USACO FEB12 Problem 'symmetry' Analysis

by Brian Dean

Several solutions proved to be effective for this problem. For starters, one can show that there are at most 4 lines of symmetry (since more lines of symmetry would require real-valued input with irrational coordiantes). However, it is not immediately clear how to get too much algorithmic leverage from this observation. One of the simplest solutions that turns out to obtain full marks is to simply try every possible line of symmetry (that is every possible perpendicular bisector between a pair of points, and every single line connecting a pair of points). For each of these, we check all points to see if they have a mirror image on the other side of the line, aborting if not. Although the worst-case running time of this approach is $O(N^3)$, it runs quickly in practice since most prospective lines of symmetry can be filtered out after checking just a few points.

For a simple $O(N^2)$ solution, fix some arbitrary point p. For all other points x, consider the line of symmetry folding p onto x (that is, the perpendicular bisector of p and x). We check all these lines of symmetry to see if they are valid, and this will find all lines of symmetry except those passing through p. Now do the same again, only from some other arbitrarily chosen point q; this will find all lines of symmetry except those passing through q. Finally, just check the line through p and q.

Mark Gordon's solution is below; it tries all possible lines of symmetry:

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <complex>
#include <cmath>
#include <map>
#include <set>
#include <cstdio>
#include <cassert>

using namespace std;

#define nabs(x) ((x) < 0 ? -(x) : (x))
typedef int num;

typedef complex<num> point;

num gcd(num a, num b) {
  a = nabs(a); b = nabs(b);
  return a ? gcd(b % a, a) : b;
}

struct line {
  line() : A(1), B(0), C(0) {}
  line(num A, num B, num C) : A(A), B(B), C(C) {}

  bool operator<(const line& x) const {
```

```
    return make_pair(make_pair(A, B), C) <
           make_pair(make_pair(x.A, x.B), x.C);
  }

  bool operator==(const line& x) const {
    return make_pair(make_pair(A, B), C) ==
           make_pair(make_pair(x.A, x.B), x.C);
  }

  line canon() {
    num g = gcd(A, gcd(B, C));
    if(make_pair(make_pair(A, B), C) <
       make_pair(make_pair(num(0), num(0)), num(0))) {
      g *= -1;
    }
    return line(A / g, B / g, C / g);
  }

  num A, B, C;
};

line get_line(point A, point B) {
  line ln(B.imag() - A.imag(), A.real() - B.real(), 0);
  return line(ln.A, ln.B, ln.A * A.real() + ln.B * A.imag());
}

line bisector(point A, point B) {
  line ln(B.real() - A.real(), B.imag() - A.imag(), 0);
  return line(ln.A * 2, ln.B * 2,
              ln.A * (A.real() + B.real()) + ln.B * (A.imag() + B.imag()));
}

point intersect(line A, line B, bool* div) {
  num det = A.A * B.B - A.B * B.A;
  if(nabs(det) <= 0) return point(0, 0);
  num xn = B.B * A.C - A.B * B.C;
  num yn = -B.A * A.C + A.A * B.C;
  if(div) *div = xn % det == 0 && yn % det == 0;
  return point(xn / det, yn / det);
}

#define MAXN 1000

point X[MAXN];

int main() {
  freopen("symmetry.in", "r", stdin);
  freopen("symmetry.out", "w", stdout);

  int N; cin >> N;
  for(int i = 0; i < N; i++) {
    cin >> X[i].real() >> X[i].imag();
  }

  /* I/O checks. */
  set<pair<num, num> > st;
  assert(2 <= N && N <= 1000);
```

```
  for(int i = 0; i < N; i++) {
    assert(-10000 <= X[i].real() && X[i].real() <= 10000);
    assert(-10000 <= X[i].imag() && X[i].imag() <= 10000);
    bool res = st.insert(make_pair(X[i].real(), X[i].imag())).second;
    assert(res);
  }

  vector<line> lns;
  for(int i = 0; i < 2; i++) {
    for(int j = i + 1; j < N; j++) {
      lns.push_back(bisector(X[i], X[j]).canon());
      lns.push_back(get_line(X[i], X[j]).canon());
    }
  }
  sort(lns.begin(), lns.end());
  lns.resize(unique(lns.begin(), lns.end()) - lns.begin());

  int res = 0;
  for(int i = 0; i < lns.size(); i++) {
    bool ok = true;
    for(int j = 0; j < N && ok; j++) {
      line la(lns[i].A, lns[i].B,
              2 * lns[i].C - lns[i].A * X[j].real() - lns[i].B *
X[j].imag());
      line lb(-lns[i].B, lns[i].A,
              -lns[i].B * X[j].real() + lns[i].A * X[j].imag());
      point pt = intersect(la, lb, &ok);
      ok = ok && st.find(make_pair(pt.real(), pt.imag())) != st.end();
    }
    res += ok;
  }
  cout << res << endl;
}
```