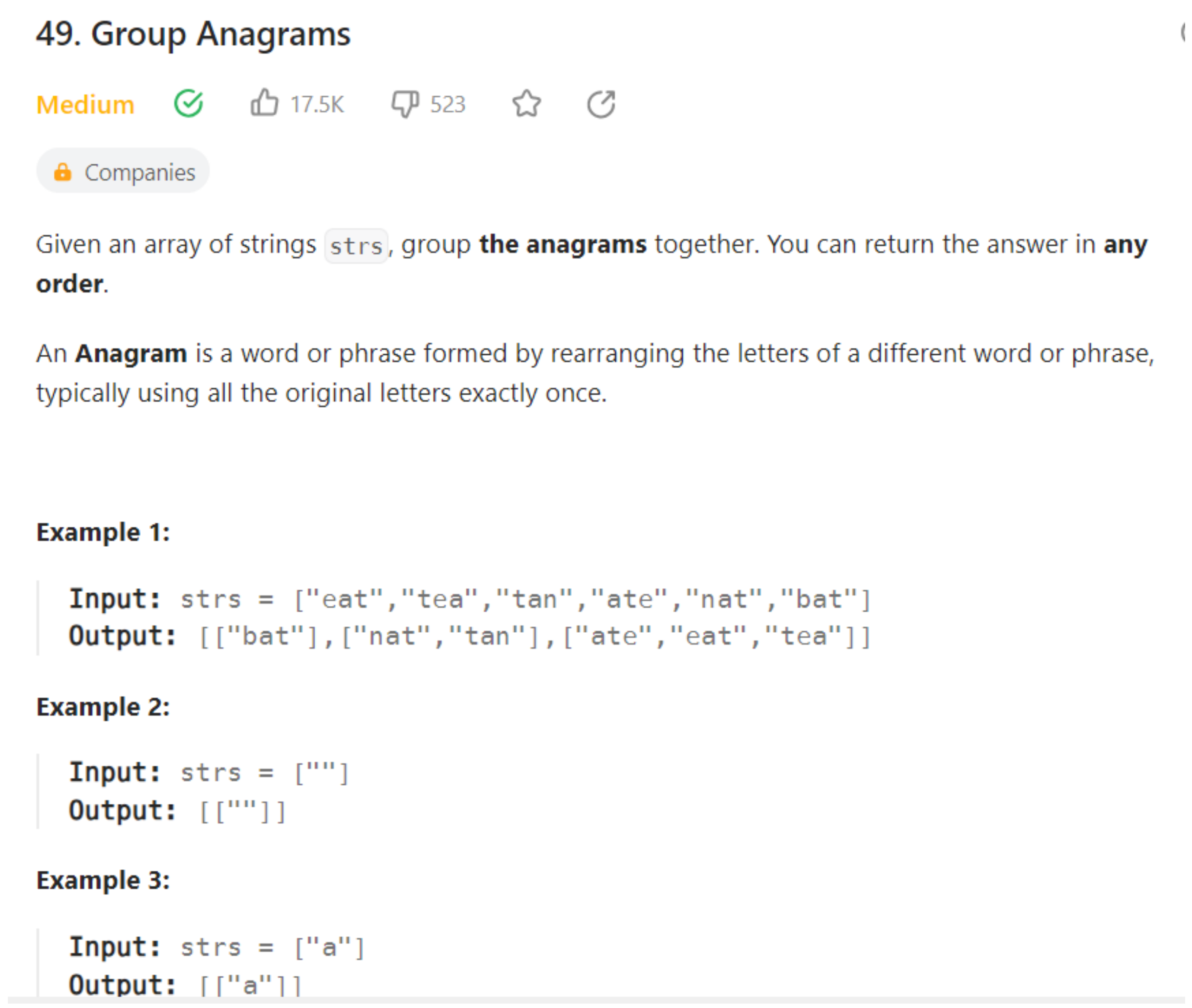

49. Group Anagrams

Given an array of strings `strs`, group **the anagrams** together. You can return the answer in **any order**.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1:

```
Input: strs = ["eat","tea","tan","ate","nat","bat"]
Output: [["bat"],["nat","tan"],["ate","eat","tea"]]
```

Example 2:

```
Input: strs = [""]
Output: [[""]]
```

Example 3:

```
Input: strs = ["a"]
Output: [["a"]]
```

Code

```
class Solution {
    func groupAnagrams(_ strs: [String]) -> [[String]] {
        if strs.count == 1 {
            return [strs]
        }
        var hash:[String:[String]] = [:]
        for i in strs {
           let sortedStr = String(i.sorted())
            hash[sortedStr, default: []].append(i)
        }
        // let res = hash.values
        return Array(hash.values)
    }
}
```

To solve this problem, the approach used in the given Swift function is based on the idea of hashing. In this approach, each string is assigned a unique hash key based on the product of the prime numbers assigned to each character in the string. The prime numbers assigned to each character are chosen in such a way that the hash keys of two strings will be equal only if they are anagrams of each other.

In the code, a dictionary is used to store the hash keys as keys and the corresponding anagrams as values. For each string in the input array, the hash key is calculated by multiplying the prime numbers assigned to each character in the string. If the hash key is already present in the dictionary, the current string is appended to the list of anagrams for that key. Otherwise, a new key-value pair is added to the dictionary with the hash key and the current string.

Finally, the function returns an array containing the values of the dictionary, which are the lists of anagrams grouped together. The order of the anagram groups in the output array can be arbitrary.

Complexity

The *time complexity* of this solution is

O(n · k)O(n \cdot k)

$O(n \cdot k)$, where

nn

n is the number of strings in the input array and

kk

k is the maximum length of a string in the input array.

The space complexity of this solution is

O(n · k)O(n \cdot k)

$O(n \cdot k)$, where

nn

n is the number of strings in the input array and

kk

k is the maximum length of a string in the input array.

Code

```
class Solution {
    func groupAnagrams(_ strs: [String]) -> [[String]] {
        let mod = 1_000_000_007
        var dict = [Int: [String]]()
        let primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83,
89, 97, 101]

        for s in strs {
            var key = 1
            for ch in s {
                key = key * primes[Int(ch.asciiValue! - Character("a").asciiValue!)] % mod
            }
            dict[key, default: []].append(s)
        }

        return Array(dict.values)
    }
}
```