[Click on the logo to find **Problem Statement**]

Intuition

The code implements a binary search algorithm to efficiently search for a target element in a 2D matrix. The matrix is treated as a one-dimensional array, and binary search is applied to this flattened representation. The algorithm initializes pointers (left and right) to the beginning and end of the flattened array and iteratively narrows down the search space by calculating the middle index. The midpoint is translated into row and column indices using divmod. If the element at the midpoint matches the target, the function returns True. If the target is greater, the search space is restricted to the right half; otherwise, it is constrained to the left half. The process continues until the target is found or the search space is exhausted.

Approach

1. **Initialization:**
    - Initialize pointers `left` and `right` to the first and last indices of the flattened matrix (`0` and `m * n - 1` , where m is the number of rows and n is the number of columns).
2. **Binary Search Iteration:**
    - Enter a while loop that continues as long as `left` is less than or equal to `right` .
    - Calculate the middle index `mid` using `(left + right) // 2` .
    - Translate the 1D midpoint `mid` into 2D row and column indices using `divmod(mid, n)` , where `n` is the number of columns.
    - Compare the element at the calculated 2D indices with the target:
        - If they are equal, return True as the target is found.
        - If the element is less than the target, update `left` to `mid + 1` to search the right half.
        - If the element is greater than the target, update `right` to `mid - 1` to search the left half.
3. **Target Not Found:**
    - If the while loop exits without finding the target, return False to indicate that the target is not present in the matrix.

Complexity

- Time complexity: O(log(m * n))
- Space complexity: O(1)

Code

```
class Solution:
    def searchMatrix(self, matrix: List[List[int]],
                     target: int) -> bool:
        if not matrix:
            return False
        m, n = len(matrix), len(matrix[0])
        left, right = 0, m * n - 1

        while left <= right:
            mid = (left + right) // 2
            mid_row, mid_col = divmod(mid, n)

            if matrix[mid_row][mid_col] == target:
                return True
            elif matrix[mid_row][mid_col] < target:
                left = mid + 1
            else:
                right = mid - 1

        return False
```

If you want to see more solutions to coding problems, you can visit:

GitHub