Problem 91: Decode Ways

Problem Information

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

Problem Description

You have intercepted a secret message encoded as a string of numbers. The message is

decoded

via the following mapping:

"1" -> 'A'

"2" -> 'B'

...

"25" -> 'Y'

"26" -> 'Z'

However, while decoding the message, you realize that there are many different ways you can decode the message because some codes are contained in other codes (

"2"

and

"5"

vs

"25"

).

For example,

"11106"

can be decoded into:

"AAJF"

with the grouping

(1, 1, 10, 6)

"KJF"

with the grouping

(11, 10, 6)

The grouping

(1, 11, 06)

is invalid because

"06"

is not a valid code (only

"6"

is valid).

Note: there may be strings that are impossible to decode.

Given a string s containing only digits, return the

number of ways

to

decode

it. If the entire string cannot be decoded in any valid way, return

0

.

The test cases are generated so that the answer fits in a

32-bit

integer.

Example 1:

Input:

s = "12"

Output:

2

Explanation:

"12" could be decoded as "AB" (1 2) or "L" (12).

Example 2:

Input:

s = "226"

Output:

3

Explanation:

"226" could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).

Example 3:

Input:

s = "06"

Output:

0

Explanation:

"06" cannot be mapped to "F" because of the leading zero ("6" is different from "06"). In this case, the string is not a valid encoding, so return 0.

Constraints:

1 <= s.length <= 100

s

contains only digits and may contain leading zero(s).

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
* Difficulty: Medium
* 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
* Difficulty: Medium
* 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
Difficulty: Medium
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
* Difficulty: Medium
* 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
* Difficulty: Medium
* 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
* Difficulty: Medium
* 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
* Difficulty: Medium
* 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
// Difficulty: Medium
// 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
// Difficulty: Medium
// 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?)
)
```