Problem 2579: Count Total Number of Colored Cells

Problem Information

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

Problem Description

There exists an infinitely large two-dimensional grid of uncolored unit cells. You are given a positive integer

n

, indicating that you must do the following routine for

n

minutes:

At the first minute, color

any

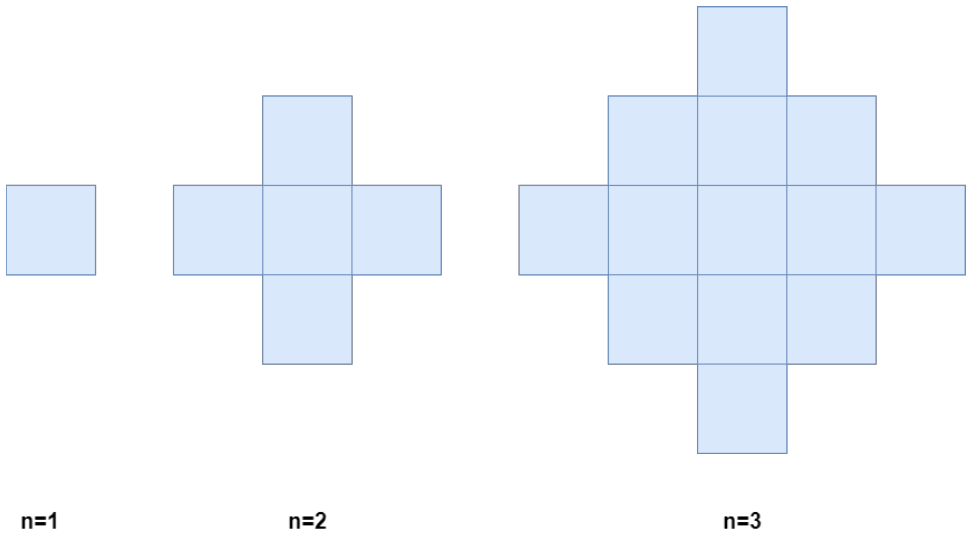
arbitrary unit cell blue.

Every minute thereafter, color blue

every

uncolored cell that touches a blue cell.

Below is a pictorial representation of the state of the grid after minutes 1, 2, and 3.



Return

the number of

colored cells

at the end of

n

minutes

.

Example 1:

Input:

n = 1

Output:

1

Explanation:

After 1 minute, there is only 1 blue cell, so we return 1.

Example 2:

Input:

n = 2

Output:

5

Explanation:

After 2 minutes, there are 4 colored cells on the boundary and 1 in the center, so we return 5.

Constraints:

1 <= n <= 10

5

Code Snippets

C++:

```
class Solution {
public:
long long coloredCells(int n) {

}
};
```

Java:

```
class Solution {
public long coloredCells(int n) {

}
}
```

Python3:

```
class Solution:
def coloredCells(self, n: int) -> int:
```

Python:

```
class Solution(object):
def coloredCells(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {number}
*/
var coloredCells = function(n) {

};
```

TypeScript:

```
function coloredCells(n: number): number {

};
```

C#:

```
public class Solution {
public long ColoredCells(int n) {

}
}
```

C:

```
long long coloredCells(int n) {

}
```

Go:

```
func coloredCells(n int) int64 {

}
```

Kotlin:

```
class Solution {
fun coloredCells(n: Int): Long {

}
}
```

Swift:

```
class Solution {
func coloredCells(_ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn colored_cells(n: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer}
def colored_cells(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function coloredCells($n) {

}
}
```

Dart:

```
class Solution {
int coloredCells(int n) {

}
}
```

Scala:

```
object Solution {
def coloredCells(n: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec colored_cells(n :: integer) :: integer
def colored_cells(n) do

end
end
```

Erlang:

```
-spec colored_cells(N :: integer()) -> integer().
colored_cells(N) ->
.
```

Racket:

```
(define/contract (colored-cells n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Total Number of Colored Cells
* 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:
long long coloredCells(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Count Total Number of Colored Cells
* 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 long coloredCells(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Total Number of Colored Cells
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 coloredCells(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def coloredCells(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Total Number of Colored Cells
* 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
* @return {number}
*/
var coloredCells = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Total Number of Colored Cells
* 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 coloredCells(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Count Total Number of Colored Cells
* 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 long ColoredCells(int n) {

}
}
```

C Solution:

```
/*
* Problem: Count Total Number of Colored Cells
* 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
*/

long long coloredCells(int n) {

}
```

Go Solution:

```
// Problem: Count Total Number of Colored Cells
// 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 coloredCells(n int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun coloredCells(n: Int): Long {

}
}
```

Swift Solution:

```
class Solution {
func coloredCells(_ n: Int) -> Int {
}
}
```

Rust Solution:

```
// Problem: Count Total Number of Colored Cells
// 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 colored_cells(n: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer}
def colored_cells(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function coloredCells($n) {

}
}
```

Dart Solution:

```
class Solution {
int coloredCells(int n) {

}
}
```

Scala Solution:

```
object Solution {
def coloredCells(n: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec colored_cells(n :: integer) :: integer
def colored_cells(n) do

end
end
```

Erlang Solution:

```
-spec colored_cells(N :: integer()) -> integer().
colored_cells(N) ->
.
```

Racket Solution:

```
(define/contract (colored-cells n)
(-> exact-integer? exact-integer?)
)
```