Recursion and Induction

• Themes
  – Recursion
  – Recurrence Definitions
  – Recursive Relations
  – Induction (prove properties of recursive programs and objects defined recursively)

• Examples
  – Tower of Hanoi
  – Gray Codes
  – Hypercube
Recursion & Recurrence Relations

• Very handy for defining functions and data types simply:
  – Consider the nth Fibonacci number, $F_n$:
    • $= 1$, if $n = 1$ or $n=2$
    • $= F_{n-1} + F_{n-2}$, for all $n>2$

• Very handy when a large problem can be broken in similar (but smaller) problems
  – We’ll look at the Towers of Hanoi in a moment
Who needs Induction?

This chapter will provide us with some tools, to reason precisely about algorithms, and programs

- Check for correctness
  - Does the program end?
  - Does it do its job?

- Check performance
  - How does the runtime of a particular algorithm grow vs. the inputs (number and/or size)?
Induction & Recursion

• Very similar notions. They have exactly the same roots
• Inductive proofs apply in a very natural way to recursive algorithms, and recurrence relations
• This chapter will present tools we’ll use for the rest of the course
• Also gives us the flavor of how we’ll approach the rest of the material
Tower of Hanoi

• There are three towers
• 64 gold disks, with decreasing sizes, placed on the first tower
• You need to move the stack of disks from one tower to another, one disk at a time
• Larger disks can not be placed on top of smaller disks
• The third tower can be used to temporarily hold disks
Tower of Hanoi

• Assume one disk can be moved in 1 second
  How long would it take to move 64 disks? N disks?

• To create an algorithm to solve this problem, it is convenient to generalize the problem to the “N-disk” problem, where in our case N = 64.
Recursive Solution
Recursive Solution
Recursive Solution
Recursive Solution
Tower of Hanoi
Tower of Hanoi
Tower of Hanoi
Tower of Hanoi
Tower of Hanoi
Tower of Hanoi
Tower of Hanoi
Tower of Hanoi
Recursive Algorithm

(see http://www.cs.drexel.edu/~kschmidt/CS520/Programs/hanoi.cc)

```c
void Hanoi( int n, string a, string b, string c)
{
    if (n == 1) /* base case */
        Move( a, b );
    else { /* recursion */
        Hanoi( n-1, a, c, b );
        Move( a, b );
        Hanoi( n-1, c, b, a );
    }
}
```
Induction

• To prove a statement $S(n)$ for positive integers $n$
  – Need a base case (typically $n=0$ or $n=1$). Prove that $S(0)$ or $S(1)$ is true
  – Assume $S(n)$ is true [inductive hypothesis]
  – Prove that $S(n+1)$ is true. You’ll need to use the hypothesis in this step, or you did something wrong

• See http://www.cs.drexel.edu/~kschmidt/CS520/Lectures/1/induction1.swf for an example
Induction Examples

Prove:

• $1 + 3 + 5 + \ldots + (2n-1) = n^2$
• $4n < 2^n$, $\forall \ n \geq 5$
• $7 \mid 8^n - 1$
• $2^n < n!$, $\forall \ n \geq 4$
Correctness

• Use induction to prove that the recursive algorithm solves the Tower of Hanoi problem.

(see http://www.cs.drexel.edu/~kschmidt/CS520/Lectures/1/hanoiCorrect.swf )
Cost

• Show that the number of moves $M(n)$ required by the algorithm to solve the $n$-disk problem satisfies the recurrence relation
  – $M(n) = 2M(n-1) + 1$
  – $M(1) = 1$

• This can be done inductively, and it would be very similar to the last proof.
Cost

• We’d like to find a closed form, a formula that is not a recurrence relation
• We can do this a couple ways:
  – We can guess at the answer (and then prove it, of course)
  – We can unwind the recurrence (still need to prove it)
Guess and Prove

- Calculate $M(n)$ for small $n$ and look for a pattern.
- Guess the result and prove your guess correct using induction.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$M(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>31</td>
</tr>
</tbody>
</table>

- See http://www.cs.drexel.edu/~kschmidt/CS520/Lectures/1/guessHanoi.swf
Substitution Method

• Unwind recurrence, by repeatedly replacing \( M(n) \) by the r.h.s. of the recurrence until the base case is encountered.

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

For other examples of unwinding recurrences, see
http://www.cs.drexel.edu/~kschmidt/Lectures/Complexity/recurrenceRelations-substitution.pdf or
http://www.cs.drexel.edu/~kschmidt/Lectures/Complexity/recurrenceRelations-substitution.ppt
Geometric Series

- After k steps:
  \[ M(n) = 2^k \cdot M(n-k) + 1 + 2 + 2^2 + \ldots + 2^{k-1} \]

- Base case, \( M(1) \), encountered when \( n-k=1 \)
  \[ \Rightarrow k = n-1 \]

- Substituting in, to get rid of all of the k’s:
  \[ M(n) = 2^{n-1} \cdot M(1) + 1 + 2 + 2^2 + \ldots + 2^{n-2} \]
  \[ = 1 + 2 + \ldots + 2^{n-1} = \sum_{i=0}^{n-1} 2^i = 2^n - 1 \]

- Use induction to reprove result for \( M(n) \) using this sum. Generalize by replacing 2 by \( x \).

To see this done, see
http://www.cs.drexel.edu/~kschmidt/CS520/Lectures/1/unwindingHanoi.swf
Gray Code

- An n-bit Gray code is a 1-1 onto mapping from $[0..2^n-1]$ such that the binary representation of consecutive numbers differ by exactly one bit.
- Invented by Frank Gray for a shaft encoder - a wheel with concentric strips and a conducting brush which can read the number of strips at a given angle. The idea is to encode $2^n$ different angles, each with a different number of strips, corresponding to the n-bit binary numbers.
Shaft Encoder (Counting Order)

Consecutive angles can have an abrupt change in the number of strips (bits) leading to potential detection errors.
Since a Gray code is used, consecutive angles have only one change in the number of strips (bits).
Binary-Reflected Gray Code

1. \( G_1 = [0,1] \)

2. \( G_n = [0G_{n-1}, 1G_{n-1}], \ G \Rightarrow \text{reverse order} \equiv \text{complement leading bit} \)

3. \( G_2 = [0G_1, 1G_1] = [00,01,11,10] \)

4. \( G_3 = [0G_2, 1G_2] = [000,001,011,010,110,111,101,100] \)

5. Use induction to prove that this is a Gray code

(See http://www.cs.drexel.edu/~kschmidt/CS520/Lectures/1/grayCodeRec.swf )
Iterative Formula

• Let $G_n(i)$ be a function from $[0, \ldots, 2^n-1]$

• $G_n(i) = i \oplus (i >> 1)$ [exclusive or of $i$ and $i/2$]
  
  $G_2(0) = 0$, $G_2(1) = 1$, $G_2(2) = 3$, $G_2(3) = 2$

• Use induction to prove that the sequence $G_n(i)$,
  $i=0, \ldots, 2^n-1$ is a binary-reflected Gray code.

(See
http://www.cs.drexel.edu/~kschmidt/CS520/Lectures/1/grayCodeRec.swf )
Gray Code & Tower of Hanoi

- Introduce coordinates \((d_0,\ldots,d_{n-1})\), where \(d_i \in \{0,1\}\)
- Associate \(d_i\) with the \(i\)th disk
- Initialize to \((0,\ldots,0)\) and flip the \(i\)th coordinate when the \(i\)-th disk is moved
- The sequence of coordinate vectors obtained from the Tower of Hanoi solution is a Gray code (why?)
Tower of Hanoi

(0,0,0)
Tower of Hanoi

(0,0,1)
Tower of Hanoi

(0,1,1)
Tower of Hanoi

$(0,1,0)$
Tower of Hanoi

(1,1,0)
Tower of Hanoi

(1,1,1)
Tower of Hanoi

(1,0,1)
Tower of Hanoi

(1,0,0)
Hypercube

• Graph (recursively defined)
• \( n \)-dimensional cube has \( 2^n \) nodes with each node connected to \( n \) vertices
• Binary labels of adjacent nodes differ in one bit
A Hamiltonian path is a sequence of edges that visit each node exactly once.

A Hamiltonian path on a hypercube provides a Gray code (why?)
Hypercube and Gray Code