Artemis

SOLUTION

Observation: Let f(x, y) be the number of trees below and to the left of (x, y). Then the number of the trees in the rectangle bounded by t_1 and t_2 is

$f(t_1.x, t_1.y) + f(t_2.x, t_2.y) - f(t_1.x, t_2.y) - f(t_1.y, t_2.x) + 1$

if t_1 lies below and to the left of t_2 (or vice versa), and a similar formula if not.

1. Trivial algorithm. Loop over all rectangles, and loop over all trees to count those inside the rectangle.

   $O(n^3)$

2. Use dynamic programming to compute $f(t_1.x, t_2.y)$ for every t_1, t_2. Then evaluate all rectangles using the formulae.

   $O(n^2)$, but also $O(n^2)$ memory

3. Place an outer loop t over the trees, representing one corner of a potential rectangle. To evaluate rectangles with corners at t, one only needs f(t.x, *) and f(*, t.y). These can be computed with DP as in algorithm (2), and requires only linear memory.

   $O(n^2)$

4. Sort the trees from left to right, and then process them in that order. As each new tree (say t_n) is added, it is inserted into a list of current trees that is sorted vertically. From this information one can calculate $f(t.x, t_n.y)$ and $f(t_n.x, t.y)$ for every t to the left of t_n, in linear time. Then one can evaluate all rectangles with one corner at t_n. This ends up being very similar to algorithm (3).

   $O(n^2)$

5. Algorithm (1), but with optimised counting. As a pre-process, associate a bitfield with each tree representing which trees lie below and to the right, and a similar bitfield for trees below and to the left. The trees inside a given rectangle may be found as the binary AND of two bitfields. A fast counting mechanism (such as a 16-bit lookup table) will accelerate counting.

   $O(n^3)$ and $O(n^2)$ memory, but with low constant factors