算法分析与设计C5-B

22373340 詹佳博

题目描述

妮妮向二维平面上撒了 n 个点，她想知道这 n 个点是否共线。

输入格式

第一行一个正整数 $t(1 \le t \le 10)$ ，表示数据组数。

对于每组数据，第一行一个正整数 $n(3 \le n \le 10^5)$ ，含义同题目描述。

接下来 n 行，每行两个整数 $x_i, y_i(-10^9 \le x_i, y_i \le 10^9)$，表示第 i 个点的坐标。

保证任意两点坐标不同。

输出格式

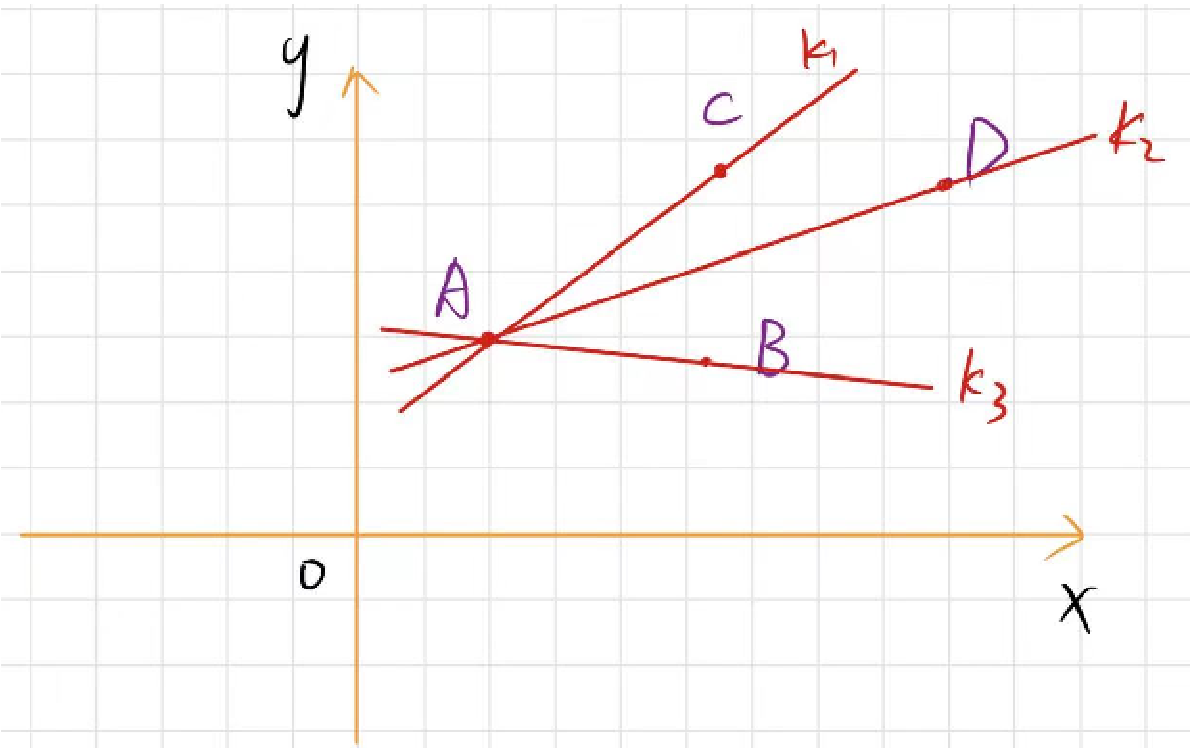
对于每组数据，输出一行一个字符串：

- 如果存在三点不共线，输出 `how?`；
- 否则，输出 `boo how! boo how!`。

题目分析

分析题目第一直觉会是将每两个点的函数求出来，再比较每个函数是否一样。但这样做的时间复杂度高达 $O(n^2)$ ，故放弃。

接着就会想，既然三点共线，那就可以固定2个点，一个为起点一个为终点，求出斜率；然后再将剩下的点与这2个点中任意一个求斜率，只需观察斜率是否均相等即可(或是判断点是否在函数上)。时间复杂度在$O(n)$，可以接受。



但是斜率面临精度问题，故转而回到《算法导论》所介绍的方法：向量法。判断三点是否共线，只需要判断其任意两点的向量叉积是否为0。这样做的理由是，向量叉积的大小意义就是其构成的平行四边形的大小，若为0必定共线。

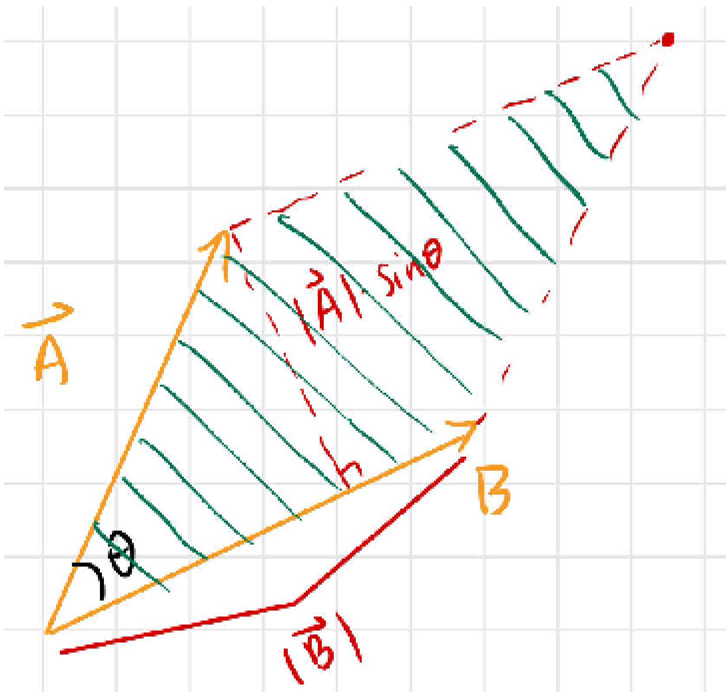
$$\text{对于}\vec{A} = (a, b)\text{和}\vec{B} = (c, d):$$

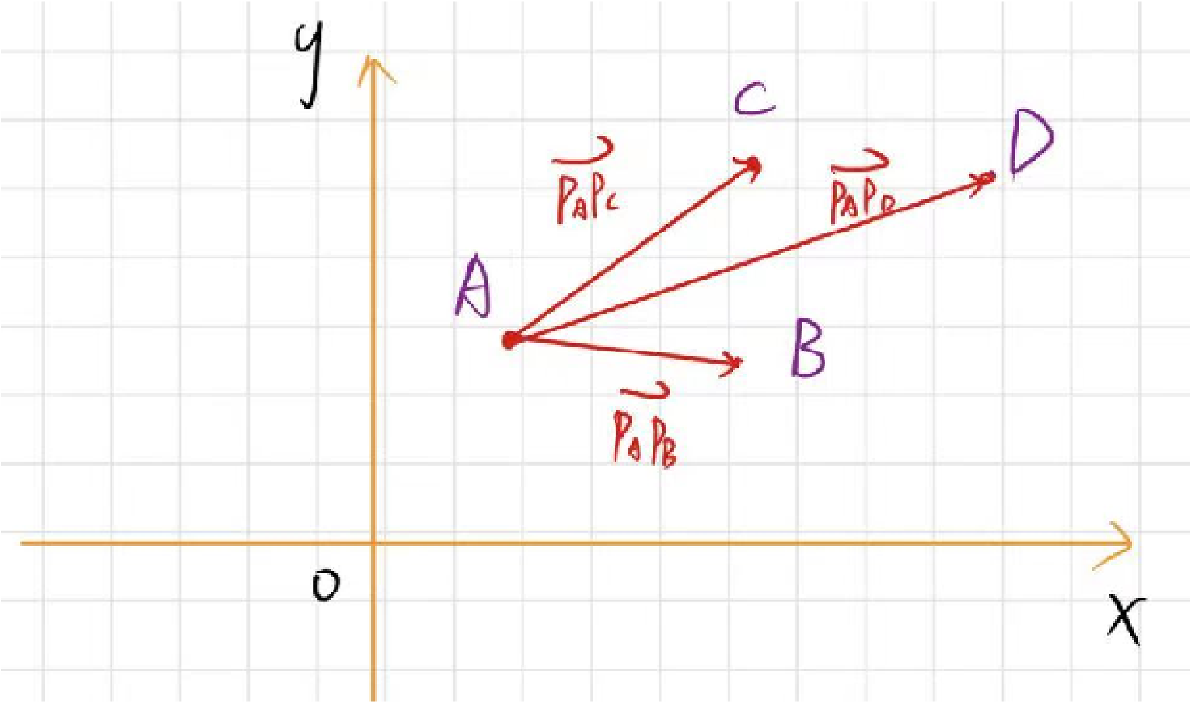
$$\vec{A} \cdot \vec{B} = ac + bd = |\vec{A}||\vec{B}| \cdot cos\theta$$

$$\text{引入}z\text{轴}: \vec{A} = (a, b, 0)\text{和}\vec{B} = (c, d, 0):$$

$$\vec{A} \times \vec{B} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ a & b & 0 \\ c & d & 0 \end{vmatrix} = (0)\vec{i} - (0)\vec{j} + (ad - bc)\vec{z}$$

$$|\vec{A} \times \vec{B}| = ad - bc = |\vec{A}||\vec{B}| \cdot sin\theta$$





题目求解

- 先准备好初始条件。

```
#include <bits/stdc++.h>
#define LL long long
const int inf_int = 0x3f3f3f3f;
const LL inf_ll = 0x3f3f3f3f3f3f3f3f;

using namespace std;

struct Point {
    LL x, y;
};

LL direction(Point pi, Point pj, Point pk) {//返回叉积大小,pipk x pipj向量
    return (pk.x - pi.x) * (pj.y - pi.y) - (pj.x - pi.x) * (pk.y - pi.y);
}
```

- 最开始想到用数组存所有的点，然后接连判断即可。

```
Point p[300010];//所有的点
int main() {
    int t;
    scanf("%d", &t);

    while (t--) {
        bool flag = false;//判断是否不共线
        int n;
        scanf("%d", &n);
        scanf("%lld%lld", &p[0].x, &p[0].y);
        scanf("%lld%lld", &p[1].x, &p[1].y);
        //初始化头两个点
        for (int i = 2; i < n; i++) {
            scanf("%lld%lld", &p[i].x, &p[i].y);
```

```
            if (direction(p[0], p[1], p[i]) != 0) {//用其他的点和初始化两个点构成3点，从而构造平行四边形，判断其面积是否为0来判断是否共线。
                if (!flag) {
                    puts("how?");
                    flag = true;
                }
            }
        }
        if(!flag){
            puts("boo how! boo how!");
        }
    }
}
```

- 写题解的时候发现每次判断之后，原有点不会再使用，那不如只用3点来存相应向量，减少空间复杂度。

```
Point p1,p2,p3;
int main() {
    int t;
    scanf("%d", &t);
    while (t--) {
        bool flag = false;//判断是否不共线，初始情况判断是共线的。
        int n;
        scanf("%d", &n);
        scanf("%lld%lld", &p1.x, &p1.y);
        scanf("%lld%lld", &p2.x, &p2.y);
        for (int i = 2; i < n; i++) {
            scanf("%lld%lld", &p3.x, &p3.y);
            if (direction(p1, p2, p3) != 0) {//等于0说明平行四边形面积为0，三点必然共线
                flag = true;
                //break;此处不能break，否则导致输入输出问题。
            }
        }
        if(!flag){
            puts("boo how! boo how!");
        }else
            puts("how?");
    }
}
```

时间复杂度

该算法时间复杂度特别直观。只有循环的$O(n)$。

核心代码

```
#include <bits/stdc++.h>
#define LL long long
const int inf_int = 0x3f3f3f3f;
const LL inf_ll = 0x3f3f3f3f3f3f3f3f;

using namespace std;
```

```
struct Point {
    LL x, y;
};

LL direction(Point pi, Point pj, Point pk) {//返回叉积大小,pipk X pipj向量
    return (pk.x - pi.x) * (pj.y - pi.y) - (pj.x - pi.x) * (pk.y - pi.y);
}

Point p1,p2,p3;
int main() {
    int t;
    scanf("%d", &t);
    while (t--) {
        bool flag = false;//判断是否不共线，初始情况判断是共线的。
        int n;
        scanf("%d", &n);
        scanf("%lld%lld", &p1.x, &p1.y);
        scanf("%lld%lld", &p2.x, &p2.y);
        for (int i = 2; i < n; i++) {
            scanf("%lld%lld", &p3.x, &p3.y);
            if (!flag&&direction(p1, p2, p3) != 0) {//等于0说明平行四边形面积为0，三点
必然共线
                flag = true;
                //break;此处不能break，否则导致输入输出问题。
            }
        }
        if(!flag){
            puts("boo how! boo how!");
        }else
            puts("how?");
    }
}
```