Problem 1980: Find Unique Binary String

Problem Information

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

Problem Description

Given an array of strings

nums

containing

n

unique

binary strings each of length

n

, return

a binary string of length

n

that

does not appear

in

nums

. If there are multiple answers, you may return

any

of them

.

Example 1:

Input:

nums = ["01","10"]

Output:

"11"

Explanation:

"11" does not appear in nums. "00" would also be correct.

Example 2:

Input:

nums = ["00","01"]

Output:

"11"

Explanation:

"11" does not appear in nums. "10" would also be correct.

Example 3:

Input:

nums = ["111","011","001"]

Output:

"101"

Explanation:

"101" does not appear in nums. "000", "010", "100", and "110" would also be correct.

Constraints:

n == nums.length

1 <= n <= 16

nums[i].length == n

nums[i]

is either

'0'

or

'1'

.

All the strings of

nums

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
string findDifferentBinaryString(vector<string>& nums) {

}
};
```

Java:

```
class Solution {
public String findDifferentBinaryString(String[] nums) {

}
}
```

Python3:

```
class Solution:
def findDifferentBinaryString(self, nums: List[str]) -> str:
```

Python:

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

JavaScript:

```
/**
* @param {string[]} nums
```

```
 * @return {string}
 */
var findDifferentBinaryString = function(nums) {

};
```

TypeScript:

```
function findDifferentBinaryString(nums: string[]): string {

};
```

C#:

```
public class Solution {
public string FindDifferentBinaryString(string[] nums) {

}
}
```

C:

```
char* findDifferentBinaryString(char** nums, int numsSize) {

}
```

Go:

```
func findDifferentBinaryString(nums []string) string {

}
```

Kotlin:

```
class Solution {
fun findDifferentBinaryString(nums: Array<String>): String {

}
}
```

Swift:

```
class Solution {
func findDifferentBinaryString(_ nums: [String]) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn find_different_binary_string(nums: Vec<String>) -> String {

}
}
```

Ruby:

```
# @param {String[]} nums
# @return {String}
def find_different_binary_string(nums)

end
```

PHP:

```
class Solution {

/**
* @param String[] $nums
* @return String
*/
function findDifferentBinaryString($nums) {

}
}
```

Dart:

```
class Solution {
String findDifferentBinaryString(List<String> nums) {

}
}
```

Scala:

```
object Solution {
def findDifferentBinaryString(nums: Array[String]): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_different_binary_string(nums :: [String.t]) :: String.t
def find_different_binary_string(nums) do

end
end
```

Erlang:

```
-spec find_different_binary_string(Nums :: [unicode:unicode_binary()]) ->
unicode:unicode_binary().
find_different_binary_string(Nums) ->
.
```

Racket:

```
(define/contract (find-different-binary-string nums)
(-> (listof string?) string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Unique Binary String
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

class Solution {
public:
string findDifferentBinaryString(vector<string>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Find Unique Binary String
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public String findDifferentBinaryString(String[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Unique Binary String
Difficulty: Medium
Tags: array, string, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def findDifferentBinaryString(self, nums: List[str]) -> str:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Unique Binary String
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string[]} nums
* @return {string}
*/
var findDifferentBinaryString = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Unique Binary String
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) for hash map
*/

function findDifferentBinaryString(nums: string[]): string {

};
```

C# Solution:

```
/*
* Problem: Find Unique Binary String
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public string FindDifferentBinaryString(string[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Find Unique Binary String
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

char* findDifferentBinaryString(char** nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Find Unique Binary String
// Difficulty: Medium
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func findDifferentBinaryString(nums []string) string {

}
```

Kotlin Solution:

```
class Solution {
fun findDifferentBinaryString(nums: Array<String>): String {

}
}
```

Swift Solution:

```
class Solution {
func findDifferentBinaryString(_ nums: [String]) -> String {

}
}
```

Rust Solution:

```
// Problem: Find Unique Binary String
// Difficulty: Medium
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn find_different_binary_string(nums: Vec<String>) -> String {
```

```
}
}
```

Ruby Solution:

```
# @param {String[]} nums
# @return {String}
def find_different_binary_string(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $nums
* @return String
*/
function findDifferentBinaryString($nums) {

}
}
```

Dart Solution:

```
class Solution {
String findDifferentBinaryString(List<String> nums) {

}
}
```

Scala Solution:

```
object Solution {
def findDifferentBinaryString(nums: Array[String]): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_different_binary_string(nums :: [String.t]) :: String.t
def find_different_binary_string(nums) do

end
end
```

Erlang Solution:

```
-spec find_different_binary_string(Nums :: [unicode:unicode_binary()]) ->
unicode:unicode_binary().
find_different_binary_string(Nums) ->
.
```

Racket Solution:

```
(define/contract (find-different-binary-string nums)
(-> (listof string?) string?)
)
```