Dynamic Programming

https://cs.pomona.edu/classes/cs140/

- Bellman explains the reasoning behind the term dynamic programming in his autobiography, Eye of the Hurricane: An Autobiography (1984):

I spent the Fall quarter (of 1950) at RAND. My first task was to find a name for multistage decision processes. An interesting question is, Where did the name, dynamic programming, come from? The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word research. I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term research in his presence. You can imagine how he felt, then, about the term mathematical.

The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word "programming". I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying. I thought, let's kill two birds with one stone. Let's take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that 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.

Outline

Topics and Learning Objectives

- Discuss the dynamic programming paradigm

Assessments

- None

Dynamic Programming (DP) is a glorified form of brute force with caching.

- William Fiset

Speed up an algorithm when it frequently repeats the same computations.

Dynamic Programming

An algorithm design technique/paradigm that typically takes one of the following forms:

1. Top-Down (memoization—cache results and use recursion)
2. Bottom-Up (tabulation—store results in a table)

Used to solve problems with the following properties:

- Overlapping subproblems and
- Optimal substructure

Overlapping Subproblems

Solving the original problem requires subproblems to be repeatedly solved

To put it another way: you must solve the same subproblem multiple times

For example, solving the Fibonacci sequence

The Fibonacci Sequence

```
Fibonacci(1) = 1
Fibonacci(2) = 1
Fibonacci(n) = Fibonacci(n - 1) + Fibonacci(n - 2)
```

```
FUNCTION Fibonacci(n)
    IF n == 1 || n == 2
        RETURN 1
    RETURN Fibonacci(n - 1) + Fibonacci(n - 2)
```

The Fibonacci Sequence

```
FUNCTION Fibonacci(n)
    IF n == 1 || n == 2
        RETURN 1
    RETURN Fibonacci(n - 1) + Fibonacci(n - 2)
```

```
Fib(6) = Fib(5) + Fib(4)
       = [Fib(4) + Fib(3)] + [Fib(3) + Fib(2)]
```

The Fibonacci Sequence

```
FUNCTION Fibonacci(n)
    IF n == 1 || n == 2
        RETURN 1
    RETURN Fibonacci(n - 1) + Fibonacci(n - 2)
```

```
Fib(6) = Fib(5) + Fib(4)
       = [Fib(4) + Fib(3)] + [Fib(3) + Fib(2)]
       = [[Fib(3) + Fib(2)] + [Fib(2) + Fib(1)]] + [[Fib(2) + Fib(1)] + Fib(2)]
```

The Fibonacci Sequence

```
FUNCTION Fibonacci(n)
    IF n == 1 || n == 2
        RETURN 1
    RETURN Fibonacci(n - 1) + Fibonacci(n - 2)
```

```
Fib(6) = Fib(5) + Fib(4)
       = [Fib(4) + Fib(3)] + [Fib(3) + Fib(2)]
       = [[Fib(3) + Fib(2)] + [Fib(2) + Fib(1)]] + [[Fib(2) + Fib(1)] + Fib(2)]
       = [[[Fib(2) + Fib(1)] + Fib(2)] + [Fib(2) + Fib(1)]] + [[Fib(2) + Fib(1)] + Fib(2)]
```

The Fibonacci Sequence

```
FUNCTION Fibonacci(n)
    IF n == 1 || n == 2
        RETURN 1
    RETURN Fibonacci(n - 1) + Fibonacci(n - 2)
```

```
Fib(6) = Fib(5) + Fib(4)
       = [Fib(4) + Fib(3)] + [Fib(3) + Fib(2)]
       = [[Fib(3) + Fib(2)] + [Fib(2) + Fib(1)]] + [[Fib(2) + Fib(1)] + Fib(2)]
       = [[[Fib(2) + Fib(1)] + Fib(2)] + [Fib(2) + Fib(1)]] + [[Fib(2) + Fib(1)] + Fib(2)]
```

The Fibonacci Sequence

```
FUNCTION Fibonacci(n)
    IF n == 1 || n == 2
        RETURN 1
    RETURN Fibonacci(n - 1) + Fibonacci(n - 2)
```

```
Fib(6) = Fib(5) + Fib(4)
       = [Fib(4) + Fib(3)] + [Fib(3) + Fib(2)]
       = [[Fib(3) + Fib(2)] + [Fib(2) + Fib(1)]] + [[Fib(2) + Fib(1)] + Fib(2)]
       = [[[Fib(2) + Fib(1)] + Fib(2)] + [Fib(2) + Fib(1)]] + [[Fib(2) + Fib(1)] + Fib(2)]
```

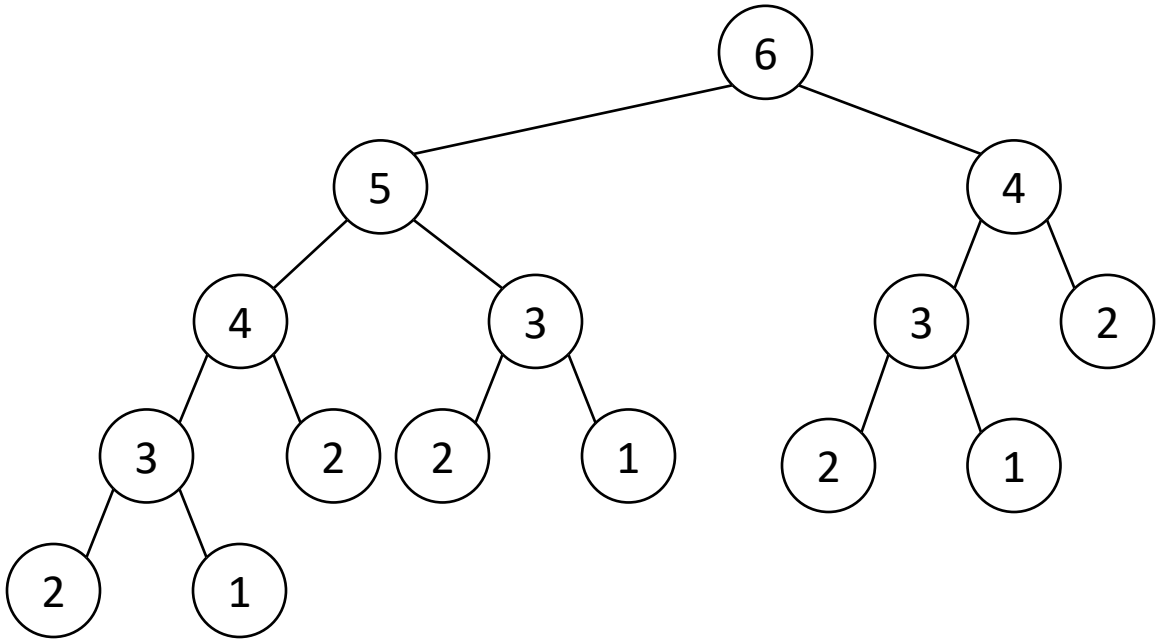
The Fibonacci Sequence

```
FUNCTION Fibonacci(n)
    IF n == 1 || n == 2
        RETURN 1
    RETURN Fibonacci(n - 1) + Fibonacci(n - 2)
```

```
Fib(6) = Fib(5) + Fib(4)
       = [Fib(4) + Fib(3)] + [Fib(3) + Fib(2)]
       = [[Fib(3) + Fib(2)] + [Fib(2) + Fib(1)]] + [[Fib(2) + Fib(1)] + Fib(2)]
       = [[[Fib(2) + Fib(1)] + Fib(2)] + [Fib(2) + Fib(1)]] + [[Fib(2) + Fib(1)] + Fib(2)]
```

The Fibonacci Sequence

```
FUNCTION Fibonacci(n)
    IF n == 1 || n == 2
        RETURN 1
    RETURN Fibonacci(n - 1) + Fibonacci(n - 2)
```

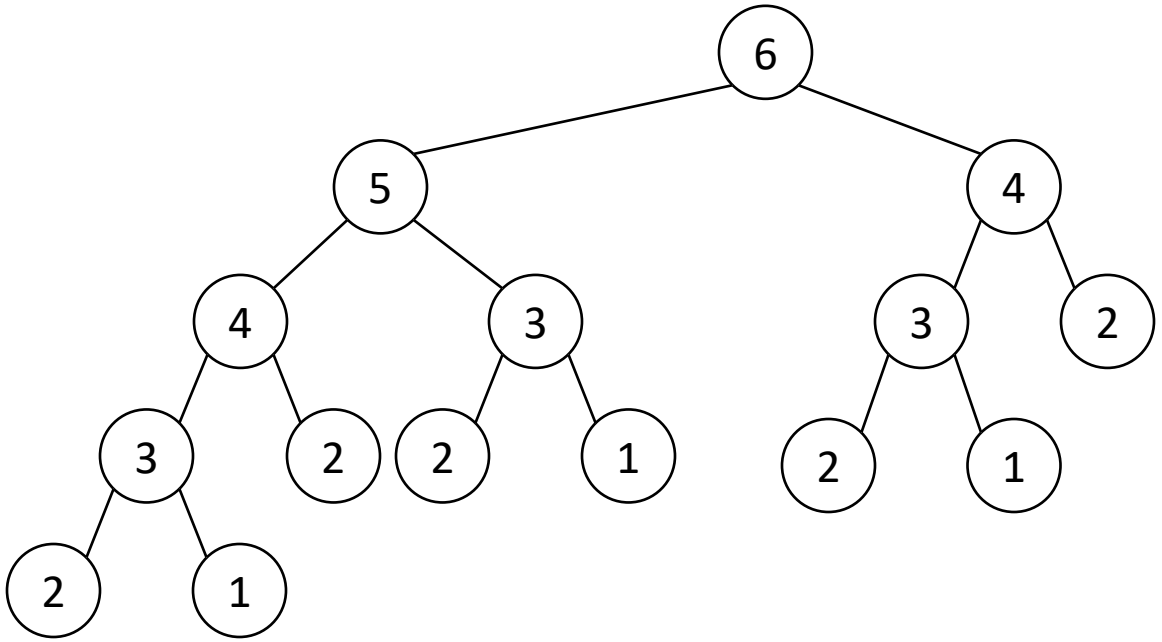


The Fibonacci Sequence

```
FUNCTION Fibonacci(n)
    IF n == 1 || n == 2
        RETURN 1
    RETURN Fibonacci(n - 1) + Fibonacci(n - 2)
```

What is the recurrence equation?

What is the time complexity?

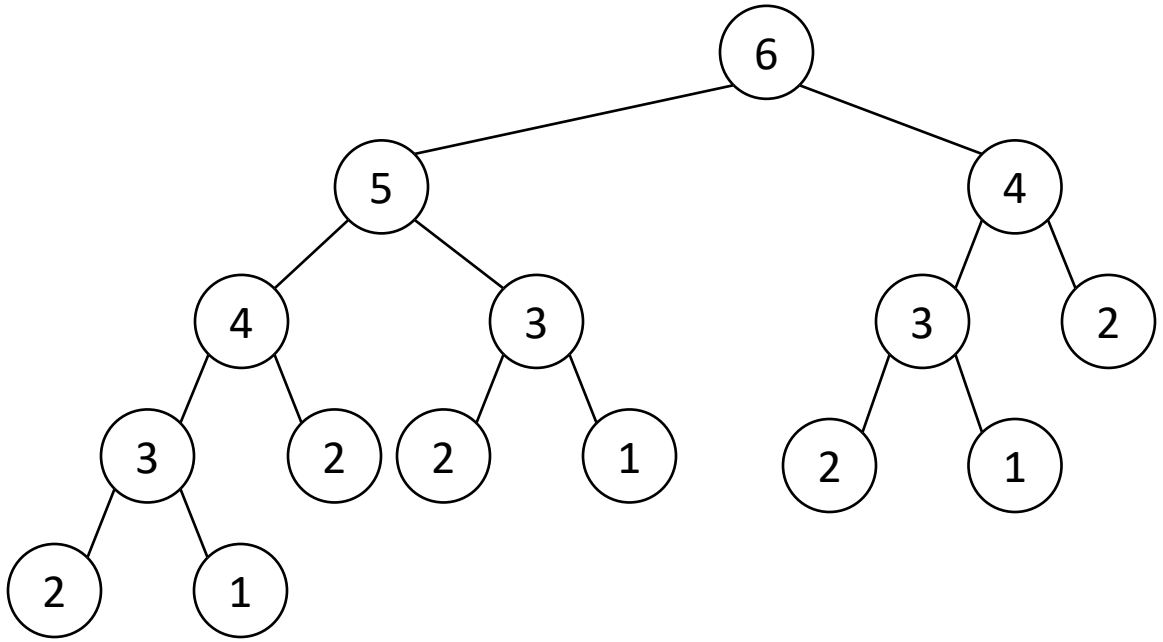


The Fibonacci Sequence

What is the recurrence equation?

What is the time complexity?

```
FUNCTION Fibonacci(n)
    IF n == 1 || n == 2
        RETURN 1
    RETURN Fibonacci(n - 1) + Fibonacci(n - 2)
```

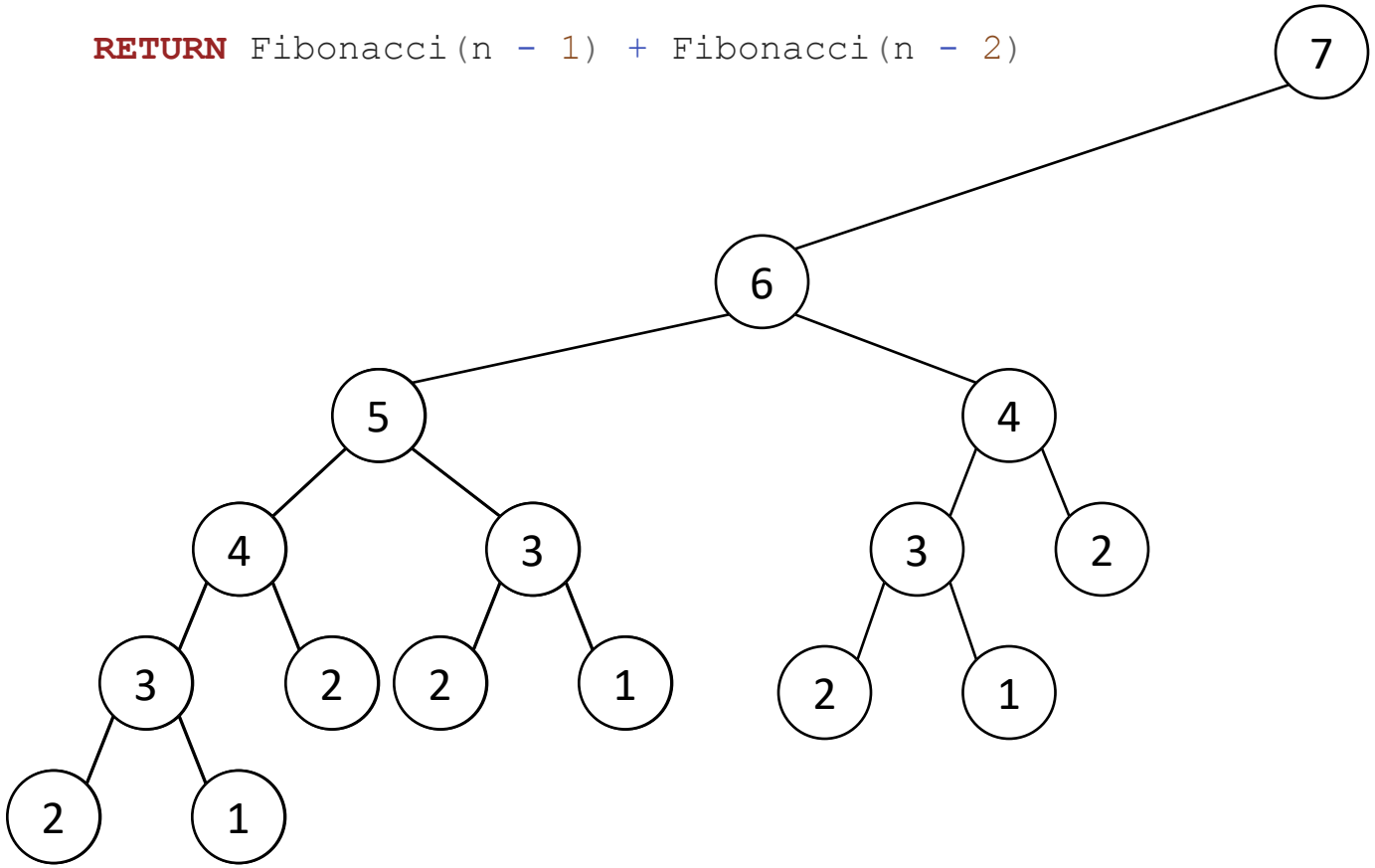


The Fibonacci Sequence

What is the recurrence equation?

What is the time complexity?

```
FUNCTION Fibonacci(n)
    IF n == 1 || n == 2
        RETURN 1
    RETURN Fibonacci(n - 1) + Fibonacci(n - 2)
```

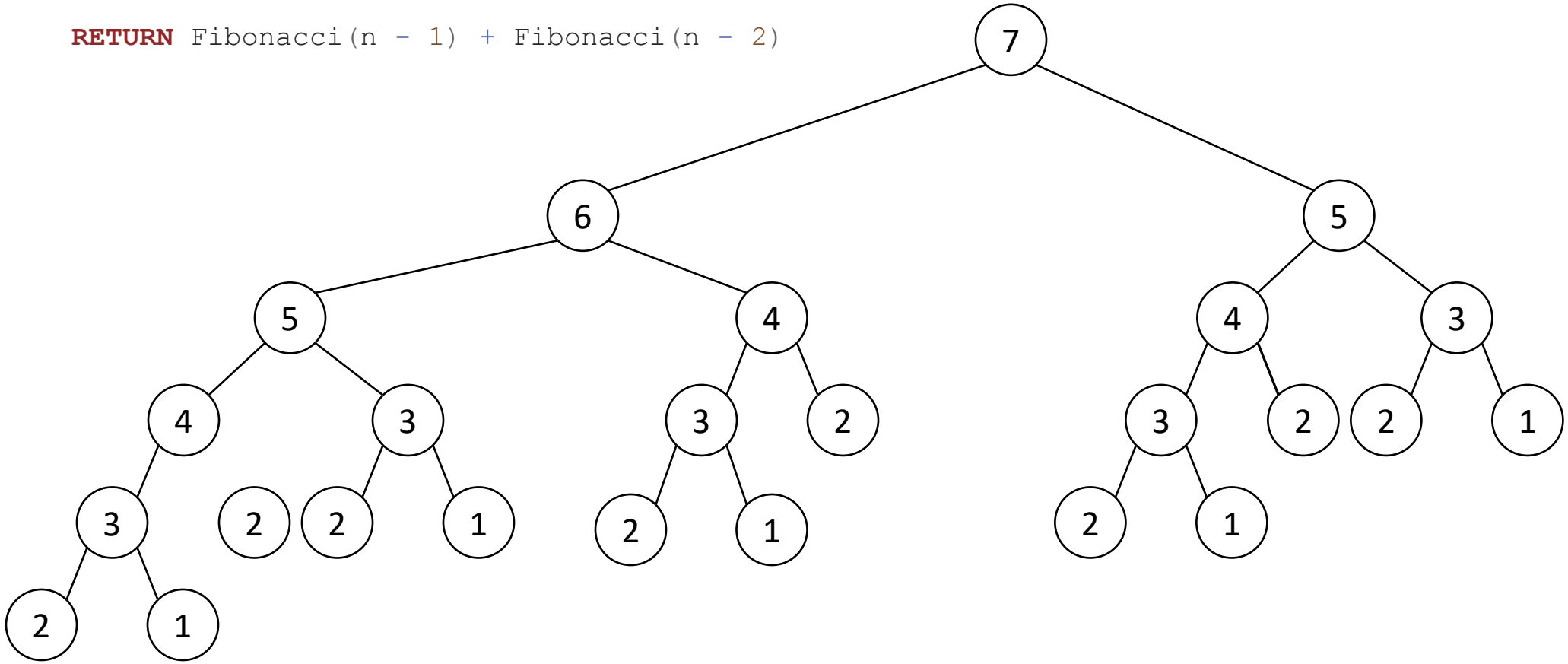


The Fibonacci Sequence

What is the recurrence equation?

What is the time complexity?

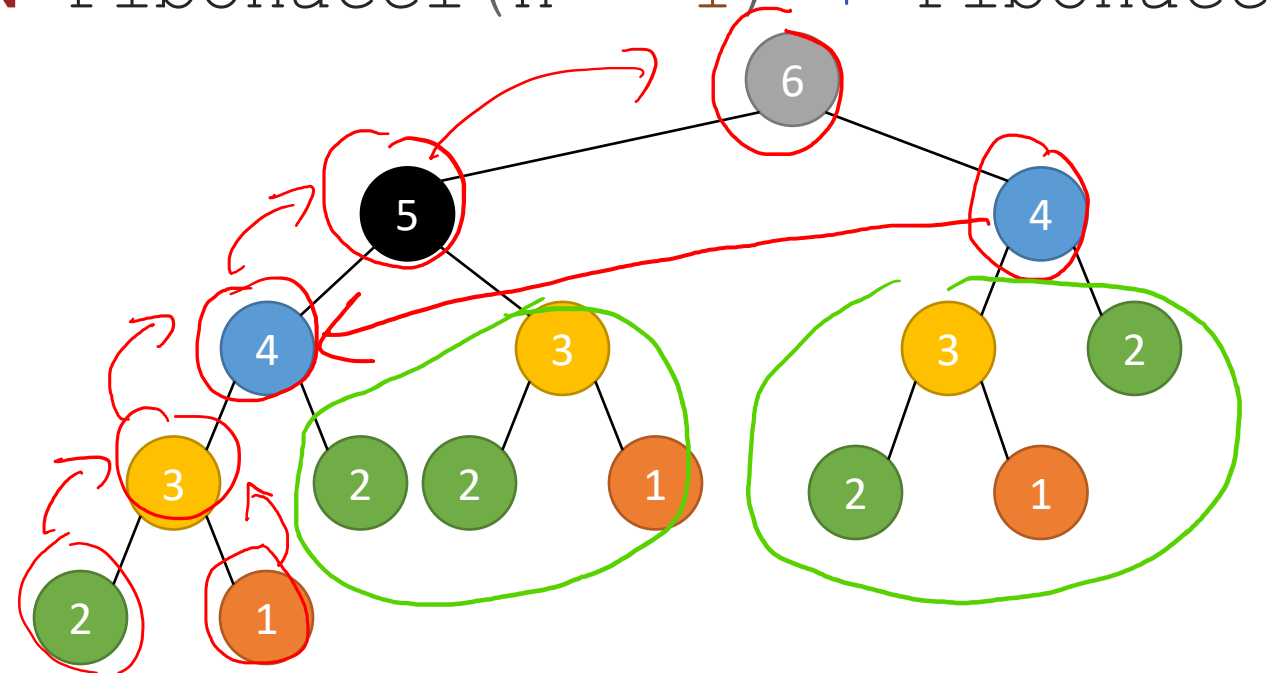
```
FUNCTION Fibonacci(n)
    IF n == 1 || n == 2
        RETURN 1
    RETURN Fibonacci(n - 1) + Fibonacci(n - 2)
```



The Fibonacci Sequence

```
FUNCTION Fibonacci(n)
    IF n == 1 || n == 2
        RETURN 1
    RETURN Fibonacci(n - 1) + Fibonacci(n - 2)
```

Really only 6 different calls (linear)



The Fibonacci Sequence (Top Down: Memoized)

```
FUNCTION Fibonacci(n, cache)
    IF n == 1 || n == 2
        RETURN 1



    fibn = Fibonacci(n - 1) + Fibonacci(n - 2)



    RETURN fibn
```

The Fibonacci Sequence (Top Down: Memoized)

```
FUNCTION Fibonacci(n, cache)
    IF n == 1 || n == 2
        RETURN 1
    IF n IN cache
        RETURN cache[n]

    fibn = Fibonacci(n - 1) + Fibonacci(n - 2)
    cache[n] = fibn

    RETURN fibn
```

The Fibonacci Sequence (Top Down: Memoized)

```
FUNCTION Fibonacci(n, cache)
    IF n == 1 || n == 2
        RETURN 1
    IF n IN cache
        RETURN cache[n]

    fibn = Fibonacci(n - 1) + Fibonacci(n - 2)
    cache[n] = fibn

    RETURN fibn
```

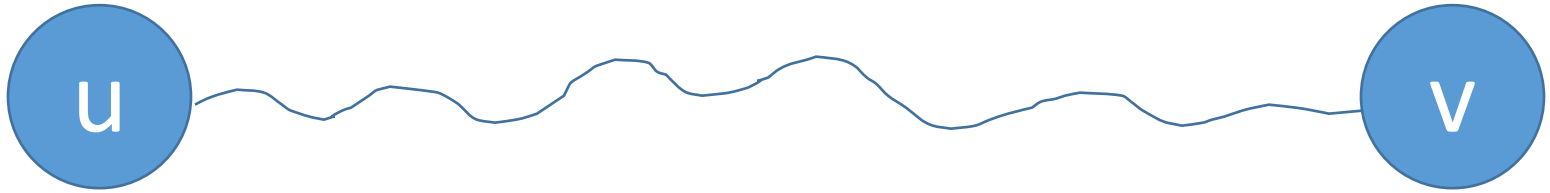
The Fibonacci Sequence (Bottom Up: Tabular)

```
FUNCTION Fibonacci(n)
    fibs = [0] * n
    # Using base-1 index for convenience
    fibs[1] = 1
    fibs[2] = 1

    FOR i IN [3 ..< n]
        fibs[i] = fibs[i - 1] + fibs[i - 2]

    RETURN fibs[n]
```

Optimal Substructure

The solution to the original problem includes optimal solutions to subproblems

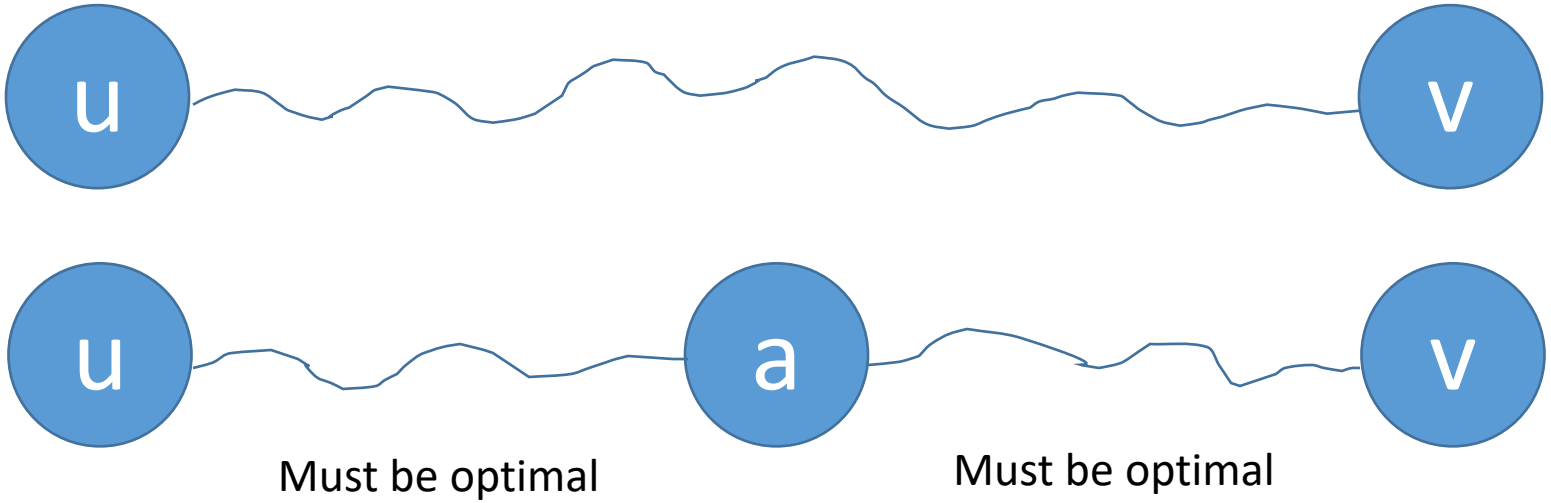
For example, the shortest path problem



Optimal Substructure

The solution to the original problem includes optimal solutions to subproblems

For example, the shortest path problem



Dynamic Programming

Principles:

1. Identify a small number of subproblems.

2. Quickly and correctly solve larger subproblems when provided solutions to the smaller subproblems (e.g., recursive calls).

3. Quickly computer the final, complete problem (often this is just the biggest subproblem and nothing special needs to happen).