HRASTOVI

We simplify the original query in the task with 4 queries, one for each side of the rectangle. Each of these 4 queries can be solved in the same way independently. Let us solve the vertical left side $(x_1, y_1) – (x_1, y_2)$ for example.

We start by sorting all oaks by their x coordinate first, breaking ties by sorting on y second. It is easy to see that all points on the vertical left side of the rectangle will be in sequence. Since the sequence is sorted, using binary search we can easily determine where points (x_1, y_1) and (x_1, y_2) would be in the sequence. The number of items between those two points in the sequence is then the number of oaks that intersect that vertical side. Care should be taken to account for cases where oaks are in points (x_1, y_1) and (x_1, y_2).

The other vertical side is solved by using corresponding points. Horizontal sides are solved by sorting by y first using x as break.

Time complexity is O(N log N + P log N).

KOLO

We use Eratosten's sieve to find the first K prime numbers.

Now we mark the square with 0 and circles with numbers 1 to N – 1. Since it would be impossible to simulate the entire game within the time limit, we first determine an algorithm that will help us find A's final position. Let's assume that currently A is on position x and the current prime is p:

- If x = 0, then A is in the square so x becomes N mod p.
- Else, if x ≠ 0, than we can determine how many times the person in the square changes place with x:
    - If x <= p mod (N−1) then x becomes (x − 1) mod N
    - Else x becomes (x – p div (N−1)) mod N

Where mod is the modulus operator and div is integer division. Now we need to solve the other way so that we may answer the question "If A is on X who is on X-1 (or X+1)?"

Let's assume that we know position x **after** the prime p was played.

- If x = p mod N, than x was at 0 before prime p.
- Else, if x ≠ p mod N:
    - x becomes (x + p div (N−1)) mod N
    - If x <= p mod (N−1) then x becomes (x + 1) mod N

The time complexity is O(K).

LOZA

Note that the only way we can influence the total number of characters is by adding or removing '-' characters on branched links. We are interested in the smallest possible number of characters. Note that this means finding the smallest possible number of characters on any link individually, since you can never reduce one link by adding characters to another.

Given this observation, it makes sense to start creating the tree bottom up. So that when finding the minimal width of a branch, we already know her subtrees.

For each node in tree already solved, we mark it's relative x coordinate of his '+' sign. Also we maintain a list of pairs of x coordinate of the leftmost and rightmost character for each tree depth, relative to the x coordinate of '+' character.

Each node is processed as follows:

1. If the node has no children, than we arbitrary select the coordinate of '+'. We create a new list containing only the leftmost and rightmost edge of this nodes rectangle.
2. If the node has one child, the '+' coordinate is equal to the child's '+' coordinate. The list is inherited from the child, adding the leftmost and rightmost edge of the new node's rectangle.
3. If the node has two children, we need to determine how much to space them so that they may be drawn. On each depth we observe the rightmost coordinate of the left subtree, and the leftmost coordinate of the right subtree. The difference of these coordinates determines the relative displacement of the left subtree to left, or the right subtree to the right. We select the smallest of these subtrees and move it. After the move, the '+' character coordinate is the center point between the children and the list can be inherited from the longer subtree, adding the current node.

Time complexity is O(n log n)

ROBOTI

Let us first solve a simple problem: what if there was only one robot, and the answer to the move function was 1 if the robot moved and 0 if he didn't. The solution is a two phase process:

1. Locating the precise location of the robot
2. Moving it to the selected extraction point

We can determine the absolute location of the robot using DFS. We create a relative map of the warehouse starting in (0, 0) and visiting the fields in DFS order. Once the DFS is finished, we have a relative map of the warehouse with the robot in known coordinates. We now overlay this map with the map given in the input and obtain the absolute location of the robot. After this moving the robot to the extraction point is a straightforward task.

What happens if there is another robot, and the function returns their distance? First, note that the function can still be interpreted in success/fail terms. If the distance changes, the robot moved, if not, it didn't. However, the second problem is that the robot can, and most will, hit the other robot. This can be identified.

First of all, the robots can only hit each other if their distance is 1. Now, if we have relative distance one, and moving the robot fails, we can move the other robot in any direction (as long as the move is successful) and try the first robot again. If the robot moves this time, we know now that we have found the second robot.

At this point there are a number of ways you may proceed. Since now we have the relative location of both robots, we can simply continue to explore the map with the first robot until we can safely overlay the maps and than move both robots to the extraction point. Another good way is to note that if the robots hit each other not moving can be treated as exchanging places and numbers.