Problem 1001: Grid Illumination

Problem Information

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

Problem Description

There is a 2D

grid

of size

n x n

where each cell of this grid has a lamp that is initially

turned off

.

You are given a 2D array of lamp positions

lamps

, where

lamps[i] = [row

i

, col

i

]

indicates that the lamp at

grid[row

i

][col

i

]

is

turned on

. Even if the same lamp is listed more than once, it is turned on.

When a lamp is turned on, it

illuminates its cell

and

all other cells

in the same

row, column, or diagonal

.

You are also given another 2D array

queries

, where

queries[j] = [row

j

, col

j

]

. For the

j

th

query, determine whether

grid[row

j

][col

j

]

is illuminated or not. After answering the

j

th

query,

turn off

the lamp at

grid[row

j

][col

j

]

and its

8 adjacent lamps

if they exist. A lamp is adjacent if its cell shares either a side or corner with

grid[row

j

][col

j

]

.

Return

an array of integers

ans

,

where

ans[j]

should be

1

if the cell in the

j

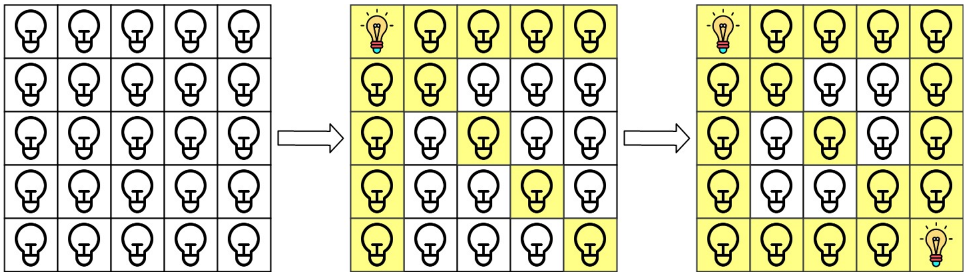
th

query was illuminated, or

0

if the lamp was not.

Example 1:



Input:

n = 5, lamps = [[0,0],[4,4]], queries = [[1,1],[1,0]]

Output:

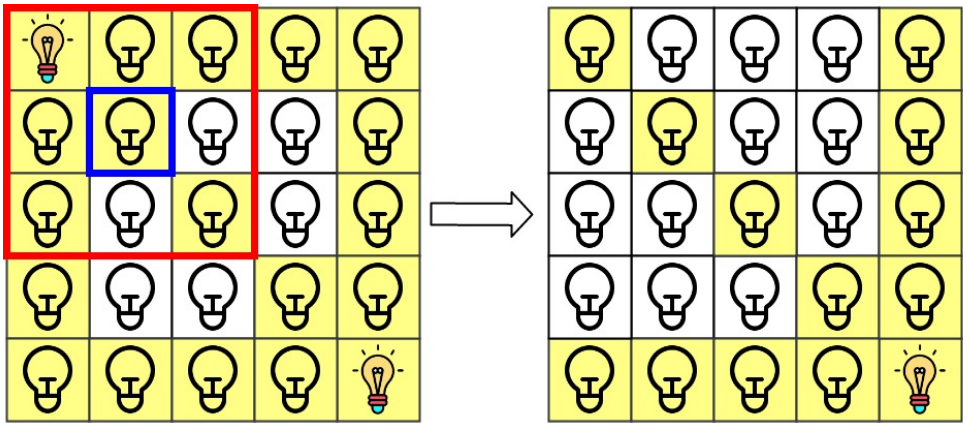
[1,0]

Explanation:

We have the initial grid with all lamps turned off. In the above picture we see the grid after turning on the lamp at grid[0][0] then turning on the lamp at grid[4][4]. The 0

th

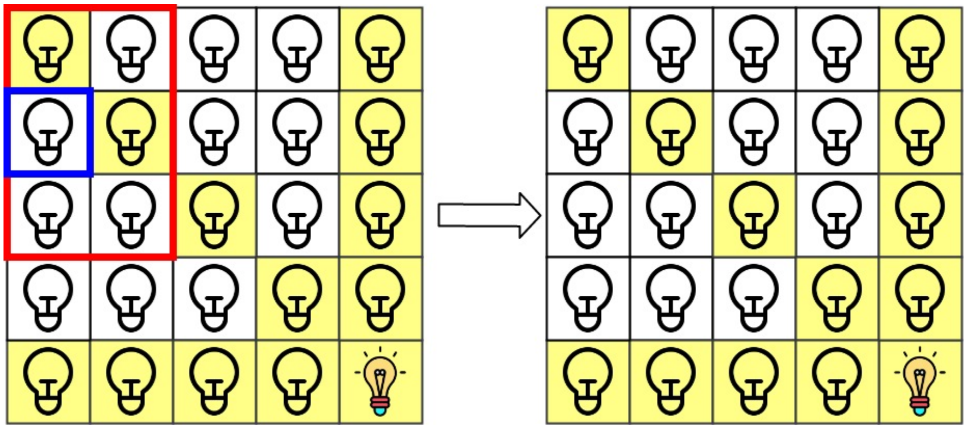
query asks if the lamp at grid[1][1] is illuminated or not (the blue square). It is illuminated, so set ans[0] = 1. Then, we turn off all lamps in the red square.



The 1

st

query asks if the lamp at grid[1][0] is illuminated or not (the blue square). It is not illuminated, so set ans[1] = 0. Then, we turn off all lamps in the red rectangle.



Example 2:

Input:

n = 5, lamps = [[0,0],[4,4]], queries = [[1,1],[1,1]] Output:

[1,1]

Example 3:

Input:

n = 5, lamps = [[0,0],[0,4]], queries = [[0,4],[0,1],[1,4]]

Output:

[1,1,0]

Constraints:

1 <= n <= 10

9

0 <= lamps.length <= 20000

0 <= queries.length <= 20000

lamps[i].length == 2

0 <= row

i

, col

i

< n

queries[j].length == 2

0 <= row

j

, col

j

< n

Code Snippets

C++:

```
class Solution {
public:
vector<int> gridIllumination(int n, vector<vector<int>>& lamps,
vector<vector<int>>& queries) {

}
};
```

Java:

```
class Solution {
public int[] gridIllumination(int n, int[][] lamps, int[][] queries) {

}
}
```

Python3:

```
class Solution:
def gridIllumination(self, n: int, lamps: List[List[int]], queries:
List[List[int]]) -> List[int]:
```

Python:

```
class Solution(object):
def gridIllumination(self, n, lamps, queries):
```

```
"""
:type n: int
:type lamps: List[List[int]]
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} lamps
* @param {number[][]} queries
* @return {number[]}
*/
var gridIllumination = function(n, lamps, queries) {

};
```

TypeScript:

```
function gridIllumination(n: number, lamps: number[][], queries: number[][]):
number[] {

};
```

C#:

```
public class Solution {
public int[] GridIllumination(int n, int[][] lamps, int[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* gridIllumination(int n, int** lamps, int lampsSize, int* lampsColSize,
int** queries, int queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go:

```
func gridIllumination(n int, lamps [][]int, queries [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun gridIllumination(n: Int, lamps: Array<IntArray>, queries:
Array<IntArray>): IntArray {

}
}
```

Swift:

```
class Solution {
func gridIllumination(_ n: Int, _ lamps: [[Int]], _ queries: [[Int]]) ->
[Int] {

}
}
```

Rust:

```
impl Solution {
pub fn grid_illumination(n: i32, lamps: Vec<Vec<i32>>, queries:
Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} lamps
# @param {Integer[][]} queries
# @return {Integer[]}
def grid_illumination(n, lamps, queries)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $lamps
* @param Integer[][] $queries
* @return Integer[]
*/
function gridIllumination($n, $lamps, $queries) {

}
}
```

Dart:

```
class Solution {
List<int> gridIllumination(int n, List<List<int>> lamps, List<List<int>>
queries) {

}
}
```

Scala:

```
object Solution {
def gridIllumination(n: Int, lamps: Array[Array[Int]], queries:
Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec grid_illumination(n :: integer, lamps :: [[integer]], queries ::
[[integer]]) :: [integer]
def grid_illumination(n, lamps, queries) do
```

```
end
end
```

Erlang:

```
-spec grid_illumination(N :: integer(), Lamps :: [[integer()]], Queries ::
[[integer()]]) -> [integer()].
grid_illumination(N, Lamps, Queries) ->
.
```

Racket:

```
(define/contract (grid-illumination n lamps queries)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Grid Illumination
* Difficulty: Hard
* 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:
vector<int> gridIllumination(int n, vector<vector<int>>& lamps,
vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Grid Illumination
* Difficulty: Hard
* 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[] gridIllumination(int n, int[][] lamps, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Grid Illumination
Difficulty: Hard
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 gridIllumination(self, n: int, lamps: List[List[int]], queries:
List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def gridIllumination(self, n, lamps, queries):
"""
:type n: int
:type lamps: List[List[int]]
```

```
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Grid Illumination
* Difficulty: Hard
* 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[][]} lamps
* @param {number[][]} queries
* @return {number[]}
*/
var gridIllumination = function(n, lamps, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Grid Illumination
* Difficulty: Hard
* 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 gridIllumination(n: number, lamps: number[][], queries: number[][]):
number[] {
};
```

C# Solution:

```
/*
* Problem: Grid Illumination
* Difficulty: Hard
* 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[] GridIllumination(int n, int[][] lamps, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Grid Illumination
* Difficulty: Hard
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* gridIllumination(int n, int** lamps, int lampsSize, int* lampsColSize,
int** queries, int queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Grid Illumination
// Difficulty: Hard
// 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 gridIllumination(n int, lamps [][]int, queries [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun gridIllumination(n: Int, lamps: Array<IntArray>, queries:
Array<IntArray>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func gridIllumination(_ n: Int, _ lamps: [[Int]], _ queries: [[Int]]) ->
[Int] {

}
}
```

Rust Solution:

```
// Problem: Grid Illumination
// Difficulty: Hard
// 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 grid_illumination(n: i32, lamps: Vec<Vec<i32>>, queries:
```

```
Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} lamps
# @param {Integer[][]} queries
# @return {Integer[]}
def grid_illumination(n, lamps, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $lamps
* @param Integer[][] $queries
* @return Integer[]
*/
function gridIllumination($n, $lamps, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> gridIllumination(int n, List<List<int>> lamps, List<List<int>>
queries) {

}
}
```

Scala Solution:

```
object Solution {
def gridIllumination(n: Int, lamps: Array[Array[Int]], queries:
Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec grid_illumination(n :: integer, lamps :: [[integer]], queries ::
[[integer]]) :: [integer]
def grid_illumination(n, lamps, queries) do

end
end
```

Erlang Solution:

```
-spec grid_illumination(N :: integer(), Lamps :: [[integer()]], Queries ::
[[integer()]]) -> [integer()].
grid_illumination(N, Lamps, Queries) ->
.
```

Racket Solution:

```
(define/contract (grid-illumination n lamps queries)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)) (listof exact-integer?))
)
```