Problem 1871: Jump Game VII

Problem Information

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

Problem Description

You are given a

0-indexed

binary string

s

and two integers

minJump

and

maxJump

. In the beginning, you are standing at index

0

, which is equal to

'0'

. You can move from index

i

to index

j

if the following conditions are fulfilled:

i + minJump <= j <= min(i + maxJump, s.length - 1)

, and

s[j] == '0'

.

Return

true

if you can reach index

s.length - 1

in

s

, or

false

otherwise.

Example 1:

Input:

s = "

0

11

0

1

0

", minJump = 2, maxJump = 3

Output:

true

Explanation:

In the first step, move from index 0 to index 3. In the second step, move from index 3 to index 5.

Example 2:

Input:

s = "01101110", minJump = 2, maxJump = 3

Output:

false

Constraints:

2 <= s.length <= 10

5

s[i]

is either

'0'

or

'1'

.

s[0] == '0'

1 <= minJump <= maxJump < s.length

Code Snippets

C++:

```
class Solution {
public:
bool canReach(string s, int minJump, int maxJump) {

}
};
```

Java:

```
class Solution {
public boolean canReach(String s, int minJump, int maxJump) {

}
}
```

Python3:

```
class Solution:
def canReach(self, s: str, minJump: int, maxJump: int) -> bool:
```

Python:

```
class Solution(object):
def canReach(self, s, minJump, maxJump):
"""
:type s: str
:type minJump: int
:type maxJump: int
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {number} minJump
* @param {number} maxJump
* @return {boolean}
*/
var canReach = function(s, minJump, maxJump) {

};
```

TypeScript:

```
function canReach(s: string, minJump: number, maxJump: number): boolean {

};
```

C#:

```
public class Solution {
public bool CanReach(string s, int minJump, int maxJump) {

}
}
```

C:

```
bool canReach(char* s, int minJump, int maxJump) {

}
```

Go:

```
func canReach(s string, minJump int, maxJump int) bool {

}
```

Kotlin:

```
class Solution {
fun canReach(s: String, minJump: Int, maxJump: Int): Boolean {

}
}
```

Swift:

```
class Solution {
func canReach(_ s: String, _ minJump: Int, _ maxJump: Int) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn can_reach(s: String, min_jump: i32, max_jump: i32) -> bool {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer} min_jump
# @param {Integer} max_jump
# @return {Boolean}
def can_reach(s, min_jump, max_jump)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer $minJump
* @param Integer $maxJump
* @return Boolean
*/
function canReach($s, $minJump, $maxJump) {

}
}
```

Dart:

```
class Solution {
bool canReach(String s, int minJump, int maxJump) {

}
}
```

Scala:

```
object Solution {
def canReach(s: String, minJump: Int, maxJump: Int): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec can_reach(s :: String.t, min_jump :: integer, max_jump :: integer) ::
boolean
def can_reach(s, min_jump, max_jump) do

end
end
```

Erlang:

```
-spec can_reach(S :: unicode:unicode_binary(), MinJump :: integer(), MaxJump
:: integer()) -> boolean().
can_reach(S, MinJump, MaxJump) ->
.
```

Racket:

```
(define/contract (can-reach s minJump maxJump)
(-> string? exact-integer? exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Jump Game VII
* Difficulty: Medium
* Tags: array, string, 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:
bool canReach(string s, int minJump, int maxJump) {

}
};
```

Java Solution:

```
/**
* Problem: Jump Game VII
* Difficulty: Medium
* Tags: array, string, 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 boolean canReach(String s, int minJump, int maxJump) {
}
}
```

Python3 Solution:

```
"""
Problem: Jump Game VII
Difficulty: Medium
Tags: array, string, 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 canReach(self, s: str, minJump: int, maxJump: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def canReach(self, s, minJump, maxJump):
"""
:type s: str
:type minJump: int
:type maxJump: int
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Jump Game VII
* Difficulty: Medium
* Tags: array, string, 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 {string} s
* @param {number} minJump
* @param {number} maxJump
* @return {boolean}
*/
var canReach = function(s, minJump, maxJump) {

};
```

TypeScript Solution:

```
/**
* Problem: Jump Game VII
* Difficulty: Medium
* Tags: array, string, 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 canReach(s: string, minJump: number, maxJump: number): boolean {

};
```

C# Solution:

```
/*
* Problem: Jump Game VII
* Difficulty: Medium
* Tags: array, string, 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 bool CanReach(string s, int minJump, int maxJump) {

}
}
```

C Solution:

```
/*
* Problem: Jump Game VII
* Difficulty: Medium
* Tags: array, string, 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
*/

bool canReach(char* s, int minJump, int maxJump) {

}
```

Go Solution:

```
// Problem: Jump Game VII
// Difficulty: Medium
// Tags: array, string, 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 canReach(s string, minJump int, maxJump int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun canReach(s: String, minJump: Int, maxJump: Int): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func canReach(_ s: String, _ minJump: Int, _ maxJump: Int) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Jump Game VII
// Difficulty: Medium
// Tags: array, string, 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 can_reach(s: String, min_jump: i32, max_jump: i32) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer} min_jump
# @param {Integer} max_jump
# @return {Boolean}
def can_reach(s, min_jump, max_jump)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param String $s
* @param Integer $minJump
* @param Integer $maxJump
* @return Boolean
*/
function canReach($s, $minJump, $maxJump) {

}
}
```

Dart Solution:

```
class Solution {
bool canReach(String s, int minJump, int maxJump) {

}
}
```

Scala Solution:

```
object Solution {
def canReach(s: String, minJump: Int, maxJump: Int): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec can_reach(s :: String.t, min_jump :: integer, max_jump :: integer) :: boolean
def can_reach(s, min_jump, max_jump) do

end
end
```

Erlang Solution:

```
-spec can_reach(S :: unicode:unicode_binary(), MinJump :: integer(), MaxJump :: integer()) -> boolean().
```

```
can_reach(S, MinJump, MaxJump) ->
.
```

Racket Solution:

```
(define/contract (can-reach s minJump maxJump)
(-> string? exact-integer? exact-integer? boolean?)
)
```