Problem Statement

You are given an array of `n` strings, 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, 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 **every string (row) in lexicographic** order. (i.e., `(strs[0][0] <= strs[0][1] <= ... <= strs[0][strs[0].length - 1])`, and `(strs[1][0] <= strs[1][1] <= ... <= strs[1][strs[1].length - 1])`, and so on). Return *the minimum possible value of* `answer.length`.

Input Format

First line contains single integer n, where n is the size of vector strs.

Second line contains the element of vector strs.

Output Format

Print *the minimum possible value of* `answer.length`.

Constraints

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

Sample Testcase 0

Testcase Input

```
2
babca bbazb
```

Testcase Output

```
3
```

Explanation

After deleting columns 0, 1, and 4, the final array is strs = ["bc", "az"].
Both these rows are individually in lexicographic order (ie. strs[0][0] <= strs[0][1] and strs[1][0] <= strs[1][1]).
Note that strs[0] > strs[1] - the array strs is not necessarily in lexicographic order.

Sample Testcase 1

Testcase Input

```
4
abc def ghi xyz
```

Testcase Output

```
0
```

Explanation

All rows are already lexicographically sorted.

CODE

```cpp
#include<bits/stdc++.h>
using namespace std;

int dp[100][101];

// i is the current column and j is the prev column
int solve(string strs[], int n, int i, int j)
{
    if(i == strs[0].size())
      return 0;

    if(dp[i][j+1] != -1)
      return dp[i][j+1];

    int left = solve(strs, n, i+1, j) + 1;
    int right = 1e9;
    if(j == -1) {
      right = solve(strs, n, i+1, i);
    } else {
      int r;
      for(r=0; r<n; r++) {
        if(strs[r][i] < strs[r][j])
          break;
      }
      if(r == n)
        right = solve(strs, n, i+1, i);
    }

    return dp[i][j+1] = min(left, right);
}

int main() {
    int n;
```

```cpp
    cin >> n;
    string strs[n];
    for(int i=0; i<n; i++) {
      cin >> strs[i];
    }
    memset(dp, -1, sizeof(dp));
    cout << solve(strs, n, 0, -1);
    return 0;
}
```

Here's an explanation of the code:

1. The code uses dynamic programming to solve the problem efficiently. It maintains a 2D array `dp` to store the results of subproblems. `dp[i][j]` represents the minimum number of deletions needed to make the substring consisting of columns from `i` to the end lexicographically ordered, given that the previous column was column `j`.
2. The `solve` function is a recursive function that calculates the minimum number of deletions for a specific substring of columns.
3. In the `main` function, it first reads the input values. It reads the integer `n`, which represents the number of strings in the array, and then reads the `n` strings into the `strs` array.
4. It initializes the `dp` array with -1 to indicate that no results have been computed yet.
5. It calls the `solve` function with the initial parameters `i = 0` (starting from the first column) and `j = -1` (no previous column). The result of the `solve` function is the minimum number of deletions needed to make all strings in the array lexicographically ordered.
6. The `solve` function itself is defined as follows:
    - If `i` has reached the end of the columns (`strs[0].size()`), it returns 0 because no further deletions are needed.
    - If the result for the current position `dp[i][j+1]` is already computed, it returns that result.
    - It calculates two possibilities:
        - `left` represents the case where the current column is deleted, so we increment `i` by 1, and the previous column `j` remains the same.
        - `right` represents the case where the current column is not deleted. To check this, it iterates through all rows `r` and checks if the character in the current column `i` is lexicographically smaller than the character in the previous column `j`. If it is, it breaks the loop, indicating that the current column cannot be retained to maintain lexicographical order. If `r` reaches `n`, it means all rows satisfy the condition, and `right` can be calculated by recursively calling `solve` with the current column `i` and updating `j` to the current column `i`.
    - The function returns the minimum of `left` and `right` as the result and stores it in `dp[i][j+1]`.
7. Finally, the `main` function prints the result obtained from the `solve` function, which represents the minimum number of deletions needed to make all strings lexicographically ordered.