Problem 3160: Find the Number of Distinct Colors Among the Balls

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given an integer

limit

and a 2D array

queries

of size

n x 2

.

There are

limit + 1

balls with

distinct

labels in the range

[0, 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 colors among the balls.

Return an array

result

of length

n

, where

result[i]

denotes the number of colors

after

i

th

query.

Note

that when answering a query, lack of a color

will not

be considered as a color.

Example 1:

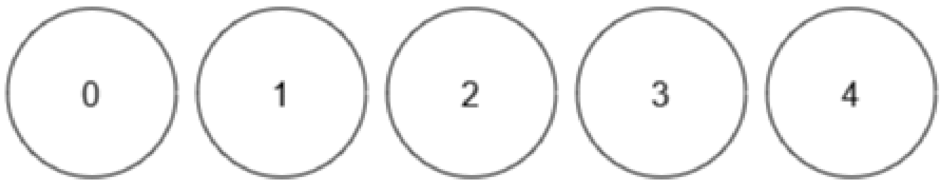
Input:

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:

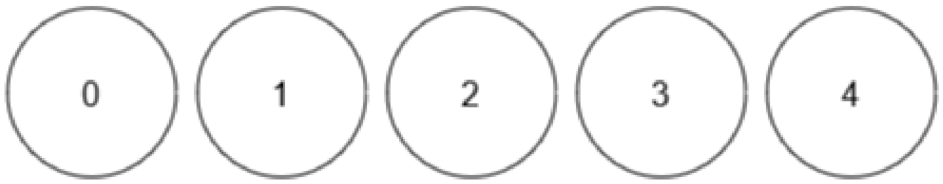
Input:

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 <= limit <= 10

9

1 <= n == queries.length <= 10

5

queries[i].length == 2

0 <= queries[i][0] <= limit

1 <= queries[i][1] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
vector<int> queryResults(int limit, vector<vector<int>>& queries) {

}
};
```

Java:

```
class Solution {
public int[] queryResults(int limit, int[][] queries) {

}
}
```

Python3:

```
class Solution:
def queryResults(self, limit: int, queries: List[List[int]]) -> List[int]:
```

Python:

```
class Solution(object):
def queryResults(self, limit, queries):
"""
:type limit: int
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number} limit
* @param {number[][]} queries
* @return {number[]}
*/
var queryResults = function(limit, queries) {

};
```

TypeScript:

```
function queryResults(limit: number, queries: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] QueryResults(int limit, int[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* queryResults(int limit, int** queries, int queriesSize, int*
queriesColSize, int* returnSize) {

}
```

Go:

```
func queryResults(limit int, queries [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun queryResults(limit: Int, queries: Array<IntArray>): IntArray {

}
}
```

Swift:

```
class Solution {
func queryResults(_ limit: Int, _ queries: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn query_results(limit: i32, queries: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} limit
# @param {Integer[][]} queries
# @return {Integer[]}
def query_results(limit, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer $limit
* @param Integer[][] $queries
* @return Integer[]
*/
function queryResults($limit, $queries) {

}

}
```

Dart:

```
class Solution {
List<int> queryResults(int limit, List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def queryResults(limit: Int, queries: Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec query_results(limit :: integer, queries :: [[integer]]) :: [integer]
def query_results(limit, queries) do

end
end
```

Erlang:

```
-spec query_results(Limit :: integer(), Queries :: [[integer()]]) ->
[integer()].
query_results(Limit, Queries) ->
.
```

Racket:

```
(define/contract (query-results limit queries)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Number of Distinct Colors Among the Balls
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
vector<int> queryResults(int limit, vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Number of Distinct Colors Among the Balls
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int[] queryResults(int limit, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Number of Distinct Colors Among the Balls
```

```
Difficulty: Medium
Tags: array, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def queryResults(self, limit: int, queries: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def queryResults(self, limit, queries):
"""
:type limit: int
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Find the Number of Distinct Colors Among the Balls
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number} limit
* @param {number[][]} queries
* @return {number[]}
*/
var queryResults = function(limit, queries) {
};
```

TypeScript Solution:

```
/**
* Problem: Find the Number of Distinct Colors Among the Balls
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function queryResults(limit: number, queries: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Find the Number of Distinct Colors Among the Balls
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int[] QueryResults(int limit, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Find the Number of Distinct Colors Among the Balls
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* queryResults(int limit, int** queries, int queriesSize, int*
queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Find the Number of Distinct Colors Among the Balls
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func queryResults(limit int, queries [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun queryResults(limit: Int, queries: Array<IntArray>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func queryResults(_ limit: Int, _ queries: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Find the Number of Distinct Colors Among the Balls
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn query_results(limit: i32, queries: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} limit
# @param {Integer[][]} queries
# @return {Integer[]}
def query_results(limit, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $limit
* @param Integer[][] $queries
* @return Integer[]
*/
function queryResults($limit, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> queryResults(int limit, List<List<int>> queries) {

}
}
```

Scala Solution:

```
object Solution {
def queryResults(limit: Int, queries: Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec query_results(limit :: integer, queries :: [[integer]]) :: [integer]
def query_results(limit, queries) do

end
end
```

Erlang Solution:

```
-spec query_results(Limit :: integer(), Queries :: [[integer()]]) ->
[integer()].
query_results(Limit, Queries) ->
.
```

Racket Solution:

```
(define/contract (query-results limit queries)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?))
)
```