Problem 2900: Longest Unequal Adjacent Groups Subsequence I

Problem Information

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

Problem Description

You are given a string array

words

and a

binary

array

groups

both of length

n

.

A

subsequence

of

words

is

alternating

if for any two

consecutive

strings in the sequence, their corresponding elements at the

same

indices in

groups

are

different

(that is, there

cannot

be consecutive 0 or 1).

Your task is to select the

longest alternating

subsequence from

words

.

Return

the selected subsequence. If there are multiple answers, return

any

of them.

Note:

The elements in

words

are distinct.

Example 1:

Input:

words = ["e","a","b"], groups = [0,0,1]

Output:

["e","b"]

Explanation:

A subsequence that can be selected is

["e","b"]

because

groups[0] != groups[2]

. Another subsequence that can be selected is

["a","b"]

because

groups[1] != groups[2]

. It can be demonstrated that the length of the longest subsequence of indices that satisfies the condition is

2

.

Example 2:

Input:

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

Output:

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

Explanation:

A subsequence that can be selected is

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

because

groups[0] != groups[1]

and

groups[1] != groups[2]

. Another subsequence that can be selected is

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

because

groups[0] != groups[1]

and

groups[1] != groups[3]

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

3

.

Constraints:

1 <= n == words.length == groups.length <= 100

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

groups[i]

is either

0

or

1.

words

consists of

distinct

strings.

words[i]

consists of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

```
List<String> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Longest Unequal Adjacent Groups Subsequence I
* Difficulty: Easy
* Tags: array, string, dp, greedy
*
* 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> getLongestSubsequence(vector<string>& words, vector<int>&
groups) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Unequal Adjacent Groups Subsequence I
* Difficulty: Easy
* Tags: array, string, dp, greedy
*
* 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> getLongestSubsequence(String[] words, int[] groups) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Unequal Adjacent Groups Subsequence I
Difficulty: Easy
Tags: array, string, dp, greedy

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Unequal Adjacent Groups Subsequence I
* Difficulty: Easy
* Tags: array, string, dp, greedy
*
* 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 getLongestSubsequence = function(words, groups) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Unequal Adjacent Groups Subsequence I
* Difficulty: Easy
* Tags: array, string, dp, greedy
*
* 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 getLongestSubsequence(words: string[], groups: number[]): string[] {

};
```

C# Solution:

```
/*
* Problem: Longest Unequal Adjacent Groups Subsequence I
* Difficulty: Easy
* Tags: array, string, dp, greedy
*
* 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> GetLongestSubsequence(string[] words, int[] groups) {

}
}
```

C Solution:

```
/*
* Problem: Longest Unequal Adjacent Groups Subsequence I
* Difficulty: Easy
* Tags: array, string, dp, greedy
*
* 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** getLongestSubsequence(char** words, int wordsSize, int* groups, int
groupsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Longest Unequal Adjacent Groups Subsequence I
// Difficulty: Easy
// Tags: array, string, dp, greedy
//
// 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 getLongestSubsequence(words []string, groups []int) []string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Longest Unequal Adjacent Groups Subsequence I
// Difficulty: Easy
// Tags: array, string, dp, greedy
//
// 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_longest_subsequence(words: Vec<String>, groups: Vec<i32>) ->
Vec<String> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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


}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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