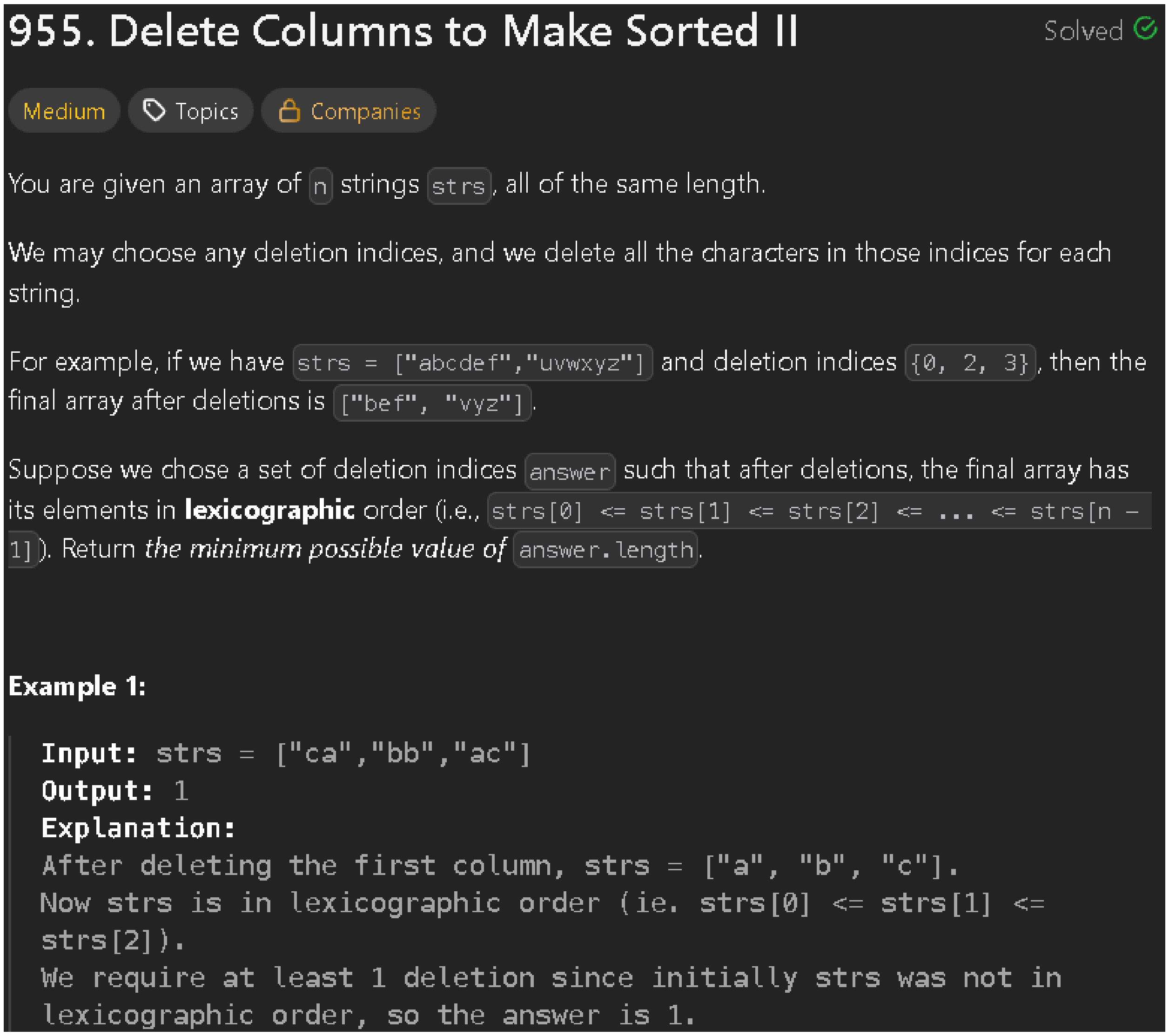


955. Delete Columns to Make Sorted II

Solved ✓

Medium · Topics · Companies

You are given an array of `n` strings `strs`, all of the same length.

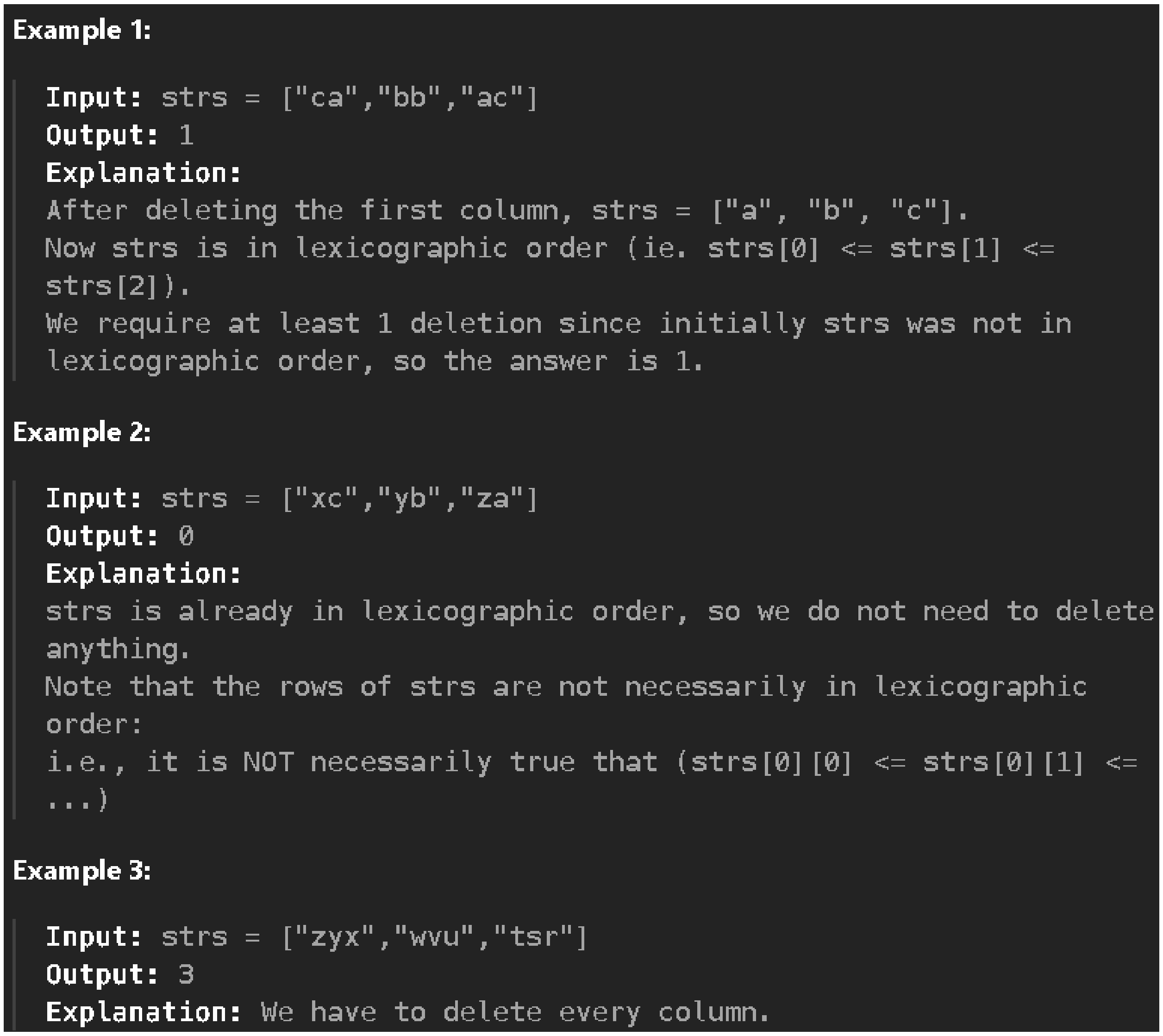
We may choose any deletion indices, and we delete all the characters in those indices for each string.

For example, if we have `strs = ["abcdef","uvwxyz"]` and deletion indices `{0, 2, 3}`, then the final array after deletions is `["bef", "vyz"]`.

Suppose we chose a set of deletion indices `answer` such that after deletions, the final array has its elements in **lexicographic** order (i.e., `strs[0] <= strs[1] <= strs[2] <= ... <= strs[n - 1]`). Return *the minimum possible value of* `answer.length`.

Example 1:

```
Input: strs = ["ca","bb","ac"]
Output: 1
Explanation:
After deleting the first column, strs = ["a", "b", "c"].
Now strs is in lexicographic order (ie. strs[0] <= strs[1] <=
strs[2]).
We require at least 1 deletion since initially strs was not in
lexicographic order, so the answer is 1.
```


Example 1:

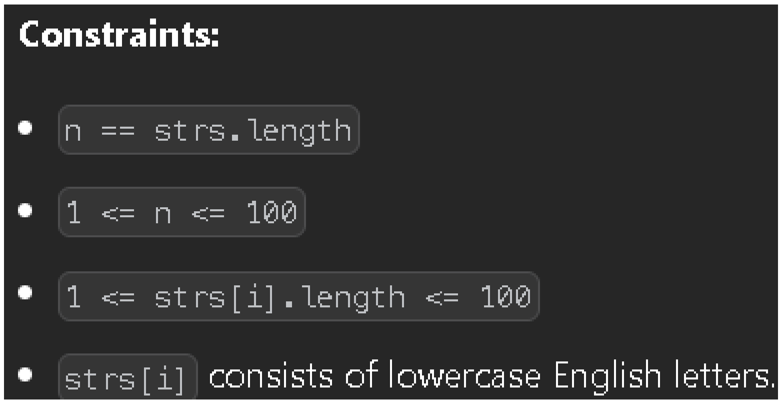
> **Input:** strs = ["ca","bb","ac"]
> **Output:** 1
> **Explanation:**
> After deleting the first column, strs = ["a", "b", "c"].
> Now strs is in lexicographic order (ie. strs[0] <= strs[1] <= strs[2]).
> We require at least 1 deletion since initially strs was not in lexicographic order, so the answer is 1.

Example 2:

> **Input:** strs = ["xc","yb","za"]
> **Output:** 0
> **Explanation:**
> strs is already in lexicographic order, so we do not need to delete anything.
> Note that the rows of strs are not necessarily in lexicographic order:
> i.e., it is NOT necessarily true that (strs[0][0] <= strs[0][1] <= ...)

Example 3:

> **Input:** strs = ["zyx","wvu","tsr"]
> **Output:** 3
> **Explanation:** We have to delete every column.


Constraints:

- `n == strs.length`
- `1 <= n <= 100`
- `1 <= strs[i].length <= 100`
- `strs[i]` consists of lowercase English letters.

Python:

```
class Solution:
    def minDeletionSize(self, strs: List[str]) -> int:
        n = len(strs)
        m = len(strs[0])

        resolved = [False] * (n - 1)
        unresolved = n - 1
        deletions = 0

        for col in range(m):
            if unresolved == 0:
                break

            bad = False
            for i in range(n - 1):
                if not resolved[i] and strs[i][col] > strs[i + 1][col]:
                    bad = True
                    break

            if bad:
                deletions += 1
                continue

            for i in range(n - 1):
                if not resolved[i] and strs[i][col] < strs[i + 1][col]:
                    resolved[i] = True
                    unresolved -= 1

        return deletions
```

JavaScript:

```
var minDeletionSize = function(strs) {
    const n = strs.length;
    const m = strs[0].length;

    // resolved[i] => strs[i] < strs[i+1] already determined
    const resolved = new Array(n - 1).fill(false);
    let unresolved = n - 1;
    let deletions = 0;

    for (let col = 0; col < m && unresolved > 0; col++) {
        let needDelete = false;

        // Check ordering violation
        for (let row = 0; row < n - 1; row++) {
            if (!resolved[row] && strs[row][col] > strs[row + 1][col]) {
                needDelete = true;
                break;
            }
        }

        if (needDelete) {
            deletions++;
            continue;
        }

        // Mark resolved row pairs
        for (let row = 0; row < n - 1; row++) {
            if (!resolved[row] && strs[row][col] < strs[row + 1][col]) {
                resolved[row] = true;
                unresolved--;
            }
        }
    }

    return deletions;
};
```

Java:

```
class Solution {
    public int minDeletionSize(String[] strs) {
        int n = strs.length;
        int m = strs[0].length();

        // Convert strings to char arrays once
        // This avoids repeated charAt() calls inside nested loops
        char[][] a = new char[n][];
        for (int i = 0; i < n; i++) {
            a[i] = strs[i].toCharArray();
        }

        // resolved[i] = true means:
        // strs[i] is already strictly smaller than strs[i+1]
        // considering previously kept columns
        boolean[] resolved = new boolean[n - 1];

        // Number of adjacent row pairs whose order is still undecided
        int unresolved = n - 1;

        int deletions = 0;
        // Process columns left to right
        for (int col = 0; col < m && unresolved > 0; col++) {
            boolean needDelete = false;

            // Check if keeping this column breaks lexicographical order
            for (int row = 0; row < n - 1; row++) {
                // Only compare rows whose order is not yet fixed
                if (!resolved[row] && a[row][col] > a[row + 1][col]) {
                    needDelete = true;
                    break;
                }
            }

            // If this column violates order, delete it
            if (needDelete) {
                deletions++;
                continue;
            }

            // Otherwise, update which row pairs become strictly ordered
            for (int row = 0; row < n - 1; row++) {
                if (!resolved[row] && a[row][col] < a[row + 1][col]) {
                    resolved[row] = true;
                    unresolved--;
                }
            }
        }

        return deletions;
    }
}
```