Problem 2901: Longest Unequal Adjacent Groups Subsequence II

Problem Information

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

Problem Description

You are given a string array

words

, and an array

groups

, both arrays having length

n

.

The

hamming distance

between two strings of equal length is the number of positions at which the corresponding characters are

different

.

You need to select the

longest

subsequence

from an array of indices

[0, 1, ..., n - 1]

, such that for the subsequence denoted as

[i

0

, i

1

, ..., i

k-1

]

having length

k

, the following holds:

For

adjacent

indices in the subsequence, their corresponding groups are

unequal

, i.e.,

groups[i

j

] != groups[i

j+1

]

, for each

j

where

0 < j + 1 < k

.

words[i

j

]

and

words[i

j+1

]

are

equal

in length, and the

hamming distance

between them is

1

, where

0 < j + 1 < k

, for all indices in the subsequence.

Return

a string array containing the words corresponding to the indices

(in order)

in the selected subsequence

. If there are multiple answers, return

any of them

.

Note:

strings in

words

may be

unequal

in length.

Example 1:

Input:

words = ["bab","dab","cab"], groups = [1,2,2]

Output:

["bab","cab"]

Explanation:

A subsequence that can be selected is

[0,2]

.

groups[0] != groups[2]

words[0].length == words[2].length

, and the hamming distance between them is 1.

So, a valid answer is

[words[0],words[2]] = ["bab","cab"]

.

Another subsequence that can be selected is

[0,1]

.

groups[0] != groups[1]

words[0].length == words[1].length

, and the hamming distance between them is

1

.

So, another valid answer is

[words[0],words[1]] = ["bab","dab"]

.

It can be shown that the length of the longest subsequence of indices that satisfies the conditions is

2

.

Example 2:

Input:

words = ["a","b","c","d"], groups = [1,2,3,4]

Output:

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

Explanation:

We can select the subsequence

[0,1,2,3]

.

It satisfies both conditions.

Hence, the answer is

[words[0],words[1],words[2],words[3]] = ["a","b","c","d"]

.

It has the longest length among all subsequences of indices that satisfy the conditions.

Hence, it is the only answer.

Constraints:

1 <= n == words.length == groups.length <= 1000

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

1 <= groups[i] <= n

words

consists of

distinct

strings.

words[i]

consists of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
vector<string> getWordsInLongestSubsequence(vector<string>& words,
vector<int>& groups) {

}
};
```

Java:

```
class Solution {
public List<String> getWordsInLongestSubsequence(String[] words, int[]
groups) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function getWordsInLongestSubsequence(words: string[], groups: number[]):
string[] {

};
```

C#:

```
public class Solution {
public IList<string> GetWordsInLongestSubsequence(string[] words, int[]
groups) {

}
}
```

C:

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

}
```

Go:

```
func getWordsInLongestSubsequence(words []string, groups []int) []string {

}
```

Kotlin:

```
class Solution {
fun getWordsInLongestSubsequence(words: Array<String>, groups: IntArray):
List<String> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {String[]} words
# @param {Integer[]} groups
# @return {String[]}
def get_words_in_longest_subsequence(words, groups)

end
```

PHP:

```
class Solution {

/**
* @param String[] $words
* @param Integer[] $groups
* @return String[]
*/
function getWordsInLongestSubsequence($words, $groups) {

}
}
```

Dart:

```
class Solution {
List<String> getWordsInLongestSubsequence(List<String> words, List<int>
```

```
groups) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec get_words_in_longest_subsequence(Words :: [unicode:unicode_binary()],
Groups :: [integer()]) -> [unicode:unicode_binary()].
get_words_in_longest_subsequence(Words, Groups) ->
.
```

Racket:

```
(define/contract (get-words-in-longest-subsequence words groups)
(-> (listof string?) (listof exact-integer?) (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Unequal Adjacent Groups Subsequence II
* Difficulty: Medium
* Tags: array, string, dp
*
* 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> getWordsInLongestSubsequence(vector<string>& words,
vector<int>& groups) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Unequal Adjacent Groups Subsequence II
* Difficulty: Medium
* Tags: array, string, dp
*
* 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> getWordsInLongestSubsequence(String[] words, int[]
groups) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Unequal Adjacent Groups Subsequence II
Difficulty: Medium
```

```
Tags: array, string, dp

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 getWordsInLongestSubsequence(self, words: List[str], groups: List[int])
-> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Unequal Adjacent Groups Subsequence II
* Difficulty: Medium
* Tags: array, string, dp
*
* 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
* @param {number[]} groups
* @return {string[]}
*/
var getWordsInLongestSubsequence = function(words, groups) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Longest Unequal Adjacent Groups Subsequence II
* Difficulty: Medium
* Tags: array, string, dp
*
* 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 getWordsInLongestSubsequence(words: string[], groups: number[]):
string[] {

};
```

C# Solution:

```
/*
* Problem: Longest Unequal Adjacent Groups Subsequence II
* Difficulty: Medium
* Tags: array, string, dp
*
* 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> GetWordsInLongestSubsequence(string[] words, int[]
groups) {

}
}
```

C Solution:

```
/*
* Problem: Longest Unequal Adjacent Groups Subsequence II
* Difficulty: Medium
* Tags: array, string, dp
*
* 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** getWordsInLongestSubsequence(char** words, int wordsSize, int* groups,
int groupsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Longest Unequal Adjacent Groups Subsequence II
// Difficulty: Medium
// Tags: array, string, dp
//
// 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 getWordsInLongestSubsequence(words []string, groups []int) []string {

}
```

Kotlin Solution:

```
class Solution {
fun getWordsInLongestSubsequence(words: Array<String>, groups: IntArray):
List<String> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Longest Unequal Adjacent Groups Subsequence II
// Difficulty: Medium
// Tags: array, string, dp
//
// 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 get_words_in_longest_subsequence(words: Vec<String>, groups: Vec<i32>)
-> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String[]} words
# @param {Integer[]} groups
# @return {String[]}
def get_words_in_longest_subsequence(words, groups)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $words
* @param Integer[] $groups
* @return String[]
*/
```

```
function getWordsInLongestSubsequence($words, $groups) {

}
}
```

Dart Solution:

```
class Solution {
List<String> getWordsInLongestSubsequence(List<String> words, List<int>
groups) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec get_words_in_longest_subsequence(Words :: [unicode:unicode_binary()],
Groups :: [integer()]) -> [unicode:unicode_binary()].
get_words_in_longest_subsequence(Words, Groups) ->
.
```

Racket Solution:

```
(define/contract (get-words-in-longest-subsequence words groups)
(-> (listof string?) (listof exact-integer?) (listof string?))
)
```