Recurrences and Recursion

R. Sekar
Recursion is one of the most versatile techniques in programming as well as algorithm design.

Closely related to induction:
- Consists of a base case and recursive case, similar to base and inductive steps.
- Correctness of recursive algorithms is proved by induction.

Example: Computing Fibonacci numbers:
Base case(s): \( F(0) = 0, F(1) = 1 \)
Recursive case: \( F(n) = F(n - 1) + F(n - 2) \)
Uses of Recursion

**Recurrences:** Typically used in the context of algorithm analysis

- **Base Case:** $T(0) = 1$
- **Recursive Case:** $T(n) = 2T(n/2) + n$

**Recursive functions:**

- **Base Case:** $\text{sum}(0) = 0$
- **Recursive Case:** $\text{sum}(n) = n + \text{sum}(n-1)$
Uses of Recursion

Recurrences: Typically used in the context of algorithm analysis

Base Case: \( T(0) = 1 \)

Recursive Case: \( T(n) = 2T(n/2) + n \)

Recursive functions: Used in programming

Base Case: \( \text{sum}(0) = 0 \)

Recursive Case: \( \text{sum}(n) = n + \text{sum}(n - 1) \)
Uses of Recursion

Recurrences: Typically used in the context of algorithm analysis

Base Case: \( T(0) = 1 \)

Recursive Case: \( T(n) = 2T(n/2) + n \)

Recursive functions: Used in programming

Base Case: \( sum(0) = 0 \)

Recursive Case: \( sum(n) = n + sum(n - 1) \)

Algebraic datatypes: Using mathematical data types to represent complex data

Base Case: Finite sets

- subsets of \( \mathbb{Z} \) and \( \mathbb{R} \), arbitrary enumerated sets, ...

Recursive Case: Type constructor operators

- Cartesian product and Set union
16.4. Solving Linear Recurrences

16.4.2 The Towers of Hanoi

According to legend, there is a temple in Hanoi with three posts and 64 gold disks of different sizes. Each disk has a hole through the center so that it fits on a post.

In the misty past, all the disks were on the first post, with the largest on the bottom and the smallest on top, as shown in Figure 16.1.

Monks in the temple have labored through the years since to move all the disks to one of the other two posts according to the following rules:

- The only permitted action is removing the top disk from one post and dropping it onto another post.
- A larger disk can never lie above a smaller disk on any post.

So, for example, picking up the whole stack of disks at once and dropping them on another post is illegal. That's good, because the legend says that when the monks complete the puzzle, the world will end!

To clarify the problem, suppose there were only 3 gold disks instead of 64. Then the puzzle could be solved in 7 steps as shown in Figure 16.2.

The questions we must answer are, “Given sufficient time, can the monks succeed?” If so, “How long until the world ends?” And, most importantly, “Will this happen before the final exam?”

A Recursive Solution

The Towers of Hanoi problem can be solved recursively. As we describe the procedure, we’ll also analyze the minimum number of steps required to solve the n-disk problem. For example, some experimentation shows that $t_1 = 1$ and $t_2 = 3$.

The procedure illustrated above uses 7 steps, which shows that $t_3$ is at most 7.

The recursive solution has three stages, which are described below and illustrated in Figure 16.3. For clarity, the largest disk is shaded in the figures.

Stage 1. Move the top $n-1$ disks from the first post to the second using the solution for $n-1$ disks. This can be done in $t_{n-1}$ steps.

Goal: Move all disks from one post to another.

Rules:

- Only the top-most disk can be moved.
- No disk can be placed on a smaller disk.

Questions:

- How do you solve the puzzle?
- How many moves will be needed?
Tower of Hanoi Problem: Example with Three Disks

Figure 16.2 The 7-step solution to the Towers of Hanoi problem when there are $n$ disks.

Figure 16.3 A recursive solution to the Towers of Hanoi problem.
Chapter 16 Generating Functions

Figure 16.2: The 7-step solution to the Towers of Hanoi problem when there are $nD3$ disks.

Figure 16.3: A recursive solution to the Towers of Hanoi problem.

A Recursive Algorithm for Tower of Hanoi Problem

MoveStack($n$, 1, 3):
MoveStack($n-1$, 1, 2)
MoveDisk($n$, 1, 3)
MoveStack($n-1$, 2, 3)

Base Case: MoveStack(1, x, y)
A Recursive Algorithm for Tower of Hanoi Problem

MoveStack\( (n, 1, 3) \):
- MoveStack\( (n - 1, 1, 2) \)
- MoveDisk\( (n, 1, 3) \)
- MoveStack\( (n - 1, 2, 3) \)
A Recursive Algorithm for Tower of Hanoi Problem

**MoveStack**($n$, 1, 3):
- **MoveStack**($n - 1$, 1, 2)
- **MoveDisk**($n$, 1, 3)
- **MoveStack**($n - 1$, 2, 3)

**Base Case:**
**MoveStack**($1$, $x$, $y$)
- **MoveDisk**($1$, $x$, $y$)
A Recurrence for the Runtime of Towers of Hanoi Algorithm

MoveStack(n, 1, 3):
  - MoveStack(n − 1, 1, 2)
  - MoveDisk(n, 1, 3)
  - MoveStack(n − 1, 2, 3)

Base Case:
MoveStack(1, x, y)
  - MoveDisk(1, x, y)
A Recurrence for the Runtime of Towers of Hanoi Algorithm

MoveStack(n, 1, 3):
- MoveStack(n − 1, 1, 2)
- MoveDisk(n, 1, 3)
- MoveStack(n − 1, 2, 3)

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

Base Case:
MoveStack(1, x, y)
- MoveDisk(1, x, y)
A Recurrence for the Runtime of Towers of Hanoi Algorithm

MoveStack(n, 1, 3):
- MoveStack(n − 1, 1, 2)
- MoveDisk(n, 1, 3)
- MoveStack(n − 1, 2, 3)

\[ T(n) = 2T(n − 1) + 1 \]

Base Case:
MoveStack(1, x, y)
- MoveDisk(1, x, y)

\[ T(1) = 1 \]
Solving Recurrences: Plug and Chug

- Expand the recurrence out for a few steps
- Identify the pattern
- Guess a solution based on the pattern
- Check the solution for a few small values of $n$
- Verify using induction
Plug and Chug for Tower of Hanoi Recurrence
Recurrences for Analyzing Runtimes: Exponentiation

\[ exp(x, n) = \begin{cases} 
1, & \text{if } n = 0 \\
x \times exp(x, n - 1), & \text{otherwise}
\end{cases} \]
Recurrences for Analyzing Runtimes: Exponentiation

\[ \exp(x, n) = \begin{cases} 
1, & \text{if } n = 0 \\
x \times \exp(x, n - 1), & \text{otherwise}
\end{cases} \]

\[ T(1) = 1 \]
\[ T(n) = T(n - 1) + 1 \]
Recurrences for Analyzing Runtimes: Fast Exponentiation

\[ fexp(x, n) = \begin{cases} 
  1, & \text{if } n = 0 \\
  x, & \text{if } n = 1 \\
  fexp(x \times x, n/2), & \text{if } n \text{ is even} \\
  fexp(x \times x, n/2) \times x, & \text{if } n \text{ is odd} 
\end{cases} \]
Recurrences for Analyzing Runtimes: Fast Exponentiation

\[ f_{\text{exp}}(x, n) = \begin{cases} 
1, & \text{if } n = 0 \\
x, & \text{if } n = 1 \\
f_{\text{exp}}(x \times x, n/2), & \text{if } n \text{ is even} \\
f_{\text{exp}}(x \times x, n/2) \times x, & \text{if } n \text{ is odd}
\end{cases} \]

\[ T(1) = 1 \]

\[ T(n) = T(n/2) + 1 \]
Solving Linear Recurrences

- **Homogeneous** linear recurrences are of the form
  \[ f(n) = \sum_{i=1}^{d} a_i f(n - i) \]

- Example: Fibonacci series \( F(n) = F(n - 1) + F(n - 2) \)
Solving Linear Recurrences

- **Homogeneous** linear recurrences are of the form

\[
f(n) = \sum_{i=1}^{d} a_i f(n - i)
\]

- Example: Fibonacci series \( F(n) = F(n - 1) + F(n - 2) \)

- They are known to have an *exponential* solution \( f(n) = x^n \) for some \( x \)
  - Substitute this solution into the recurrence and solve for \( x \):
Solving Linear Recurrences

- **Homogeneous** linear recurrences are of the form
  \[ f(n) = \sum_{i=1}^{d} a_if(n - i) \]

- Example: Fibonacci series \( F(n) = F(n - 1) + F(n - 2) \)
- They are known to have an **exponential** solution \( f(n) = x^n \) for some \( x \)
  - Substitute this solution into the recurrence and solve for \( x \):
    \[
    x^n = \sum_{i=1}^{d} a_ix^{n-i}
    \]
    \[
    x^d = \sum_{i=1}^{d} a_ix^{d-i} \quad \text{(Dividing all terms by } x^{n-d})
    \]
    \[
    \sum_{i=0}^{d} a_ix^{d-i} = 0 \quad \text{(Rearrange terms to arrive at a polynomial, with } a_0 = 1)\]
Find the roots $r_1, ..., r_d$ of this polynomial $\sum_{i=0}^{d} a_i x^{d-i} = 0$.

Solve for $k_i$ using known values for $f(0)$ through $f(d-1)$.

Note: if the polynomial has fewer than $d$ roots, the general form of the solution gets more complicated — we will ignore this case here.
Find the roots $r_1, \ldots, r_d$ of this polynomial
\[
\sum_{i=0}^{d} a_i x^{d-i} = 0
\]
The general solution to the recurrence is
\[
f(n) = \sum_{i=1}^{d} k_i r_i^n
\]
Find the roots $r_1, \ldots, r_d$ of this polynomial $\sum_{i=0}^{d} a_i x^{d-i} = 0$.

The general solution to the recurrence is

$$f(n) = \sum_{i=1}^{d} k_i r_i^n$$

Solve for $k_i$ using known values for $f(0)$ through $f(d-1)$. 

Note: if the polynomial has fewer than $d$ roots, the general form of the solution gets more complicated — we will ignore this case here.
Find the roots $r_1, \ldots, r_d$ of this polynomial $\sum_{i=0}^{d} a_i x^{d-i} = 0$

The general solution to the recurrence is

$$f(n) = \sum_{i=1}^{d} k_i r_i^n$$

Solve for $k_i$ using known values for $f(0)$ through $f(d-1)$.

Note: if the polynomial has fewer than $d$ roots, the general form of the solution gets more complicated — we will ignore this case here.
Solving Homogeneous Linear Recurrences: Fibonacci Example

\[ f(n) = f(n - 1) + f(n - 2) \]
Solving Homogeneous Linear Recurrences: Fibonacci Example

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

1. Substitute \( f(n) = x^n \) in this equation, simplify to get characteristic equation \( x^2 = x + 1 \)
Solving Homogeneous Linear Recurrences: Fibonacci Example

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

1. Substitute \( f(n) = x^n \) in this equation, simplify to get \textit{characteristic equation} \( x^2 = x + 1 \)

2. Solve this quadratic equation to obtain roots \( p = \frac{1 + \sqrt{5}}{2} \) and \( q = \frac{1 - \sqrt{5}}{2} \)

3. By the homogeneous linear recurrence method, the general solution is \( f(n) = k_1 p^n + k_2 q^n \)

4. Plug in \( f(0) = 0 \) and \( f(1) = 1 \) to obtain the following equations:
   - \( k_1 p^0 + k_2 q^0 = k_1 + k_2 = f(0) = 0 \) which means \( k_2 = -k_1 \)
   - \( k_1 p^1 + k_2 q^1 = k_1 \left( \frac{1 + \sqrt{5}}{2} \right) + k_2 \left( \frac{1 - \sqrt{5}}{2} \right) = (k_1 + k_2)/2 + \sqrt{5}(k_1 - k_2)/2 = f(1) = 1 \)
   - Substituting \( k_2 = -k_1 \) in this equation and simplifying, we get \( k_1 = 1/\sqrt{5} \).

5. Thus, the solution is

\[ f(n) = \frac{1}{\sqrt{5}} \left( \frac{1 + \sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left( \frac{1 - \sqrt{5}}{2} \right)^n \]
Observations about Fibonacci Recurrence Solution

- All Fibonacci numbers are integers — it is mind-boggling that its closed form solution contains not just fractions, but *irrational* numbers!
- No wonder that this solution was unknown for six centuries!
Observations about Fibonacci Recurrence Solution

- All Fibonacci numbers are integers — it is mind-boggling that its closed form solution contains not just fractions, but \textit{irrational} numbers!
- No wonder that this solution was unknown for six centuries!
- Note that $|q| = \left| \frac{1 - \sqrt{5}}{2} \right| = 0.6180 < 1$ so $q^n$ rapidly approaches zero. For instance, $q^{20} \approx 0.00006$, and the error in $f(n)$ due to ignoring $q$ is less than one in $10^{-8}$.
Observations about Fibonacci Recurrence Solution

- All Fibonacci numbers are integers — it is mind-boggling that its closed form solution contains not just fractions, but *irrational* numbers!
- No wonder that this solution was unknown for six centuries!
- Note that $|q| = \frac{|1-\sqrt{5}|}{2} = 0.6180 < 1$ so $q^n$ rapidly approaches zero. For instance, $q^{20} \approx 0.00006$, and the error in $f(n)$ due to ignoring $q$ is less than one in $10^{-8}$.
- So, for larger $n$, $f(n)$ is determined almost entirely by the first term $\frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2}\right)^n$
- $p^n/\sqrt{5}$ is very close to an integer value, although $p$ is irrational!
Observations about Fibonacci Recurrence Solution

- All Fibonacci numbers are integers — it is mind-boggling that its closed form solution contains not just fractions, but *irrational* numbers!
- No wonder that this solution was unknown for six centuries!
- Note that \( |q| = \left| \frac{1 - \sqrt{5}}{2} \right| = 0.6180 < 1 \) so \( q^n \) rapidly approaches zero. For instance, \( q^{20} \approx 0.00006 \), and the error in \( f(n) \) due to ignoring \( q \) is less than one in \( 10^{-8} \).
- So, for larger \( n \), \( f(n) \) is determined almost entirely by the first term \( \frac{1}{\sqrt{5}} \left( \frac{1 + \sqrt{5}}{2} \right)^n \)
- \( p^n / \sqrt{5} \) is very close to an integer value, although \( p \) is irrational!
- The ratio between successive Fibonacci numbers converges to \( p = 1.618 \), which is called the *golden ratio*
Asymptotic Complexity

Expressing complexity in terms of “number of steps” is a simplification:
- Each such operation may in fact take a different amount of time.
- But it is too complex to worry about the details, esp. because they differ across programming languages, processor types, etc.
Expressing complexity in terms of “number of steps” is a simplification

- Each such operation may in fact take a different amount of time
- But it is too complex to worry about the details, esp. because they differ across programming languages, processor types, etc.

Why not simplify further?

- Capture just the growth rate of $T(n)$ as a function of $n$
- Ignore constant factors
  - No need to count operations in a loop (their number should be bounded by a constant)
- Ignore exceptions from the formula for small values of $n$
Asymptotic Complexity: Big-O notation

**Definition**

Given functions \( f, g : \mathbb{R} \to \mathbb{R} \), we say \( f = O(g) \), i.e., \( f \) grows no faster than \( g \), iff

\[
\lim_{x \to \infty} \frac{f(x)}{g(x)} < c \text{ for some constant } c
\]
Big-$O$ notation: Examples

- $10n = O(n)$
- $0.0001n^3 + n = O(n^3)$
- $2^n + 10^n + n^2 + 2 = O(10^n)$
- $0.0001n \log n + 10000n = O(n \log n)$
Solving Divide-and-Conquer Recurrences: Master Theorem

If $T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$ for constants $a > 0$, $b > 1$, and $d \geq 0$, then

$$T(n) = \begin{cases} 
O(n^d), & \text{if } d > \log_b a \\
O(n^d \log n), & \text{if } d = \log_b a \\
O(n^{\log_b a}), & \text{if } d < \log_b a 
\end{cases}$$
Solving Recurrences: Examples Using Master Theorem

\[ T(n) = 2T\left(\frac{n}{2}\right) + n \]

\[ T(n) = aT\left(\frac{n}{b}\right) + O(n^d) \]

\[ T(n) = \begin{cases} 
  O(n^d), & \text{if } d > \log_b a \\
  O(n^d \log n), & \text{if } d = \log_b a \\
  O(n^{\log_b a}), & \text{if } d < \log_b a 
\end{cases} \]
T(n) = 4T(n/2) + n^3

T(n) = aT(n/b) + O(n^d)

T(n) = \begin{cases} 
O(n^d), & \text{if } d > \log_b a \\
O(n^d \log n), & \text{if } d = \log_b a \\
O(n^{\log_b a}), & \text{if } d < \log_b a 
\end{cases}
T(n) = 3T(n/2) + n

\[ T(n) = aT\left(\frac{n}{b}\right) + O(n^d) \]

\[ T(n) = \begin{cases} 
O(n^d), & \text{if } d > \log_b a \\
O(n^d \log n), & \text{if } d = \log_b a \\
O(n^{\log_b a}), & \text{if } d < \log_b a 
\end{cases} \]