Problem – Climbing Stairs

Easy

LeetCode leetcode.com/problems/climbing-stairs

Problem

- You need to climb a staircase with **n steps**
- Each time, you can only climb either **1 step** or **2 steps**
- Find out how **many different ways** you can **climb to the top**
- **Example:**

Input

n = 2

Output: 2

Explanation:

1. Climb 1 step, then climb 1 step again
2. Climb 2 steps in one go

Solution – Climbing Stairs

Easy

LeetCode leetcode.com/problems/climbing-stairs

Solution

- **You need to climb a staircase with n steps** and want to find the total number of distinct ways to reach the top
- **At each step, you can either take 1 step or 2 steps -** this gives you two choices at most positions
- **This follows the Fibonacci sequence pattern** - the number of ways to reach step n equals ways to reach (n-1) plus ways to reach (n-2)
- **Use dynamic programming to avoid recalculating subproblems** - either bottom-up tabulation or memoized recursion works well
- **Base cases are crucial**: typically f(1) = 1 and f(2) = 2, representing the ways to reach the first and second steps

Solution – Climbing Stairs

Easy

LeetCode leetcode.com/problems/climbing-stairs

Code

```
std::unordered_map<int, int> memo;

int climbStairs(int n) {
    // Identify the sequence, when:
    // n = 0 (0 way), there is no way to get up
    // n = 1 (1 way): only one way : 1-step
    // n = 2 (2 ways): 1s + 1s | 2s
    // n = 3 (3 ways): 1s + 1s + 1s |  1s + 2s | 2s + 1s
    // n = 4 (5 ways): 1s + 1s + 1s + 1s | 1s + 1s + 2s | 1s + 2s + 1s | 2s + 1s + 1s | 2s + 2s |

    if (n <= 2) {
        return n;
    }

    if (memo.find(n) != memo.end()) {
        return memo[n];
    }

    memo[n] = climbStairs(n - 1) + climbStairs(n - 2);
    return memo[n];
}
```

Problem – 1143. Longest Common Subsequence

Medium

LeetCode https://leetcode.com/problems/longest-common-subsequence

Problem

- You are given two strings, example:

  text1 = "abcd"

  text2 = "ace"

- `Find the longest subsequence between them`

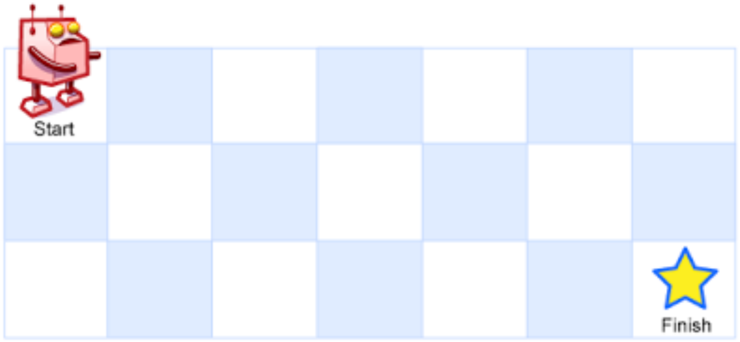
Problem – 62. Unique Paths

Medium

LeetCode https://leetcode.com/problems/unique-paths

Problem

- The robot is placed in a m x n grid
- It starts at the top-left cell (0,0) and must reach the bottom-right (m – 1, n – 1)
- The robot can only move right or down at any point
- Return the number of unique paths the robot can take to reach the destination



Solution – 62. Unique Paths

Medium

LeetCode https://leetcode.com/problems/unique-paths

Solution 1 (recursive)

- Define a recursive function countPaths(m, n)
- **Base case**

  If m == 1 or n == 1, there's only one way to reach that cell (either all downs or all rights).
- **Recursive case**

  To reach cell (m, n) the robot must come from:

  Cell (m - 1, n) → from above

  Cell (m, n - 1) → from left

  So the number of of paths to (m, n) is the **sum** of the paths to those two cells
- **Memoization**

  Use a 2D vector[m + 1][n + 1]

Code – 62. Unique Paths

Medium

LeetCode https://leetcode.com/problems/unique-paths

Code

Time: **O(m * n)** Space: **O(m * n)**

```
int countPaths(int m, int n, vector<vector<int>>& memo) {
    if (m == 1 || n == 1) return 1;
    if (memo[m][n] != -1) return memo[m][n];
    memo[m][n] = countPaths(m, n - 1, memo) + countPaths(m - 1, n, memo);
    return memo[m][n];
}
int uniquePaths(int m, int n) {
    /*
      count(m, n) = count(m, n + 1) + count(m + 1, n)
      same as (imagine robot going from m,n to 0,0 up and left)
      count(m, n) = count(m, n - 1) + count(m - 1, n)
    */
    vector<vector<int>> memo(m + 1, vector<int>(n + 1, -1));
    return countPaths(m, n, memo);
}
```

Solution – 62. Unique Paths

Medium

LeetCode https://leetcode.com/problems/unique-paths

Solution 2 (iterative)

- Create a 2D vector dp of size (m+1) × (n+1) to store intermediate results
- Set dp[1][1] = 1 because there is exactly one way to stand on the starting cell
- Iterate through each cell (row, col) from (1, 1) to (m, n):

  - Skip (1, 1) since it's already initialized

  - For every other cell, the number of unique paths to it is the sum of:

    - Paths from the cell above: dp[row-1][col]

    - Paths from the cell to the left: dp[row][col-1]

    dp[row][col] = dp[row - 1][col] + dp[row][col - 1]