Problem 3747: Count Distinct Integers After Removing Zeros

Problem Information

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

Problem Description

You are given a

positive

integer

n

.

For every integer

x

from 1 to

n

, we write down the integer obtained by removing all zeros from the decimal representation of

x

.

Return an integer denoting the number of distinct

integers written down.

Example 1:

Input:

n = 10

Output:

9

Explanation:

The integers we wrote down are 1, 2, 3, 4, 5, 6, 7, 8, 9, 1. There are 9 distinct integers (1, 2, 3, 4, 5, 6, 7, 8, 9).

Example 2:

Input:

n = 3

Output:

3

Explanation:

The integers we wrote down are 1, 2, 3. There are 3 distinct integers (1, 2, 3).

Constraints:

1 <= n <= 10

15

Code Snippets

C++:

```
class Solution {
public:
long long countDistinct(long long n) {

}
};
```

Java:

```
class Solution {
public long countDistinct(long n) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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


};
```

TypeScript:

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


};
```

C#:

```
public class Solution {
public long CountDistinct(long n) {


}
}
```

C:

```
long long countDistinct(long long n) {


}
```

Go:

```
func countDistinct(n int64) int64 {


}
```

Kotlin:

```
class Solution {
fun countDistinct(n: Long): Long {


}
}
```

Swift:

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


}
}
```

Rust:

```
impl Solution {
pub fn count_distinct(n: i64) -> i64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

```
object Solution {
def countDistinct(n: Long): Long = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (count-distinct n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Distinct Integers After Removing Zeros
* Difficulty: Medium
* 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 countDistinct(long long n) {

}
};
```

Java Solution:

```
/**
* Problem: Count Distinct Integers After Removing Zeros
* Difficulty: Medium
* 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 countDistinct(long n) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Distinct Integers After Removing Zeros
Difficulty: Medium
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 countDistinct(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Distinct Integers After Removing Zeros
* Difficulty: Medium
* 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
* @return {number}
*/
var countDistinct = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Distinct Integers After Removing Zeros
* Difficulty: Medium
* 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 countDistinct(n: number): number {
};
```

C# Solution:

```
/*
* Problem: Count Distinct Integers After Removing Zeros
* Difficulty: Medium
* 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 CountDistinct(long n) {

}
}
```

C Solution:

```
/*
* Problem: Count Distinct Integers After Removing Zeros
* Difficulty: Medium
* 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 countDistinct(long long n) {

}
```

Go Solution:

```
// Problem: Count Distinct Integers After Removing Zeros
// Difficulty: Medium
// 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 countDistinct(n int64) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun countDistinct(n: Long): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Distinct Integers After Removing Zeros
// Difficulty: Medium
// 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 count_distinct(n: i64) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

```
object Solution {
def countDistinct(n: Long): Long = {

}
}
```

Elixir Solution:

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

```



end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (count-distinct n)
(-> exact-integer? exact-integer?)
)
```