Dynamic Programming

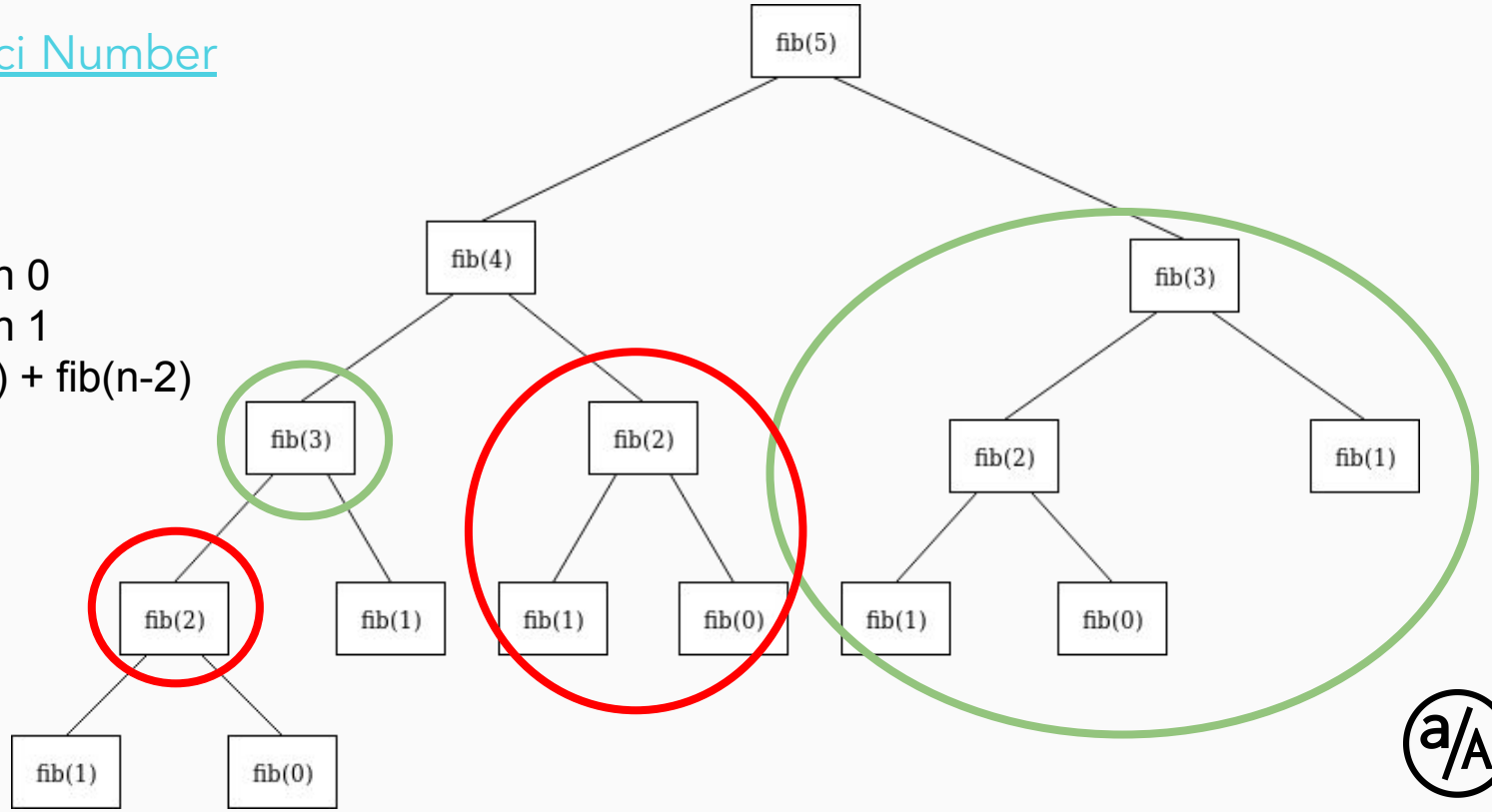
Intro to Dynamic Programming (DP)

- DP is not a standalone algorithm or technique.
- Rather, DP is a technique used to optimize less efficient solutions.
  - We have to understand the “brute force” solutions if we are to come up with DP solutions.
- We can identify when we can use DP by looking for repeated sub-problems.
- When you have large complexities, it may be useful to think about whether DP could be used or not.
- 1D DP refers to the solution space of the problem.
- Types of DP
  - Memoization (top-down DP)
  - Tabulation (bottom-up DP)

Demo

- Fibonacci Number
- $O(2^n)$

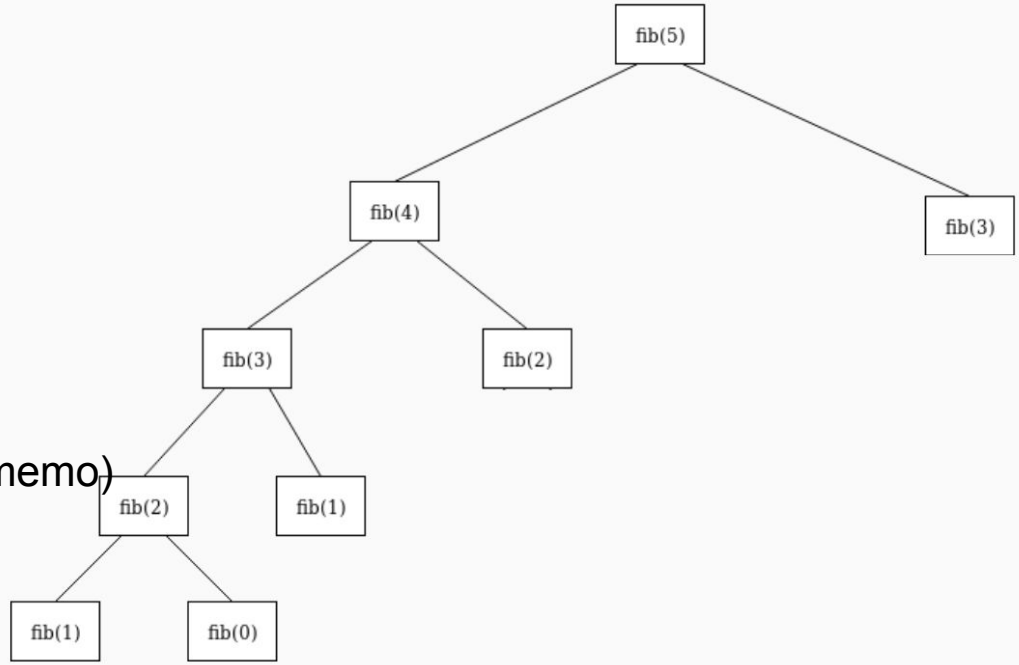
```
def fib(n):
    if n==0: return 0
    if n==1: return 1
    return fib(n-1) + fib(n-2)
```



Memoization (top-down)

- Fibonacci Number
- $O(2^n) => O(2n) => O(n)$

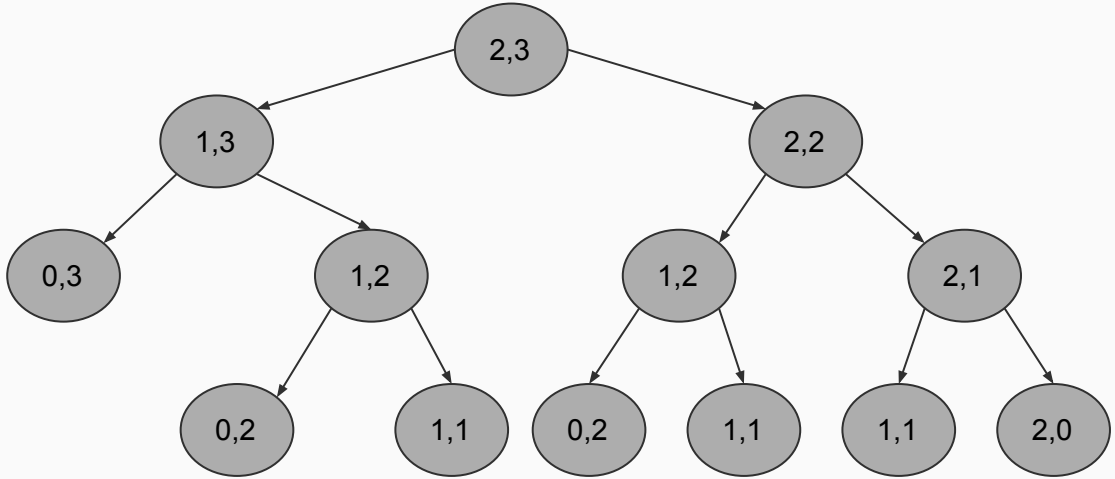
```
def fib(n, memo = {}):
    if n==0: return 0
    if n==1: return 1
    if memo[n]: return memo[n]

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

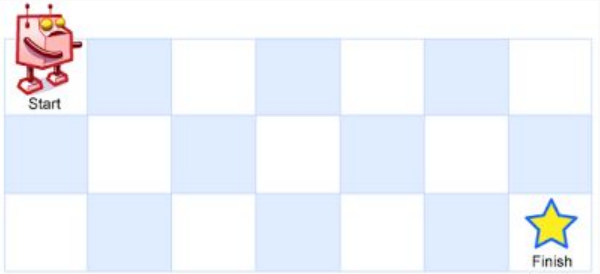


Unique Paths

Unique Paths

Runtime: $O(2^{m+n})$





Tabulation (bottom-up)

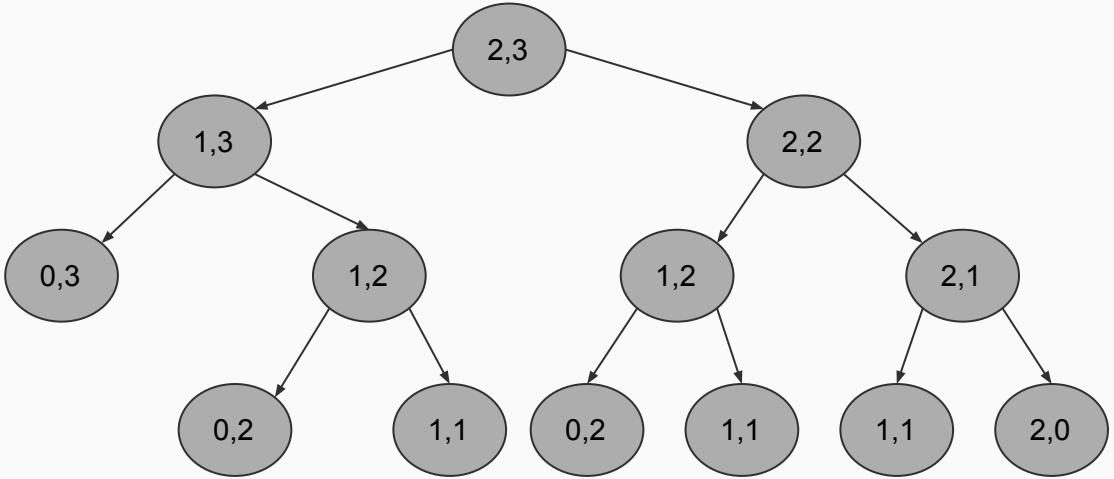
```
def fib(n):
    if n < 2: return n
    dp = [0,1]
    i = 2

    while i <= n:
        tmp = dp[1]
        dp[1] = dp[0] + dp[1]
        dp[0] = tmp
        i +=1
    return dp[1]
```

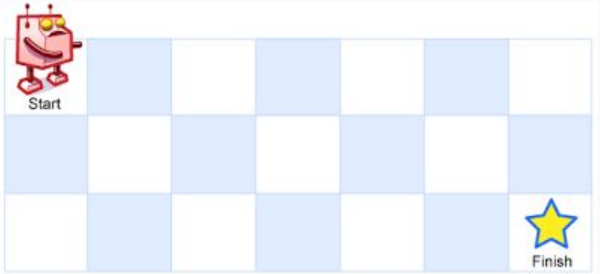
0	1	2	3	4	5

- The idea with bottom-up DP is to start from the bottom of our tree (i.e. our base case) and then work our way up towards the root (i.e. our original input).
- This solution is usually more difficult and requires some pre-planning to come up with rather than just modifying an existing recursive solution.

Unique Paths (tabulation)

Unique Paths





Questions?

Let’s practice!

- Review
  - Climbing Stairs
  - Coin Change
- Bonus
  - House Robber
  - Palindromic Substrings