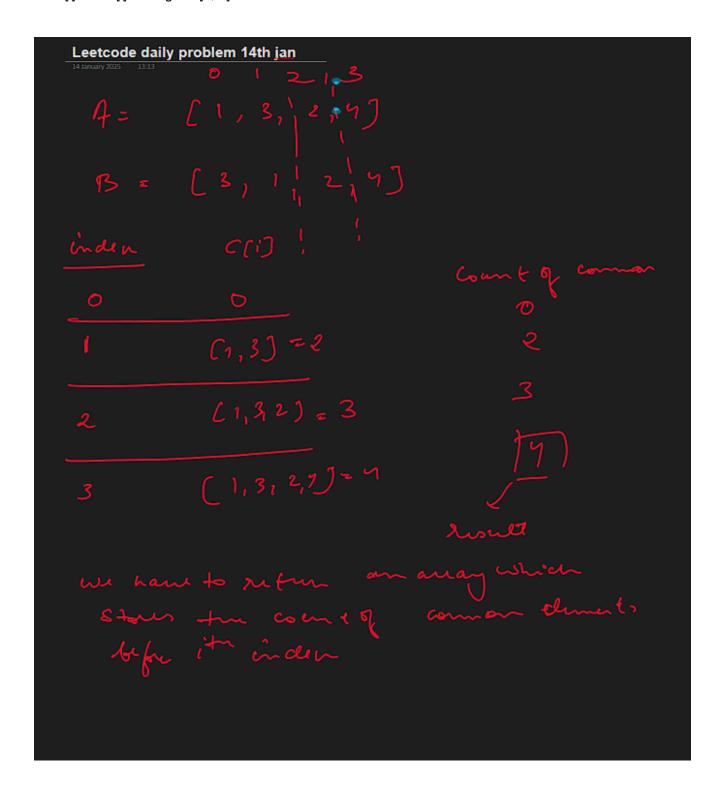
### 14.01.2025

## **Problem:**

- 1. two integers A and B of length n
- 2. a prefix common array of A and B is an array C such that **C[i]** is equal to count of numbers that are present at or before the index in both A and B.
- 3. return the prefix common array of A and B.
- 4. A[i] and B[i] belongs to [1, n].



# **Brute-force approach**

time complexity: O(n^3) space complexity: O(1)

### Better approach

let us say that if i = 2, that is in third iteration, if we have to search 1 of A array in B array, we will store that we had found 1 in range 0 to i index.

```
using boolean array
```

```
isPresentA
isPresentB
```

initially both the arrays will be marked as false

size of both arrays would be n+1 where n is the size of A and B, kyuki elements would range from 1 to n.

```
for i = 0, mark isPresenA[A[0]] as true and also mark isPresenB[B[0]] as true
```

now traverse both the boolean arrays and check whether both arrays are marked as true.

for i = 1, mark isPresenA[A[1]] as true and also mark isPresenB[B[1]] as true

now we will have,

```
isPresentA = {F, T, F, T, F}
isPresentB = {F, T, F, T, F}
```

therefore traversing both the array again and incrementing the count where true in both.

for i = 2, mark isPresenA[A[2]] as true and also mark isPresenB[B[2]] as true

now we will have,

```
isPresentA = {F, T, T, T, F}
isPresentB = {F, T, T, T, F}
```

therefore traversing both the array again and incrementing the count where true in both.

```
count = 3
```

for i = 3, mark isPresenA[A[3]] as true and also mark isPresenB[B[3]] as true

now we will have,

```
isPresentA = {F, T, F, T, T}
isPresentB = {F, T, F, T, T}
```

therefore traversing both the array again and incrementing the count where true in both. count = 4

```
for (int i = 0; i < n; i++){
    isPresentA[A[i]] = true;
    isPresentB[B[i]] = true;
    int count = 0;
    for(int j = 1; i <=n; i++){
        if(presentA[j] == true && ispresentb[B[j]] == true){
            count++;
        }
    }
    result[i] = count;
}
// time cpmplexity: 0(n^2)
// space complexity: 0(2 * (n + 1))</pre>
```

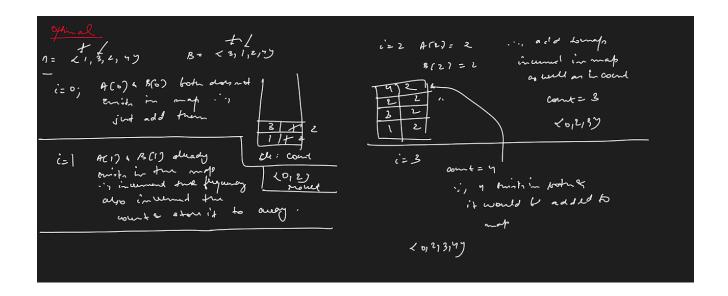
## **Optimal Solution:**

the logic behind the solution:

since we are given that there would be no duplicates in both the arrays also the elements would also belong to 1 to n both inclusive, this ensures that the count of an element would be at max 2. therefore, after adding both the elements to the map, we would check that if it's frequency is equal to 2 or not, if it is the we would increase the count.

we will use map data structure to store count of element with key as element.

dry run



#### using map:

```
class Solution {
public:
    vector<int> findThePrefixCommonArray(vector<int>& A, vector<int>& B) {
        int n = A.size();
        unordered_map<int, int>ump; // to store the element and frequnecy.
        int count = 0;
        vector<int>res(n);
        for(int i = 0; i < n; i++){
            // add current element to map.
            ump[A[i]]++;
           // no duplicates in an array therefore at max an element could be
            // 2 times
            if(ump[A[i]] == 2){
                count++;
            3
            ump[B[i]]++;
            if(ump[B[i]] == 2){
                count++;
            res[i] = count;
        return res;
   3
3;
```

#### using frequency array:

```
class Solution {
public:
    vector<int> findThePrefixCommonArray(vector<int>& A, vector<int>& B) {
    int n = A.size();
    int count = 0;
}
```

```
vector<int>res(n), freq(n+1, 0);
        for(int i = 0; i < n; i++){
            // add current element to map.
            freq[A[i]]++;
            \ensuremath{//} no duplicates in an array therefore at max an element could be
            // 2 times
            if(freq[A[i]] == 2){
                count++;
            }
            freq[B[i]]++;
            if(freq[B[i]] == 2){
               count++;
            res[i] = count;
        }
        return res;
   }
};
```