice v) { return v.id == b; }));
    }

    ll valAresta(int a, int b)
    {
        for (auto it : g[a])
            if (it.id == b)
                return it.dist;

        return 0;
    }

    void modificaAresta(int a, int b, ll dif)
    {
        for (auto &it : g[a])
            if (it.id == b)
            {
                it.dist += dif;
                break;
            }
```

```
    g[a].erase(remove_if(g[a].begin(), g[a].end(), [b](Vertice v) { return v.dist == 0; }));
}

ll dijkstra(int s, int d)
{
    priority_queue<Vertice> fila;
    bool visitados[n] = {0};

    fill(pais.begin(), pais.end(), -1);

    fila.push(Vertice(s, 0));

    auto top = fila.top();

    while (top.id != d)
    {
        if (!visitados[top.id])
        {
            for (auto &it : g[top.id])
                if (!visitados[it.id])
                    fila.push(Vertice(it.id, it.dist + top.dist, top.id));

            visitados[top.id] = 1;
            pais[top.id] = top.pai;
        }

        fila.pop();

        if (fila.empty())
            return -1;

        top = fila.top();
    }

    pais[top.id] = top.pai;

    return top.dist;
}

ll busca(int s, int d)
{
    queue<Vertice> fila;
    bool visitados[n] = {0};

    fill(pais.begin(), pais.end(), -1);

    fila.push(Vertice(s, 0));

    auto top = fila.front();

    while (top.id != d)
    {
        if (!visitados[top.id])
        {
            for (auto &it : g[top.id])
                if (!visitados[it.id])
                    fila.push(Vertice(it.id, it.dist + 1, top.id));

            visitados[top.id] = 1;
            pais[top.id] = top.pai;
        }

        fila.pop();

        if (fila.empty())
            return -1;

        top = fila.front();
    }

    pais[top.id] = top.pai;

    return top.dist;
}

ll fluxo_maximo(int s, int d)
{
```

```
        int u, v;
        ll flow = 0;

        Grafo g2 = *this;

        while (g2.busca(s, d) >= 0)
        {
            ll path = 1ll << 50;

            for (v = d; v != s; v = u)
            {
                u = g2.pais[v];
                path = min(path, valAresta(u, v));
            }

            for (v = d; v != s; v = u)
            {
                u = g2.pais[v];
                g2.modificaAresta(u, v, -path);
                g2.modificaAresta(v, u, path);
            }

            flow += path;
        }

        return flow;
    }
};

int main()
{
    Grafo g(20);

    g.addAresta(1, 2, 1);
    g.addAresta(1, 3, 5);
    g.addAresta(2, 1, 6);
    g.addAresta(3, 2, 10);

    g.removeAresta(1, 2);

    for (auto it : g.g[1])
        cout << it.id << endl; // 3

    cout << g.dijkstra(1, 2) << endl; // 15

    cout << g.fluxo_maximo(1, 2) << endl; // 5
}
```

4.2 Wavelet-tree

```
#include <bits/stdc++.h>
using namespace std;

const int N = 10000;

struct KthSmallest
{
        struct Seg
        {
                int l, r, mid;

                void set(int _l, int _r)
                {
                        l = _l;
                        r = _r;
                        mid = l + r >> 1;
                }
        } seg[N << 2];

        int b[25][N], left[25][N], sorted[N];

        void init(int *a, int n)
        {
                for (int i = 0; i < n; i++)
```

```
                b[0][i] = sorted[i] = a[i];

        sort(sorted, sorted + n);
        build(0, n, 0, 1);
}

void build(int l, int r, int d, int idx)
{
        seg[idx].set(l, r);

        if (l + 1 == r)
                return;

        int mid = seg[idx].mid;
        int lsame = mid - l;

        for (int i = l; i < r; i++)
                if (b[d][i] < sorted[mid])
                        lsame--;

        int lpos = l, rpos = mid, same = 0;

        for (int i = l; i < r; ++i)
        {
                left[d][i] = (i != l ? left[d][i - 1] : 0);

                if (b[d][i] < sorted[mid])
                {
                        left[d][i]++;
                        b[d + 1][lpos++] = b[d][i];
                }
                else if (b[d][i] > sorted[mid])
                        b[d + 1][rpos++] = b[d][i];
                else
                {
                        if (same < lsame)
                        {
                                same++;
                                left[d][i]++;
                                b[d + 1][lpos++] = b[d][i];
                        }
                        else
                        {
                                b[d + 1][rpos++] = b[d][i];
                        }
                }
        }

        build(l, mid, d + 1, idx << 1);
        build(mid, r, d + 1, idx << 1 | 1);
}

//Quando ordernarmos [l, r), qual é o k-ésimo termo?
int kth(int l, int r, int k, int d = 0, int idx = 1)
{ // k : 1-origin!!!
        if (l + 1 == r)
                return b[d][l];

        int ltl = (l != seg[idx].l ? left[d][l - 1] : 0);
        int tl = left[d][r - 1] - ltl;

        if (tl >= k)
        {
                int newl = seg[idx].l + ltl;
                int newr = seg[idx].l + ltl + tl;

                return kth(newl, newr, k, d + 1, idx << 1);
        }
        else
        {
                int mid = seg[idx].mid;
                int tr = r - l - tl;
                int ltr = l - seg[idx].l - ltl;
                int newl = mid + ltr;
                int newr = mid + ltr + tr;
```

```
                        return kth(newl, newr, k - tl, d + 1, idx << 1 | 1);
                }
        }

        //When sorting [l, r), what number will x come in?
        //If there are two or more x's, return the rank of the last one.
        //If there is no x, return the rank of the largest but less than x.
        //When there is no less than x, 0 is returned.
        int rank(int l, int r, int x, int d = 0, int idx = 1)
        {
                if (seg[idx].l + 1 == seg[idx].r)
                        return l + 1 == r && sorted[l] <= x;

                int ltl = (l != seg[idx].l ? left[d][l - 1] : 0);
                int tl = left[d][r - 1] - ltl;
                int mid = seg[idx].mid;

                if (x < sorted[mid])
                {
                        int newl = seg[idx].l + ltl;
                        int newr = seg[idx].l + ltl + tl;

                        return rank(newl, newr, x, d + 1, idx << 1);
                }
                else
                {
                        int tr = r - l - tl;
                        int ltr = l - seg[idx].l - ltl;
                        int newl = mid + ltr;
                        int newr = mid + ltr + tr;

                        return tl + rank(newl, newr, x, d + 1, idx << 1 | 1);
                }
        }

        // Quantos x existem entre [l,r)
        int freq(int l, int r, int x)
        {
                return rank(l, r, x) - rank(l, r, x - 1);
        }
} kth;

int main()
{
        int a[8] = {6, 12, 5, 17, 10, 2, 7, 3};

        kth.init(a, 8);

        cout << kth.kth(2, 7, 3) << endl;  // 7
        cout << kth.rank(2, 7, 7) << endl; // 3
}
```

4.3 Seg-tree

```
#include <algorithm>

using namespace std;

#define MAX 1000000 // O valor aqui tem que ser >= 2 * tamanho do maior n
#define INF 1 << 28

// Não necessáriamente é um int, pode ser uma segtree de struct etc;
int init[MAX], tree[MAX], lazy[MAX];

void build_tree(int node, int a, int b)
{
        if (a > b)
                return;

        // Se folha
        if (a == b)
        {
                tree[node] = init[a];
                lazy[node] = 0;
```

```
                return;
        }

        build_tree(node * 2, a, (a + b) / 2);
        build_tree(node * 2 + 1, 1 + (a + b) / 2, b);

        // Se nó
        tree[node] = tree[node * 2] + tree[node * 2 + 1];
        lazy[node] = 0;
}

void update_tree(int node, int a, int b, int i, int j, int value)
{
        // Se fora do intervalo - retorna
        if (a > b || a > j || b < i)
                return;

        if (lazy[node] != 0)
        {
                //Atualizacão atrasada.
                tree[node] += lazy[node];

                // Passa lazy para filhos
                if (a != b)
                {
                        lazy[node * 2] += lazy[node];
                        lazy[node * 2 + 1] += lazy[node];
                }

                //Reseta o nó
                lazy[node] = 0;
        }

        // Se o nó atual cobre todo o intervalo
        if (a >= i && b <= j)
        {
                tree[node] += value;

                if (a != b)
                {
                        lazy[node * 2] += value;
                        lazy[node * 2 + 1] += value;
                }

                return;
        }

        // Se tem um pedaco em cada filho.

        // Atualiza os filhos.
        update_tree(node * 2, a, (a + b) / 2, i, j, value);
        update_tree(1 + node * 2, 1 + (a + b) / 2, b, i, j, value);

        // Atualiza o pai.
        tree[node] = tree[node * 2] + tree[node * 2 + 1];
}

int query_tree(int node, int a, int b, int i, int j)
{
        // Se fora do intervalo
        if (a > b || a > j || b < i)
        {
                //Aqui deverá ser retornado o elemento neutro para a operacão desejada
                return 0;
        }

        if (lazy[node] != 0)
        {
                //Atualizacão atrasada.
                tree[node] += lazy[node];

                //Se não folha, passa lazy pros filhos
                if (a != b)
                {
                        lazy[node * 2] += lazy[node];
                        lazy[node * 2 + 1] += lazy[node];
```

```
                }

                //Reseta o nó
                lazy[node] = 0;
        }

        // Se o nó cobre o intervalo
        if (a >= i && b <= j)
                return tree[node];


        // Se o intervalo está um pedaco em cada filho.
        int q1 = query_tree(node * 2, a, (a + b) / 2, i, j);
        int q2 = query_tree(1 + node * 2, 1 + (a + b) / 2, b, i, j);

        // Retorna a combinacão dos intervalos.
        return q1 + q2;
}

/*

Uso:

Assumindo que "n" é o numero de termos que o segmento tem
Inicialize "init" com os valores iniciais:

*       for(i = 0; i < n; scanf("%d", val), i++)
*      init[i] = val;

E mande construir a arvore:

*   build_tree(1, 0, n-1);

Para atualizar a arvore:

*   update_tree(1, 0, n-1, inicio, fim, val);
*   Onde inicio é a posicão inicial do segmento desejado e fim é a posicão final do mesmo
*       e val é o quanto você quer alterar os valores desse seguimento

Para fazer queries

*   query_tree(1, 0, n-1, inicio, fim);
*       Onde inicio é a posicão inicial do segmento desejado e fim é a posicão final do mesmo
*       o retorno terá o mesmo tipo que os dados guardados na arvore e será o resultado do segmento pesquisado

*/
```

4.4 Geometria

```
#include <bits/stdc++.h>

using namespace std;

const double EPS = 1e-10;

inline int cmp( double x, double y = 0, double tol = EPS ) {
    return (x <= y + tol ) ? ( x + tol < y ) ? -1 : 0 : 1;
}

struct Point {
    double x, y;

    Point( double x = 0, double y = 0 ) : x( x ), y( y ) {}

    Point operator+( Point q ) const {
        return Point( x + q.x, y + q.y );
    }

    Point operator-( Point q ) const {
        return Point( x - q.x, y - q.y );
    }

    Point operator*( double t ) const {
        return Point( x * t, y * t );
    }
```

```
    Point operator/( double t ) const {
        return Point( x / t, y / t );
    }

    double operator*( Point q )const {
        return x * q.x + y * q.y;
    }

    double operator^( Point q ) const {
        return x * q.y - y * q.x;
    }

    int cmp( Point q ) const {
        if ( int t = ::cmp( x, q.x ) )
                        return t;
        return ::cmp( y, q.y );
    }

    bool operator==( Point q ) const {
        return cmp( q ) == 0;
    }

    bool operator!=( Point q ) const {
        return cmp( q ) != 0;
    }

    bool operator<( Point q ) const {
        return cmp( q ) < 0;
    }

    static Point pivot;
};

Point Point::pivot;

typedef vector<Point> Polygon;

inline double abs( Point& p ) {
    return hypot( p.x, p.y );
}

inline double arg( Point& p ) {
    return atan2( p.y, p.x );
}

//Verifica o sinal do produto vetorial entre os vetores (p-r) e (q - r)
inline int ccw( Point& p, Point& q, Point& r ) {
    return cmp( ( p - r ) ^ ( q - r ) );
}

//calcula o angulo orientado entre os vetores (p-q) e (r - q)
inline double angle( Point& p, Point &q, Point& r ) {
    Point u = p - q, w = r - q;
    return atan2( u ^ w, u * w );
}

//Decide se o ponto p esta sobre a reta que passa por p1p2.
bool pointoSobreReta( Point& p1, Point &p, Point& p2 ) {
    return ccw( p1, p2, p ) == 0;
}

//Decide de p esta sobre o segmento p1p2
bool between( Point& p1, Point &p, Point& p2 ) {
    return ccw( p1, p2, p ) == 0 && cmp( ( p1 - p ) * ( p2 - p ) ) <= 0;
}

//Calcula a distancia do ponto p a reta que passa por p1p2
double retaDistance( Point& p1, Point& p2, Point &p ) {
    Point A = p1 - p, B = p2 - p1;
    return fabs( A ^ B ) / sqrt( B * B );
}

//Calcula a distancia do ponto p ao segmento de reta que passa por p1p2
double segDistance( Point& p1, Point& p2, Point &p ) {
    Point A = p1 - p, B = p1 - p2, C = p2 - p;
```

```
    double a = A * A, b = B * B, c = C * C;
    if ( cmp( a, b + c ) >= 0 ) return sqrt( c );
    if ( cmp( c, a + b ) >= 0 )return sqrt( a );
    return fabs( A ^ C ) / sqrt( b );

}
//Calcula a area orientada do poligono T.
double polygonArea( Polygon& T ) {
    double s = 0.0;
    int n = T.size( );
    for ( int i = 0; i < n; i++ )
    {
        s += T[i] ^ T[( i + 1 ) % n];
    }
    return s / 2.0; //Retorna a area com sinal
}
//Classifica o ponto p em relacao ao poligono T dependendo se ele está
//na fronteira (-1) no exterior (0) ou no interior (1).
int inpoly( Point& p, Polygon& T ) {
    //-1 sobre, 0 fora, 1 dentro
    double a = 0.0;
    int n = T.size( );
    for ( int i = 0; i < n; i++ )
    {
        if ( between( T[i], p, T[( i + 1 ) % n] ) ) return -1;
        a += angle( T[i], p, T[( i + 1 ) % n] );
    }
    return cmp( a ) != 0;
}
//Ordenacao radial.
bool radialSort( Point p, Point q ) {
    Point P = p - Point::pivot, Q = q - Point::pivot;
    double R = P ^ Q;
    if ( cmp( R ) ) return R > 0;
    return cmp( P * P, Q * Q ) < 0;
}

//Determina o convex hull de T. ATENCAO. A lista de pontos T e destruida.
Polygon convexHull( vector<Point>& T ) {
    int j = 0, k, n = T.size( );
    Polygon U( n );
    Point::pivot = *min_element( T.begin( ), T.end( ) );
    sort( T.begin( ), T.end( ), radialSort );

    for ( k = n - 2; k >= 0 && ccw( T[0], T[n - 1], T[k] ) == 0; k-- );
    reverse( ( k + 1 ) + T.begin( ), T.end( ) );

    for ( int i = 0; i < n; i++ )
    {
        // troque o >= por > para manter pontos colineares
        while (j > 1 && ccw( U[j - 1], U[j - 2], T[i] ) >= 0) j--;
        U[j++] = T[i];
    }
    U.resize( j );
    return U;
}
//Intersecão de semi-retas (p1 -> p2), (p3 -> p4)
bool segIntercept(Point p1, Point p2, Point p3, Point p4) {
        return ccw(p1, p2, p3) != ccw(p1, p2, p4) && ccw(p3, p4, p1) != ccw(p3, p4, p2);
}
```