Newton School of Technology

Dynamic Programming

by Gladden Rumao

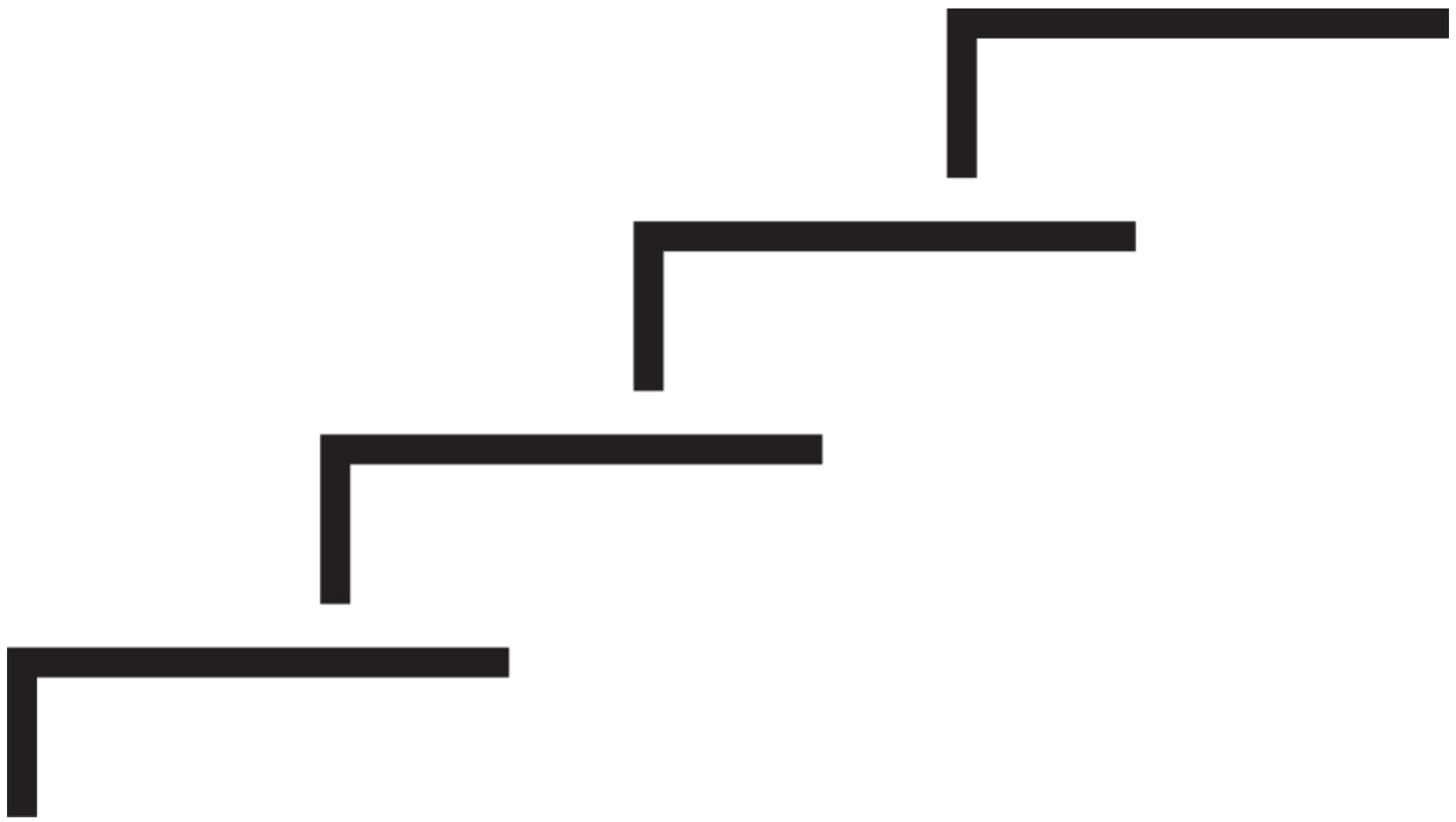
Data Structures and Algorithms

Lecture Agenda :

- What is DP ?
- Why DP ?
- DP vs Recursion
- Memoization

Recap - Climbing Stairs :

Given 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?

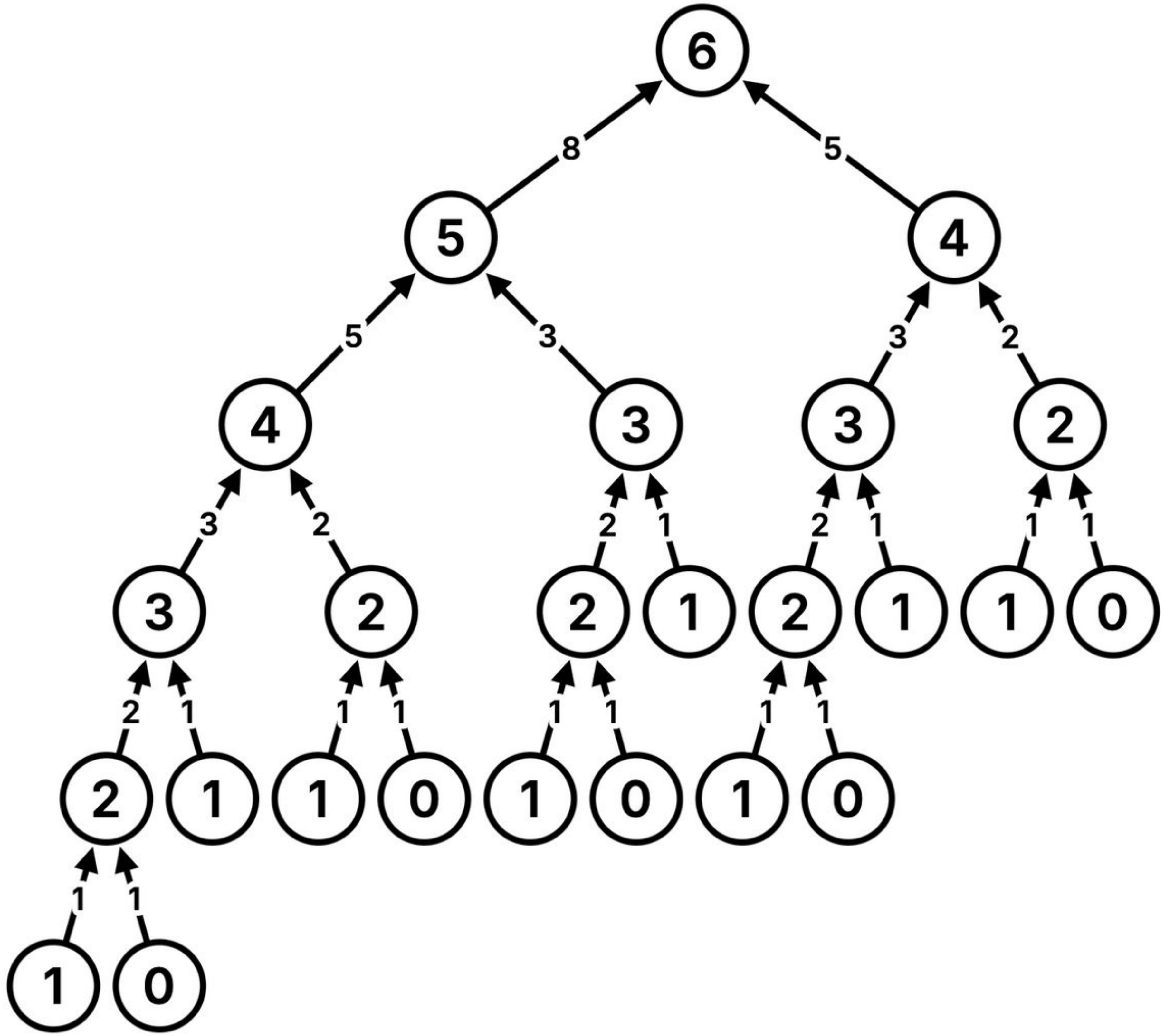


Code

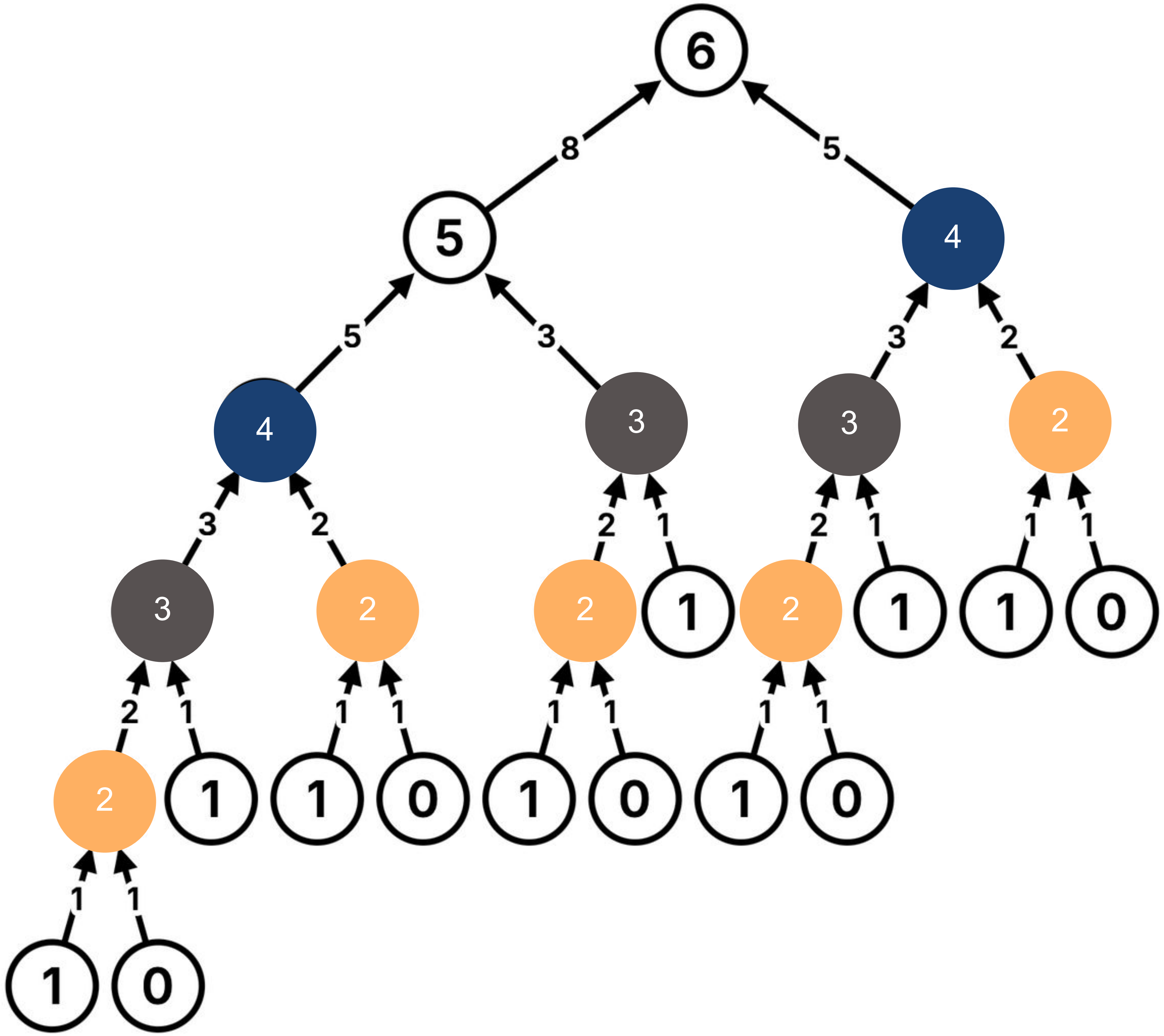
```
def climbStairs(n):
    if n == 0 or n == 1:
        return 1
    return climbStairs(n-1) + climbStairs(n-2)
```

Recursion Tree, n=6

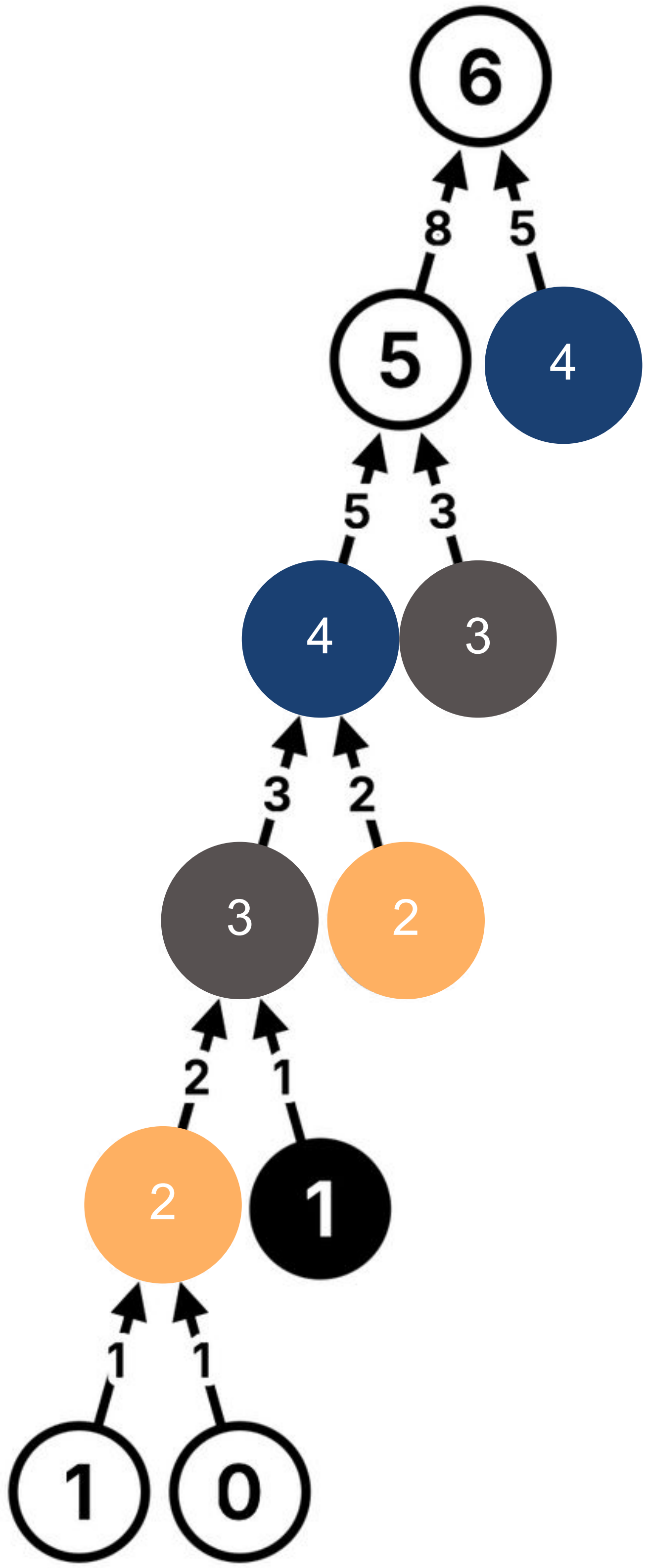
Newton School of Technology



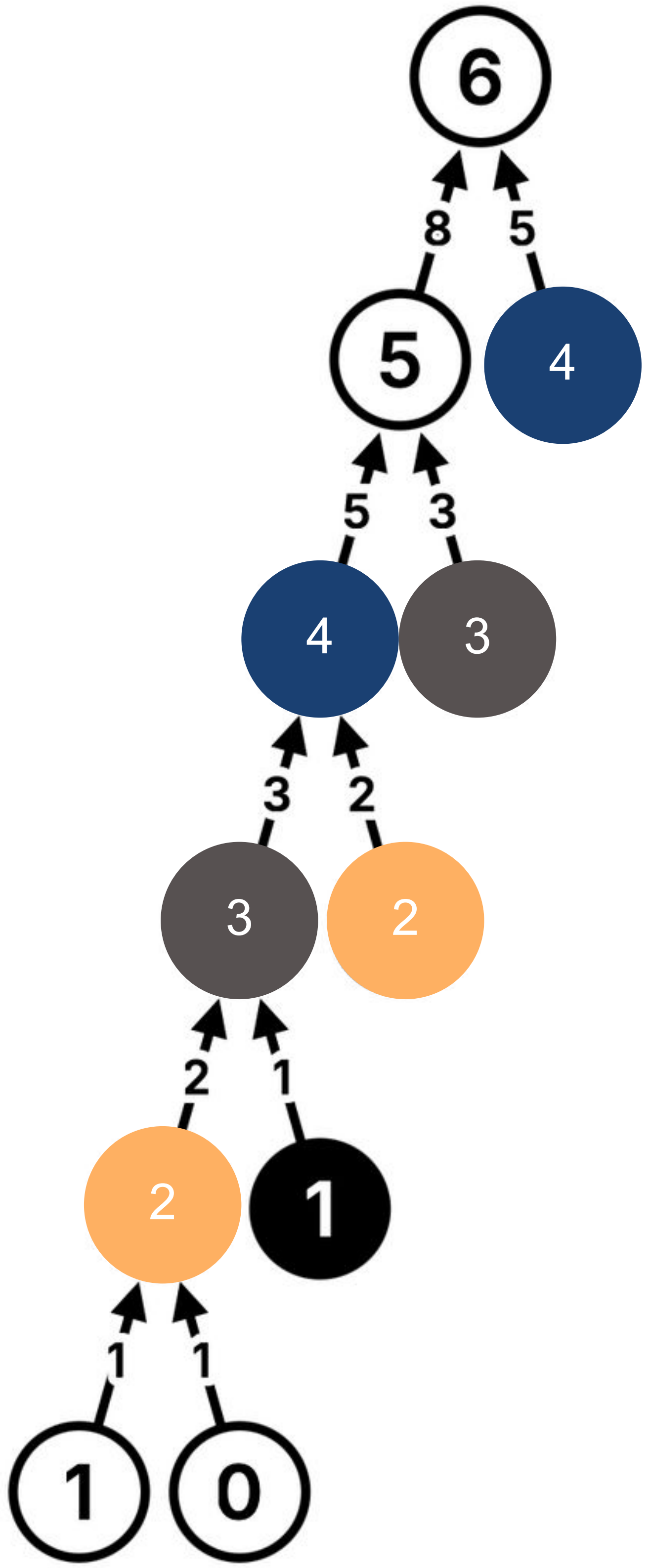
Overlapping subproblems :



Solution :



Solution:



- We store the answer of a subproblem.
- Whenever we need the answer of that subproblem, instead of calculating it again we use the stored value.


Dynamic programming

Dynamic Programming :

Newton School of Technology

- Optimization over plain recursion.
- If a recursive solution makes repeated calls for the same inputs, we optimize it using Dynamic Programming.
- By storing subproblem results, we avoid redundant computations, often reducing time complexity from exponential to polynomial.

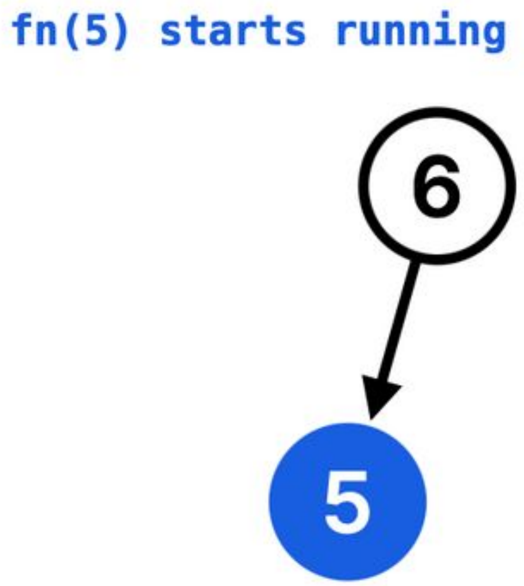
Solution:

`fn(6) starts running`

0	1	2	3	4	5	6
-1	-1	-1	-1	-1	-1	-1

- Check if we have computed ans of n=6.
- No

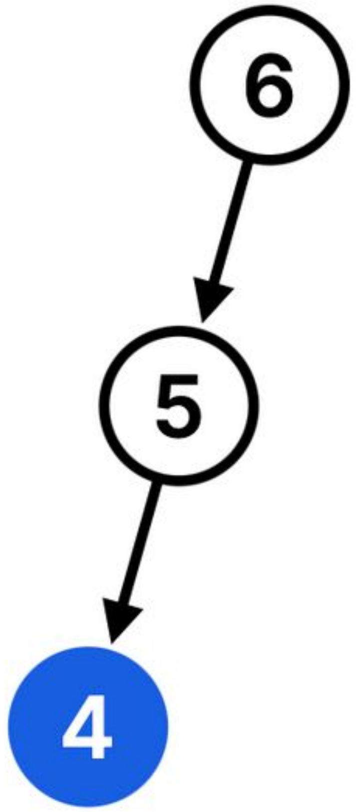
Solution:



0	1	2	3	4	5	6
-1	-1	-1	-1	-1	-1	-1

- Check if we have computed ans of n=5.
- No

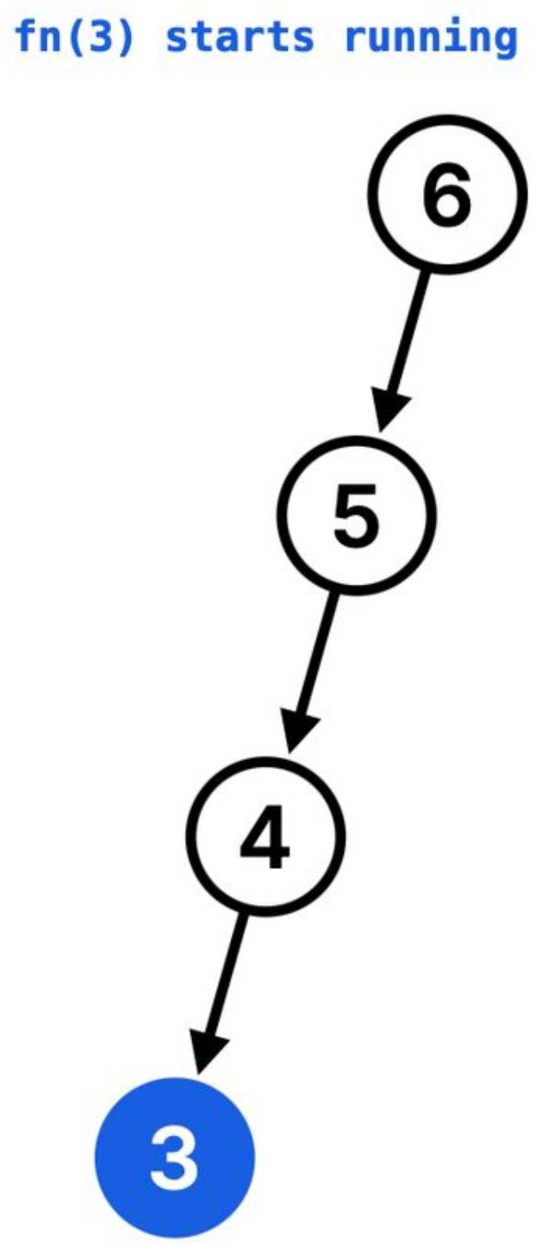
Solution:



0	1	2	3	4	5	6
-1	-1	-1	-1	-1	-1	-1

- Check if we have computed ans of n=4.
- No

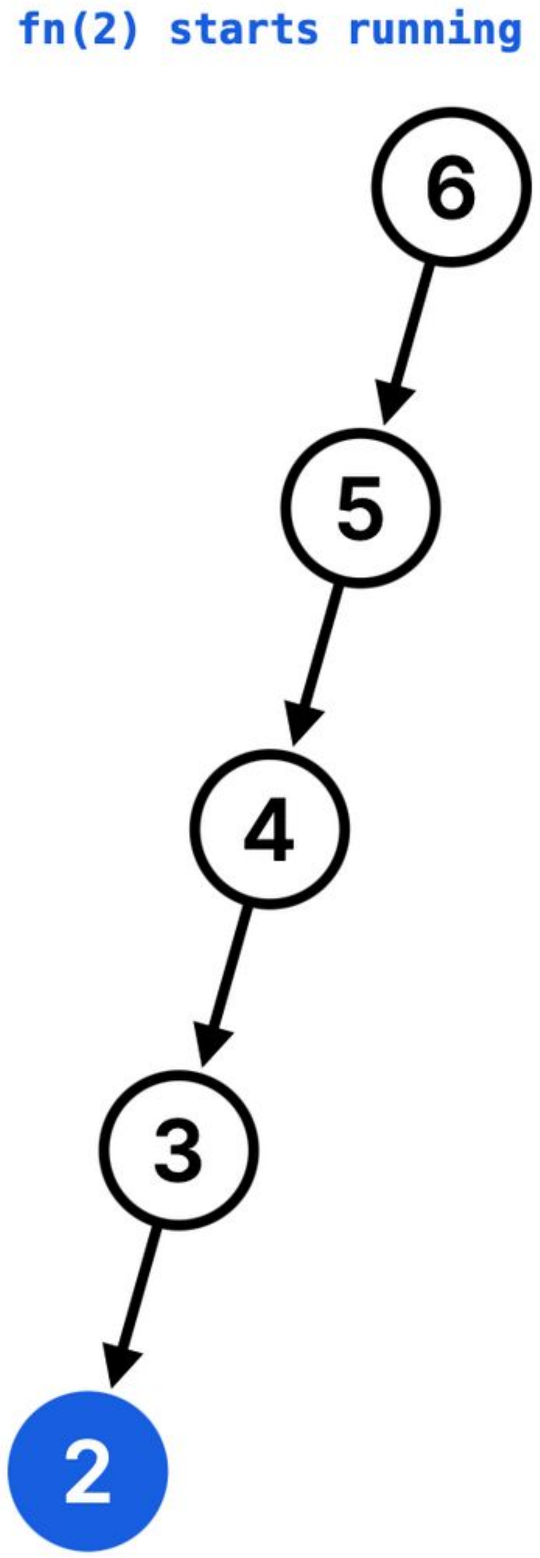
Solution:



0	1	2	3	4	5	6
-1	-1	-1	-1	-1	-1	-1

- Check if we have computed ans of n=3.
- No

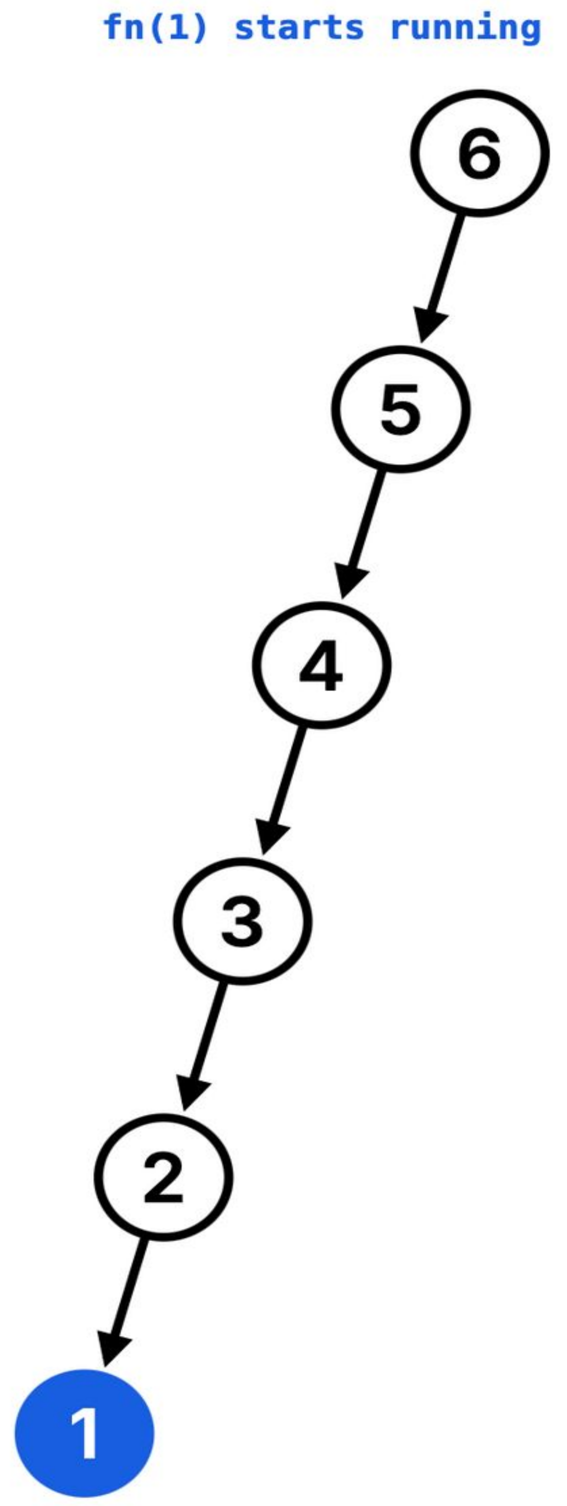
Solution:



0	1	2	3	4	5	6
-1	-1	-1	-1	-1	-1	-1

- Check if we have computed ans of n=2.
- No

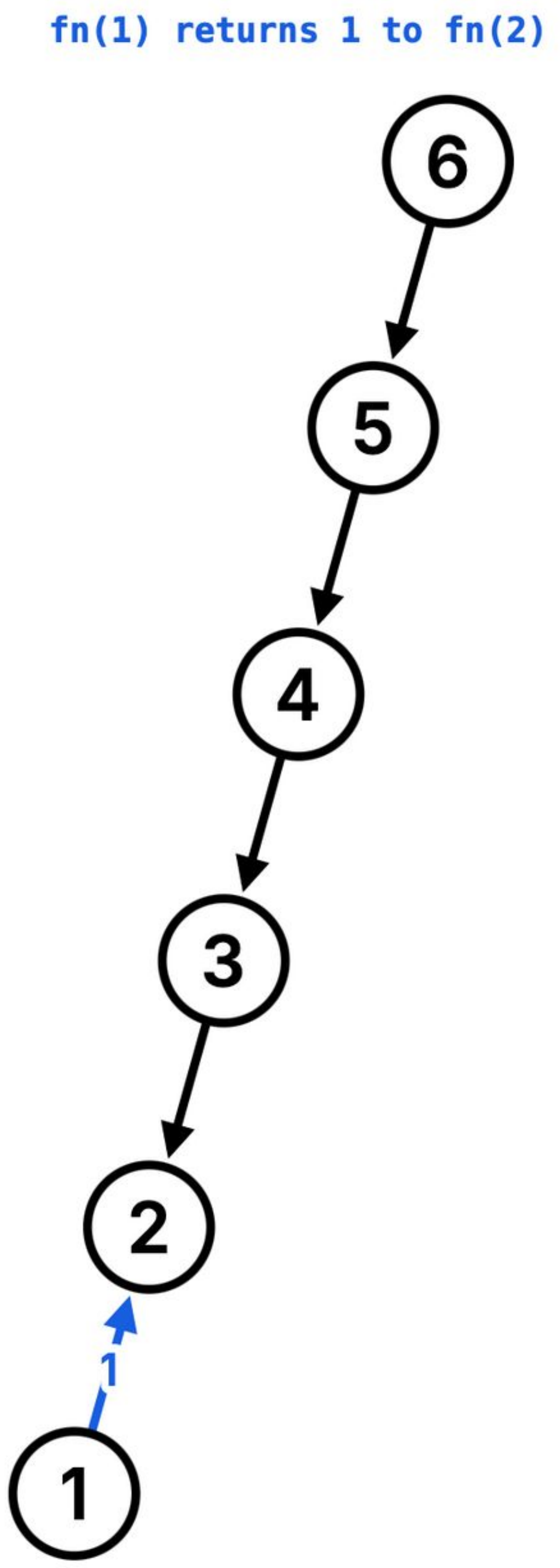
Solution:



0	1	2	3	4	5	6
-1	-1	-1	-1	-1	-1	-1

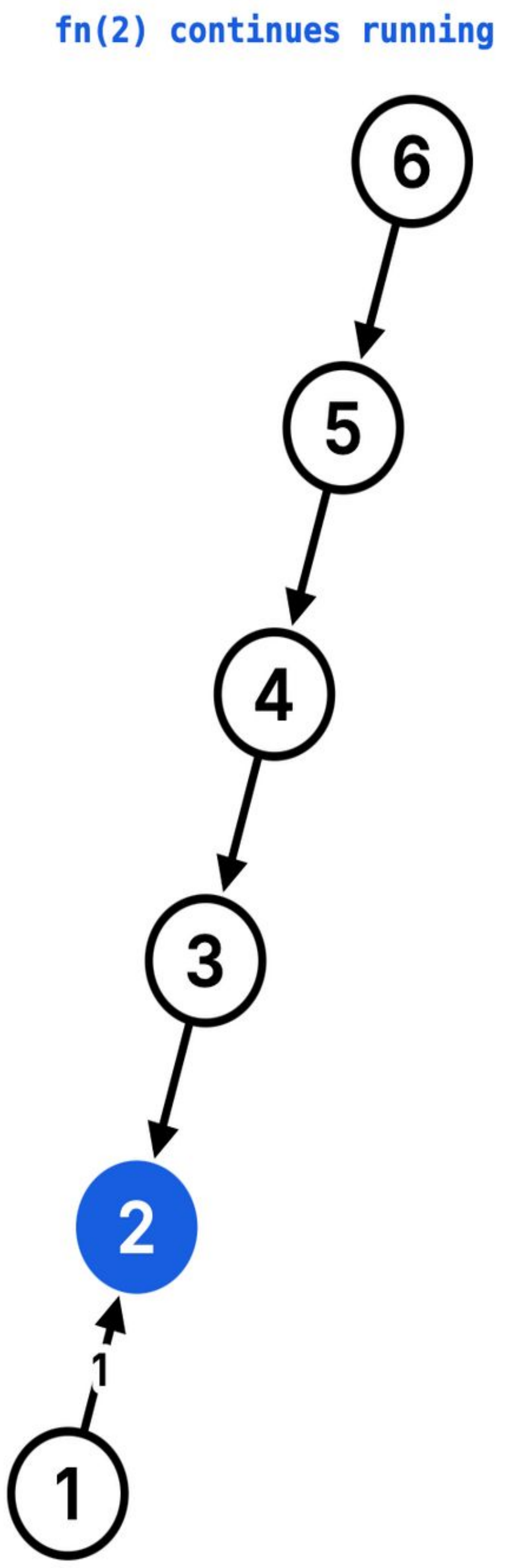
- Check if we have computed ans of n=1.
- Base case (return 1)

Solution:



0	1	2	3	4	5	6
-1	-1	-1	-1	-1	-1	-1

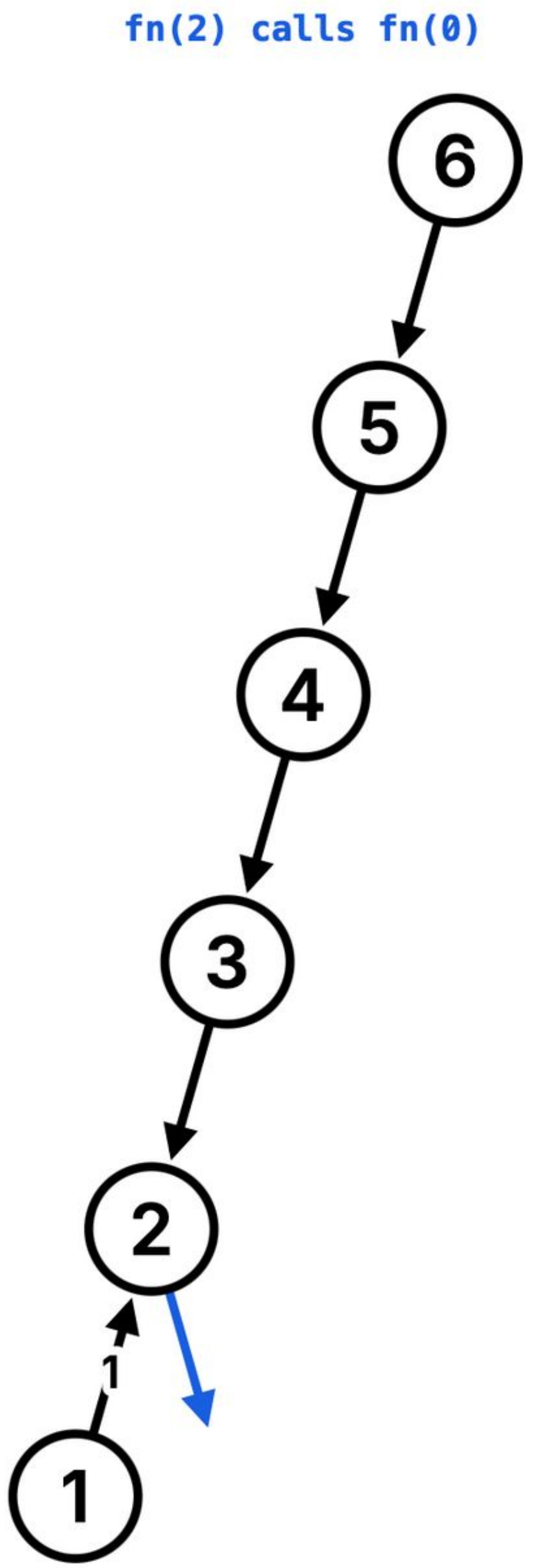
Solution:



0	1	2	3	4	5	6
-1	-1	-1	-1	-1	-1	-1

- Check if we have computed ans of n=2.
- arr[2] != -1

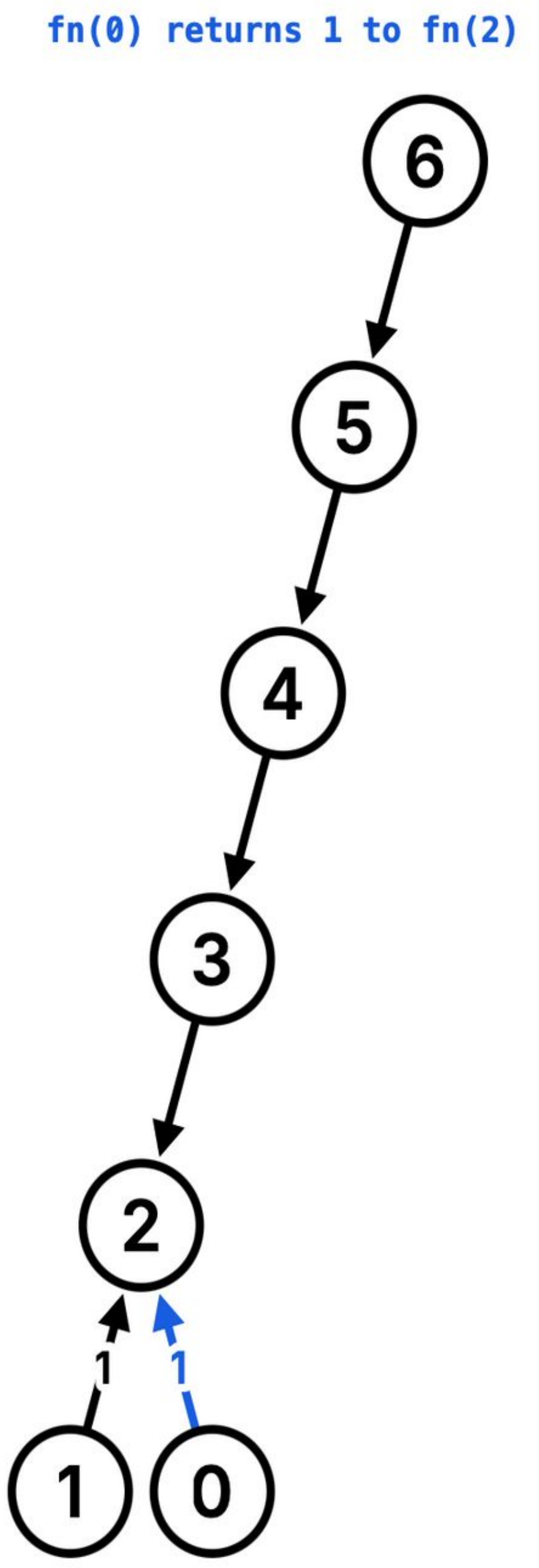
Solution:



0	1	2	3	4	5	6
-1	-1	-1	-1	-1	-1	-1

- Check if we have computed ans of n=0.
- arr[0] != -1

Solution:

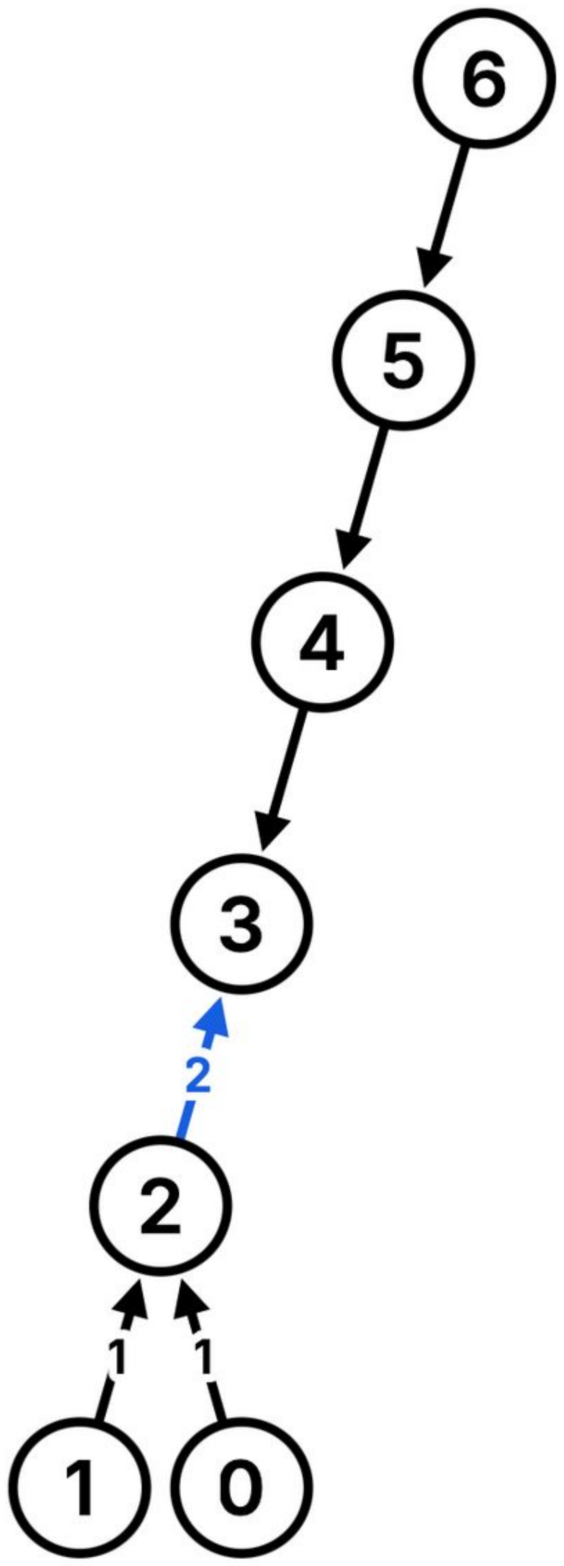


0	1	2	3	4	5	6
-1	-1	-1	-1	-1	-1	-1

- Check if we have computed ans of n=0.
- arr[0] != -1

Solution:

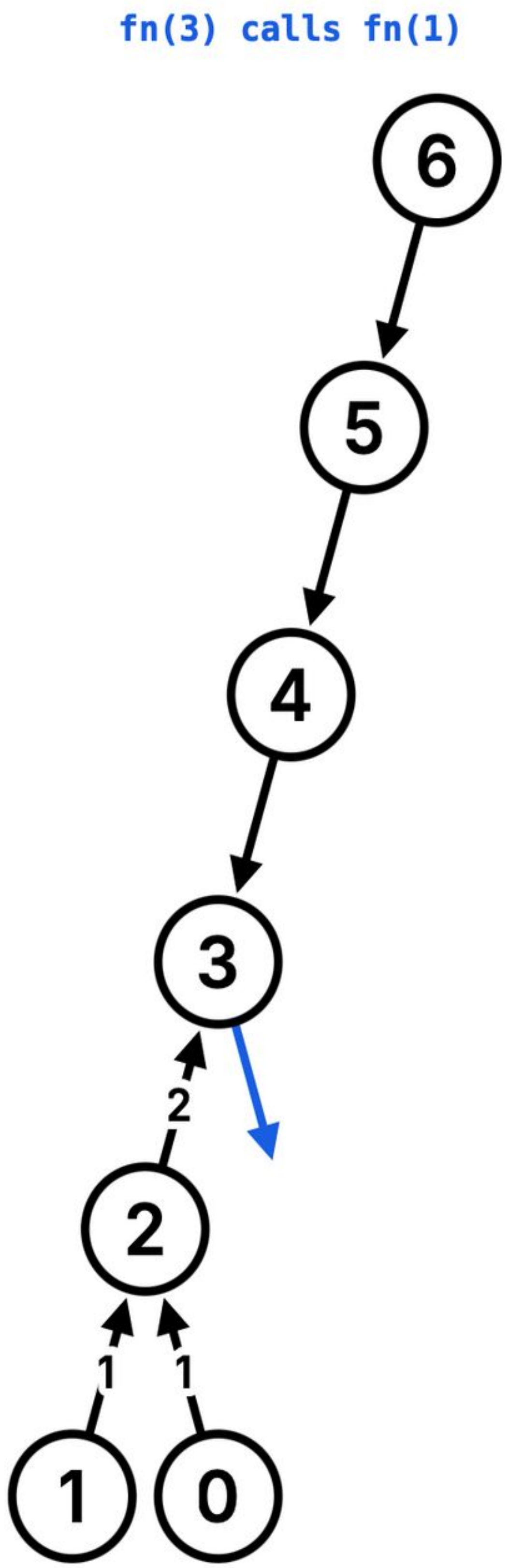
fn(2) returns 2 to fn(3)



0	1	2	3	4	5	6
-1	-1	2	-1	-1	-1	-1

- While returning store the ans in Array.

Solution:

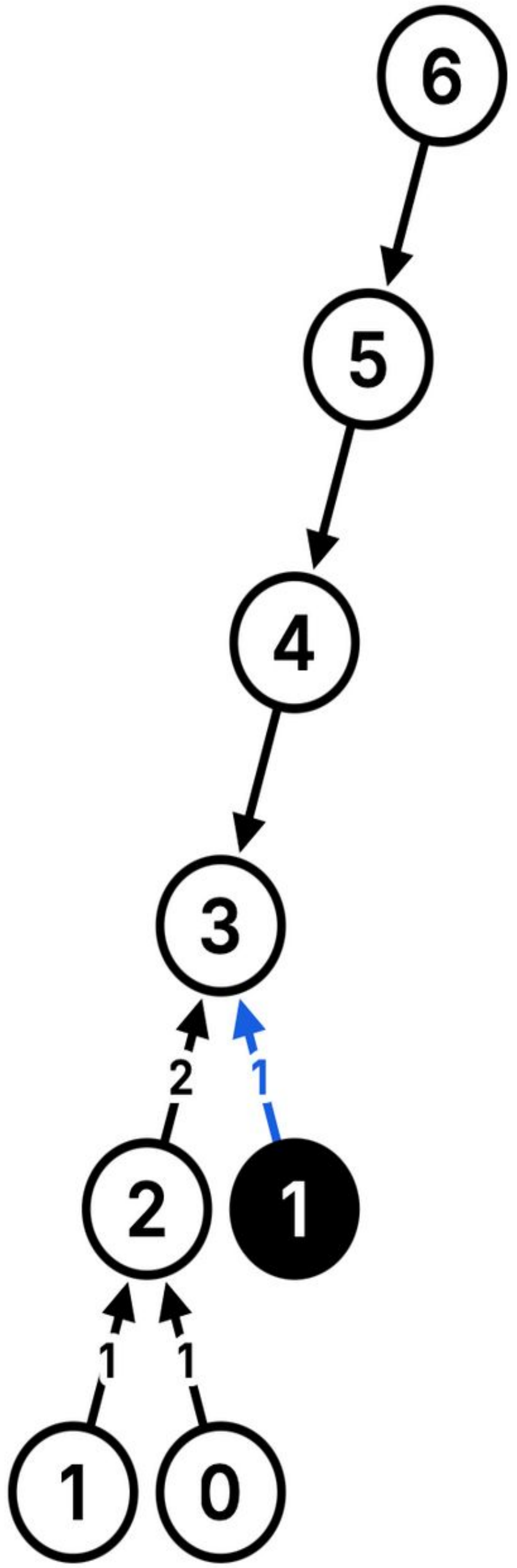


0	1	2	3	4	5	6
-1	-1	2	-1	-1	-1	-1

- Check if we have computed ans of n=3.
- arr[3] != -1

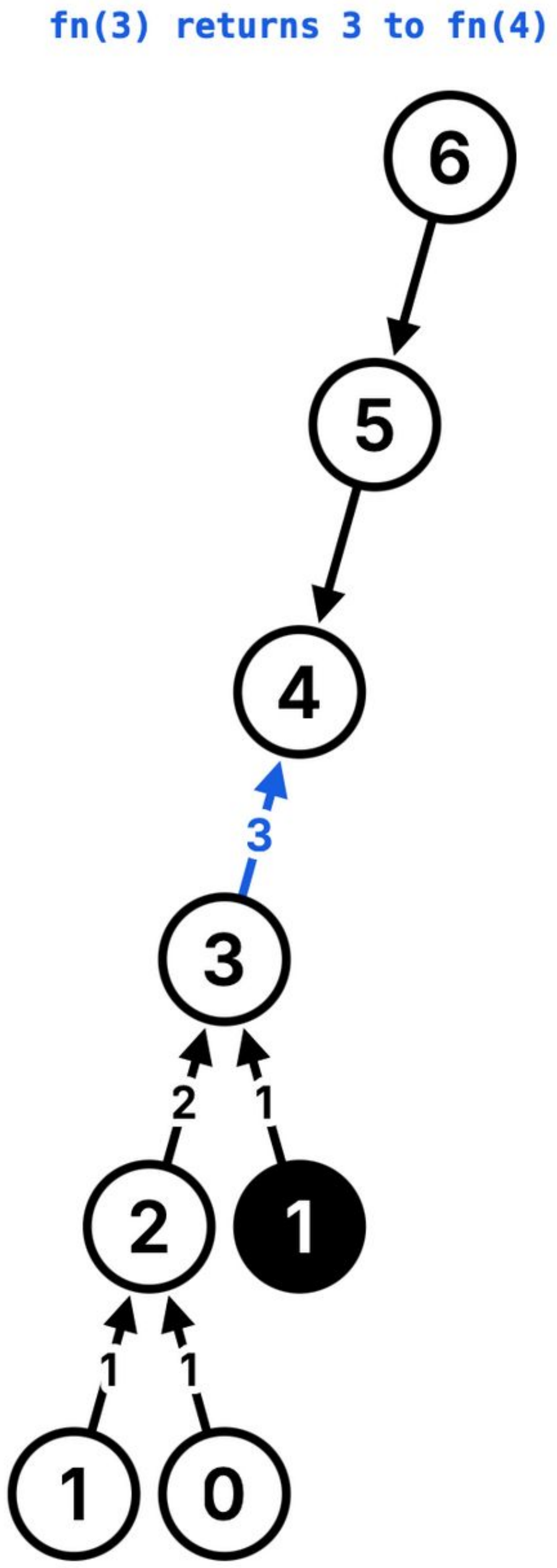
Solution:

fn(1) gets 1 from memory and returns it to fn(3)



0	1	2	3	4	5	6
-1	-1	2	-1	-1	-1	-1

Solution:



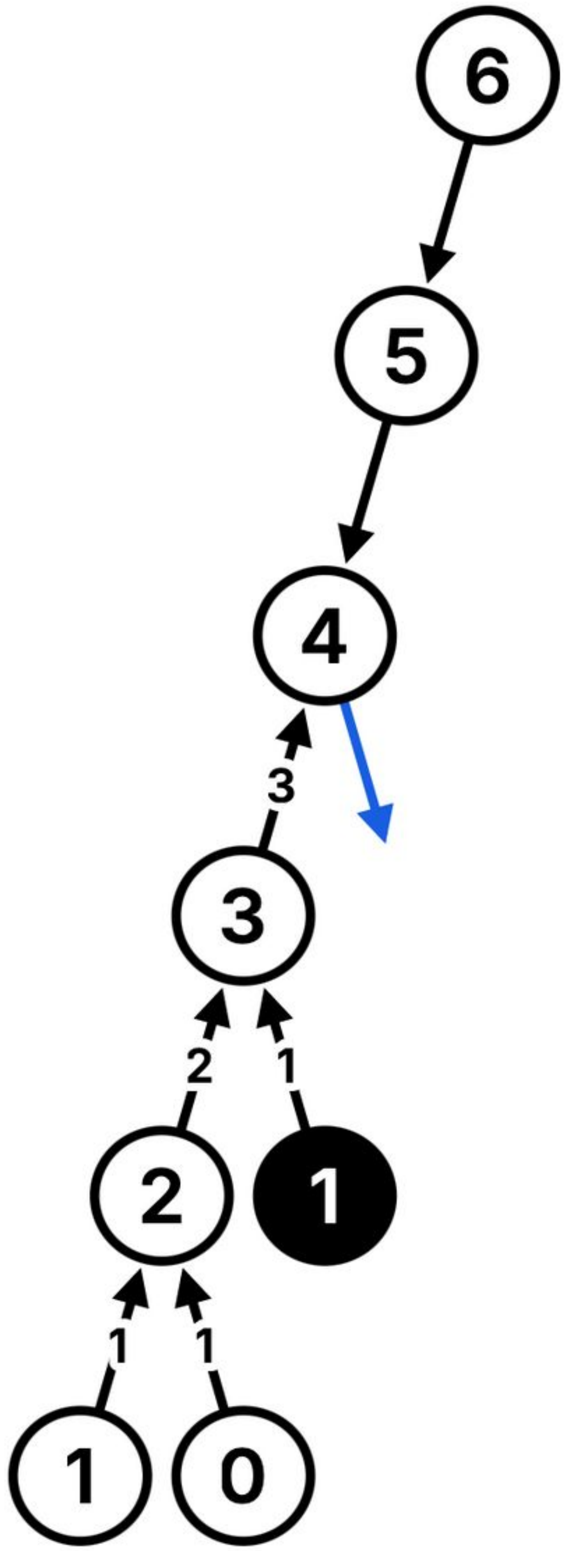
0	1	2	3	4	5	6
-1	-1	2	3	-1	-1	-1

- While returning store the ans in Array.

Solution:

Newton School of Technology

fn(4) calls fn(2)



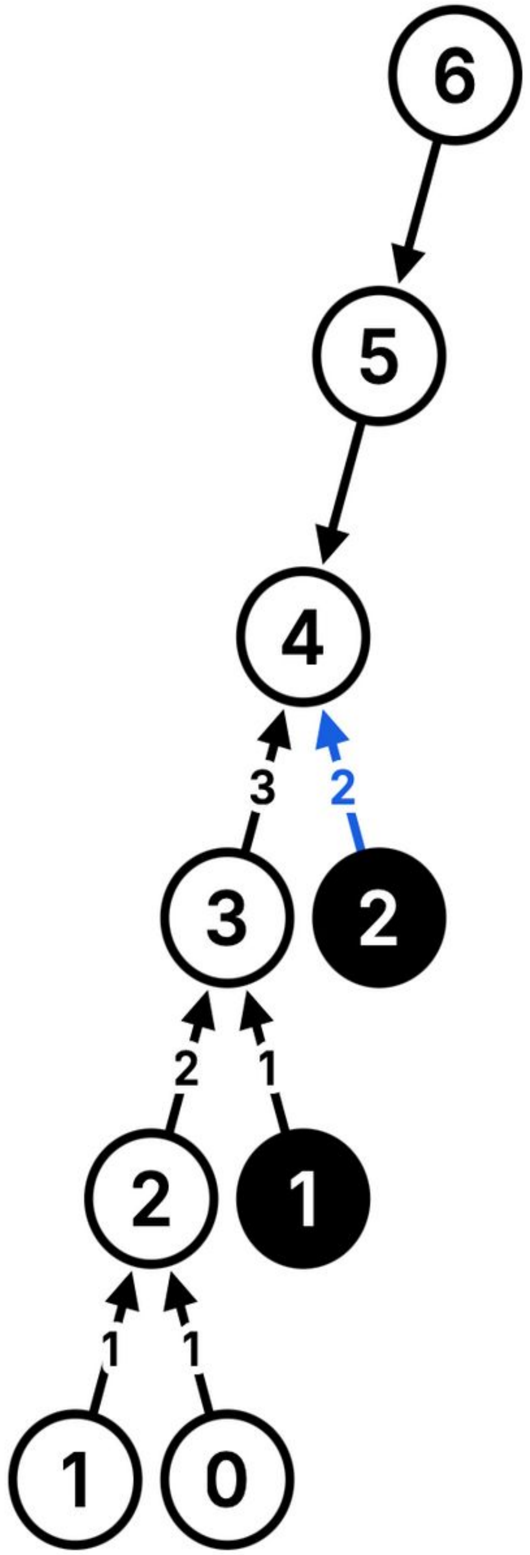
0	1	2	3	4	5	6
-1	-1	2	3	-1	-1	-1

- Check if we have computed ans of n=2.
- arr[2] != -1

Solution:

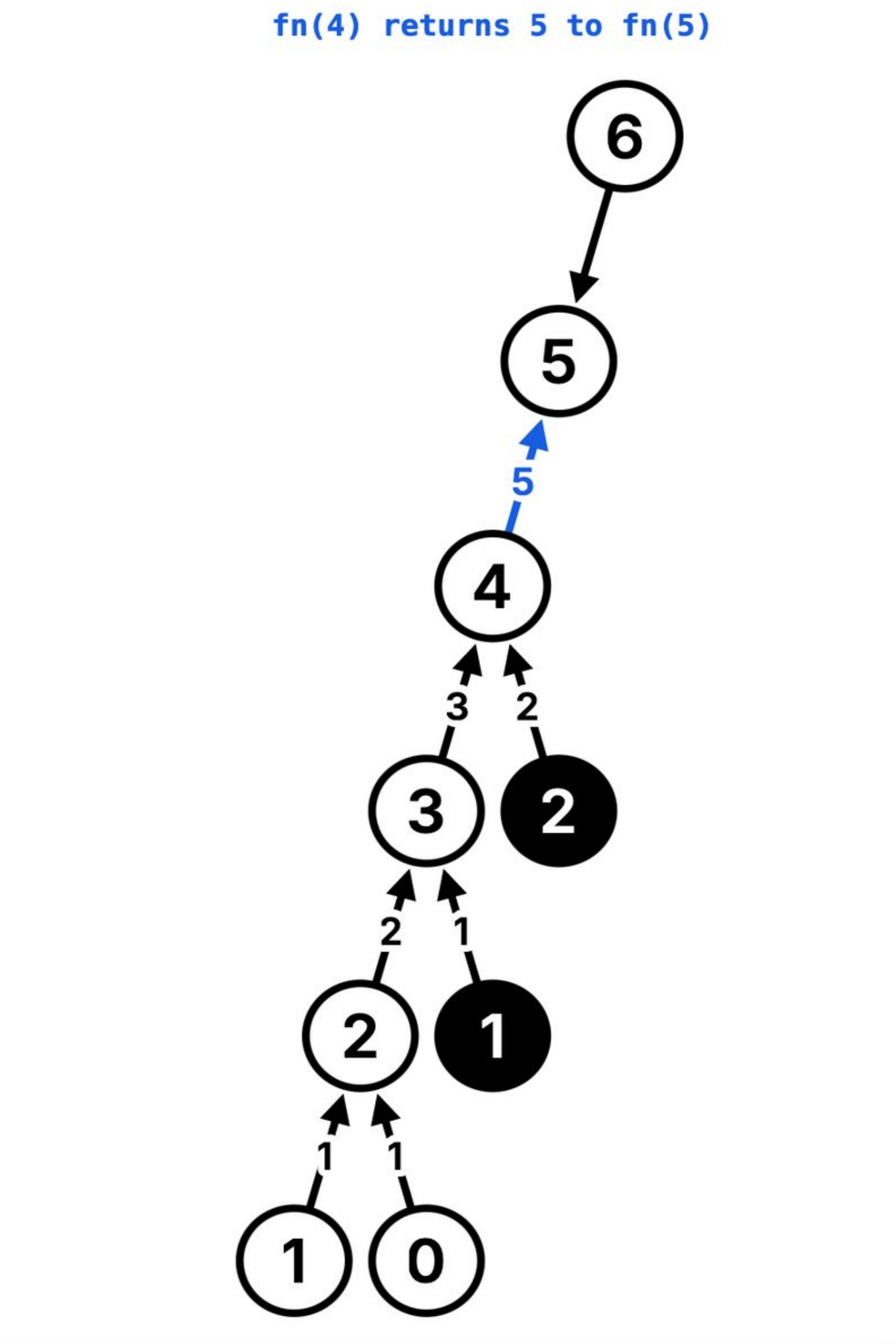
Newton School of Technology

fn(2) gets 2 from memory and returns it to fn(4)



0	1	2	3	4	5	6
-1	-1	2	3	-1	-1	-1

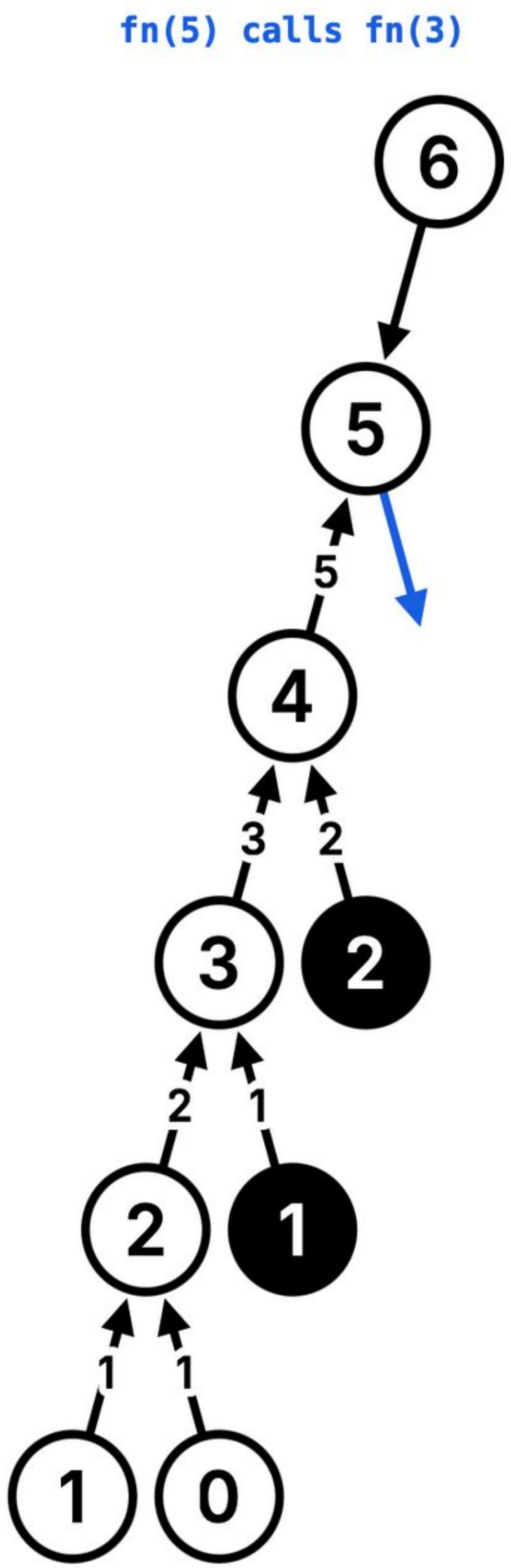
Solution:



0	1	2	3	4	5	6
-1	-1	2	3	5	-1	-1

- While returning store the ans in Array.

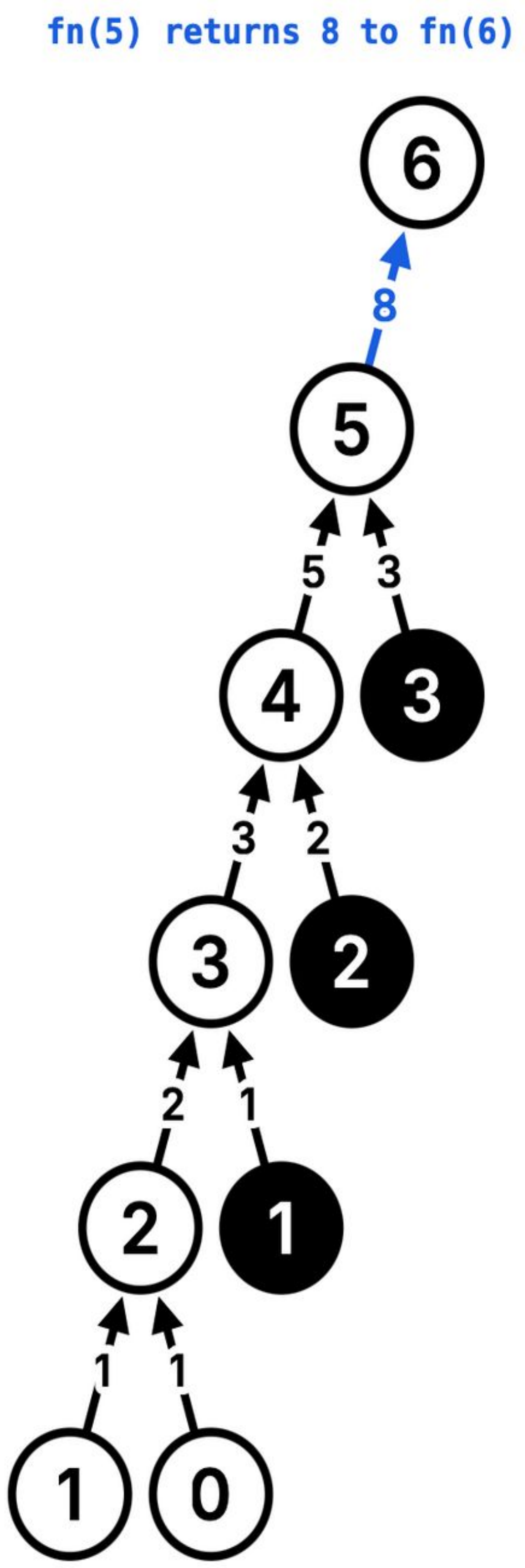
Solution:



0	1	2	3	4	5	6
-1	-1	2	3	5	-1	-1

- Check if we have computed ans of n=3.
- arr[3] != -1

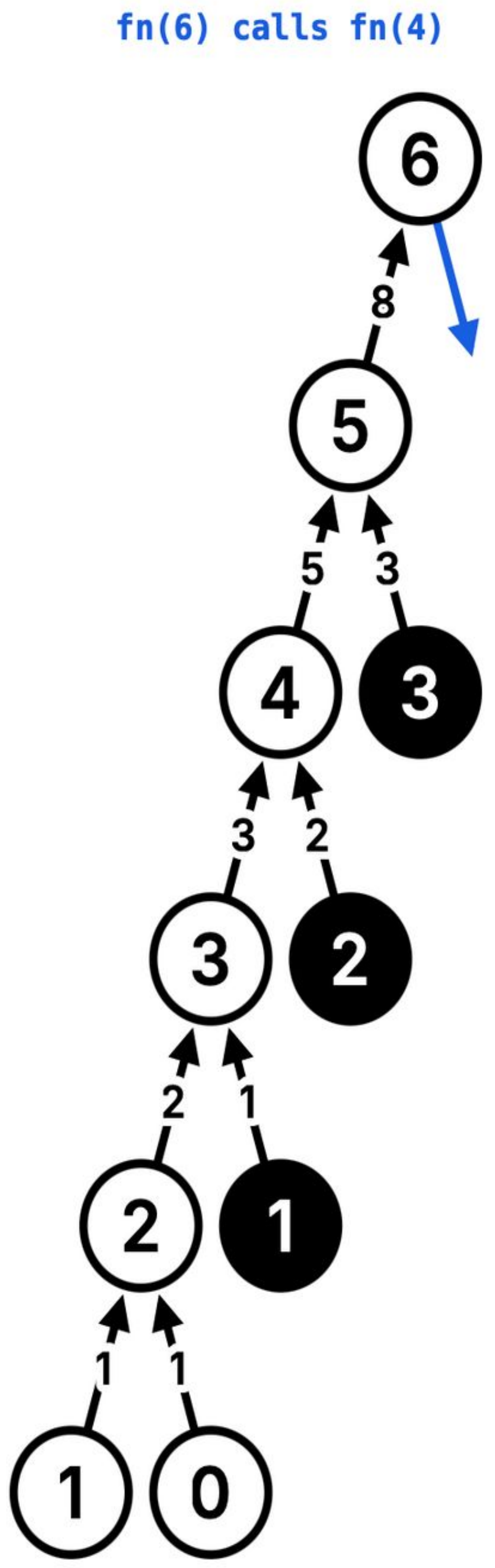
Solution:



0	1	2	3	4	5	6
-1	-1	2	3	5	8	-1

- While returning store the ans in Array.

Solution:

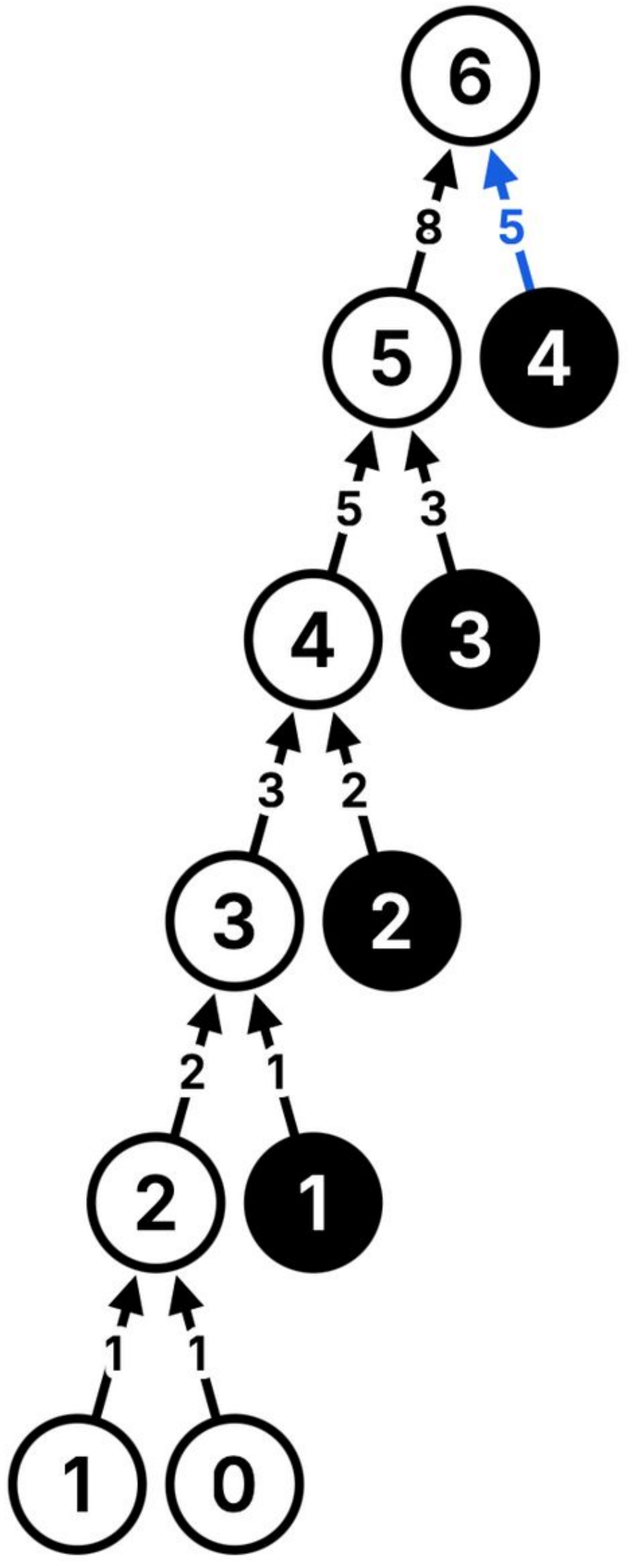


0	1	2	3	4	5	6
-1	-1	2	3	5	8	-1

- Check if we have computed ans of n=4.
- arr[4] != -1

Solution:

fn(4) gets 5 from memory and returns it to fn(6)

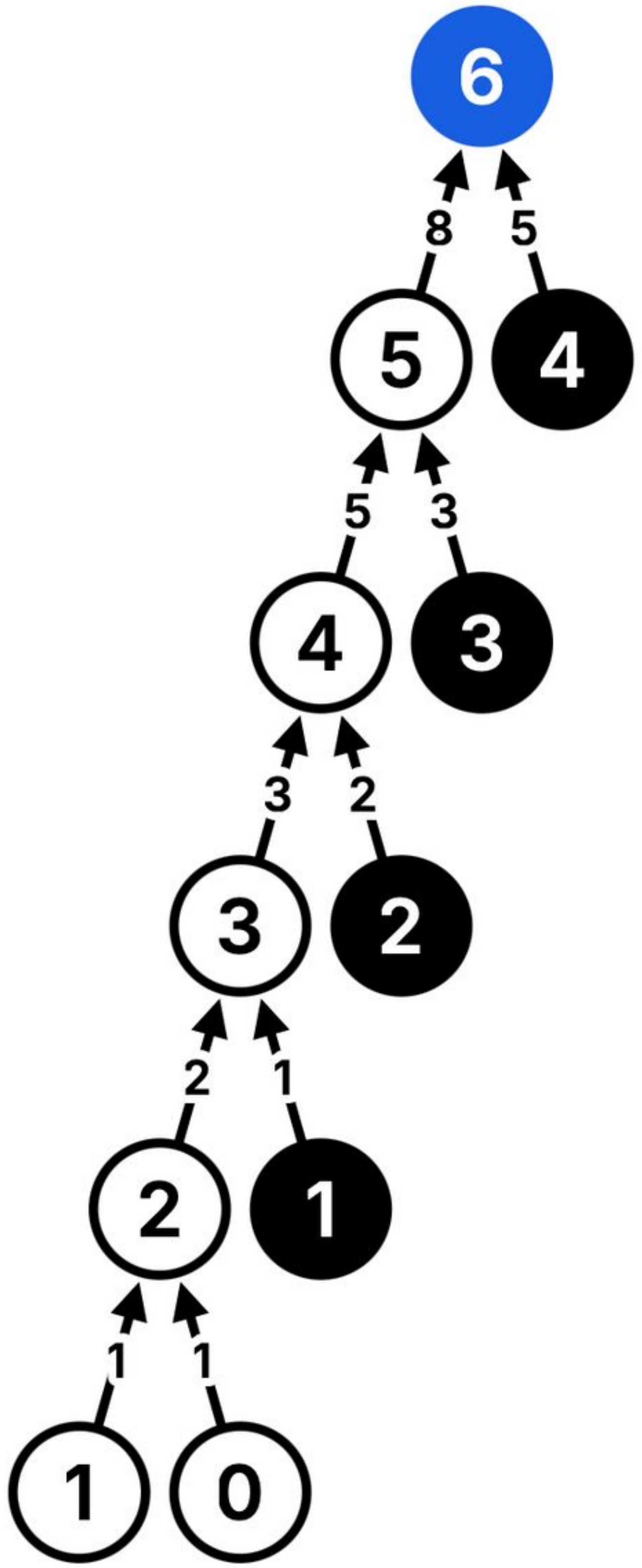


0	1	2	3	4	5	6
-1	-1	2	3	5	8	-1

Solution:

Newton School of Technology

fn(6) continues running



-1	-1	2	3	5	8	13

Solution:

```python
def countWaysRec(n, memo):
    # Base cases
    if n == 0 or n == 1:
        return 1
    # if the result for this subproblem is already computed then return it
    if memo[n] != -1:
        return memo[n]

    memo[n] = countWaysRec(n - 1, memo) + countWaysRec(n - 2, memo)
    return memo[n]

def countWays(n):
    # Memoization array to store the results
    memo = [-1] * (n + 1)
    return countWaysRec(n, memo)
```

Time and Space Complexity :

Time Complexity : O(n)

Space Complexity : O(n)

Any other data structure that we can use to memoize ans ?

Newton School of Technology

We can store values against each Key ?

Dictionary (HashMap)

Solution:

```
def count_ways(n, memo):
    MOD = 10**9 + 7
    # Base cases
    if n == 0:
        return 1
    if n < 0:
        return 0
    # If already calculated, return stored result
    if n in memo:
        return memo[n]
    # Recursive relation
    memo[n] = (count_ways(n - 1, memo) + count_ways(n - 2, memo))%MOD
    return memo[n]

n = int(input())
memo = {}
result = count_ways(n, memo)
print(result)
```