题目

C 寄蒜几盒 I

时间限制：1000ms 内存限制：65536kb

通过率：153/185 (82.70%) 正确率：153/559 (27.37%)

题目描述

小 A 想给小 B 寄几盒蒜。

小 A 在 $\mathbb{R}^2$ 平面的点 A 处，小 B 可以在线段 PQ 上任意移动。

为了使得寄蒜的邮费最小，小 A 想知道他与小 B 之间的最小距离。

输入

本题测试点包含多组数据。

第一行，一个正整数 T（$1 \leq T \leq 10^5$），表示数据组数。

对于每组数据：

第一行，两个正整数 x_A, y_A（$|x_A|, |y_A| \leq 10^6$），表示 A 的坐标。

第二行，四个正整数 x_P, y_P, x_Q, y_Q（$|x_P|, |y_P|, |x_Q|, |y_Q| \leq 10^6$），表示 P 与 Q 的坐标。

输出

对于每组数据：

输出一行，一个实数，表示 A 与线段 PQ 间的最小距离，答案保留三位小数。

思路

- 如果 a 等于 b，说明给定的线段是一个点，直接返回 p 到 a 的距离。
- 向量 v1 表示线段的方向，即 b - a。
- 向量 v2 表示从 a 指向 p 的向量。

- 向量 v3 表示从 b 指向 p 的向量。
- 如果 v1 和 v2 的点积小于等于零，说明 p 在线段 ab 的逆时针方向（或在线段上），直接返回 v2 的长度。
- 如果 v1 和 v3 的点积大于等于零，说明 p 在线段 ba 的逆时针方向（或在线段上），直接返回 v3 的长度。
- 如果 p 不在上述两个情况中，说明 p 在 ab 线段的内部，计算 v1 和 v2 的叉积，并除以 v1 的模长，得到 p 到线段 ab 的垂直距离。

这样，Dis_P_S 函数可以正确计算点到线段的距离。

代码

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

using namespace std;
const double eps = 1e-8;
const double Pi = acos(-1.0);

int dcmp(const double x) {
    if (fabs(x) <= eps) return 0;
    return x > 0 ? 1 : -1;
}

struct Point {
    double x, y;
    Point() {}
    Point(double x, double y) : x(x), y(y) {}

    Point operator+(const Point& a) const { return Point(x + a.x, y + a.y); }
    Point operator-(const Point& a) const { return Point(x - a.x, y - a.y); }
    Point operator*(const double& a) const { return Point(x * a, y * a); }
    Point operator/(const double& a) const { return Point(x / a, y / a); }
    double operator*(const Point& a) const { return x * a.x + y * a.y; }  // 点积
    double operator^(const Point& a) const { return x * a.y - y * a.x; }  // 叉积
    bool operator<(const Point& a) const { return (x != a.x) ? x < a.x : y < a.y;
}
    bool operator==(const Point& a) const { return !dcmp(x - a.x) && !dcmp(y -
a.y); }
};

typedef Point Vector;
//求模长
double lth(const Vector a){return sqrt(a*a);}
//求向量夹角
double ang(const Vector a){return atan2(a.y,a.x);}
double ang(const Vector a,const Vector b){return acos(a*b/lth(a)/lth(b));}
//三角形面积
double S_(const Point a,const Point b,const Point c){return ((b-a)^(c-a))/2;}
```

```
//法向量
Vector Nol(const Vector a){return Vector(-a.y,a.x)/lth(a);}
bool P_S(Point p,Point a,Point b)
{
Vector A = a-p,B = b-p;
return dcmp(A*B)<0 && dcmp(A^B)==0;//点积小于0->p不在a、b上， 叉积==0 (p在直线ab上)
}

double Dis_P_S(Point p,Point a,Point b)
{
if(a==b)return lth(p-a);
Vector v1 = b-a,v2 = p-a,v3 = p-b;
if(v1*v2<=0)return lth(v2);
else if(v1*v3>=0)return lth(v3);
else return (v1^v2)/lth(v1);
}


int main() {
    int t;
    cin >> t;

    while(t --) {

        vector<Point> points(3);

        for (int j = 0; j < 3; ++j) {
            cin >> points[j].x >> points[j].y;
        }

        double res = fabs( Dis_P_S(points[0],points[1],points[2]));
        cout<<fixed<<setprecision(3)<<res<<endl;
    }

    return 0;
}
```

要注意的细节

点到线段最短距离的运算与点到直线的最短距离的运算二者之间存在一定的差别，即求点到线段最短距离时需要考虑参考点在沿线段方向的投影点是否在线段上，若在线段上才可采用点到直线距离公式