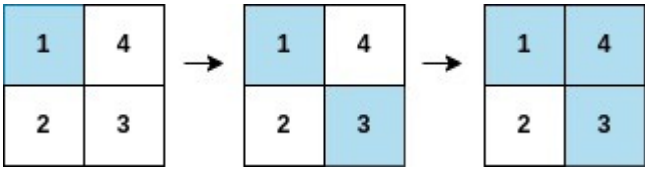
2661. First Completely Painted Row or Column

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]`.

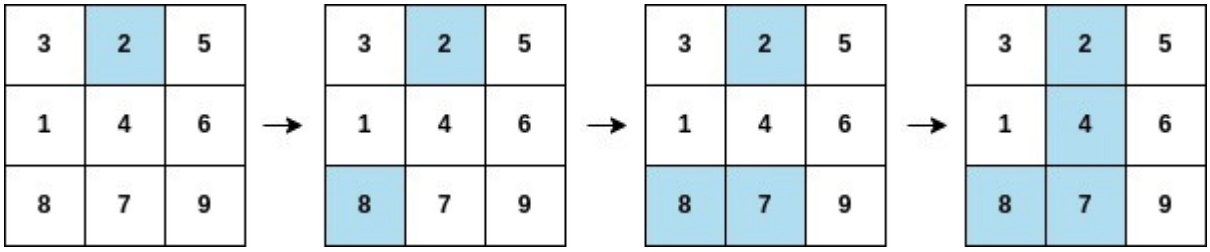
Return *the smallest index* `i` *at which either a row or a column will be completely painted in* `mat`.

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**

Overview

We are given two inputs: an array `arr` and a matrix `mat`. The array `arr` is a list of numbers, and the matrix `mat` is a grid where each cell contains one of these numbers. Both `arr` and `mat` contain all integers from 1 to $m \cdot n$, where m is the number of rows in the matrix and n is the number of its columns.

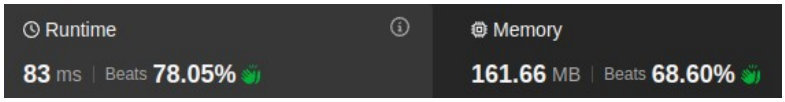
Our goal is to simulate a process where we "paint" the cells of the matrix in the order defined by `arr`. Starting from the first number in `arr`, we find the corresponding cell in `mat` and mark it as painted. As we progress through `arr`, more cells in `mat` will become painted.

We need to find the smallest index `i` in `arr` such that, after painting the cell corresponding to $arr[i]$, either:

1. An entire row in the matrix becomes completely painted (all cells in the row are marked).
2. An entire column in the matrix becomes completely painted (all cells in the column are marked).

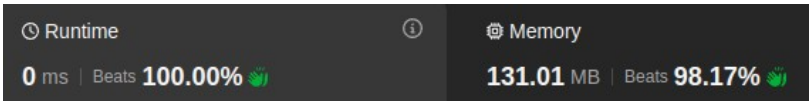
Note: Each number in `arr` corresponds to a unique cell in `mat`. This means no number is repeated, and every cell in the matrix will eventually be painted.

2661. First Completely Painted Row or Column




```
/*
   Mapping+Counting
   Time complexity: O(2mn)
   Space complexity: O(mn+(m+n))
*/
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;

class Solution {
   public:
      int firstCompleteIndex(vi& arr, vvi& mat) {
         int m=mat.size();
         int n=mat[0].size();

         // Preprocessing by each cell value in the matrix to its position (row,col)
         vvi val_pos(m*n+1); // Store cell_value: (row,col)
         for(int row=0;row<m;++row){
            for(int col=0;col<n;++col){
               int cell_value=mat[row][col];
               val_pos[cell_value]={row,col};
            }
         }

         // Create frequency array:
         //  --rows--columns--
         // [0~(m-1)~m~(m+n-1)]
         vi freq(m+n,0);

         // For each row and column, determine the number of painting
         for(int i=0;i<m*n;++i){
            // Element to paint
            int e=arr[i];

            // Lookup its position in matrix
            int row=val_pos[e][0];
            int col=val_pos[e][1];

            // Increment the row and column that belong to the element to paint
            freq[row]++;
            freq[m+col]++;

            // If the whole row is painted, or the whole column is painted return the index
            if(freq[row]==n || freq[m+col]==m) return i;
         }
         // Never reached, because it is guarantee that we have answer
         return m+n-1;
      }};
```

2661. First Completely Painted Row or Column




```
/*
   Reverse mapping
   Time complexity: O(3mn)
   Space complexity: O(mn)
*/
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
class Solution {
   public:
      int firstCompleteIndex(vi& arr, vvi& mat) {
         int m=mat.size();
         int n=mat[0].size();

         // Store the index of each number in the arr
         vi val_pos(m*n+1);
         for(int i=0;i<m*n;++i){
            val_pos[arr[i]]=i;
         }

         int ans=INT_MAX;

         // Determines the smallest index row completely painted
         for(int row=0;row<m;++row){
            // Tracks the greatest index in this row
            int max_row_index=INT_MIN;
            for(int col=0;col<n;++col){
               int e=mat[row][col];
               max_row_index=max(max_row_index,val_pos[e]);
            }
            // Update result with the minimum index where this row is fully
            // painted
            ans=std::min(ans,max_row_index);
         }

         // Determines the smallest index column completely painted
         for(int col=0;col<n;++col){
            // Tracks the greatest index in this column
            int max_col_index=INT_MIN;
            for(int row=0;row<m;++row){
               int e=mat[row][col];
               max_col_index=max(max_col_index,val_pos[e]);
            }
            // Update the answer with the minimum index where this column is fully
            // painted
            ans=std::min(ans,max_col_index);
         }
         return ans;
      }};
```