Introduction to Leetcode

Table of contents

01
Introduction
What and why leetcode?

02
Problems
Discuss sample problems

03
Resources
Useful links

04
Outro
QnA

https://leetcode.com

Platform to practice algorithmic problems

Why LeetCode ?

Interview

Specifically technical interviews

OA

Stands for Online Assessment

Exposure

They say,
“Practice is the key to success.”

Problem 1. Majority Elements

Abridged Problem Statement

Given an array **A** of size **N**.
Return the element that appears more than **⌊N / 2⌋** times.
It is guaranteed that such majority element exists.

Problem 1. Majority Elements

Naive Solution

Store elements’ frequencies in a memo (dictionary) then find return the one with highest frequency.

```
def majorityElement(self, nums: List[int]) ->
int:
     freq = {}
     for num in A:
          if num not in freq:
                freq[num] = 1
          else:
                freq[num] += 1
          if freq[num] > len(num) // 2:
               return key
```

Complexity Analysis:

- Time: O(N)
- Memory: O(N)

Problem 1. Majority Elements

Optimized Solution

Bayer-Moore’s Algorithm.

```
majority, count = -1, 0
for num in A:
  if count == 0:
    majority = num
  if majority == num:
    count += 1
  else:
    count -= 1
return majority
```

Suppose the algorithm doesn’t return majority element. This means that the count of majority element will decrease to 0. However, by definition, there is no element that can decrease the majority element’s count to 0.

Hence, a contradiction.

Complexity Analysis:

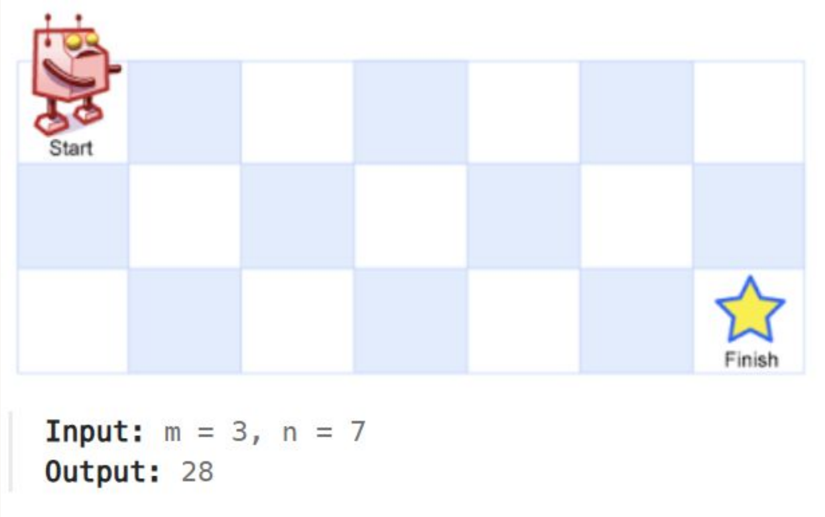
- Time: O(N)
- Memory: O(1)

Problem 2. Unique Paths

Abridged Problem Statement

There is a grid of size **M x N**. You are located at the **top-left corner (0, 0)**. Your goal is to move to the bottom-right corner **(M-1, N-1)**. You can only move either down or right at any point in time.

What is the number of possible unique paths?

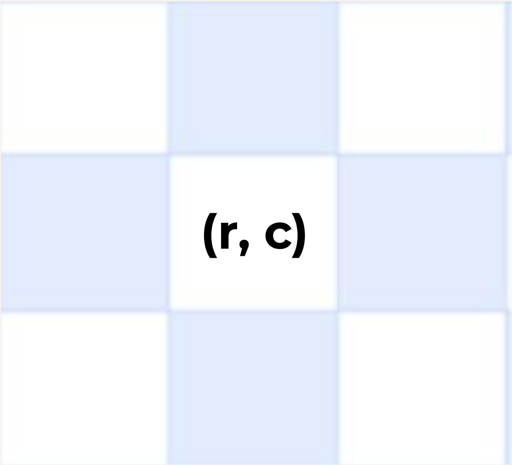


```
Input: m = 3, n = 7
Output: 28
```

Problem 2. Unique Paths

Solution

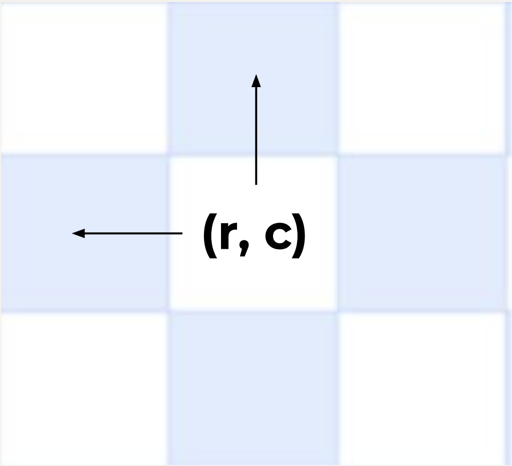
Let **ways[r][c]** denote the number of ways to reach the position **(r, c)**.
What relations can you draw?



Problem 2. Unique Paths

Solution

Notice that to reach **(r, c)**, we must be either coming from **(r-1, c)** or **(r, c-1).**
Thus, **ways[r][c] = ways[r-1][c] + ways[r][c-1]**

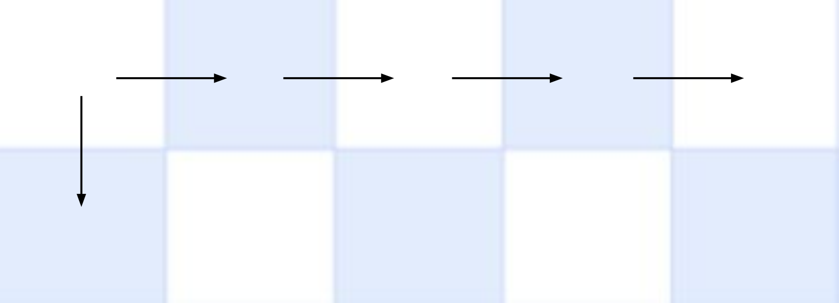


Problem 2. Unique Paths

Solution

What about the borders?

The number of ways is **1**.



Problem 2. Unique Paths

Solution

```
def uniquePaths(self, m: int, n: int) -> int:
        ways = [[-1] * n for _ in range(m)]
        def solve(r, c):
            if r == 0 or c == 0:
                return 1
            if ways[r][c] != -1:
                return ways[r][c]
            ways[r][c] = solve(r-1, c) + solve(r, c-1)
            return ways[r][c]

        return solve(m-1, n-1)
```

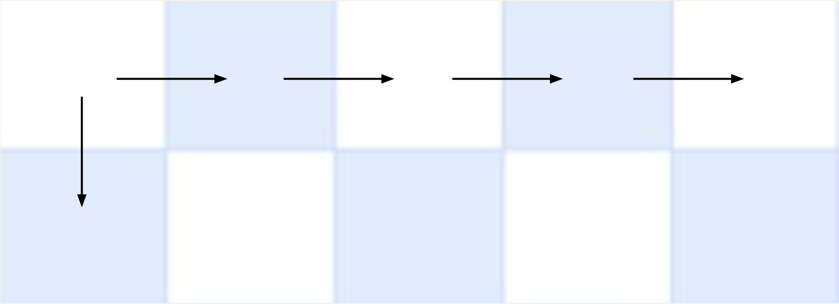
Complexity Analysis:

- Time: O(NM)
- Memory: O(NM)

Problem 2. Unique Paths

Solution

We can also depict the number of ways as ways to rearrange the arrows



In the above example, we have **5** arrows, **4** right and **1** left. Notice that the number of ways to rearrange the arrows is “**Combination(5, 1) = 5**”

Problem 2. Unique Paths

Solution

Generally, in a grid of size **M x N**, we have **M-1** down arrows and **N-1** right arrows. We need to choose locations for **M-1** (or **N-1**) arrows out of the **M+N-2** available slots.

Thus, the general formula is **Combination(M+N-2, M-1)**

```
math.comb(m+n-2, m-1)
```

Complexity Analysis:

- Time: O(max (N, M))
- Memory: O(1)

Topics Recap

- Streaming Algorithm
- Dynamic Programming
- Combinatorics

Topics to Learn

- Divide and Conquer
- Dynamic Programming
- Greedy
- Graph Traversals (DFS, BFS, Dijkstra, A*)
- Data Structures (Heap, Disjoint Set, AVL Tree, Segment Tree)
- Flow*

Resources

- **CSC236/240, CSC263/265, CSC373, CSC473**
- Google / Youtube
- Blind-75
- Competitive Programming Handbook
- CSES Problemset

Thanks

Any questions ?

Check CSSU instagram for information on future **LeetCode Nights**!

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, infographics & images by **Freepik**

Please keep this slide for attribution