Problem 600: Non-negative Integers without Consecutive Ones

Problem Information

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

Problem Description

Given a positive integer

n

, return the number of the integers in the range

[0, n]

whose binary representations

do not

contain consecutive ones.

Example 1:

Input:

n = 5

Output:

5

Explanation:

Here are the non-negative integers <= 5 with their corresponding binary representations: 0 : 0 1 : 1 2 : 10 3 : 11 4 : 100 5 : 101 Among them, only integer 3 disobeys the rule (two consecutive ones) and the other 5 satisfy the rule.

Example 2:

Input:

n = 1

Output:

2

Example 3:

Input:

n = 2

Output:

3

Constraints:

1 <= n <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int findIntegers(int n) {
}
};
```

Java:

```
class Solution {
public int findIntegers(int n) {

}
}
```

Python3:

```
class Solution:
def findIntegers(self, n: int) -> int:
```

Python:

```
class Solution(object):
def findIntegers(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {number}
*/
var findIntegers = function(n) {

};
```

TypeScript:

```
function findIntegers(n: number): number {

};
```

C#:

}

```
public class Solution {
public int FindIntegers(int n) {

}
}
```

C:

```
int findIntegers(int n) {

}
```

Go:

```
func findIntegers(n int) int {

}
```

Kotlin:

```
class Solution {
fun findIntegers(n: Int): Int {

}
}
```

Swift:

```
class Solution {
func findIntegers(_ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_integers(n: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer}
def find_integers(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function findIntegers($n) {

}
}
```

Dart:

```
class Solution {
int findIntegers(int n) {

}
}
```

Scala:

```
object Solution {
def findIntegers(n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_integers(n :: integer) :: integer
def find_integers(n) do

end
end
```

Erlang:

```
-spec find_integers(N :: integer()) -> integer().
find_integers(N) ->
.
```

Racket:

```
(define/contract (find-integers n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Non-negative Integers without Consecutive Ones
* Difficulty: Hard
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int findIntegers(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Non-negative Integers without Consecutive Ones
* Difficulty: Hard
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int findIntegers(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Non-negative Integers without Consecutive Ones
Difficulty: Hard
Tags: dp

Approach: Dynamic programming with memoization or tabulation
Time Complexity: O(n * m) where n and m are problem dimensions
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def findIntegers(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findIntegers(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Non-negative Integers without Consecutive Ones
* Difficulty: Hard
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} n
* @return {number}
*/
var findIntegers = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Non-negative Integers without Consecutive Ones
* Difficulty: Hard
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

function findIntegers(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Non-negative Integers without Consecutive Ones
* Difficulty: Hard
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

public class Solution {
public int FindIntegers(int n) {

}
}
```

C Solution:

```
/*
* Problem: Non-negative Integers without Consecutive Ones
* Difficulty: Hard
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

int findIntegers(int n) {

}
```

Go Solution:

```
// Problem: Non-negative Integers without Consecutive Ones
// Difficulty: Hard
// Tags: dp
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

func findIntegers(n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findIntegers(n: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func findIntegers(_ n: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Non-negative Integers without Consecutive Ones
// Difficulty: Hard
// Tags: dp
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn find_integers(n: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer}
def find_integers(n)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer $n
* @return Integer
*/
function findIntegers($n) {

}
}
```

Dart Solution:

```
class Solution {
int findIntegers(int n) {

}
}
```

Scala Solution:

```
object Solution {
def findIntegers(n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_integers(n :: integer) :: integer
def find_integers(n) do

end
end
```

Erlang Solution:

```
-spec find_integers(N :: integer()) -> integer().
find_integers(N) ->
.
```

Racket Solution:

```
(define/contract (find-integers n)
(-> exact-integer? exact-integer?)
)
```