Problem 1404: Number of Steps to Reduce a Number in Binary Representation to One

Problem Information

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

Problem Description

Given the binary representation of an integer as a string

s

, return

the number of steps to reduce it to

1

under the following rules

:

If the current number is even, you have to divide it by

2

.

If the current number is odd, you have to add

1

to it.

It is guaranteed that you can always reach one for all test cases.

Example 1:

Input:

s = "1101"

Output:

6

Explanation:

"1101" corressponds to number 13 in their decimal representation. Step 1) 13 is odd, add 1 and obtain 14. Step 2) 14 is even, divide by 2 and obtain 7. Step 3) 7 is odd, add 1 and obtain 8. Step 4) 8 is even, divide by 2 and obtain 4. Step 5) 4 is even, divide by 2 and obtain 2. Step 6) 2 is even, divide by 2 and obtain 1.

Example 2:

Input:

s = "10"

Output:

1

Explanation:

"10" corresponds to number 2 in their decimal representation. Step 1) 2 is even, divide by 2 and obtain 1.

Example 3:

Input:

s = "1"

Output:

0

Constraints:

1 <= s.length <= 500

s

consists of characters '0' or '1'

s[0] == '1'

Code Snippets

C++:

```
class Solution {
public:
int numSteps(string s) {

}
};
```

Java:

```
class Solution {
public int numSteps(String s) {

}
}
```

Python3:

```
class Solution:
def numSteps(self, s: str) -> int:
```

Python:

```
class Solution(object):
def numSteps(self, s):
"""
:type s: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} s
* @return {number}
*/
var numSteps = function(s) {

};
```

TypeScript:

```
function numSteps(s: string): number {

};
```

C#:

```
public class Solution {
public int NumSteps(string s) {

}
}
```

C:

```
int numSteps(char* s) {

}
```

Go:

```
func numSteps(s string) int {

}
```

Kotlin:

```
class Solution {
fun numSteps(s: String): Int {

}
}
```

Swift:

```
class Solution {
func numSteps(_ s: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_steps(s: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} s
# @return {Integer}
def num_steps(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return Integer
*/
function numSteps($s) {

}
```

```
}
```

Dart:

```
class Solution {
int numSteps(String s) {

}
}
```

Scala:

```
object Solution {
def numSteps(s: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_steps(s :: String.t) :: integer
def num_steps(s) do

end
end
```

Erlang:

```
-spec num_steps(S :: unicode:unicode_binary()) -> integer().
num_steps(S) ->
.
```

Racket:

```
(define/contract (num-steps s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Steps to Reduce a Number in Binary Representation to One
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int numSteps(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Steps to Reduce a Number in Binary Representation to One
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int numSteps(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Steps to Reduce a Number in Binary Representation to One
Difficulty: Medium
Tags: string
```

```
Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def numSteps(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numSteps(self, s):
"""
:type s: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Steps to Reduce a Number in Binary Representation to One
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string} s
* @return {number}
*/
var numSteps = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Steps to Reduce a Number in Binary Representation to One
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function numSteps(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Number of Steps to Reduce a Number in Binary Representation to One
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int NumSteps(string s) {

}
}
```

C Solution:

```
/*
* Problem: Number of Steps to Reduce a Number in Binary Representation to One
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

int numSteps(char* s) {

}
```

Go Solution:

```
// Problem: Number of Steps to Reduce a Number in Binary Representation to
One
// Difficulty: Medium
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func numSteps(s string) int {

}
```

Kotlin Solution:

```
class Solution {
fun numSteps(s: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func numSteps(_ s: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Steps to Reduce a Number in Binary Representation to
One
```

```
// Difficulty: Medium
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn num_steps(s: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {Integer}
def num_steps(s)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return Integer
*/
function numSteps($s) {

}
}
```

Dart Solution:

```
class Solution {
int numSteps(String s) {

}
}
```

Scala Solution:

```
object Solution {
def numSteps(s: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_steps(s :: String.t) :: integer
def num_steps(s) do

end
end
```

Erlang Solution:

```
-spec num_steps(S :: unicode:unicode_binary()) -> integer().
num_steps(S) ->
.
```

Racket Solution:

```
(define/contract (num-steps s)
(-> string? exact-integer?)
)
```