Standard Code Library

Radium

November 3, 2019

Contents

一切的开始

宏定义

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

#ifndef ONLINE_JUDGE
#define dbg(args...)                                    \
    do {                                                \
        cerr << "\033[32;1m" << #args << " -> "; \
        err(args);                                      \
    } while (0)
#else
#define dbg(...)
#endif
void err() { cerr << "\033[39;0m" << endl; }
template <template <typename...> class T, typename t, typename... Args>
void err(T<t> a, Args... args) {
    for (auto x : a) cerr << x << ' ';
    err(args...);
}
template <typename T, typename... Args>
void err(T a, Args... args) {
    cerr << a << ' ';
    err(args...);
}
// ------------------------------------------------------------------------------
```

读入挂

```cpp
inline char nc() {
    static char buf[100000], *p1 = buf, *p2 = buf;
    return p1 == p2 &&
                   (p2 = (p1 = buf) + fread(buf, 1, 100000, stdin), p1 == p2)
               ? EOF
               : *p1++;
}
template <typename T>
bool rn(T& v) {
    static char ch;
    while (ch != EOF && !isdigit(ch)) ch = nc();
    if (ch == EOF) return false;
    for (v = 0; isdigit(ch); ch = nc()) v = v * 10 + ch - '0';
    return true;
}

template <typename T>
void o(T p) {
    static int stk[70], tp;
```

```
    if (p == 0) {
        putchar('0');
        return;
    }
    if (p < 0) {
        p = -p;
        putchar('-');
    }
    while (p) stk[++tp] = p % 10, p /= 10;
    while (tp) putchar(stk[tp--] + '0');
}
```

快速幂

```
ll pow(ll x, ll n, ll mod) {
    assert(n >= 0);
    ll ret = mod != 1;
    for (x %= mod; n; n >>= 1, x = x * x % mod)
        if (n & 1) ret = ret * x % mod;
    return ret;
}
ll inv(ll a, ll p) { return pow(a, p - 2, p); }
inline ll mul(ll a, ll b, ll mod) {
    if (mod <= 1000000000)
        return a * b % mod;
    else if (mod <= 100000000000001l)
        return (((a * (b >> 20) % mod) << 20) + (a * (b & ((1 << 20) - 1)))) % mod;
    else {
        ll d = (ll)floor(a * (long double)b / mod + 0.5);
        ll ret = (a * b - d * mod) % mod;
        if (ret < 0) ret += mod;
        return ret;
    }
}
```

计算几何

二维几何基础

```cpp
/*************** 基本定义 ***************/

// 玄学 eps，可手动调
const double eps = 1e-10;
// 精度较高的 PI
const double pi = acos(-1.0);
// 三态函数，绝对值小于 eps 时认为是 0
int dcmp(double x) {
    if (fabs(x) < eps) return 0;
    else return x<0?-1:1;
}

/*************** 点操作 ***************/

// 点定义
struct Point {
    double x, y;
    Point(double x=0, double y=0):x(x),y(y) {} // 构造函数
};

// 点比较（先比较 x 坐标，再比较 y 坐标）
bool operator < (const Point &a, const Point &b) {
    return a.x<b.x || (a.x==b.x && a.y<b.y);
}
// 点相等
bool operator == (const Point &a, const Point &b) {
    return dcmp(a.x-b.x)==0 && dcmp(a.y-b.y)==0;
}

/*************** 向量操作 ***************/

// 程序实现上，Vector 只是 Point 的别名
typedef Point Vector;

// 向量加法
Vector operator + (Vector A, Vector B) {
    return Vector(A.x+B.x, A.y+B.y);
}
// 向量减法
Vector operator - (Vector A, Vector B) {
    return Vector(A.x-B.x, A.y-B.y);
}
// 向量数乘
Vector operator * (Vector A, double p) {
    return Vector(A.x*p, A.y*p);
}
// 向量数除
```

```
48  Vector operator / (Vector A, double p) {
49      return Vector(A.x/p, A.y/p);
50  }
51  // 向量点积
52  double Dot(Vector A, Vector B) {
53      return A.x*B.x + A.y*B.y;
54  }
55  // 向量长度
56  double Length(Vector A) {
57      return sqrt(Dot(A, A));
58  }
59  // 向量夹角
60  double Angle(Vector A, Vector B) {
61      return acos(Dot(A, B) / Length(A) / Length(B));
62  }
63  // 向量叉积
64  double Cross(Vector A, Vector B) {
65      return A.x*B.y - A.y*B.x;
66  }
67  // 向量旋转
68  Vector Rotate(Vector A, double rad) {
69      return Vector(A.x*cos(rad)-A.y*sin(rad), A.x*sin(rad)+A.y*cos(rad));
70  }
71
72  /*************** 距离 ***************/
73
74  // 两点间欧几里得距离
75  double Odistance(Point a, Point b) {
76      return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
77  }
78  // 两点间曼哈顿距离 |a.x-b.x|+|a.y-b.y|
79  double Mdistance(Point a, Point b) {
80      return fabs(a.x-b.x) + fabs(a.y-b.y);
81  }
82  // 两点间切比雪夫距离 max(|a.x-b.x|,|a.y-b.y|)
83  double Cdistance(Point a, Point b) {
84      return max(fabs(a.x-b.x), fabs(a.y-b.y));
85  }
86  // 曼哈顿距离和切比雪夫距离的转换（见下）
87
88  // 点到直线距离
89  double DistanceToLine(Point P, Point A, Point B) {
90      Vector v1 = B-A, v2 = P-A;
91      // 去掉绝对值得到有向线段
92      return fabs(Cross(v1, v2)) / Length(v1);
93  }
94  // 点到线段距离，垂线不在线段上时距离为 PA 或 PB 的长度
95  double DistanceToSegment(Point P, Point A, Point B) {
96      if (A == B) return Length(P-A);
97      Vector v1 = B-A, v2 = P-A, v3 = P-B;
98      if (dcmp(Dot(v1, v2)) < 0) return Length(v2);
99      else if (dcmp(Dot(v1, v3)) < 0) return Length(v3);
```

```
    else return fabs(Cross(v1, v2)) / Length(v1);
}

/*************** 求面积 ***************/

// 三角形两倍有向面积，可能为负!
double Area2(Point A, Point B, Point C) {
    return Cross(B-A, C-A);
}
// 多边形有向面积，可能为负!
double PolygonArea(Point *p, int n) {
    double area = 0.0;
    for (int i = 1; i < n-1; i++)
        area += Cross(p[i]-p[0], p[i+1]-p[0]);
    return area/2.0;
}

/*************** 相交判定 ***************/
// 直线相交，注意 Cross(v,w) 不能为 0，为 0 两直线平行
Point GetLineInt(Point P, Vector v, Point Q, Vector w) {
    ector u = P - Q;
    double t = Cross(w,u) / Cross(v,w);
    return P+v*t;
}
// 点在线上
bool OnSegment(Point p, Point a1, Point a2) {
    return dcmp(Cross(a1-p, a2-p))==0 && dcmp(Dot(a1-p, a2-p)) < 0;
}
// 线段相交，线段 a1a2, b1b2
// 注意不包含端点，端点需用 OnSegment 特判
bool SegProInt(Point a1, Point a2, Point b1, Point b2) {
    double c1 = Cross(a2-a1, b1-a1), c2 = Cross(a2-a1, b2-a1),
           c3 = Cross(b2-b1, a1-b1), c4 = Cross(b2-b1, a2-b1);
    return dcmp(c1)*dcmp(c2)<0 && dcmp(c3)*dcmp(c4)<0;
}
// 线段与直线相交，直线 a1a2, 线段 b1b2
bool SegLineInt(Point a1, Point a2, Point b1, Point b2) {
    double c1 = Cross(a2-a1,b1-a1), c2 = Cross(a2-a1,b2-a1);
    return dcmp(c1)*dcmp(c2)<=0;
}
```

距离

欧氏距离：一般也称作欧几里得距离。在平面直角坐标系中，设点 A, B 的坐标分别为 $(x1, y1), (x2, y2)$ 则两点间的欧氏距离为：

$$|AB| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

曼哈顿距离：在二维空间内，两个点之间的曼哈顿距离为它们横坐标之差的绝对值与纵坐标之差的绝对值之和。设点 $A(x1, y1), B(x2, y2)$ ，则 A, B 之间的曼哈顿距离用公式可以表示为：

$$d(A, B) = |x_1 - x_2| + |y_1 - y_2|$$

切比雪夫距离：在二维空间内，两个点之间的切比雪夫距离为它们横坐标之差的绝对值与纵坐标之差的绝对值的最大值。设点 $A(x1, y1), B(x2, y2)$ ，则 A, B 之间的切比雪夫距离用公式可以表示为：

$d(A, B) = max(|x_1 - x_2|, |y_1 - y_2|)$

曼哈顿距离与切比雪夫距离的转换：曼哈顿坐标系是通过切比雪夫坐标系旋转 45° 后，再缩小到原来的一半得到的。

- 将一个点 (x, y) 的坐标变为 $(x + y, x - y)$ 后，原坐标系中的曼哈顿距离等于新坐标系中的切比雪夫距离
- 将一个点 (x, y) 的坐标变为 $(\frac{x+y}{2}, \frac{x-y}{2})$ 后，原坐标系中的切比雪夫距离等于新坐标系中的曼哈顿距离

```cpp
// 两点间欧几里得距离
double Odistance(Point a, Point b) {
    return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
}
// 两点间曼哈顿距离 |a.x-b.x|+|a.y-b.y|
double Mdistance(Point a, Point b) {
    return fabs(a.x-b.x) + fabs(a.y-b.y);
}
// 两点间切比雪夫距离 max(|a.x-b.x|,|a.y-b.y|)
double Cdistance(Point a, Point b) {
    return max(fabs(a.x-b.x), fabs(a.y-b.y));
}
```

极角排序

```cpp
// 利用 atan2, 精度低, 速度快
bool cmp1(Point p1, Point p2) {
    if (atan2(p1.y,p1.x) == atan2(p2.y,p2.x))
        return p1.x<p2.x;
    return atan2(p1.y,p1.x) < atan2(p2.y,p2.x);
}

// 利用叉积, 精度高, 速度慢
bool cmp2(Point p1, Point p2) {
    Point c(0,0);
    if (dcmp(Cross(p1-c, p2-c))==0)
        return p1.x<p2.x;
    return dcmp(Cross(p1-c, p2-c))>0;
}

// 象限排序, 注意包含四个坐标轴
int Quadrant(Point p) {
    if (p.x>0 && p.y>=0)  return 1;
    if (p.x<=0 && p.y>0)  return 2;
    if (p.x<0 && p.y<=0)  return 3;
    if (p.x>=0 && p.y<0)  return 4;
}

// 先按象限排, 再按极角排
bool cmp3(Point p1, Point p2) {
```

```
    if (Quadrant(p1)==Quadrant(p2))
        return cmp1(p1, p2);
    return Quadrant(p1)<Quadrant(p2);
}
```

Pick 定理

Pick 定理：给定顶点座标均是整点（或正方形格子点）的简单多边形，皮克定理说明了其面积 A 和内部格点数目 i、边上格点数目 b 的关系：$A = i + \frac{b}{2} - 1$ 。

它有以下推广：

- 取格点的组成图形的面积为一单位。在平行四边形格点，皮克定理依然成立。套用于任意三角形格点，皮克定理则是 $A = 2 \times i + b - 2$。
- 对于非简单的多边形 P，皮克定理 $A = i + \frac{b}{2} - \chi(P)$，其中 $\chi(P)$ 表示 P 的 **欧拉特征数**。
- 高维推广：Ehrhart 多项式
- 皮克定理和 **欧拉公式** $V - E + F = 2$ 等价。

边上格点数目 b 求法：以格子点为顶点的线段，覆盖的点的个数为 $gcd(dx, dy)$，其中，dx, dy 分别为线段横向占的点数和纵向占的点数。如果 dx 或 dy 为 0，则覆盖的点数为 dy **或** dx。

```
struct Point {
    int x, y;
    Point(int x=0, int y=0):x(x),y(y) {} // 构造函数
};
typedef Point Vector;
Vector operator - (Vector A, Vector B) {
    return Vector(A.x-B.x, A.y-B.y);
}
int Cross(Vector A, Vector B) {
    return A.x*B.y - A.y*B.x;
}
int getb(Point A, Point B) {
    int dx=abs(A.x-B.x),dy=abs(A.y-B.y);
    if (dx==0) return dy;
    if (dy==0) return dx;
    return __gcd(dx,dy);
}
int Area2(Point A, Point B, Point C) {
    return abs(Cross(B-A, C-A));
}
int calA(int i, int b) {
    return i+b/2-1;
}
int cali(int A, int b) {
    return A-b/2+1;
}
```

凸包

Andrew 算法：

```cpp
int Andrew(Point *p, int n, Point *ch) {
    sort(p, p+n); // 需要重载 < 号
    int m = 0;
    for (int i=0; i<n; i++) {
        while(m>1 && Cross(ch[m-1]-ch[m-2], p[i]-ch[m-2]) <= 0) m--;
        ch[m++]=p[i];
    }
    int k=m;
    for (int i=n-2; i>=0; i--) {
        while(m>k && Cross(ch[m-1]-ch[m-2], p[i]-ch[m-2]) <= 0) m--;
        ch[m++]=p[i];
    }
    if (n>1) m--;
    return m;
}
```