Problem 3648: Minimum Sensors to Cover Grid

Problem Information

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

Problem Description

You are given

n × m

grid and an integer

k

.

A sensor placed on cell

(r, c)

covers all cells whose

Chebyshev distance

from

(r, c)

is

at most

k

.

The

Chebyshev distance

between two cells

(r

1

, c

1

)

and

(r

2

, c

2

)

is

max(|r

1

– r

2

|,|c

1

– c

2

|)

.

Your task is to return the

minimum

number of sensors required to cover every cell of the grid.

Example 1:

Input:

n = 5, m = 5, k = 1

Output:

4

Explanation:

Placing sensors at positions

(0, 3)

,

(1, 0)

,

(3, 3)

, and

(4, 1)

ensures every cell in the grid is covered. Thus, the answer is 4.

Example 2:

Input:

n = 2, m = 2, k = 2

Output:

1

Explanation:

With

k = 2

, a single sensor can cover the entire

2 * 2

grid regardless of its position. Thus, the answer is 1.

Constraints:

1 <= n <= 10

3

1 <= m <= 10

3

0 <= k <= 10

3

Code Snippets

C++:

```
class Solution {
public:
int minSensors(int n, int m, int k) {

}
};
```

Java:

```
class Solution {
public int minSensors(int n, int m, int k) {

}
}
```

Python3:

```
class Solution:
def minSensors(self, n: int, m: int, k: int) -> int:
```

Python:

```
class Solution(object):
def minSensors(self, n, m, k):
"""
:type n: int
:type m: int
:type k: int
```

```
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number} m
* @param {number} k
* @return {number}
*/
var minSensors = function(n, m, k) {

};
```

TypeScript:

```
function minSensors(n: number, m: number, k: number): number {

};
```

C#:

```
public class Solution {
public int MinSensors(int n, int m, int k) {

}
}
```

C:

```
int minSensors(int n, int m, int k) {

}
```

Go:

```
func minSensors(n int, m int, k int) int {

}
```

Kotlin:

```
class Solution {
fun minSensors(n: Int, m: Int, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func minSensors(_ n: Int, _ m: Int, _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_sensors(n: i32, m: i32, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} m
# @param {Integer} k
# @return {Integer}
def min_sensors(n, m, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $m
* @param Integer $k
* @return Integer
*/
function minSensors($n, $m, $k) {
```

```
}
}
```

Dart:

```
class Solution {
int minSensors(int n, int m, int k) {

}
}
```

Scala:

```
object Solution {
def minSensors(n: Int, m: Int, k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_sensors(n :: integer, m :: integer, k :: integer) :: integer
def min_sensors(n, m, k) do

end
end
```

Erlang:

```
-spec min_sensors(N :: integer(), M :: integer(), K :: integer()) ->
integer().
min_sensors(N, M, K) ->
.
```

Racket:

```
(define/contract (min-sensors n m k)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Sensors to Cover Grid
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int minSensors(int n, int m, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Sensors to Cover Grid
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int minSensors(int n, int m, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Sensors to Cover Grid
Difficulty: Medium
Tags: math

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def minSensors(self, n: int, m: int, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minSensors(self, n, m, k):
"""
:type n: int
:type m: int
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Sensors to Cover Grid
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} n
* @param {number} m
* @param {number} k
```

```
* @return {number}
*/
var minSensors = function(n, m, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Sensors to Cover Grid
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function minSensors(n: number, m: number, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Sensors to Cover Grid
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int MinSensors(int n, int m, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Sensors to Cover Grid
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

int minSensors(int n, int m, int k) {

}
```

Go Solution:

```
// Problem: Minimum Sensors to Cover Grid
// Difficulty: Medium
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func minSensors(n int, m int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minSensors(n: Int, m: Int, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minSensors(_ n: Int, _ m: Int, _ k: Int) -> Int {

}
```

```
}
```

Rust Solution:

```
// Problem: Minimum Sensors to Cover Grid
// Difficulty: Medium
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn min_sensors(n: i32, m: i32, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} m
# @param {Integer} k
# @return {Integer}
def min_sensors(n, m, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $m
* @param Integer $k
* @return Integer
*/
function minSensors($n, $m, $k) {

}
```

```
}
```

Dart Solution:

```
class Solution {
int minSensors(int n, int m, int k) {

}
}
```

Scala Solution:

```
object Solution {
def minSensors(n: Int, m: Int, k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_sensors(n :: integer, m :: integer, k :: integer) :: integer
def min_sensors(n, m, k) do

end
end
```

Erlang Solution:

```
-spec min_sensors(N :: integer(), M :: integer(), K :: integer()) ->
integer().
min_sensors(N, M, K) ->
.
```

Racket Solution:

```
(define/contract (min-sensors n m k)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```