Problem 3093: Longest Common Suffix Queries

Problem Information

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

Problem Description

You are given two arrays of strings

wordsContainer

and

wordsQuery

.

For each

wordsQuery[i]

, you need to find a string from

wordsContainer

that has the

longest common suffix

with

wordsQuery[i]

. If there are two or more strings in

wordsContainer

that share the longest common suffix, find the string that is the

smallest

in length. If there are two or more such strings that have the

same

smallest length, find the one that occurred

earlier

in

wordsContainer

.

Return

an array of integers

ans

, where

ans[i]

is the index of the string in

wordsContainer

that has the

longest common suffix

with

wordsQuery[i]

.

Example 1:

Input:

wordsContainer = ["abcd","bcd","xbcd"], wordsQuery = ["cd","bcd","xyz"]

Output:

[1,1,1]

Explanation:

Let's look at each

wordsQuery[i]

separately:

For

wordsQuery[0] = "cd"

, strings from

wordsContainer

that share the longest common suffix

"cd"

are at indices 0, 1, and 2. Among these, the answer is the string at index 1 because it has the shortest length of 3.

For

wordsQuery[1] = "bcd"

, strings from

wordsContainer

that share the longest common suffix

"bcd"

are at indices 0, 1, and 2. Among these, the answer is the string at index 1 because it has the shortest length of 3.

For

wordsQuery[2] = "xyz"

, there is no string from

wordsContainer

that shares a common suffix. Hence the longest common suffix is

""

, that is shared with strings at index 0, 1, and 2. Among these, the answer is the string at index 1 because it has the shortest length of 3.

Example 2:

Input:

wordsContainer = ["abcdefgh","poiuygh","ghghgh"], wordsQuery = ["gh","acbfgh","acbfegh"]

Output:

[2,0,2]

Explanation:

Let's look at each

wordsQuery[i]

separately:

For

wordsQuery[0] = "gh"

, strings from

wordsContainer

that share the longest common suffix

"gh"

are at indices 0, 1, and 2. Among these, the answer is the string at index 2 because it has the shortest length of 6.

For

wordsQuery[1] = "acbfgh"

, only the string at index 0 shares the longest common suffix

"fgh"

. Hence it is the answer, even though the string at index 2 is shorter.

For

wordsQuery[2] = "acbfegh"

, strings from

wordsContainer

that share the longest common suffix

"gh"

are at indices 0, 1, and 2. Among these, the answer is the string at index 2 because it has the shortest length of 6.

Constraints:

1 <= wordsContainer.length, wordsQuery.length <= 10

4

1 <= wordsContainer[i].length <= 5 * 10

3

1 <= wordsQuery[i].length <= 5 * 10

3

wordsContainer[i]

consists only of lowercase English letters.

wordsQuery[i]

consists only of lowercase English letters.

Sum of

wordsContainer[i].length

is at most

5 * 10

5

.

Sum of

wordsQuery[i].length

is at most

5 * 10

5

.

Code Snippets

C++:

```
class Solution {
public:
vector<int> stringIndices(vector<string>& wordsContainer, vector<string>&
wordsQuery) {

}
};
```

Java:

```
class Solution {
public int[] stringIndices(String[] wordsContainer, String[] wordsQuery) {

}
}
```

Python3:

```
class Solution:
def stringIndices(self, wordsContainer: List[str], wordsQuery: List[str]) ->
List[int]:
```

Python:

```
class Solution(object):
def stringIndices(self, wordsContainer, wordsQuery):
"""
:type wordsContainer: List[str]
:type wordsQuery: List[str]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {string[]} wordsContainer
* @param {string[]} wordsQuery
* @return {number[]}
*/
var stringIndices = function(wordsContainer, wordsQuery) {

};
```

TypeScript:

```
function stringIndices(wordsContainer: string[], wordsQuery: string[]):
number[] {

};
```

C#:

```
public class Solution {
public int[] StringIndices(string[] wordsContainer, string[] wordsQuery) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* stringIndices(char** wordsContainer, int wordsContainerSize, char**
wordsQuery, int wordsQuerySize, int* returnSize) {

}
```

Go:

```
func stringIndices(wordsContainer []string, wordsQuery []string) []int {

}
```

Kotlin:

```
class Solution {
fun stringIndices(wordsContainer: Array<String>, wordsQuery: Array<String>):
IntArray {

}
}
```

Swift:

```
class Solution {
func stringIndices(_ wordsContainer: [String], _ wordsQuery: [String]) ->
[Int] {

}
}
```

Rust:

```
impl Solution {
pub fn string_indices(words_container: Vec<String>, words_query: Vec<String>)
-> Vec<i32> {

}
}
```

Ruby:

```
# @param {String[]} words_container
# @param {String[]} words_query
# @return {Integer[]}
def string_indices(words_container, words_query)

end
```

PHP:

```
class Solution {

/**
* @param String[] $wordsContainer
* @param String[] $wordsQuery
* @return Integer[]
*/
function stringIndices($wordsContainer, $wordsQuery) {

}
}
```

Dart:

```
class Solution {
List<int> stringIndices(List<String> wordsContainer, List<String> wordsQuery)
{

}
}
```

Scala:

```
object Solution {
def stringIndices(wordsContainer: Array[String], wordsQuery: Array[String]):
Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec string_indices(words_container :: [String.t], words_query ::
```

```
[String.t]) :: [integer]
def string_indices(words_container, words_query) do

end
end
```

Erlang:

```
-spec string_indices(WordsContainer :: [unicode:unicode_binary()], WordsQuery
:: [unicode:unicode_binary()]) -> [integer()].
string_indices(WordsContainer, WordsQuery) ->
.
```

Racket:

```
(define/contract (string-indices wordsContainer wordsQuery)
(-> (listof string?) (listof string?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Common Suffix Queries
* Difficulty: Hard
* Tags: array, string
*
* 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<int> stringIndices(vector<string>& wordsContainer, vector<string>&
wordsQuery) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Common Suffix Queries
* Difficulty: Hard
* Tags: array, string
*
* 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 int[] stringIndices(String[] wordsContainer, String[] wordsQuery) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Common Suffix Queries
Difficulty: Hard
Tags: array, string

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 stringIndices(self, wordsContainer: List[str], wordsQuery: List[str]) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def stringIndices(self, wordsContainer, wordsQuery):
"""
:type wordsContainer: List[str]
:type wordsQuery: List[str]
```

```
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Longest Common Suffix Queries
* Difficulty: Hard
* Tags: array, string
*
* 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[]} wordsContainer
* @param {string[]} wordsQuery
* @return {number[]}
*/
var stringIndices = function(wordsContainer, wordsQuery) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Common Suffix Queries
* Difficulty: Hard
* Tags: array, string
*
* 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 stringIndices(wordsContainer: string[], wordsQuery: string[]):
number[] {

};
```

C# Solution:

```
/*
* Problem: Longest Common Suffix Queries
* Difficulty: Hard
* Tags: array, string
*
* 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 int[] StringIndices(string[] wordsContainer, string[] wordsQuery) {

}
}
```

C Solution:

```
/*
* Problem: Longest Common Suffix Queries
* Difficulty: Hard
* Tags: array, string
*
* 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().
*/
int* stringIndices(char** wordsContainer, int wordsContainerSize, char**
wordsQuery, int wordsQuerySize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Longest Common Suffix Queries
// Difficulty: Hard
// Tags: array, string
```

```
//
// 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 stringIndices(wordsContainer []string, wordsQuery []string) []int {

}
```

Kotlin Solution:

```
class Solution {
fun stringIndices(wordsContainer: Array<String>, wordsQuery: Array<String>):
IntArray {

}
}
```

Swift Solution:

```
class Solution {
func stringIndices(_ wordsContainer: [String], _ wordsQuery: [String]) ->
[Int] {

}
}
```

Rust Solution:

```
// Problem: Longest Common Suffix Queries
// Difficulty: Hard
// Tags: array, string
//
// 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 string_indices(words_container: Vec<String>, words_query: Vec<String>)
-> Vec<i32> {
```

```
}
}
```

Ruby Solution:

```
# @param {String[]} words_container
# @param {String[]} words_query
# @return {Integer[]}
def string_indices(words_container, words_query)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $wordsContainer
* @param String[] $wordsQuery
* @return Integer[]
*/
function stringIndices($wordsContainer, $wordsQuery) {

}
}
```

Dart Solution:

```
class Solution {
List<int> stringIndices(List<String> wordsContainer, List<String> wordsQuery)
{

}
}
```

Scala Solution:

```
object Solution {
def stringIndices(wordsContainer: Array[String], wordsQuery: Array[String]):
Array[Int] = {
```

```
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec string_indices(words_container :: [String.t], words_query ::
[String.t]) :: [integer]
def string_indices(words_container, words_query) do

end
end
```

Erlang Solution:

```
-spec string_indices(WordsContainer :: [unicode:unicode_binary()], WordsQuery
:: [unicode:unicode_binary()]) -> [integer()].
string_indices(WordsContainer, WordsQuery) ->
.
```

Racket Solution:

```
(define/contract (string-indices wordsContainer wordsQuery)
(-> (listof string?) (listof string?) (listof exact-integer?))
)
```