Problem 1079: Letter Tile Possibilities

Problem Information

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

Problem Description

You have

n

tiles

, where each tile has one letter

tiles[i]

printed on it.

Return

the number of possible non-empty sequences of letters

you can make using the letters printed on those

tiles

.

Example 1:

Input:

tiles = "AAB"

Output:

8

Explanation:

The possible sequences are "A", "B", "AA", "AB", "BA", "AAB", "ABA", "BAA".

Example 2:

Input:

tiles = "AAABBC"

Output:

188

Example 3:

Input:

tiles = "V"

Output:

1

Constraints:

1 <= tiles.length <= 7

tiles

consists of uppercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int numTilePossibilities(string tiles) {

}
};
```

Java:

```
class Solution {
public int numTilePossibilities(String tiles) {

}
}
```

Python3:

```
class Solution:
def numTilePossibilities(self, tiles: str) -> int:
```

Python:

```
class Solution(object):
def numTilePossibilities(self, tiles):
"""
:type tiles: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} tiles
* @return {number}
*/
var numTilePossibilities = function(tiles) {

};
```

TypeScript:

```
function numTilePossibilities(tiles: string): number {

};
```

C#:

```
public class Solution {
public int NumTilePossibilities(string tiles) {

}
}
```

C:

```
int numTilePossibilities(char* tiles) {

}
```

Go:

```
func numTilePossibilities(tiles string) int {

}
```

Kotlin:

```
class Solution {
fun numTilePossibilities(tiles: String): Int {

}
}
```

Swift:

```
class Solution {
func numTilePossibilities(_ tiles: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_tile_possibilities(tiles: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} tiles
# @return {Integer}
def num_tile_possibilities(tiles)

end
```

PHP:

```
class Solution {

/**
* @param String $tiles
* @return Integer
*/
function numTilePossibilities($tiles) {

}
}
```

Dart:

```
class Solution {
int numTilePossibilities(String tiles) {

}
}
```

Scala:

```
object Solution {
def numTilePossibilities(tiles: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_tile_possibilities(tiles :: String.t) :: integer
def num_tile_possibilities(tiles) do

end
end
```

Erlang:

```
-spec num_tile_possibilities(Tiles :: unicode:unicode_binary()) -> integer().
num_tile_possibilities(Tiles) ->
.
```

Racket:

```
(define/contract (num-tile-possibilities tiles)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Letter Tile Possibilities
* Difficulty: Medium
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int numTilePossibilities(string tiles) {

}
};
```

Java Solution:

```
/**
* Problem: Letter Tile Possibilities
* Difficulty: Medium
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int numTilePossibilities(String tiles) {

}
}
```

Python3 Solution:

```
"""
Problem: Letter Tile Possibilities
Difficulty: Medium
Tags: string, hash

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

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

Python Solution:

```
class Solution(object):
def numTilePossibilities(self, tiles):
"""
:type tiles: str
:rtype: int
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Letter Tile Possibilities
* Difficulty: Medium
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string} tiles
* @return {number}
*/
var numTilePossibilities = function(tiles) {

};
```

TypeScript Solution:

```
/**
* Problem: Letter Tile Possibilities
* Difficulty: Medium
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function numTilePossibilities(tiles: string): number {

};
```

C# Solution:

```
/*
* Problem: Letter Tile Possibilities
* Difficulty: Medium
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int NumTilePossibilities(string tiles) {

}
}
```

C Solution:

```
/*
* Problem: Letter Tile Possibilities
* Difficulty: Medium
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int numTilePossibilities(char* tiles) {

}
```

Go Solution:

```
// Problem: Letter Tile Possibilities
// Difficulty: Medium
// Tags: string, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map
```

```
func numTilePossibilities(tiles string) int {

}
```

Kotlin Solution:

```
class Solution {
fun numTilePossibilities(tiles: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func numTilePossibilities(_ tiles: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Letter Tile Possibilities
// Difficulty: Medium
// Tags: string, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn num_tile_possibilities(tiles: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} tiles
# @return {Integer}
def num_tile_possibilities(tiles)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $tiles
* @return Integer
*/
function numTilePossibilities($tiles) {

}
}
```

Dart Solution:

```
class Solution {
int numTilePossibilities(String tiles) {

}
}
```

Scala Solution:

```
object Solution {
def numTilePossibilities(tiles: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_tile_possibilities(tiles :: String.t) :: integer
def num_tile_possibilities(tiles) do

end
end
```

Erlang Solution:

```
-spec num_tile_possibilities(Tiles :: unicode:unicode_binary()) -> integer().
num_tile_possibilities(Tiles) ->
.
```

Racket Solution:

```
(define/contract (num-tile-possibilities tiles)
(-> string? exact-integer?)
)
```