Problem 1945: Sum of Digits of String After Convert

Problem Information

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

Problem Description

You are given a string

s

consisting of lowercase English letters, and an integer

k

. Your task is to

convert

the string into an integer by a special process, and then

transform

it by summing its digits repeatedly

k

times. More specifically, perform the following steps:

Convert

s

into an integer by replacing each letter with its position in the alphabet (i.e. replace

'a'

with

1

,

'b'

with

2

, ...,

'z'

with

26

).

T

ransform

the integer by replacing it with the

sum of its digits

.

Repeat the

transform

operation (step 2)

k

times

in total.

For example, if

s = "zbax"

and

k = 2

, then the resulting integer would be

8

by the following operations:

Convert

:

"zbax" → "(26)(2)(1)(24)" → "262124" → 262124

Transform #1

:

262124 → 2 + 6 + 2 + 1 + 2 + 4 → 17

Transform #2

:

17 → 1 + 7 → 8

Return the

resulting

integer

after performing the

operations

described above.

Example 1:

Input:

s = "iiii", k = 1

Output:

36

Explanation:

The operations are as follows:

- Convert: "iiii" → "(9)(9)(9)(9)" → "9999" → 9999

- Transform #1: 9999 → 9 + 9 + 9 + 9 → 36

Thus the resulting integer is 36.

Example 2:

Input:

s = "leetcode", k = 2

Output:

6

Explanation:

The operations are as follows:

- Convert: "leetcode" → "(12)(5)(5)(20)(3)(15)(4)(5)" → "12552031545" → 12552031545

- Transform #1: 12552031545 → 1 + 2 + 5 + 5 + 2 + 0 + 3 + 1 + 5 + 4 + 5 → 33

- Transform #2: 33 → 3 + 3 → 6

Thus the resulting integer is 6.

Example 3:

Input:

s = "zbax", k = 2

Output:

8

Constraints:

1 <= s.length <= 100

1 <= k <= 10

s

consists of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int getLucky(string s, int k) {

}
};
```

Java:

```
class Solution {
public int getLucky(String s, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

```
function getLucky(s: string, k: number): number {

};
```

C#:

```
public class Solution {
public int GetLucky(string s, int k) {

}
}
```

C:

```
int getLucky(char* s, int k) {

}
```

Go:

```
func getLucky(s string, k int) int {

}
```

Kotlin:

```
class Solution {
fun getLucky(s: String, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func getLucky(_ s: String, _ k: Int) -> Int {

}
```

```
}
```

Rust:

```
impl Solution {
pub fn get_lucky(s: String, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer} k
# @return {Integer}
def get_lucky(s, k)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return Integer
*/
function getLucky($s, $k) {

}
}
```

Dart:

```
class Solution {
int getLucky(String s, int k) {

}
}
```

Scala:

```
object Solution {
def getLucky(s: String, k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec get_lucky(s :: String.t, k :: integer) :: integer
def get_lucky(s, k) do

end
end
```

Erlang:

```
-spec get_lucky(S :: unicode:unicode_binary(), K :: integer()) -> integer().
get_lucky(S, K) ->
.
```

Racket:

```
(define/contract (get-lucky s k)
(-> string? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum of Digits of String After Convert
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
class Solution {
public:
int getLucky(string s, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of Digits of String After Convert
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int getLucky(String s, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum of Digits of String After Convert
Difficulty: Easy
Tags: string

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def getLucky(self, s: str, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Digits of String After Convert
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string} s
* @param {number} k
* @return {number}
*/
var getLucky = function(s, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum of Digits of String After Convert
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
function getLucky(s: string, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Sum of Digits of String After Convert
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int GetLucky(string s, int k) {

}
}
```

C Solution:

```
/*
* Problem: Sum of Digits of String After Convert
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int getLucky(char* s, int k) {

}
```

Go Solution:

```
// Problem: Sum of Digits of String After Convert
// Difficulty: Easy
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func getLucky(s string, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun getLucky(s: String, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func getLucky(_ s: String, _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Sum of Digits of String After Convert
// Difficulty: Easy
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn get_lucky(s: String, k: i32) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer} k
# @return {Integer}
def get_lucky(s, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return Integer
*/
function getLucky($s, $k) {

}
}
```

Dart Solution:

```
class Solution {
int getLucky(String s, int k) {

}
}
```

Scala Solution:

```
object Solution {
def getLucky(s: String, k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec get_lucky(s :: String.t, k :: integer) :: integer
def get_lucky(s, k) do

end
end
```

Erlang Solution:

```
-spec get_lucky(S :: unicode:unicode_binary(), K :: integer()) -> integer().
get_lucky(S, K) ->
.
```

Racket Solution:

```
(define/contract (get-lucky s k)
(-> string? exact-integer? exact-integer?)
)
```