Problem 890: Find and Replace Pattern

Problem Information

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

Problem Description

Given a list of strings

words

and a string

pattern

, return

a list of

words[i]

that match

pattern

. You may return the answer in

any order

.

A word matches the pattern if there exists a permutation of letters

p

so that after replacing every letter

x

in the pattern with

p(x)

, we get the desired word.

Recall that a permutation of letters is a bijection from letters to letters: every letter maps to another letter, and no two letters map to the same letter.

Example 1:

Input:

words = ["abc","deq","mee","aqq","dkd","ccc"], pattern = "abb"

Output:

["mee","aqq"]

Explanation:

"mee" matches the pattern because there is a permutation {a -> m, b -> e, ...}. "ccc" does not match the pattern because {a -> c, b -> c, ...} is not a permutation, since a and b map to the same letter.

Example 2:

Input:

words = ["a","b","c"], pattern = "a"

Output:

["a","b","c"]

Constraints:

1 <= pattern.length <= 20

1 <= words.length <= 50

words[i].length == pattern.length

pattern

and

words[i]

are lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
vector<string> findAndReplacePattern(vector<string>& words, string pattern) {

}
};
```

Java:

```
class Solution {
public List<String> findAndReplacePattern(String[] words, String pattern) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function findAndReplacePattern(words: string[], pattern: string): string[] {

};
```

C#:

```
public class Solution {
public IList<string> FindAndReplacePattern(string[] words, string pattern) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
```

```
char** findAndReplacePattern(char** words, int wordsSize, char* pattern, int*
returnSize) {

}
```

Go:

```
func findAndReplacePattern(words []string, pattern string) []string {

}
```

Kotlin:

```
class Solution {
fun findAndReplacePattern(words: Array<String>, pattern: String):
List<String> {

}
}
```

Swift:

```
class Solution {
func findAndReplacePattern(_ words: [String], _ pattern: String) -> [String]
{

}
}
```

Rust:

```
impl Solution {
pub fn find_and_replace_pattern(words: Vec<String>, pattern: String) ->
Vec<String> {

}
}
```

Ruby:

```
# @param {String[]} words
# @param {String} pattern
```

```
# @return {String[]}
def find_and_replace_pattern(words, pattern)

end
```

PHP:

```
class Solution {

/**
* @param String[] $words
* @param String $pattern
* @return String[]
*/
function findAndReplacePattern($words, $pattern) {

}
}
```

Dart:

```
class Solution {
List<String> findAndReplacePattern(List<String> words, String pattern) {

}
}
```

Scala:

```
object Solution {
def findAndReplacePattern(words: Array[String], pattern: String):
List[String] = {

}
}
```

Elixir:

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

```
end
end
```

Erlang:

```
-spec find_and_replace_pattern(Words :: [unicode:unicode_binary()], Pattern
:: unicode:unicode_binary()) -> [unicode:unicode_binary()].
find_and_replace_pattern(Words, Pattern) ->
.
```

Racket:

```
(define/contract (find-and-replace-pattern words pattern)
(-> (listof string?) string? (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find and Replace Pattern
* 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<string> findAndReplacePattern(vector<string>& words, string pattern) {

}
};
```

Java Solution:

```
/**
* Problem: Find and Replace Pattern
* 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<String> findAndReplacePattern(String[] words, String pattern) {

}
}
```

Python3 Solution:

```
"""
Problem: Find and Replace Pattern
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 findAndReplacePattern(self, words: List[str], pattern: str) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find and Replace Pattern
* 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[]} words
* @param {string} pattern
* @return {string[]}
*/
var findAndReplacePattern = function(words, pattern) {

};
```

TypeScript Solution:

```
/**
* Problem: Find and Replace Pattern
* 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 findAndReplacePattern(words: string[], pattern: string): string[] {

};
```

C# Solution:

```
/*
* Problem: Find and Replace Pattern
* 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<string> FindAndReplacePattern(string[] words, string pattern) {

}
}
```

C Solution:

```
/*
* Problem: Find and Replace Pattern
* 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
*/

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

}
```

Go Solution:

```
// Problem: Find and Replace Pattern
// 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 findAndReplacePattern(words []string, pattern string) []string {

}
```

Kotlin Solution:

```
class Solution {
fun findAndReplacePattern(words: Array<String>, pattern: String):
List<String> {

}
}
```

Swift Solution:

```
class Solution {
func findAndReplacePattern(_ words: [String], _ pattern: String) -> [String]
{

}
}
```

Rust Solution:

```
// Problem: Find and Replace Pattern
// 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 find_and_replace_pattern(words: Vec<String>, pattern: String) ->
Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String[]} words
# @param {String} pattern
# @return {String[]}
def find_and_replace_pattern(words, pattern)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $words
* @param String $pattern
* @return String[]
*/
function findAndReplacePattern($words, $pattern) {

}
}
```

Dart Solution:

```
class Solution {
List<String> findAndReplacePattern(List<String> words, String pattern) {

}
}
```

Scala Solution:

```
object Solution {
def findAndReplacePattern(words: Array[String], pattern: String):
List[String] = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec find_and_replace_pattern(Words :: [unicode:unicode_binary()], Pattern
:: unicode:unicode_binary()) -> [unicode:unicode_binary()].
find_and_replace_pattern(Words, Pattern) ->
.
```

Racket Solution:

```
(define/contract (find-and-replace-pattern words pattern)
(-> (listof string?) string? (listof string?))
)
```