Problem 2201: Count Artifacts That Can Be Extracted

Problem Information

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

Problem Description

There is an

n x n

0-indexed

grid with some artifacts buried in it. You are given the integer

n

and a

0-indexed

2D integer array

artifacts

describing the positions of the rectangular artifacts where

artifacts[i] = [r1

i

, c1

i

, r2

i

, c2

i

]

denotes that the

i

th

artifact is buried in the subgrid where:

(r1

i

, c1

i

)

is the coordinate of the

top-left

cell of the

i

th

artifact and

(r2

i

, c2

i

)

is the coordinate of the

bottom-right

cell of the

i

th

artifact.

You will excavate some cells of the grid and remove all the mud from them. If the cell has a part of an artifact buried underneath, it will be uncovered. If all the parts of an artifact are uncovered, you can extract it.

Given a

0-indexed

2D integer array

dig

where

dig[i] = [r

i

, c

i

]

indicates that you will excavate the cell

(r

i

, c

i

)

, return

the number of artifacts that you can extract

.

The test cases are generated such that:

No two artifacts overlap.

Each artifact only covers at most

4

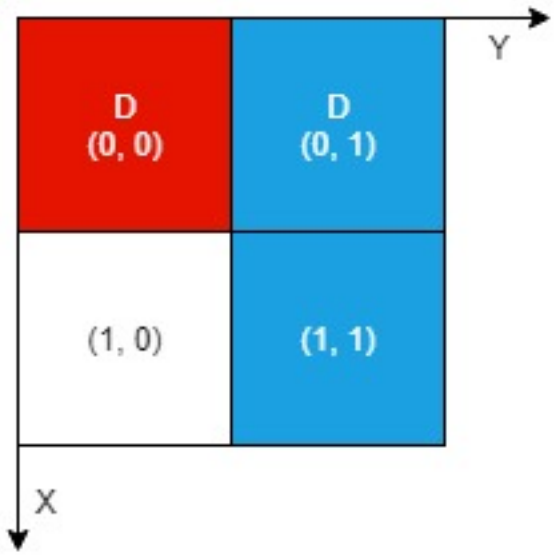
cells.

The entries of

dig

are unique.

Example 1:



Input:

n = 2, artifacts = [[0,0,0,0],[0,1,1,1]], dig = [[0,0],[0,1]]

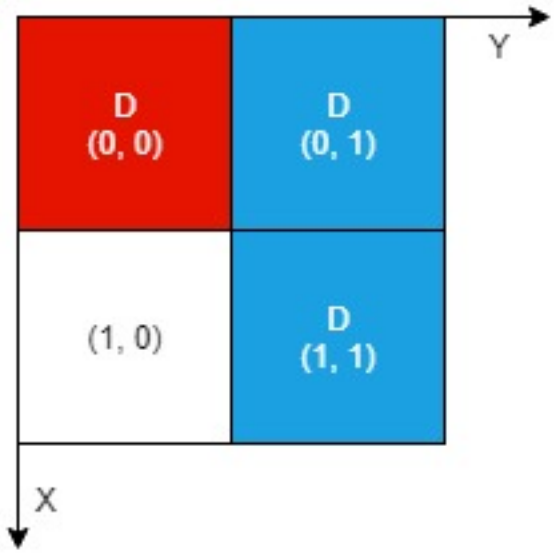
Output:

1

Explanation:

The different colors represent different artifacts. Excavated cells are labeled with a 'D' in the grid. There is 1 artifact that can be extracted, namely the red artifact. The blue artifact has one part in cell (1,1) which remains uncovered, so we cannot extract it. Thus, we return 1.

Example 2:



Input:

n = 2, artifacts = [[0,0,0,0],[0,1,1,1]], dig = [[0,0],[0,1],[1,1]]

Output:

2

Explanation:

Both the red and blue artifacts have all parts uncovered (labeled with a 'D') and can be extracted, so we return 2.

Constraints:

1 <= n <= 1000

1 <= artifacts.length, dig.length <= min(n

2

, 10

5

)

artifacts[i].length == 4

dig[i].length == 2

0 <= r1

i

, c1

i

, r2

i

, c2

i

, r

i

, c

i

<= n - 1

r1

i

<= r2

i

c1

i

<= c2

i

No two artifacts will overlap.

The number of cells covered by an artifact is

at most

4

.

The entries of

dig

are unique.

Code Snippets

C++:

```
class Solution {
public:
int digArtifacts(int n, vector<vector<int>>& artifacts, vector<vector<int>>&
dig) {

}
};
```

Java:

```
class Solution {
public int digArtifacts(int n, int[][] artifacts, int[][] dig) {

}
}
```

Python3:

```
class Solution:
def digArtifacts(self, n: int, artifacts: List[List[int]], dig:
List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def digArtifacts(self, n, artifacts, dig):
"""
:type n: int
:type artifacts: List[List[int]]
:type dig: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} artifacts
* @param {number[][]} dig
* @return {number}
*/
var digArtifacts = function(n, artifacts, dig) {

};
```

TypeScript:

```
function digArtifacts(n: number, artifacts: number[][], dig: number[][]):
number {

};
```

C#:

```
public class Solution {
public int DigArtifacts(int n, int[][] artifacts, int[][] dig) {

}
}
```

C:

```
int digArtifacts(int n, int** artifacts, int artifactsSize, int*
artifactsColSize, int** dig, int digSize, int* digColSize) {

}
```

Go:

```
func digArtifacts(n int, artifacts [][]int, dig [][]int) int {

}
```

Kotlin:

```
class Solution {
fun digArtifacts(n: Int, artifacts: Array<IntArray>, dig: Array<IntArray>):
Int {

}
}
```

Swift:

```
class Solution {
func digArtifacts(_ n: Int, _ artifacts: [[Int]], _ dig: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn dig_artifacts(n: i32, artifacts: Vec<Vec<i32>>, dig: Vec<Vec<i32>>) ->
i32 {

}
```

```
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} artifacts
# @param {Integer[][]} dig
# @return {Integer}
def dig_artifacts(n, artifacts, dig)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $artifacts
* @param Integer[][] $dig
* @return Integer
*/
function digArtifacts($n, $artifacts, $dig) {

}
}
```

Dart:

```
class Solution {
int digArtifacts(int n, List<List<int>> artifacts, List<List<int>> dig) {

}
}
```

Scala:

```
object Solution {
def digArtifacts(n: Int, artifacts: Array[Array[Int]], dig:
Array[Array[Int]]): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec dig_artifacts(n :: integer, artifacts :: [[integer]], dig ::
[[integer]]) :: integer
def dig_artifacts(n, artifacts, dig) do

end
end
```

Erlang:

```
-spec dig_artifacts(N :: integer(), Artifacts :: [[integer()]], Dig ::
[[integer()]]) -> integer().
dig_artifacts(N, Artifacts, Dig) ->
.
```

Racket:

```
(define/contract (dig-artifacts n artifacts dig)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Artifacts That Can Be Extracted
* 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 Solution {
public:
int digArtifacts(int n, vector<vector<int>>& artifacts, vector<vector<int>>&
dig) {

}
};
```

Java Solution:

```
/**
* Problem: Count Artifacts That Can Be Extracted
* 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 Solution {
public int digArtifacts(int n, int[][] artifacts, int[][] dig) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Artifacts That Can Be Extracted
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 Solution:
def digArtifacts(self, n: int, artifacts: List[List[int]], dig:
List[List[int]]) -> int:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def digArtifacts(self, n, artifacts, dig):
"""
:type n: int
:type artifacts: List[List[int]]
:type dig: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Artifacts That Can Be Extracted
* 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
*/

/**
* @param {number} n
* @param {number[][]} artifacts
* @param {number[][]} dig
* @return {number}
*/
var digArtifacts = function(n, artifacts, dig) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Artifacts That Can Be Extracted
* 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
*/

function digArtifacts(n: number, artifacts: number[][], dig: number[][]):
number {

};
```

C# Solution:

```
/*
* Problem: Count Artifacts That Can Be Extracted
* 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 Solution {
public int DigArtifacts(int n, int[][] artifacts, int[][] dig) {

}
}
```

C Solution:

```
/*
* Problem: Count Artifacts That Can Be Extracted
* 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
*/
```

```
int digArtifacts(int n, int** artifacts, int artifactsSize, int*
artifactsColSize, int** dig, int digSize, int* digColSize) {

}
```

Go Solution:

```
// Problem: Count Artifacts That Can Be Extracted
// 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

func digArtifacts(n int, artifacts [][]int, dig [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun digArtifacts(n: Int, artifacts: Array<IntArray>, dig: Array<IntArray>):
Int {

}
}
```

Swift Solution:

```
class Solution {
func digArtifacts(_ n: Int, _ artifacts: [[Int]], _ dig: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Artifacts That Can Be Extracted
// 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

impl Solution {
pub fn dig_artifacts(n: i32, artifacts: Vec<Vec<i32>>, dig: Vec<Vec<i32>>) ->
i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} artifacts
# @param {Integer[][]} dig
# @return {Integer}
def dig_artifacts(n, artifacts, dig)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $artifacts
* @param Integer[][] $dig
* @return Integer
*/
function digArtifacts($n, $artifacts, $dig) {

}
}
```

Dart Solution:

```
class Solution {
int digArtifacts(int n, List<List<int>> artifacts, List<List<int>> dig) {

}
}
```

Scala Solution:

```
object Solution {
def digArtifacts(n: Int, artifacts: Array[Array[Int]], dig:
Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec dig_artifacts(n :: integer, artifacts :: [[integer]], dig ::
[[integer]]) :: integer
def dig_artifacts(n, artifacts, dig) do

end
end
```

Erlang Solution:

```
-spec dig_artifacts(N :: integer(), Artifacts :: [[integer()]], Dig ::
[[integer()]]) -> integer().
dig_artifacts(N, Artifacts, Dig) ->
.
```

Racket Solution:

```
(define/contract (dig-artifacts n artifacts dig)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)) exact-integer?)
)
```