Problem 1956: Minimum Time For K Virus Variants to Spread

Problem Information

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

Problem Description

There are

n

unique

virus variants in an infinite 2D grid. You are given a 2D array

points

, where

points[i] = [x

i

, y

i

]

represents a virus originating at

(x

i

, y

i

)

on day

0

. Note that it is possible for

multiple

virus variants to originate at the

same

point.

Every day, each cell infected with a virus variant will spread the virus to

all

neighboring points in the

four

cardinal directions (i.e. up, down, left, and right). If a cell has multiple variants, all the variants will spread without interfering with each other.

Given an integer

k

, return

the

minimum integer

number of days for

any

point to contain

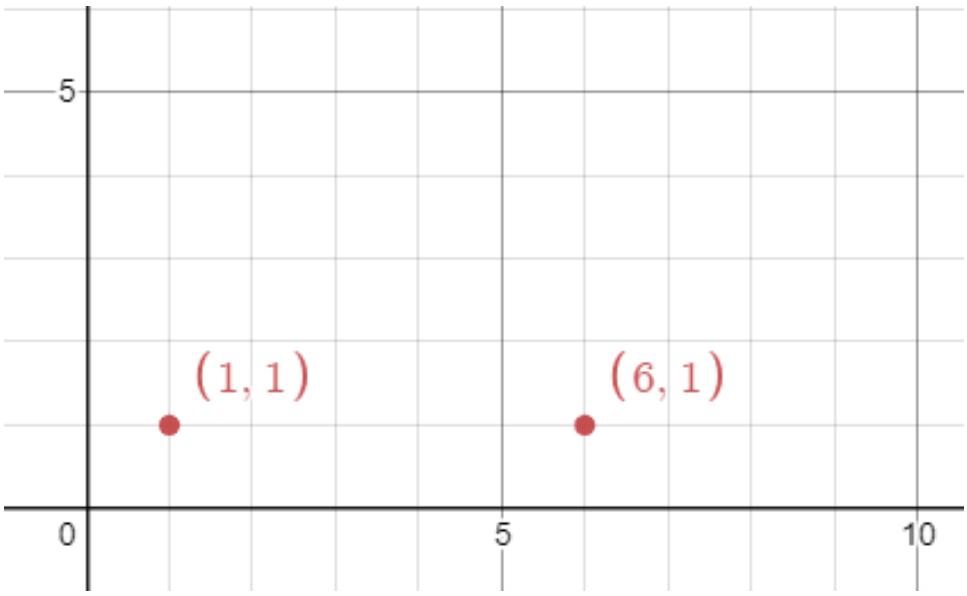
at least

k

of the unique virus variants

.

Example 1:



Input:

points = [[1,1],[6,1]], k = 2

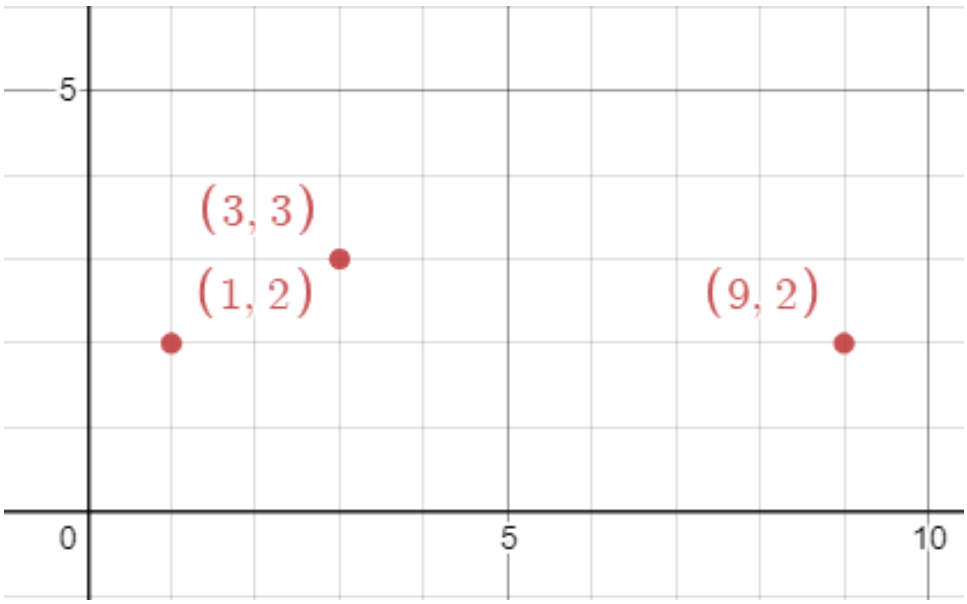
Output:

3

Explanation:

On day 3, points (3,1) and (4,1) will contain both virus variants. Note that these are not the only points that will contain both virus variants.

Example 2:



Input:

points = [[3,3],[1,2],[9,2]], k = 2

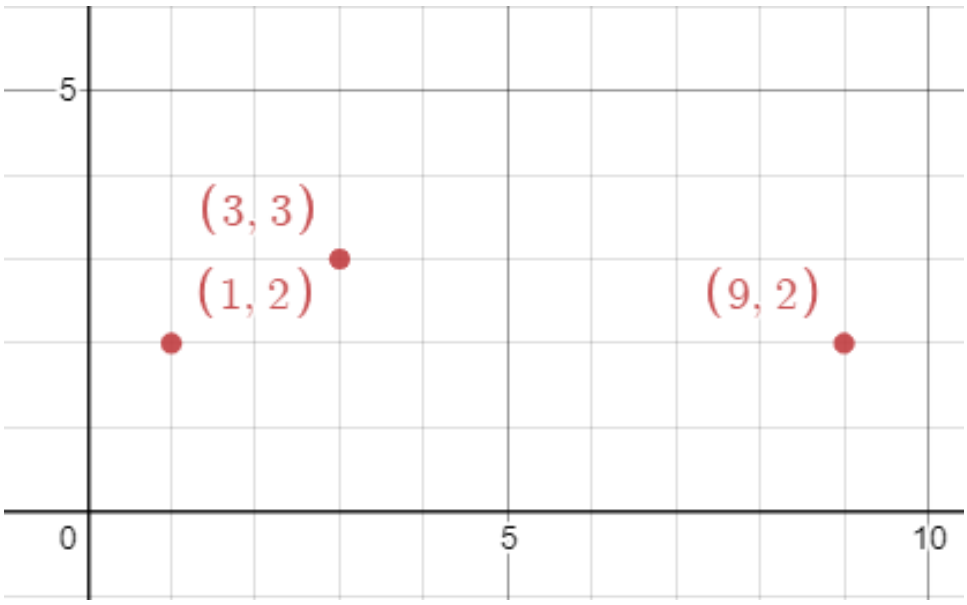
Output:

2

Explanation:

On day 2, points (1,3), (2,3), (2,2), and (3,2) will contain the first two viruses. Note that these are not the only points that will contain both virus variants.

Example 3:



Input:

points = [[3,3],[1,2],[9,2]], k = 3

Output:

4

Explanation:

On day 4, the point (5,2) will contain all 3 viruses. Note that this is not the only point that will contain all 3 virus variants.

Constraints:

n == points.length

2 <= n <= 50

points[i].length == 2

1 <= x

i

, y

i

<= 100

2 <= k <= n

Code Snippets

C++:

```
class Solution {
public:
int minDayskVariants(vector<vector<int>>& points, int k) {

}
};
```

Java:

```
class Solution {
public int minDayskVariants(int[][] points, int k) {

}
}
```

Python3:

```
class Solution:
def minDayskVariants(self, points: List[List[int]], k: int) -> int:
```

Python:

```
class Solution(object):
def minDayskVariants(self, points, k):
"""
:type points: List[List[int]]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} points
* @param {number} k
* @return {number}
*/
var minDayskVariants = function(points, k) {

};
```

TypeScript:

```
function minDayskVariants(points: number[][], k: number): number {

};
```

C#:

```
public class Solution {
public int MinDayskVariants(int[][] points, int k) {

}
}
```

C:

```
int minDayskVariants(int** points, int pointsSize, int* pointsColSize, int k)
{

}
```

Go:

```
func minDayskVariants(points [][]int, k int) int {

}
```

Kotlin:

```
class Solution {
fun minDayskVariants(points: Array<IntArray>, k: Int): Int {
```

```
}
}
```

Swift:

```
class Solution {
func minDayskVariants(_ points: [[Int]], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_daysk_variants(points: Vec<Vec<i32>>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} points
# @param {Integer} k
# @return {Integer}
def min_daysk_variants(points, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $points
* @param Integer $k
* @return Integer
*/
function minDayskVariants($points, $k) {

}
}
```

Dart:

```
class Solution {
int minDayskVariants(List<List<int>> points, int k) {

}
}
```

Scala:

```
object Solution {
def minDayskVariants(points: Array[Array[Int]], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_daysk_variants(points :: [[integer]], k :: integer) :: integer
def min_daysk_variants(points, k) do

end
end
```

Erlang:

```
-spec min_daysk_variants(Points :: [[integer()]], K :: integer()) ->
integer().
min_daysk_variants(Points, K) ->
.
```

Racket:

```
(define/contract (min-daysk-variants points k)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Time For K Virus Variants to Spread
* Difficulty: Hard
* Tags: array, math, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int minDayskVariants(vector<vector<int>>& points, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Time For K Virus Variants to Spread
* Difficulty: Hard
* Tags: array, math, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int minDayskVariants(int[][] points, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Time For K Virus Variants to Spread
Difficulty: Hard
Tags: array, math, search
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

```
class Solution(object):
def minDayskVariants(self, points, k):
"""
:type points: List[List[int]]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Time For K Virus Variants to Spread
* Difficulty: Hard
* Tags: array, math, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[][]} points
* @param {number} k
* @return {number}
*/
var minDayskVariants = function(points, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Time For K Virus Variants to Spread
* Difficulty: Hard
* Tags: array, math, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function minDayskVariants(points: number[][], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Time For K Virus Variants to Spread
* Difficulty: Hard
* Tags: array, math, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int MinDayskVariants(int[][] points, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Time For K Virus Variants to Spread
* Difficulty: Hard
* Tags: array, math, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(1) to O(n) depending on approach
*/

int minDayskVariants(int** points, int pointsSize, int* pointsColSize, int k)
{

}
```

Go Solution:

```
// Problem: Minimum Time For K Virus Variants to Spread
// Difficulty: Hard
// Tags: array, math, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func minDayskVariants(points [][]int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minDayskVariants(points: Array<IntArray>, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minDayskVariants(_ points: [[Int]], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Time For K Virus Variants to Spread
// Difficulty: Hard
// Tags: array, math, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn min_daysk_variants(points: Vec<Vec<i32>>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} points
# @param {Integer} k
# @return {Integer}
def min_daysk_variants(points, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $points
* @param Integer $k
* @return Integer
*/
function minDayskVariants($points, $k) {

}
}
```

Dart Solution:

```
class Solution {
int minDayskVariants(List<List<int>> points, int k) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def minDayskVariants(points: Array[Array[Int]], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_daysk_variants(points :: [[integer]], k :: integer) :: integer
def min_daysk_variants(points, k) do

end
end
```

Erlang Solution:

```
-spec min_daysk_variants(Points :: [[integer()]], K :: integer()) ->
integer().
min_daysk_variants(Points, K) ->
.
```

Racket Solution:

```
(define/contract (min-daysk-variants points k)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```