Problem 3210: Find the Encrypted String

Problem Information

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

Problem Description

You are given a string

s

and an integer

k

. Encrypt the string using the following algorithm:

For each character

c

in

s

, replace

c

with the

k

th

character after

c

in the string (in a cyclic manner).

Return the

encrypted string

.

Example 1:

Input:

s = "dart", k = 3

Output:

"tdar"

Explanation:

For

i = 0

, the 3

rd

character after

'd'

is

't'

.

For

i = 1

, the 3

rd

character after

'a'

is

'd'

.

For

i = 2

, the 3

rd

character after

'r'

is

'a'

.

For

i = 3

, the 3

rd

character after

't'

is

'r'

.

Example 2:

Input:

s = "aaa", k = 1

Output:

"aaa"

Explanation:

As all the characters are the same, the encrypted string will also be the same.

Constraints:

1 <= s.length <= 100

1 <= k <= 10

4

s

consists only of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Find the Encrypted String
* 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:
string getEncryptedString(string s, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Encrypted String
* 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 String getEncryptedString(String s, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Encrypted String
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 getEncryptedString(self, s: str, k: int) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the Encrypted String
* 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 {string}
*/
var getEncryptedString = function(s, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Encrypted String
* 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 getEncryptedString(s: string, k: number): string {

};
```

C# Solution:

```
/*
* Problem: Find the Encrypted String
* 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 string GetEncryptedString(string s, int k) {

}
}
```

C Solution:

```
/*
* Problem: Find the Encrypted String
* 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
*/

char* getEncryptedString(char* s, int k) {

}
```

Go Solution:

```
// Problem: Find the Encrypted String
// 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 getEncryptedString(s string, k int) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find the Encrypted String
// 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_encrypted_string(s: String, k: i32) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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