Problem 3435: Frequencies of Shortest Supersequences

Problem Information

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

Problem Description

You are given an array of strings

words

. Find all

shortest common supersequences (SCS)

of

words

that are not

permutations

of each other.

A

shortest common supersequence

is a string of

minimum

length that contains each string in

words

as a

subsequence

.

Return a 2D array of integers

freqs

that represent all the SCSs. Each

freqs[i]

is an array of size 26, representing the frequency of each letter in the lowercase English alphabet for a single SCS. You may return the frequency arrays in any order.

Example 1:

Input:

words = ["ab","ba"]

Output:

[[1,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[2,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]]

Explanation:

The two SCSs are

"aba"

and

"bab"

. The output is the letter frequencies for each one.

Example 2:

Input:

words = ["aa","ac"]

Output:

[[2,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]]

Explanation:

The two SCSs are

"aac"

and

"aca"

. Since they are permutations of each other, keep only

"aac"

.

Example 3:

Input:

words =

["aa","bb","cc"]

Output:

[[2,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]]

Explanation:

"aabbcc"

and all its permutations are SCSs.

Constraints:

1 <= words.length <= 256

words[i].length == 2

All strings in

words

will altogether be composed of no more than 16 unique lowercase letters.

All strings in

words

are unique.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function supersequences(words: string[]): number[][] {

};
```

C#:

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

```
}
}
```

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().
*/
int** supersequences(char** words, int wordsSize, int* returnSize, int**
returnColumnSizes) {

}
```

Go:

```
func supersequences(words []string) [][]int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn supersequences(words: Vec<String>) -> Vec<Vec<i32>> {
```

```
}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
List<List<int>> supersequences(List<String> words) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (supersequences words)
(-> (listof string?) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Frequencies of Shortest Supersequences
* Difficulty: Hard
* Tags: array, string, graph, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Frequencies of Shortest Supersequences
* Difficulty: Hard
* Tags: array, string, graph, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Frequencies of Shortest Supersequences
Difficulty: Hard
Tags: array, string, graph, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Frequencies of Shortest Supersequences
* Difficulty: Hard
* Tags: array, string, graph, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Frequencies of Shortest Supersequences
* Difficulty: Hard
* Tags: array, string, graph, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function supersequences(words: string[]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Frequencies of Shortest Supersequences
* Difficulty: Hard
* Tags: array, string, graph, sort
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

C Solution:

```
/*
* Problem: Frequencies of Shortest Supersequences
* Difficulty: Hard
* Tags: array, string, graph, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* 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().
*/
int** supersequences(char** words, int wordsSize, int* returnSize, int**
returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Frequencies of Shortest Supersequences
// Difficulty: Hard
// Tags: array, string, graph, sort
//
// Approach: Use two pointers or sliding window technique
```

```
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func supersequences(words []string) [][]int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Frequencies of Shortest Supersequences
// Difficulty: Hard
// Tags: array, string, graph, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn supersequences(words: Vec<String>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> supersequences(List<String> words) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
```

```
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (supersequences words)
(-> (listof string?) (listof (listof exact-integer?)))
)
```