🧩 LeetCode #62 — Unique Paths

🔷 Problem Statement

A robot is placed on an `m x n` grid, starting from the **top-left corner** `(0,0)` and aiming to reach the **bottom-right corner** `(m-1, n-1)`.

At any point, the robot can only move:

- ➡️ **Right**
- ⬇️ **Down**

We must find **how many unique paths** exist from start to end.

🧠 Intuition

Every move the robot makes splits into **two choices**:

- Move **right** (→)
- Move **down** (↓)

This naturally forms a **recursive tree**, where each node represents a position `(sr, sc)` on the grid.

We keep exploring until we reach the **destination cell**.

But... many subproblems repeat!

Hence, we apply **Dynamic Programming (DP)** to cache results and avoid redundant work.

⚙️ Approach: Top-Down DP (Memoization)

We use recursion to explore all paths and memoization to store results of already visited states.

Steps:

1. Start recursion from `(0, 0)` and aim for `(m-1, n-1)`.
2. Base cases:
   - ✅ If `(sr == er && sc == ec)` → return 1 (reached destination)
   - 🚫 If `(sr > er || sc > ec)` → return 0 (out of grid)
3. Check if the current cell `(sr, sc)` is already computed in `dp`:
   - If yes → directly return it.
   - If no → calculate:
     - Move right: `helper(sr, sc + 1, er, ec, dp)`
     - Move down: `helper(sr + 1, sc, er, ec, dp)`
4. Store the sum of both paths in `dp[sr][sc]`.
5. Return the final result.

💻 Code Implementation

```
class Solution {
public:
    int helper(int sr, int sc, int er, int ec, vector<vector<int>>& dp) {
        if (sr == er && sc == ec) return 1;
        if (sr > er || sc > ec) return 0;
        if (dp[sr][sc] != -1) return dp[sr][sc];

        int rightway = helper(sr, sc + 1, er, ec, dp);
        int downway = helper(sr + 1, sc, er, ec, dp);

        return dp[sr][sc] = rightway + downway;
    }

    int uniquePaths(int m, int n) {
        vector<vector<int>> dp(m, vector<int>(n, -1));
        return helper(0, 0, m - 1, n - 1, dp);
```

```
    }
};
```

🧾 Dry Run Example

Input:

```
m = 3, n = 2
```

Grid visualization:

```
(0,0) → (0,1)
 ↓       ↓
(1,0) → (1,1)
 ↓       ↓
(2,0) → (2,1)
```

Possible Paths:

1️⃣ Right → Down → Down

2️⃣ Down → Down → Right

3️⃣ Down → Right → Down

✅ **Total Paths = 3**

🧮 Time & Space Complexity

Type	Complexity	Explanation
Time	`O(m × n)`	Each cell `(i,j)` computed once
Space	`O(m × n)`	For DP table
Auxiliary (Recursion Stack)	`O(m + n)`	Depth of recursion

💡 Alternative (Combinatorics Formula)

The robot must take exactly `(m-1)` down moves and `(n-1)` right moves → total `(m+n-2)` moves.

So total unique paths =

[

$$C(m+n-2, m-1) = \frac{(m+n-2)!}{(m-1)! \times (n-1)!}$$

]

This is faster and uses **O(1)** space.

🏁 Key Takeaways

- A grid traversal problem → naturally recursive.
- Memoization optimizes recursion to $O(m \times n)$.
- DP helps avoid recomputation of overlapping subproblems.
- Can also be solved mathematically using combinations.
