Problem 1969: Minimum Non-Zero Product of the Array Elements

Problem Information

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

Problem Description

You are given a positive integer

p

. Consider an array

nums

(

1-indexed

) that consists of the integers in the

inclusive

range

[1, 2

p

- 1]

in their binary representations. You are allowed to do the following operation

any

number of times:

Choose two elements

x

and

y

from

nums

.

Choose a bit in

x

and swap it with its corresponding bit in

y

. Corresponding bit refers to the bit that is in the

same position

in the other integer.

For example, if

x = 11

0

1

and

y = 00

1

1

, after swapping the

2

nd

bit from the right, we have

x = 11

1

1

and

y = 00

0

1

.

Find the

minimum non-zero

product of

nums

after performing the above operation

any

number of times. Return

this product

modulo

10

9

+ 7

.

Note:

The answer should be the minimum product

before

the modulo operation is done.

Example 1:

Input:

p = 1

Output:

1

Explanation:

nums = [1]. There is only one element, so the product equals that element.

Example 2:

Input:

p = 2

Output:

6

Explanation:

nums = [01, 10, 11]. Any swap would either make the product 0 or stay the same. Thus, the array product of 1 * 2 * 3 = 6 is already minimized.

Example 3:

Input:

p = 3

Output:

1512

Explanation:

nums = [001, 010, 011, 100, 101, 110, 111] - In the first operation we can swap the leftmost bit of the second and fifth elements. - The resulting array is [001,

1

10, 011, 100,

0

01, 110, 111]. - In the second operation we can swap the middle bit of the third and fourth elements. - The resulting array is [001, 110, 0

0

1, 1

1

0, 001, 110, 111]. The array product is 1 * 6 * 1 * 6 * 1 * 6 * 7 = 1512, which is the minimum possible product.

Constraints:

1 <= p <= 60

Code Snippets

C++:

```
class Solution {
public:
int minNonZeroProduct(int p) {

}
};
```

Java:

```
class Solution {
public int minNonZeroProduct(int p) {

}
}
```

Python3:

```
class Solution:
def minNonZeroProduct(self, p: int) -> int:
```

Python:

```
class Solution(object):
def minNonZeroProduct(self, p):
"""
:type p: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} p
* @return {number}
*/
var minNonZeroProduct = function(p) {

};
```

TypeScript:

```
function minNonZeroProduct(p: number): number {

};
```

C#:

```
public class Solution {
public int MinNonZeroProduct(int p) {

}
}
```

C:

```
int minNonZeroProduct(int p) {

}
```

Go:

```
func minNonZeroProduct(p int) int {
```

```
}
```

Kotlin:

```
class Solution {
fun minNonZeroProduct(p: Int): Int {

}
}
```

Swift:

```
class Solution {
func minNonZeroProduct(_ p: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_non_zero_product(p: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} p
# @return {Integer}
def min_non_zero_product(p)

end
```

PHP:

```
class Solution {

/**
* @param Integer $p
* @return Integer
*/
```

```
function minNonZeroProduct($p) {

}
}
```

Dart:

```
class Solution {
int minNonZeroProduct(int p) {

}
}
```

Scala:

```
object Solution {
def minNonZeroProduct(p: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_non_zero_product(p :: integer) :: integer
def min_non_zero_product(p) do

end
end
```

Erlang:

```
-spec min_non_zero_product(P :: integer()) -> integer().
min_non_zero_product(P) ->
.
```

Racket:

```
(define/contract (min-non-zero-product p)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Non-Zero Product of the Array Elements
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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 minNonZeroProduct(int p) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Non-Zero Product of the Array Elements
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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 minNonZeroProduct(int p) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Non-Zero Product of the Array Elements
Difficulty: Medium
Tags: array, greedy, math

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 minNonZeroProduct(self, p: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minNonZeroProduct(self, p):
"""
:type p: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Non-Zero Product of the Array Elements
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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} p
* @return {number}
*/
var minNonZeroProduct = function(p) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Minimum Non-Zero Product of the Array Elements
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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 minNonZeroProduct(p: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Non-Zero Product of the Array Elements
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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 MinNonZeroProduct(int p) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Non-Zero Product of the Array Elements
* Difficulty: Medium
```

```
* Tags: array, greedy, math
*
* 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
*/

int minNonZeroProduct(int p) {

}
```

Go Solution:

```
// Problem: Minimum Non-Zero Product of the Array Elements
// Difficulty: Medium
// Tags: array, greedy, math
//
// 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 minNonZeroProduct(p int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minNonZeroProduct(p: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minNonZeroProduct(_ p: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Non-Zero Product of the Array Elements
// Difficulty: Medium
// Tags: array, greedy, math
//
// 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_non_zero_product(p: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} p
# @return {Integer}
def min_non_zero_product(p)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $p
* @return Integer
*/
function minNonZeroProduct($p) {

}
}
```

Dart Solution:

```
class Solution {
int minNonZeroProduct(int p) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def minNonZeroProduct(p: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_non_zero_product(p :: integer) :: integer
def min_non_zero_product(p) do

end
end
```

Erlang Solution:

```
-spec min_non_zero_product(P :: integer()) -> integer().
min_non_zero_product(P) ->
.
```

Racket Solution:

```
(define/contract (min-non-zero-product p)
(-> exact-integer? exact-integer?)
)
```