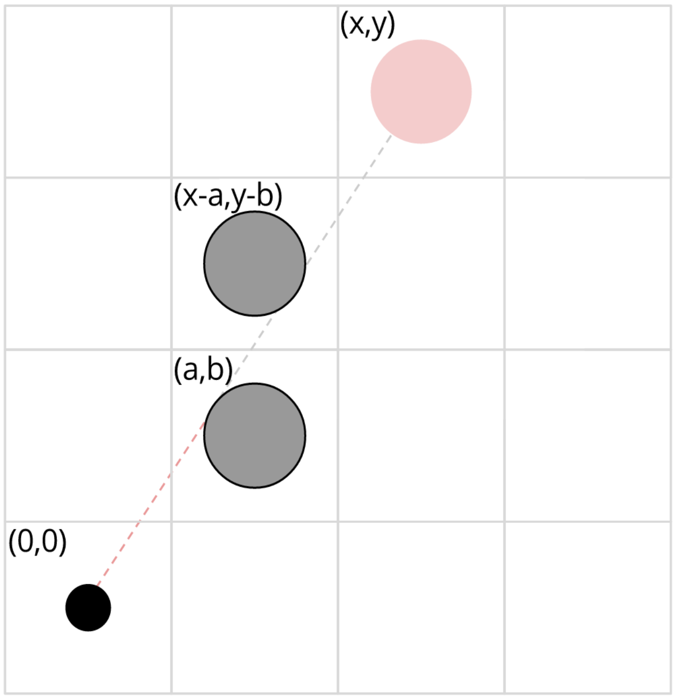
Code Jam 2016 - World Finals

Analysis: Gallery of Pillars

Gallery of Pillars: Analysis

A convenient way to look at this problem is to set up a coordinate system with the viewer at the origin and the center of every pillar at other coordinates (x,y) with x and y non-negative integers less than **N**.

Let us focus on a given pillar centered at (x,y). Suppose some other pillar centered at (a,b) intersects the segment [(0,0),(x,y)]. Then, (a,b) blocks at least half the view of (x,y), from the segment towards one of the sides. Then, if we consider the pillar centered at (x - a, y - b), that is, the one symmetric across the midpoint of the segment [(0,0),(x,y)], the distance from that pillar to the segment is the same, so it also covers the segment; it's located on the other side of the segment from (a,b), and so it blocks the other half of the view. It follows that the pillar centered at (x,y) is visible from the origin if and only if the segment [(0,0),(x,y)] does not intersect any other pillar. The following picture illustrates the situation.



Assume the pillar centered at (a,b) intersects the segment [(0,0),(x,y)]. Clearly, a ≤ x and b ≤ y. Moreover, the square of the distance between the point and the segment can be expressed as $d^2 = (ay - bx)^2 / (x^2 + y^2)$.
Since the pillar intersects the segment, $d^2 \times 10^6 \le \mathbf{R}$.

From this expression, we can tell in constant time whether a given pillar blocks the view of some other pillar. This immediately gives an $O(\mathbf{N}^4)$ solution to the problem: for each pillar, check every other pillar to see whether it blocks it. This is just shy of fast enough for the Small, but there is a simple improvement: instead of checking every possible (a,b) for each (x,y), it's enough, for each possible a between 0 and x - 1, to try only b = a * y / x, rounded up and down (that is, the two points with that a coordinate that are clearly closest to the segment). This makes the complexity O($\mathbf{N}^3$), which is definitely fast enough to solve the Small dataset.

The Large dataset, as usual, requires a bit more work. Let us pick up where we left off: our expression for the squared distance
$d^2 = (ay - bx)^2 / (x^2 + y^2)$.
If x and y are not relatively prime, choosing a = x / gcd(x,y) and b = y / gcd(x,y) yields a pair (a,b) that fullfills every condition to cover (x,y). This is to be expected, as (a,b) clearly covers all pillars (ka,kb) for each k > 1. If x and y are not relatively prime, then there exist positive a and b such that |ay - bx| = 1. Since |ay - bx| is an integer, and it can only be 0 within our constraints when x and y are not relatively prime, choosing a pair that makes it 1 yields the closest distance. Thus, a pillar (x,y) is visible if and only if x and y are relatively prime and $10^6 / (x^2 + y^2) < \mathbf{R}$. That is, we need to count the pairs of relatively prime integers within a circle of radius 10^6 / **R** and a square of size **N** by **N**.

If $\mathbf{N} \geq 10^6 / \mathbf{R}$, then the square contains the circle, so let us assume further that $\mathbf{N} \leq 10^6 / \mathbf{R} \leq 10^6$. We can count all points inside the area, then subtract the multiples of 2, 3, 5, ..., etc, then add the multiples of 2 × 3 = 6, 2 × 5 = 10, 3 × 5 = 15, etc, that were subtracted twice, and keep doing the inclusion-exclusion argument for each possible divisor k between 1 and **N**. The count for points that are multiples of a given k should be multiplied by the Möbius function of k. Calculating that function for each number between 1 and **N** can be done efficiently with a slight modification of the sieve of Eratosthenes, which runs in linear time. For each k, we count by iterating the possible values of x, k, 2k, 3k, etc, and for each one we find the range of possible y values. We can do that by taking the minimum of **N** and the square root of $(10^6 / \mathbf{R})^2 - x^2$, or by binary search, or by linear search. Notice that the maximum value of y decreases from one value of x to the next, so if we pick up at the value considered for the previous x, the linear search will only take linear time overall (constant amortized time per each value of x). Linear and binary search are guaranteed to give a precise number, but using the square root also works because we are dealing with fairly small values. For the most efficient versions, the time complexity of the algorithm to get the count for a given value of k is O(**N** / k). Since the summation of 1 / k up to k = **N** can be approximated by log **N**, the overall complexity of the algorithm is O(**N** log **N**). Using binary search to find the range of values for y only adds a second log factor, yielding an overall O($\mathbf{N} \log^2 \mathbf{N}$). Both are efficient enough for **N** up to 10^6.