Problem 3629: Minimum Jumps to Reach End via Prime Teleportation

Problem Information

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

Problem Description

You are given an integer array

nums

of length

n

.

You start at index 0, and your goal is to reach index

n - 1

.

From any index

i

, you may perform one of the following operations:

Adjacent Step

: Jump to index

i + 1

or

i - 1

, if the index is within bounds.

Prime Teleportation

: If

nums[i]

is a

prime number

p

, you may instantly jump to any index

j != i

such that

nums[j] % p == 0

.

Return the

minimum

number of jumps required to reach index

n - 1

.

Example 1:

Input:

nums = [1,2,4,6]

Output:

2

Explanation:

One optimal sequence of jumps is:

Start at index

i = 0

. Take an adjacent step to index 1.

At index

i = 1

,

nums[1] = 2

is a prime number. Therefore, we teleport to index

i = 3

as

nums[3] = 6

is divisible by 2.

Thus, the answer is 2.

Example 2:

Input:

nums = [2,3,4,7,9]

Output:

2

Explanation:

One optimal sequence of jumps is:

Start at index

i = 0

. Take an adjacent step to index

i = 1

.

At index

i = 1

,

nums[1] = 3

is a prime number. Therefore, we teleport to index

i = 4

since

nums[4] = 9

is divisible by 3.

Thus, the answer is 2.

Example 3:

Input:

nums = [4,6,5,8]

Output:

3

Explanation:

Since no teleportation is possible, we move through

0 → 1 → 2 → 3

. Thus, the answer is 3.

Constraints:

1 <= n == nums.length <= 10

5

1 <= nums[i] <= 10

6

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int minJumps(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minJumps(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int MinJumps(int[] nums) {

}
}
```

C:

```
int minJumps(int* nums, int numsSize) {

}
```

Go:

```
func minJumps(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun minJumps(nums: IntArray): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int minJumps(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def minJumps(nums: Array[Int]): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (min-jumps nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Jumps to Reach End via Prime Teleportation
* Difficulty: Medium
* Tags: array, math, hash, search
*
* 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:
int minJumps(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Jumps to Reach End via Prime Teleportation
* Difficulty: Medium
* Tags: array, math, hash, search
*
* 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 int minJumps(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Jumps to Reach End via Prime Teleportation
Difficulty: Medium
Tags: array, math, hash, search

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

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Jumps to Reach End via Prime Teleportation
* Difficulty: Medium
* Tags: array, math, hash, search
*
* 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 {number[]} nums
* @return {number}
*/
var minJumps = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Jumps to Reach End via Prime Teleportation
* Difficulty: Medium
* Tags: array, math, hash, search
*
* 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 minJumps(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Jumps to Reach End via Prime Teleportation
* Difficulty: Medium
* Tags: array, math, hash, search
*
* 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 int MinJumps(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Jumps to Reach End via Prime Teleportation
* Difficulty: Medium
* Tags: array, math, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int minJumps(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimum Jumps to Reach End via Prime Teleportation
// Difficulty: Medium
// Tags: array, math, hash, search
//
// 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 minJumps(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minJumps(nums: IntArray): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Jumps to Reach End via Prime Teleportation
// Difficulty: Medium
// Tags: array, math, hash, search
//
// 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 min_jumps(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int minJumps(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def minJumps(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-jumps nums)
(-> (listof exact-integer?) exact-integer?)
)
```