CMPS 2200 – Fall 2017

Dynamic Programming I

Carola Wenk

Slides courtesy of Charles Leiserson with changes and additions by Carola Wenk

Dynamic programming

- Algorithm design technique
- A technique for solving problems that have
    1. an optimal substructure property (recursion)
    2. overlapping subproblems
- **Idea:** Do not repeatedly solve the same subproblems, but solve them only once and store the solutions in a **dynamic programming table**

Example: Fibonacci numbers

- $F(0)=0$; $F(1)=1$; $F(n)=F(n-1)+F(n-2)$ for $n \geq 2$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

Dynamic-programming hallmark #1

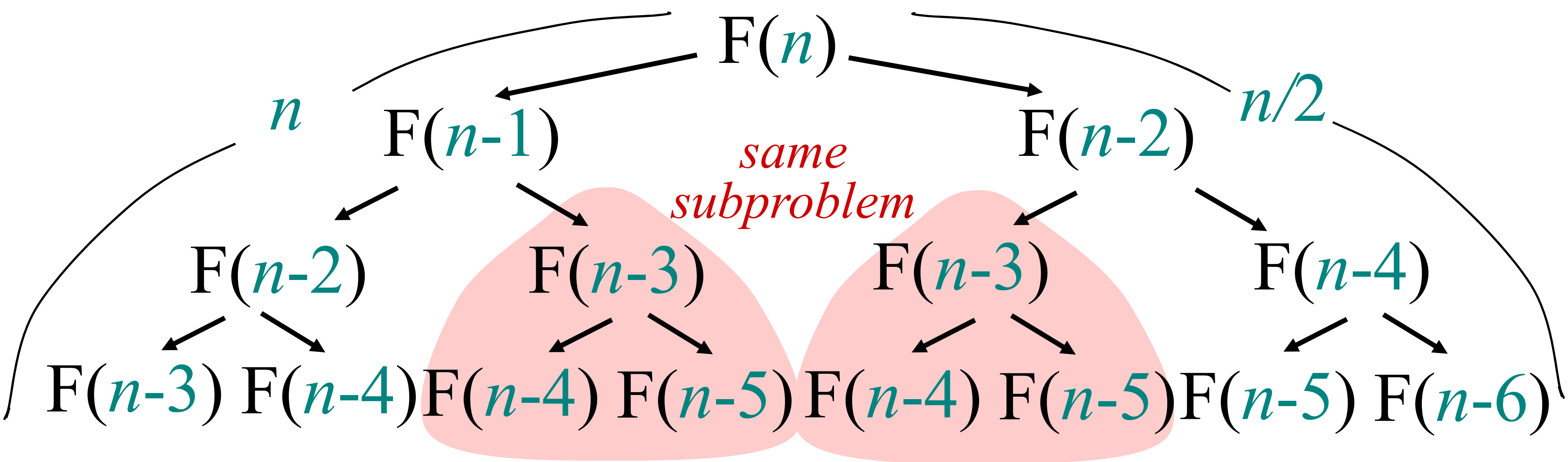
Optimal substructure

An optimal solution to a problem (instance) contains optimal solutions to subproblems.

→ ***Recursion***

Example: Fibonacci numbers

- $F(0)=0$; $F(1)=1$; $F(n)=F(n-1)+F(n-2)$ for $n \geq 2$
- Implement this recursion directly:



- Runtime is exponential: $2^{n/2} \leq T(n) \leq 2^n$
- But we are repeatedly solving the same subproblems

Dynamic-programming hallmark #2

Overlapping subproblems
A recursive solution contains a "small" number of distinct subproblems repeated many times.

The number of distinct Fibonacci subproblems is only n.

Dynamic-programming

There are two variants of dynamic programming:

1. Bottom-up dynamic programming (often referred to as “dynamic programming”)

2. Memoization

Bottom-up dynamic-programming algorithm

- Store 1D DP-table and fill bottom-up:

F:											
	0	1	1	2	3	5	8				

fibBottomUpDP(n)
    F[0] $\leftarrow$ 0
    F[1] $\leftarrow$ 1
    for ($i \leftarrow 2$, $i \leq n$, $i{+}{+}$)
        F[i] $\leftarrow$ F[i-1]+F[i-2]
    return F[n]

- Time $= \Theta(n)$, space $= \Theta(n)$

Memoization algorithm

Memoization: Use recursive algorithm. After computing a solution to a subproblem, store it in a table. Subsequent calls check the table to avoid redoing work.

fibMemoization(n)
    for all i**:** F[i] = null
    fibMemoizationRec(n,F)
    return F[n]

fibMemoizationRec(n,F)
    if (F[n]= null**)**
        if (n=0) F[n] ← 0
        if (n=1) F[n] ← 1
        F[n] ← fibMemoizationRec(n-1,F)
                + fibMemoizationRec(n-2,F)
    return F[n]

- Time = $\Theta(n)$, space = $\Theta(n)$