Problem 1018: Binary Prefix Divisible By 5

Problem Information

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

Problem Description

You are given a binary array

nums

(

0-indexed

).

We define

x

i

as the number whose binary representation is the subarray

nums[0..i]

(from most-significant-bit to least-significant-bit).

For example, if

nums = [1,0,1]

, then

x

0

= 1

,

x

1

= 2

, and

x

2

= 5

.

Return

an array of booleans

answer

where

answer[i]

is

true

if

x

i

is divisible by

5

.

Example 1:

Input:

nums = [0,1,1]

Output:

[true,false,false]

Explanation:

The input numbers in binary are 0, 01, 011; which are 0, 1, and 3 in base-10. Only the first number is divisible by 5, so answer[0] is true.

Example 2:

Input:

nums = [1,1,1]

Output:

[false,false,false]

Constraints:

1 <= nums.length <= 10

5

nums[i]

is either

0

or

1

.

Code Snippets

C++:

```
class Solution {
public:
vector<bool> prefixesDivBy5(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public List<Boolean> prefixesDivBy5(int[] nums) {

}
}
```

Python3:

```
class Solution:
def prefixesDivBy5(self, nums: List[int]) -> List[bool]:
```

Python:

```
class Solution(object):
def prefixesDivBy5(self, nums):
"""
:type nums: List[int]
:rtype: List[bool]
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @return {boolean[]}
*/
var prefixesDivBy5 = function(nums) {

};
```

TypeScript:

```
function prefixesDivBy5(nums: number[]): boolean[] {

};
```

C#:

```
public class Solution {
public IList<bool> PrefixesDivBy5(int[] nums) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
bool* prefixesDivBy5(int* nums, int numsSize, int* returnSize) {

}
```

Go:

```
func prefixesDivBy5(nums []int) []bool {

}
```

Kotlin:

```
class Solution {
fun prefixesDivBy5(nums: IntArray): List<Boolean> {

}
}
```

Swift:

```
class Solution {
func prefixesDivBy5(_ nums: [Int]) -> [Bool] {

}
}
```

Rust:

```
impl Solution {
pub fn prefixes_div_by5(nums: Vec<i32>) -> Vec<bool> {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Boolean[]}
def prefixes_div_by5(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Boolean[]
*/
function prefixesDivBy5($nums) {

}
}
```

Dart:

```
class Solution {
List<bool> prefixesDivBy5(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def prefixesDivBy5(nums: Array[Int]): List[Boolean] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec prefixes_div_by5(nums :: [integer]) :: [boolean]
def prefixes_div_by5(nums) do

end
end
```

Erlang:

```
-spec prefixes_div_by5(Nums :: [integer()]) -> [boolean()].
prefixes_div_by5(Nums) ->
.
```

Racket:

```
(define/contract (prefixes-div-by5 nums)
(-> (listof exact-integer?) (listof boolean?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Binary Prefix Divisible By 5
* 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<bool> prefixesDivBy5(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Binary Prefix Divisible By 5
* 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 List<Boolean> prefixesDivBy5(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Binary Prefix Divisible By 5
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 prefixesDivBy5(self, nums: List[int]) -> List[bool]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def prefixesDivBy5(self, nums):
"""
:type nums: List[int]
:rtype: List[bool]
"""
```

JavaScript Solution:

```
/**
* Problem: Binary Prefix Divisible By 5
* 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 {boolean[]}
*/
var prefixesDivBy5 = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Binary Prefix Divisible By 5
* 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 prefixesDivBy5(nums: number[]): boolean[] {

};
```

C# Solution:

```
/*
* Problem: Binary Prefix Divisible By 5
* 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 IList<bool> PrefixesDivBy5(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Binary Prefix Divisible By 5
* 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().
*/
bool* prefixesDivBy5(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Binary Prefix Divisible By 5
// 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 prefixesDivBy5(nums []int) []bool {

}
```

Kotlin Solution:

```
class Solution {
fun prefixesDivBy5(nums: IntArray): List<Boolean> {

}
}
```

Swift Solution:

```
class Solution {
func prefixesDivBy5(_ nums: [Int]) -> [Bool] {

}
}
```

Rust Solution:

```
// Problem: Binary Prefix Divisible By 5
// 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 prefixes_div_by5(nums: Vec<i32>) -> Vec<bool> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Boolean[]}
def prefixes_div_by5(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Boolean[]
*/
function prefixesDivBy5($nums) {

}
}
```

Dart Solution:

```
class Solution {
List<bool> prefixesDivBy5(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def prefixesDivBy5(nums: Array[Int]): List[Boolean] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec prefixes_div_by5(nums :: [integer]) :: [boolean]
def prefixes_div_by5(nums) do

end
end
```

Erlang Solution:

```
-spec prefixes_div_by5(Nums :: [integer()]) -> [boolean()].
prefixes_div_by5(Nums) ->
.
```

Racket Solution:

```
(define/contract (prefixes-div-by5 nums)
(-> (listof exact-integer?) (listof boolean?))
)
```