Problem 944: Delete Columns to Make Sorted

Problem Information

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

Problem Description

You are given an array of

n

strings

strs

, all of the same length.

The strings can be arranged such that there is one on each line, making a grid.

For example,

strs = ["abc", "bce", "cae"]

can be arranged as follows:

abc bce cae

You want to

delete

the columns that are

not sorted lexicographically

. In the above example (

0-indexed

), columns 0 (

'a'

,

'b'

,

'c'

) and 2 (

'c'

,

'e'

,

'e'

) are sorted, while column 1 (

'b'

,

'c'

,

'a'

) is not, so you would delete column 1.

Return

the number of columns that you will delete

.

Example 1:

Input:

strs = ["cba","daf","ghi"]

Output:

1

Explanation:

The grid looks as follows: cba daf ghi Columns 0 and 2 are sorted, but column 1 is not, so you only need to delete 1 column.

Example 2:

Input:

strs = ["a","b"]

Output:

0

Explanation:

The grid looks as follows: a b Column 0 is the only column and is sorted, so you will not delete any columns.

Example 3:

Input:

strs = ["zyx","wvu","tsr"]

Output:

3

Explanation:

The grid looks as follows: zyx wvu tsr All 3 columns are not sorted, so you will delete all 3.

Constraints:

n == strs.length

1 <= n <= 100

1 <= strs[i].length <= 1000

strs[i]

consists of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int minDeletionSize(vector<string>& strs) {

}
};
```

Java:

```
class Solution {
public int minDeletionSize(String[] strs) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minDeletionSize(strs: string[]): number {

};
```

C#:

```
public class Solution {
public int MinDeletionSize(string[] strs) {

}
}
```

C:

```
int minDeletionSize(char** strs, int strsSize) {

}
```

Go:

```
func minDeletionSize(strs []string) int {

}
```

Kotlin:

```
class Solution {
fun minDeletionSize(strs: Array<String>): Int {

}
}
```

Swift:

```
class Solution {
func minDeletionSize(_ strs: [String]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_deletion_size(strs: Vec<String>) -> i32 {

}
}
```

Ruby:

```
# @param {String[]} strs
# @return {Integer}
def min_deletion_size(strs)

end
```

PHP:

```
class Solution {

/**
* @param String[] $strs
* @return Integer
*/
function minDeletionSize($strs) {

}
}
```

Dart:

```
class Solution {
int minDeletionSize(List<String> strs) {

}
}
```

Scala:

```
object Solution {
def minDeletionSize(strs: Array[String]): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec min_deletion_size(Strs :: [unicode:unicode_binary()]) -> integer().
min_deletion_size(Strs) ->
.
```

Racket:

```
(define/contract (min-deletion-size strs)
(-> (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Delete Columns to Make Sorted
* Difficulty: Easy
* Tags: array, string, graph, 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:
int minDeletionSize(vector<string>& strs) {

}
};
```

Java Solution:

```
/**
* Problem: Delete Columns to Make Sorted
* Difficulty: Easy
* Tags: array, string, graph, 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 int minDeletionSize(String[] strs) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Delete Columns to Make Sorted
Difficulty: Easy
Tags: array, string, graph, 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 minDeletionSize(self, strs: List[str]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Delete Columns to Make Sorted
* Difficulty: Easy
* Tags: array, string, graph, 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[]} strs
* @return {number}
*/
var minDeletionSize = function(strs) {

};
```

TypeScript Solution:

```
/**
* Problem: Delete Columns to Make Sorted
* Difficulty: Easy
* Tags: array, string, graph, 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 minDeletionSize(strs: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Delete Columns to Make Sorted
* Difficulty: Easy
* Tags: array, string, graph, 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 int MinDeletionSize(string[] strs) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Delete Columns to Make Sorted
* Difficulty: Easy
* Tags: array, string, graph, 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
*/

int minDeletionSize(char** strs, int strsSize) {

}
```

Go Solution:

```
// Problem: Delete Columns to Make Sorted
// Difficulty: Easy
// Tags: array, string, graph, 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 minDeletionSize(strs []string) int {

}
```

Kotlin Solution:

```
class Solution {
fun minDeletionSize(strs: Array<String>): Int {

}
}
```

Swift Solution:

```
class Solution {
func minDeletionSize(_ strs: [String]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Delete Columns to Make Sorted
// Difficulty: Easy
// Tags: array, string, graph, 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 min_deletion_size(strs: Vec<String>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[]} strs
# @return {Integer}
def min_deletion_size(strs)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $strs
* @return Integer
*/
function minDeletionSize($strs) {

}
}
```

Dart Solution:

```
class Solution {
int minDeletionSize(List<String> strs) {

}
}
```

Scala Solution:

```
object Solution {
def minDeletionSize(strs: Array[String]): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec min_deletion_size(Strs :: [unicode:unicode_binary()]) -> integer().
min_deletion_size(Strs) ->
.
```

Racket Solution:

```
(define/contract (min-deletion-size strs)
(-> (listof string?) exact-integer?)
)
```