Problem 1610: Maximum Number of Visible Points

Problem Information

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

Problem Description

You are given an array

points

, an integer

angle

, and your

location

, where

location = [pos

x

, pos

y

]

and

points[i] = [x

i

, y

i

]

both denote

integral coordinates

on the X-Y plane.

Initially, you are facing directly east from your position. You

cannot move

from your position, but you can

rotate

. In other words,

pos

x

and

pos

y

cannot be changed. Your field of view in degrees

is represented by

angle

, determining how wide you can see from any given view direction. Let

d

be the amount in degrees that you rotate counterclockwise. Then, your field of view is the

inclusive

range of angles

[d - angle/2, d + angle/2]

.

Your browser does not support the video tag or this video format.

You can

see

some set of points if, for each point, the

angle

formed by the point, your position, and the immediate east direction from your position is

in your field of view

.

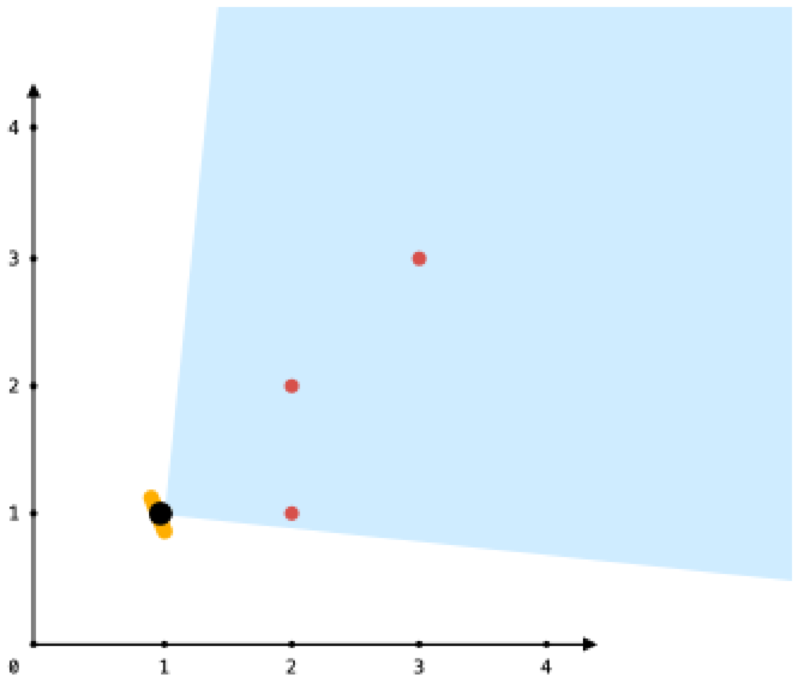
There can be multiple points at one coordinate. There may be points at your location, and you can always see these points regardless of your rotation. Points do not obstruct your vision to other points.

Return

the maximum number of points you can see

.

Example 1:



Input:

points = [[2,1],[2,2],[3,3]], angle = 90, location = [1,1]

Output:

3

Explanation:

The shaded region represents your field of view. All points can be made visible in your field of view, including [3,3] even though [2,2] is in front and in the same line of sight.

Example 2:

Input:

points = [[2,1],[2,2],[3,4],[1,1]], angle = 90, location = [1,1]

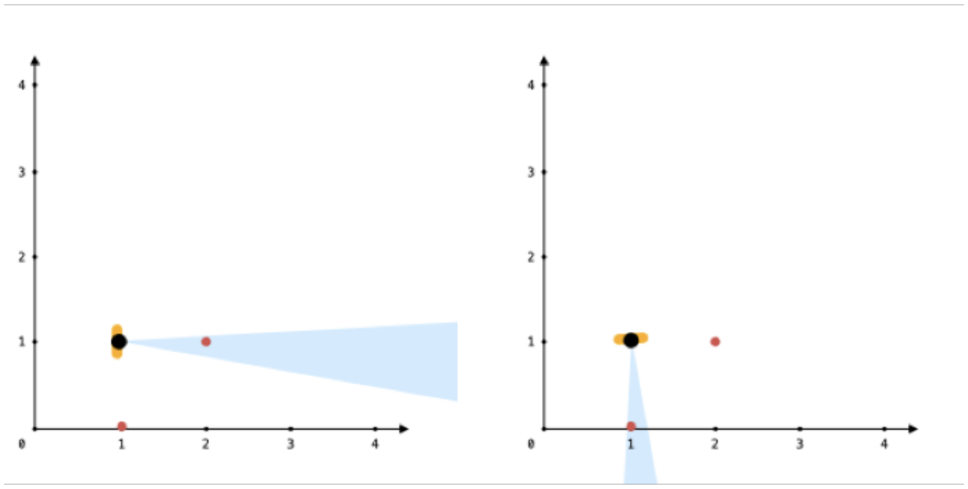
Output:

4

Explanation:

All points can be made visible in your field of view, including the one at your location.

Example 3:



Input:

points = [[1,0],[2,1]], angle = 13, location = [1,1]

Output:

1

Explanation:

You can only see one of the two points, as shown above.

Constraints:

1 <= points.length <= 10

5

points[i].length == 2

location.length == 2

0 <= angle < 360

0 <= pos

x

, pos

y

, x

i

, y

i

<= 100

Code Snippets

C++:

```
class Solution {
public:
int visiblePoints(vector<vector<int>>& points, int angle, vector<int>&
location) {

}
};
```

Java:

```
class Solution {
public int visiblePoints(List<List<Integer>> points, int angle, List<Integer>
location) {

}
}
```

Python3:

```
class Solution:
def visiblePoints(self, points: List[List[int]], angle: int, location:
List[int]) -> int:
```

Python:

```
class Solution(object):
def visiblePoints(self, points, angle, location):
"""
:type points: List[List[int]]
:type angle: int
:type location: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} points
* @param {number} angle
* @param {number[]} location
* @return {number}
*/
var visiblePoints = function(points, angle, location) {

};
```

TypeScript:

```
function visiblePoints(points: number[][], angle: number, location:
number[]): number {

};
```

C#:

```
public class Solution {
public int VisiblePoints(IList<IList<int>> points, int angle, IList<int>
location) {

}
}
```

C:

```
int visiblePoints(int** points, int pointsSize, int* pointsColSize, int
angle, int* location, int locationSize) {

}
```

Go:

```
func visiblePoints(points [][]int, angle int, location []int) int {

}
```

Kotlin:

```
class Solution {
fun visiblePoints(points: List<List<Int>>, angle: Int, location: List<Int>):
Int {

}
}
```

Swift:

```
class Solution {
func visiblePoints(_ points: [[Int]], _ angle: Int, _ location: [Int]) -> Int
{

}
}
```

Rust:

```
impl Solution {
pub fn visible_points(points: Vec<Vec<i32>>, angle: i32, location: Vec<i32>)
-> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} points
# @param {Integer} angle
# @param {Integer[]} location
# @return {Integer}
def visible_points(points, angle, location)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $points
* @param Integer $angle
* @param Integer[] $location
* @return Integer
*/
function visiblePoints($points, $angle, $location) {

}
}
```

Dart:

```
class Solution {
int visiblePoints(List<List<int>> points, int angle, List<int> location) {

}
}
```

Scala:

```
object Solution {
def visiblePoints(points: List[List[Int]], angle: Int, location: List[Int]):
Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec visible_points(points :: [[integer]], angle :: integer, location ::
[integer]) :: integer
def visible_points(points, angle, location) do

end
end
```

Erlang:

```
-spec visible_points(Points :: [[integer()]], Angle :: integer(), Location ::
[integer()]) -> integer().
visible_points(Points, Angle, Location) ->
.
```

Racket:

```
(define/contract (visible-points points angle location)
(-> (listof (listof exact-integer?)) exact-integer? (listof exact-integer?)
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Visible Points
* Difficulty: Hard
* Tags: array, tree, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
int visiblePoints(vector<vector<int>>& points, int angle, vector<int>&
location) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Visible Points
* Difficulty: Hard
* Tags: array, tree, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int visiblePoints(List<List<Integer>> points, int angle, List<Integer>
location) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Visible Points
Difficulty: Hard
```

```
Tags: array, tree, math, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def visiblePoints(self, points: List[List[int]], angle: int, location:
List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def visiblePoints(self, points, angle, location):
"""
:type points: List[List[int]]
:type angle: int
:type location: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Number of Visible Points
* Difficulty: Hard
* Tags: array, tree, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number[][]} points
* @param {number} angle
* @param {number[]} location
* @return {number}
```

```
*/
var visiblePoints = function(points, angle, location) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Visible Points
* Difficulty: Hard
* Tags: array, tree, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function visiblePoints(points: number[][], angle: number, location:
number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Visible Points
* Difficulty: Hard
* Tags: array, tree, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int VisiblePoints(IList<IList<int>> points, int angle, IList<int>
location) {

}
}
```

C Solution:

```c
/*
* Problem: Maximum Number of Visible Points
* Difficulty: Hard
* Tags: array, tree, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int visiblePoints(int** points, int pointsSize, int* pointsColSize, int
angle, int* location, int locationSize) {

}
```

Go Solution:

```go
// Problem: Maximum Number of Visible Points
// Difficulty: Hard
// Tags: array, tree, math, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func visiblePoints(points [][]int, angle int, location []int) int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun visiblePoints(points: List<List<Int>>, angle: Int, location: List<Int>):
Int {

}
}
```

Swift Solution:

```
class Solution {
func visiblePoints(_ points: [[Int]], _ angle: Int, _ location: [Int]) -> Int
{

}
}
```

Rust Solution:

```
// Problem: Maximum Number of Visible Points
// Difficulty: Hard
// Tags: array, tree, math, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn visible_points(points: Vec<Vec<i32>>, angle: i32, location: Vec<i32>)
-> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} points
# @param {Integer} angle
# @param {Integer[]} location
# @return {Integer}
def visible_points(points, angle, location)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $points
* @param Integer $angle
* @param Integer[] $location
* @return Integer
*/
function visiblePoints($points, $angle, $location) {

}
}
```

Dart Solution:

```
class Solution {
int visiblePoints(List<List<int>> points, int angle, List<int> location) {

}
}
```

Scala Solution:

```
object Solution {
def visiblePoints(points: List[List[Int]], angle: Int, location: List[Int]):
Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec visible_points(points :: [[integer]], angle :: integer, location ::
[integer]) :: integer
def visible_points(points, angle, location) do

end
end
```

Erlang Solution:

```
-spec visible_points(Points :: [[integer()]], Angle :: integer(), Location ::
[integer()]) -> integer().
visible_points(Points, Angle, Location) ->
.
```

Racket Solution:

```
(define/contract (visible-points points angle location)
(-> (listof (listof exact-integer?)) exact-integer? (listof exact-integer?)
exact-integer?)
)
```