Lecture 7: Dynamic Programming I

Overview

- Memoization & subproblems; bottom up
- Fibonacci
- Shortest paths

Dynamic Programming (DP)

- *Powerful algorithmic design technique*
- *Large class of seemingly exponential problems have a polynomial solution ("only") via DP*
- *Particularly for optimization problems (min / max)*

DP ~ «careful brute force»
DP ~ subproblems + «re-use»

Fibonacci Numbers

Problem

$F_1 = F_2 = 1;$
$F_n = F_{n-1} + F_{n-2}$

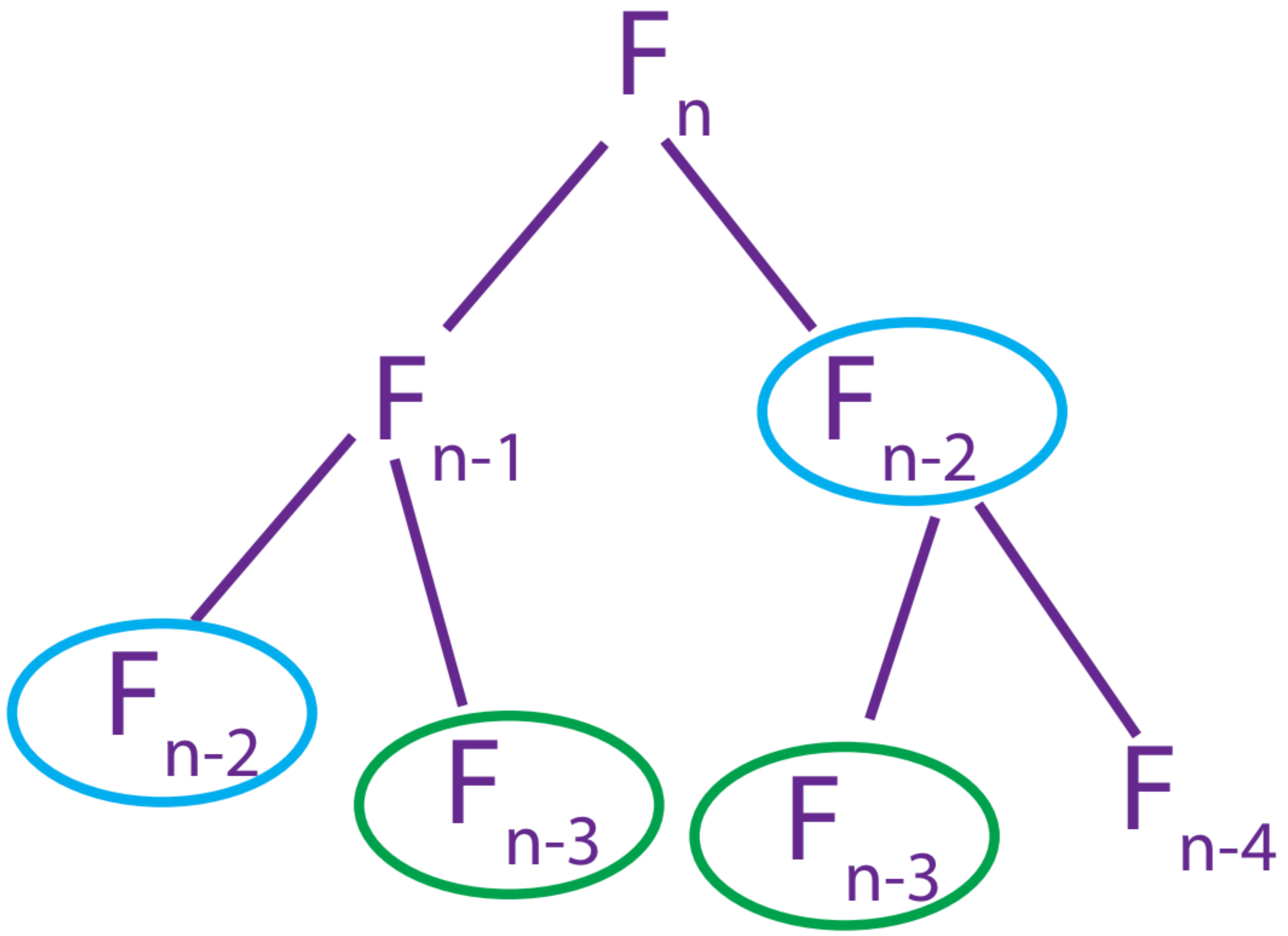
Goal: compute F_n

Solution: Naive Algorithm:

```
fib(n):
    if n ≤ 2: f = 1
    else: f = fib(n - 1) + fib(n - 2)
    return f
```

$T(n) = T(n-1) + T(n-2) + O(1)$
$\geq F_n \approx \phi^n$
$\geq 2T(n-2) + O(1) \geq 2^{n/2}$

Fibonacci Numbers Memoized Algorithm



Solution: Memoized Algorithm:

```
memo = {}
fib(n):
   if n in memo: return memo[n]
   else: if n ≤ 2 : f = 1
     else: f = fib(n – 1) + fib(n – 2)
     memo[n] = f
     return f
```

Fibonacci Numbers Memoized Algorithm

- fib(k) only recurses first time called, ∀k
- only ***n*** nonmemoized calls: k = n, n−1,…,1
- memoized calls free (Θ(1) time)
- Θ(1) time per call (ignoring recursion)

Solution: Memoized Algorithm:

```
memo = {}
fib(n):
   if n in memo: return memo[n]
   else: if n ≤ 2 : f = 1
     else: f = fib(n – 1) + fib(n – 2)
     memo[n] = f
     return f
```

Memoized DP Algorithm

DP ~ recursion + memoization

- memoize (remember) & re-use solutions to subproblems that help solve problem

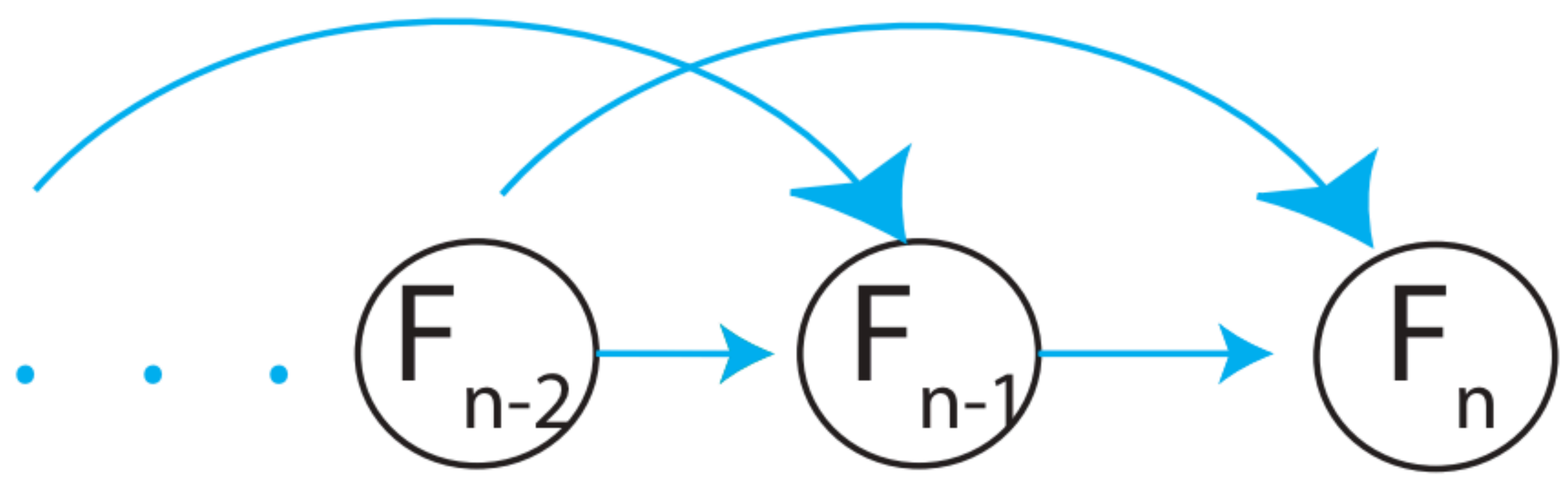
– in Fibonacci, subproblems are F1, F2, . . . , F

- time = # of subproblems · time/subproblem

- Fibonacci: # of subproblems is n, and time/subproblem is Θ(1) = Θ(n)

Fibonacci Numbers Bottom-up DP Algorithm

- exactly the same computation as memoized DP (recursion "unrolled")
- in general: topological sort of subproblem dependency DAG



- practically faster: no recursion
- can save space: just remember last 2 fibs

Solution: Memoized Algorithm:

```
fib = {}
for k in [1, 2, . . . , n]:
   if k ≤ 2: f = 1
   else: f = fib[k – 1] + fib[k – 2]
   fib[k] = f
return fib[n]
```