Code Jam 2008 - World Finals

Analysis: Ping Pong Balls

Special Cases

The small dataset can be solved by a brute force. With algorithms like breadth first search (BFS), one can find all the points that are triggered.

Another case we can use BFS is when the two displacement vectors are collinear. In this case all the points we need to consider are in a line, and there will be no more than 1000000 points on any line.

General Case

Interestingly, for the general case, where the two vectors are not collinear, one can use a shorter program with a simpler data structure. Here we introduce a solution that only involves vector additions. This is not the most efficient solution, but it is good enough for our purposes.

Note: This is not the first time we've encountered a 2-dimensional integer lattice where it is better to change the coordinate system. See the similar picture we have in the analysis for Problem D in online round 3.

Let us use [x, y] to denote the points in the ordinary coordinate system, and (a, b) to denote the points in the new system. Suppose the two displacement vectors are V1 = [δ_{x1}, δ_{y1}], V2 = [δ_{x2}, δ_{y2}], Suppose that the first ball hits position P = [x_0, y_0]. The points in the new system is given by

$$(a, b) := P + aV_1 + bV_2 = [x_0 + a\delta_{x1} + b\delta_{x2}, y_0 + a\delta_{y1} + b\delta_{y2}]. \quad (*)$$

The problem is to figure out, from (0, 0), how many points will be hit by repeatedly adding (1, 0) or (0, 1).

It is not hard to prove, using (*) and the fact that the two vectors are not collinear, that for any points (a, b) inside the room, the numbers a and b are bounded by some quantity **Q**, where **Q** is the size of the room times the maximum value of the δs. With the limits in our problem, **Q** ≤ 2×10^7.

For any fixed **a**, it is easy to see that there is a contiguous sequence of numbers **b** such that (a, b) is hit. In other words, there are numbers b_a and b'_a such that (a, b) is hit if and only if $b_a \le b \le b'_a$. This is because, if we let b_a and b'_a be the minimum and maximum **b** that get hit, respectively, then, by convexity, (a, b) is inside the room for all **b** between b_a and b'_a. Once we hit (a, b_a), we keep adding (0, 1) and get all the other points. It is clear that (a, b'_a) is the last point that stays in the room, and (a, $b'_a + 1$) is outside the room.

It is good enough if we can iterate over **a** = 0, 1, 2, ..., (at most **Q**), and quickly compute b_a and b'_a for each **a**. Notice that, by definition, (a, $b_a - 1$) is not hit. In order to hit (a, b_a), the last step must be from (a - 1, b_a). So

$$b_{a-1} \le b_a \le b'_{a-1}.$$

To find b_a, we can simply start from b_{a-1} and keep increasing until we hit a point inside the room. Note that for a single **a** this might take a long time. However, the b_a's are monotone, so the cost never exceeds **Q**.

Once we find b_a, we can use a binary search to find b'_a. Although a binary search is simple and quick enough, another approach seems dumber but actually works faster. Similarly to the way we get b_a from b_{a-1}, we may also get b'_a from b'_{a-1}. Here the b'_a are not monotone, so we need to try both increasing and decreasing. Nevertheless, based on the rectangular shape of the room, the direction will not change more than once. The total cost is still O(Q).

Sample code from the judges for the non-collinear case:

```
int T, W, H;
int x, y, dx1, dx2, dy1, dy2;

bool inside(int a, int b) {
  int xx = x + a*dx1 + b*dx2;
  int yy = y + a*dy1 + b*dy2;
  if(xx<0 || xx>=W) return false;
  if(yy<0 || yy>=H) return false;
  return true;
}

long long play() {
  long long ans=0;
  int b1=0; int b2=1000001;
  for(int a=0; ; a++) {
    while(!inside(a, b1)) {
      b1++;
      if(b1>b2) return ans;
    }
    if(inside(a, b2)) {
      while(inside(a, b2)) b2++;
      b2--;
    } else {
      while(!inside(a, b2)) b2--;
    }
    ans+=(b2-b1+1);
  }
  return 0;
}
```