Problem 2273: Find Resultant Array After Removing Anagrams

Problem Information

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

Problem Description

You are given a

0-indexed

string array

words

, where

words[i]

consists of lowercase English letters.

In one operation, select any index

i

such that

0 < i < words.length

and

words[i - 1]

and

words[i]

are

anagrams

, and

delete

words[i]

from

words

. Keep performing this operation as long as you can select an index that satisfies the conditions.

Return

words

after performing all operations

. It can be shown that selecting the indices for each operation in

any

arbitrary order will lead to the same result.

An

Anagram

is a word or phrase formed by rearranging the letters of a different word or phrase using all the original letters exactly once. For example, "dacb"

is an anagram of

"abdc"

.

Example 1:

Input:

words = ["abba","baba","bbaa","cd","cd"]

Output:

["abba","cd"]

Explanation:

One of the ways we can obtain the resultant array is by using the following operations: - Since words[2] = "bbaa" and words[1] = "baba" are anagrams, we choose index 2 and delete words[2]. Now words = ["abba","baba","cd","cd"]. - Since words[1] = "baba" and words[0] = "abba" are anagrams, we choose index 1 and delete words[1]. Now words = ["abba","cd","cd"]. - Since words[2] = "cd" and words[1] = "cd" are anagrams, we choose index 2 and delete words[2]. Now words = ["abba","cd"]. We can no longer perform any operations, so ["abba","cd"] is the final answer.

Example 2:

Input:

words = ["a","b","c","d","e"]

Output:

["a","b","c","d","e"]

Explanation:

No two adjacent strings in words are anagrams of each other, so no operations are performed.

Constraints:

1 <= words.length <= 100

1 <= words[i].length <= 10

words[i]

consists of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
vector<string> removeAnagrams(vector<string>& words) {

}
};
```

Java:

```
class Solution {
public List<String> removeAnagrams(String[] words) {

}
}
```

Python3:

```
class Solution:
def removeAnagrams(self, words: List[str]) -> List[str]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function removeAnagrams(words: string[]): string[] {

};
```

C#:

```
public class Solution {
public IList<string> RemoveAnagrams(string[] words) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** removeAnagrams(char** words, int wordsSize, int* returnSize) {

}
```

Go:

```
func removeAnagrams(words []string) []string {

}
```

Kotlin:

```
class Solution {
fun removeAnagrams(words: Array<String>): List<String> {

}
}
```

Swift:

```
class Solution {
func removeAnagrams(_ words: [String]) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn remove_anagrams(words: Vec<String>) -> Vec<String> {

}
}
```

Ruby:

```
# @param {String[]} words
# @return {String[]}
def remove_anagrams(words)

end
```

PHP:

```
class Solution {

/**
* @param String[] $words
* @return String[]
*/
function removeAnagrams($words) {

}
}
```

Dart:

```
class Solution {
List<String> removeAnagrams(List<String> words) {

}
}
```

Scala:

```
object Solution {
def removeAnagrams(words: Array[String]): List[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec remove_anagrams(words :: [String.t]) :: [String.t]
def remove_anagrams(words) do

end
end
```

Erlang:

```
-spec remove_anagrams(Words :: [unicode:unicode_binary()]) ->
[unicode:unicode_binary()].
remove_anagrams(Words) ->
.
```

Racket:

```
(define/contract (remove-anagrams words)
(-> (listof string?) (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Resultant Array After Removing Anagrams
* Difficulty: Easy
* 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<string> removeAnagrams(vector<string>& words) {

}
};
```

Java Solution:

```
/**
* Problem: Find Resultant Array After Removing Anagrams
* Difficulty: Easy
* 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<String> removeAnagrams(String[] words) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Resultant Array After Removing Anagrams
Difficulty: Easy
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 removeAnagrams(self, words: List[str]) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Resultant Array After Removing Anagrams
* Difficulty: Easy
* 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[]} words
* @return {string[]}
*/
```

```

var removeAnagrams = function(words) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Resultant Array After Removing Anagrams
* Difficulty: Easy
* 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 removeAnagrams(words: string[]): string[] {

};
```

C# Solution:

```
/*
* Problem: Find Resultant Array After Removing Anagrams
* Difficulty: Easy
* 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<string> RemoveAnagrams(string[] words) {

}
}
```

C Solution:

```
/*
* Problem: Find Resultant Array After Removing Anagrams
* Difficulty: Easy
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** removeAnagrams(char** words, int wordsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Find Resultant Array After Removing Anagrams
// Difficulty: Easy
// 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 removeAnagrams(words []string) []string {

}
```

Kotlin Solution:

```
class Solution {
fun removeAnagrams(words: Array<String>): List<String> {

}
}
```

Swift Solution:

```
class Solution {
func removeAnagrams(_ words: [String]) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Find Resultant Array After Removing Anagrams
// Difficulty: Easy
// 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 remove_anagrams(words: Vec<String>) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String[]} words
# @return {String[]}
def remove_anagrams(words)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $words
* @return String[]
*/
function removeAnagrams($words) {

}
}
```

Dart Solution:

```
class Solution {
List<String> removeAnagrams(List<String> words) {

}
}
```

Scala Solution:

```
object Solution {
def removeAnagrams(words: Array[String]): List[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec remove_anagrams(words :: [String.t]) :: [String.t]
def remove_anagrams(words) do

end
end
```

Erlang Solution:

```
-spec remove_anagrams(Words :: [unicode:unicode_binary()]) ->
[unicode:unicode_binary()].
remove_anagrams(Words) ->
.
```

Racket Solution:

```
(define/contract (remove-anagrams words)
(-> (listof string?) (listof string?))
)
```