IPv6

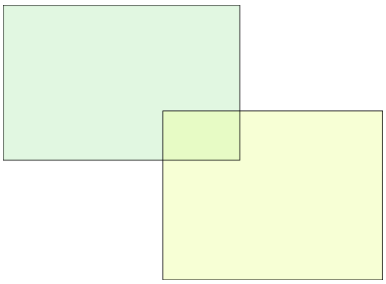
This task requires no special algorithms, just careful string manipulation and a way to handle all the cases as easily as possible. Any correct solution must be able to split the input address into groups (to add leading zeroes), find whether the address is compressed and insert groups of zeroes if so.

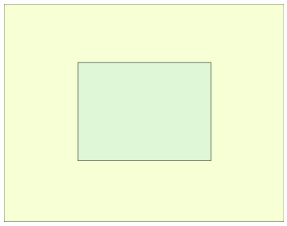
The official source code first checks the input for the substring `::`. If it is present, split the input in two halves, solve them separately and merge the partial outputs, inserting groups of zeroes as needed. Merging is done carefully to correctly handle inputs like `::1` and `1::`.

LOGO

Because we can draw the same segment more than once, two rectangles can be drawn without raising the pencil if any two of their sides intersect or touch.

Two rectangles have intersecting sides if their intersection is non-empty, except in the special case when one rectangle is completely contained in the other. The intersection of two rectangles is non-empty when the intersections of projections on both axes are non-empty; this condition can be easily checked (see official source code). The following two figures illustrate the two cases: in the first case it is possible to draw both rectangles without raising the pencil, but it is not possible in the second case.





If rectangles A and B can be drawn without raising the pencil as well as rectangles B and C, then all three rectangles can be drawn without raising the pencil. Thanks to this transitive property, we can build a graph with rectangles as vertices and edges between rectangles that can be drawn without raising the pencil. We need to find the number of connected components in this graph; the output is then the number of components minus one. Components can be found using depth-first or breadth-first search. The complexity is $O(N^2)$.

A small detail to take care of is the pencil initially being lowered at coordinates (0, 0), but this can be easily fixed by inserting a fake degenerate rectangle (0, 0)-(0, 0) into the set of rectangles to be drawn.

PLINOVOD

The problem can be solved with a recursive (backtracking) greedy algorithm. For every square on the left side of the grid (starting from the topmost) try to push a gas path to the right edge, trying all three possible neighbours, again starting from the topmost. The complete algorithm is:

algorithm push(row, column)

(returns 1 if it succeeds in pushing a path through the given cell, 0 if it fails)

1. If the cell (row, column) is inaccessible, return 0.
2. If the algorithm was already in cell (row, column), return 0. There is either already a path going through this square, or it is impossible to push a path through it (because the algorithm would have found it the previous time it was here).
3. If push(row−1, column+1) = 1, return 1.
4. If push(row, column+1) = 1, return 1.
5. If push(row+1, column+1) = 1, return 1.
6. Return 0.

The algorithm is correct because it is not incorrect. For it to be incorrect, there would have to be an arrangement of gas paths the algorithm finds, but which can be improved because, at some point, paths already pushed interfered with the pushing of new paths. However, because of the order in which the algorithm looks for paths (always top-down), such an improvement is not possible.

The complexity of the algorithm is O(R·C); it will branch from every cell at most once.