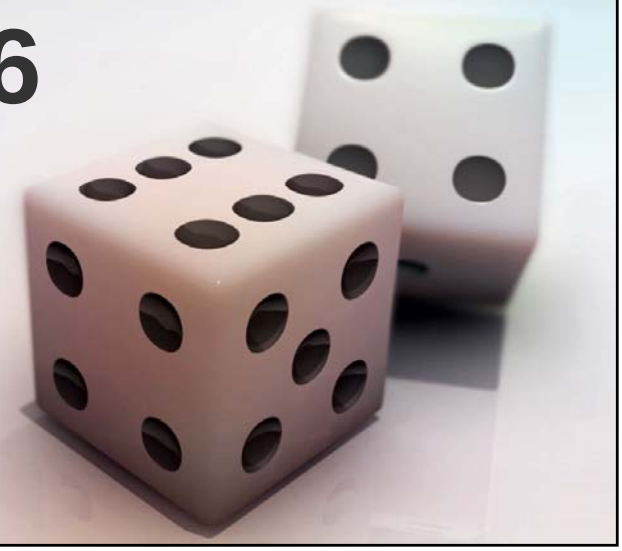
MA/CSSE 473 Day 06

Euclid's Algorithm



MA/CSSE 473 Day 06

- **Student Questions**
- Odd Pie Fight
- Modular exponentiation
- Euclid's algorithm
- (if there is time) extended Euclid's algorithm

Quick look at review topics in textbook

REVIEW THREAD

Another Induction Example

- Pie survivor
  - An odd number of people stand in various positions (2D or 3D) 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 $n \geq 1$

Odd Pie fight solution

- The base case is easy: If n = 1 the two persons (A and B) with the smallest pairwise distance between them throw at each other, while the third person throws at one of them (whoever is closer). Therefore, this third person remains "unharmed".
- For the inductive step, assume that the assertion is true for odd $k \geq 1$, and consider $2k+3$ persons. Again, the two persons with the smallest pairwise distance between them (the closest pair) throw at each other.
- Consider two possible cases as follows.
  - If the remaining n persons all throw at one another, at least one of them survives, by the inductive assumption.
  - If at least one of the remaining n persons throws at one of A or B, among the remaining $2k+1$ persons, at most $2k$ are hit, so there must be a survivor because there is not enough pies to hit everybody. This completes the proof.

Recap: Modular addition and multiplication

Euclid's Algorithm

Heading toward Primality Testing

ARITHMETIC THREAD

Modular Addition and Multiplication

- To **add** two integers x and y modulo N (where $k = \lceil \log N \rceil$), begin by doing regular addition.
  - x and y are in the range **0 to N-1**, so x + y is in range **0 to 2N-2**
  - If the sum is greater than N-1, subtract N, else return x + y
  - Run time is Θ (**k**)
- To **multiply** x and y, begin with regular multiplication, which is quadratic in k.
  - The result is in range **0 to $(N-1)^2$** so has at most **2k** bits.
  - Then compute the remainder when xy dividing by N, quadratic time in k. So entire operation is Θ(**k^2**)

Modular Exponentiation

- In some cryptosystems, we need to compute **x^y modulo N**, where all three numbers are several hundred bits long. Can it be done quickly?
- Can we simply take x^y and then figure out the remainder modulo N?
- Suppose x and y are only 20 bits long.
  - x^y is at least $(2^{19})^{(2^{19})}$, which is about 10 million bits long.
  - Imagine how big it will be if y is a 500-bit number!
- To save space, we could repeatedly multiply by x, taking the remainder modulo N each time.
  - If y is 500 bits, then there would be 2^{500} bit multiplications.
  - This algorithm is exponential in the length of y.
  - Ouch!

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 k 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( )

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 **k** recursive calls
- Each call is Θ($\mathbf{k^2}$)
- So the overall algorithm is Θ($\mathbf{k^3}$)

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 elementary school:
  - Completely factor each number
  - find common factors (with multiplicity)
  - multiply the common factors together to get the gcd
- Finding factors of large numbers is hard!
- A 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 x, so $\gcd(y, x-y) \leq \gcd(y, x)$
  - Putting these together: gcd(y, x-y) = 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?

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 `euclid` calls, both **a** and **b** are less than half of what they were, (i.e. reduced by at least 1 bit)
  - Thus if a and b have k bits, at most 2k recursive calls are needed.
  - Each recursive call involves a division, $\Theta(k^2)$
  - Thus entire algorithm is at most $k^2 * 2k$, which is in $O(k^3)$
  - You can look up refinements of this.

Euclid's Algorithm: practical use

- Divide 210 by 45, and get the result 4 with remainder 30, so $210=4\cdot45+30$.
- Divide 45 by 30, and get the result 1 with remainder 15, so $45=1\cdot30+15$.
- Divide 30 by 15, and get the result 2 with remainder 0, so $30=2\cdot15+0$.
- The greatest common divisor of 210 and 45 is 15.

gcd and linear combinations

- Lemma: If **d** is a common divisor of **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** divides both **a** and **b**. No common divisor can exceed their 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, for any given a and b, find the x and y as in the lemma, we have found the gcd.
- It turns out that a simple modification of Euclid's algorithm will allow us to calculate the x and y.

Forward-backward Example: gcd (33, 14)

- 33 = 2*14 + 5
- 14 = 2 * 5 + 4
- 5 = 1 * 4 + 1
- 4 = 4 * 1 + 0, so gcd(33, 14) = 1.
- **Now work backwards**
- 1 = 5 - 4. Substitute 4 = 14 - 2*5.
- 1 = 5 – (14 - 2*5) = 3*5 - 14. Substitute 5 = 33 - 2*14
- 1 = 3(33 - 2*14) -14 = 3 * 33 – 7 * 14
- Thus x = 3 and y = -7 Done!

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
  - I decided that it is a bit advanced for students who may have just seen Modular Arithmetic for the first time yesterday.
  - If you are interested, look up “extended Euclid proof”
  - We’ll do a couple of convincing examples.

Another example (same basic computation, different order): gcd (97, 20)

- 97 = 4·20+17
- 20 = 1·17+3
- 17 = 5·3+2
- 3 = 1·2+1 so GCD is 1.
- **Now figure out the x and y**
- 17 = 1·97-4·20
- 20-1·17 = 3 so 3 = 1·20-1·17 = 1·20-(1·97-4·20) = -1·97+5·20
- 17=5·3+2 so 2 = 17-5·3 = (1·97-4·20)-5(-1·97+5·20) = 6·97-29·20
- 1 = 3-2 = (-1·97+5·20)-(6·97-29·20) = -7·97+34·20
- Thus x = -7 and y = 34 Done!