Problem 1639: Number of Ways to Form a Target String Given a Dictionary

Problem Information

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

Problem Description

You are given a list of strings of the

same length

words

and a string

target

.

Your task is to form

target

using the given

words

under the following rules:

target

should be formed from left to right.

To form the

i

th

character (

0-indexed

) of

target

, you can choose the

k

th

character of the

j

th

string in

words

if

target[i] = words[j][k]

.

Once you use the

k

th

character of the

j

th

string of

words

, you

can no longer

use the

x

th

character of any string in

words

where

x <= k

. In other words, all characters to the left of or at index

k

become unusuable for every string.

Repeat the process until you form the string

target

.

Notice

that you can use

multiple characters

from the

same string

in

words

provided the conditions above are met.

Return

the number of ways to form

target

from

words

. Since the answer may be too large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

words = ["acca","bbbb","caca"], target = "aba"

Output:

6

Explanation:

There are 6 ways to form target. "aba" -> index 0 ("

a

cca"), index 1 ("b

b

bb"), index 3 ("cac

a

") "aba" -> index 0 ("

a

cca"), index 2 ("bb

b

b"), index 3 ("cac

a

") "aba" -> index 0 ("

a

cca"), index 1 ("b

b

bb"), index 3 ("acc

a

") "aba" -> index 0 ("

a

cca"), index 2 ("bb

b

b"), index 3 ("acc

a

") "aba" -> index 1 ("c

a

ca"), index 2 ("bb

b

b"), index 3 ("acc

a

") "aba" -> index 1 ("c

a

ca"), index 2 ("bb

b

b"), index 3 ("cac

a

")

Example 2:

Input:

words = ["abba","baab"], target = "bab"

Output:

4

Explanation:

There are 4 ways to form target. "bab" -> index 0 ("

b

aab"), index 1 ("b

a

ab"), index 2 ("ab

b

a") "bab" -> index 0 ("

b

aab"), index 1 ("b

a

ab"), index 3 ("baa

b

") "bab" -> index 0 ("

b

aab"), index 2 ("ba

a

b"), index 3 ("baa

b

") "bab" -> index 1 ("a

b

ba"), index 2 ("ba

a

b"), index 3 ("baa

b

")

Constraints:

1 <= words.length <= 1000

1 <= words[i].length <= 1000

All strings in

words

have the same length.

1 <= target.length <= 1000

words[i]

and

target

contain only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int numWays(vector<string>& words, string target) {

}
};
```

Java:

```
class Solution {
public int numWays(String[] words, String target) {

}
}
```

Python3:

```
class Solution:
def numWays(self, words: List[str], target: str) -> int:
```

Python:

```
class Solution(object):
def numWays(self, words, target):
"""
:type words: List[str]
:type target: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string[]} words
* @param {string} target
* @return {number}
*/
var numWays = function(words, target) {

};
```

TypeScript:

```
function numWays(words: string[], target: string): number {

};
```

C#:

```
public class Solution {
public int NumWays(string[] words, string target) {

}
}
```

C:

```
int numWays(char** words, int wordsSize, char* target) {

}
```

Go:

```
func numWays(words []string, target string) int {

}
```

Kotlin:

```
class Solution {
fun numWays(words: Array<String>, target: String): Int {

}
}
```

Swift:

```
class Solution {
func numWays(_ words: [String], _ target: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_ways(words: Vec<String>, target: String) -> i32 {

}
}
```

Ruby:

```
# @param {String[]} words
# @param {String} target
# @return {Integer}
def num_ways(words, target)

end
```

PHP:

```
class Solution {
```

```
/**
* @param String[] $words
* @param String $target
* @return Integer
*/
function numWays($words, $target) {

}
}
```

Dart:

```
class Solution {
int numWays(List<String> words, String target) {

}
}
```

Scala:

```
object Solution {
def numWays(words: Array[String], target: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_ways(words :: [String.t], target :: String.t) :: integer
def num_ways(words, target) do

end
end
```

Erlang:

```
-spec num_ways(Words :: [unicode:unicode_binary()], Target ::
unicode:unicode_binary()) -> integer().
num_ways(Words, Target) ->
.
```

Racket:

```
(define/contract (num-ways words target)
(-> (listof string?) string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Ways to Form a Target String Given a Dictionary
* Difficulty: Hard
* Tags: array, string, dp
*
* 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 numWays(vector<string>& words, string target) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Ways to Form a Target String Given a Dictionary
* Difficulty: Hard
* Tags: array, string, dp
*
* 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 numWays(String[] words, String target) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Number of Ways to Form a Target String Given a Dictionary
Difficulty: Hard
Tags: array, string, dp

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 numWays(self, words: List[str], target: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numWays(self, words, target):
"""
:type words: List[str]
:type target: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Ways to Form a Target String Given a Dictionary
* Difficulty: Hard
* Tags: array, string, dp
*
* 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[]} words
* @param {string} target
* @return {number}
*/
var numWays = function(words, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Ways to Form a Target String Given a Dictionary
* Difficulty: Hard
* Tags: array, string, dp
*
* 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 numWays(words: string[], target: string): number {

};
```

C# Solution:

```
/*
* Problem: Number of Ways to Form a Target String Given a Dictionary
* Difficulty: Hard
* Tags: array, string, dp
*
* 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 NumWays(string[] words, string target) {
```

```
}
}
```

C Solution:

```
/*
* Problem: Number of Ways to Form a Target String Given a Dictionary
* Difficulty: Hard
* Tags: array, string, dp
*
* 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 numWays(char** words, int wordsSize, char* target) {

}
```

Go Solution:

```
// Problem: Number of Ways to Form a Target String Given a Dictionary
// Difficulty: Hard
// Tags: array, string, dp
//
// 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 numWays(words []string, target string) int {

}
```

Kotlin Solution:

```
class Solution {
fun numWays(words: Array<String>, target: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func numWays(_ words: [String], _ target: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Ways to Form a Target String Given a Dictionary
// Difficulty: Hard
// Tags: array, string, dp
//
// 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 num_ways(words: Vec<String>, target: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[]} words
# @param {String} target
# @return {Integer}
def num_ways(words, target)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $words
* @param String $target
* @return Integer
*/
```

```
function numWays($words, $target) {

}
}
```

Dart Solution:

```
class Solution {
int numWays(List<String> words, String target) {

}
}
```

Scala Solution:

```
object Solution {
def numWays(words: Array[String], target: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_ways(words :: [String.t], target :: String.t) :: integer
def num_ways(words, target) do

end
end
```

Erlang Solution:

```
-spec num_ways(Words :: [unicode:unicode_binary()], Target ::
unicode:unicode_binary()) -> integer().
num_ways(Words, Target) ->
.
```

Racket Solution:

```
(define/contract (num-ways words target)
(-> (listof string?) string? exact-integer?)
```

```
)
```