**Magic Territory**

**Brief Problem Statement**

รับตำแหน่งมุมของสามเหลี่ยมในระบบพิกัดฉาก 2 มิติ (มีโอกาสทับกันเป็นเส้นตรงได้) ทั้งสิ้น N สามเหลี่ยม ให้นับจำนวนสามเหลี่ยมที่เส้นขอบ(ด้านของสามเหลี่ยม) ไม่ตัดหรือทับซ้อนกับสามเหลี่ยมอื่นๆ

**Solution**

หลักการของข้อนี้คือสร้างสมการเส้นตรง แทนด้านทั้งสามของสามเหลี่ยม แล้วพิจารณาว่าสามเหลี่ยม  *กับ ตัดกันไหม ให้พิจารณาว่าประพจน์ข้างล่างนี้เป็นจริงหรือไม่*

ซึ่งการที่ isIntersect นั้นจะเป็นจริงได้ มีอยู่ 2 กรณีดั้งนี้

1.) เส้นทั้งสองไม่ขนานกันและจุดตัดของเส้นทั้งสองอยู่ภายในเส้น(ระหว่างจุดเริ่มกับจุดจบ) ทั้งสองเส้น

2.) เส้นทั้งสองขนานกันและทับซ้อนกันพอดี

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  struct Point {      double x, y;      Point() : x(0), y(0) {}      Point(double x, double y) : x(x), y(y) {}  };  struct Line {      double a, b, c;  // ax + by - c = 0      Point st, ed;      Line() : a(0), b(0), c(0), st(Point()), ed(Point()) {}      Line(double a, double b, double c) : a(a), b(b), c(c), st(Point()), ed(Point()) {}      Line(double x1, double y1, double x2, double y2) {          a = y2 - y1;          b = x1 - x2;          c = x1 \* y2 - x2 \* y1;          st = Point(x1, y1);          ed = Point(x2, y2);      }      Line(Point p1, Point p2) {          a = p2.y - p1.y;          b = p1.x - p2.x;          c = p1.x \* p2.y - p2.x \* p1.y;          st = p1;          ed = p2;      }      pair<int, Point> operator&(const Line L) const {          if (a \* L.b == b \* L.a) {              if (a \* L.c == c \* L.a && b \* L.c == c \* L.b) {                  return {2, Point{0, 0}};              }              return {0, Point{0, 0}};          }          double d = a \* L.b - b \* L.a;          return {1, Point{(c \* L.b - b \* L.c) / d, (a \* L.c - c \* L.a) / d}};      }      bool isContain(Point p) {          if (st.x > ed.x) swap(st, ed);          if (st.x <= p.x && p.x <= ed.x) {              if (st.y > ed.y) swap(st, ed);              if (st.y <= p.y && p.y <= ed.y) {                  return true;              }          }          return false;      }      bool isIntersect(Line L) {          pair<int, Point> p = (\*this) & L;          if (p.first == 0) return false;          if (p.first == 2 && (isContain(L.st) || isContain(L.ed) || L.isContain(st) || L.isContain(ed))) return true;          return isContain(p.second) && L.isContain(p.second);      }  };  struct Poly {      Point p1, p2, p3;      Line l1, l2, l3;      Poly() : p1(Point()), p2(Point()), p3(Point()), l1(Line()), l2(Line()), l3(Line()) {}      Poly(Point p1, Point p2, Point p3) : p1(p1), p2(p2), p3(p3), l1(Line(p1, p2)), l2(Line(p2, p3)), l3(Line(p3, p1)) {}      Poly(double x1, double y1, double x2, double y2, double x3, double y3) : p1(Point(x1, y1)), p2(Point(x2, y2)), p3(Point(x3, y3)), l1(Line(x1, y1, x2, y2)), l2(Line(x2, y2, x3, y3)), l3(Line(x3, y3, x1, y1)) {}      bool isIntersect(Poly P) {          return l1.isIntersect(P.l1) || l1.isIntersect(P.l2) || l1.isIntersect(P.l3) ||                 l2.isIntersect(P.l1) || l2.isIntersect(P.l2) || l2.isIntersect(P.l3) ||                 l3.isIntersect(P.l1) || l3.isIntersect(P.l2) || l3.isIntersect(P.l3);      }  };  Poly p[2020];  int ch[2020];  int main() {      int n, i, j;      double x1, y1, x2, y2, x3, y3;      cin >> n;      for (i = 0; i < n; i++) {          cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3;          p[i] = Poly(x1, y1, x2, y2, x3, y3);      }      for (i = 0; i < n; i++) {          for (j = i + 1; j < n; j++) {              if (p[i].isIntersect(p[j])) {                  ch[i] = ch[j] = 1;              }          }      }      int ans = 0;      for (i = 0; i < n; i++) {          if (!ch[i]) ans++;      }      cout << ans << "\n";  } |