Problem 249: Group Shifted Strings

Problem Information

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

Problem Description

Perform the following shift operations on a string:

Right shift

: Replace every letter with the

successive

letter of the English alphabet, where 'z' is replaced by 'a'. For example,

"abc"

can be right-shifted to

"bcd"

or

"xyz"

can be right-shifted to

"yza"

.

Left shift

: Replace every letter with the

preceding

letter of the English alphabet, where 'a' is replaced by 'z'. For example,

"bcd"

can be left-shifted to

"abc"

or

"yza"

can be left-shifted to

"xyz"

.

We can keep shifting the string in both directions to form an

endless

shifting sequence

.

For example, shift

"abc"

to form the sequence:

... <-> "abc" <-> "bcd" <-> ... <-> "xyz" <-> "yza" <-> ...

.

<-> "zab" <-> "abc" <-> ...

You are given an array of strings

strings

, group together all

strings[i]

that belong to the same shifting sequence. You may return the answer in

any order

.

Example 1:

Input:

strings = ["abc","bcd","acef","xyz","az","ba","a","z"]

Output:

[["acef"],["a","z"],["abc","bcd","xyz"],["az","ba"]]

Example 2:

Input:

strings = ["a"]

Output:

[["a"]]

Constraints:

1 <= strings.length <= 200

1 <= strings[i].length <= 50

strings[i]

consists of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
vector<vector<string>> groupStrings(vector<string>& strings) {

}
};
```

Java:

```
class Solution {
public List<List<String>> groupStrings(String[] strings) {

}
}
```

Python3:

```
class Solution:
def groupStrings(self, strings: List[str]) -> List[List[str]]:
```

Python:

```
class Solution(object):
def groupStrings(self, strings):
"""
:type strings: List[str]
```

```
:rtype: List[List[str]]
"""
```

JavaScript:

```
/**
* @param {string[]} strings
* @return {string[][]}
*/
var groupStrings = function(strings) {

};
```

TypeScript:

```
function groupStrings(strings: string[]): string[][] {

};
```

C#:

```
public class Solution {
public IList<IList<string>> GroupStrings(string[] strings) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
char*** groupStrings(char** strings, int stringsSize, int* returnSize, int**
returnColumnSizes) {

}
```

Go:

```
func groupStrings(strings []string) [][]string {

}
```

Kotlin:

```
class Solution {
fun groupStrings(strings: Array<String>): List<List<String>> {

}
}
```

Swift:

```
class Solution {
func groupStrings(_ strings: [String]) -> [[String]] {

}
}
```

Rust:

```
impl Solution {
pub fn group_strings(strings: Vec<String>) -> Vec<Vec<String>> {

}
}
```

Ruby:

```
# @param {String[]} strings
# @return {String[][]}
def group_strings(strings)

end
```

PHP:

```
class Solution {

/**
* @param String[] $strings
* @return String[][]
*/
function groupStrings($strings) {

}
}
```

Dart:

```
class Solution {
List<List<String>> groupStrings(List<String> strings) {

}
}
```

Scala:

```
object Solution {
def groupStrings(strings: Array[String]): List[List[String]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec group_strings(strings :: [String.t]) :: [[String.t]]
def group_strings(strings) do

end
end
```

Erlang:

```
-spec group_strings(Strings :: [unicode:unicode_binary()]) ->
[[unicode:unicode_binary()]].
group_strings(Strings) ->
.
```

Racket:

```
(define/contract (group-strings strings)
(-> (listof string?) (listof (listof string?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Group Shifted Strings
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
vector<vector<string>> groupStrings(vector<string>& strings) {

}
};
```

Java Solution:

```
/**
* Problem: Group Shifted Strings
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public List<List<String>> groupStrings(String[] strings) {

}
}
```

Python3 Solution:

```
"""
Problem: Group Shifted Strings
Difficulty: Medium
Tags: array, string, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def groupStrings(self, strings: List[str]) -> List[List[str]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def groupStrings(self, strings):
"""
:type strings: List[str]
:rtype: List[List[str]]
"""
```

JavaScript Solution:

```
/**
* Problem: Group Shifted Strings
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string[]} strings
* @return {string[][]}
*/
```

```
var groupStrings = function(strings) {

};
```

TypeScript Solution:

```
/**
* Problem: Group Shifted Strings
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function groupStrings(strings: string[]): string[][] {

};
```

C# Solution:

```
/*
* Problem: Group Shifted Strings
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public IList<IList<string>> GroupStrings(string[] strings) {

}
}
```

C Solution:

```
/*
* Problem: Group Shifted Strings
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
char*** groupStrings(char** strings, int stringsSize, int* returnSize, int**
returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Group Shifted Strings
// Difficulty: Medium
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func groupStrings(strings []string) [][]string {

}
```

Kotlin Solution:

```
class Solution {
fun groupStrings(strings: Array<String>): List<List<String>> {

}
}
```

Swift Solution:

```swift
class Solution {
func groupStrings(_ strings: [String]) -> [[String]] {

}
}
```

Rust Solution:

```rust
// Problem: Group Shifted Strings
// Difficulty: Medium
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn group_strings(strings: Vec<String>) -> Vec<Vec<String>> {

}
}
```

Ruby Solution:

```ruby
# @param {String[]} strings
# @return {String[][]}
def group_strings(strings)

end
```

PHP Solution:

```php
class Solution {

/**
* @param String[] $strings
* @return String[][]
*/
function groupStrings($strings) {
}
}
```

Dart Solution:

```
class Solution {
List<List<String>> groupStrings(List<String> strings) {

}
}
```

Scala Solution:

```
object Solution {
def groupStrings(strings: Array[String]): List[List[String]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec group_strings(strings :: [String.t]) :: [[String.t]]
def group_strings(strings) do

end
end
```

Erlang Solution:

```
-spec group_strings(Strings :: [unicode:unicode_binary()]) ->
[[unicode:unicode_binary()]].
group_strings(Strings) ->
.
```

Racket Solution:

```
(define/contract (group-strings strings)
(-> (listof string?) (listof (listof string?)))
)
```