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.

• Binary search, mergesort, FFT, GCD.
• Linked data structures.
• A folder contains files and other folders.

Closely related to mathematical induction.

Mathematical Induction

Mathematical induction. Prove a statement involving an integer N by

• base case: Prove it for some specific N (usually 0 or 1).
• induction step: Assume it to be true for all positive integers less than N, use that fact to prove it for N.

Ex. Sum of the first N odd integers is N².

Base case: True for N = 1.

Induction step:

• Let T(N) be the sum of the first N odd integers: 1 + 3 + 5 + ... + (2N - 1).
• Assume that T(N-1) = (N-1)².
• T(N) = T(N-1) + (2N - 1)
  = (N-1)² + (2N - 1)
  = N² - 2N + 1 + (2N - 1)
  = N²
Recursive Program

Recursive Program. Implement a function having integer arguments by
• base case: Do something specific in response to “base” argument values.
• reduction step: Assume the function works for all smaller argument
  values, and use the function to implement itself for general argument
  values.

public static String convert(int x)
{
  if (x == 1) return "1";
  return convert(x/2) + (x % 2);
}

Ex 1. Convert positive int to binary String.
Base case: return "1" for x = 1.
Reduction step:
• convert x/2 to binary
• append "0" if x even
• append "1" if x odd

public class Binary
{
  public static String convert(int x)
  {
    if (x == 1) return "1";
    return convert(x/2) + (x % 2);
  }

  public static void main(String[] args)
  {
    int x = Integer.parseInt(args[0]);
    System.out.println(convert(x));
  }
}

% java Binary 6
110
% java Binary 37
100101
% java Binary 999999
11110100001000111111

convert(6)

public class Binary
{
  public static String convert(int x)
  {
    if (x == 0) return "";
    return convert(x/2) + (x % 2);
  }

  public static void main(String[] args)
  {
    int x = Integer.parseInt(args[0]);
    System.out.println(convert(x));
  }
}

% java Binary 6
110
% java Binary 999999
11110100001000111111

convert(6)
Recursion vs. Iteration

Every program with 1 recursive call corresponds to a loop.

Reasons to use recursion:
- code more compact
- easier to understand
- easier to reason about correctness
- easy to add multiple recursive calls (stay tuned)

Reasons not to use recursion: (stay tuned)

public static String convert(int x) {
    if (x == 1) return "1";
    return convert(x/2) + (x % 2);
}

public static String convertNR(int x) {
    String s = "1";
    while (x > 1) {
        s = (x % 2) + s;
        x = x/2;
    } return s;
}

Greatest Common Divisor

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

Ex. gcd(4032, 1272) = 24.

4032 = 2^6 * 3^2 * 7
1272 = 2^3 * 3^1 * 53^1
gcd = 2^3 * 3^1 = 24

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

Euclid’s Algorithm

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

Euclid’s algorithm. [Euclid 300 BCE]

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

gcd(4032, 1272) = gcd(1272, 216)
    = gcd(216, 192)
    = gcd(192, 24)
    = gcd(24, 0)
    = 24.

Gcd(p, q) = gcd(3x, 2x) = x
Euclid’s Algorithm

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

$$gcd(p, q) = \begin{cases} 
  p & \text{if } q = 0 \\
  gcd(q, p \mod q) & \text{otherwise}
\end{cases}$$

Recursive program

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

Possible debugging challenges with recursion

Missing base case.

```java
public static double BAD(int N)
{
    return BAD(N-1) + 1.0/N;
}
```

No convergence guarantee.

```java
public static double BAD(int N)
{
    if (N == 1)
        return 1.0;
    return BAD(1 + N/2) + 1.0/N;
}
```

Both lead to INFINITE RECURSIVE LOOP (bad news).

So that you can recognize and deal with it if it later happens to you.
Collatz Sequence

Collatz sequence.
• If n is 1, stop.
• If n is even, divide by 2.
• If n is odd, multiply by 3 and add 1.

Ex. 35 106 53 160 80 40 20 10 5 16 8 4 2 1.

No one knows whether or not this function terminates for all N (!)
[usually we decrease N for all recursive calls]

```java
public static void collatz(int N) {
    StdOut.print(N + " ");
    if (N == 1) return;
    if (N % 2 == 0) collatz(N / 2);
    else collatz(3*N + 1);
}
```
**Htree**

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

**Htree in Java**

```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, x0, y1);
        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.** 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.

Moves all discs from the leftmost peg to the rightmost one.

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?

```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
Towers of Hanoi: Recursive Solution

```java
java TowersOfHanoi 3
1 left
2 right
1 left
3 left
1 left
2 right
1 left
```

```java
java TowersOfHanoi 4
1 right
2 left
1 right
3 right
2 left
1 right
4 left
1 right
2 left
1 right
3 right
1 right
2 left
1 right
```

Towers of Hanoi: Recursion Tree

```
  n, left
 /     \
3, true   2, false
 /     \   /     \ 
1, true 1, true 1, true 1, true
 /     \   /     \   /     \   /     \ 
1 left 2 right 1 left 3 left 1 left 2 right 1 left
```

Towers of Hanoi: Properties of 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.

```
Divide et impera. Veni, vidi, vici. - Julius Caesar
```

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, 1, 2, 3, 5, 8, 13, 21, 34, ...

$$F_n = \begin{cases} 
0 & \text{if } n = 0 \\
1 & \text{if } n = 1 \\
F_{n-1} + F_{n-2} & \text{otherwise}
\end{cases}$$

A natural for recursion?

FYI (classical math):

$$F(n) = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}}$$

$$\phi = \text{golden ratio} \approx 1.618$$

Ex: $F(50) \approx 1.2 \times 10^{10}$

see much, much more at www.youtube.com/user/Vihart
Recursion Challenge 1 (difficult but important)

Is this an efficient way to compute $F(50)$?

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

Recursion Challenge 2 (easy and also important)

Is this an efficient way to compute $F(50)$?

```java
long[] F = new long[51];
F[0] = 0; F[1] = 1;
if (n == 1) return 1;
for (int i = 2; i <= 50; i++)
    F[i] = F[i-1] + F[i-2];
```

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.

Exponential time.
- Easy to specify recursive program that takes exponential time.
- Don’t do it unless you plan to (and are working on a small problem).