Climbing Stairs Problem Documentation

Problem Statement

You are climbing a staircase that takes n steps to reach the top. Each time you can either climb 1 step or 2 steps. The problem is to determine the number of distinct ways you can climb to the top.

Example 1

- **Input:** n = 2
- **Output:** 2
- **Explanation:** There are two ways to climb to the top:
    1. 1 step + 1 step
    2. 2 steps

Example 2

- **Input:** n = 3
- **Output:** 3
- **Explanation:** There are three ways to climb to the top:
    1. 1 step + 1 step + 1 step
    2. 1 step + 2 steps
    3. 2 steps + 1 step

Constraints

- (1 leq n leq 45)

Solution

To solve this problem, we can use a dynamic programming approach. The key idea is to recognize that the number of ways to reach step n is the sum of the ways to reach step n-1 and step n-2. This is because from step n-1, you can take a 1-step to reach n, and from step n-2, you can take a 2-step to reach n.

Detailed Steps

1. **Base Cases:**

- If n == 1, there is only one way to climb to the top: 1 step.
- If n == 2, there are two ways to climb to the top: 1 step + 1 step, or 2 steps.

2. **Dynamic Programming Array Initialization:**

- Create a list dp of size n+1 to store the number of ways to reach each step.
- Initialize dp[1] to 1 and dp[2] to 2.

3. **Filling the DP Array:**

- For each step from 3 to n, compute the number of ways to reach that step by summing the ways to reach the two preceding steps:

[

dp[i] = dp[i-1] + dp[i-2]

]

4. **Return the Result:**

- The value at dp[n] will be the number of distinct ways to reach the top of the staircase.

Explanation of the Code

1. **Initialization:** Check for the base cases where n is 1 or 2.
2. **DP Array Creation:** Create an array dp to store the number of ways to reach each step.
3. **Base Case Assignment:** Set dp[1] to 1 and dp[2] to 2.
4. **Filling DP Array:** Use a for loop to fill the rest of the dp array using the recurrence relation.
5. **Return the Result:** The final number of ways to reach the top is found in dp[n].

This solution ensures efficient computation of the number of ways to climb the stairs using dynamic programming with a time complexity of (O(n)) and a space complexity of (O(n)).