Problem 1326: Minimum Number of Taps to Open to Water a Garden

Problem Information

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

Problem Description

There is a one-dimensional garden on the x-axis. The garden starts at the point

0

and ends at the point

n

. (i.e., the length of the garden is

n

).

There are

n + 1

taps located at points

[0, 1, ..., n]

in the garden.

Given an integer

n

and an integer array

ranges

of length

n + 1

where

ranges[i]

(0-indexed) means the

i-th

tap can water the area

[i - ranges[i], i + ranges[i]]

if it was open.

Return

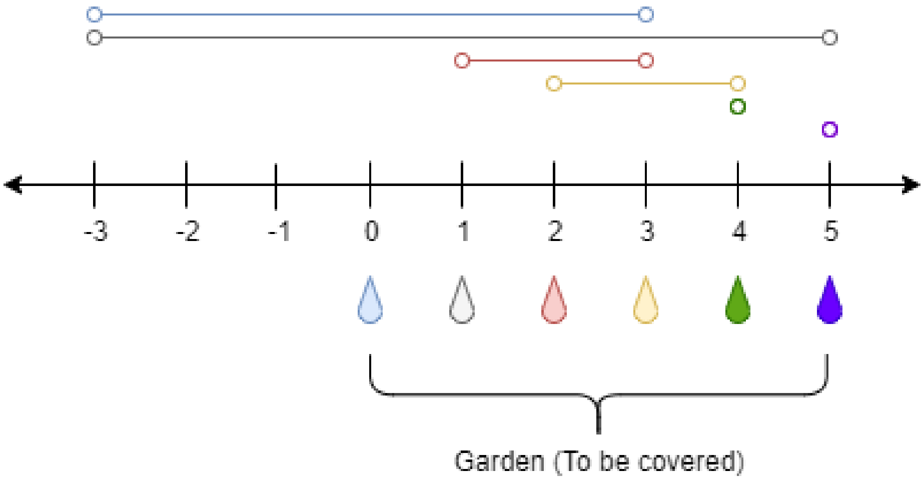
the minimum number of taps

that should be open to water the whole garden, If the garden cannot be watered return

-1

.

Example 1:



Input:

n = 5, ranges = [3,4,1,1,0,0]

Output:

1

Explanation:

The tap at point 0 can cover the interval [-3,3] The tap at point 1 can cover the interval [-3,5] The tap at point 2 can cover the interval [1,3] The tap at point 3 can cover the interval [2,4] The tap at point 4 can cover the interval [4,4] The tap at point 5 can cover the interval [5,5] Opening Only the second tap will water the whole garden [0,5]

Example 2:

Input:

n = 3, ranges = [0,0,0,0]

Output:

-1

Explanation:

Even if you activate all the four taps you cannot water the whole garden.

Constraints:

1 <= n <= 10

4

ranges.length == n + 1

0 <= ranges[i] <= 100

Code Snippets

C++:

```
class Solution {
public:
int minTaps(int n, vector<int>& ranges) {

}
};
```

Java:

```
class Solution {
public int minTaps(int n, int[] ranges) {

}
}
```

Python3:

```
class Solution:
def minTaps(self, n: int, ranges: List[int]) -> int:
```

Python:

```
class Solution(object):
def minTaps(self, n, ranges):
"""
:type n: int
```

```
:type ranges: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[]} ranges
* @return {number}
*/
var minTaps = function(n, ranges) {

};
```

TypeScript:

```
function minTaps(n: number, ranges: number[]): number {

};
```

C#:

```
public class Solution {
public int MinTaps(int n, int[] ranges) {

}
}
```

C:

```
int minTaps(int n, int* ranges, int rangesSize) {

}
```

Go:

```
func minTaps(n int, ranges []int) int {

}
```

Kotlin:

```
class Solution {
fun minTaps(n: Int, ranges: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minTaps(_ n: Int, _ ranges: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_taps(n: i32, ranges: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[]} ranges
# @return {Integer}
def min_taps(n, ranges)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $ranges
* @return Integer
*/
function minTaps($n, $ranges) {

}

}
```

Dart:

```
class Solution {
int minTaps(int n, List<int> ranges) {

}
}
```

Scala:

```
object Solution {
def minTaps(n: Int, ranges: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_taps(n :: integer, ranges :: [integer]) :: integer
def min_taps(n, ranges) do

end
end
```

Erlang:

```
-spec min_taps(N :: integer(), Ranges :: [integer()]) -> integer().
min_taps(N, Ranges) ->
.
```

Racket:

```
(define/contract (min-taps n ranges)
(-> exact-integer? (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Taps to Open to Water a Garden
* Difficulty: Hard
* Tags: array, 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 minTaps(int n, vector<int>& ranges) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Taps to Open to Water a Garden
* Difficulty: Hard
* Tags: array, 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 minTaps(int n, int[] ranges) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Taps to Open to Water a Garden
Difficulty: Hard
Tags: array, 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 minTaps(self, n: int, ranges: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minTaps(self, n, ranges):
"""
:type n: int
:type ranges: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Number of Taps to Open to Water a Garden
* Difficulty: Hard
* Tags: array, 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} n
* @param {number[]} ranges
* @return {number}
*/
var minTaps = function(n, ranges) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Taps to Open to Water a Garden
* Difficulty: Hard
* Tags: array, 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 minTaps(n: number, ranges: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Taps to Open to Water a Garden
* Difficulty: Hard
* Tags: array, 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 MinTaps(int n, int[] ranges) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Taps to Open to Water a Garden
* Difficulty: Hard
* Tags: array, 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 minTaps(int n, int* ranges, int rangesSize) {

}
```

Go Solution:

```
// Problem: Minimum Number of Taps to Open to Water a Garden
// Difficulty: Hard
// Tags: array, 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 minTaps(n int, ranges []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minTaps(n: Int, ranges: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minTaps(_ n: Int, _ ranges: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Taps to Open to Water a Garden
// Difficulty: Hard
// Tags: array, 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_taps(n: i32, ranges: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[]} ranges
# @return {Integer}
def min_taps(n, ranges)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $ranges
* @return Integer
*/
function minTaps($n, $ranges) {

}
}
```

Dart Solution:

```
class Solution {
int minTaps(int n, List<int> ranges) {
}
}
```

Scala Solution:

```
object Solution {
def minTaps(n: Int, ranges: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_taps(n :: integer, ranges :: [integer]) :: integer
def min_taps(n, ranges) do

end
end
```

Erlang Solution:

```
-spec min_taps(N :: integer(), Ranges :: [integer()]) -> integer().
min_taps(N, Ranges) ->
.
```

Racket Solution:

```
(define/contract (min-taps n ranges)
(-> exact-integer? (listof exact-integer?) exact-integer?)
)
```