Problem 1434: Number of Ways to Wear Different Hats to Each Other

Problem Information

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

Problem Description

There are

n

people and

40

types of hats labeled from

1

to

40

.

Given a 2D integer array

hats

, where

hats[i]

is a list of all hats preferred by the

i

th

person.

Return the number of ways that

n

people can wear

different

hats from each other.

Since the answer may be too large, return it modulo

10

9

+ 7

.

Example 1:

Input:

hats = [[3,4],[4,5],[5]]

Output:

1

Explanation:

There is only one way to choose hats given the conditions. First person choose hat 3, Second person choose hat 4 and last one hat 5.

Example 2:

Input:

hats = [[3,5,1],[3,5]]

Output:

4

Explanation:

There are 4 ways to choose hats: (3,5), (5,3), (1,3) and (1,5)

Example 3:

Input:

hats = [[1,2,3,4],[1,2,3,4],[1,2,3,4],[1,2,3,4]]

Output:

24

Explanation:

Each person can choose hats labeled from 1 to 4. Number of Permutations of (1,2,3,4) = 24.

Constraints:

n == hats.length

1 <= n <= 10

1 <= hats[i].length <= 40

1 <= hats[i][j] <= 40

hats[i]

contains a list of

unique

integers.

Code Snippets

C++:

```
class Solution {
public:
int numberWays(vector<vector<int>>& hats) {

}
};
```

Java:

```
class Solution {
public int numberWays(List<List<Integer>> hats) {

}
}
```

Python3:

```
class Solution:
def numberWays(self, hats: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def numberWays(self, hats):
```

```
"""
:type hats: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} hats
* @return {number}
*/
var numberWays = function(hats) {

};
```

TypeScript:

```
function numberWays(hats: number[][]): number {

};
```

C#:

```
public class Solution {
public int NumberWays(IList<IList<int>> hats) {

}
}
```

C:

```
int numberWays(int** hats, int hatsSize, int* hatsColSize) {

}
```

Go:

```
func numberWays(hats [][]int) int {

}
```

Kotlin:

```
class Solution {
fun numberWays(hats: List<List<Int>>): Int {

}
}
```

Swift:

```
class Solution {
func numberWays(_ hats: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn number_ways(hats: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} hats
# @return {Integer}
def number_ways(hats)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $hats
* @return Integer
*/
function numberWays($hats) {

}
}
```

Dart:

```
class Solution {
int numberWays(List<List<int>> hats) {

}
}
```

Scala:

```
object Solution {
def numberWays(hats: List[List[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec number_ways(hats :: [[integer]]) :: integer
def number_ways(hats) do

end
end
```

Erlang:

```
-spec number_ways(Hats :: [[integer()]]) -> integer().
number_ways(Hats) ->
.
```

Racket:

```
(define/contract (number-ways hats)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Ways to Wear Different Hats to Each Other
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int numberWays(vector<vector<int>>& hats) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Ways to Wear Different Hats to Each Other
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int numberWays(List<List<Integer>> hats) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Ways to Wear Different Hats to Each Other
Difficulty: Hard
Tags: array, dp
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

```
class Solution(object):
def numberWays(self, hats):
"""
:type hats: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Ways to Wear Different Hats to Each Other
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[][]} hats
* @return {number}
*/
var numberWays = function(hats) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Ways to Wear Different Hats to Each Other
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function numberWays(hats: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Ways to Wear Different Hats to Each Other
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int NumberWays(IList<IList<int>> hats) {

}
}
```

C Solution:

```
/*
* Problem: Number of Ways to Wear Different Hats to Each Other
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

int numberWays(int** hats, int hatsSize, int* hatsColSize) {

}
```

Go Solution:

```
// Problem: Number of Ways to Wear Different Hats to Each Other
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func numberWays(hats [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numberWays(hats: List<List<Int>>): Int {

}
}
```

Swift Solution:

```
class Solution {
func numberWays(_ hats: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Ways to Wear Different Hats to Each Other
// Difficulty: Hard
// Tags: array, dp
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn number_ways(hats: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} hats
# @return {Integer}
def number_ways(hats)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $hats
* @return Integer
*/
function numberWays($hats) {

}
}
```

Dart Solution:

```
class Solution {
int numberWays(List<List<int>> hats) {

}
}
```

Scala Solution:

```
object Solution {
def numberWays(hats: List[List[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec number_ways(hats :: [[integer]]) :: integer
def number_ways(hats) do

end
end
```

Erlang Solution:

```
-spec number_ways(Hats :: [[integer()]]) -> integer().
number_ways(Hats) ->
.
```

Racket Solution:

```
(define/contract (number-ways hats)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```