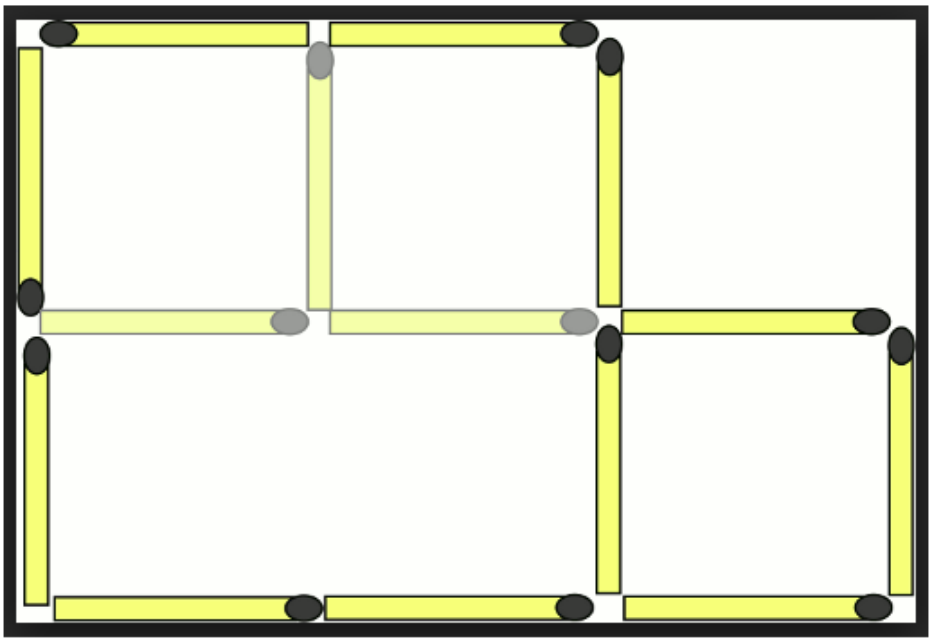
Counting Squares

The motivation for this problem comes from the following puzzle

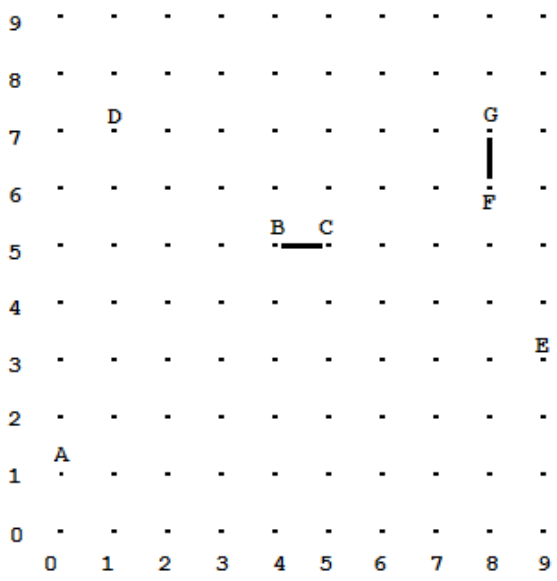
How many squares do you see



Consider the following 10 x 10 grid with 10 rows and 10 columns with 100 Lattice points. The following grid has 7 points labeled (letter A through G) and 2 edges. The seven points are located at the following points:

Note: the upper left corner is at point (9, 0),
the upper right corner is at point (9, 9),
the lower left corner is at point (0, 0),
and the lower right corner is at point (0, 9)

A. is located above point (1, 0)
B. is located above point (5, 4)
C. is located above point (5, 5)
D. is located above point (7, 1)
E. is located above point (3, 9)
F. is located **below** point (6, 8)
G. is located above point (7, 8)

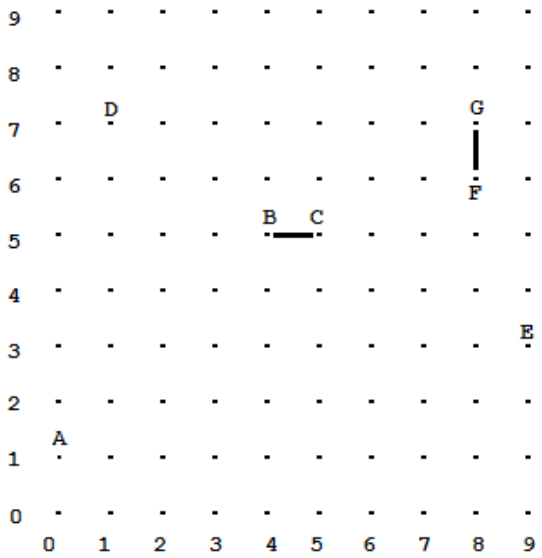


This problem is concerned with finding the number of squares formed by edges in a 10 x 10 grid. The size of a square is defined as the number of Edges that make up one side of the square. A quick reminder that a square has four equal straight sides and four right angles. Example shown at top contains 3 squares of size 1 and one square of size 2.

The 10 x 10 grid is repeated for your convience.

Two edges are shown in the grid: One edge connects points B and Point C and the second edge connects points F and G

Note: All Edges are length 1 and only connect points that are adjacent vertically or adjacent horizontally. No diagonal edges exist in this grid. In addition, Edges go both ways. That is, the edge that connects point B and C is exactly the same as the edge that connects point C and B.



This problem has two completed helper classes, the `Point` class and the `Edge` class. The `Point` class represents a Lattice point in the 10 x 10 grid with accessor methods `getX()` and `getY()` and a correctly implemented `equals` method. The `Edge` class represents an edge in the 10 x 10 grid with accessor methods `getPointA()` and `getPointB()` and a correctly implemented `equals` method.

In this problem you will complete three methods in the `CountingSquares` class. The `CountingSquares` class has an `ArrayList<Edge> myEdges` as an instance variable. The instance variable is initialized by the constructor. All problems assume a 10 x 10 grid and all Edges in `myEdges` are valid (`0 <= getPointA() <= 9` and `0 <= getPointB() <= 9`).

The first method to complete is: `boolean hasSquare(Point lowLeftCorner, int length)`. `hasSquare` returns true if the grid contains the edges required to complete a square with size equal to `length` and the lower left corner of the square at Point `lowLeftCorner`.

The following code shows the results of the `hasSquare` method.

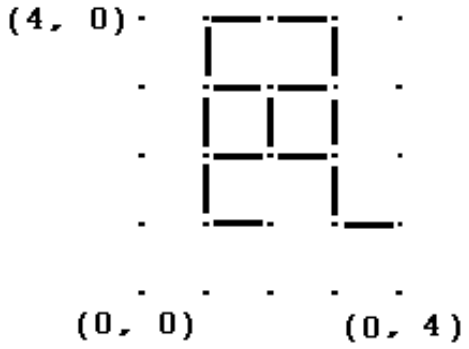
The following code	Returns

`List<Edge> edgs1 = new ArrayList<Edge>();`
`edgs1.add( new Edge( new Point(1, 1), new Point(1, 2)));`
`edgs1.add( new Edge( new Point(1, 1), new Point(2, 1)));`
`edgs1.add( new Edge( new Point(1, 2), new Point(2, 2)));`
`CountingSquares cs = new CountingSquares(edgs1);`	

`cs.hasSquare(new Point(1,1), 1);` `// missing Edge connecting Point (2,1) & Point (2,2)`	`false`
`edgs1.add( new Edge( new Point(2, 1), new Point(2, 2)));` `cs = new CountingSquares(edgs1);`	
`cs.hasSquare(new Point(1,1), 1);`	`true`

One more example on following page

The code in the following tables shows the code to create the indicated `Edge`s in a 10 x 10 grid.



The following code shows the results of the `hasSquare` method using the indicated 10 x 10 Grid.

The following code	Returns

`List<Edge> edgs2 = new ArrayList<Edge>();`
`edgs2.add( new Edge( new Point(1, 1), new Point(1, 2)));`
`edgs2.add( new Edge( new Point(1, 3), new Point(1, 4)));`
`edgs2.add( new Edge( new Point(1, 1), new Point(2, 1)));`
`edgs2.add( new Edge( new Point(2, 1), new Point(2, 2)));`
`edgs2.add( new Edge( new Point(2, 1), new Point(3, 1)));`
`edgs2.add( new Edge( new Point(2, 2), new Point(2, 3)));`
`edgs2.add( new Edge( new Point(2, 2), new Point(3, 2)));`
`edgs2.add( new Edge( new Point(2, 3), new Point(3, 3)));`
`edgs2.add( new Edge( new Point(3, 1), new Point(3, 2)));`
`edgs2.add( new Edge( new Point(3, 3), new Point(3, 2)));`
`edgs2.add( new Edge( new Point(3, 1), new Point(4, 1)));`
`edgs2.add( new Edge( new Point(4, 2), new Point(4, 1)));`
`edgs2.add( new Edge( new Point(4, 3), new Point(4, 2)));`
`edgs2.add( new Edge( new Point(4, 3), new Point(3, 3)));`
`cs = new CountingSquares(edgs2);`	

`cs.hasSquare(new Point(1, 1), 1);`	`false`
`cs.hasSquare(new Point(2, 1), 1);`	`true`
`cs.hasSquare(new Point(2, 2), 1);`	`true`
`cs.hasSquare(new Point(1, 1), 2);`	`false`
`cs.hasSquare(new Point(2, 1), 2);`	`true`

Continue on for the second and third methods.

The second method to complete is: `int getNumSquares(int length)`. `getNumSquares` returns the number of squares of size `length` in the 10 x 10 grid.

The following code shows the results of the `getNumSquares` method.

The following code	Returns

`List<Edge> edgs3 = new ArrayList<Edge>();`
`edgs3.add( new Edge( new Point(1, 1), new Point(1, 2)));`
`edgs3.add( new Edge( new Point(1, 1), new Point(2, 1)));`
`edgs3.add( new Edge( new Point(1, 2), new Point(2, 2)));`
`edgs3.add( new Edge( new Point(2, 1), new Point(2, 2)));`
`cs = new CountingSquares(edgs3);`	

`cs.getNumSquares(1);`	1
`//  adding more Edges to the Grid`	

`edgs3.add( new Edge( new Point(2, 1), new Point(3, 1)));`
`edgs3.add( new Edge( new Point(2, 2), new Point(2, 3)));`
`edgs3.add( new Edge( new Point(4, 3), new Point(4, 2)));`
`edgs3.add( new Edge( new Point(3, 3), new Point(3, 2)));`
`edgs3.add( new Edge( new Point(2, 3), new Point(3, 3)));`
`edgs3.add( new Edge( new Point(3, 1), new Point(3, 2)));`
`edgs3.add( new Edge( new Point(4, 3), new Point(3, 3)));`
`edgs3.add( new Edge( new Point(4, 2), new Point(3, 2)));`
`edgs3.add( new Edge( new Point(2, 2), new Point(3, 2)));`
`cs = new CountingSquares(edgs3);`	

`cs.getNumSquares(1);`	4

Continue on for the third method.

The third method to complete is: `int getSizeOfLargestSquare(). getSizeOfLargestSquare` returns the size of the largest square in the 10 x 10 grid.

The following code shows the results of the `getNumSquares` method.

The following code	Returns

`List<Edge> edgs4 = new ArrayList<Edge>();`
`edgs4.add( new Edge( new Point(1, 1), new Point(1, 2)));`
`edgs4.add( new Edge( new Point(1, 1), new Point(2, 1)));`
`edgs4.add( new Edge( new Point(1, 2), new Point(2, 2)));`
`edgs4.add( new Edge( new Point(2, 1), new Point(2, 2)));`
`edgs4.add( new Edge( new Point(2, 1), new Point(3, 1)));`
`edgs4.add( new Edge( new Point(2, 2), new Point(2, 3)));`
`edgs4.add( new Edge( new Point(4, 3), new Point(4, 2)));`
`edgs4.add( new Edge( new Point(3, 3), new Point(3, 2)));`
`edgs4.add( new Edge( new Point(2, 3), new Point(3, 3)));`
`edgs4.add( new Edge( new Point(3, 1), new Point(3, 2)));`
`edgs4.add( new Edge( new Point(4, 3), new Point(3, 3)));`
`edgs4.add( new Edge( new Point(4, 2), new Point(3, 2)));`
`edgs4.add( new Edge( new Point(2, 2), new Point(3, 2)));`
`edgs4.add( new Edge( new Point(3, 1), new Point(4, 1)));`
`edgs4.add( new Edge( new Point(4, 1), new Point(4, 2)));`
`cs = new CountingSquares(edgs4);`	

`cs.getSizeOfLargestSquare();`	2