PATRIK

The large limit on N will make quadratic solutions time out.

To start with, suppose the heights of all people are distinct. Imagine going through the line in order. Observe any person A waiting in line. If, after person A, we encounter a taller person B, then person A surely can't see anyone after person B.

Because of this, when going through the line, we can maintain a stack of "open subproblems", where an open subproblem is a person already encountered in the line, but who still has a possibility of seeing someone whom we haven't yet encountered. It should be obvious that the subproblems on the stack are sorted in descending order of height (topmost subproblem = shortest person).

When we encounter a new person, that person can see everyone on the stack that is shorter, and also takes those people off the stack (closing their subproblems). If the stack isn't empty after this, the person can also see the first person remaining on the stack. The person then enters the stack (there is a possibility that they will see someone later in line).

Even though the solution has two nested loops, the overall complexity is O(N), because every person enters and leaves the stack exactly once, and each iteration of the inner loop pops one element off the stack.

To complete the solution, we need to consider the effect of persons equally tall. One way is to have the stack hold ordered pairs (height, number of people) and maintain it accordingly.

POLICIJA

A single depth-first search collects all the information needed to answer the queries. While searching, we store the following numbers for each vertex:

- Discovery time – discrete time index at which we started processing the vertex
- Finishing time – time index when we finished processing the vertex
- Depth – the depth of the vertex in the DFS tree
- lowlink – the vertex with the least discovery time, that is reachable from the current vertex or from its descendants (via a single back edge)

These numbers allow us to construct two useful functions:

- is_descendant(A, B) – is vertex A in the subtree rooted at B
- find_related_child(A, B) – when A is in vertex B's subtree, find the immediate child of B such that A is a descendant of that child (in other words, if B has multiple children, find which of those subtrees A is in)

With some case analysis, it is possible to answer the queries using the described functions. See the sample code for details.

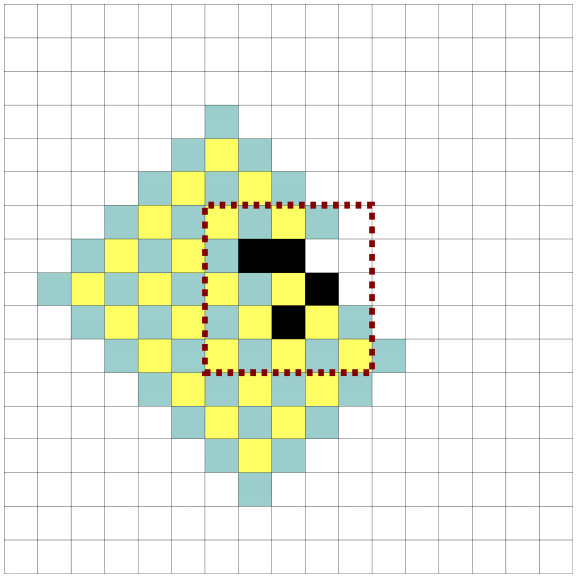
For an in-depth discussion of depth-first search and its applications, see "Introduction to Algorithms" by Cormen et al. This problem is closely related to finding articulation points and bridges in a graph.

SABOR

Upon closer inspection, the task asks to count, among all cells at most K steps from the origin, the number of cells with even distance and the number of cells with odd distance. One might say that we're colouring the cells with two colours.

The colour of a cell is uniquely determined by its coordinates, since each step changes the parity of the expression x+y. Because of this, two adjacent cells are necessarily coloured differently. Additionally, the distances of two adjacent cells differ by exactly 1.

The image shows the second sample test, with a slightly larger number of steps S. Observe the smallest rectangle surrounding all obstacles and the origin, and expand it one cell in all four directions:



The absolute values of the obstacles' coordinates are at most 1000 so the distances of all cells inside the rectangle can be found with breadth-first search.

For example, consider a cell is on the left edge of the rectangle (but inside it), distance some D steps from the origin. Then exactly K–D cells to its left (outside the rectangle) will be coloured, and simple formulas will give the number of cells of one and the other colour in O(1). We proceed identically for cells on the other three edges of the rectangle.

We still need to count the cells off the corners; for example, the cells above and to the left of the yellow cell in the upper-left corner of the rectangle. A triangular pattern is obvious and there are formulas (see sample code) for this case too. But since S is on the order of millions, doing it in O(S) for each corner instead of O(1) will do, too.