Problem 902: Numbers At Most N Given Digit Set

Problem Information

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

Problem Description

Given an array of

digits

which is sorted in

non-decreasing

order. You can write numbers using each

digits[i]

as many times as we want. For example, if

digits = ['1','3','5']

, we may write numbers such as

'13'

,

'551'

, and

'1351315'

.

Return

the number of positive integers that can be generated

that are less than or equal to a given integer

n

.

Example 1:

Input:

digits = ["1","3","5","7"], n = 100

Output:

20

Explanation:

The 20 numbers that can be written are: 1, 3, 5, 7, 11, 13, 15, 17, 31, 33, 35, 37, 51, 53, 55, 57, 71, 73, 75, 77.

Example 2:

Input:

digits = ["1","4","9"], n = 1000000000

Output:

29523

Explanation:

We can write 3 one digit numbers, 9 two digit numbers, 27 three digit numbers, 81 four digit numbers, 243 five digit numbers, 729 six digit numbers, 2187 seven digit numbers, 6561 eight digit numbers, and 19683 nine digit numbers. In total, this is 29523 integers that can be written using the digits array.

Example 3:

Input:

digits = ["7"], n = 8

Output:

1

Constraints:

1 <= digits.length <= 9

digits[i].length == 1

digits[i]

is a digit from

'1'

to

'9'

.

All the values in

digits

are

unique

.

digits

is sorted in

non-decreasing

order.

1 <= n <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int atMostNGivenDigitSet(vector<string>& digits, int n) {

}
};
```

Java:

```
class Solution {
public int atMostNGivenDigitSet(String[] digits, int n) {

}
}
```

Python3:

```
class Solution:
def atMostNGivenDigitSet(self, digits: List[str], n: int) -> int:
```

Python:

```
class Solution(object):
def atMostNGivenDigitSet(self, digits, n):
"""
:type digits: List[str]
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string[]} digits
* @param {number} n
* @return {number}
*/
var atMostNGivenDigitSet = function(digits, n) {

};
```

TypeScript:

```
function atMostNGivenDigitSet(digits: string[], n: number): number {

};
```

C#:

```
public class Solution {
public int AtMostNGivenDigitSet(string[] digits, int n) {

}
}
```

C:

```
int atMostNGivenDigitSet(char** digits, int digitsSize, int n) {

}
```

Go:

```
func atMostNGivenDigitSet(digits []string, n int) int {

}
```

Kotlin:

```
class Solution {
fun atMostNGivenDigitSet(digits: Array<String>, n: Int): Int {

}
}
```

Swift:

```
class Solution {
func atMostNGivenDigitSet(_ digits: [String], _ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn at_most_n_given_digit_set(digits: Vec<String>, n: i32) -> i32 {

}
}
```

Ruby:

```
# @param {String[]} digits
# @param {Integer} n
# @return {Integer}
def at_most_n_given_digit_set(digits, n)

end
```

PHP:

```
class Solution {
```

```
/**
* @param String[] $digits
* @param Integer $n
* @return Integer
*/
function atMostNGivenDigitSet($digits, $n) {

}
}
```

Dart:

```
class Solution {
int atMostNGivenDigitSet(List<String> digits, int n) {

}
}
```

Scala:

```
object Solution {
def atMostNGivenDigitSet(digits: Array[String], n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec at_most_n_given_digit_set(digits :: [String.t], n :: integer) ::
integer
def at_most_n_given_digit_set(digits, n) do

end
end
```

Erlang:

```
-spec at_most_n_given_digit_set(Digits :: [unicode:unicode_binary()], N ::
integer()) -> integer().
at_most_n_given_digit_set(Digits, N) ->
.
```

Racket:

```
(define/contract (at-most-n-given-digit-set digits n)
(-> (listof string?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Numbers At Most N Given Digit Set
* Difficulty: Hard
* Tags: array, string, dp, math, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int atMostNGivenDigitSet(vector<string>& digits, int n) {

}
};
```

Java Solution:

```
/**
* Problem: Numbers At Most N Given Digit Set
* Difficulty: Hard
* Tags: array, string, dp, math, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int atMostNGivenDigitSet(String[] digits, int n) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Numbers At Most N Given Digit Set
Difficulty: Hard
Tags: array, string, dp, math, sort, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def atMostNGivenDigitSet(self, digits: List[str], n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def atMostNGivenDigitSet(self, digits, n):
"""
:type digits: List[str]
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Numbers At Most N Given Digit Set
* Difficulty: Hard
* Tags: array, string, dp, math, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

/**
* @param {string[]} digits
* @param {number} n
* @return {number}
*/
var atMostNGivenDigitSet = function(digits, n) {

};
```

TypeScript Solution:

```
/**
* Problem: Numbers At Most N Given Digit Set
* Difficulty: Hard
* Tags: array, string, dp, math, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function atMostNGivenDigitSet(digits: string[], n: number): number {

};
```

C# Solution:

```
/*
* Problem: Numbers At Most N Given Digit Set
* Difficulty: Hard
* Tags: array, string, dp, math, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int AtMostNGivenDigitSet(string[] digits, int n) {
```

```
}
}
```

C Solution:

```
/*
* Problem: Numbers At Most N Given Digit Set
* Difficulty: Hard
* Tags: array, string, dp, math, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int atMostNGivenDigitSet(char** digits, int digitsSize, int n) {

}
```

Go Solution:

```
// Problem: Numbers At Most N Given Digit Set
// Difficulty: Hard
// Tags: array, string, dp, math, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func atMostNGivenDigitSet(digits []string, n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun atMostNGivenDigitSet(digits: Array<String>, n: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func atMostNGivenDigitSet(_ digits: [String], _ n: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Numbers At Most N Given Digit Set
// Difficulty: Hard
// Tags: array, string, dp, math, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn at_most_n_given_digit_set(digits: Vec<String>, n: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[]} digits
# @param {Integer} n
# @return {Integer}
def at_most_n_given_digit_set(digits, n)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $digits
* @param Integer $n
* @return Integer
*/
```

```
function atMostNGivenDigitSet($digits, $n) {

}
}
```

Dart Solution:

```
class Solution {
int atMostNGivenDigitSet(List<String> digits, int n) {

}
}
```

Scala Solution:

```
object Solution {
def atMostNGivenDigitSet(digits: Array[String], n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec at_most_n_given_digit_set(digits :: [String.t], n :: integer) ::
integer
def at_most_n_given_digit_set(digits, n) do

end
end
```

Erlang Solution:

```
-spec at_most_n_given_digit_set(Digits :: [unicode:unicode_binary()], N ::
integer()) -> integer().
at_most_n_given_digit_set(Digits, N) ->
.
```

Racket Solution:

```
(define/contract (at-most-n-given-digit-set digits n)
(-> (listof string?) exact-integer? exact-integer?)
)
```