7. Recursion
7. Recursion

- Foundations
- A classic example
- Recursive graphics
- Avoiding exponential waste
- Dynamic programming
Overview

Q. What is recursion?

A. When something is specified in terms of itself.

Why learn recursion?

- Represents a new mode of thinking.
- Provides a powerful programming paradigm.
- Enables reasoning about correctness.
- Gives insight into the nature of computation.

Many computational artifacts are naturally self-referential.

- File system with folders containing folders.
- Fractal graphical patterns.
- Divide-and-conquer algorithms (stay tuned).
Example: Convert an integer to binary

Recursive program

To compute a function of $N$

- **Base case.** Return a value small $N$.
- **Reduction step.** Assuming that it works for smaller values of its argument, use the function to compute a return value for $N$.

```java
public class Binary {
    public static String convert(int N) {
        if (N == 1) return "1";
        return convert(N/2) + (N % 2);
    }
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        System.out.println(convert(N));
    }
}
```

Q. How can we be convinced that this method is correct?

A. Use *mathematical induction.*

% java Binary 6
110
% java Binary 37
100101
% java Binary 999999
11110100001000111111
Mathematical induction (quick review)

To prove a statement involving $N$

- **Base case.** Prove it for some specific values of $N$.
- **Induction step.** Assuming that the statement is true for all positive integers less than $N$, use that fact to prove it for $N$.

**Example**

The sum of the first $N$ odd integers is $N^2$.

**Base case.** True for $N = 1$.

**Induction step.** The $N$th odd integer is $2N - 1$.

Let $T_N = 1 + 3 + 5 + \ldots + (2N - 1)$ be the sum of the first $N$ odd integers.

- Assume that $T_{N-1} = (N - 1)^2$.
- Then $T_N = (N - 1)^2 + (2N - 1) = N^2$. 

An alternate proof
Proving a recursive program correct

Recursion

To compute a function of \( N \)
- **Base case.** Return a value for small \( N \).
- **Reduction step.** Assuming that it works for smaller values of its argument, use the function to compute a return value for \( N \).

Mathematical induction

To prove a statement involving \( N \)
- **Base case.** Prove it for small \( N \).
- **Induction step.** Assuming that the statement is true for all positive integers less than \( N \), use that fact to prove it for \( N \).

Recursive program

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

Correctness proof, by induction

convert() computes the binary representation of \( N \)
- **Base case.** Returns "1" for \( N = 1 \).
- **Induction step.** Assume that convert() works for \( N/2 \)
  1. Correct to append "0" if \( N \) is even, since \( N = 2(N/2) \).
  2. Correct to append "1" if \( N \) is odd since \( N = 2(N/2) + 1 \).
Mechanics of a function call

System actions when any function is called
• *Save environment* (values of all variables).
• *Initialize values* of argument variables.
• *Transfer control* to the function.
• *Restore environment* (and assign return value)
• *Transfer control* back to the calling code.

```java
class Binary {
    public static String convert(int N) {
        if (N == 1) return "1";
        return convert(N/2) + (N % 2);
    }
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        System.out.println(convert(N));
    }
}
```

```
convert(26)
if (N == 1) return "1";
return "1101" + "0";

convert(13)
if (N == 1) return "1";
return "110" + "1";

convert(6)
if (N == 1) return "1";
return "11" + "0";

convert(3)
if (N == 1) return "1";
return "1" + "1";

convert(1)
if (N == 1) return "1";
return convert(0) + "1";
```

% java Convert 26
11010
Programming with recursion: typical bugs

**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;
}
```

Try $N = 2$

Both lead to *infinite recursive loops* (bad news).
Collatz Sequence

Collatz function of $N$.
- If $N$ is 1, stop.
- If $N$ is even, divide by 2.
- If $N$ is odd, multiply by 3 and add 1.

```
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);
}
```

Amazing fact. No one knows whether or not this function terminates for all $N$ (!)

Note. We usually ensure termination by only making recursive calls for smaller $N$. 
The Collatz Conjecture states that if you pick a number, and if it's even divide it by two and if it's odd multiply it by three and add one, and you repeat this procedure long enough, eventually your friends will stop calling to see if you want to hang out.
7. Recursion

- Foundations
- A classic example
- Recursive graphics
- Avoiding exponential waste
- Dynamic programming
Warmup: subdivisions of a ruler (revisited)

\texttt{ruler(n)}: create subdivisions of a ruler to $1/2^n$ inches.

- Return one space for $n = 0$.
- Otherwise, sandwich $n$ between two copies of \texttt{ruler(n-1)}.

```java
public class RulerR {
    public static String ruler(int n) {
        if (n == 0) return " ";
        return ruler(n-1) + n + ruler(n-1);
    }
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        StdOut.println(ruler(n));
    }
}
```

```
% java RulerR 1
1
% java RulerR 2
1 2 1
% java RulerR 3
1 2 1 3 1 2 1
% java RulerR 4
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
% java RulerR 50
Exception in thread "main" java.lang.OutOfMemoryError:
Java heap space
```

$2^{50} - 1$ chars in output.
Tracing a recursive program

Use a *recursive call tree*

- One node for each recursive call.
- Label node with return value after children are labelled.
Towers of Hanoi puzzle

A legend of uncertain origin

• $n = 64$ discs of differing size; 3 posts; discs on one of the posts from largest to smallest.
• An ancient prophecy has commanded monks to move the discs to another post.
• When the task is completed, *the world will end.*

**Rules**

• Move discs one at a time.
• Never put a larger disc on a smaller disc.

Q. Generate list of instruction for monks?

Q. When might the world end?
Towers of Hanoi

For simple instructions, use cyclic wraparound

• Move right means 1 to 2, 2 to 3, or 3 to 1.
• Move left means 1 to 3, 3 to 2, or 2 to 1.

A recursive solution

• Move $n - 1$ discs to the left (recursively).
• Move largest disc to the right.
• Move $n - 1$ discs to the left (recursively).
Towers of Hanoi solution (n = 3)

1R  2L  1R  3R  1R  2L  1R
Towers of Hanoi: recursive solution

hanoi(n): Print moves for n discs.
- Return one space for n = 0.
- Otherwise, set move to the specified move for disc n.
- Then sandwich move between two copies of hanoi(n-1).

```java
public class HanoiR {
    public static String hanoi(int n, boolean left) {
        if (n == 0) return " ";
        String move;
        if (left) move = n + "L";
        else move = n + "R";
        return hanoi(n-1, !left) + move + hanoi(n-1, !left);
    }
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        StdOut.println(hanoi(n, false));
    }
}
```

% java HanoiR 3
1R 2L 1R 3R 1R 2L 1R
**Recursive call tree for towers of Hanoi**

Structure is the *same* as for the ruler function and suggests 3 useful and easy-to-prove facts.
- Each disc always moves in the same direction.
- Moving smaller disc always alternates with a unique legal move.
- Moving $n$ discs requires $2^n - 1$ moves.
**Answers for towers of Hanoi**

**Q.** Generate list of instructions for monks?

**A.** (Long form). 1L 2R 1L 3L 1L 2R 1L 4R 1L 2R 1L 3L 1L 2R 1L 5L 1L 2R 1L 3L 1L 2R 1L 4R ...  

**A.** (Short form). Alternate "1L" with the only legal move not involving the disc 1. "L" or "R" depends on whether $N$ is odd or even.

**Q.** When might the world end?

**A.** Not soon: need $2^{64} - 1$ moves.

Note: Recursive solution has been proven optimal.

<table>
<thead>
<tr>
<th>moves per second</th>
<th>end of world</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.84 billion centuries</td>
</tr>
<tr>
<td>1 billion</td>
<td>5.84 centuries</td>
</tr>
</tbody>
</table>
7. Recursion

- Foundations
- A classic example
- Recursive graphics
- Avoiding exponential waste
- Dynamic programming
Recursive graphics in the wild
"Hello, World" of recursive graphics: H-trees

H-tree of order \( n \)
- If \( n \) is 0, do nothing.
- Draw an H, centered.
- Draw four H-trees of order \( n - 1 \) and half the size, centered at the tips of the H.
**H-trees**

*Application.* Connect a large set of regularly spaced sites to a single source.
Recursive H-tree implementation

```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);
    }
}
```

- Draw the H, centered on (x, y)
- Draw four half-size H-trees

% java Htree 3
Deluxe H-tree implementation

```java
public class HtreeDeluxe {
    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);
        StdAudio.play(PlayThatNote.note(n, .25*n));
        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);
    }
}
```
Fractional Brownian motion

A process that models many phenomenon.
  • Price of stocks.
  • Dispersion of fluids.
  • Rugged shapes of mountains and clouds.
  • Shape of nerve membranes.

...
Fractional Brownian motion simulation

**Midpoint displacement method**
- Consider a line segment from \((x_0, y_0)\) to \((x_1, y_1)\).
- If sufficiently short draw it *and return*
- Divide the line segment in half, at \((x_m, y_m)\).
- Choose \(\delta\) at random *from Gaussian distribution*.
- Add \(\delta\) to \(y_m\).
- Recur on the left and right line segments.
Brownian motion implementation

```java
public class Brownian {
    public static void curve(double x0, double y0, double x1, double y1,
                               double var, double s) {
        if (x1 - x0 < .01) {
            StdDraw.line(x0, y0, x1, y1); return; }
        double xm = (x0 + x1) / 2;
        double ym = (y0 + y1) / 2;
        double stddev = Math.sqrt(var);
        double delta = StdRandom.gaussian(0, stddev);
        curve(x0, y0, xm, ym+delta, var/s, s);
        curve(xm, ym+delta, x1, y1, var/s, s);
    }
    public static void main(String[] args) {
        double H = Double.parseDouble(args[0]);
        double s = Math.pow(2, 2^H);
        curve(0, .5, 1.0, .5, .01, s);
    }
}
```
A 2D Brownian model: plasma clouds

Midpoint displacement method

- Consider a rectangle centered at \((x, y)\) with pixels at the four corners.
- If the rectangle is small, do nothing.
- Color the midpoints of each side the average of the endpoint colors.
- Choose \(\delta\) at random from Gaussian distribution.
- Color the center pixel the average of the four corner colors plus \(\delta\)
- Recurse on the four quadrants.
A Brownian cloud
A Brownian landscape
7. Recursion

• Foundations
• A classic example
• Recursive graphics
• Avoiding exponential waste
• Dynamic programming
Fibonacci numbers

Let $F_n = F_{n-1} + F_{n-1}$ for $n > 1$ with $F_0 = 0$ and $F_1 = 1$.

<table>
<thead>
<tr>
<th>$n$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_n$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>13</td>
<td>21</td>
<td>34</td>
<td>55</td>
<td>89</td>
<td>144</td>
<td>233</td>
<td>...</td>
</tr>
</tbody>
</table>

Models many natural phenomena and is widely found in art and architecture.

Examples.
- Model for reproducing rabbits.
- Nautilus shell.
- Mona Lisa.
- ...

Facts (known for centuries).
- $F_n / F_{n-1} \to \Phi = 1.618...$ as $n \to \infty$
- $F_n$ is the closest integer to $\Phi^n / \sqrt{5}$

Leonardo Fibonacci
C. 1170 – C. 1250
Fibonacci numbers and the golden ratio in the wild
Computing Fibonacci numbers

Q. [Curious individual.] What is the exact value of \( F_{60} \)?

A. [Novice programmer.] Just a second. I'll write a recursive program to compute it.

```java
class FibonacciR {  
    public static long F(int n)  
    {  
        if (n == 0) return 0;  
        if (n == 1) return 1;  
        return F(n-1) + F(n-2);  
    }  
    public static void main(String[] args)  
    {  
        int n = Integer.parseInt(args[0]);  
        StdOut.println(F(n));  
    }  
}
```

% java FibonacciR 5
5
% java FibonacciR 6
8
% java FibonacciR 10
55
% java FibonacciR 12
144
% java FibonacciR 50
12586269025
% java FibonacciR 60

Is something wrong with my computer?

Takes a few minutes

Hmm. Why is that?
Recursive call tree for Fibonacci numbers
Exponential waste

Let \(C_n\) be the number of times \(F(n)\) is called when computing \(F(60)\).

<table>
<thead>
<tr>
<th>(n)</th>
<th>(C_n)</th>
<th>(F_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>1</td>
<td>(F_1)</td>
</tr>
<tr>
<td>59</td>
<td>1</td>
<td>(F_2)</td>
</tr>
<tr>
<td>58</td>
<td>2</td>
<td>(F_3)</td>
</tr>
<tr>
<td>57</td>
<td>3</td>
<td>(F_4)</td>
</tr>
<tr>
<td>56</td>
<td>5</td>
<td>(F_5)</td>
</tr>
<tr>
<td>55</td>
<td>8</td>
<td>(F_6)</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>(\geq 2.5 \times 10^{12})</td>
<td>(F_{61})</td>
</tr>
</tbody>
</table>

Exponentially wasteful to recompute all these values. (trillions of calls on \(F(0)\), not to mention calls on \(F(1), F(2), \ldots\))
Exponential waste dwarfs progress in technology.

If you engage in exponential waste, you will not be able to solve a large problem.

<table>
<thead>
<tr>
<th>1970s</th>
<th>2010s: 10,000+ times faster</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n)</td>
<td>(n)</td>
</tr>
<tr>
<td>30</td>
<td>50</td>
</tr>
<tr>
<td>40</td>
<td>60</td>
</tr>
<tr>
<td>50</td>
<td>70</td>
</tr>
<tr>
<td>60</td>
<td>80</td>
</tr>
<tr>
<td>70</td>
<td>90</td>
</tr>
<tr>
<td>80</td>
<td>100</td>
</tr>
</tbody>
</table>

1970s: "That program won't compute \(F_{60}\) before you graduate!"

2010s: "That program won't compute \(F_{80}\) before you graduate!"
Avoiding exponential waste

Memoization

• Maintain an array memo[] to remember all computed values.
• If value known, just return it.
• Otherwise, compute it, remember it, and then return it.
7. Recursion

- Foundations
- A classic example
- Recursive graphics
- Avoiding exponential waste
- Dynamic programming
An efficient alternative to recursion

Dynamic programming.
• Build computation from the "bottom up".
• Solve small subproblems and save solutions.
• Use those solutions to build bigger solutions.

Fibonacci numbers

public class Fibonacci
{
    public static void main(String[] args)
    {
        int n = Integer.parseInt(args[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];
        StdOut.println(F[n]);
    }
}

Key advantage over recursive solution. Each subproblem is addressed only once.
How many ways to change a dollar?

**Q.** How many ways to change a dollar with quarters?

**A.** 1

**Q.** How many ways to change a dollar with quarters and dimes?

**A.** 3

**Q.** How many ways to change a dollar with quarters, dimes and nickels?

**Q.** How many ways to change a dollar with quarters, dimes, nickels and pennies?
How many ways to change a dollar?

Dynamic programming solution (Pólya).
- Count 1 way to change 0 cents.
- Maintain an array change[] for the number of known ways so far.
- For each coin V, pass through and update the array:
  \[
  \text{for (int } k = V; k <= N; k++) a[k] += a[k-V];
  \]
Pop quiz on changing a dollar

**Q.** What happens with amounts that are not multiples of 5?

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Quarter" /></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image2.png" alt="Nickel" /></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image3.png" alt="Dime" /></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image4.png" alt="Penny" /></td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Pop quiz on changing a dollar

**Q.** What happens with amounts that are not multiples of 5?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

**A.** They are 0 until V is 1. Then they take the value of the next lower multiple of 5.
How many ways to change a dollar?

Dynamic programming solution (Pólya).

- Count 1 way to change 0 cents.
- Maintain an array \( \text{change}[] \) for the number of known ways so far.
- For each coin \( V \), pass through and update the array:

  ```
  for (int k = V; k <= N; k++) a[k] += a[k-V];
  ```

|     | 0  | 5  | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 | 75 | 80 | 85 | 90 | 95 | 100 |
|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|     | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 2  | 1  | 2  | 1  | 2  | 2  | 2  | 2  | 2  | 2  | 2  | 2  | 2  | 3  |
|     | 1  | 1  | 2  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 10 | 11 | 13 | 14 | 16 | 18 | 20 | 22 | 24 | 26 | 29 |
|     | 1  | 2  | 4  | 6  | 9  | 13 | 18 | 24 | 31 | 39 | 49 | 60 | 73 | 87 | 103 | 121 | 141 | 163 | 187 | 213 | 242 |
How many ways to change a dollar?

**Dynamic programming solution**

```java
public class Change {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        int[] a = new int[N+1];
        a[0] = 1;
        for (int k = 25; k<=N; k++) a[k] += a[k-25];
        for (int k = 10; k<=N; k++) a[k] += a[k-10];
        for (int k = 5; k<=N; k++) a[k] += a[k-5];
        for (int k = 1; k<=N; k++) a[k] += a[k-1];
        StdOut.println(a[N]);
    }
}
```

Note. Recursive solution is *much more complicated* and can be *exponentially wasteful.*

% java Change 100
242
Dynamic programming and recursion

*Broadly useful* approaches to solving problems by combining solutions to smaller subproblems.

**Why learn DP and recursion?**
- Represent a new mode of thinking.
- Provide powerful programming paradigms.
- Give insight into the nature of computation.
- Successfully used for decades.

<table>
<thead>
<tr>
<th></th>
<th><em>recursion</em></th>
<th><em>dynamic programming</em></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>advantages</strong></td>
<td>Decomposition often obvious. Easy to reason about correctness.</td>
<td>Avoids exponential waste. Often simpler than memoization.</td>
</tr>
<tr>
<td><strong>pitfalls</strong></td>
<td>Potential for exponential waste. Decomposition may not be simple.</td>
<td>Uses significant space. Not suited for real-valued arguments. Challenging to determine order of computation</td>
</tr>
</tbody>
</table>
7. Recursion