Dynamic prog

Wednesday, 1 November 2023 8:09 AM

① Introduction to Dynamic Prog

→ optimization over plain Recursion

→ Idea is to reuse the solutions of subproblems when there are over-lapping sub-problems

Two ways

① Memoization (Top-Down Approach)

② Tabulation (Bottom-Up Approach)

Applications

① Bellman Ford Algo

② Floyd Warshall Algo

③ Diff Utility (LCS)

④ Search Closed Words (Edit Distance)

⑤ Resource Allocation (0-1 Knapsack)

② Memoization (Top-Down)

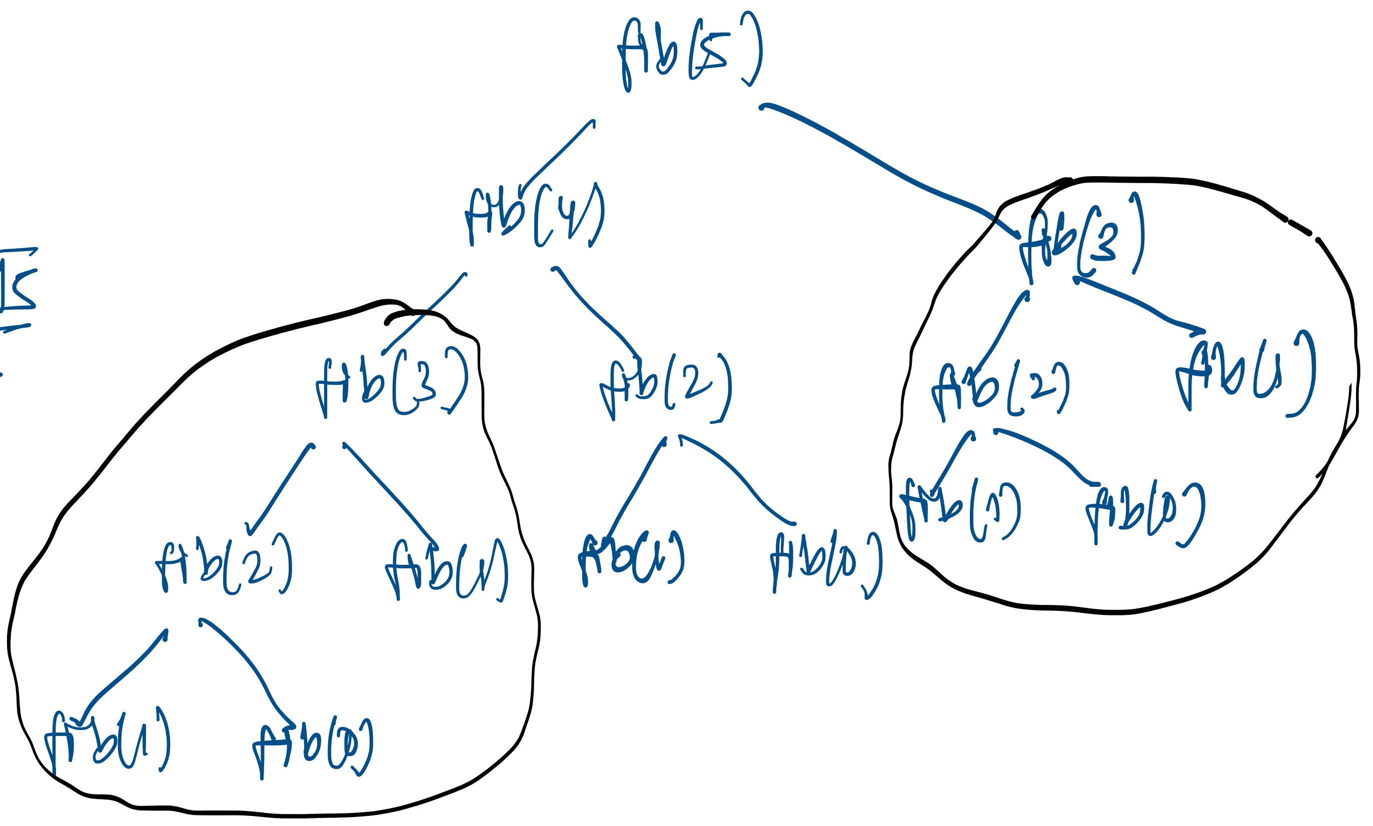
Ex: Fibonacci Number

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

Time: $O(\phi^n)$

$\phi \rightarrow \frac{1+\sqrt{5}}{2}$

- simple recursive solution with exponential time complexity
- we are going to optimize it using memoization



→ We solve subproblems again

→ Memoization Idea:

- store solutions and before proceeding further, check if already computed

```
memo = [None]*100
def fib(n):
    if memo[n] != None:
        return memo[n]
    if n==0 or n==1:
        memo[n]=n
    else:
        memo[n] = fib(n-1)+fib(n-2)
    return memo[n]
```

- We have 2n-1 function calls now
- Time: $O(n)$

③ Tabulation (Bottom-Up)

```
def fib(n):
    dp = [None] * (n+1)
    dp[0] = 0
    dp[1] = 1
    for i in range(2, n+1):
        dp[i] = dp[i-1] + dp[i-2]
    return dp[n]
```