Problem 527: Word Abbreviation

Problem Information

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

Problem Description

Given an array of

distinct

strings

words

, return

the minimal possible

abbreviations

for every word

.

The following are the rules for a string abbreviation:

The

initial

abbreviation for each word is: the first character, then the number of characters in between, followed by the last character.

If more than one word shares the

same

abbreviation, then perform the following operation:

Increase

the prefix (characters in the first part) of each of their abbreviations by

1

.

For example, say you start with the words

["abcdef","abndef"]

both initially abbreviated as

"a4f"

. Then, a sequence of operations would be

["a4f","a4f"]

->

["ab3f","ab3f"]

->

["abc2f","abn2f"]

.

This operation is repeated until every abbreviation is

unique

.

At the end, if an abbreviation did not make a word shorter, then keep it as the original word.

Example 1:

Input:

words = ["like","god","internal","me","internet","interval","intension","face","intrusion"]

Output:

["l2e","god","internal","me","i6t","interval","inte4n","f2e","intr4n"]

Example 2:

Input:

words = ["aa","aaa"]

Output:

["aa","aaa"]

Constraints:

1 <= words.length <= 400

2 <= words[i].length <= 400

words[i]

consists of lowercase English letters.

All the strings of

words

are

unique

.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public List<String> wordsAbbreviation(List<String> words) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public IList<string> WordsAbbreviation(IList<string> words) {

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
```

```
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (words-abbreviation words)
(-> (listof string?) (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Word Abbreviation
* Difficulty: Hard
* Tags: array, string, greedy, 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<string> wordsAbbreviation(vector<string>& words) {

}
};
```

Java Solution:

```
/**
* Problem: Word Abbreviation
* Difficulty: Hard
* Tags: array, string, greedy, 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<String> wordsAbbreviation(List<String> words) {

}
}
```

Python3 Solution:

```
"""
Problem: Word Abbreviation
Difficulty: Hard
Tags: array, string, greedy, 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 wordsAbbreviation(self, words: List[str]) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Word Abbreviation
* Difficulty: Hard
* Tags: array, string, greedy, 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 {string[]}
*/
var wordsAbbreviation = function(words) {

};
```

TypeScript Solution:

```
/**
* Problem: Word Abbreviation
* Difficulty: Hard
* Tags: array, string, greedy, 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 wordsAbbreviation(words: string[]): string[] {

};
```

C# Solution:

```
/*
* Problem: Word Abbreviation
* Difficulty: Hard
* Tags: array, string, greedy, 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<string> WordsAbbreviation(IList<string> words) {

}
}
```

C Solution:

```
/*
* Problem: Word Abbreviation
* Difficulty: Hard
* Tags: array, string, greedy, 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
```

```
*/

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

}
```

Go Solution:

```
// Problem: Word Abbreviation
// Difficulty: Hard
// Tags: array, string, greedy, 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 wordsAbbreviation(words []string) []string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Word Abbreviation
// Difficulty: Hard
// Tags: array, string, greedy, 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 words_abbreviation(words: Vec<String>) -> Vec<String> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (words-abbreviation words)
(-> (listof string?) (listof string?))
)
```