Recursive Definitions
And Applications to Counting
Lecture 17
Tower of Hanoi

Move entire stack of disks to another peg
- Move one from the top of one stack to the top of another
- A disk cannot be placed on top of a smaller disk

How many moves needed?
- Optimal number not known when 4 pegs and over ≈30 disks!
- Optimal solution known for 3 pegs (and any number of disks)

Tower of Hanoi

Recursive algorithm (optimal for 3 pegs)

Transfer(n,A,C):
  If n=1, move the single disk from peg A to peg C
  Else
    Transfer(n-1,A,B) (leaving the largest disk out of play)
    Move largest disk to peg C
    Transfer(n-1,B,C) (leaving the largest disk out of play)
Tower of Hanoi

Recursive algorithm (optimal for 3 pegs)

Transfer(n,A,C):
  If n=1, move the single disk from peg A to peg C
  Else
    Transfer(n-1,A,B) (leaving the largest disk out of play)
    Move largest disk to peg C
    Transfer(n-1,B,C) (leaving the largest disk out of play)

How many moves are made by this algorithm?

M(n) be the number of moves made by the above algorithm

M(n) = 2M(n-1) + 1 with M(1) = 1

So, M(n) = ?
Catalan Numbers

How many paths are there in the grid from (0,0) to (n,n) without ever crossing over to the y>x region?

Any path can be constructed as follows

Pick minimum k>0 s.t. (k,k) reached

(0,0) → (1,0) ⇒ (k,k-1) → (k,k) ⇒ (n,n)

where ⇒ denotes a Catalan path

Cat(n) = \sum_{k=1}^{n} Cat(k-1) \cdot Cat(n-k)

Cat(0) = 1

So, Cat(n) = ?
Recursive Definitions

E.g.,  
\[ f(0) = 1 \]
\[ f(n) = n \cdot f(n-1) \quad \forall n \in \mathbb{Z} \text{ s.t. } n > 0 \]

f(n) = n \cdot (n-1) \cdot \ldots \cdot 1 \cdot 1 = n!

This is the formal definition of n! (without using “…”)

A recursive program to compute factorial:

```c
factorial(n\in\mathbb{N}) \{ 
    if (n==0) return 1;
    else return n*factorial(n-1);
}\}
```
Question

$f(0) = 5; \ f(n) = 3 \cdot f(n-1) \ for \ n \in \mathbb{Z}^+. \ Then \ for \ n \in \mathbb{N}$

A. $f(n) = 5^{n+1}$
B. $f(n) = 3 \cdot 5^n$
C. $f(n) = 5 \cdot 3^n$
D. $f(n) = 15 \cdot 3^n$
E. None of the above
Fibonacci Sequence

- $F(0) = 0$
- $F(1) = 1$
- $F(n) = F(n-1) + F(n-2) \ \forall n \geq 2$

$F(n)$ called the $n^{th}$ Fibonacci number (starting with $0^{th}$)
Counting Strings

How many ternary strings of length $n$ which don’t have “00” as a substring?

Set up a recurrence

- $A(n) =$ # such strings starting with 0
- $B(n) =$ # such strings not starting with 0

$A(n) = B(n-1)$ . $B(n) = 2(A(n-1) + B(n-1))$. [Why?]

Initial condition: $A(0) = 0$; $B(0) = 1$ (empty string)

Required count: $A(n) + B(n)$

Can rewrite in terms of just $B$

- $B(0) = 1$. $B(1) = 2$. $B(n) = 2B(n-1) + 2B(n-2)$ $\forall n \geq 2$
- Required count: $B(n-1) + B(n)$. 
Consider bit strings of length \( n \) which have “01” as a substring. Let \( A(n) = \# \) such strings starting with 0 and \( B(n) = \# \) such strings starting with 1. Then:

A. \( A(n) = A(n-1) + B(n-1) \)
B. \( A(n) = A(n-1) + 2^{n-2} \)
C. \( A(n) = 2^{n-1} + B(n-1) \)
D. \( A(n) = 1 + B(n-1) \)
E. None of the above

Exercise: Count directly, by counting “bad” strings.
Closed Form

Sometimes possible to get a “closed form” expression for a quantity defined recursively (in terms of simpler operations)

- e.g., \( f(0)=0 \) & \( f(n) = f(n-1) + n, \forall n>0 \)
- \( f(n) = \frac{n(n+1)}{2} \)

Sometimes, we just give it a name

- e.g., \( n! \), Fibonacci\((n)\), Cat\((n)\)

In fact, formal definitions of integers, addition, multiplication etc. are recursive

- e.g., \( 0 \cdot a = 0 \) & \( n \cdot a = (n-1) \cdot a + a, \forall n>0 \)
- e.g., \( 2^0 = 1 \) & \( 2^n = 2 \cdot 2^{n-1} \)

Sometimes both

- e.g., Fibonacci\((n)\), Cat\((n)\) have closed forms (later)
Understanding a Recursive Definition

Suppose \( g(1) = 1 \) & \( g(n) = 2 \cdot g(n-1) + n \) \( \forall n \geq 1 \).

\( g(n) \) is growing “exponentially” by (more than) doubling for each increment in \( n \).

\( g(n) = ? \)

Make a “guess”. Then prove by induction

How do we guess? (More ideas later.)

\[
g(n) = n + 2 \cdot g(n-1)
= n + 2 \cdot ( (n-1) + 2 \cdot g(n-2) )
= n + 2 \cdot ( (n-1) + 2 \cdot ( (n-2) + 2 \cdot g(n-3) ) )
= n + 2 \cdot (n-1) + 2^2 \cdot (n-2) + 2^3 \cdot g(n-3)
\]

\( g(n) = \sum_{k=0}^{n-1} 2^k \cdot (n-k) \) (make sure the base case matches)
Recursion & Induction

Claim: \( F(3n) \) is even, where \( F(n) \) is the \( n \)th Fibonacci number, \( \forall n \geq 0 \)

Proof by induction:

Base case:
- \( n=0: \) \( F(3n) = F(0) = 0 \) ✔
- \( n=1: \) \( F(3n) = F(3) = 2 \) ✔

Induction step: for all \( k \geq 2 \)

Induction hypothesis: suppose for \( 0 \leq n \leq k-1 \), \( F(3n) \) is even

To prove: \( F(3k) \) is even

\[ F(3k) = F(3k-1) + F(3k-2) = ? \]

Unroll further:
- \( F(3k-1) = F(3k-2) + F(3k-3) \)
- \( F(3k) = 2 \cdot F(3k-2) + F(3(k-1)) = \) even, by induction hypothesis

Stronger claim (but easier to prove by induction):
\( F(n) \) is even iff \( n \) is a multiple of 3
Recursion & Induction

\[ f(0) = c. \quad f(1) = d. \quad f(n) = a \cdot f(n-1) + b \cdot f(n-2) \quad \forall n \geq 2. \]

Suppose \( X^2 - aX - b = 0 \) has two distinct (possibly complex) solutions, \( x \) and \( y \).

Claim: \( f(n) = p \cdot x^n + q \cdot y^n \) for some \( p, q \)

Base cases satisfied by \( p = (d-cy)/(x-y), \quad q = (d-cx)/(y-x) \)

Inductive step: for all \( k \geq 2 \)

Induction hypothesis: \( \forall n \text{ s.t. } 1 \leq n \leq k-1, \quad f(n) = px^n + qy^n \)

To prove: \( f(k) = px^k - qy^k \)

\[
\begin{align*}
    f(k) &= a \cdot f(k-1) + b \cdot f(k-2) \\
    &= a \cdot (px^{k-1} + qy^{k-1}) + b \cdot (px^{k-2} + qy^{k-2}) - px^k - qy^k + px^k + qy^k \\
    &= -px^{k-2}(x^2-ax-b) - qy^{k-2}(y^2-ay-b) + px^k + qy^k = px^k + qy^k \quad \checkmark
\end{align*}
\]

Example: Fibonacci numbers

Characteristic equation: replace \( f(n) \) by \( X^n \) in the recurrence
Recursion & Induction

- \( f(0) = c \)
- \( f(1) = d \)
- \( f(n) = a \cdot f(n-1) + b \cdot f(n-2) \quad \forall n \geq 2 \).

Suppose \( X^2 - aX - b = 0 \) has only one solution, \( x \neq 0 \).
\[ i.e., \ a = 2x, \ b = -x^2, \text{ so that } X^2 - aX - b = (X-x)^2. \]

Claim: \( f(n) = (p + q \cdot n)x^n \) for some \( p, q \)

Base cases satisfied by \( p = c, \ q = d/x - c \)

Inductive step: for all \( k \geq 2 \)

Induction hypothesis: \( \forall n \text{ s.t. } 1 \leq n \leq k-1, \ f(n) = (p + qn)y^n \)

To prove: \( f(k) = (p+qk)x^k \)

\[
\begin{align*}
f(k) &= a \cdot f(k-1) + b \cdot f(k-2) \\
&= a \cdot (p+qk-2q)x^{k-1} + b \cdot (p+qk-2q)x^{k-2} \\
&= -(p+qk)x^{k-2}x^2 - (p+qk)x^{k-2}(ax-2b) + (p+qk)x^k \\
&= (p+qk)x^k \quad \checkmark
\end{align*}
\]
Solving a Recurrence

Often, once a correct guess is made, easy to prove by induction

How does one guess?

Will see a couple of approaches

By unrolling the recursion into a chain or a “rooted tree”

Using the “method of generating functions” (next time)
Unrolling a recursion

Often helpful to try “unrolling” the recursion to see what is happening

E.g., expand into a chain:

\[ T(0) = 0 \quad \& \quad T(n) = T(n-1) + n^2 \quad \forall n \geq 1 \]
\[ T(n-1) = T(n-2) + (n-1)^2, \quad T(n-2) = T(n-3) + (n-2)^2, \quad \ldots \]
\[ T(n) = n^2 + (n-1)^2 + (n-2)^2 + T(n-3) \quad \forall n \geq 3 \]
\[ T(n) = \sum_{k=1}^{n} k^2 + T(0) \quad \forall n \geq 0 \]
Another example

- \( T(1) = 0 \)
- \( T(N) = T\left(\left\lfloor N/2 \right\rfloor\right) + 1 \quad \forall N \geq 2 \)

Let us consider \( N \) of the form \( 2^n \) (so we can forget the floor)
- \( T(N) = 1 + T(N/2) \)
  - \( = 1 + 1 + T(N/4) \)
  - \( = \ldots \)
  - \( = 1 + 1 + \ldots + T(1) \)
- \( T(2^n) = n \)
- \( T(N) = \log_2 N \) (or simply \( \log N \)) for \( N \) a power of 2

General \( N \)? \( T \) monotonically increasing (by strong induction). So,
  - \( T(2 \left\lfloor \log N \right\rfloor) \leq T(N) \leq T(2 \left\lceil \log N \right\rceil) \) : i.e., \( \left\lfloor \log N \right\rfloor \leq T(N) \leq \left\lceil \log N \right\rceil \)

In fact, \( T(N) = T(2 \left\lfloor \log N \right\rfloor) = \left\lfloor \log N \right\rfloor \) (Exercise)

How many 1's are there?

A slowly growing function
Recursive algorithm (optimal for 3 pegs)

Transfer(n,A,C):
    If n=1, move the single disk from peg A to peg C
    Else
    Transfer(n-1,A,B) (leaving the largest disk out of play)
    Move largest disk to peg C
    Transfer(n-1,B,C) (leaving the largest disk out of play)

M(n) be the number of moves made by the above algorithm

M(n) = 2M(n-1) + 1  with M(1) = 1

Unroll the recursion into a “rooted tree”
A tree, with a special node, designated as the root.

Typically drawn “upside down”

Parent and child relation: u is v’s parent if the unique path from v to root contains edge \{v,u\} (parent unique; root has no parent)

- If u is v’s parent v, then v is a child of u
- u is an ancestor of v, and v a descendent of u if the v-root path passes through u

Leaf is redefined for a rooted tree, as a node with no child

- Root is a leaf iff it has degree 0 (if deg(root)=1, not called a leaf)
**Rooted Tree**

- **Leaf**: no children. **Internal node**: has a child
- **Ancestor, descendant**: partial orders
- **Subtree rooted at** u: with all descendants of u
- **Depth of a node**: distance from root.
- **Height of a tree**: maximum depth
- **Level** i: Set of nodes at depth i.
- **Note**: tree edges are between adjacent levels
- **Arity of a tree**: Max (over all nodes) number of children. **m-ary** if arity ≤ m.
- **Full m-ary tree**: Every internal node has exactly m children.
- **Complete & Full**: All leaves at same level
**Rooted Tree**

- **Number of nodes in Complete & Full m-ary tree**
  - One root node with m children at level 1
  - Each level 1 node has m children at level 2
    - $m^2$ nodes at level 2
  - At level $i$, $m^i$ nodes
  - $m^h$ leaves, where $h$ is the height

**Total number of nodes:**
- $m^0 + m^1 + m^2 + \ldots + m^h = (m^{h+1}-1)/(m-1)$

- Prove by induction:
  - $(m^h-1)/(m-1) + m^h = (m^{h+1}-1)/(m-1)$

- **Binary tree (m=2)**
  - $2^h$ leaves, $2^h-1$ internal nodes
Tower of Hanoi

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

Doing it bottom-up. Could also think top-down.
Tower of Hanoi

- $M(1) = 1$
- $M(n) = 2M(n-1) + 1$

- Exponential growth

- $M(2) = 3$, $M(3) = 7$, ...

- $M(n) = \text{#nodes in a complete and full binary tree of height } n-1$

- $M(n) = 2^n - 1$