

Optimization Problems, Lecture 2, Segment 2

John Guttag

MIT Department of Electrical Engineering and
Computer Science

Search Tree Algorithm

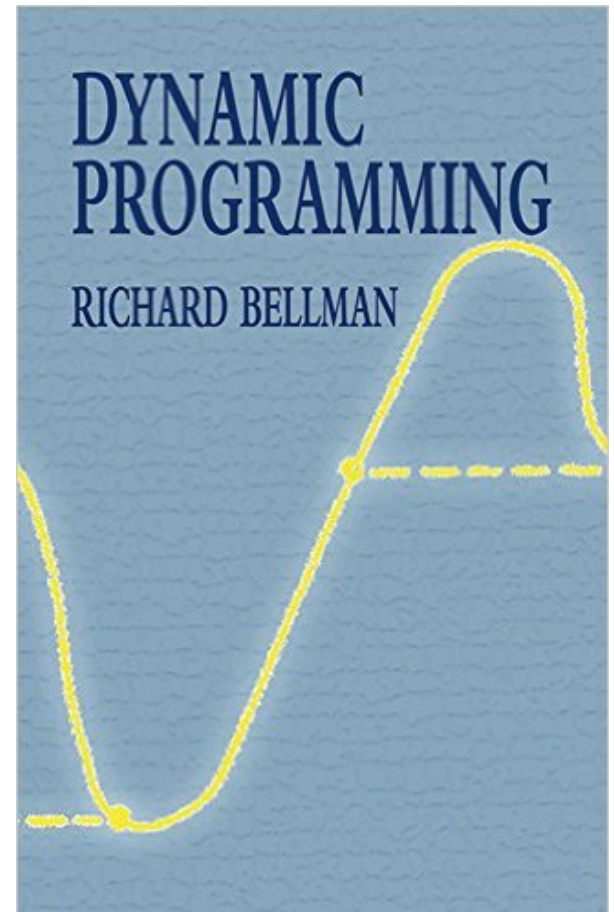
- Gave us a better answer than any of the greedy solutions
- Finished quickly
- But 2^8 is not a large number
- Let's look at what happens when we have a more extensive menu to choose from

Code to Try Larger Examples

```
import random ←  
  
def buildLargeMenu(numItems, maxVal, maxCost):  
    items = []  
    for i in range(numItems):  
        items.append(Food(str(i),  
                           random.randint(1, maxVal), sequence of int  
                           random.randint(1, maxCost)))  
    return items  
  
for numItems in (5, 10, 15, 20, 25, 30, 35, 40, 45):  
    items = buildLargeMenu(numItems, 90, 250)  
    testMaxVal(items, 750, False)
```

Is It Hopeless?

- In theory, yes
- In practice, no!
- Dynamic programming to the rescue



Dynamic Programming?

Sometimes a name is just a name

“The 1950s were not good years for mathematical research... I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics... What title, what name, could I choose? ... It's impossible to use the word dynamic in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities.

-- Richard Bellman

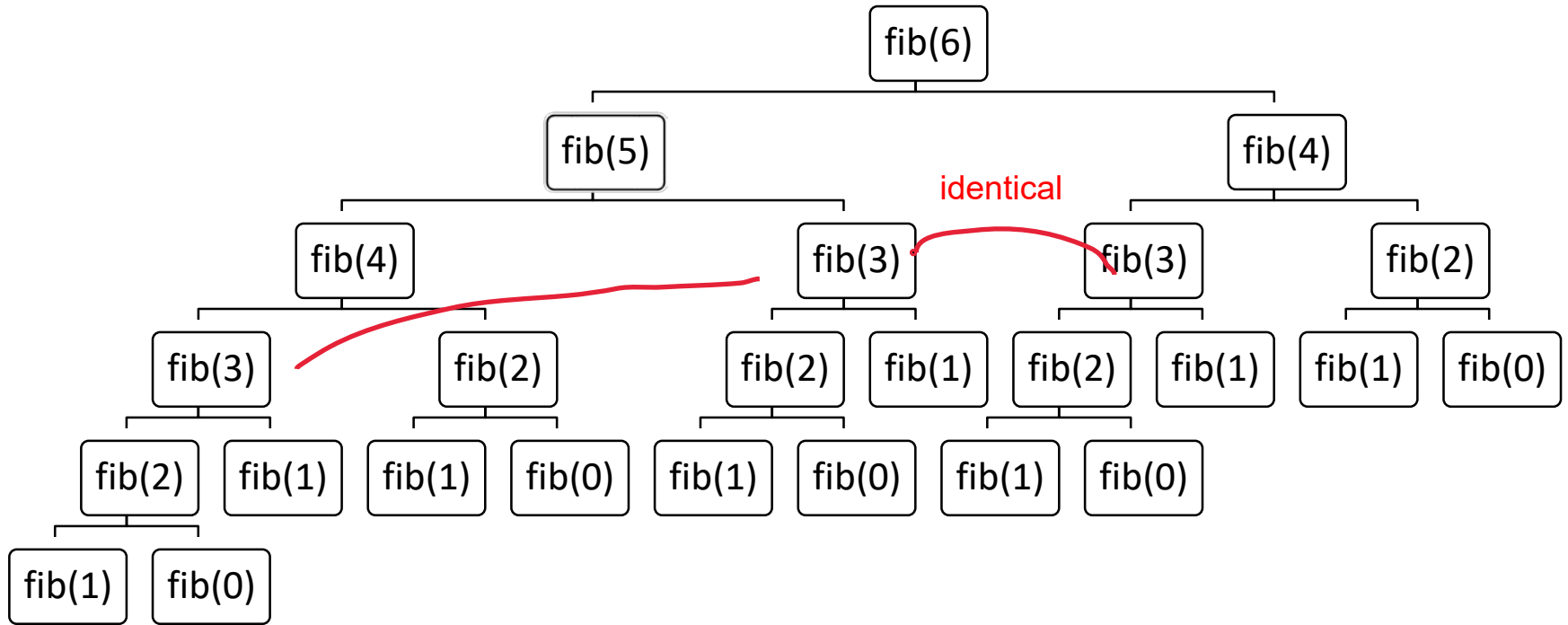
Recursive Implementation of Fibonacci

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

`fib(120)` = 8,670,007,398,507,948,658,051,921

Call Tree for Recursive Fibonacci(6) = 13

fib calls itself so many times



Clearly a Bad Idea to Repeat Work

- Trade a time for space
- Create a table to record what we've done
 - Before computing $\text{fib}(x)$, check if value of $\text{fib}(x)$ already stored in the table
 - If so, look it up
 - If not, compute it and then add it to table
 - Called **memoization**

Using a Memo to Compute Fibonacci

```
def fastFib(n, memo = {}):  
    """Assumes n is an int >= 0, memo used only by  
        recursive calls  
        Returns Fibonacci of n"""  
    if n == 0 or n == 1:  
        return 1  
    try:  
        return memo[n]  
    except KeyError:  
        result = fastFib(n-1, memo) +\  
                 fastFib(n-2, memo)  
        memo[n] = result  
        return result
```

When Does It Work?

- **Optimal substructure**: a globally optimal solution can be found by combining optimal solutions to local subproblems

- For $x > 1$, $\text{fib}(x) = \text{fib}(x - 1) + \text{fib}(x - 2)$

- **Overlapping subproblems**: finding an optimal solution involves solving the same problem multiple times

- Compute $\text{fib}(x)$ or many times

What About 0/1 Knapsack Problem?

- Do these conditions hold?

