

Problem 3314: Construct the Minimum Bitwise Array I

Problem Information

Difficulty: [Easy](#)

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given an array

nums

consisting of

n

prime

integers.

You need to construct an array

ans

of length

n

, such that, for each index

i

, the bitwise

OR

of

$\text{ans}[i]$

and

$\text{ans}[i] + 1$

is equal to

$\text{nums}[i]$

, i.e.

$\text{ans}[i] \text{ OR } (\text{ans}[i] + 1) == \text{nums}[i]$

Additionally, you must

minimize

each value of

$\text{ans}[i]$

in the resulting array.

If it is

not possible

to find such a value for

$\text{ans}[i]$

that satisfies the

condition

, then set

`ans[i] = -1`

Example 1:

Input:

`nums = [2,3,5,7]`

Output:

`[-1,1,4,3]`

Explanation:

For

`i = 0`

, as there is no value for

`ans[0]`

that satisfies

`ans[0]` OR `(ans[0] + 1) = 2`

, so

`ans[0] = -1`

For

i = 1

, the smallest

ans[1]

that satisfies

ans[1] OR (ans[1] + 1) = 3

is

1

, because

1 OR (1 + 1) = 3

.

For

i = 2

, the smallest

ans[2]

that satisfies

ans[2] OR (ans[2] + 1) = 5

is

4

, because

$$4 \text{ OR } (4 + 1) = 5$$

.

For

$$i = 3$$

, the smallest

$$\text{ans}[3]$$

that satisfies

$$\text{ans}[3] \text{ OR } (\text{ans}[3] + 1) = 7$$

is

$$3$$

, because

$$3 \text{ OR } (3 + 1) = 7$$

.

Example 2:

Input:

$$\text{nums} = [11, 13, 31]$$

Output:

$$[9, 12, 15]$$

Explanation:

For

$i = 0$

, the smallest

$\text{ans}[0]$

that satisfies

$$\text{ans}[0] \text{ OR } (\text{ans}[0] + 1) = 11$$

is

9

, because

$$9 \text{ OR } (9 + 1) = 11$$

.

For

$i = 1$

, the smallest

$\text{ans}[1]$

that satisfies

$$\text{ans}[1] \text{ OR } (\text{ans}[1] + 1) = 13$$

is

12

, because

$$12 \text{ OR } (12 + 1) = 13$$

.

For

$$i = 2$$

, the smallest

$$\text{ans}[2]$$

that satisfies

$$\text{ans}[2] \text{ OR } (\text{ans}[2] + 1) = 31$$

is

$$15$$

, because

$$15 \text{ OR } (15 + 1) = 31$$

.

Constraints:

$$1 \leq \text{nums.length} \leq 100$$

$$2 \leq \text{nums}[i] \leq 1000$$

$$\text{nums}[i]$$

is a prime number.

Code Snippets

C++:

```
class Solution {  
public:  
vector<int> minBitwiseArray(vector<int>& nums) {  
  
}  
};
```

Java:

```
class Solution {  
public int[] minBitwiseArray(List<Integer> nums) {  
  
}  
}
```

Python3:

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

Python:

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

JavaScript:

```
/**  
 * @param {number[]} nums  
 * @return {number[]}  
 */  
var minBitwiseArray = function(nums) {  
  
};
```

TypeScript:

```
function minBitwiseArray(nums: number[]): number[] {  
}  
};
```

C#:

```
public class Solution {  
    public int[] MinBitwiseArray(IList<int> nums) {  
          
    }  
}
```

C:

```
/**  
 * Note: The returned array must be malloced, assume caller calls free().  
 */  
int* minBitwiseArray(int* nums, int numSize, int* returnSize) {  
  
}
```

Go:

```
func minBitwiseArray(nums []int) []int {  
  
}
```

Kotlin:

```
class Solution {  
    fun minBitwiseArray(nums: List<Int>): IntArray {  
          
    }  
}
```

Swift:

```
class Solution {  
    func minBitwiseArray(_ nums: [Int]) -> [Int] {  
  
}
```

```
}
```

Rust:

```
impl Solution {
    pub fn min_bitwise_array(nums: Vec<i32>) -> Vec<i32> {
        }
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer[]}
def min_bitwise_array(nums)

end
```

PHP:

```
class Solution {

    /**
     * @param Integer[] $nums
     * @return Integer[]
     */
    function minBitwiseArray($nums) {

    }
}
```

Dart:

```
class Solution {
    List<int> minBitwiseArray(List<int> nums) {
        }
}
```

Scala:

```
object Solution {  
    def minBitwiseArray(nums: List[Int]): Array[Int] = {  
        }  
        }  
}
```

Elixir:

```
defmodule Solution do  
  @spec min_bitwise_array(nums :: [integer]) :: [integer]  
  def min_bitwise_array(nums) do  
  
  end  
  end
```

Erlang:

```
-spec min_bitwise_array(Nums :: [integer()]) -> [integer()].  
min_bitwise_array(Nums) ->  
.
```

Racket:

```
(define/contract (min-bitwise-array nums)  
  (-> (listof exact-integer?) (listof exact-integer?))  
)
```

Solutions

C++ Solution:

```
/*  
 * Problem: Construct the Minimum Bitwise Array I  
 * Difficulty: Easy  
 * Tags: array  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(1) to O(n) depending on approach  
 */
```

```
class Solution {  
public:  
vector<int> minBitwiseArray(vector<int>& nums) {  
  
}  
};
```

Java Solution:

```
/**  
* Problem: Construct the Minimum Bitwise Array I  
* Difficulty: Easy  
* Tags: array  
*  
* Approach: Use two pointers or sliding window technique  
* Time Complexity: O(n) or O(n log n)  
* Space Complexity: O(1) to O(n) depending on approach  
*/  
  
class Solution {  
public int[] minBitwiseArray(List<Integer> nums) {  
  
}  
}
```

Python3 Solution:

```
"""  
Problem: Construct the Minimum Bitwise Array I  
Difficulty: Easy  
Tags: array  
  
Approach: Use two pointers or sliding window technique  
Time Complexity: O(n) or O(n log n)  
Space Complexity: O(1) to O(n) depending on approach  
"""  
  
class Solution:  
def minBitwiseArray(self, nums: List[int]) -> List[int]:  
# TODO: Implement optimized solution  
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
 * Problem: Construct the Minimum Bitwise Array I
 * Difficulty: Easy
 * Tags: array
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

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

};
```

TypeScript Solution:

```
/**
 * Problem: Construct the Minimum Bitwise Array I
 * Difficulty: Easy
 * Tags: array
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

function minBitwiseArray(nums: number[]): number[] {
```

```
};
```

C# Solution:

```
/*
 * Problem: Construct the Minimum Bitwise Array I
 * Difficulty: Easy
 * Tags: array
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

public class Solution {
    public int[] MinBitwiseArray(IList<int> nums) {
        return null;
    }
}
```

C Solution:

```
/*
 * Problem: Construct the Minimum Bitwise Array I
 * Difficulty: Easy
 * Tags: array
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* minBitwiseArray(int* nums, int numsSize, int* returnSize) {
    return NULL;
}
```

Go Solution:

```

// Problem: Construct the Minimum Bitwise Array I
// Difficulty: Easy
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func minBitwiseArray(nums []int) []int {
}

```

Kotlin Solution:

```

class Solution {
    fun minBitwiseArray(nums: List<Int>): IntArray {
        }
    }

```

Swift Solution:

```

class Solution {
    func minBitwiseArray(_ nums: [Int]) -> [Int] {
        }
    }

```

Rust Solution:

```

// Problem: Construct the Minimum Bitwise Array I
// Difficulty: Easy
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
    pub fn min_bitwise_array(nums: Vec<i32>) -> Vec<i32> {
    }
}

```

```
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer[]}
def min_bitwise_array(nums)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[] $nums
     * @return Integer[]
     */
    function minBitwiseArray($nums) {

    }
}
```

Dart Solution:

```
class Solution {
List<int> minBitwiseArray(List<int> nums) {

}
```

Scala Solution:

```
object Solution {
def minBitwiseArray(nums: List[Int]): Array[Int] = {

}
```

Elixir Solution:

```
defmodule Solution do
@spec min_bitwise_array(nums :: [integer]) :: [integer]
def min_bitwise_array(nums) do

end
end
```

Erlang Solution:

```
-spec min_bitwise_array(Nums :: [integer()]) -> [integer()].
min_bitwise_array(Nums) ->
.
```

Racket Solution:

```
(define/contract (min-bitwise-array nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```