The Fibonacci Numbers

Fibonacci Numbers

\[
\begin{align*}
F_0 &= 0 \\
F_1 &= 1 \\
F_n &= F_{n-1} + F_{n-2} \text{ for } n \geq 2.
\end{align*}
\]

\[0 \ 1 \ 1 \ 2 \ 3 \ 5 \ 8 \ 13 \ 21 \ 34 \ 55 \ 89 \ \ldots\]

Why are they important?

- Fibonacci heaps (data structure)
- Appear in analysis of algorithms (e.g. Euclid’s algorithm)
- Appear everywhere in nature
- Interesting and instructive computational problem
source: realworldmathematics at wordpress
source: brian koberlein
Computing the Fibonacci Numbers

Naïve Algorithm

```
Require: Integer \( n \geq 0 \)
if \( n \leq 1 \) then
    return \( n \)
else
    return \( \text{FIB}(n - 1) + \text{FIB}(n - 2) \)
```

What is the runtime of this algorithm?

Runtime:
- Without recursive calls, runtime is \( O(1) \)
- Hence, runtime is \( O(\text{“number of recursive calls”}) \)
Define Recurrence:

\[ T(n) : \text{number of recursive calls to \texttt{FIB} when called with parameter } n \]

\[
\begin{align*}
T(0) &= T(1) = 1 \\
T(n) &= 1 + T(n-1) + T(n-2), \text{ for } n \geq 2.
\end{align*}
\]

How to Solve this Recurrence?

- We will use the recursion tree technique to obtain a guess for an upper bound
- We will verify the guess with the substitution method
Recursion Tree for $T$

Observe:
- Each node contributes 1
- Hence, $T(n)$ equals number of nodes
- Number of levels of recursion tree: $n$
- Our guess: $T(n) \leq c^n$ (we believe $c \leq 2$)
Recall:

\[ T(0) = T(1) = 1 \]
\[ T(n) = 1 + T(n - 1) + T(n - 2), \text{ for } n \geq 2. \]

Our guess: \( T(n) \leq c^n \)

Substitute Guess into Recurrence:

\[ T(n) = 1 + T(n - 1) + T(n - 2) \leq 1 + c^{n-1} + c^{n-2} \]

- It is required that \( 1 + c^{n-1} + c^{n-2} \leq c^n \)
- The additive 1 prevents us from getting a similar form as \( c^n \)
- Try different guess: \( T(n) \leq c^n - 1 \)
New Guess: $T(n) \leq c^n - 1$

$$
T(n) = 1 + T(n-1) + T(n-2) \\
\leq 1 + (c^{n-1} - 1) + (c^{n-2} - 1) = c^{n-1} + c^{n-2} - 1.
$$

Select smallest possible $c$:

$$
c^{n-1} + c^{n-2} = c^n \\
0 = c^2 - c - 1 \\
c = \frac{1 + \sqrt{5}}{2} \approx 1.618033989. \text{ Golden Ratio!}
$$

Base Case:

- $T(0) = T(1) = 1$
- $c^0 - 1 = 0$ and $c^1 - 1 \approx 0.61$ $\times$
Another New Guess: \( T(n) \leq k \cdot c^n - 1 \)

\[
T(n) = 1 + T(n-1) + T(n-2) \\
\leq 1 + (k \cdot c^{n-1} - 1) + (k \cdot c^{n-2} - 1) \\
= k (c^{n-1} + c^{n-2}) - 1.
\]

Select smallest possible \( c \): \( c = \frac{1+\sqrt{5}}{2} \) as before

Base Case:

- \( T(0) = T(1) = 1 \)
- \( k \cdot c^0 - 1 = k - 1 \) and \( k \cdot c^1 - 1 > k - 1 \checkmark \)
- We can hence select \( k = 2! \)

We proved \( T(n) \leq 2 \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n - 1 \). Hence \( T(n) \in O\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right) \).
Fibonacci Numbers: Closed-form Expression

Golden Ratio:

\[ \phi = \frac{1 + \sqrt{5}}{2} \approx 1.61803 \]

Closed-form Expression:

\[ F_n = \frac{\phi^n - (-\phi)^{-n}}{\sqrt{5}}. \]

Why not compute Fibonacci Numbers this way?

- Floating point operations, precision
- Large numbers involved
- Impractical
Experiments

Exponential growth

Dr. Christian Konrad
Lecture 15: The Fibonacci Numbers 11 / 15
Experiments

Logarithmic scale to base \((1+\sqrt{5})/2\)
Why is this Algorithm so slow?

Discussion:
- We compute solutions to subproblems many times ($T(i)$ is computed often, for most values of $i$)
- How can we avoid this?

Dynamic Programming!
Dynamic Programming (will be discussed in more detail later)

- Store solutions to subproblems in a table
- Compute table bottom up

```
 Require: Integer \( n \geq 0 \)
 if \( n \leq 1 \) then
     return \( n \)
 else
     \( A \leftarrow \) array of size \( n \)
     \( A[0] \leftarrow 1, A[1] \leftarrow 1 \)
     for \( i \leftarrow 2 \ldots n \) do
     return \( A[n] \)

\textbf{DynPrgFib}(n)
```
Analysis:
- DynPrgFib() runs in time $O(n)$
- It uses space $\Theta(n)$ since it uses an array of size $n$

Can we reduce the space to $O(1)$?

Improvement:
- Observe that when $T(i)$ is computed, the values $T(1), T(2), \ldots, T(i - 3)$ are no longer needed
- Only store the last two values of $T$
Improved Algorithm

\begin{algorithm}
\textbf{Require:} Integer \( n \geq 0 \)
\begin{align*}
\text{if } n \leq 1 \text{ then} & \\
\text{return } n \\
\text{else} & \\
\text{a } & \leftarrow 0 \\
\text{b } & \leftarrow 1 \\
\text{for } i & \leftarrow 2 \ldots n \text{ do} \\
\text{c } & \leftarrow a + b \\
\text{a } & \leftarrow b \\
\text{b } & \leftarrow c \\
\text{return } c
\end{align*}
\end{algorithm}

\textbf{Correctness:} via loop invariant!