Problem 639: Decode Ways II

Problem Information

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

Problem Description

A message containing letters from

A-Z

can be

encoded

into numbers using the following mapping:

'A' -> "1" 'B' -> "2" ... 'Z' -> "26"

To

decode

an encoded message, all the digits must be grouped then mapped back into letters using the reverse of the mapping above (there may be multiple ways). For example,

"11106"

can be mapped into:

"AAJF"

with the grouping

(1 1 10 6)

"KJF"

with the grouping

(11 10 6)

Note that the grouping

(1 11 06)

is invalid because

"06"

cannot be mapped into

'F'

since

"6"

is different from

"06"

.

In addition

to the mapping above, an encoded message may contain the

'*'

character, which can represent any digit from

'1'

to

'9'

(

'0'

is excluded). For example, the encoded message

"1*"

may represent any of the encoded messages

"11"

,

"12"

,

"13"

,

"14"

,

"15"

,

"16"

,

"17"

,

"18"

, or

"19"

. Decoding

"1*"

is equivalent to decoding

any

of the encoded messages it can represent.

Given a string

s

consisting of digits and

'*'

characters, return

the

number

of ways to

decode

it

.

Since the answer may be very large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

s = "*"

Output:

9

Explanation:

The encoded message can represent any of the encoded messages "1", "2", "3", "4", "5", "6", "7", "8", or "9". Each of these can be decoded to the strings "A", "B", "C", "D", "E", "F", "G", "H", and "I" respectively. Hence, there are a total of 9 ways to decode "*".

Example 2:

Input:

s = "1*"

Output:

18

Explanation:

The encoded message can represent any of the encoded messages "11", "12", "13", "14", "15", "16", "17", "18", or "19". Each of these encoded messages have 2 ways to be decoded (e.g. "11" can be decoded to "AA" or "K"). Hence, there are a total of 9 * 2 = 18 ways to decode "1*".

Example 3:

Input:

s = "2*"

Output:

15

Explanation:

The encoded message can represent any of the encoded messages "21", "22", "23", "24", "25", "26", "27", "28", or "29". "21", "22", "23", "24", "25", and "26" have 2 ways of being decoded, but "27", "28", and "29" only have 1 way. Hence, there are a total of (6 * 2) + (3 * 1) = 12 + 3 = 15 ways to decode "2*".

Constraints:

1 <= s.length <= 10

5

s[i]

is a digit or

'*'

.

Code Snippets

C++:

```
class Solution {
public:
int numDecodings(string s) {

}
};
```

Java:

```
class Solution {
public int numDecodings(String s) {

}
}
```

Python3:

```
class Solution:
def numDecodings(self, s: str) -> int:
```

Python:

```
class Solution(object):
def numDecodings(self, s):
"""
:type s: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} s
* @return {number}
*/
var numDecodings = function(s) {
```

```
};
```

TypeScript:

```
function numDecodings(s: string): number {

};
```

C#:

```
public class Solution {
public int NumDecodings(string s) {

}
}
```

C:

```
int numDecodings(char* s) {

}
```

Go:

```
func numDecodings(s string) int {

}
```

Kotlin:

```
class Solution {
fun numDecodings(s: String): Int {

}
}
```

Swift:

```
class Solution {
func numDecodings(_ s: String) -> Int {

}
```

```
}
```

Rust:

```
impl Solution {
pub fn num_decodings(s: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} s
# @return {Integer}
def num_decodings(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return Integer
*/
function numDecodings($s) {

}
}
```

Dart:

```
class Solution {
int numDecodings(String s) {

}
}
```

Scala:

```
object Solution {
def numDecodings(s: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_decodings(s :: String.t) :: integer
def num_decodings(s) do

end
end
```

Erlang:

```
-spec num_decodings(S :: unicode:unicode_binary()) -> integer().
num_decodings(S) ->
.
```

Racket:

```
(define/contract (num-decodings s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Decode Ways II
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

```
class Solution {
public:
int numDecodings(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Decode Ways II
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int numDecodings(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Decode Ways II
Difficulty: Hard
Tags: string, dp

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def numDecodings(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numDecodings(self, s):
"""
:type s: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Decode Ways II
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {string} s
* @return {number}
*/
var numDecodings = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Decode Ways II
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function numDecodings(s: string): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Decode Ways II
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* 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 NumDecodings(string s) {

}
}
```

C Solution:

```
/*
* Problem: Decode Ways II
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int numDecodings(char* s) {

}
```

Go Solution:

```
// Problem: Decode Ways II
// Difficulty: Hard
```

```
// Tags: string, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func numDecodings(s string) int {

}
```

Kotlin Solution:

```
class Solution {
fun numDecodings(s: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func numDecodings(_ s: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Decode Ways II
// Difficulty: Hard
// Tags: string, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn num_decodings(s: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {Integer}
def num_decodings(s)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return Integer
*/
function numDecodings($s) {

}
}
```

Dart Solution:

```
class Solution {
int numDecodings(String s) {

}
}
```

Scala Solution:

```
object Solution {
def numDecodings(s: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_decodings(s :: String.t) :: integer
def num_decodings(s) do
```

```
end
end
```

Erlang Solution:

```
-spec num_decodings(S :: unicode:unicode_binary()) -> integer().
num_decodings(S) ->
.
```

Racket Solution:

```
(define/contract (num-decodings s)
(-> string? exact-integer?)
)
```