Problem 3076: Shortest Uncommon Substring in an Array

Problem Information

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

Problem Description

You are given an array

arr

of size

n

consisting of

non-empty

strings.

Find a string array

answer

of size

n

such that:

answer[i]

is the

shortest

substring

of

arr[i]

that does

not

occur as a substring in any other string in

arr

. If multiple such substrings exist,

answer[i]

should be the

lexicographically smallest

. And if no such substring exists,

answer[i]

should be an empty string.

Return

the array

answer

.

Example 1:

Input:

arr = ["cab","ad","bad","c"]

Output:

["ab","","ba",""]

Explanation:

We have the following: - For the string "cab", the shortest substring that does not occur in any other string is either "ca" or "ab", we choose the lexicographically smaller substring, which is "ab". - For the string "ad", there is no substring that does not occur in any other string. - For the string "bad", the shortest substring that does not occur in any other string is "ba". - For the string "c", there is no substring that does not occur in any other string.

Example 2:

Input:

arr = ["abc","bcd","abcd"]

Output:

["","","abcd"]

Explanation:

We have the following: - For the string "abc", there is no substring that does not occur in any other string. - For the string "bcd", there is no substring that does not occur in any other string. - For the string "abcd", the shortest substring that does not occur in any other string is "abcd".

Constraints:

n == arr.length

2 <= n <= 100

1 <= arr[i].length <= 20

arr[i]

consists only of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
vector<string> shortestSubstrings(vector<string>& arr) {

}
};
```

Java:

```
class Solution {
public String[] shortestSubstrings(String[] arr) {

}
}
```

Python3:

```
class Solution:
def shortestSubstrings(self, arr: List[str]) -> List[str]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function shortestSubstrings(arr: string[]): string[] {

};
```

C#:

```
public class Solution {
public string[] ShortestSubstrings(string[] arr) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** shortestSubstrings(char** arr, int arrSize, int* returnSize) {

}
```

Go:

```
func shortestSubstrings(arr []string) []string {

}
```

Kotlin:

```
class Solution {
fun shortestSubstrings(arr: Array<String>): Array<String> {
```

```
}
}
```

Swift:

```
class Solution {
func shortestSubstrings(_ arr: [String]) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn shortest_substrings(arr: Vec<String>) -> Vec<String> {

}
}
```

Ruby:

```
# @param {String[]} arr
# @return {String[]}
def shortest_substrings(arr)

end
```

PHP:

```
class Solution {

/**
* @param String[] $arr
* @return String[]
*/
function shortestSubstrings($arr) {

}
}
```

Dart:

```
class Solution {
List<String> shortestSubstrings(List<String> arr) {

}
}
```

Scala:

```
object Solution {
def shortestSubstrings(arr: Array[String]): Array[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec shortest_substrings(arr :: [String.t]) :: [String.t]
def shortest_substrings(arr) do

end
end
```

Erlang:

```
-spec shortest_substrings(Arr :: [unicode:unicode_binary()]) ->
[unicode:unicode_binary()].
shortest_substrings(Arr) ->
.
```

Racket:

```
(define/contract (shortest-substrings arr)
(-> (listof string?) (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shortest Uncommon Substring in an Array
* Difficulty: Medium
* Tags: array, string, tree, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
vector<string> shortestSubstrings(vector<string>& arr) {

}
};
```

Java Solution:

```
/**
* Problem: Shortest Uncommon Substring in an Array
* Difficulty: Medium
* Tags: array, string, tree, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public String[] shortestSubstrings(String[] arr) {

}
}
```

Python3 Solution:

```
"""
Problem: Shortest Uncommon Substring in an Array
Difficulty: Medium
Tags: array, string, tree, graph, hash
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Shortest Uncommon Substring in an Array
* Difficulty: Medium
* Tags: array, string, tree, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {string[]} arr
* @return {string[]}
*/
var shortestSubstrings = function(arr) {

};
```

TypeScript Solution:

```
/**
* Problem: Shortest Uncommon Substring in an Array
* Difficulty: Medium
* Tags: array, string, tree, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function shortestSubstrings(arr: string[]): string[] {

};
```

C# Solution:

```
/*
* Problem: Shortest Uncommon Substring in an Array
* Difficulty: Medium
* Tags: array, string, tree, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public string[] ShortestSubstrings(string[] arr) {

}
}
```

C Solution:

```
/*
* Problem: Shortest Uncommon Substring in an Array
* Difficulty: Medium
* Tags: array, string, tree, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** shortestSubstrings(char** arr, int arrSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Shortest Uncommon Substring in an Array
// Difficulty: Medium
// Tags: array, string, tree, graph, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func shortestSubstrings(arr []string) []string {

}
```

Kotlin Solution:

```
class Solution {
fun shortestSubstrings(arr: Array<String>): Array<String> {

}
}
```

Swift Solution:

```
class Solution {
func shortestSubstrings(_ arr: [String]) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Shortest Uncommon Substring in an Array
// Difficulty: Medium
// Tags: array, string, tree, graph, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn shortest_substrings(arr: Vec<String>) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String[]} arr
# @return {String[]}
def shortest_substrings(arr)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $arr
* @return String[]
*/
function shortestSubstrings($arr) {

}
}
```

Dart Solution:

```
class Solution {
List<String> shortestSubstrings(List<String> arr) {

}
}
```

Scala Solution:

```
object Solution {
def shortestSubstrings(arr: Array[String]): Array[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec shortest_substrings(arr :: [String.t]) :: [String.t]
def shortest_substrings(arr) do

end
end
```

Erlang Solution:

```
-spec shortest_substrings(Arr :: [unicode:unicode_binary()]) ->
[unicode:unicode_binary()].
shortest_substrings(Arr) ->
.
```

Racket Solution:

```
(define/contract (shortest-substrings arr)
(-> (listof string?) (listof string?))
)
```