Problem 3320: Count The Number of Winning Sequences

Problem Information

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

Problem Description

Alice and Bob are playing a fantasy battle game consisting of

n

rounds where they summon one of three magical creatures each round: a Fire Dragon, a Water Serpent, or an Earth Golem. In each round, players

simultaneously

summon their creature and are awarded points as follows:

If one player summons a Fire Dragon and the other summons an Earth Golem, the player who summoned the

Fire Dragon

is awarded a point.

If one player summons a Water Serpent and the other summons a Fire Dragon, the player who summoned the

Water Serpent

is awarded a point.

If one player summons an Earth Golem and the other summons a Water Serpent, the player who summoned the

Earth Golem

is awarded a point.

If both players summon the same creature, no player is awarded a point.

You are given a string

s

consisting of

n

characters

'F'

,

'W'

, and

'E'

, representing the sequence of creatures Alice will summon in each round:

If

s[i] == 'F'

, Alice summons a Fire Dragon.

If

s[i] == 'W'

, Alice summons a Water Serpent.

If

s[i] == 'E'

, Alice summons an Earth Golem.

Bob’s sequence of moves is unknown, but it is guaranteed that Bob will never summon the same creature in two consecutive rounds. Bob

beats

Alice if the total number of points awarded to Bob after

n

rounds is

strictly greater

than the points awarded to Alice.

Return the number of distinct sequences Bob can use to beat Alice.

Since the answer may be very large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

s = "FFF"

Output:

3

Explanation:

Bob can beat Alice by making one of the following sequences of moves:

"WFW"

,

"FWF"

, or

"WEW"

. Note that other winning sequences like

"WWE"

or

"EWW"

are invalid since Bob cannot make the same move twice in a row.

Example 2:

Input:

s = "FWEFW"

Output:

18

Explanation:

Bob can beat Alice by making one of the following sequences of moves:

"FWFWF"

,

"FWFWE"

,

"FWEFE"

,

"FWEWE"

,

"FEFWF"

,

"FEFWE"

,

"FEFEW"

,

"FEWFE"

,

"WFEFE"

,

"WFEWE"

,

"WEFWF"

,

"WEFWE"

,

"WEFEF"

,

"WEFEW"

,

"WEWFW"

,

"WEWFE"

,

"EWFWE"

, or

"EWEWE"

.

Constraints:

1 <= s.length <= 1000

s[i]

is one of

'F'

,

'W'

, or

'E'

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int countWinningSequences(char* s) {

}
```

Go:

```
func countWinningSequences(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (count-winning-sequences s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count The Number of Winning Sequences
* 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 countWinningSequences(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Count The Number of Winning Sequences
* 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 countWinningSequences(String s) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Count The Number of Winning Sequences
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 countWinningSequences(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count The Number of Winning Sequences
* 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 countWinningSequences = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Count The Number of Winning Sequences
* 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 countWinningSequences(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Count The Number of Winning Sequences
* 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 CountWinningSequences(string s) {

}
}
```

C Solution:

```
/*
* Problem: Count The Number of Winning Sequences
* 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 countWinningSequences(char* s) {

}
```

Go Solution:

```
// Problem: Count The Number of Winning Sequences
// 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 countWinningSequences(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Count The Number of Winning Sequences
// 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 count_winning_sequences(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (count-winning-sequences s)
(-> string? exact-integer?)
)
```