Problem 1345: Jump Game IV

Problem Information

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

Problem Description

Given an array of integers

arr

, you are initially positioned at the first index of the array.

In one step you can jump from index

i

to index:

i + 1

where:

i + 1 < arr.length

.

i - 1

where:

i - 1 >= 0

.

j

where:

arr[i] == arr[j]

and

i != j

.

Return

the minimum number of steps

to reach the

last index

of the array.

Notice that you can not jump outside of the array at any time.

Example 1:

Input:

arr = [100,-23,-23,404,100,23,23,23,3,404]

Output:

3

Explanation:

You need three jumps from index 0 --> 4 --> 3 --> 9. Note that index 9 is the last index of the array.

Example 2:

Input:

arr = [7]

Output:

0

Explanation:

Start index is the last index. You do not need to jump.

Example 3:

Input:

arr = [7,6,9,6,9,6,9,7]

Output:

1

Explanation:

You can jump directly from index 0 to index 7 which is last index of the array.

Constraints:

1 <= arr.length <= 5 * 10

4

-10

8

<= arr[i] <= 10

8

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int minJumps(int* arr, int arrSize) {

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Jump Game IV
* Difficulty: Hard
* Tags: array, 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>& arr) {

}
};
```

Java Solution:

```
/**
* Problem: Jump Game IV
* Difficulty: Hard
* Tags: array, 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[] arr) {

}
}
```

Python3 Solution:

```
"""
Problem: Jump Game IV
Difficulty: Hard
Tags: array, 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, arr: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Jump Game IV
* Difficulty: Hard
* Tags: array, 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[]} arr
* @return {number}
*/
var minJumps = function(arr) {

};
```

TypeScript Solution:

```
/**
* Problem: Jump Game IV
* Difficulty: Hard
* Tags: array, 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(arr: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Jump Game IV
* Difficulty: Hard
* Tags: array, 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[] arr) {

}
}
```

C Solution:

```
/*
* Problem: Jump Game IV
* Difficulty: Hard
* Tags: array, 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* arr, int arrSize) {

}
```

Go Solution:

```
// Problem: Jump Game IV
// Difficulty: Hard
// Tags: array, 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(arr []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Jump Game IV
// Difficulty: Hard
// Tags: array, 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(arr: Vec<i32>) -> i32 {

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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