Problem 3032: Count Numbers With Unique Digits II

Problem Information

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

Problem Description

Given two

positive

integers

a

and

b

, return

the count of numbers having

unique

digits in the range

[a, b]

(

inclusive

).

Example 1:

Input:

a = 1, b = 20

Output:

19

Explanation:

All the numbers in the range [1, 20] have unique digits except 11. Hence, the answer is 19.

Example 2:

Input:

a = 9, b = 19

Output:

10

Explanation:

All the numbers in the range [9, 19] have unique digits except 11. Hence, the answer is 10.

Example 3:

Input:

a = 80, b = 120

Output:

27

Explanation:

There are 41 numbers in the range [80, 120], 27 of which have unique digits.

Constraints:

1 <= a <= b <= 1000

Code Snippets

C++:

```
class Solution {
public:
int numberCount(int a, int b) {

}
};
```

Java:

```
class Solution {
public int numberCount(int a, int b) {

}
}
```

Python3:

```
class Solution:
def numberCount(self, a: int, b: int) -> int:
```

Python:

```
class Solution(object):
def numberCount(self, a, b):
"""
:type a: int
```

```
:type b: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} a
* @param {number} b
* @return {number}
*/
var numberCount = function(a, b) {

};
```

TypeScript:

```
function numberCount(a: number, b: number): number {

};
```

C#:

```
public class Solution {
public int NumberCount(int a, int b) {

}
}
```

C:

```
int numberCount(int a, int b) {

}
```

Go:

```
func numberCount(a int, b int) int {

}
```

Kotlin:

```
class Solution {
fun numberCount(a: Int, b: Int): Int {

}
}
```

Swift:

```
class Solution {
func numberCount(_ a: Int, _ b: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn number_count(a: i32, b: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} a
# @param {Integer} b
# @return {Integer}
def number_count(a, b)

end
```

PHP:

```
class Solution {

/**
* @param Integer $a
* @param Integer $b
* @return Integer
*/
function numberCount($a, $b) {

}
```

```
}
```

Dart:

```
class Solution {
int numberCount(int a, int b) {

}
}
```

Scala:

```
object Solution {
def numberCount(a: Int, b: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec number_count(a :: integer, b :: integer) :: integer
def number_count(a, b) do

end
end
```

Erlang:

```
-spec number_count(A :: integer(), B :: integer()) -> integer().
number_count(A, B) ->
.
```

Racket:

```
(define/contract (number-count a b)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Numbers With Unique Digits II
* Difficulty: Easy
* Tags: dp, math, hash
*
* 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 numberCount(int a, int b) {

}
};
```

Java Solution:

```
/**
* Problem: Count Numbers With Unique Digits II
* Difficulty: Easy
* Tags: dp, math, hash
*
* 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 numberCount(int a, int b) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Numbers With Unique Digits II
Difficulty: Easy
Tags: dp, math, hash
```

```
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 numberCount(self, a: int, b: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numberCount(self, a, b):
"""
:type a: int
:type b: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Numbers With Unique Digits II
* Difficulty: Easy
* Tags: dp, math, hash
*
* 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} a
* @param {number} b
* @return {number}
*/
var numberCount = function(a, b) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Numbers With Unique Digits II
* Difficulty: Easy
* Tags: dp, math, hash
*
* 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 numberCount(a: number, b: number): number {

};
```

C# Solution:

```
/*
* Problem: Count Numbers With Unique Digits II
* Difficulty: Easy
* Tags: dp, math, hash
*
* 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 NumberCount(int a, int b) {

}
}
```

C Solution:

```
/*
* Problem: Count Numbers With Unique Digits II
* Difficulty: Easy
* Tags: dp, math, hash
*
* 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 numberCount(int a, int b) {

}
```

Go Solution:

```
// Problem: Count Numbers With Unique Digits II
// Difficulty: Easy
// Tags: dp, math, hash
//
// 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 numberCount(a int, b int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numberCount(a: Int, b: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func numberCount(_ a: Int, _ b: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Numbers With Unique Digits II
// Difficulty: Easy
// Tags: dp, math, hash
//
// 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 number_count(a: i32, b: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} a
# @param {Integer} b
# @return {Integer}
def number_count(a, b)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $a
* @param Integer $b
* @return Integer
*/
function numberCount($a, $b) {

}
}
```

Dart Solution:

```
class Solution {
int numberCount(int a, int b) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def numberCount(a: Int, b: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec number_count(a :: integer, b :: integer) :: integer
def number_count(a, b) do

end
end
```

Erlang Solution:

```
-spec number_count(A :: integer(), B :: integer()) -> integer().
number_count(A, B) ->
.
```

Racket Solution:

```
(define/contract (number-count a b)
(-> exact-integer? exact-integer? exact-integer?)
)
```