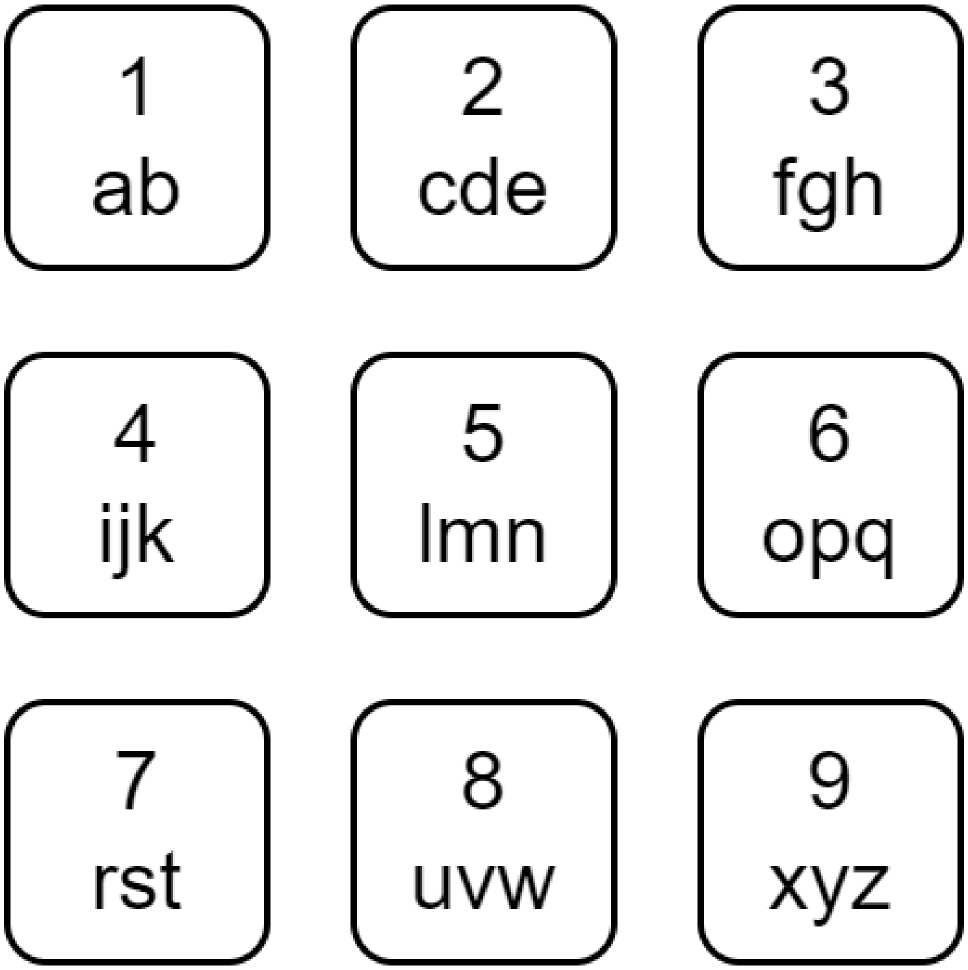
Problem 2950: Number of Divisible Substrings

Problem Information

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

Problem Description

Each character of the English alphabet has been mapped to a digit as shown below.



A string is

divisible

if the sum of the mapped values of its characters is divisible by its length.

Given a string

s

, return

the number of

divisible substrings

of

s

.

A

substring

is a contiguous non-empty sequence of characters within a string.

Example 1:

Substring

Mapped

Sum

Length

Divisible?

a

1

1

1

Yes

s

7

7

1

Yes

d

2

2

1

Yes

f

3

3

1

Yes

as

1, 7

8

2

Yes

sd

7, 2

9

2

No

df

2, 3

5

2

No

asd

1, 7, 2

10

3

No

sdf

7, 2, 3

12

3

Yes

asdf

1, 7, 2, 3

13

4

No

Input:

word = "asdf"

Output:

6

Explanation:

The table above contains the details about every substring of word, and we can see that 6 of them are divisible.

Example 2:

Input:

word = "bdh"

Output:

4

Explanation:

The 4 divisible substrings are: "b", "d", "h", "bdh". It can be shown that there are no other substrings of word that are divisible.

Example 3:

Input:

word = "abcd"

Output:

6

Explanation:

The 6 divisible substrings are: "a", "b", "c", "d", "ab", "cd". It can be shown that there are no other substrings of word that are divisible.

Constraints:

1 <= word.length <= 2000

word

consists only of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
```

```
int countDivisibleSubstrings(string word) {

}
};
```

Java:

```
class Solution {
public int countDivisibleSubstrings(String word) {

}
}
```

Python3:

```
class Solution:
def countDivisibleSubstrings(self, word: str) -> int:
```

Python:

```
class Solution(object):
def countDivisibleSubstrings(self, word):
"""
:type word: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} word
* @return {number}
*/
var countDivisibleSubstrings = function(word) {

};
```

TypeScript:

```
function countDivisibleSubstrings(word: string): number {

};
```

C#:

```
public class Solution {
public int CountDivisibleSubstrings(string word) {

}
}
```

C:

```
int countDivisibleSubstrings(char* word) {

}
```

Go:

```
func countDivisibleSubstrings(word string) int {

}
```

Kotlin:

```
class Solution {
fun countDivisibleSubstrings(word: String): Int {

}
}
```

Swift:

```
class Solution {
func countDivisibleSubstrings(_ word: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_divisible_substrings(word: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} word
# @return {Integer}
def count_divisible_substrings(word)

end
```

PHP:

```
class Solution {

/**
* @param String $word
* @return Integer
*/
function countDivisibleSubstrings($word) {

}
}
```

Dart:

```
class Solution {
int countDivisibleSubstrings(String word) {

}
}
```

Scala:

```
object Solution {
def countDivisibleSubstrings(word: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_divisible_substrings(word :: String.t) :: integer
def count_divisible_substrings(word) do
```

```

end
end
```

Erlang:

```
-spec count_divisible_substrings(Word :: unicode:unicode_binary()) ->
integer().
count_divisible_substrings(Word) ->
.
```

Racket:

```
(define/contract (count-divisible-substrings word)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Divisible Substrings
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
int countDivisibleSubstrings(string word) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Divisible Substrings
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int countDivisibleSubstrings(String word) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Divisible Substrings
Difficulty: Medium
Tags: array, string, tree, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

```
class Solution(object):
def countDivisibleSubstrings(self, word):
"""
:type word: str
:rtype: int
"""
```

JavaScript Solution:

```javascript
/**
* Problem: Number of Divisible Substrings
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {string} word
* @return {number}
*/
var countDivisibleSubstrings = function(word) {

};
```

TypeScript Solution:

```typescript
/**
* Problem: Number of Divisible Substrings
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function countDivisibleSubstrings(word: string): number {

};
```

C# Solution:

```csharp
/*
* Problem: Number of Divisible Substrings
* Difficulty: Medium
* Tags: array, string, tree, hash
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int CountDivisibleSubstrings(string word) {

}
}
```

C Solution:

```
/*
* Problem: Number of Divisible Substrings
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int countDivisibleSubstrings(char* word) {

}
```

Go Solution:

```
// Problem: Number of Divisible Substrings
// Difficulty: Medium
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func countDivisibleSubstrings(word string) int {

}
```

Kotlin Solution:

```
class Solution {
fun countDivisibleSubstrings(word: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func countDivisibleSubstrings(_ word: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Divisible Substrings
// Difficulty: Medium
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn count_divisible_substrings(word: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} word
# @return {Integer}
def count_divisible_substrings(word)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $word
* @return Integer
*/
function countDivisibleSubstrings($word) {

}
}
```

Dart Solution:

```
class Solution {
int countDivisibleSubstrings(String word) {

}
}
```

Scala Solution:

```
object Solution {
def countDivisibleSubstrings(word: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_divisible_substrings(word :: String.t) :: integer
def count_divisible_substrings(word) do

end
end
```

Erlang Solution:

```
-spec count_divisible_substrings(Word :: unicode:unicode_binary()) ->
integer().
count_divisible_substrings(Word) ->
.
```

Racket Solution:

```
(define/contract (count-divisible-substrings word)
(-> string? exact-integer?)
)
```