

3160. Find the Number of Distinct Colors Among the Balls

You are given an integer `limit` and a 2D array `queries` of size $n \times 2$.

There are `limit + 1` balls with **distinct** labels in the range $[0, \text{limit}]$. Initially, all balls are uncolored. For every query in `queries` that is of the form $[x, y]$, you mark ball x with the color y . After each query, you need to find the number of **distinct** colors among the balls.

Return an array `result` of length n , where `result[i]` denotes the number of distinct colors *after* i th query.

Note that when answering a query, lack of a color *will not* be considered as a color.

Example 1:

Input: `limit = 4, queries = [[1,4],[2,5],[1,3],[3,4]]`

Output: `[1,2,2,3]`

Explanation:

- After query 0, ball 1 has color 4.
- After query 1, ball 1 has color 4, and ball 2 has color 5.
- After query 2, ball 1 has color 3, and ball 2 has color 5.
- After query 3, ball 1 has color 3, ball 2 has color 5, and ball 3 has color 4.

Example 2:

Input: `limit = 4, queries = [[0,1],[1,2],[2,2],[3,4],[4,5]]`

Output: `[1,2,2,3,4]`

Explanation:

- After query 0, ball 0 has color 1.
- After query 1, ball 0 has color 1, and ball 1 has color 2.
- After query 2, ball 0 has color 1, and balls 1 and 2 have color 2.
- After query 3, ball 0 has color 1, balls 1 and 2 have color 2, and ball 3 has color 4.
- After query 4, ball 0 has color 1, balls 1 and 2 have color 2, ball 3 has color 4, and ball 4 has color 5.

Constraints:


- $1 \leq \text{limit} \leq 10^9$
- $1 \leq n == \text{queries.length} \leq 10^5$
- $\text{queries}[i].\text{length} == 2$
- $0 \leq \text{queries}[i][0] \leq \text{limit}$
- $1 \leq \text{queries}[i][1] \leq 10^9$

3160. Find the Number of Distinct Colors Among the Balls

Overview


Our task is to return an array listing the number of distinct colors after each query. Note that in this case, distinct means the number of total colors. It does not mean that the color only appears one time.

Let's look at an example of a potential set of queries:



Colors
4: 

i = 0
Limit = 6
Queries: **[1, 4]**, [3, 6], [5, 6],
[3, 2], [4, 3], [1, 6], [3, 7]

After query 0:
- Ball 1 has color 4




There is **1** distinct color
after this query.
Result = [**1**, __, __, __, __, __, __]

Colors
4: 
6: 

i = 1
Limit = 6
Queries: [[1, 4], **[3, 6]**, [5, 6],
[3, 2], [4, 3], [1, 6], [3, 7]]

After query 1:
- Ball 1 has color 4
- Ball 2 has color 6



There are **2** distinct colors
after this query.
Result = [1, **2**, __, __, __, __, __]

Colors

4: 

6: 

i = 2

Limit = 6

Queries: [[1, 4], [3, 6], **[5, 6]**, [3, 2], [4, 3], [1, 6], [3, 7]]

After query 2:

- Ball 1 has color 4
- Ball 3 has color 6
- Ball 5 has color 6



There are **2** distinct colors
after this query.

Result = [1, 2, **2**, __, __, __, __]

Colors

2: 

4: 

6: 

i = 3

Limit = 6

Queries: [[1, 4], [3, 6], [5, 6], **[3, 2]**, [4, 3], [1, 6], [3, 7]]

After query 3:


- Ball 1 has color 4
- Ball 3 has color 2
- Ball 5 has color 6



There are **3** distinct colors
after this query.

Result = [1, 2, 2, **3**, __, __, __]

Colors

2: 
3: 
4: 
6: 

i = 4

Limit = 6

Queries: [[1, 4], [3, 6], [5, 6],
[3, 2], [4, 3], [1, 6], [3, 7]]

After query 4:




- Ball 1 has color 4
- Ball 3 has color 2
- Ball 4 has color 3
- Ball 5 has color 6



There are 4 distinct colors
after this query.

Result = [1, 2, 2, 3, 4, __, __]

Colors

2: 
3: 
6: 

i = 5

Limit = 6

Queries: [[1, 4], [3, 6], [5, 6],
[3, 2], [4, 3], [1, 6], [3, 7]]

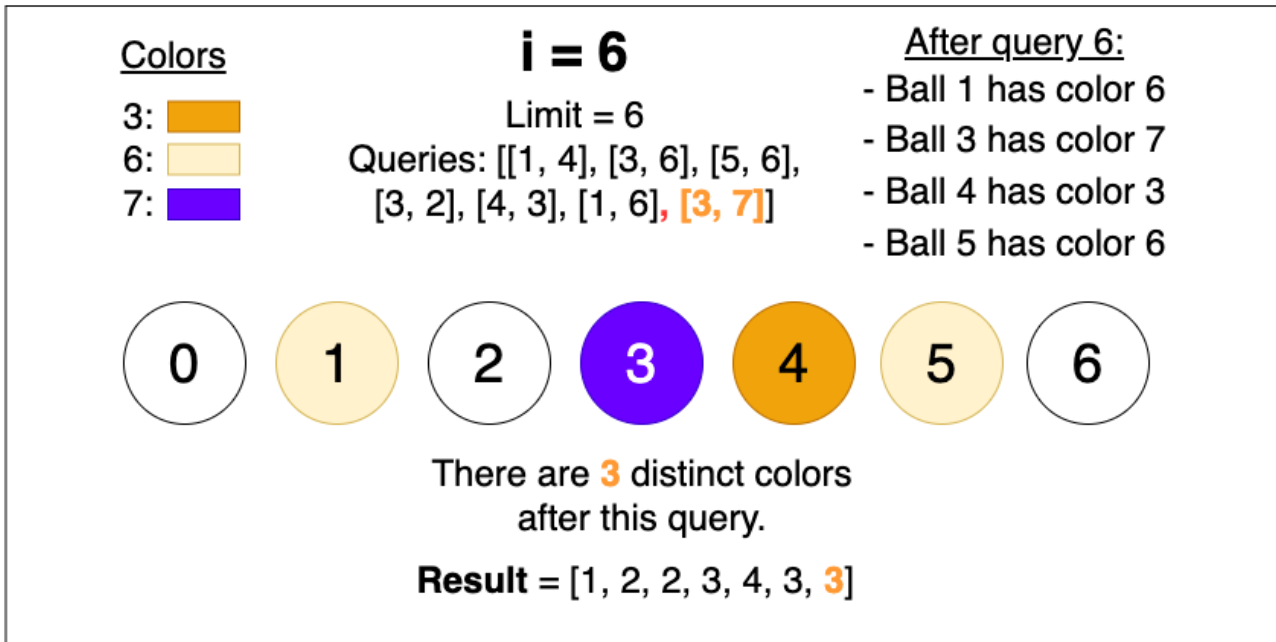
After query 5:

- Ball 1 has color 6
- Ball 3 has color 2
- Ball 4 has color 3
- Ball 5 has color 6



There are 3 distinct colors
after this query.

Result = [1, 2, 2, 3, 4, 3, __]



In this problem, two main scenarios can occur at each query:

1. **Uncolored Ball** - adding a color to a ball that did not already have a color
2. **Colored Ball** - adding a color to an already colored ball, replacing the previous color on the ball with the new color

Intuition

When approaching this problem, the main challenge is efficiently tracking and updating the colors of the balls after each query.

To solve this problem, we'll need to track both the number of times each color appears, and the number of distinct colors.

Let's consider the two different scenarios that occur when a query is applied to a ball. If the ball is:

1. Uncolored: the count of the newly assigned color is increased.
2. Colored: the count of the new color is increased and the count of the previously assigned color decreases.

Whether or not the number of distinct colors is impacted will depend on the total number of balls of that color already present.

A **hashmap** can be used for this purpose since it efficiently associates counts with specific colors.

We also need to track the current color of each ball because the problem involves overwriting existing colors. A straightforward solution is to use an **array** to store the color of each ball, where the index represents the ball and the value at that index represents the current color of the ball.

With these data structures in place, we can now proceed to process the queries. For each query, we update the color of the ball and adjust the count of distinct colors accordingly. As we process each query, we maintain the color count and track the balls' colors.

However, this solution ultimately fails due to exceeding the memory limit allowed for this problem.

The main challenge from the previous solution is identifying and addressing areas where large amounts of memory are used.

A significant portion of our memory usage comes from the array of size `limit + 1`. When we look at the constraints, we can see that the value of `limit` can be extremely large, with the range $1 \leq \text{limit} \leq 10^9$. Contrarily, the queries only range from $1 \leq n \leq 10^5$, where `n` is the length of `queries`. As we navigate through the queries, we can see that not all of the ball labels are guaranteed to be accessed by the queries, leading to unnecessary memory usage.

We can improve our storage efficiency by eliminating wasted space. Here, we need to choose a data structure that only allocates space as needed. Similar to how the colors are stored, we can utilize a **hash map** to store only the necessary labels accessed by the queries. By doing so, we can optimize the space complexity and prevent memory overuse.

After making this adjustment, we can apply the same logic and procedure as the previous solution. With this space optimization, we can process and track the results from the queries while staying within the memory limit.

3160. Find the Number of Distinct Colors Among the Balls

```
/*
Hash map+Naive way to compute distinct colors
Time complexity:  $O(m^2)$ 
Space complexity:  $O(n+m)$ 
n: #balls
m: #colors
*/
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
class Solution{
public:
    vi queryResults(int n, vvi& queries){
        vi ans;
        // Track colored balls and compute the frequency of each color
        std::unordered_map<int,int> balls,color_freq;

        // For each query
        for(auto& query: queries){
            int ball=query[0];
            int color=query[1];

            // If the ball is colored before, reduce the frequency of its color
            if(balls[ball]) color_freq[balls[ball]]--;

            // Color the ball
            balls[ball]=color;

            // Increment the frequency the the color
            color_freq[color]++;

            // Iterate oven the colors and compute the colors with frequencies
            // different to 0. Means, the colors which still exists
            int distinct_colored_balls=0;
            for(auto& [color,f]: color_freq) if(f>0) distinct_colored_balls++;

            // Update the answer table.
            ans.push_back(distinct_colored_balls);
        }
        return ans;
    }
};
```

3160. Find the Number of Distinct Colors Among the Balls

```
/*
    Hash map+erase non-existing colors
    Time complexity: O(m)
    Space complexity: O(n+m)
    n: #balls
    m: #colors
*/
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
class Solution{
public:
    vi queryResults(int n, vvi& queries) {
        vi ans;

        // Track colored balls and compute the frequency of each color
        std::unordered_map<int,int> balls,color_freq;

        int distinct_colored_balls=0; // To track the distinct colors
        for(auto& query: queries){
            int ball=query[0];
            int color=query[1];

            //If the ball is colored before,
            if(balls[ball]){
                // reduce the frequency of its color
                color_freq[balls[ball]]--;

                // If the color of the current ball does not exist any more, remove
                // its color from the hash map
                if(!color_freq[balls[ball]]) color_freq.erase(balls[ball]);
            }

            // Increment the frequency the color
            color_freq[color]++;

            // Color the ball
            balls[ball]=color;

            // Update the answer table
            ans.push_back(color_freq.size());
        }
        return ans;
    }
};
```