Introduction to Dynamic Programming

Lecture 13

Thursday, October 8, 2020

LaTeXed: October 13, 2020 09:52

13.1
Recursion and Memoization

13.1.1
Fibonacci Numbers

Fibonacci Numbers

Fibonacci numbers defined by recurrence:

$$F(n) = F(n-1) + F(n-2) \text{ and } F(0) = 0, F(1) = 1.$$

These numbers have many interesting properties. A journal The Fibonacci Quarterly!

1. **Binet's formula**: $F(n) = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}} \approx \frac{1.618^n - (-0.618)^n}{\sqrt{5}} \approx \frac{1.618^n}{\sqrt{5}}$

   φ is the golden ratio $(1+\sqrt{5})/2 \simeq 1.618$.
2. $\lim_{n\to\infty} F(n+1)/F(n) = \varphi$

Fibonacci Numbers

Fibonacci numbers defined by recurrence:

$$F(n) = F(n-1) + F(n-2) \text{ and } F(0) = 0, F(1) = 1.$$

These numbers have many interesting properties. A journal The Fibonacci Quarterly!

1. **Binet's formula**: $F(n) = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}} \approx \frac{1.618^n - (-0.618)^n}{\sqrt{5}} \approx \frac{1.618^n}{\sqrt{5}}$

   φ is the golden ratio $(1+\sqrt{5})/2 \simeq 1.618$.
2. $\lim_{n\to\infty} F(n+1)/F(n) = \varphi$

How many bits?

Consider the nth Fibonacci number $F(n)$. Writing the number $F(n)$ in base 2 requires

- $\Theta(n^2)$ bits.
- $\Theta(n)$ bits.
- $\Theta(\log n)$ bits.
- $\Theta(\log \log n)$ bits.

Recursive Algorithm for Fibonacci Numbers

Question: Given n, compute $F(n)$.

```
Fib(n):
    if (n = 0)
        return 0
    else if (n = 1)
        return 1
    else
        return Fib(n − 1) + Fib(n − 2)
```

Running time? Let $T(n)$ be the number of additions in Fib(n).

$$T(n) = T(n-1) + T(n-2) + 1 \text{ and } T(0) = T(1) = 0$$

Recursive Algorithm for Fibonacci Numbers

Question: Given n, compute $F(n)$.

```
Fib(n):
    if (n = 0)
        return 0
    else if (n = 1)
        return 1
    else
        return Fib(n − 1) + Fib(n − 2)
```

Running time? Let $T(n)$ be the number of additions in Fib(n).

$$T(n) = T(n-1) + T(n-2) + 1 \text{ and } T(0) = T(1) = 0$$

Recursive Algorithm for Fibonacci Numbers

Question: Given n, compute $F(n)$.

```
Fib(n):
    if (n = 0)
        return 0
    else if (n = 1)
        return 1
    else
        return Fib(n − 1) + Fib(n − 2)
```

Running time? Let $T(n)$ be the number of additions in Fib(n).

$$T(n) = T(n-1) + T(n-2) + 1 \text{ and } T(0) = T(1) = 0$$

Recursive Algorithm for Fibonacci Numbers

Question: Given n, compute $F(n)$.

```
Fib(n):
    if (n = 0)
        return 0
    else if (n = 1)
        return 1
    else
        return Fib(n − 1) + Fib(n − 2)
```

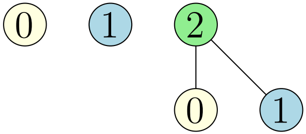
Running time? Let $T(n)$ be the number of additions in Fib(n).

$$T(n) = T(n-1) + T(n-2) + 1 \text{ and } T(0) = T(1) = 0$$

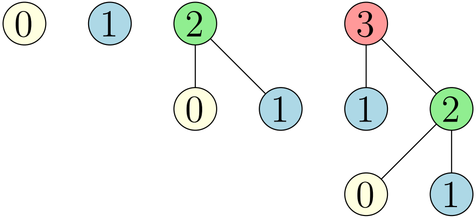
Roughly same as $F(n)$: $T(n) = \Theta(\varphi^n)$.
The number of additions is exponential in n. Can we do better?

Recursion tree for the Recursive Fibonacci

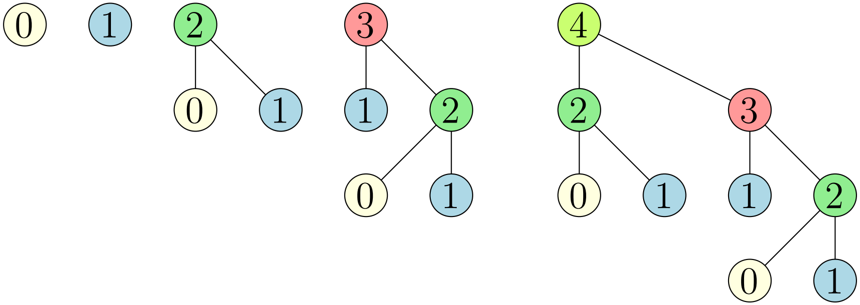
Recursion tree for the Recursive Fibonacci



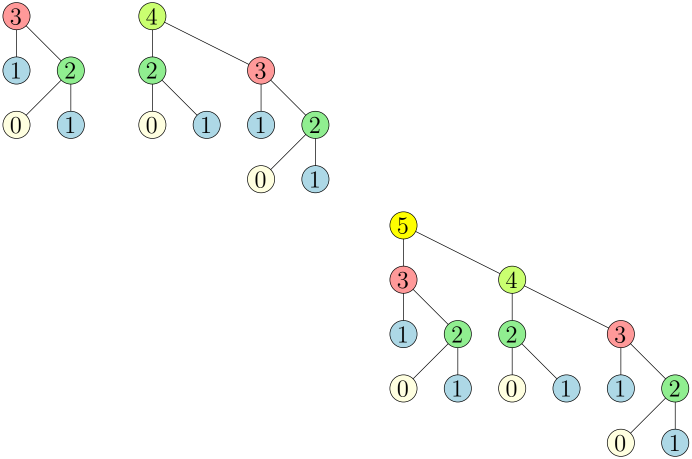
Recursion tree for the Recursive Fibonacci



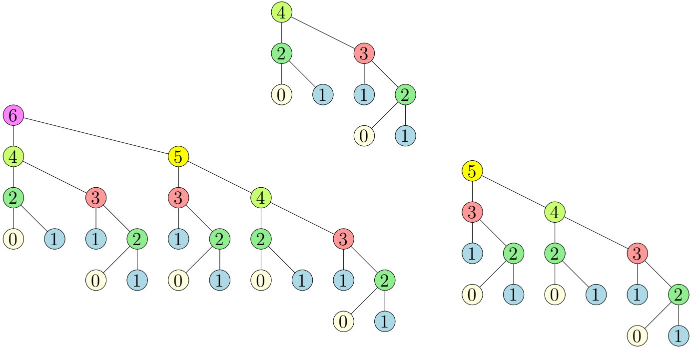
Recursion tree for the Recursive Fibonacci



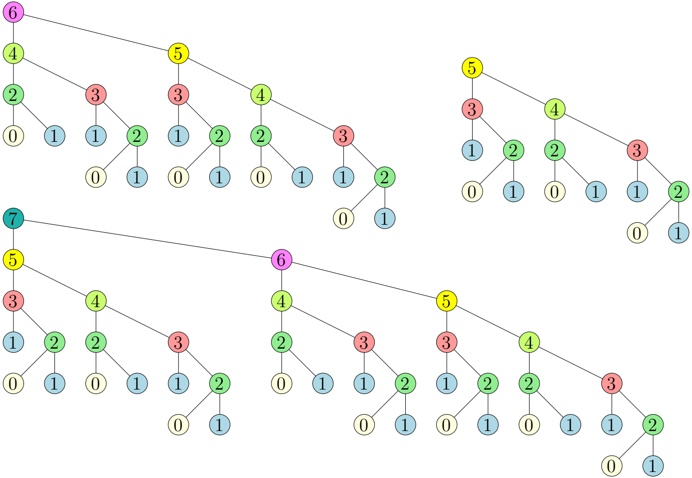
Recursion tree for the Recursive Fibonacci



Recursion tree for the Recursive Fibonacci



Recursion tree for the Recursive Fibonacci



An iterative algorithm for Fibonacci numbers

```
FibIter(n):
    if (n = 0) then
        return 0
    if (n = 1) then
        return 1
    F[0] = 0
    F[1] = 1
    for i = 2 to n do
        F[i] = F[i - 1] + F[i - 2]
    return F[n]
```

What is the running time of the algorithm? $O(n)$ additions.

An iterative algorithm for Fibonacci numbers

```
FibIter(n):
    if (n = 0) then
        return 0
    if (n = 1) then
        return 1
    F[0] = 0
    F[1] = 1
    for i = 2 to n do
        F[i] = F[i − 1] + F[i − 2]
    return F[n]
```

What is the running time of the algorithm? $O(n)$ additions.

An iterative algorithm for Fibonacci numbers

```
FibIter(n):
    if (n = 0) then
        return 0
    if (n = 1) then
        return 1
    F[0] = 0
    F[1] = 1
    for i = 2 to n do
        F[i] = F[i - 1] + F[i - 2]
    return F[n]
```

What is the running time of the algorithm? $O(n)$ additions.

What is the difference?

1. Recursive algorithm is computing the same numbers again and again.
2. Iterative algorithm is storing computed values and building bottom up the final value. Memoization.

Dynamic Programming:

Finding a recursion that can be effectively/efficiently memoized.

Leads to polynomial time algorithm if number of sub-problems is polynomial in input size.

What is the difference?

1. Recursive algorithm is computing the same numbers again and again.
2. Iterative algorithm is storing computed values and building bottom up the final value. Memoization.

Dynamic Programming:

Finding a recursion that can be effectively/efficiently memoized.

Leads to polynomial time algorithm if number of sub-problems is polynomial in input size.

What is the difference?

1. Recursive algorithm is computing the same numbers again and again.
2. Iterative algorithm is storing computed values and building bottom up the final value. Memoization.

Dynamic Programming:

Finding a recursion that can be effectively/efficiently memoized.

Leads to polynomial time algorithm if number of sub-problems is polynomial in input size.

THE END

...

(for now)