Problem 1268: Search Suggestions System

Problem Information

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

Problem Description

You are given an array of strings

products

and a string

searchWord

.

Design a system that suggests at most three product names from

products

after each character of

searchWord

is typed. Suggested products should have common prefix with

searchWord

. If there are more than three products with a common prefix return the three lexicographically minimums products.

Return

a list of lists of the suggested products after each character of

searchWord

is typed

.

Example 1:

Input:

products = ["mobile","mouse","moneypot","monitor","mousepad"], searchWord = "mouse"

Output:

[["mobile","moneypot","monitor"],["mobile","moneypot","monitor"],["mouse","mousepad"],["mouse","mousepad"],["mouse","mousepad"]]

Explanation:

products sorted lexicographically = ["mobile","moneypot","monitor","mouse","mousepad"].
After typing m and mo all products match and we show user ["mobile","moneypot","monitor"].
After typing mou, mous and mouse the system suggests ["mouse","mousepad"].

Example 2:

Input:

products = ["havana"], searchWord = "havana"

Output:

[["havana"],["havana"],["havana"],["havana"],["havana"],["havana"]]

Explanation:

The only word "havana" will be always suggested while typing the search word.

Constraints:

1 <= products.length <= 1000

1 <= products[i].length <= 3000

1 <= sum(products[i].length) <= 2 * 10

4

All the strings of

products

are

unique

.

products[i]

consists of lowercase English letters.

1 <= searchWord.length <= 1000

searchWord

consists of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
vector<vector<string>> suggestedProducts(vector<string>& products, string
searchWord) {
}
};
```

Java:

```
class Solution {
public List<List<String>> suggestedProducts(String[] products, String
searchWord) {

}
}
```

Python3:

```
class Solution:
def suggestedProducts(self, products: List[str], searchWord: str) ->
List[List[str]]:
```

Python:

```
class Solution(object):
def suggestedProducts(self, products, searchWord):
"""
:type products: List[str]
:type searchWord: str
:rtype: List[List[str]]
"""
```

JavaScript:

```
/**
* @param {string[]} products
* @param {string} searchWord
* @return {string[][]}
*/
var suggestedProducts = function(products, searchWord) {

};
```

TypeScript:

```
function suggestedProducts(products: string[], searchWord: string):
string[][] {

};
```

C#:

```
public class Solution {
public IList<IList<string>> SuggestedProducts(string[] products, string
searchWord) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
char*** suggestedProducts(char** products, int productsSize, char*
searchWord, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func suggestedProducts(products []string, searchWord string) [][]string {

}
```

Kotlin:

```
class Solution {
fun suggestedProducts(products: Array<String>, searchWord: String):
List<List<String>> {

}
}
```

Swift:

```
class Solution {
func suggestedProducts(_ products: [String], _ searchWord: String) ->
[[String]] {

}
}
```

Rust:

```
impl Solution {
pub fn suggested_products(products: Vec<String>, search_word: String) ->
Vec<Vec<String>> {

}
}
```

Ruby:

```
# @param {String[]} products
# @param {String} search_word
# @return {String[][]}
def suggested_products(products, search_word)

end
```

PHP:

```
class Solution {

/**
* @param String[] $products
* @param String $searchWord
* @return String[][]
*/
function suggestedProducts($products, $searchWord) {

}
}
```

Dart:

```
class Solution {
List<List<String>> suggestedProducts(List<String> products, String
searchWord) {

}
}
```

Scala:

```
object Solution {
def suggestedProducts(products: Array[String], searchWord: String):
List[List[String]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec suggested_products(products :: [String.t], search_word :: String.t) ::
[[String.t]]
def suggested_products(products, search_word) do

end
end
```

Erlang:

```
-spec suggested_products(Products :: [unicode:unicode_binary()], SearchWord
:: unicode:unicode_binary()) -> [[unicode:unicode_binary()]].
suggested_products(Products, SearchWord) ->
.
```

Racket:

```
(define/contract (suggested-products products searchWord)
(-> (listof string?) string? (listof (listof string?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Search Suggestions System
* Difficulty: Medium
* Tags: array, string, graph, sort, search, queue, heap
*
* 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<vector<string>> suggestedProducts(vector<string>& products, string
searchWord) {

}
};
```

Java Solution:

```
/**
* Problem: Search Suggestions System
* Difficulty: Medium
* Tags: array, string, graph, sort, search, queue, heap
*
* 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<List<String>> suggestedProducts(String[] products, String
searchWord) {

}
}
```

Python3 Solution:

```
"""
Problem: Search Suggestions System
Difficulty: Medium
```

```
Tags: array, string, graph, sort, search, queue, heap

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 suggestedProducts(self, products: List[str], searchWord: str) ->
List[List[str]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def suggestedProducts(self, products, searchWord):
"""
:type products: List[str]
:type searchWord: str
:rtype: List[List[str]]
"""
```

JavaScript Solution:

```
/**
* Problem: Search Suggestions System
* Difficulty: Medium
* Tags: array, string, graph, sort, search, queue, heap
*
* 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[]} products
* @param {string} searchWord
* @return {string[][]}
*/
var suggestedProducts = function(products, searchWord) {
};
```

TypeScript Solution:

```
/**
* Problem: Search Suggestions System
* Difficulty: Medium
* Tags: array, string, graph, sort, search, queue, heap
*
* 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 suggestedProducts(products: string[], searchWord: string):
string[][] {

};
```

C# Solution:

```
/*
* Problem: Search Suggestions System
* Difficulty: Medium
* Tags: array, string, graph, sort, search, queue, heap
*
* 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<IList<string>> SuggestedProducts(string[] products, string
searchWord) {

}
}
```

C Solution:

```
/*
* Problem: Search Suggestions System
* Difficulty: Medium
* Tags: array, string, graph, sort, search, queue, heap
*
* 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
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume caller calls free().
*/
char*** suggestedProducts(char** products, int productsSize, char* searchWord, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Search Suggestions System
// Difficulty: Medium
// Tags: array, string, graph, sort, search, queue, heap
//
// 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 suggestedProducts(products []string, searchWord string) [][]string {

}
```

Kotlin Solution:

```
class Solution {
fun suggestedProducts(products: Array<String>, searchWord: String): List<List<String>> {

}
}
```

Swift Solution:

```
class Solution {
func suggestedProducts(_ products: [String], _ searchWord: String) ->
[[String]] {

}
}
```

Rust Solution:

```
// Problem: Search Suggestions System
// Difficulty: Medium
// Tags: array, string, graph, sort, search, queue, heap
//
// 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 suggested_products(products: Vec<String>, search_word: String) ->
Vec<Vec<String>> {

}
}
```

Ruby Solution:

```
# @param {String[]} products
# @param {String} search_word
# @return {String[][]}
def suggested_products(products, search_word)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param String[] $products
* @param String $searchWord
* @return String[][]
*/
function suggestedProducts($products, $searchWord) {

}
}
```

Dart Solution:

```
class Solution {
List<List<String>> suggestedProducts(List<String> products, String searchWord) {

}
}
```

Scala Solution:

```
object Solution {
def suggestedProducts(products: Array[String], searchWord: String): List[List[String]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec suggested_products(products :: [String.t], search_word :: String.t) :: [[String.t]]
def suggested_products(products, search_word) do

end
end
```

Erlang Solution:

```
-spec suggested_products(Products :: [unicode:unicode_binary()], SearchWord
:: unicode:unicode_binary()) -> [[unicode:unicode_binary()]].
suggested_products(Products, SearchWord) ->
.
```

Racket Solution:

```
(define/contract (suggested-products products searchWord)
(-> (listof string?) string? (listof (listof string?)))
)
```