2639. Find the Width of Columns of a Grid

Easy Array Matrix

Problem Description

In this problem, we are given a two-dimensional grid of integers, which represents a matrix with `m` rows and `n` columns. The task is to calculate the width of each column in this matrix. The width of a column is defined as the length of the longest number in that particular column. It's important to note that the length of a number includes all its digits and, additionally, the negative sign if the number is negative. For example, the number -123 would have a length of 4. The goal is to return an array that contains the width of each column in the grid.

Intuition

The intuition behind the solution is quite straightforward. We iterate through each cell of the matrix and convert each number to a string to calculate its length. If the number is negative, the string representation will include a minus sign, which will increase the length by one as expected.

During the iteration, we keep track of the maximum width seen so far for each column. We do this by maintaining an array `ans` where each index corresponds to a column of the grid, and the value at each index indicates the maximum width encountered in that column up to the current point in the iteration. Initially, all values in `ans` are set to 0.

Solution Approach

The solution to this problem uses a simple algorithm that iterates through each element of the matrix and applies basic string manipulation and comparison operations. The data structure used to keep track of the maximum width of each column is a list which is as long as the number of columns in the grid. This list is initialized with zeros, since initially, we do not have any width measured.

Here is an explanation of the steps in the algorithm using the provided Python code:

1. Initialize a list `ans` which has the same number of elements as there are columns in the grid (`len(grid[0])`). Each element of `ans` is set to `0`.
2. Loop through each row of the grid using a `for` loop with `row` as the iterator variable. During each iteration of this loop, you are looking at one row of numbers from the matrix.
3. Inside the row loop, another nested `for` loop goes through each number `x` in the row, with `j` tracking the current column index. So, `x` is the current number, and `j` is the index of the column `x` is in.
4. Convert the number `x` to a string using `str(x)` and calculate the width `w`, which is the length of this string. According to the rule, if `x` is negative, the length of the string will correctly include the negative sign.
5. Update the maximum width for the current column. Compare the width `w` with the current stored value in `ans[j]`. If `w` is larger, replace `ans[j]` with `w`.
6. Repeat steps 2 to 5 for all rows and columns in the grid. By the end of these loops, the `ans` list will have the maximum width for each column.
7. Return the `ans` list, which now contains the calculated width of each column within the matrix.

The simplicity of this algorithm makes it quite efficient; it has a time complexity of `O(m*n)` where `m` is the number of rows and `n` is the number of columns in the grid, because each element of the grid is visited exactly once.

Example Walkthrough

Let's use the solution approach to walk through a small example. Suppose we are given the following matrix:

```
[
  [12, -1],
  [134, 53],
  [-9, 6]
]
```

This matrix has 3 rows and 2 columns (`m=3`, `n=2`). Let's follow the steps to find the width of each column.

1. Initialize an array `ans` with the same number of elements as columns in the grid: `ans = [0, 0]`.
2. Start with the first row `[12, -1]`. Loop through the elements:
   - Consider the first element `12`, which, converted to string, has a width `2`. Compare it with `ans[0]` which is `0`. Since `2` is greater than `0`, update `ans[0]` to `2`.
   - Move to the second element `-1`, with a string width of `2` as well. Compare it with `ans[1]` which is `0`. Update `ans[1]` to `2`.
3. Move to the second row `[134, 53]` and repeat:
   - The element `134` has a width `3`. Compare it with `ans[0]` which is `2`. Update `ans[0]` to `3`.
   - Then, consider `53`, which has a width of `2`, but `ans[1]` is already `2`, so it remains unchanged.
4. Finally, for the third row `[-9, 6]`:
   - `-9` has a width `2`. `ans[0]` is `3`, so it remains unchanged.
   - `6` has a width of `1`. `ans[1]` is `2`, so no change.
5. After completing the iterations, `ans` now contains the maximum width for each column, which is `[3, 2]`.
6. The algorithm returns the array `[3, 2]`. This is the final output, indicating that the first column has a maximum width of 3 digits and the second column has a maximum width of 2 digits.

Solution Implementation

Python

```python
from typing import List

class Solution:
    def findColumnWidth(self, grid: List[List[int]]) -> List[int]:
        # Initialize a list to store maximum width for each column
        max_widths = [0] * len(grid[0])

        # Iterate over each row in the grid
        for row in grid:
            # Iterate over each element in the row
            for col_index, value in enumerate(row):
                # Convert the number to a string and get its length (character width)
                number_width = len(str(value))
                # Update the max_widths list with the maximum width for this column so far
                max_widths[col_index] = max(max_widths[col_index], number_width)

        # Return the list of maximum column widths
        return max_widths
```

Java

```java
class Solution {

    // Method that finds the maximum width needed for each column to fit the numbers
    // when printed in a table format.
    public int[] findColumnWidth(int[][] grid) {
        // Number of columns in the grid, which is the length of the first row.
        int numColumns = grid[0].length;

        // Array to store the maximum width required for each column.
        int[] maxWidths = new int[numColumns];

        // Loop through each row in the grid.
        for (int[] row : grid) {
            // Iterate over each column in the current row.
            for (int colIndex = 0; colIndex < numColumns; ++colIndex) {
                // Calculate the number of digits of the current cell's value.
                int width = String.valueOf(row[colIndex]).length();
                // Update the maximum width for the current column, if necessary.
                maxWidths[colIndex] = Math.max(maxWidths[colIndex], width);
            }
        }
        // Return the array containing the maximum widths for each column.
        return maxWidths;
    }
}
```

C++

```cpp
#include <vector>
#include <algorithm>
#include <string>
using namespace std;

class Solution {
public:
    // Function to determine the maximum width required for each column of a 2D grid.
    // The width is based on the number of digits in the numbers present in the column.
    vector<int> findColumnWidth(vector<vector<int>>& grid) {
        // Assuming grid is non-empty, get the number of columns from the first row.
        int numColumns = grid[0].size();
        // Initialize a vector to store the maximum width for each column.
        vector<int> columnWidths(numColumns, 0);

        // Iterate over each row in the grid.
        for (const auto& row : grid) {
            // Iterate over each value in the row.
            for (int columnIndex = 0; columnIndex < numColumns; ++columnIndex) {
                // Get the number of digits in the current value.
                int width = to_string(row[columnIndex]).size();
                // Update the maximum width for the column if the current value requires more space.
                columnWidths[columnIndex] = max(columnWidths[columnIndex], width);
            }
        }

        // Return the vector of maximum column widths.
        return columnWidths;
    }
};
```

TypeScript

```typescript
// Determines the maximum width needed for each column to display the numbers.
// It considers the number of digits (including any negative sign) of the largest number in each column.
// @param {number[][]} grid - The 2D array of numbers representing the grid.
// @return {number[]} An array of widths where each element represents the width required for the corresponding column.
function findColumnWidth(grid: number[][]): number[] {
    // Determine the number of columns based on the first row's length.
    const columnCount = grid[0].length;

    // Initialize an array to store the maximum widths with initial value 0 for each column.
    const columnWidths: number[] = new Array(columnCount).fill(0);

    // Iterate through each row of the grid.
    for (const row of grid) {
        // Iterate through each column in the row.
        for (let columnIndex = 0; columnIndex < columnCount; ++columnIndex) {
            // Calculate the width of the current entry (number of characters it contains).
            const entryWidth: number = String(row[columnIndex]).length;

            // Update the maximum width for the current column if necessary.
            columnWidths[columnIndex] = Math.max(columnWidths[columnIndex], entryWidth);
        }
    }

    // Return the array of maximum column widths.
    return columnWidths;
}
```

```python
from typing import List

class Solution:
    def findColumnWidth(self, grid: List[List[int]]) -> List[int]:
        # Initialize a list to store maximum width for each column
        max_widths = [0] * len(grid[0])

        # Iterate over each row in the grid
        for row in grid:
            # Iterate over each element in the row
            for col_index, value in enumerate(row):
                # Convert the number to a string and get its length (character width)
                number_width = len(str(value))
                # Update the max_widths list with the maximum width for this column so far
                max_widths[col_index] = max(max_widths[col_index], number_width)

        # Return the list of maximum column widths
        return max_widths
```

Time and Space Complexity

Time Complexity

The time complexity of the function `findColumnWidth` depends on the number of rows (`R`) and the number of columns (`C`) in the input `grid`. The outer loop iterates over each row, while the inner loop iterates over each column. The conversion of the integer `x` to a string has a time complexity proportional to the number of digits in `x`, which is bounded by a constant in this context since the values in `grid` are integers and are likely to have a bounded size in a practical scenario. Therefore, the time complexity is `O(R * C)`.

Space Complexity

The space complexity includes the space taken by the answer list, which is initialized with a length equal to the number of columns `C`. No other significant space is used, therefore the space complexity is `O(C)`.