Problem 898: Bitwise ORs of Subarrays

Problem Information

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

Problem Description

Given an integer array

arr

, return

the number of distinct bitwise ORs of all the non-empty subarrays of

arr

.

The bitwise OR of a subarray is the bitwise OR of each integer in the subarray. The bitwise OR of a subarray of one integer is that integer.

A

subarray

is a contiguous non-empty sequence of elements within an array.

Example 1:

Input:

arr = [0]

Output:

1

Explanation:

There is only one possible result: 0.

Example 2:

Input:

arr = [1,1,2]

Output:

3

Explanation:

The possible subarrays are [1], [1], [2], [1, 1], [1, 2], [1, 1, 2]. These yield the results 1, 1, 2, 1, 3, 3. There are 3 unique values, so the answer is 3.

Example 3:

Input:

arr = [1,2,4]

Output:

6

Explanation:

The possible results are 1, 2, 3, 4, 6, and 7.

Constraints:

1 <= arr.length <= 5 * 10

4

0 <= arr[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int subarrayBitwiseORs(vector<int>& arr) {

}
};
```

Java:

```
class Solution {
public int subarrayBitwiseORs(int[] arr) {

}
}
```

Python3:

```
class Solution:
def subarrayBitwiseORs(self, arr: List[int]) -> int:
```

Python:

```
class Solution(object):
def subarrayBitwiseORs(self, arr):
"""
:type arr: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} arr
* @return {number}
*/
var subarrayBitwiseORs = function(arr) {

};
```

TypeScript:

```
function subarrayBitwiseORs(arr: number[]): number {

};
```

C#:

```
public class Solution {
public int SubarrayBitwiseORs(int[] arr) {

}
}
```

C:

```
int subarrayBitwiseORs(int* arr, int arrSize) {

}
```

Go:

```
func subarrayBitwiseORs(arr []int) int {

}
```

Kotlin:

```
class Solution {
fun subarrayBitwiseORs(arr: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func subarrayBitwiseORs(_ arr: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn subarray_bitwise_o_rs(arr: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @return {Integer}
def subarray_bitwise_o_rs(arr)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer
*/
function subarrayBitwiseORs($arr) {

}
}
```

Dart:

```
class Solution {
int subarrayBitwiseORs(List<int> arr) {

}
```

```
}
```

Scala:

```
object Solution {
def subarrayBitwiseORs(arr: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec subarray_bitwise_o_rs(arr :: [integer]) :: integer
def subarray_bitwise_o_rs(arr) do

end
end
```

Erlang:

```
-spec subarray_bitwise_o_rs(Arr :: [integer()]) -> integer().
subarray_bitwise_o_rs(Arr) ->
.
```

Racket:

```
(define/contract (subarray-bitwise-o-rs arr)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Bitwise ORs of Subarrays
* Difficulty: Medium
* Tags: array, dp
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int subarrayBitwiseORs(vector<int>& arr) {

}
};
```

Java Solution:

```
/**
* Problem: Bitwise ORs of Subarrays
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int subarrayBitwiseORs(int[] arr) {

}
}
```

Python3 Solution:

```
"""
Problem: Bitwise ORs of Subarrays
Difficulty: Medium
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""
```

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

Python Solution:

```
class Solution(object):
def subarrayBitwiseORs(self, arr):
"""
:type arr: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Bitwise ORs of Subarrays
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} arr
* @return {number}
*/
var subarrayBitwiseORs = function(arr) {

};
```

TypeScript Solution:

```
/**
* Problem: Bitwise ORs of Subarrays
* Difficulty: Medium
* Tags: array, dp
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function subarrayBitwiseORs(arr: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Bitwise ORs of Subarrays
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int SubarrayBitwiseORs(int[] arr) {

}
}
```

C Solution:

```
/*
* Problem: Bitwise ORs of Subarrays
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int subarrayBitwiseORs(int* arr, int arrSize) {
```

```
}
```

Go Solution:

```
// Problem: Bitwise ORs of Subarrays
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func subarrayBitwiseORs(arr []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun subarrayBitwiseORs(arr: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func subarrayBitwiseORs(_ arr: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Bitwise ORs of Subarrays
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
```

```
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn subarray_bitwise_o_rs(arr: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @return {Integer}
def subarray_bitwise_o_rs(arr)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer
*/
function subarrayBitwiseORs($arr) {

}
}
```

Dart Solution:

```
class Solution {
int subarrayBitwiseORs(List<int> arr) {

}
}
```

Scala Solution:

```
object Solution {
def subarrayBitwiseORs(arr: Array[Int]): Int = {
```

```
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec subarray_bitwise_o_rs(arr :: [integer]) :: integer
def subarray_bitwise_o_rs(arr) do

end
end
```

Erlang Solution:

```
-spec subarray_bitwise_o_rs(Arr :: [integer()]) -> integer().
subarray_bitwise_o_rs(Arr) ->
.
```

Racket Solution:

```
(define/contract (subarray-bitwise-o-rs arr)
(-> (listof exact-integer?) exact-integer?)
)
```