Climbing Stairs

Wenbo

Last time's Follow up

What if the power is not an integer? I.e. 3^8.5

Follow up

What if the power is not an integer? I.e. 3^8.5

8.5 = 8 + 0.5

So 3^8.5 = 3^8 * 3^(0.5)

Then apply last time's algorithm to 3^8

Climbing Stairs

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
```

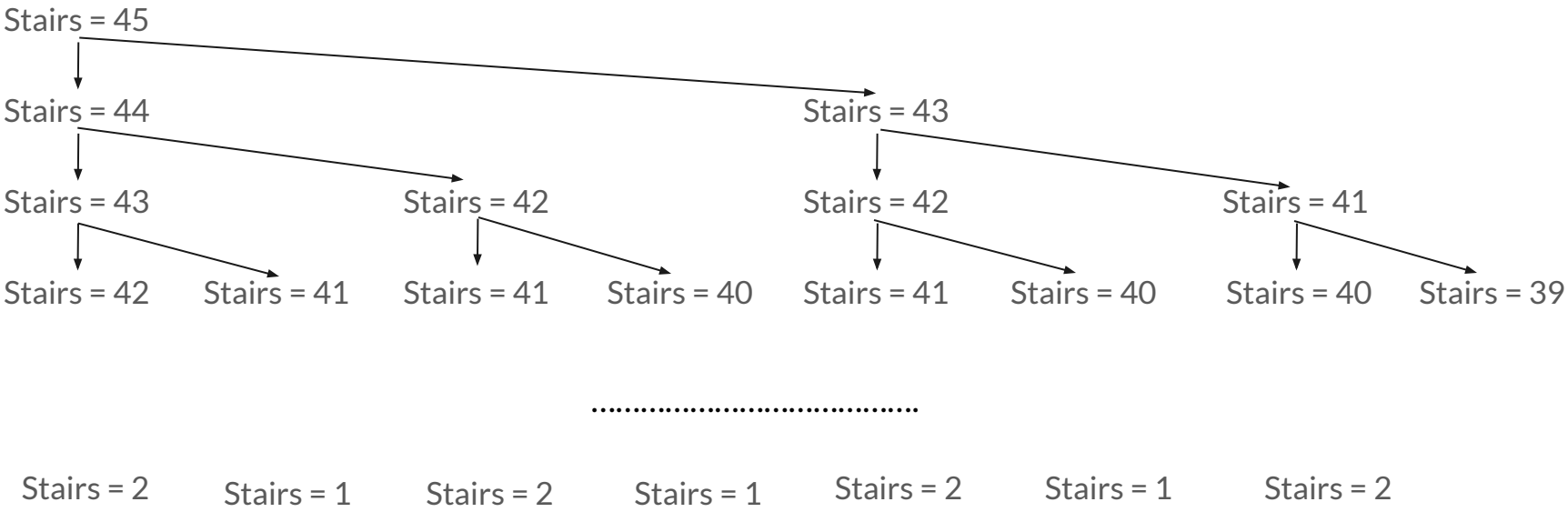
Example 2:

```
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
```

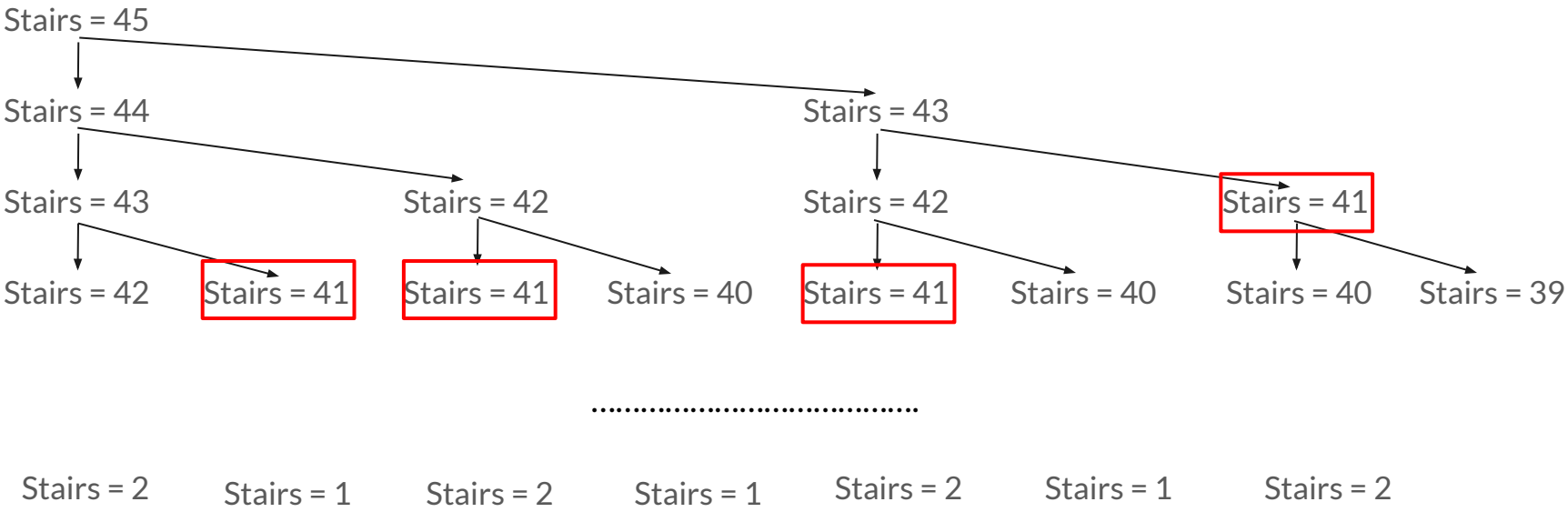
Constraints:

- `1 <= n <= 45`

Approach 1 - Recursive



Approach 1 - Recursive



Approach 2 - Dynamic Programing

Reuse intermediate results as much as possible(i.e. Avoid repeated work as much as possible)

Approach 2 - Dynamic Programing

Stairs = 1, result = 1, save it as **dp[1]** = 1
Stairs = 2, result = 2, save it as **dp[2]** = 2
Then:
Stairs = 3, result or dp[3] = **dp[2]** + **dp[1]** = 3
Stairs = 4, result or dp[4] = **dp[3]** + **dp[2]** = 5
Stairs = 5, result or dp[5] = **dp[4]** + **dp[3]**
....
Stairs = n, result or dp[n] = **dp[n-1]** + **dp[n-2]**

Return dp[n] as final result

Follow up

What if the you can step 1, 2 or 3 steps at one time?

I.e. for stairs = 3, there are 3 ways:

1. 1,1,1
2. 1,2
3. 2,1
4. 3

Next Week: Min Cost Climbing Stairs

You are given an integer array `cost` where `cost[i]` is the cost of i^{th} step on a staircase. Once you pay the cost, you can either climb one or two steps.

You can either start from the step with index `0`, or the step with index `1`.

Return *the minimum cost to reach the top of the floor.*

Example 1:

```
Input: cost = [10,15,20]
Output: 15
Explanation: You will start at index 1.
- Pay 15 and climb two steps to reach the top.
The total cost is 15.
```

Example 2:

```
Input: cost = [1,100,1,1,1,100,1,1,100,1]
Output: 6
Explanation: You will start at index 0.
- Pay 1 and climb two steps to reach index 2.
- Pay 1 and climb two steps to reach index 4.
- Pay 1 and climb two steps to reach index 6.
- Pay 1 and climb one step to reach index 7.
- Pay 1 and climb two steps to reach index 9.
- Pay 1 and climb one step to reach the top.
The total cost is 6.
```

Constraints:

- `2 <= cost.length <= 1000`
- `0 <= cost[i] <= 999`