Design of Algorithms
Short Tutorial on Recurrence Relations

Jeffrey A. Graves
Tennessee Tech. University
September 30, 2015
Short Tutorial on Recurrence Relations I

1. Sequences and Recurrence Relations
   - Sequences
   - Recurrence Relations
   - Method of Forward Substitutions
   - Method of Backward Substitutions

2. Common Recurrence Types
   - Decrease-by-One
   - Decrease-by-a-Constant-Factor
     - Smoothness Rule
   - Divide-and-Conquer
     - Master Theorem
A (numerical) sequence is an ordered list of numbers. We normally denote a sequence by a letter with an index surrounded by curly brackets, that is \( \{x_n\} \). Given a sequence \( x = \{x_n\} \), we denote the \( n \)th element in the sequence by \( x_n \) or \( x(n) \), and we call this a generic term.

Examples:

2, 4, 6, 8, 10, 12, \ldots \) (positive even integers)
0, 1, 1, 2, 3, 4, 8, \ldots \) (the Fibonacci numbers)
0, 1, 3, 6, 10, 15, \ldots \) (number of key comparisons in selection sort)
Defining A Sequence

Defining A Sequence
Defining A Sequence

- List Each Term
  0, 2, 4, 6, 8, 10, 12
Defining A Sequence

- List Each Term
  0, 2, 4, 6, 8, 10, 12

- Explicit Formula
  \( x(n) = 2n \)
Defining A Sequence

- List Each Term
  0, 2, 4, 6, 8, 10, 12

- Explicit Formula
  \[ x(n) = 2n \]

- Recurrence Equation / Recurrence Relation
  \[ x(n) = x(n-1) + 2 \text{ for } n > 0 \]
  \[ x(0) = 0 \]
A recurrence relation (or recurrence equation) is an equation that recursively defines a sequence. Once one or more initial terms are given while each further term of the sequence are defined as a function of the preceding terms.

A general solution to a recurrence relation is a closed form formula representing the recurrence equation that disregards the initial conditions. A particular solution is a closed form formula representing the recurrence equation and satisfies the initial conditions.
Example: General and Particular Solutions

Consider the following recurrence relation:

\[ x(n) = x(n-1) + n \text{ for all } n > 0 \]
\[ x(0) = 0 \]
Example: General and Particular Solutions

Consider the following recurrence relation:

\[ x(n) = x(n - 1) + n \text{ for all } n > 0 \]
\[ x(0) = 0 \]

The general solution to this recurrence relation is

\[ x(n) = c + \frac{n(n + 1)}{2}, \]

while the particular solution is

\[ x(n) = \frac{n(n + 1)}{2}. \]
Two Problems

We Have Two Problems To Consider
Two Problems

We Have Two Problems To Consider

- Finding a Closed Form Formula
  
  \[ x(n) = x(n - 1) + 2 \text{ for } n > 0 \]
  \[ x(0) = 0 \]
Two Problems

We Have Two Problems To Consider

- Finding a Closed Form Formula
  \[ x(n) = x(n - 1) + 2 \text{ for } n > 0 \]
  \[ x(0) = 0 \]

- Proving a Formula is Correct
  \[ x(n) = 2n \]
Proving a Formula is Correct

Two Ways To Prove a Formula is Correct

- Proof By Substitution
- Proof By Mathematical Induction
There is no universal method to solve every recurrence relation.

- Method of Forward Substitutions
- Method of Backward Substitutions
- Using Characteristic Equations
Method of Forward Substitutions: use the recurrence relation to generate a few terms in the sequence and try to identify a pattern. Express the pattern as a closed form formula.
Example: Method of Forward Substitutions

Find the particular solution for the following recurrence relation:

\[ x(n) = x(n-1) + 2 \text{ for } n > 0 \]
\[ x(0) = 0 \]

\[ x(0) = 0 \]
\[ x(1) = x(0) + 2 = 0 + 2 = 2 \]
\[ x(2) = x(1) + 2 = 2 + 2 = 4 \]
\[ x(3) = x(2) + 2 = 4 + 2 = 6 \]

It looks like \( x(n) = 2n \).
Method of Backward Substitutions: start by expressing $x(n)$ in terms of $x(n-1)$, and then in terms of $x(n-2)$ and then in terms of $x(n-3)$, etc., and try to identify a pattern. Express the pattern as a closed form formula.
Example: Method of Backward Substitutions

Find the particular solution for the following recurrence relation:

\[ x(n) = x(n - 1) + 2 \quad \text{for} \quad n > 0 \]
\[ x(0) = 0 \]

\[ x(n) = x(n - 1) + 2 \]
\[ = x(n - 2) + 4 \]
\[ = x(n - 3) + 6 \]
\[ = x(n - 4) + 8 \]

It looks like \( x(n) = 2n \).
Decrease-by-One

Time efficiency of decrease-by-one algorithms typically have the form:

\[ T(n) = T(n-1) + f(n) \]  

(1)
Decrease-by-One

We can solve by backward substitution:

\[ T(n) = T(n - 1) + f(n) \]
Decrease-by-One

We can solve by backward substitution:

\[ T(n) = T(n-1) + f(n) \]
\[ = (T(n-2) + f(n-1)) + f(n) \]
Decrease-by-One

We can solve by backward substitution:

\[ T(n) = T(n-1) + f(n) \]
\[ = (T(n-2) + f(n-1)) + f(n) \]
\[ = ((T(n-3) + f(n-2)) + f(n-1)) + f(n) \]

\[ = ... \]
\[ = T(0) + \sum_{i=1}^{n} f(i) \]
Decrease-by-One

We can solve by backward substitution:

\[ T(n) = T(n - 1) + f(n) \]
\[ = (T(n - 2) + f(n - 1)) + f(n) \]
\[ = ((T(n - 3) + f(n - 2)) + f(n - 1)) + f(n) \]
\[ = \ldots \]
\[ = T(0) + \sum_{i=1}^{n} f(i) \]
Decrease-by-One

We can solve by backward substitution:

\[ T(n) = T(n - 1) + f(n) \]
\[ = (T(n - 2) + f(n - 1)) + f(n) \]
\[ = ((T(n - 3) + f(n - 2)) + f(n - 1)) + f(n) \]
\[ = \ldots \]
\[ = T(0) + \sum_{i=1}^{n} f(i) \]
Decrease-by-a-Constant-Factor

Time efficiency of decrease-by-a-constant algorithms typically have the form:

\[ T(n) = T\left(\frac{n}{b}\right) + f(n) \]  

Most of the time \( b = 2 \). If \( n = b^k \) for some nonnegative integer \( k \), then we can find a closed form formula for \( T(n) \). What if \( n \neq b^k \)?
Definition: Eventually Nondecreasing

A nonnegative function $f : \mathbb{N} \to \mathbb{R}$ defined on the natural numbers is said to be eventually nondecreasing if there exists an $x_0$ such that $f(x_1) \leq f(x_2)$ for all $x_0 \leq x_1 \leq x_2$.

All polynomials of odd degree with a nonnegative leading coefficient are eventually nondecreasing.
**Definition: Smooth**

A nonnegative function $f : \mathbb{N} \rightarrow \mathbb{R}$ defined on the natural numbers is said to be smooth if it is eventually nondecreasing and $f(2n) \in \Theta(f(n))$. 

All polynomials that are nondecreasing are smooth. No exponential function is smooth.
Definition: Smooth

A nonnegative function \( f : \mathbb{N} \rightarrow \mathbb{R} \) defined on the natural numbers is said to be smooth if it is eventually nondecreasing and \( f(2n) \in \Theta(f(n)) \).

All polynomials that are nondecreasing are smooth. No exponential functions is smooth.
Smoothness Example

Show that $f(n) = n \log(n)$ is smooth.
Smoothness Example

Show that \( f(n) = n \log(n) \) is smooth. Observe that

\[
\begin{align*}
  f(2n) &= 2n \log(2n) \\
         &= 2n (\log(2) + \log(n)) \\
         &= 2 \log(2) n + 2n \log(n),
\end{align*}
\]

where \( 2 \log(2) n + 2n \log(n) \in \Theta(n \log(n)) \).
Theorem: Smoothness Rule

**Theorem (Smoothness Rule)**

Let $T(n)$ be an eventually nondecreasing function and $f(n)$ be a smooth function. If $T(n) \in \Theta(f(n))$ for values of $n$ that are powers of $b \geq 2$, then $T(n) \in \Theta(f(n))$.

Similar results hold for $O$ and $\Omega$. 
Time efficiency of divide-and-conquer algorithms typically have the form:

$$T(n) = aT(n/b) + f(n),$$

(7)

where $a \geq 1$ and $b \geq 2$. Here, there are $a$ subproblems of size $n/b$ to be solved, and the time to decompose the problem and recombine the subsolutions is $f(n)$. 
Master Theorem

Theorem

Let $F(n)$ be an eventually nondecreasing function that satisfies the recurrence

$$T(n) = aT(n/b) + f(n) \quad n = b, k = 1, 2, \ldots$$
$$T(1) = c$$

where $a \geq 1$, $b \geq 2$, and $c > 0$. If $f(n) \in \Theta(n^d)$ where $d \geq 0$, then

$$T(n) \in \begin{cases} 
\Theta(n^d) & \text{if } a < b^d, \\
\Theta(n^d \log n) & \text{if } a = b^d, \\
\Theta(n^{\log_b a}) & \text{if } a > b^d.
\end{cases}$$

(9)