Problem 788: Rotated Digits

Problem Information

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

Problem Description

An integer

x

is a

good

if after rotating each digit individually by 180 degrees, we get a valid number that is different from

x

. Each digit must be rotated - we cannot choose to leave it alone.

A number is valid if each digit remains a digit after rotation. For example:

0

,

1

, and

8

rotate to themselves,

2

and

5

rotate to each other (in this case they are rotated in a different direction, in other words,

2

or

5

gets mirrored),

6

and

9

rotate to each other, and

the rest of the numbers do not rotate to any other number and become invalid.

Given an integer

n

, return

the number of

good

integers in the range

[1, n]

.

Example 1:

Input:

n = 10

Output:

4

Explanation:

There are four good numbers in the range [1, 10] : 2, 5, 6, 9. Note that 1 and 10 are not good numbers, since they remain unchanged after rotating.

Example 2:

Input:

n = 1

Output:

0

Example 3:

Input:

n = 2

Output:

1

Constraints:

1 <= n <= 10

4

Code Snippets

C++:

```
class Solution {
public:
int rotatedDigits(int n) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int rotatedDigits(int n) {

}
```

Go:

```
func rotatedDigits(n int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
```

```
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (rotated-digits n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Rotated Digits
* 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:
int rotatedDigits(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Rotated Digits
* 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 int rotatedDigits(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Rotated Digits
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 rotatedDigits(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Rotated Digits
* 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 rotatedDigits = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Rotated Digits
* 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 rotatedDigits(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Rotated Digits
* 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 int RotatedDigits(int n) {

}
}
```

C Solution:

```
/*
* Problem: Rotated Digits
* 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
*/

int rotatedDigits(int n) {
```

```
}
```

Go Solution:

```
// Problem: Rotated Digits
// 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 rotatedDigits(n int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Rotated Digits
// 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 rotated_digits(n: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

```
}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (rotated-digits n)
(-> exact-integer? exact-integer?)
)
```