**Short description.**

The logic behind the implementation has the starting point in the fact that the sides of the rectangle must be parallel to X and Y axis. Two points forms a line. For two lines to be parallel to each other they must have the same gradient.

**A few mathematical relations (we use gradient/slope = ).**

On X axis all y’s are 0, so the slope is 0. On Y axis all x’s are 0, so we have something/0 which of course, cannot be done, so we will say that Y axis has an infinite slope.

Furthermore, in order to have a line parallel with X axis we need y2-y1 = 0 so any value assigned to any y on the line the same. Similarly, if it is parallel with Y axis any value assigned to x has to be the same.

Now let us presume we have two points A(a,α) and B(b,β) with α, β unknown and a≠b. Considering the above if we want AB to be parallel with X axis we will assign to α and β the same value, γ. Now, we have a rectangle side parallel with X axis. In order to find the other one, we look for another points C(a, α’) and D(b, β’) where, again α’= β’ =γ’≠ γ. And now we have two sides parallel with X axis, but what about Y axis? Well if we choose A(a, γ) and C(a, γ’) you can easily observe they have the same value for x, so AC line is parallel with Y axis, analogously BD. Now, we have two pairs of lines each one parallel to each other and to X axis / Y axis.

**How do we know we obtained a rectangle?**

Well, the conditions for a quadrilateral to be rectangular are: have the opposite sides parallel and congruent and all the angles have 90**°**. This constraints are actually resolved because AB = CD and AC = BD(if we calculate the distance between two points). The parallelism has already been explained, and the angles comes from the following: X axis is **⊥** on Y axis so if d1 is || X axis and d2 is || Y axis than, obviously, d1 **⊥** d2.

Note: a square is just a particular rectangle where all the sides have the same value, not only the opposite, so I considered this fact it in my solution. If you consider it a mistake here is the way I would have probably solved the situation: pick the perpendicular sides and check the length to be different using the well known mathematical formula for calculating the distance between two points.

**What do we really need?**

In the end, we only need those four points, two having the X axis value equal to “a”, the others equal to “b”, that have two common values for the Y axis {γ, γ’} in order to obtain a rectangle parallel with X,Y axis. In other words we need the intersection of each two values from X axis considering their associated values on the Y axis and then we count pairs of two numbers. The result will be the total number of rectangles.

**Where does come from?**

Suppose we have (v1,v2,v3…vn) intersection set of two points. With v1 we can form n-1 pairs, with v2, n-2 pairs and so on up to 1 pair. The total sum: 1+2+3…+n-1. Using the Guass’s sum for first n-1 terms we obtain **.**

**Why using a map and unordered set?**

To increase the efficiency of the algorithm we associate any x value (key) on X axis all it’s possible values from Y axis. Because we have to make an intersection, and we don’t want to do that in O(n\*m), we keep those associated values into sets, because in C++ unordered sets work using a hash table. That means we reduce the search to constant-time complexity, O(1). So the overall complexity of the intersection will depend by the minimum length of the set we are taking values from, let’s say O(n).