2.3 Recursion

Overview

What is recursion? When one function calls itself directly or indirectly.

Why learn recursion?
- New mode of thinking.
- Powerful programming paradigm.

Many computations are naturally self-referential.
- Mergesort, FFT, gcd, depth-first search.
- Linked data structures.
- A folder contains files and other folders.

Closely related to mathematical induction.

Greatest Common Divisor

Gcd. Find largest integer that evenly divides into \( p \) and \( q \).

Ex. \( \text{gcd}(4032, 1272) = 24 \).

\[
\begin{align*}
4032 & = 2^6 \times 3^2 \times 7^1 \\
1272 & = 2^3 \times 3^1 \times 53^1 \\
gcd & = 2^3 \times 3^1 = 24
\end{align*}
\]

Applications.
- Simplify fractions: \( 1272/4032 = 53/168 \).
- RSA cryptosystem.

Euclid’s algorithm. [Euclid 300 BCE]

\[
\text{gcd}(p,q) = \begin{cases} 
  p, & \text{if } q=0 \\
  \text{gcd}(q, p \% q), & \text{otherwise}
\end{cases}
\]

\[
\begin{align*}
\text{gcd}(4032, 1272) & = \text{gcd}(1272, 216) \\
& = \text{gcd}(216, 192) \\
& = \text{gcd}(192, 24) \\
& = \text{gcd}(24, 0) \\
& = 24.
\end{align*}
\]
Greatest Common Divisor

Gcd. Find largest integer d that evenly divides into p and q.

\[
gcd(p, q) = \begin{cases} 
  p & \text{if } q = 0 \\
  gcd(q, p \% q) & \text{otherwise}
\end{cases}
\]

- base case
- reduction step, converges to base case

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>p % q</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

\[\text{gcd} = x\]

p = 8x
q = 3x
\[\text{gcd}(p, q) = x\]

Java implementation.

```java
public static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

Greatest Common Divisor

Gcd. Find largest integer d that evenly divides into p and q.

\[
gcd(p, q) = \begin{cases} 
  p & \text{if } q = 0 \\
  gcd(q, p \% q) & \text{otherwise}
\end{cases}
\]

- base case
- reduction step, converges to base case

Recursive Graphics
H-tree of order n.
- Draw an H.
- Recursively draw 4 H-trees of order n-1, one connected to each tip.

```java
public class Htree {
    public static void draw(int n, double sz, double x, double y) {
        if (n == 0) return;
        double x0 = x - sz/2, x1 = x + sz/2;
        double y0 = y - sz/2, y1 = y + sz/2;
        StdDraw.line(x0, y, x1, y);
        StdDraw.line(x0, y0, x0, y1);
        StdDraw.line(x1, y0, x1, y1);
        draw(n-1, sz/2, x0, y0);
        draw(n-1, sz/2, x1, y0);
        draw(n-1, sz/2, x1, y1);
    }
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        draw(n, .5, .5, .5);
    }
}
```

Animated H-tree.
Animated H-tree. Pause for 1 second after drawing each H.

Towers of Hanoi

Towers of Hanoi

Move all the discs from the leftmost peg to the rightmost one.
- Only one disc may be moved at a time.
- A disc can be placed either on empty peg or on top of a larger disc.

Towers of Hanoi demo

Edouard Lucas (1883)

Towers of Hanoi Legend

Q. Is world going to end (according to legend)?
- 64 golden discs on 3 diamond pegs.
- World ends when certain group of monks accomplish task.

Q. Will computer algorithms help?

Towers of Hanoi: Recursive Solution

```java
public class TowersOfHanoi {
    public static void moves(int n, boolean left) {
        if (n == 0) return;
        moves(n-1, !left);
        if (left) System.out.println(n + " left");
        else System.out.println(n + " right");
        moves(n-1, !left);
    }

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        moves(N, true);
    }
}
```

moves(n, true): move discs 1 to n one pole to the left
moves(n, false): move discs 1 to n one pole to the right

smallest disc
Towers of Hanoi: Recursive Solution

Remarkable properties of recursive solution.
- Takes \(2^n - 1\) moves to solve \(n\) disc problem.
- Sequence of discs is same as subdivisions of ruler.
- Every other move involves smallest disc.

Recursive algorithm yields non-recursive solution!
- Alternate between two moves: 
  - move smallest disc to right if \(n\) is even
  - make only legal move not involving smallest disc

Recursive algorithm may reveal fate of world.
- Takes 585 billion years for \(n = 64\) (at rate of 1 disc per second).
- Reassuring fact: any solution takes at least this long!

Divide-and-Conquer

Divide-and-conquer paradigm.
- Break up problem into smaller subproblems of same structure.
- Solve subproblems recursively using same method.
- Combine results to produce solution to original problem.

Many important problems succumb to divide-and-conquer.
- FFT for signal processing.
- Parsers for programming languages.
- Multigrid methods for solving PDEs.
- Quicksort and mergesort for sorting.
- Hilbert curve for domain decomposition.
- Quad-tree for efficient N-body simulation.
- Midpoint displacement method for fractional Brownian motion.
Fibonacci Numbers

Fibonacci numbers. 0, 1, 2, 3, 5, 8, 13, 21, 34, ...

Fibonacci numbers. 0, 1, 2, 3, 5, 8, 13, 21, 34, ...

Fibonacci numbers. 0, 1, 2, 3, 5, 8, 13, 21, 34, ...

A Possible Pitfall With Recursion

A natural for recursion?

public static long F(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return F(n-1) + F(n-2);
}
Q. Is this an efficient way to compute $F(50)$?

A. No, no, no! This code is **spectacularly inefficient**.

```java
public static long F(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return F(n-1) + F(n-2);
}
```

F(50) is called once.
F(49) is called once.
F(48) is called 2 times.
F(47) is called 3 times.
F(46) is called 5 times.
F(45) is called 8 times.
...
F(1) is called 12,586,269,025 times.

Recursion tree for naïve Fibonacci function

Q. Is this a more efficient way to compute $F(50)$?

A. Yes. This code does it with 50 additions.

```java
FYI: classic math
$F(n) = \frac{\phi^n - (1 - \phi)^n}{\sqrt{5}}$
$\phi = \frac{1 + \sqrt{5}}{2} 
\approx 1.618$

public static long F(int n) {
    if (n == 0) return 0;
    long[] F = new long[n+1];
    F[0] = 0;
    F[1] = 1;
    for (int i = 2; i <= n; i++)
        F[i] = F[i-1] + F[i-2];
    return F[n];
}
```

Lesson. Don’t use recursion to engage in exponential waste.

**Context.** This is a special case of an important programming technique known as **dynamic programming** (stay tuned).

**Summary**

**How to write simple recursive programs?**
- Base case, reduction step.
- Trace the execution of a recursive program.
- Use pictures.

**Why learn recursion?**
- New mode of thinking.
- Powerful programming tool.

**Divide-and-conquer.** Elegant solution to many important problems.