Problem 1824: Minimum Sideway Jumps

Problem Information

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

Problem Description

There is a

3 lane road

of length

n

that consists of

n + 1

points

labeled from

0

to

n

. A frog

starts

at point

0

in the

second

lane

and wants to jump to point

n

. However, there could be obstacles along the way.

You are given an array

obstacles

of length

n + 1

where each

obstacles[i]

(

ranging from 0 to 3

) describes an obstacle on the lane

obstacles[i]

at point

i

. If

obstacles[i] == 0

, there are no obstacles at point

i

. There will be

at most one

obstacle in the 3 lanes at each point.

For example, if

obstacles[2] == 1

, then there is an obstacle on lane 1 at point 2.

The frog can only travel from point

i

to point

i + 1

on the same lane if there is not an obstacle on the lane at point

i + 1

. To avoid obstacles, the frog can also perform a

side jump

to jump to

another

lane (even if they are not adjacent) at the

same

point if there is no obstacle on the new lane.

For example, the frog can jump from lane 3 at point 3 to lane 1 at point 3.

Return

the

minimum number of side jumps

the frog needs to reach

any lane

at point n starting from lane

2

at point 0.

Note:

There will be no obstacles on points

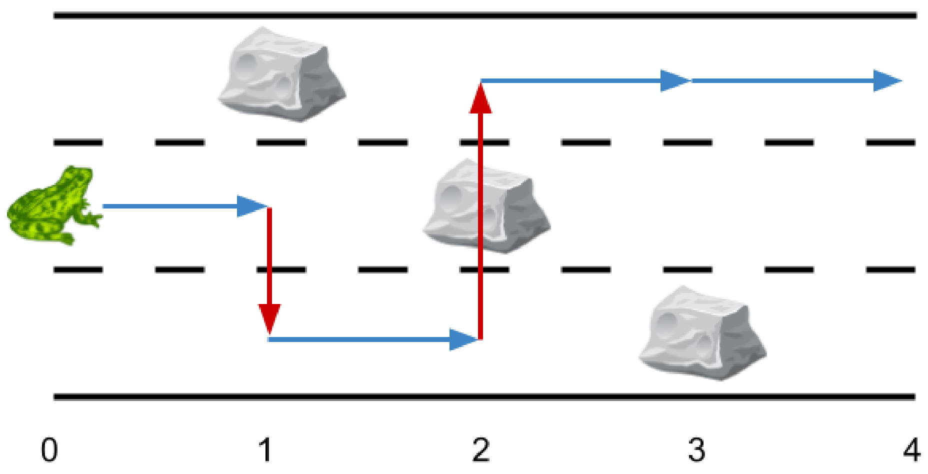
0

and

n

.

Example 1:



Input:

obstacles = [0,1,2,3,0]

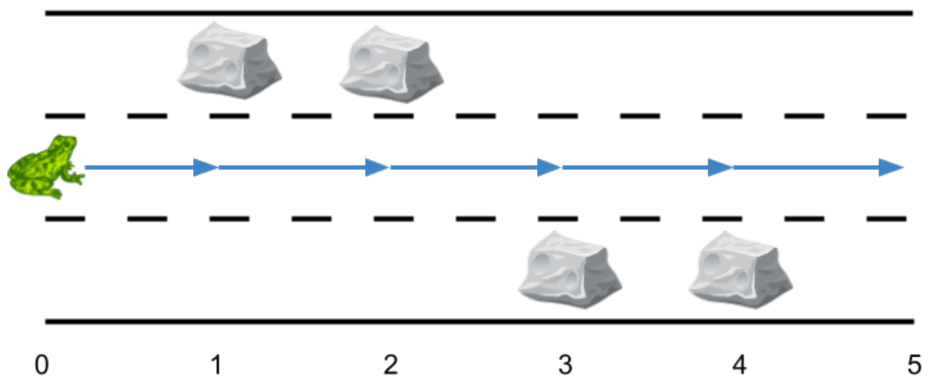
Output:

2

Explanation:

The optimal solution is shown by the arrows above. There are 2 side jumps (red arrows). Note that the frog can jump over obstacles only when making side jumps (as shown at point 2).

Example 2:



Input:

obstacles = [0,1,1,3,3,0]

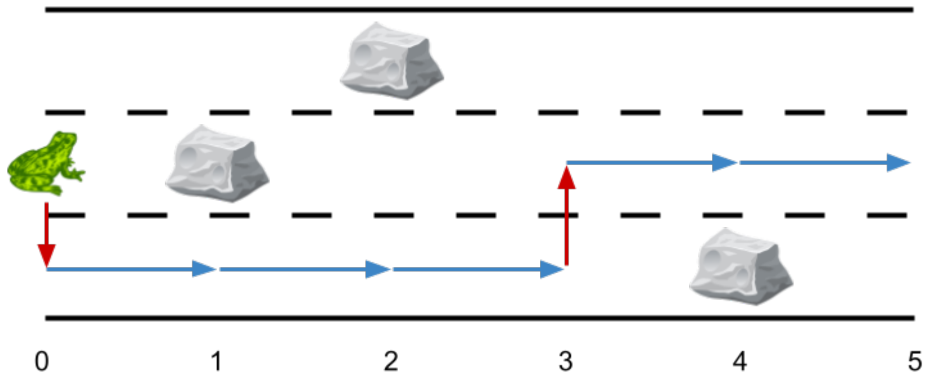
Output:

0

Explanation:

There are no obstacles on lane 2. No side jumps are required.

Example 3:



Input:

obstacles = [0,2,1,0,3,0]

Output:

2

Explanation:

The optimal solution is shown by the arrows above. There are 2 side jumps.

Constraints:

obstacles.length == n + 1

1 <= n <= 5 * 10

5

0 <= obstacles[i] <= 3

obstacles[0] == obstacles[n] == 0

Code Snippets

C++:

```
class Solution {
public:
int minSideJumps(vector<int>& obstacles) {

}
};
```

Java:

```
class Solution {
public int minSideJumps(int[] obstacles) {

}
}
```

Python3:

```
class Solution:
def minSideJumps(self, obstacles: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minSideJumps(obstacles: number[]): number {

};
```

C#:

```
public class Solution {
public int MinSideJumps(int[] obstacles) {

}
}
```

C:

```
int minSideJumps(int* obstacles, int obstaclesSize) {

}
```

Go:

```
func minSideJumps(obstacles []int) int {

}
```

Kotlin:

```
class Solution {
fun minSideJumps(obstacles: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minSideJumps(_ obstacles: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_side_jumps(obstacles: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} obstacles
# @return {Integer}
def min_side_jumps(obstacles)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $obstacles
* @return Integer
*/
function minSideJumps($obstacles) {

}
}
```

Dart:

```
class Solution {
int minSideJumps(List<int> obstacles) {

}
}
```

Scala:

```
object Solution {
def minSideJumps(obstacles: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_side_jumps(obstacles :: [integer]) :: integer
def min_side_jumps(obstacles) do

end
end
```

Erlang:

```
-spec min_side_jumps(Obstacles :: [integer()]) -> integer().
min_side_jumps(Obstacles) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Sideway Jumps
* Difficulty: Medium
* Tags: array, tree, dp, greedy
*
* 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 minSideJumps(vector<int>& obstacles) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Sideway Jumps
* Difficulty: Medium
* Tags: array, tree, dp, greedy
*
* 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 minSideJumps(int[] obstacles) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Sideway Jumps
Difficulty: Medium
Tags: array, tree, dp, greedy

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 minSideJumps(self, obstacles: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Sideway Jumps
* Difficulty: Medium
* Tags: array, tree, dp, greedy
*
* 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[]} obstacles
* @return {number}
*/
var minSideJumps = function(obstacles) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Sideway Jumps
* Difficulty: Medium
* Tags: array, tree, dp, greedy
*
* 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 minSideJumps(obstacles: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Sideway Jumps
* Difficulty: Medium
* Tags: array, tree, dp, greedy
*
* 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 MinSideJumps(int[] obstacles) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Sideway Jumps
* Difficulty: Medium
* Tags: array, tree, dp, greedy
*
* 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
*/

int minSideJumps(int* obstacles, int obstaclesSize) {

}
```

Go Solution:

```
// Problem: Minimum Sideway Jumps
// Difficulty: Medium
// Tags: array, tree, dp, greedy
//
// 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 minSideJumps(obstacles []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minSideJumps(obstacles: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minSideJumps(_ obstacles: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Sideway Jumps
// Difficulty: Medium
// Tags: array, tree, dp, greedy
//
// 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 min_side_jumps(obstacles: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} obstacles
# @return {Integer}
def min_side_jumps(obstacles)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $obstacles
* @return Integer
*/
function minSideJumps($obstacles) {

}
}
```

Dart Solution:

```
class Solution {
int minSideJumps(List<int> obstacles) {

}
}
```

Scala Solution:

```
object Solution {
def minSideJumps(obstacles: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_side_jumps(obstacles :: [integer]) :: integer
def min_side_jumps(obstacles) do

end
end
```

Erlang Solution:

```
-spec min_side_jumps(Obstacles :: [integer()]) -> integer().
min_side_jumps(Obstacles) ->
.
```

Racket Solution:

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