Problem 2013: Detect Squares

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given a stream of points on the X-Y plane. Design an algorithm that:

Adds

new points from the stream into a data structure.

Duplicate

points are allowed and should be treated as different points.

Given a query point,

counts

the number of ways to choose three points from the data structure such that the three points and the query point form an

axis-aligned square

with

positive area

.

An

axis-aligned square

is a square whose edges are all the same length and are either parallel or perpendicular to the x-axis and y-axis.

Implement the

DetectSquares

class:

DetectSquares()

Initializes the object with an empty data structure.

void add(int[] point)

Adds a new point

point = [x, y]

to the data structure.

int count(int[] point)

Counts the number of ways to form

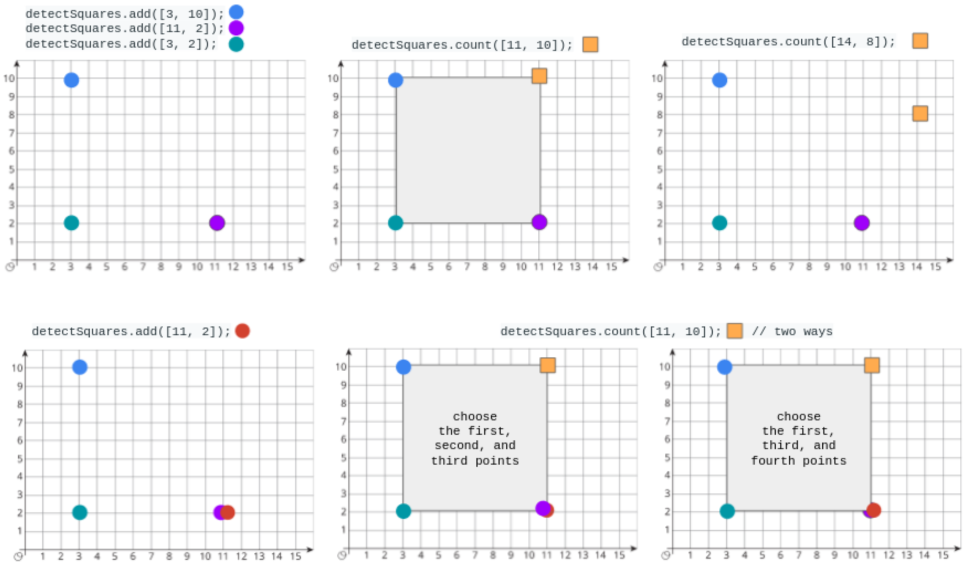
axis-aligned squares

with point

point = [x, y]

as described above.

Example 1:



```
detectSquares.add([3, 10]);
detectSquares.add([11, 2]);
detectSquares.add([3, 2]);
detectSquares.count([11, 10]);
detectSquares.count([14, 8]);
detectSquares.add([11, 2]);
detectSquares.count([11, 10]); // two ways
```

choose the first, second, and third points
choose the first, third, and fourth points
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1 2 3 4 5 6 7 8 9 10

Input

["DetectSquares", "add", "add", "add", "count", "count", "add", "count"] [[], [[3, 10]], [[11, 2]], [[3, 2]], [[11, 10]], [[14, 8]], [[11, 2]], [[11, 10]]]

Output

[null, null, null, null, 1, 0, null, 2]

Explanation

DetectSquares detectSquares = new DetectSquares(); detectSquares.add([3, 10]); detectSquares.add([11, 2]); detectSquares.add([3, 2]); detectSquares.count([11, 10]); // return 1. You can choose: // - The first, second, and third points detectSquares.count([14, 8]); // return 0. The query point cannot form a square with any points in the data structure. detectSquares.add([11, 2]); // Adding duplicate points is allowed. detectSquares.count([11, 10]); // return 2. You can choose: // - The first, second, and third points // - The first, third, and fourth points

Constraints:

point.length == 2

0 <= x, y <= 1000

At most

3000

calls

in total

will be made to

add

and

count

.

Code Snippets

C++:

```
class DetectSquares {
public:
DetectSquares() {

}

void add(vector<int> point) {

}

int count(vector<int> point) {

}
};

/**
* Your DetectSquares object will be instantiated and called as such:
* DetectSquares* obj = new DetectSquares();
* obj->add(point);
* int param_2 = obj->count(point);
```

```
*/
```

Java:

```
class DetectSquares {

public DetectSquares() {

}

public void add(int[] point) {

}

public int count(int[] point) {

}
}

/**
* Your DetectSquares object will be instantiated and called as such:
* DetectSquares obj = new DetectSquares();
* obj.add(point);
* int param_2 = obj.count(point);
*/
```

Python3:

```
class DetectSquares:

def __init__(self):



def add(self, point: List[int]) -> None:



def count(self, point: List[int]) -> int:




# Your DetectSquares object will be instantiated and called as such:
# obj = DetectSquares()
# obj.add(point)
# param_2 = obj.count(point)
```

Python:

```
class DetectSquares(object):

def __init__(self):



def add(self, point):
"""
:type point: List[int]
:rtype: None
"""



def count(self, point):
"""
:type point: List[int]
:rtype: int
"""





# Your DetectSquares object will be instantiated and called as such:
# obj = DetectSquares()
# obj.add(point)
# param_2 = obj.count(point)
```

JavaScript:

```
var DetectSquares = function() {

};

/**
* @param {number[]} point
* @return {void}
*/
DetectSquares.prototype.add = function(point) {
};

/**
* @param {number[]} point
* @return {number}
*/
DetectSquares.prototype.count = function(point) {

};

/**
* Your DetectSquares object will be instantiated and called as such:
* var obj = new DetectSquares()
* obj.add(point)
* var param_2 = obj.count(point)
*/
```

TypeScript:

```
class DetectSquares {
constructor() {

}

add(point: number[]): void {

}

count(point: number[]): number {

}
}

/**
* Your DetectSquares object will be instantiated and called as such:
* var obj = new DetectSquares()
* obj.add(point)
* var param_2 = obj.count(point)
*/
```

C#:

```
public class DetectSquares {

public DetectSquares() {

}

public void Add(int[] point) {

}

public int Count(int[] point) {

}
}

/**
* Your DetectSquares object will be instantiated and called as such:
* DetectSquares obj = new DetectSquares();
* obj.Add(point);
* int param_2 = obj.Count(point);
*/
```

C:

```



typedef struct {

} DetectSquares;


DetectSquares* detectSquaresCreate() {

}

void detectSquaresAdd(DetectSquares* obj, int* point, int pointSize) {

}

int detectSquaresCount(DetectSquares* obj, int* point, int pointSize) {
}

void detectSquaresFree(DetectSquares* obj) {

}

/**
* Your DetectSquares struct will be instantiated and called as such:
* DetectSquares* obj = detectSquaresCreate();
* detectSquaresAdd(obj, point, pointSize);

* int param_2 = detectSquaresCount(obj, point, pointSize);

* detectSquaresFree(obj);
*/
```

Go:

```
type DetectSquares struct {

}


func Constructor() DetectSquares {

}


func (this *DetectSquares) Add(point []int) {

}


func (this *DetectSquares) Count(point []int) int {

}


/**
* Your DetectSquares object will be instantiated and called as such:
* obj := Constructor();
* obj.Add(point);
* param_2 := obj.Count(point);
*/
```

Kotlin:

```
class DetectSquares() {

fun add(point: IntArray) {

}

fun count(point: IntArray): Int {

}

}

/**
* Your DetectSquares object will be instantiated and called as such:
* var obj = DetectSquares()
* obj.add(point)
* var param_2 = obj.count(point)
*/
```

Swift:

```
class DetectSquares {

init() {

}

func add(_ point: [Int]) {

}

func count(_ point: [Int]) -> Int {

}
}
```

```
/**
* Your DetectSquares object will be instantiated and called as such:
* let obj = DetectSquares()
* obj.add(point)
* let ret_2: Int = obj.count(point)
*/
```

Rust:

```
struct DetectSquares {

}



/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl DetectSquares {

fn new() -> Self {

}

fn add(&self, point: Vec<i32>) {

}

fn count(&self, point: Vec<i32>) -> i32 {

}
}

/**
* Your DetectSquares object will be instantiated and called as such:
* let obj = DetectSquares::new();
* obj.add(point);
* let ret_2: i32 = obj.count(point);
*/
```

Ruby:

```ruby
class DetectSquares
def initialize()

end


=begin
:type point: Integer[]
:rtype: Void
=end
def add(point)

end


=begin
:type point: Integer[]
:rtype: Integer
=end
def count(point)

end


end

# Your DetectSquares object will be instantiated and called as such:
# obj = DetectSquares.new()
# obj.add(point)
# param_2 = obj.count(point)
```

PHP:

```php
class DetectSquares {
/**
*/
function __construct() {

}

/**
* @param Integer[] $point
* @return NULL
*/
function add($point) {

}

/**
* @param Integer[] $point
* @return Integer
*/
function count($point) {

}
}

/**
* Your DetectSquares object will be instantiated and called as such:
* $obj = DetectSquares();
* $obj->add($point);
* $ret_2 = $obj->count($point);
*/
```

Dart:

```
class DetectSquares {

DetectSquares() {

}

void add(List<int> point) {

}

int count(List<int> point) {

}
}

/**
* Your DetectSquares object will be instantiated and called as such:
```

```
* DetectSquares obj = DetectSquares();
* obj.add(point);
* int param2 = obj.count(point);
*/
```

Scala:

```
class DetectSquares() {

def add(point: Array[Int]): Unit = {

}

def count(point: Array[Int]): Int = {

}

}

/**
* Your DetectSquares object will be instantiated and called as such:
* val obj = new DetectSquares()
* obj.add(point)
* val param_2 = obj.count(point)
*/
```

Elixir:

```
defmodule DetectSquares do
@spec init_() :: any
def init_() do

end

@spec add(point :: [integer]) :: any
def add(point) do

end

@spec count(point :: [integer]) :: integer
def count(point) do
```

```
end
end

# Your functions will be called as such:
# DetectSquares.init_()
# DetectSquares.add(point)
# param_2 = DetectSquares.count(point)

# DetectSquares.init_ will be called before every test case, in which you can
do some necessary initializations.
```

Erlang:

```
-spec detect_squares_init_() -> any().
detect_squares_init_() ->
.

-spec detect_squares_add(Point :: [integer()]) -> any().
detect_squares_add(Point) ->
.

-spec detect_squares_count(Point :: [integer()]) -> integer().
detect_squares_count(Point) ->
.



%% Your functions will be called as such:
%% detect_squares_init_(),
%% detect_squares_add(Point),
%% Param_2 = detect_squares_count(Point),

%% detect_squares_init_ will be called before every test case, in which you
can do some necessary initializations.
```

Racket:

```
(define detect-squares%
(class object%
(super-new)

(init-field)
; add : (listof exact-integer?) -> void?
(define/public (add point)
)
; count : (listof exact-integer?) -> exact-integer?
(define/public (count point)
)))

;; Your detect-squares% object will be instantiated and called as such:
;; (define obj (new detect-squares%))
;; (send obj add point)
;; (define param_2 (send obj count point))
```

Solutions

C++ Solution:

```
/*
* Problem: Detect Squares
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class DetectSquares {
public:
DetectSquares() {

}

void add(vector<int> point) {

}

int count(vector<int> point) {

}
};
```

```
/**
* Your DetectSquares object will be instantiated and called as such:
* DetectSquares* obj = new DetectSquares();
* obj->add(point);
* int param_2 = obj->count(point);
*/
```

Java Solution:

```
/**
* Problem: Detect Squares
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class DetectSquares {

public DetectSquares() {

}

public void add(int[] point) {

}

public int count(int[] point) {

}
}

/**
* Your DetectSquares object will be instantiated and called as such:
* DetectSquares obj = new DetectSquares();
* obj.add(point);
* int param_2 = obj.count(point);
*/
```

Python3 Solution:

```
"""
Problem: Detect Squares
Difficulty: Medium
Tags: array, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class DetectSquares:

def __init__(self):



def add(self, point: List[int]) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class DetectSquares(object):

def __init__(self):



def add(self, point):
"""
:type point: List[int]
:rtype: None
"""



def count(self, point):
"""
:type point: List[int]
:rtype: int
"""
```

```
# Your DetectSquares object will be instantiated and called as such:
# obj = DetectSquares()
# obj.add(point)
# param_2 = obj.count(point)
```

JavaScript Solution:

```
/**
* Problem: Detect Squares
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/


var DetectSquares = function() {

};

/**
* @param {number[]} point
* @return {void}
*/
DetectSquares.prototype.add = function(point) {

};

/**
* @param {number[]} point
* @return {number}
*/
DetectSquares.prototype.count = function(point) {

};

/**
* Your DetectSquares object will be instantiated and called as such:
```

```
* var obj = new DetectSquares()
* obj.add(point)
* var param_2 = obj.count(point)
*/
```

TypeScript Solution:

```
/**
* Problem: Detect Squares
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class DetectSquares {
constructor() {

}

add(point: number[]): void {

}

count(point: number[]): number {

}
}

/**
* Your DetectSquares object will be instantiated and called as such:
* var obj = new DetectSquares()
* obj.add(point)
* var param_2 = obj.count(point)
*/
```

C# Solution:

```
/*
* Problem: Detect Squares
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class DetectSquares {

public DetectSquares() {

}

public void Add(int[] point) {

}

public int Count(int[] point) {

}
}

/**
* Your DetectSquares object will be instantiated and called as such:
* DetectSquares obj = new DetectSquares();
* obj.Add(point);
* int param_2 = obj.Count(point);
*/
```

C Solution:

```
/*
* Problem: Detect Squares
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/




typedef struct {

} DetectSquares;


DetectSquares* detectSquaresCreate() {

}

void detectSquaresAdd(DetectSquares* obj, int* point, int pointSize) {

}

int detectSquaresCount(DetectSquares* obj, int* point, int pointSize) {

}

void detectSquaresFree(DetectSquares* obj) {

}

/**
* Your DetectSquares struct will be instantiated and called as such:
* DetectSquares* obj = detectSquaresCreate();
* detectSquaresAdd(obj, point, pointSize);

* int param_2 = detectSquaresCount(obj, point, pointSize);

* detectSquaresFree(obj);
*/
```

Go Solution:

```
// Problem: Detect Squares
// Difficulty: Medium
// Tags: array, hash
```

```
}
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

type DetectSquares struct {

}



func Constructor() DetectSquares {

}



func (this *DetectSquares) Add(point []int) {

}



func (this *DetectSquares) Count(point []int) int {

}



/**
* Your DetectSquares object will be instantiated and called as such:
* obj := Constructor();
* obj.Add(point);
* param_2 := obj.Count(point);
*/
```

Kotlin Solution:

```
class DetectSquares() {

fun add(point: IntArray) {

}

fun count(point: IntArray): Int {
}

}

/**
* Your DetectSquares object will be instantiated and called as such:
* var obj = DetectSquares()
* obj.add(point)
* var param_2 = obj.count(point)
*/
```

Swift Solution:

```
class DetectSquares {

init() {

}

func add(_ point: [Int]) {

}

func count(_ point: [Int]) -> Int {

}
}

/**
* Your DetectSquares object will be instantiated and called as such:
* let obj = DetectSquares()
* obj.add(point)
* let ret_2: Int = obj.count(point)
*/
```

Rust Solution:

```
// Problem: Detect Squares
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

struct DetectSquares {

}



/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl DetectSquares {

fn new() -> Self {

}

fn add(&self, point: Vec<i32>) {

}

fn count(&self, point: Vec<i32>) -> i32 {

}
}

/**
* Your DetectSquares object will be instantiated and called as such:
* let obj = DetectSquares::new();
* obj.add(point);
* let ret_2: i32 = obj.count(point);
*/
```

Ruby Solution:

```
class DetectSquares
def initialize()
```

```
end



=begin
:type point: Integer[]
:rtype: Void
=end
def add(point)

end



=begin
:type point: Integer[]
:rtype: Integer
=end
def count(point)

end



end

# Your DetectSquares object will be instantiated and called as such:
# obj = DetectSquares.new()
# obj.add(point)
# param_2 = obj.count(point)
```

PHP Solution:

```
class DetectSquares {
/**
*/
function __construct() {

}

/**
* @param Integer[] $point
* @return NULL
*/
function add($point) {

}

/**
* @param Integer[] $point
* @return Integer
*/
function count($point) {

}
}

/**
* Your DetectSquares object will be instantiated and called as such:
* $obj = DetectSquares();
* $obj->add($point);
* $ret_2 = $obj->count($point);
*/
```

Dart Solution:

```
class DetectSquares {

DetectSquares() {

}

void add(List<int> point) {

}

int count(List<int> point) {

}
}

/**
* Your DetectSquares object will be instantiated and called as such:
* DetectSquares obj = DetectSquares();
* obj.add(point);
* int param2 = obj.count(point);
*/
```

Scala Solution:

```
class DetectSquares() {

def add(point: Array[Int]): Unit = {

}

def count(point: Array[Int]): Int = {

}

}

/**
* Your DetectSquares object will be instantiated and called as such:
* val obj = new DetectSquares()
* obj.add(point)
* val param_2 = obj.count(point)
*/
```

Elixir Solution:

```
defmodule DetectSquares do
@spec init_() :: any
def init_() do

end

@spec add(point :: [integer]) :: any
def add(point) do

end

@spec count(point :: [integer]) :: integer
def count(point) do
```

```
end
end

# Your functions will be called as such:
# DetectSquares.init_()
# DetectSquares.add(point)
# param_2 = DetectSquares.count(point)

# DetectSquares.init_ will be called before every test case, in which you can
do some necessary initializations.
```

Erlang Solution:

```
-spec detect_squares_init_() -> any().
detect_squares_init_() ->
.

-spec detect_squares_add(Point :: [integer()]) -> any().
detect_squares_add(Point) ->
.

-spec detect_squares_count(Point :: [integer()]) -> integer().
detect_squares_count(Point) ->
.



%% Your functions will be called as such:
%% detect_squares_init_(),
%% detect_squares_add(Point),
%% Param_2 = detect_squares_count(Point),

%% detect_squares_init_ will be called before every test case, in which you
can do some necessary initializations.
```

Racket Solution:

```
(define detect-squares%
(class object%
(super-new)

(init-field)
; add : (listof exact-integer?) -> void?
(define/public (add point)
)
; count : (listof exact-integer?) -> exact-integer?
(define/public (count point)
)))

;; Your detect-squares% object will be instantiated and called as such:
;; (define obj (new detect-squares%))
;; (send obj add point)
;; (define param_2 (send obj count point))
```