Problem 49: Group Anagrams

Problem Information

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

Problem Description

Given an array of strings

strs

, group the

anagrams

together. You can return the answer in

any order

.

Example 1:

Input:

strs = ["eat","tea","tan","ate","nat","bat"]

Output:

[["bat"],["nat","tan"],["ate","eat","tea"]]

Explanation:

There is no string in strs that can be rearranged to form

"bat"

.

The strings

"nat"

and

"tan"

are anagrams as they can be rearranged to form each other.

The strings

"ate"

,

"eat"

, and

"tea"

are anagrams as they can be rearranged to form each other.

Example 2:

Input:

strs = [""]

Output:

[[""]]

Example 3:

Input:

strs = ["a"]

Output:

[["a"]]

Constraints:

1 <= strs.length <= 10

4

0 <= strs[i].length <= 100

strs[i]

consists of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
vector<vector<string>> groupAnagrams(vector<string>& strs) {

}
};
```

Java:

```
class Solution {
public List<List<String>> groupAnagrams(String[] strs) {

}


}
```

Python3:

```
class Solution:
def groupAnagrams(self, strs: List[str]) -> List[List[str]]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function groupAnagrams(strs: string[]): string[][] {

};
```

C#:

```
public class Solution {
public IList<IList<string>> GroupAnagrams(string[] strs) {

}
}
```

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*** groupAnagrams(char** strs, int strsSize, int* returnSize, int**
returnColumnSizes) {

}
```

Go:

```
func groupAnagrams(strs []string) [][]string {

}
```

Kotlin:

```
class Solution {
fun groupAnagrams(strs: Array<String>): List<List<String>> {

}
}
```

Swift:

```
class Solution {
func groupAnagrams(_ strs: [String]) -> [[String]] {

}
}
```

Rust:

```
impl Solution {
pub fn group_anagrams(strs: Vec<String>) -> Vec<Vec<String>> {

}
}
```

Ruby:

```
# @param {String[]} strs
# @return {String[][]}
def group_anagrams(strs)

end
```

PHP:

```
class Solution {

/**
* @param String[] $strs
* @return String[][]
*/
function groupAnagrams($strs) {

}
}
```

Dart:

```
class Solution {
List<List<String>> groupAnagrams(List<String> strs) {

}
}
```

Scala:

```
object Solution {
def groupAnagrams(strs: Array[String]): List[List[String]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec group_anagrams(strs :: [String.t]) :: [[String.t]]
def group_anagrams(strs) do

end
end
```

Erlang:

```
-spec group_anagrams(Strs :: [unicode:unicode_binary()]) ->
[[unicode:unicode_binary()]].
group_anagrams(Strs) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Group Anagrams
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* 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>> groupAnagrams(vector<string>& strs) {

}
};
```

Java Solution:

```
/**
* Problem: Group Anagrams
* Difficulty: Medium
* Tags: array, string, hash, sort
*
```

```
* 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>> groupAnagrams(String[] strs) {

}
}
```

Python3 Solution:

```
"""
Problem: Group Anagrams
Difficulty: Medium
Tags: array, string, hash, sort

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 groupAnagrams(self, strs: List[str]) -> List[List[str]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Group Anagrams
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* 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[]} strs
* @return {string[][]}
*/
var groupAnagrams = function(strs) {

};
```

TypeScript Solution:

```
/**
* Problem: Group Anagrams
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* 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 groupAnagrams(strs: string[]): string[][] {

};
```

C# Solution:

```
/*
* Problem: Group Anagrams
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* 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>> GroupAnagrams(string[] strs) {

}
}
```

C Solution:

```
/*
* Problem: Group Anagrams
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* 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*** groupAnagrams(char** strs, int strsSize, int* returnSize, int**
returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Group Anagrams
// Difficulty: Medium
// Tags: array, string, hash, sort
//
// 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 groupAnagrams(strs []string) [][]string {

}
```

Kotlin Solution:

```
class Solution {
fun groupAnagrams(strs: Array<String>): List<List<String>> {

}
}
```

Swift Solution:

```
class Solution {
func groupAnagrams(_ strs: [String]) -> [[String]] {

}
}
```

Rust Solution:

```
// Problem: Group Anagrams
// Difficulty: Medium
// Tags: array, string, hash, sort
//
// 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_anagrams(strs: Vec<String>) -> Vec<Vec<String>> {

}
}
```

Ruby Solution:

```
# @param {String[]} strs
# @return {String[][]}
```

```
def group_anagrams(strs)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $strs
* @return String[][]
*/
function groupAnagrams($strs) {

}
}
```

Dart Solution:

```
class Solution {
List<List<String>> groupAnagrams(List<String> strs) {

}
}
```

Scala Solution:

```
object Solution {
def groupAnagrams(strs: Array[String]): List[List[String]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec group_anagrams(strs :: [String.t]) :: [[String.t]]
def group_anagrams(strs) do

end
end
```

Erlang Solution:

```
-spec group_anagrams(Strs :: [unicode:unicode_binary()]) ->
[[unicode:unicode_binary()]].
group_anagrams(Strs) ->
.
```

Racket Solution:

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