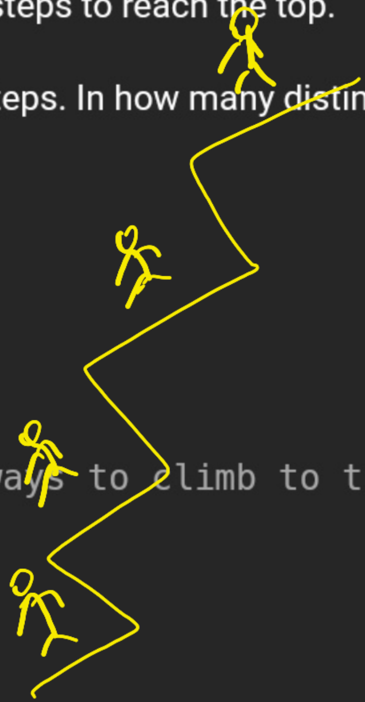
70. Climbing Stairs

Easy · Topics · Companies · Hint

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

dp[i] states no of ways you can reach to top from i^{th} step

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

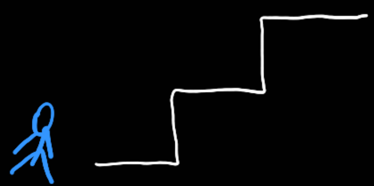
Constraints:

- `1 <= n <= 45`

$dp[n] = dp[n-1] = 1$

Approach 1: Recursion

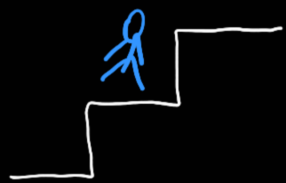
$n = 3$



now we can make 1 or 2 steps
lets say we made 1 step

now no.of ways you can climb 3 steps is equal to no.of ways you can climb 2 steps and also

lets say we made 2 steps



now no.of ways you can climb 3 steps is equal to no.of ways you can climb 1 step.

So

no.of ways you can climb n steps = no.of ways you can climb (n-1) steps + no.of ways you can climb (n-2) steps

This problem is basically a "Fibonacci problem"

```
f(n)
{
    if (n ≤ 1)
        return 1
    
    return f(n-1) + f(n-2)
}
```

note that $f_0 = 1$ and not 0.

$T(n) : O(2^n)$
$S(n) : O(n)$

Approach 2: Memoization

we avoid solving overlapping subproblems again and again. we only solve one time and use it in future when encountered.

```
f(n, dp)
{
    if (n <= 1)
        return 1

    if ( dp[n] != -1 )
        return dp[n]

    return dp[n] = f(n-1) + f(n-2)
}
```

$T(n) : O(n)$

$S(n) : \underbrace{O(n)}_{\text{Recursion stack space}} + \underbrace{O(n)}_{dp[\,]}$

Approach 3: Tabulation

instead of going from given n to base case, we start from base case and go to n.

dp[0] = 1
dp[1] = 1

```
for (i : 2 to n)
    dp[i] = dp[i-1] + dp[i-2]
```

$T(n) : O(n)$
$S(n) : O(n)$

Approach 4: Space Optimization of Tabulation method

If we observe carefully, we can see that at any index i, we only need previous two values. So we don't need a dp array we can just carry two previous values through each iteration.

```
prev1 = 1
prev2 = 2

for (i : 2 to n)
{   curr = prev1 + prev2
    prev1 = prev2
    prev2 = curr
}
```

$T(n) : O(n)$
$S(n) : O(1)$