MA/CSSE 473 Day 08

Euclid's Algorithm

Modular Division

Fermat's little theorem

MA/CSSE 473 Day 08

- A corrected version of Trominoes has been posted
  - Without the vestigial "convex hull" reference.
- **Student Questions**
- Extended Euclid Algorithm
- Modular Division
- Fermat's Little Theorem

QuiIck look at review topics in textbook

REVIEW THREAD

Textbook Topics I Won't Cover

- **Chapter 1 topics** that I will not discuss in detail unless you have questions. They should be review:
  - Sieve of Eratosthenes
  - Algorithm Specification, Design, Proof, Coding
  - Problem types : sorting, searching, string processing, graph problems, combinatorial problems, geometric problems, numerical problems
  - Data Structures: ArrayLists, LinkedLists, Graphs and their representations, Weighted graphs (a.k.a Networks), trees, search trees, sets, dictionaries,

Textbook Topics I Won't Cover*

- Chapter 2
  - Empirical analysis of algorithms should be review
  - I believe that we have covered everything else in the chapter except amortized algorithms and recurrence relations
  - We will discuss amortized algorithms
  - Recurrence relations are covered in CSSE 230 and MA 375. We'll review particular types as we encounter them.

*Unless you ask me to

Textbook Topics I Won't Cover*

- Chapter 3 - Review
  - Bubble sort, selection sort, and their analysis
  - Sequential search and simple string matching
- Probably new, but quite simple to understand
  - Closest Pair and Convex Hull
  - Traveling Salesman
  - Knapsack
  - Assignment

*Unless you ask me to

Textbook Topics I Won't Cover*

- Chapter 4 - Review
  - Mergesort, quicksort, and their analysis
  - Binary search
  - Binary Tree Traversals

*Unless you ask me to

Textbook Topics I Won't Cover*

- Chapter 5 - Review
  - Insertion Sort and its analysis
  - Depth-first search and Breadth-first Search
  - Binary Tree Traversals
  - Interpolation Search
  - Search, insertion, delete in Binary Tree
  - AVL tree insertion and rebalance

*Unless you ask me to

Extended Euclid's Algorithm

Heading toward Primality Testing

ARITHMETIC THREAD

Recap: Euclid's Algorithm for gcd

```
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)
```

Recap: 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 – we did it yesterday

Q1

Extended Euclid Algorithm

```
def euclidExtended(a, b):
    """ INPUT:  Two integers a and b with a >= b >= 0
        OUTPUT: Integers x, y, d such that d = gcd(a, b)
                and d = ax + by"""
    print ("    ", a, b) # so we can see the process.
    if b == 0:
        return 1, 0, a
    x, y, d =  euclidExtended(b, a % b)
    return y, x - a//b*y, d
```

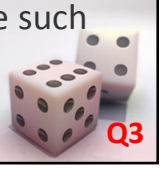
- Proof that it works
  - First, the number d it produces really is the gcd of a and b.
    - We can just ignore the x and y values, and we have the same algorithm as before.

Q2-3

Extended Euclid Algorithm: proof

```
def euclidExtended(a, b):
    """ INPUT:  Two integers a and b with a >= b >= 0
        OUTPUT: Integers x, y, d such that d = gcd(a, b)
                and d = ax + by"""
    print ("    ", a, b) # so we can see the process.
    if b == 0:
        return 1, 0, a
    x, y, d =  euclidExtended(b, a % b)
    return y, x - a//b*y, d
```

- Proof that it works
  - First, the number d it produces really is gcd (a,b)
    - We can just ignore the x and y values, and we have the same algorithm as before.
  - We must show that the x and y it returns are such that ax + by =d.
  - We do that by induction on b.



Q3

Proof that ax+by = d (induction on b)

```
def euclidExtended(a, b):
    """ INPUT:  Two integers a and b with a >= b >= 0
        OUTPUT: Integers x, y, d such that d = gcd(a, b)
                and d = ax + by"""
    print ("    ", a, b) # so we can see the process.
    if b == 0:
        return 1, 0, a
    x, y, d =  euclidExtended(b, a % b)
    return y, x - a//b*y, d
```

- **Base case:** b=0 Then gcd(a,b) = a, and the algorithm produces x = 1, y = 0. ☑
- **Induction step:** b > 0.
  - It finds gcd(a, b) by calling euclidExtended(b, a%b)
  - Since a%b is smaller than b, by induction the x' and y' returned by the recursive call are such that
    gcd(b, a %b) = bx'+ (a%b)y'
  - We can write a % b as a – ⌊a/b⌋ * b
  - d = gcd(a, b) = gcd(b, a%b) = bx' + (a%b)y'
    = bx' +(a – ⌊a/b⌋ * b)y' = ay' +b(x' - ⌊a/b⌋ y')
  - Thus x = y' and y = x' - ⌊a/b⌋ y' are the numbers that make ax + by = d
  - This x and y are the numbers returned by the algorithm.

Q3

Example: gcd (25, 11)

- 25 = 2*11 + 3
- 11 = 3 * 3 + 2
- 3 = 1 * 2 + 1
- 2 = 2 * 1 + 0, so gcd(11, 25) = 1.
- **Now work backwards**
- 1 = 1 - 0. Substitute 0 = 2 - 2*1
- 1 = 1 - (2 - 2*1) = -1*2 + 3*1. Substitute 1 = 3 - 1*2
- 1 = -1*2 + 3(3 - 1*2) = 3*3 - 4*2. Substitute 2 = 11 - 3*3
- 1 = 3*3 - 4*(11 - 3*3) = -4*11 + 15*3. Substitute 3 = 25 - 2*11
- 1 = -4(11) + 15(25 - 2*11) = -34*11 + 15*25
- Thus x = 15 and y = -34 Done!

Q4

Modular Inverse

- In arithmetic over the real or rational numbers, every non-zero number **a** has an inverse 1/**a**.
- **Definition** x is the **multiplicative inverse of a modulo N** if **a**x ≡ 1 (mod N).
- We denote this inverse $\mathbf{a}^{-1}$ (if it exists)
- 2 has no inverse modulo 6
- **a** has an inverse modulo N if and only if gcd(a, N) = 1
  - i.e. **a** and N are **relatively prime**
- If $\mathbf{a}^{-1}$ exists, it is unique

Q5

Calculate Modular Inverse (if it exists)

- Assume that gcd(a, N) = 1.
- The extended Euclid's algorithm gives us integers x and y such that ax + Ny = 1
- This implies $ax \equiv 1 \pmod{N}$, so x is the inverse of a
- **Example:** Find $11^{-1} \bmod 25$
  - We saw before that -34*11 + 15*25 = 1
  - $-34 \equiv 16 \pmod{25}$
  - So $11^{-1} = 16 \bmod 25$
- Recall that Euclid's algorithm is $\Theta(n^3)$, where n is the number of bits of N.

Q6

Modular division

- We can only divide b by a (modulo N) if N and a are relatively prime
- In that case $b/a = b \cdot a^{-1}$
- What is the running time for modular division?

Q7-8

Primality testing

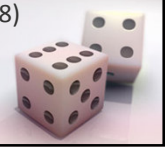
- The numbers 7, 17, 19, 71, and 79 are primes, but what about 717197179 (a typical social security number)?
- There are some tricks that might help. For example:
  - If n is even and not equal to 2, it's not prime
  - n is divisible by 3 **iff** the sum of its decimal digits is divisible by 3,
  - n is divisible by 5 **iff** it ends in 5 or 0
  - n is divisible by 7 **iff** $\lfloor n/10 \rfloor - 2*n\%10$ is divisible by 7
  - when checking for factors, we only need to consider prime numbers as candidates
  - When checking for factors, we only need to look for numbers up to sqrt(n)
-

Like a few other things that we have done so far ion this course, this discussion follows Dasgupta, *et. al.*, *Algortihms* (McGraw-Hill 2008)

Primality testing

- But this approach is not very fast. Factoring is harder than primality testing.
- Is there a way to tell whether a number is prime without actually factoring the number?

Like a few other things that we have done so far ion this course, this discussion follows Dasgupta, *et. al.*, *Algortihms* (McGraw-Hill 2008)



Fermat's Little Theorem (1640)

- **Formulation 1:** If p is prime, then for every number a with $1 \le a < p$, $a^{p-1} \equiv 1 \pmod{p}$
- **Formulation 2:** If p is prime, then for every number a with $1 \le a < p$, $a^{p} \equiv a \pmod{p}$
- These are clearly equivalent.
- We will examine a combinatorial proof of the first formulation.

Q9