MA/CSSE 473 Day 07

More Mathematical Induction

Euclid's Algorithm

MA/CSSE 473 Day 07

- HW 4 is due tomorrow
- Don't forget to let me know if you are looking for a partner for the Trominoes implementation problem (due Friday).
- **Student Questions**
- More mathematical induction review
  - Pie survivor
  - Tiling with Trominoes
- Euclid's algorithm

(More on arithmetic later; this is the review thread)

ANOTHER INDUCTION EXAMPLE: ODD PIE FIGHT

Another Induction Example

- Pie survivor
  - An odd number of people stand in various positions such that no two distances between people are equal
    - Each person has a pie
    - A whistle blows, and each person simultaneously and accurately throws his/her pie at the nearest neighbor
  - **Claim:** No matter how the people are arranged, at least one person does not get hit by a pie
  - Let P(n) denote the statement: "There is a survivor in every odd pie fight with 2n + 1 people"
  - Prove by induction that P(n) is true for all n ≥ 1

Q1

Structural Induction

- When a structure is defined recursively, use induction on the structural definition to prove that the property is true for everything covered by the definition
- Base case: The base cases in the recursive definition
- Induction step: Each recursive part of the definition
- We could express it as ordinary induction based on some metric of the structure, but it is often easier to do the induction on the structure itself.

Structural Induction Example 1

- Consider the following oversimplified definition of expressions in a programming language:
  - <Exp> ::= <number>
  - <Exp> ::= <Exp> <op> <Exp>
  - <Exp> ::= **(** <Exp> **)**
  - <op> ::= **+** | **-** | ***** | **/**
- Prove by structural induction: anything that can be derived from this grammar has an even number of parentheses

Structural Induction Example 2

- An Extended Binary Tree (EBT) **T** is either:
  - An external node, (designated by a square in diagrams), or
  - An internal node (designated by a circle in diagrams), and two subtrees (T_L and T_R) which are themselves EBTs. This internal node is called the **root** of the tree
- **Notation:** EN(**T**) and IN(**T**) denote the number of external nodes and internal nodes, respectively, in the Extended Binary Tree **T**.
- Prove by structural induction:
  In every EBT **T**, EN(**T**) = IN(**T**) + 1

Proof is on an earlier ICQ solution

Modular Exponentiation
Euclid's Algorithm

BACK TO ARITHMETIC THREAD

Modular Exponentiation Algorithm

```
def modexp(x, y, N):
    if y==0:
        return 1
    z = modexp(x, y//2, N)
    if y%2 == 0:
        return (z*z) % N
    return (x*z*z) % N
```

- Let n be the maximum number of bits in x, y, or N
- The algorithm requires at most ___ recursive calls
- Each call is Θ( )
- So the overall algorithm is Θ( )

Q2-4

Modular Exponentiation Algorithm

```
def modexp(x, y, N):
    if y==0:
        return 1
    z = modexp(x, y//2, N)
    if y%2 == 0:
        return (z*z) % N
    return (x*z*z) % N
```

- Let n be the maximum number of bits in x, y, or N
- The algorithm requires at most **n** recursive calls
- Each call is Θ($\mathbf{n^2}$)
- So the overall algorithm is Θ($\mathbf{n^3}$)

Euclid's Algorithm: the problem

- One of the oldest known algorithms (about 2500 years old)
- **The problem:** Find the greatest common divisor (gcd) of two non-negative integers a and b.
- The approach you learned in grade school:
  - Completely factor each number
  - find common factors (with multiplicity)
  - multiply the common factors together to get the gcd
- Factoring is hard!
- Simpler approach is needed

Euclid's Algorithm: the basis

- Based on the following rule:
  - If x and y are positive integers with $x \geq y$, then gcd(x, y) = gcd(y, x mod y)
- Proof of Euclid's rule:
  - It suffices to show the simpler rule
    gcd(x, y) = gcd(y, x - y)
    since x mod y can be obtained from x and y by repeated subtraction
  - Any integer that divides both x and y must also divide $x - y$, so $\gcd(x, y) \leq \gcd(y, x - y)$
  - Any integer that divides both y and x - y must also divide y, so $\gcd(y, x-y) \leq \gcd(y, x)$

Euclid's Algorithm: the algorithm

```
def euclid(a, b):
    """ INPUT:  Two integers a and b with a >= b >= 0
        OUTPUT: gcd(a, b)"""
    if b == 0:
        return a
    return euclid(b, a % b)
```

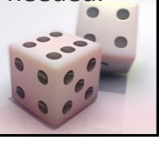
- Example: euclid(60, 36)
- Does the algorithm work?
- How efficient is it?

Q5

Euclid's Algorithm: the analysis

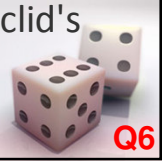
```
def euclid(a, b):
    """ INPUT:  Two integers a and b with a >= b >= 0
        OUTPUT: gcd(a, b)"""
    if b == 0:
        return a
    return euclid(b, a % b)
```

- Lemma: If **a** ≥ **b**, then **a** % **b** < **a**/2
- Proof
  - If **b** ≤ **a**/2, then **a** % **b** < **b** ≤ **a**/2
  - If **b** > **a**/2, then **a** % **b** = **a** – **b** < **a**/2
- Application
  - After two recursive calls, both **a** and **b** are at most half of what they were, (i.e. reduced by at least 1 bit)
  - Thus if a and b have n bits, at most 2n recursive calls are needed.
  - Each recursive call involves a division, $\Theta(n^2)$
  - Entire algorithm is $\Theta(n^3)$



gcd and linear combinations

- Lemma: If **d** divides both **a** and **b**, and **d** = **a**x + **b**y for some integers x and y, then **d** = gcd(**a**,**b**)
- Proof
  - By the first of the two conditions, **d** is a common divisor of **a** and **b**. It cannot exceed the greatest common divisor, so **d** ≤ gcd(**a**, **b**)
  - gcd(**a**, **b**) is a common divisor of **a** and **b**, so it must divide **a**x + **b**y = **d**. Thus gcd(**a**, **b**) ≤ **d**
  - Putting these together, gcd(**a**, **b**) = **d**
- If we can supply the x and y as in the lemma, we know that d is the gcd.
- It turns out that a simple modification of Euclid's algorithm will calculate the x and y.



Q6