USACO OPEN14 Problem 'optics' Analysis

by Mark Gordon

In this problem a laser originating at the origin is shining north. There are several mirrors that may intersect the laser's path and we are asked in how many positions can a mirror be placed so that the laser hits a barn located at (bx, by).

Immediately it is clear that wherever we place a mirror it must be on that path that the laser would take if there were no mirror there at all. Otherwise, because we are told the laser does not initially hit the barn, the laser's path would not change at all and continue to not hit the barn. This exact same reasoning applies in reverse as well; if we shine new light beam from the barn in the direction that the laser hits then the new light beam will follow the exact same path as the laser except in the reverse.

This means that any mirror must be placed somewhere on the laser's path and somewhere on the path of a light beam shone in any of the four directions of the barn. Fortunately, this condition is also sufficient for it to be possible to place a mirror in a given position. The laser and light beam from the barn must meet at right angles (otherwise the laser would hit the barn) so turning the laser one of left or right must take the laser to the barn.

A final corner case, illustrated by the example, is when a light beam from the barn forms a cycle, terminating back at the barn. In this case a mirror placed in either the / or \ orientation at an intersection with the laser will both direct the laser into the barn so care must be taken to avoid this.

So this problem now reduces to counting the number of intersections of a set of vertical and horizontal lines. This can be accomplished by processing the lines from left to right. When a horizontal line starts we add its y value to a tree (a segment tree or Fenwick tree, for example), when it ends we remove it from the tree. Then when we reach a vertical line we can simply query the number of values in a given range.

Below is my implementation of the above

```
#include <iostream>
#include <vector>
#include <map>
#include <cstring>
#include <algorithm>
#include <cstdio>

using namespace std;

#define MAXN (1 << 17)
#define MAXVAL 1000000000

int dx[] = {0, 1, 0, -1};
int dy[] = {1, 0, -1, 0};

map<int, vector<pair<int, char> > > objx;
map<int, vector<pair<int, char> > > objy;

pair<pair<int, int>, char> getnext(int x, int y, int dir) {
  bool vmove = dir % 2 == 0;
  int a = vmove ? x : y;
  int b = vmove ? y : x;
  int db = vmove ? dy[dir] : dx[dir];
  vector<pair<int, char> >& objs = (vmove ? objx : objy)[a];

  int id = lower_bound(objs.begin(), objs.end(), make_pair(b, (char)0))
```

```
                - objs.begin();

  id += db;
  char ch = '?';
  if(id < 0) {
    b = -(MAXVAL + 1);
  } else if(id == objs.size()) {
    b = MAXVAL + 1;
  } else {
    b = objs[id].first;
    ch = objs[id].second;
  }
  return make_pair(vmove ? make_pair(a, b) : make_pair(b, a), ch);
}

vector<pair<int, int> > getpath(int x, int y, int dir) {
  pair<int, int> pos(x, y);
  vector<pair<int, int> > path(1, pos);
  for(;;) {
    pair<pair<int, int>, char> res = getnext(pos.first, pos.second,
dir);
    pos = res.first;
    path.push_back(pos);

    if(res.second == '/') {
      dir = (dir + (dir % 2 != 0 ? 3 : 1)) % 4;
    } else if(res.second == '\\') {
      dir = (dir + (dir % 2 == 0 ? 3 : 1)) % 4;
    } else {
      break;
    }
  }
  return path;
}

vector<pair<int, pair<int, int> > >
getverts(vector<pair<int, int> >& path) {
  vector<pair<int, pair<int, int> > > ret;
  for(int i = 0; i + 1 < path.size(); i++) {
    if(path[i].first == path[i + 1].first) {
      ret.push_back(make_pair(path[i].first,
                      make_pair(path[i].second, path[i + 1].second)));
      if(ret.back().second.second < ret.back().second.first) {
        swap(ret.back().second.first, ret.back().second.second);
      }
    }
  }
  return ret;
}

vector<pair<int, pair<int, int> > >
gethorz(vector<pair<int, int> >& path) {
  vector<pair<int, pair<int, int> > > ret;
  for(int i = 0; i + 1 < path.size(); i++) {
    if(path[i].second == path[i + 1].second) {
      ret.push_back(make_pair(path[i].second,
                      make_pair(path[i].first, path[i + 1].first)));
      if(ret.back().second.second < ret.back().second.first) {
        swap(ret.back().second.first, ret.back().second.second);
      }
    }
  }
  return ret;
```

```
}

int BT[MAXN];

/* Logically executes array[x] += v. */
void bit_add(int x, int v) {
  for(int i = x | MAXN; i < (MAXN << 1); i += i & -i) {
    BT[i ^ MAXN] += v;
  }
}

/* Returns the sum of array[i] for 0 <= i < x */
int bit_get(int x) {
  int ret = 0;
  for(int i = x - 1; x != 0; i &= i - 1) {
    ret += BT[i];
    if(!i) break;
  }
  return ret;
}

int countints(vector<pair<int, pair<int, int> > > vs,
              vector<pair<int, pair<int, int> > > hs) {
  /* Start with a coordinate compression of y values. */
  vector<int> ys;
  for(int i = 0; i < vs.size(); i++) {
    ys.push_back(vs[i].second.first);
    ys.push_back(vs[i].second.second);
  }
  for(int i = 0; i < hs.size(); i++) {
    ys.push_back(hs[i].first);
  }
  sort(ys.begin(), ys.end());
  ys.resize(unique(ys.begin(), ys.end()) - ys.begin());
  for(int i = 0; i < vs.size(); i++) {
    vs[i].second.first = lower_bound(ys.begin(), ys.end(),
                                     vs[i].second.first) - ys.begin();
    vs[i].second.second = lower_bound(ys.begin(), ys.end(),
                                      vs[i].second.second) - ys.begin();
  }
  for(int i = 0; i < hs.size(); i++) {
    hs[i].first = lower_bound(ys.begin(), ys.end(), hs[i].first) - ys.begin();
  }

  /* Sort vertical intervals by x, create event list. */
  sort(vs.begin(), vs.end());
  vector<pair<pair<int, int>, int> > events;
  for(int i = 0; i < hs.size(); i++) {
    events.push_back(make_pair(make_pair(hs[i].second.first, hs[i].first), 1));
    events.push_back(make_pair(make_pair(hs[i].second.second,
                                         hs[i].first), -1));
  }
  sort(events.begin(), events.end());

  /* Finally, count the intersections using a Fenwick tree. */
  int result = 0;
  memset(BT, 0, sizeof(BT));
  for(int i = 0, j = 0; i < events.size(); i++) {
    int x = events[i].first.first;
    for(; j < vs.size() && vs[j].first < x; j++) {
      result += bit_get(vs[j].second.second) - bit_get(vs[j].second.first + 1);
    }
    bit_add(events[i].first.second, events[i].second);
```

```
  }
  return result;
}

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

  int N, bx, by;
  cin >> N >> bx >> by;

  objx[0].push_back(make_pair(0, 'S'));
  objy[0].push_back(make_pair(0, 'S'));
  objx[bx].push_back(make_pair(by, 'B'));
  objy[by].push_back(make_pair(bx, 'B'));
  for(int i = 0; i < N; i++) {
    int x, y;
    string mr;
    cin >> x >> y >> mr;

    objx[x].push_back(make_pair(y, mr[0]));
    objy[y].push_back(make_pair(x, mr[0]));
  }
  for(map<int, vector<pair<int, char> > >::iterator it =
objx.begin();
      it != objx.end(); ++it) {
    sort(it->second.begin(), it->second.end());
  }
  for(map<int, vector<pair<int, char> > >::iterator it =
objy.begin();
      it != objy.end(); ++it) {
    sort(it->second.begin(), it->second.end());
  }

  int result = 0;
  vector<pair<int, int> > plaser = getpath(0, 0, 0);
  for(int i = 0; i < 4; i++) {
    vector<pair<int, int> > pbarn = getpath(bx, by, i);

    int res = countints(getverts(plaser), gethorz(pbarn)) +
              countints(getverts(pbarn), gethorz(plaser));
    if(pbarn[0] == pbarn.back()) {
      result += res;
    } else {
      result += 2 * res;
    }
  }
  cout << result / 2 << endl;

  return 0;
}
```