Leetcode 70. Climbing Stairs

2024.01.21

원루빈

Problem Description

You are climbing a staircase. It takes n steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you 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:

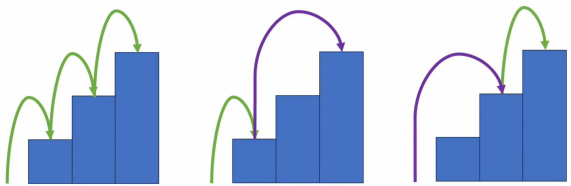
- `1 <= n <= 45`

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



1 You can either take 1 step or 2 steps (Recursive Approach)

We have only two choices:

If you choose to take 1 step → how many ways can you climb the remaining stairs?

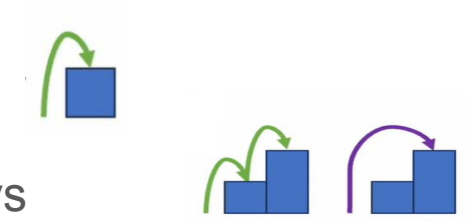
If you choose to take 2 steps → how many ways can you climb the remaining stairs?

Recursive Definition:

# of ways to climb n-1 steps + # of ways to climb n-2 steps

Base Cases:

- 1 step: 1 way
- 2 steps : 2 ways

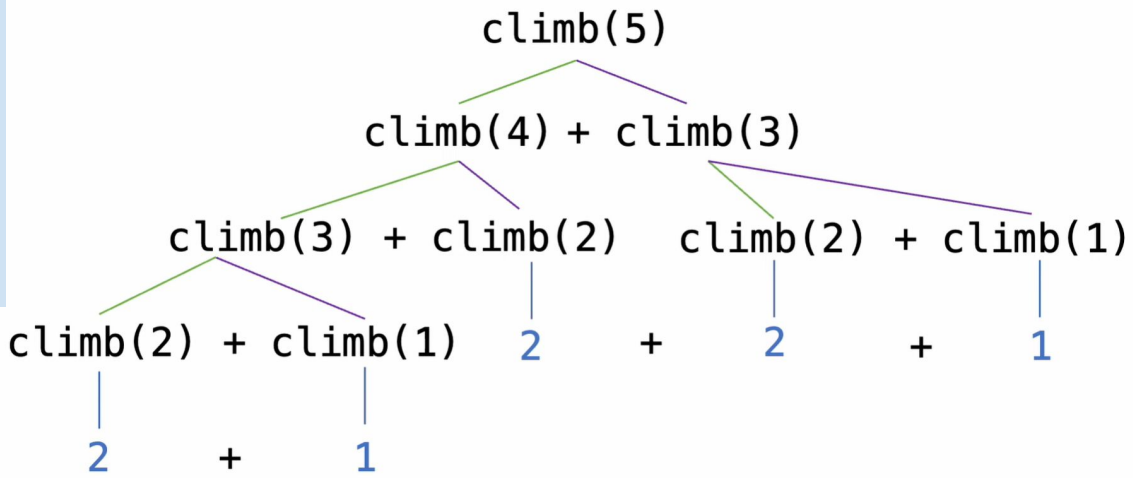


1 Recursion Visualization (Recursive Approach)

Input: n = 5
Output: 8

Time Complexity: O(2^n)

(Tree of n-1 level with 2 branches)



Answer: 8 ways

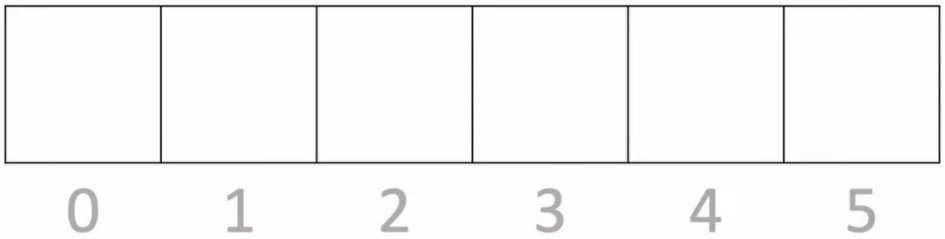
1 Code: (Recursive Approach)

```python
# Recursive Approach: Time Limit Exceeded
def climbStairs(self, n: int) -> int:
    if n == 1:
        return 1
    if n == 2:
        return 2
    return self.climbStairs(n - 1) + self.climbStairs(n - 2)
```

2 Eliminating recursion & saving our prev. results

(Bottom-up Dynamic programming approach Approach)

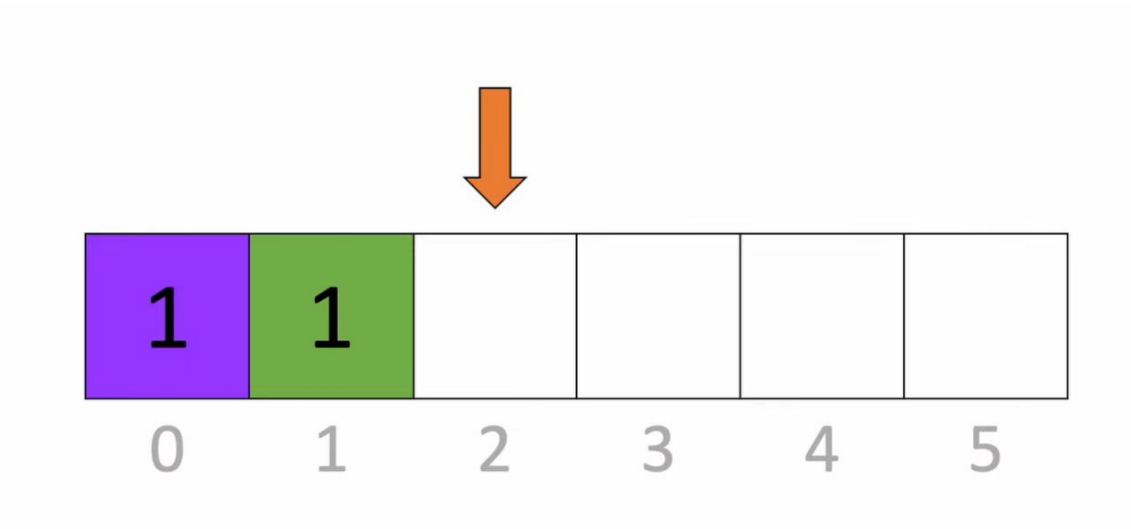
Step1. Initialize an array and fill in our results



2 Eliminating recursion & saving our prev. results

(Bottom-up Dynamic programming approach Approach)

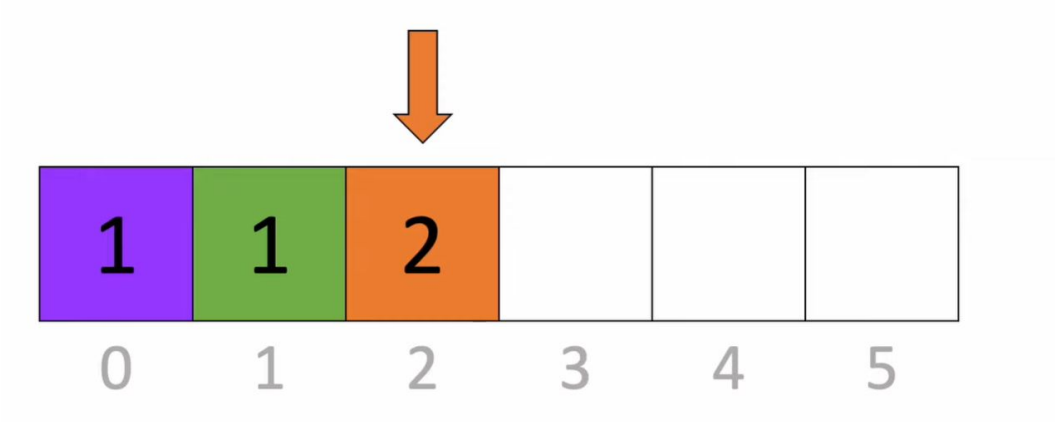
Step2. Fill in the base case

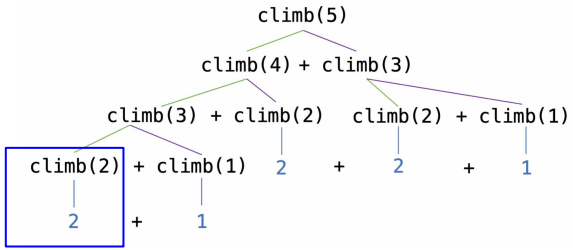


2 Eliminating recursion & saving our prev. results

(Bottom-up Dynamic programming approach Approach)

Step3. Fill in the array (each value of the position represents # of ways)

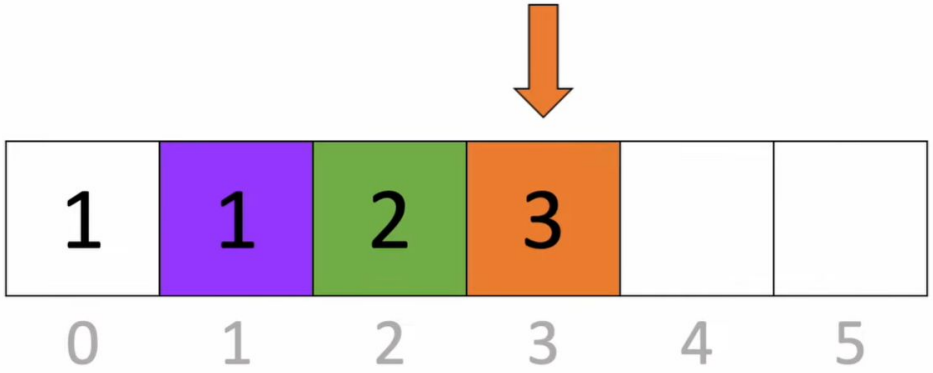


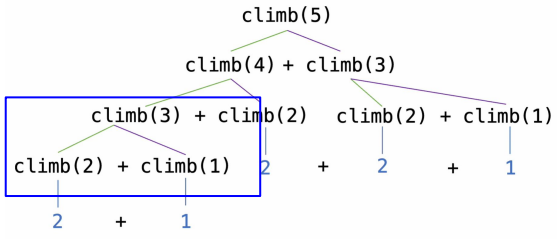


2 Eliminating recursion & saving our prev. results

(Bottom-up Dynamic programming approach Approach)

Step3. Fill in the array (each value of the position represents # of ways)

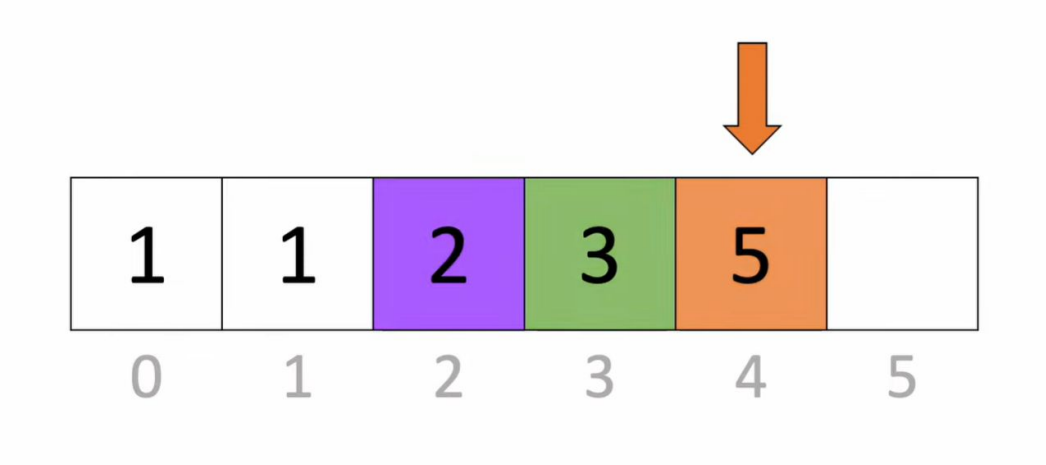


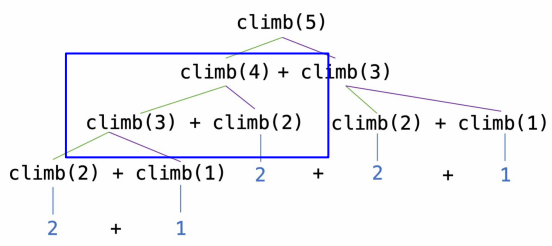


2 Eliminating recursion & saving our prev. results

(Bottom-up Dynamic programming approach Approach)

Step3. Fill in the array (each value of the position represents # of ways)

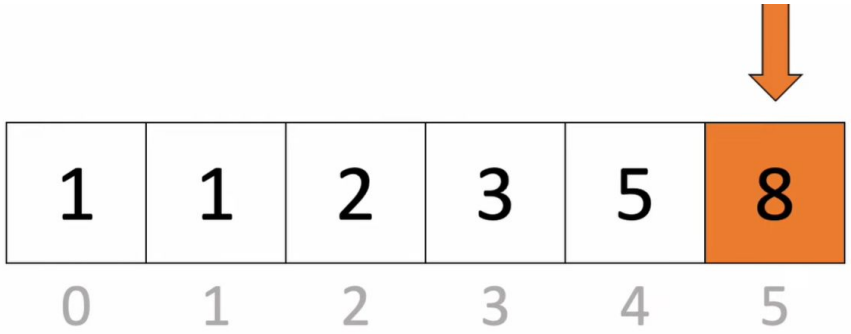




2 Eliminating recursion & saving our prev. results

(Bottom-up Dynamic programming approach Approach)

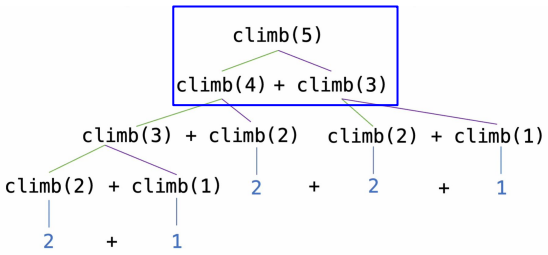
Step3. Fill in the array (each value of the position represents # of ways)



Time Complexity: O(n) Time

- We are computing each steps only once
-

Memory Complexity: O(n)

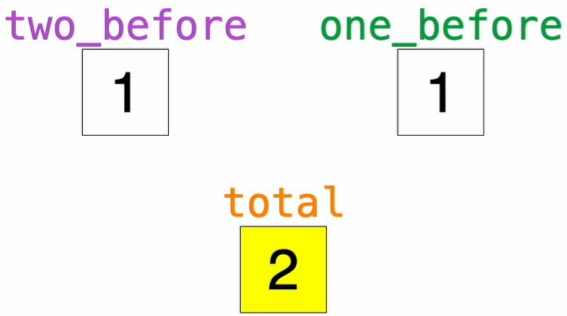


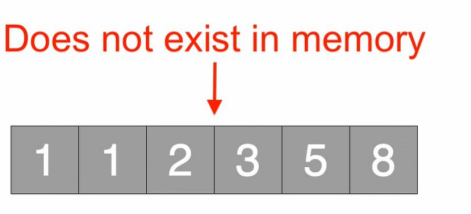
2 Code: (Bottom-up Dynamic programming approach)

```python
def climbStairs_dynamic_array(self, n: int) -> int:
    if n == 1:
        return 1
    if n == 2:
        return 2

    # Initialize an array to store the number of ways to reach each step
    dp = [0] * (n + 1)
    dp[1] = 1  # 1 way to reach the first step
    dp[2] = 2  # 2 ways to reach the second step

    # Fill the array iteratively
    for i in range(3, n + 1):
        dp[i] = dp[i - 1] + dp[i - 2]

    return dp[n]
```

3 Using two variables instead of the array

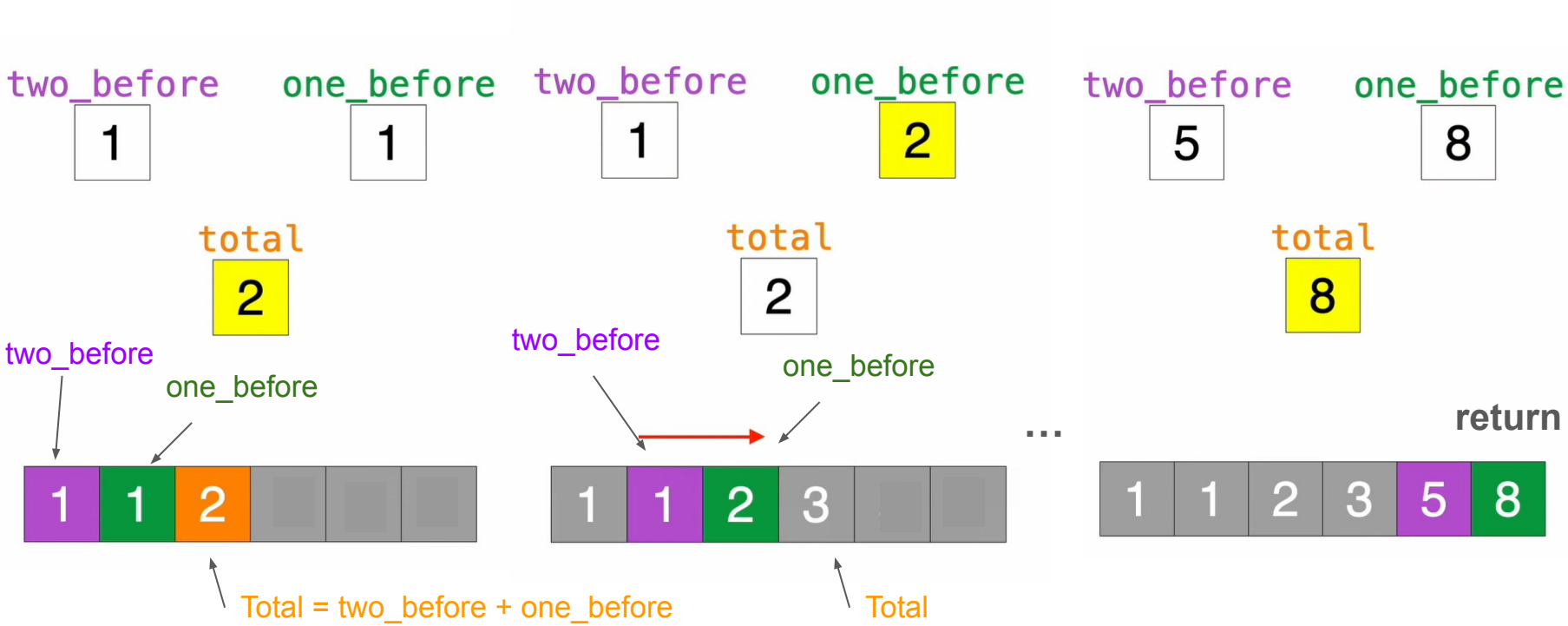
(Bottom-up Dynamic programming approach Approach without using array)





3 Using two variables instead of the array

(Bottom-up Dynamic programming approach Approach without using array)



3 Code: (Bottom-up Dynamic programming approach without array)

```python
def climbStairs(self, n: int) -> int:
    if n == 1:
        return 1
    if n == 2:
        return 2

    # Initialize the variables
    two_before = 1  # Number of ways to reach the step before the first step
    one_before = 2  # Number of ways to reach the first step

    # Iterate from the third step to the nth step
    for i in range(3, n + 1):
        total = one_before + two_before  # The total ways to reach the current step
        two_before = one_before          # Update for next iteration
        one_before = total               # Update for next iteration

    return one_before  # Return the number of ways to reach the nth step
```

Time Complexity: O(n) Time
Memory Complexity: O(1)

감사합니다!

THANK YOU