Problem 3621: Number of Integers With Popcount-Depth Equal to K I

Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given two integers

n

and

k

.

For any positive integer

x

, define the following sequence:

p

0

= x

p

i+1

= popcount(p

i

)

for all

i >= 0

, where

popcount(y)

is the number of set bits (1's) in the binary representation of

y

.

This sequence will eventually reach the value 1.

The

popcount-depth

of

x

is defined as the

smallest

integer

d >= 0

such that

p

d

= 1

.

For example, if

x = 7

(binary representation

"111"

). Then, the sequence is:

7 → 3 → 2 → 1

, so the popcount-depth of 7 is 3.

Your task is to determine the number of integers in the range

[1, n]

whose popcount-depth is

exactly

equal to

k

.

Return the number of such integers.

Example 1:

Input:

n = 4, k = 1

Output:

2

Explanation:

The following integers in the range

[1, 4]

have popcount-depth exactly equal to 1:

x

Binary

Sequence

2

"10"

2 → 1

4

"100"

4 → 1

Thus, the answer is 2.

Example 2:

Input:

n = 7, k = 2

Output:

3

Explanation:

The following integers in the range

[1, 7]

have popcount-depth exactly equal to 2:

x

Binary

Sequence

3

"11"

3 → 2 → 1

5

"101"

5 → 2 → 1

6

"110"

6 → 2 → 1

Thus, the answer is 3.

Constraints:

1 <= n <= 10

15

0 <= k <= 5

Code Snippets

C++:

```
class Solution {
public:
long long popcountDepth(long long n, int k) {

}
};
```

Java:

```
class Solution {
public long popcountDepth(long n, int k) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def popcountDepth(self, n, k):
```

```
"""
:type n: int
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number} k
* @return {number}
*/
var popcountDepth = function(n, k) {

};
```

TypeScript:

```
function popcountDepth(n: number, k: number): number {

};
```

C#:

```
public class Solution {
public long PopcountDepth(long n, int k) {

}
}
```

C:

```
long long popcountDepth(long long n, int k) {

}
```

Go:

```
func popcountDepth(n int64, k int) int64 {

}
```

Kotlin:

```
class Solution {
fun popcountDepth(n: Long, k: Int): Long {

}
}
```

Swift:

```
class Solution {
func popcountDepth(_ n: Int, _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn popcount_depth(n: i64, k: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} k
# @return {Integer}
def popcount_depth(n, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return Integer
*/
function popcountDepth($n, $k) {
}
}
```

Dart:

```
class Solution {
int popcountDepth(int n, int k) {

}
}
```

Scala:

```
object Solution {
def popcountDepth(n: Long, k: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec popcount_depth(n :: integer, k :: integer) :: integer
def popcount_depth(n, k) do

end
end
```

Erlang:

```
-spec popcount_depth(N :: integer(), K :: integer()) -> integer().
popcount_depth(N, K) ->
.
```

Racket:

```
(define/contract (popcount-depth n k)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Integers With Popcount-Depth Equal to K I
* Difficulty: Hard
* Tags: dp, math
*
* 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:
long long popcountDepth(long long n, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Integers With Popcount-Depth Equal to K I
* Difficulty: Hard
* Tags: dp, math
*
* 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 long popcountDepth(long n, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Integers With Popcount-Depth Equal to K I
```

```
Difficulty: Hard
Tags: dp, math

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 popcountDepth(self, n: int, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Integers With Popcount-Depth Equal to K I
* Difficulty: Hard
* Tags: dp, math
*
* 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
* @param {number} k
* @return {number}
*/
var popcountDepth = function(n, k) {
};
```

TypeScript Solution:

```
/**
* Problem: Number of Integers With Popcount-Depth Equal to K I
* Difficulty: Hard
* Tags: dp, math
*
* 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 popcountDepth(n: number, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Number of Integers With Popcount-Depth Equal to K I
* Difficulty: Hard
* Tags: dp, math
*
* 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 long PopcountDepth(long n, int k) {

}
}
```

C Solution:

```
/*
* Problem: Number of Integers With Popcount-Depth Equal to K I
* Difficulty: Hard
* Tags: dp, math
*
* 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
*/

long long popcountDepth(long long n, int k) {

}
```

Go Solution:

```
// Problem: Number of Integers With Popcount-Depth Equal to K I
// Difficulty: Hard
// Tags: dp, math
//
// 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 popcountDepth(n int64, k int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun popcountDepth(n: Long, k: Int): Long {

}
}
```

Swift Solution:

```
class Solution {
func popcountDepth(_ n: Int, _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Integers With Popcount-Depth Equal to K I
// Difficulty: Hard
// Tags: dp, math
//
// 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 popcount_depth(n: i64, k: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} k
# @return {Integer}
def popcount_depth(n, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return Integer
*/
function popcountDepth($n, $k) {

}
}
```

Dart Solution:

```
class Solution {
int popcountDepth(int n, int k) {

}
}
```

Scala Solution:

```
object Solution {
def popcountDepth(n: Long, k: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec popcount_depth(n :: integer, k :: integer) :: integer
def popcount_depth(n, k) do

end
end
```

Erlang Solution:

```
-spec popcount_depth(N :: integer(), K :: integer()) -> integer().
popcount_depth(N, K) ->
.
```

Racket Solution:

```
(define/contract (popcount-depth n k)
(-> exact-integer? exact-integer? exact-integer?)
)
```