Problem 472: Concatenated Words

Problem Information

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

Problem Description

Given an array of strings

words

(

without duplicates

), return

all the

concatenated words

in the given list of

words

.

A

concatenated word

is defined as a string that is comprised entirely of at least two shorter words (not necessarily distinct) in the given array.

Example 1:

Input:

words = ["cat","cats","catsdogcats","dog","dogcatsdog","hippopotamuses","rat","ratcatdogcat"]

Output:

["catsdogcats","dogcatsdog","ratcatdogcat"]

Explanation:

"catsdogcats" can be concatenated by "cats", "dog" and "cats"; "dogcatsdog" can be concatenated by "dog", "cats" and "dog"; "ratcatdogcat" can be concatenated by "rat", "cat", "dog" and "cat".

Example 2:

Input:

words = ["cat","dog","catdog"]

Output:

["catdog"]

Constraints:

1 <= words.length <= 10

4

1 <= words[i].length <= 30

words[i]

consists of only lowercase English letters.

All the strings of

words

are

unique

.

1 <= sum(words[i].length) <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def findAllConcatenatedWordsInADict(self, words):
```

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (find-all-concatenated-words-in-a-dict words)
(-> (listof string?) (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Concatenated Words
* Difficulty: Hard
* Tags: array, string, dp, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Concatenated Words
* Difficulty: Hard
* Tags: array, string, dp, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Concatenated Words
Difficulty: Hard
Tags: array, string, dp, sort, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Concatenated Words
* Difficulty: Hard
* Tags: array, string, dp, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Concatenated Words
* Difficulty: Hard
* Tags: array, string, dp, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function findAllConcatenatedWordsInADict(words: string[]): string[] {

};
```

C# Solution:

```
/*
* Problem: Concatenated Words
* Difficulty: Hard
* Tags: array, string, dp, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

C Solution:

```c
/*
* Problem: Concatenated Words
* Difficulty: Hard
* Tags: array, string, dp, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
```

Go Solution:

```go
// Problem: Concatenated Words
// Difficulty: Hard
// Tags: array, string, dp, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func findAllConcatenatedWordsInADict(words []string) []string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Concatenated Words
// Difficulty: Hard
// Tags: array, string, dp, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $words
* @return String[]
*/
function findAllConcatenatedWordsInADict($words) {
}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (find-all-concatenated-words-in-a-dict words)
(-> (listof string?) (listof string?))
)
```