Problem 2138: Divide a String Into Groups of Size k

Problem Information

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

Problem Description

A string

s

can be partitioned into groups of size

k

using the following procedure:

The first group consists of the first

k

characters of the string, the second group consists of the next

k

characters of the string, and so on. Each element can be a part of

exactly one

group.

For the last group, if the string

does not

have

k

characters remaining, a character

fill

is used to complete the group.

Note that the partition is done so that after removing the

fill

character from the last group (if it exists) and concatenating all the groups in order, the resultant string should be

s

.

Given the string

s

, the size of each group

k

and the character

fill

, return

a string array denoting the

composition of every group

s

has been divided into, using the above procedure

.

Example 1:

Input:

s = "abcdefghi", k = 3, fill = "x"

Output:

["abc","def","ghi"]

Explanation:

The first 3 characters "abc" form the first group. The next 3 characters "def" form the second group. The last 3 characters "ghi" form the third group. Since all groups can be completely filled by characters from the string, we do not need to use fill. Thus, the groups formed are "abc", "def", and "ghi".

Example 2:

Input:

s = "abcdefghij", k = 3, fill = "x"

Output:

["abc","def","ghi","jxx"]

Explanation:

Similar to the previous example, we are forming the first three groups "abc", "def", and "ghi". For the last group, we can only use the character 'j' from the string. To complete this group,

we add 'x' twice. Thus, the 4 groups formed are "abc", "def", "ghi", and "jxx".

Constraints:

1 <= s.length <= 100

s

consists of lowercase English letters only.

1 <= k <= 100

fill

is a lowercase English letter.

Code Snippets

C++:

```
class Solution {
public:
vector<string> divideString(string s, int k, char fill) {

}
};
```

Java:

```
class Solution {
public String[] divideString(String s, int k, char fill) {

}
}
```

Python3:

```
class Solution:
def divideString(self, s: str, k: int, fill: str) -> List[str]:
```

Python:

```
class Solution(object):
def divideString(self, s, k, fill):
"""
:type s: str
:type k: int
:type fill: str
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {number} k
* @param {character} fill
* @return {string[]}
*/
var divideString = function(s, k, fill) {

};
```

TypeScript:

```
function divideString(s: string, k: number, fill: string): string[] {

};
```

C#:

```
public class Solution {
public string[] DivideString(string s, int k, char fill) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** divideString(char* s, int k, char fill, int* returnSize) {
```

```
}
```

Go:

```
func divideString(s string, k int, fill byte) []string {

}
```

Kotlin:

```
class Solution {
fun divideString(s: String, k: Int, fill: Char): Array<String> {

}
}
```

Swift:

```
class Solution {
func divideString(_ s: String, _ k: Int, _ fill: Character) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn divide_string(s: String, k: i32, fill: char) -> Vec<String> {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer} k
# @param {Character} fill
# @return {String[]}
def divide_string(s, k, fill)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @param String $fill
* @return String[]
*/
function divideString($s, $k, $fill) {

}
}
```

Dart:

```
class Solution {
List<String> divideString(String s, int k, String fill) {

}
}
```

Scala:

```
object Solution {
def divideString(s: String, k: Int, fill: Char): Array[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec divide_string(s :: String.t, k :: integer, fill :: char) :: [String.t]
def divide_string(s, k, fill) do

end
end
```

Erlang:

```
-spec divide_string(S :: unicode:unicode_binary(), K :: integer(), Fill ::
char()) -> [unicode:unicode_binary()].
divide_string(S, K, Fill) ->
.
```

Racket:

```
(define/contract (divide-string s k fill)
(-> string? exact-integer? char? (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Divide a String Into Groups of Size k
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<string> divideString(string s, int k, char fill) {

}
};
```

Java Solution:

```
/**
* Problem: Divide a String Into Groups of Size k
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public String[] divideString(String s, int k, char fill) {

}
}
```

Python3 Solution:

```
"""
Problem: Divide a String Into Groups of Size k
Difficulty: Easy
Tags: array, string

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def divideString(self, s: str, k: int, fill: str) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def divideString(self, s, k, fill):
"""
:type s: str
:type k: int
:type fill: str
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Divide a String Into Groups of Size k
```

```
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* 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
* @param {character} fill
* @return {string[]}
*/
var divideString = function(s, k, fill) {

};
```

TypeScript Solution:

```
/**
* Problem: Divide a String Into Groups of Size k
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function divideString(s: string, k: number, fill: string): string[] {

};
```

C# Solution:

```
/*
* Problem: Divide a String Into Groups of Size k
* Difficulty: Easy
* Tags: array, string
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public string[] DivideString(string s, int k, char fill) {

}
}
```

C Solution:

```
/*
* Problem: Divide a String Into Groups of Size k
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** divideString(char* s, int k, char fill, int* returnSize) {

}
```

Go Solution:

```
// Problem: Divide a String Into Groups of Size k
// Difficulty: Easy
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func divideString(s string, k int, fill byte) []string {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun divideString(s: String, k: Int, fill: Char): Array<String> {

}
}
```

Swift Solution:

```
class Solution {
func divideString(_ s: String, _ k: Int, _ fill: Character) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Divide a String Into Groups of Size k
// Difficulty: Easy
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn divide_string(s: String, k: i32, fill: char) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer} k
# @param {Character} fill
# @return {String[]}
```

```
def divide_string(s, k, fill)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @param String $fill
* @return String[]
*/
function divideString($s, $k, $fill) {

}
}
```

Dart Solution:

```
class Solution {
List<String> divideString(String s, int k, String fill) {

}
}
```

Scala Solution:

```
object Solution {
def divideString(s: String, k: Int, fill: Char): Array[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec divide_string(s :: String.t, k :: integer, fill :: char) :: [String.t]
def divide_string(s, k, fill) do
```

```
end
end
```

Erlang Solution:

```
-spec divide_string(S :: unicode:unicode_binary(), K :: integer(), Fill ::
char()) -> [unicode:unicode_binary()].
divide_string(S, K, Fill) ->
.
```

Racket Solution:

```
(define/contract (divide-string s k fill)
(-> string? exact-integer? char? (listof string?))
)
```