Problem 1239: Maximum Length of a Concatenated String with Unique Characters

Problem Information

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

Problem Description

You are given an array of strings

arr

. A string

s

is formed by the

concatenation

of a

subsequence

of

arr

that has

unique characters

.

Return

the

maximum

possible length

of

s

.

A

subsequence

is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.

Example 1:

Input:

arr = ["un","iq","ue"]

Output:

4

Explanation:

All the valid concatenations are: - "" - "un" - "iq" - "ue" - "uniq" ("un" + "iq") - "ique" ("iq" + "ue") Maximum length is 4.

Example 2:

Input:

arr = ["cha","r","act","ers"]

Output:

6

Explanation:

Possible longest valid concatenations are "chaers" ("cha" + "ers") and "acters" ("act" + "ers").

Example 3:

Input:

arr = ["abcdefghijklmnopqrstuvwxyz"]

Output:

26

Explanation:

The only string in arr has all 26 characters.

Constraints:

1 <= arr.length <= 16

1 <= arr[i].length <= 26

arr[i]

contains only lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int maxLength(List<String> arr) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxLength(arr: string[]): number {
};
```

C#:

```
public class Solution {
public int MaxLength(IList<string> arr) {

}
}
```

C:

```
int maxLength(char** arr, int arrSize) {

}
```

Go:

```
func maxLength(arr []string) int {

}
```

Kotlin:

```
class Solution {
fun maxLength(arr: List<String>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn max_length(arr: Vec<String>) -> i32 {
}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int maxLength(List<String> arr) {

}
}
```

Scala:

```
object Solution {
def maxLength(arr: List[String]): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-length arr)
(-> (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Length of a Concatenated String with Unique Characters
* Difficulty: Medium
* 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 maxLength(vector<string>& arr) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Length of a Concatenated String with Unique Characters
* Difficulty: Medium
* 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 maxLength(List<String> arr) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Length of a Concatenated String with Unique Characters
Difficulty: Medium
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 maxLength(self, arr: List[str]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```javascript
/**
* Problem: Maximum Length of a Concatenated String with Unique Characters
* Difficulty: Medium
* 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[]} arr
* @return {number}
*/
var maxLength = function(arr) {

};
```

TypeScript Solution:

```typescript
/**
* Problem: Maximum Length of a Concatenated String with Unique Characters
* Difficulty: Medium
* 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 maxLength(arr: string[]): number {

};
```

C# Solution:

```csharp
/*
* Problem: Maximum Length of a Concatenated String with Unique Characters
* Difficulty: Medium
* 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 MaxLength(IList<string> arr) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Length of a Concatenated String with Unique Characters
* Difficulty: Medium
* 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
*/

int maxLength(char** arr, int arrSize) {

}
```

Go Solution:

```
// Problem: Maximum Length of a Concatenated String with Unique Characters
// Difficulty: Medium
// 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 maxLength(arr []string) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxLength(arr: List<String>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Length of a Concatenated String with Unique Characters
// Difficulty: Medium
// 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 max_length(arr: Vec<String>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int maxLength(List<String> arr) {

}
}
```

Scala Solution:

```
object Solution {
def maxLength(arr: List[String]): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-length arr)
(-> (listof string?) exact-integer?)
)
```