<https://leetcode.com/problems/first-completely-painted-row-or-column/description/>

You are given a **0-indexed** integer array arr, and an m x n integer **matrix** mat. arr and mat both contain **all** the integers in the range [1, m \* n].

Go through each index i in arr starting from index 0 and paint the cell in mat containing the integer arr[i].

Return *the smallest index* i *at which either a row or a column will be completely painted in* mat.

**Example 1:**

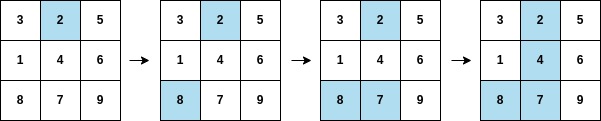


**Input:** arr = [1,3,4,2], mat = [[1,4],[2,3]]

**Output:** 2

**Explanation:** The moves are shown in order, and both the first row and second column of the matrix become fully painted at arr[2].

**Example 2:**



**Input:** arr = [2,8,7,4,1,3,5,6,9], mat = [[3,2,5],[1,4,6],[8,7,9]]

**Output:** 3

**Explanation:** The second column becomes fully painted at arr[3].

**Constraints:**

m == mat.length

n = mat[i].length

arr.length == m \* n

1 <= m, n <= 105

1 <= m \* n <= 105

1 <= arr[i], mat[r][c] <= m \* n

All the integers of arr are **unique**.

All the integers of mat are **unique**.

**Attempt 1: 2025-08-05**

**Solution 1: Hash Table (30 min)**

class Solution {

    public int firstCompleteIndex(int[] arr, int[][] mat) {

        int m = mat.length;

        int n = mat[0].length;

        Map<Integer, int[]> map = new HashMap<>();

        // Map to store positions of each element: element -> [row, col]

        for(int i = 0; i < m; i++) {

            for(int j = 0; j < n; j++) {

                map.put(mat[i][j], new int[] {i, j});

            }

        }

        // Track count of painted elements in each row and column

        int[] rowCount = new int[m];

        int[] colCount = new int[n];

        for(int i = 0; i < arr.length; i++) {

            int[] index = map.get(arr[i]);

            int row = index[0];

            int col = index[1];

            // Update counts

            rowCount[row]++;

            colCount[col]++;

            // Check if row or column is complete

            if(rowCount[row] == n || colCount[col] == m) {

                return i;

            }

        }

        // Shouldn't reach here per problem constraints

        return -1;

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to Deepseek**

**Approach**

**Precompute Positions**: Create a mapping from each matrix element to its (row, column) position

**Track Progress**: Maintain counts of painted elements for each row and column

**Check Completion**: For each element in arr:

Update the counts for its row and column

If either its row or column becomes fully painted, return the current index

**Solution Code**

class Solution {

public int firstCompleteIndex(int[] arr, int[][] mat) {

int m = mat.length;

int n = mat[0].length;

// Map to store positions of each element: element -> [row, col]

Map<Integer, int[]> posMap = new HashMap<>();

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

posMap.put(mat[i][j], new int[]{i, j});

}

}

// Track count of painted elements in each row and column

int[] rowCount = new int[m];

int[] colCount = new int[n];

for (int i = 0; i < arr.length; i++) {

int[] pos = posMap.get(arr[i]);

int r = pos[0];

int c = pos[1];

// Update counts

rowCount[r]++;

colCount[c]++;

// Check if row or column is complete

if (rowCount[r] == n || colCount[c] == m) {

return i;

}

}

return -1; // Shouldn't reach here per problem constraints

}

}

**Explanation**

**Position Mapping**: We first create a HashMap (posMap) that maps each value in the matrix to its (row, column) position. This allows O(1) access to any element's position.

**Tracking Progress**:

rowCount array tracks how many elements have been painted in each row

colCount array tracks how many elements have been painted in each column

**Processing the Array**:

For each element in arr, we:

Retrieve its position from posMap

Increment its row count and column count

Check if either its row count equals the number of columns (n) or its column count equals the number of rows (m)

The first index where either condition is met is returned immediately

This approach efficiently checks for complete rows/columns in O(mn) preprocessing time and O(len(arr)) processing time, with O(mn) space complexity for storing positions. The solution leverages direct access to element positions and constant-time updates to detect completion.

**Refer to**

[L2133.Check if Every Row and Column Contains All Numbers (Ref.L36,L37)](note://WEBfdf31b1322c8a4b1fb7c2cf200177f7b)