Problem 3660: Jump Game IX

Problem Information

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

Problem Description

You are given an integer array

nums

.

From any index

i

, you can jump to another index

j

under the following rules:

Jump to index

j

where

j > i

is allowed only if

nums[j] < nums[i]

.

Jump to index

j

where

j < i

is allowed only if

nums[j] > nums[i]

.

For each index

i

, find the

maximum

value

in

nums

that can be reached by following

any

sequence of valid jumps starting at

i

.

Return an array

ans

where

ans[i]

is the

maximum

value

reachable starting from index

i

.

Example 1:

Input:

nums = [2,1,3]

Output:

[2,2,3]

Explanation:

For

i = 0

: No jump increases the value.

For

i = 1

: Jump to

j = 0

as

nums[j] = 2

is greater than

nums[i]

.

For

i = 2

: Since

nums[2] = 3

is the maximum value in

nums

, no jump increases the value.

Thus,

ans = [2, 2, 3]

.

Example 2:

Input:

nums = [2,3,1]

Output:

[3,3,3]

Explanation:

For

i = 0

: Jump forward to

j = 2

as

nums[j] = 1

is less than

nums[i] = 2

, then from

i = 2

jump to

j = 1

as

nums[j] = 3

is greater than

nums[2]

.

For

i = 1

: Since

nums[1] = 3

is the maximum value in

nums

, no jump increases the value.

For

i = 2

: Jump to

j = 1

as

nums[j] = 3

is greater than

nums[2] = 1

.

Thus,

ans = [3, 3, 3]

.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun maxValue(nums: IntArray): IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-value nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Jump Game IX
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Jump Game IX
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int[] maxValue(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Jump Game IX
Difficulty: Medium
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def maxValue(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Jump Game IX
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Jump Game IX
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxValue(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Jump Game IX
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int[] MaxValue(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Jump Game IX
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
```

Go Solution:

```
// Problem: Jump Game IX
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxValue(nums []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun maxValue(nums: IntArray): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Jump Game IX
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-value nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```