Problem 717: 1-bit and 2-bit Characters

Problem Information

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

Problem Description

We have two special characters:

The first character can be represented by one bit

0

.

The second character can be represented by two bits (

10

or

11

).

Given a binary array

bits

that ends with

0

, return

true

if the last character must be a one-bit character.

Example 1:

Input:

bits = [1,0,0]

Output:

true

Explanation:

The only way to decode it is two-bit character and one-bit character. So the last character is one-bit character.

Example 2:

Input:

bits = [1,1,1,0]

Output:

false

Explanation:

The only way to decode it is two-bit character and two-bit character. So the last character is not one-bit character.

Constraints:

1 <= bits.length <= 1000

bits[i]

is either

0

or

1

.

Code Snippets

C++:

```
class Solution {
public:
bool isOneBitCharacter(vector<int>& bits) {

}
};
```

Java:

```
class Solution {
public boolean isOneBitCharacter(int[] bits) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def isOneBitCharacter(self, bits):
```

```
"""
:type bits: List[int]
:rtype: bool
"""
```

JavaScript:

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

};
```

TypeScript:

```
function isOneBitCharacter(bits: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool IsOneBitCharacter(int[] bits) {

}
}
```

C:

```
bool isOneBitCharacter(int* bits, int bitsSize) {

}
```

Go:

```
func isOneBitCharacter(bits []int) bool {

}
```

Kotlin:

```
class Solution {
fun isOneBitCharacter(bits: IntArray): Boolean {

}
}
```

Swift:

```
class Solution {
func isOneBitCharacter(_ bits: [Int]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn is_one_bit_character(bits: Vec<i32>) -> bool {

}
}
```

Ruby:

```
# @param {Integer[]} bits
# @return {Boolean}
def is_one_bit_character(bits)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $bits
* @return Boolean
*/
function isOneBitCharacter($bits) {

}
}
```

Dart:

```
class Solution {
bool isOneBitCharacter(List<int> bits) {

}
}
```

Scala:

```
object Solution {
def isOneBitCharacter(bits: Array[Int]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec is_one_bit_character(bits :: [integer]) :: boolean
def is_one_bit_character(bits) do

end
end
```

Erlang:

```
-spec is_one_bit_character(Bits :: [integer()]) -> boolean().
is_one_bit_character(Bits) ->
.
```

Racket:

```
(define/contract (is-one-bit-character bits)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: 1-bit and 2-bit Characters
* 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:
bool isOneBitCharacter(vector<int>& bits) {

}
};
```

Java Solution:

```
/**
* Problem: 1-bit and 2-bit Characters
* 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 boolean isOneBitCharacter(int[] bits) {

}
}
```

Python3 Solution:

```
"""
Problem: 1-bit and 2-bit Characters
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 isOneBitCharacter(self, bits: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: 1-bit and 2-bit Characters
* 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[]} bits
* @return {boolean}
*/
var isOneBitCharacter = function(bits) {

};
```

TypeScript Solution:

```
/**
* Problem: 1-bit and 2-bit Characters
* 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 isOneBitCharacter(bits: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: 1-bit and 2-bit Characters
* 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 bool IsOneBitCharacter(int[] bits) {

}
}
```

C Solution:

```
/*
* Problem: 1-bit and 2-bit Characters
* 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
```

```
*/

bool isOneBitCharacter(int* bits, int bitsSize) {

}
```

Go Solution:

```
// Problem: 1-bit and 2-bit Characters
// 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 isOneBitCharacter(bits []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun isOneBitCharacter(bits: IntArray): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func isOneBitCharacter(_ bits: [Int]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: 1-bit and 2-bit Characters
// 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 is_one_bit_character(bits: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[]} bits
# @return {Boolean}
def is_one_bit_character(bits)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $bits
* @return Boolean
*/
function isOneBitCharacter($bits) {

}
}
```

Dart Solution:

```
class Solution {
bool isOneBitCharacter(List<int> bits) {

}
}
```

Scala Solution:

```
object Solution {
def isOneBitCharacter(bits: Array[Int]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec is_one_bit_character(bits :: [integer]) :: boolean
def is_one_bit_character(bits) do

end
end
```

Erlang Solution:

```
-spec is_one_bit_character(Bits :: [integer()]) -> boolean().
is_one_bit_character(Bits) ->
.
```

Racket Solution:

```
(define/contract (is-one-bit-character bits)
(-> (listof exact-integer?) boolean?)
)
```