Problem 3141: Maximum Hamming Distances

Problem Information

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

Problem Description

Given an array

nums

and an integer

m

, with each element

nums[i]

satisfying

0 <= nums[i] < 2

m

, return an array

answer

. The

answer

array should be of the same length as

nums

, where each element

answer[i]

represents the

maximum

Hamming distance

between

nums[i]

and any other element

nums[j]

in the array.

The

Hamming distance

between two binary integers is defined as the number of positions at which the corresponding bits differ (add leading zeroes if needed).

Example 1:

Input:

nums = [9,12,9,11], m = 4

Output:

[2,3,2,3]

Explanation:

The binary representation of

nums = [1001,1100,1001,1011]

.

The maximum hamming distances for each index are:

nums[0]

: 1001 and 1100 have a distance of 2.

nums[1]

: 1100 and 1011 have a distance of 3.

nums[2]

: 1001 and 1100 have a distance of 2.

nums[3]

: 1011 and 1100 have a distance of 3.

Example 2:

Input:

nums = [3,4,6,10], m = 4

Output:

[3,3,2,3]

Explanation:

The binary representation of

nums = [0011,0100,0110,1010]

.

The maximum hamming distances for each index are:

nums[0]

: 0011 and 0100 have a distance of 3.

nums[1]

: 0100 and 0011 have a distance of 3.

nums[2]

: 0110 and 1010 have a distance of 2.

nums[3]

: 1010 and 0100 have a distance of 3.

Constraints:

1 <= m <= 17

2 <= nums.length <= 2

m

0 <= nums[i] < 2

m

Code Snippets

C++:

```
class Solution {
public:
vector<int> maxHammingDistances(vector<int>& nums, int m) {

}
};
```

Java:

```
class Solution {
public int[] maxHammingDistances(int[] nums, int m) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} m
* @return {number[]}
*/
var maxHammingDistances = function(nums, m) {

};
```

TypeScript:

```
function maxHammingDistances(nums: number[], m: number): number[] {

};
```

C#:

```
public class Solution {
public int[] MaxHammingDistances(int[] nums, int m) {

}
}
```

C:

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

}
```

Go:

```
func maxHammingDistances(nums []int, m int) []int {

}
```

Kotlin:

```
class Solution {
fun maxHammingDistances(nums: IntArray, m: Int): IntArray {

}
}
```

Swift:

```
class Solution {
func maxHammingDistances(_ nums: [Int], _ m: Int) -> [Int] {

}
```

```
}
```

Rust:

```
impl Solution {
pub fn max_hamming_distances(nums: Vec<i32>, m: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} m
# @return {Integer[]}
def max_hamming_distances(nums, m)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $m
* @return Integer[]
*/
function maxHammingDistances($nums, $m) {

}
}
```

Dart:

```
class Solution {
List<int> maxHammingDistances(List<int> nums, int m) {

}
}
```

Scala:

```
object Solution {
def maxHammingDistances(nums: Array[Int], m: Int): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_hamming_distances(nums :: [integer], m :: integer) :: [integer]
def max_hamming_distances(nums, m) do

end
end
```

Erlang:

```
-spec max_hamming_distances(Nums :: [integer()], M :: integer()) ->
[integer()].
max_hamming_distances(Nums, M) ->
.
```

Racket:

```
(define/contract (max-hamming-distances nums m)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Hamming Distances
* Difficulty: Hard
* Tags: array, search
*
* 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> maxHammingDistances(vector<int>& nums, int m) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Hamming Distances
* Difficulty: Hard
* Tags: array, search
*
* 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[] maxHammingDistances(int[] nums, int m) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Hamming Distances
Difficulty: Hard
Tags: array, search

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 maxHammingDistances(self, nums: List[int], m: int) -> List[int]:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Hamming Distances
* Difficulty: Hard
* Tags: array, search
*
* 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
* @param {number} m
* @return {number[]}
*/
var maxHammingDistances = function(nums, m) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Hamming Distances
* Difficulty: Hard
* Tags: array, search
*
* 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 maxHammingDistances(nums: number[], m: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Maximum Hamming Distances
* Difficulty: Hard
* Tags: array, search
*
* 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[] MaxHammingDistances(int[] nums, int m) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Hamming Distances
* Difficulty: Hard
* Tags: array, search
*
* 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* maxHammingDistances(int* nums, int numsSize, int m, int* returnSize) {

}
```

Go Solution:

```
// Problem: Maximum Hamming Distances
// Difficulty: Hard
// Tags: array, search
//
// 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 maxHammingDistances(nums []int, m int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun maxHammingDistances(nums: IntArray, m: Int): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func maxHammingDistances(_ nums: [Int], _ m: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Maximum Hamming Distances
// Difficulty: Hard
// Tags: array, search
//
// 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 max_hamming_distances(nums: Vec<i32>, m: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} m
# @return {Integer[]}
def max_hamming_distances(nums, m)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $m
* @return Integer[]
*/
function maxHammingDistances($nums, $m) {

}
}
```

Dart Solution:

```
class Solution {
List<int> maxHammingDistances(List<int> nums, int m) {

}
}
```

Scala Solution:

```
object Solution {
def maxHammingDistances(nums: Array[Int], m: Int): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_hamming_distances(nums :: [integer], m :: integer) :: [integer]
def max_hamming_distances(nums, m) do

end
end
```

Erlang Solution:

```
-spec max_hamming_distances(Nums :: [integer()], M :: integer()) ->
[integer()].
max_hamming_distances(Nums, M) ->
.
```

Racket Solution:

```
(define/contract (max-hamming-distances nums m)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```