Problem 3624: Number of Integers With Popcount-Depth Equal to K II

Problem Information

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

Problem Description

You are given an integer array

nums

.

For any positive integer

x

, define the following sequence:

p

0

= x

p

i+1

= popcount(p

i

)

for all

i >= 0

, where

popcount(y)

is the number of set bits (1's) in the binary representation of

y

.

This sequence will eventually reach the value 1.

The

popcount-depth

of

x

is defined as the

smallest

integer

d >= 0

such that

p

d

= 1

.

For example, if

x = 7

(binary representation

"111"

). Then, the sequence is:

$7 \to 3 \to 2 \to 1$

, so the popcount-depth of 7 is 3.

You are also given a 2D integer array

queries

, where each

queries[i]

is either:

[1, l, r, k]

-

Determine

the number of indices

j

such that

l <= j <= r

and the

popcount-depth

of

nums[j]

is equal to

k

.

[2, idx, val]

-

Update

nums[idx]

to

val

.

Return an integer array

answer

, where

answer[i]

is the number of indices for the

i

th

query of type

[1, l, r, k]

.

Example 1:

Input:

nums = [2,4], queries = [[1,0,1,1],[2,1,1],[1,0,1,0]]

Output:

[2,1]

Explanation:

i

queries[i]

nums

binary(

nums

)

popcount-

depth

[l, r]

k

Valid

nums[j]

updated

nums

Answer

0

[1,0,1,1]

[2,4]

[10, 100]

[1, 1]

[0, 1]

1

[0, 1]

—

2

1

[2,1,1]

[2,4]

[10, 100]

[1, 1]

—

—

—

[2,1]

—

2

[1,0,1,0]

[2,1]

[10, 1]

[1, 0]

[0, 1]

0

[1]

—

1

Thus, the final

answer

is

[2, 1]

.

Example 2:

Input:

nums = [3,5,6], queries = [[1,0,2,2],[2,1,4],[1,1,2,1],[1,0,1,0]]

Output:

[3,1,0]

Explanation:

i

queries[i]

nums

binary(

nums

)

popcount-

depth

[l, r]

k

Valid

nums[j]

updated

nums

Answer

0

[1,0,2,2]

[3, 5, 6]

[11, 101, 110]

[2, 2, 2]

[0, 2]

2

[0, 1, 2]

—

3

1

[2,1,4]

[3, 5, 6]

[11, 101, 110]

[2, 2, 2]

—

—

—

[3, 4, 6]

—

2

[1,1,2,1]

[3, 4, 6]

[11, 100, 110]

[2, 1, 2]

[1, 2]

1

[1]

—

1

3

[1,0,1,0]

[3, 4, 6]

[11, 100, 110]

[2, 1, 2]

[0, 1]

0

[]

—

0

Thus, the final

answer

is

[3, 1, 0]

.

Example 3:

Input:

nums = [1,2], queries = [[1,0,1,1],[2,0,3],[1,0,0,1],[1,0,0,2]]

Output:

[1,0,1]

Explanation:

i

queries[i]

nums

binary(

nums

)

popcount-

depth

[l, r]

k

Valid

nums[j]

updated

nums

Answer

0

[1,0,1,1]

[1, 2]

[1, 10]

[0, 1]

[0, 1]

1

[1]

—

1

1

[2,0,3]

[1, 2]

[1, 10]

[0, 1]

—

—

—

[3, 2]

2

[1,0,0,1]

[3, 2]

[11, 10]

[2, 1]

[0, 0]

1

[]

—

0

3

[1,0,0,2]

[3, 2]

[11, 10]

[2, 1]

[0, 0]

2

[0]

—

1

Thus, the final

answer

is

[1, 0, 1]

.

Constraints:

1 <= n == nums.length <= 10

5

1 <= nums[i] <= 10

15

1 <= queries.length <= 10

5

queries[i].length == 3

or

4

queries[i] == [1, l, r, k]

or,

queries[i] == [2, idx, val]

0 <= l <= r <= n - 1

0 <= k <= 5

0 <= idx <= n - 1

1 <= val <= 10

15

Code Snippets

C++:

```
class Solution {
public:
vector<int> popcountDepth(vector<long long>& nums, vector<vector<long long>>&
queries) {

}
};
```

Java:

```
class Solution {
public int[] popcountDepth(long[] nums, long[][] queries) {

}
}
```

Python3:

```
class Solution:
def popcountDepth(self, nums: List[int], queries: List[List[int]]) ->
List[int]:
```

Python:

```
class Solution(object):
def popcountDepth(self, nums, queries):
"""
:type nums: List[int]
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number[][]} queries
* @return {number[]}
*/
var popcountDepth = function(nums, queries) {

};
```

TypeScript:

```
function popcountDepth(nums: number[], queries: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] PopcountDepth(long[] nums, long[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* popcountDepth(long long* nums, int numsSize, long long** queries, int
queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go:

```
func popcountDepth(nums []int64, queries [][]int64) []int {

}
```

Kotlin:

```
class Solution {
fun popcountDepth(nums: LongArray, queries: Array<LongArray>): IntArray {

}
}
```

Swift:

```
class Solution {
func popcountDepth(_ nums: [Int], _ queries: [[Int]]) -> [Int] {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn popcount_depth(nums: Vec<i64>, queries: Vec<Vec<i64>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[][]} queries
# @return {Integer[]}
def popcount_depth(nums, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[][] $queries
* @return Integer[]
*/
function popcountDepth($nums, $queries) {

}
}
```

Dart:

```
class Solution {
List<int> popcountDepth(List<int> nums, List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def popcountDepth(nums: Array[Long], queries: Array[Array[Long]]): Array[Int]
= {

}
}
```

Elixir:

```
defmodule Solution do
@spec popcount_depth(nums :: [integer], queries :: [[integer]]) :: [integer]
def popcount_depth(nums, queries) do

end
end
```

Erlang:

```
-spec popcount_depth(Nums :: [integer()], Queries :: [[integer()]]) ->
[integer()].
popcount_depth(Nums, Queries) ->
.
```

Racket:

```
(define/contract (popcount-depth nums queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Integers With Popcount-Depth Equal to K II
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
vector<int> popcountDepth(vector<long long>& nums, vector<vector<long long>>&
queries) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Integers With Popcount-Depth Equal to K II
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int[] popcountDepth(long[] nums, long[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Integers With Popcount-Depth Equal to K II
Difficulty: Hard
Tags: array, tree

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""
```

```
class Solution:
def popcountDepth(self, nums: List[int], queries: List[List[int]]) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def popcountDepth(self, nums, queries):
"""
:type nums: List[int]
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Integers With Popcount-Depth Equal to K II
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number[]} nums
* @param {number[][]} queries
* @return {number[]}
*/
var popcountDepth = function(nums, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Integers With Popcount-Depth Equal to K II
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function popcountDepth(nums: number[], queries: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Number of Integers With Popcount-Depth Equal to K II
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int[] PopcountDepth(long[] nums, long[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Number of Integers With Popcount-Depth Equal to K II
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* popcountDepth(long long* nums, int numsSize, long long** queries, int
queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Number of Integers With Popcount-Depth Equal to K II
// Difficulty: Hard
// Tags: array, tree
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func popcountDepth(nums []int64, queries [][]int64) []int {

}
```

Kotlin Solution:

```
class Solution {
fun popcountDepth(nums: LongArray, queries: Array<LongArray>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func popcountDepth(_ nums: [Int], _ queries: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Number of Integers With Popcount-Depth Equal to K II
// Difficulty: Hard
// Tags: array, tree
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn popcount_depth(nums: Vec<i64>, queries: Vec<Vec<i64>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[][]} queries
# @return {Integer[]}
def popcount_depth(nums, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[][] $queries
* @return Integer[]
*/
function popcountDepth($nums, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> popcountDepth(List<int> nums, List<List<int>> queries) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def popcountDepth(nums: Array[Long], queries: Array[Array[Long]]): Array[Int]
= {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec popcount_depth(nums :: [integer], queries :: [[integer]]) :: [integer]
def popcount_depth(nums, queries) do

end
end
```

Erlang Solution:

```
-spec popcount_depth(Nums :: [integer()], Queries :: [[integer()]]) ->
[integer()].
popcount_depth(Nums, Queries) ->
.
```

Racket Solution:

```
(define/contract (popcount-depth nums queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```