# A Random Walk through CS70, Pt. III: Number Theory, Polynomials, etc.

CS70 Summer 2016 - Lecture 8D

David Dinh

11 August 2016

UC Berkeley

# Today

Last lecture!

Fun with number theory and polynomials.

Again, slides marked with a * are totally optional “fun stuff”.

# Modular Arithmetic

Covered in more detail in M115.

$x$ is congruent to $y$ modulo $m$, denoted "$x \equiv y \pmod{m}$"

# Modular Arithmetic

Covered in more detail in M115.

$x$ is congruent to $y$ modulo $m$, denoted "$x \equiv y \pmod{m}$" if and only if $(x - y)$ is divisible by $m$ (denoted $m|(x - y)$)

# Modular Arithmetic

Covered in more detail in M115.

$x$ is congruent to $y$ modulo $m$, denoted "$x \equiv y \pmod{m}$" if and only if $(x - y)$ is divisible by $m$ (denoted $m|(x-y)$) if and only if $x$ and $y$ have the same remainder w.r.t. $m$.

# Modular Arithmetic

Covered in more detail in M115.

$x$ is congruent to $y$ modulo $m$, denoted "$x \equiv y \pmod{m}$" if and only if $(x-y)$ is divisible by $m$ (denoted $m|(x-y)$) if and only if $x$ and $y$ have the same remainder w.r.t. $m$. if and only if $x = y + km$ for some integer $k$.

# Modular Arithmetic

Covered in more detail in M115.

$x$ is congruent to $y$ modulo $m$, denoted "$x \equiv y \pmod{m}$" if and only if $(x-y)$ is divisible by $m$ (denoted $m|(x-y)$) if and only if $x$ and $y$ have the same remainder w.r.t. $m$. if and only if $x = y + km$ for some integer $k$.

Congruence partitions the integers into equivalence classes ("congruence classes"), e.g. these for mod 7: $\{\ldots, -7, 0, 7, 14, \ldots\}$, $\{\ldots, -6, 1, 8, 15, \ldots\}$.

# Modular Arithmetic

Covered in more detail in M115.

$x$ is congruent to $y$ modulo $m$, denoted "$x \equiv y \pmod{m}$" if and only if $(x - y)$ is divisible by $m$ (denoted $m|(x - y)$) if and only if $x$ and $y$ have the same remainder w.r.t. $m$. if and only if $x = y + km$ for some integer $k$.

Congruence partitions the integers into equivalence classes ("congruence classes"), e.g. these for mod 7: $\{\ldots, -7, 0, 7, 14, \ldots\}$, $\{\ldots, -6, 1, 8, 15, \ldots\}$.

If $a \equiv c \pmod{m}$ and $b \equiv d \pmod{m}$, then $a + b \equiv c + d \pmod{m}$ and $a \cdot b \equiv c \cdot d \pmod{m}$.

# Modular Arithmetic

Covered in more detail in M115.

$x$ is congruent to $y$ modulo $m$, denoted "$x \equiv y \pmod{m}$" if and only if $(x - y)$ is divisible by $m$ (denoted $m|(x - y)$) if and only if $x$ and $y$ have the same remainder w.r.t. $m$. if and only if $x = y + km$ for some integer $k$.

Congruence partitions the integers into equivalence classes ("congruence classes"), e.g. these for mod 7: $\{\ldots, -7, 0, 7, 14, \ldots\}$, $\{\ldots, -6, 1, 8, 15, \ldots\}$.

If $a \equiv c \pmod{m}$ and $b \equiv d \pmod{m}$, then $a + b \equiv c + d \pmod{m}$ and $a \cdot b \equiv c \cdot d \pmod{m}$.

Division: multiplication by multiplicative inverse. How do we find MI? EGCD!

# Euclidean Algorithm

Multiplicative inverse of $a \pmod{m}$ exists if and only if $\gcd(a, m) = 1$.

Find inverse (and check GCD) with extended Euclid.

# Euclidean Algorithm

Multiplicative inverse of $a \pmod{m}$ exists if and only if $\gcd(a,m) = 1$.

Find inverse (and check GCD) with extended Euclid.

Inputs: $x \geq y \geq 0$ with $x > 0$. Outputs: integers $(d, a, b)$ where $d = \gcd(x,y) = ax + by$.

# Euclidean Algorithm

Multiplicative inverse of $a \pmod{m}$ exists if and only if $\gcd(a,m)=1$.

Find inverse (and check GCD) with extended Euclid.

Inputs: $x \geq y \geq 0$ with $x > 0$. Outputs: integers $(d,a,b)$ where $d = \gcd(x,y) = ax+by$.

1. If $y=0$, return $(x,1,0)$: $x = 1x+0y$.
2. Otherwise, let $(d,a,b)$ be the return value of the extended GCD algorithm on $(y, x-y\lfloor x/y \rfloor)$.
3. Return $(d,b,a-b\lfloor x/y \rfloor)$.

# Euclidean Algorithm

Multiplicative inverse of $a \pmod m$ exists if and only if $\gcd(a, m) = 1$.

Find inverse (and check GCD) with extended Euclid.

Inputs: $x \geq y \geq 0$ with $x > 0$. Outputs: integers $(d, a, b)$ where $d = \gcd(x, y) = ax + by$.

1. If $y = 0$, return $(x, 1, 0)$: $x = 1x + 0y$.
2. Otherwise, let $(d, a, b)$ be the return value of the extended GCD algorithm on $(y, x - y\lfloor x/y \rfloor)$.
3. Return $(d, b, a - b\lfloor x/y \rfloor)$.

How do we find multiplicative inverse? Solve $ax + bm = 1$.

# Exponentiation in Modular Arithmetic

Repeated squaring!
$51^{43} \equiv 51^{32} \cdot 51^{8} \cdot 51^{2} \cdot 51^{1} \equiv (60) * (53) * (60) * (51) \equiv 2 \pmod{77}.$

# Exponentiation in Modular Arithmetic

Repeated squaring!
$51^{43} \equiv 51^{32} \cdot 51^{8} \cdot 51^{2} \cdot 51^{1} \equiv (60) * (53) * (60) * (51) \equiv 2 \pmod{77}$.

Euler's Theorem: Suppose $\gcd(a, n) = 1$. Then $a^{\phi(n)} \equiv 1 \pmod{n}$, where $\phi(n)$, the totient function, represents the number of numbers in $[1, n]$ that are relatively prime with $n$.

# Exponentiation in Modular Arithmetic

Repeated squaring!
$51^{43} \equiv 51^{32} \cdot 51^{8} \cdot 51^{2} \cdot 51^{1} \equiv (60) * (53) * (60) * (51) \equiv 2 \pmod{77}$.

Euler's Theorem: Suppose $\gcd(a, n) = 1$. Then $a^{\phi(n)} \equiv 1 \pmod{n}$, where $\phi(n)$, the totient function, represents the number of numbers in $[1, n]$ that are relatively prime with $n$.

Immediate corollary: Fermat's little theorem. Suppose $p$ is prime. Then $a^p \equiv a \pmod{p}$. Furthermore, if $p \not| a$, then $a^{p-1} \equiv 1 \pmod{p}$.

# (Another) Combinatorial Proof of FLT

How many ways are there to assign $a$ colors to $p$ numbers, $\{1, \ldots, p\}$ such that not all colors are the same?

# (Another) Combinatorial Proof of FLT

How many ways are there to assign $a$ colors to $p$ numbers, $\{1, \ldots, p\}$ such that not all colors are the same?

**Answer 1:** $a^p - a$ (all colorings - monochromatic ones).

# (Another) Combinatorial Proof of FLT

How many ways are there to assign $a$ colors to $p$ numbers, $\{1, \ldots, p\}$ such that not all colors are the same?

**Answer 1:** $a^p - a$ (all colorings - monochromatic ones).

**Answer 2:** Divide colorings into equivalence classes; two colorings are equivalent if I can get from one to the other by performing a shift.

# (Another) Combinatorial Proof of FLT

How many ways are there to assign $a$ colors to $p$ numbers, $\{1, \ldots, p\}$ such that not all colors are the same?

**Answer 1:** $a^p - a$ (all colorings - monochromatic ones).

**Answer 2:** Divide colorings into equivalence classes; two colorings are equivalent if I can get from one to the other by performing a shift. All colorings in class must be different. Why?

# (Another) Combinatorial Proof of FLT

How many ways are there to assign $a$ colors to $p$ numbers, $\{1, \dots, p\}$ such that not all colors are the same?

**Answer 1:** $a^p - a$ (all colorings - monochromatic ones).

**Answer 2:** Divide colorings into equivalence classes; two colorings are equivalent if I can get from one to the other by performing a shift. All colorings in class must be different. Why? If I can shift by some number smaller than $p$ to get back to my original result, that means that either the coloring isn't monochromatic, or that $p$ isn't a prime!

# (Another) Combinatorial Proof of FLT

How many ways are there to assign $a$ colors to $p$ numbers, $\{1, \ldots, p\}$ such that not all colors are the same?

**Answer 1:** $a^p - a$ (all colorings - monochromatic ones).

**Answer 2:** Divide colorings into equivalence classes; two colorings are equivalent if I can get from one to the other by performing a shift. All colorings in class must be different. Why? If I can shift by some number smaller than $p$ to get back to my original result, that means that either the coloring isn't monochromatic, or that $p$ isn't a prime! Size of each class is $p$ since we can shift $p$ ways. That means $a^p - a$ must be a multiple of $p$!

# Example Problem: Dot Product over Finite Fields

Here's a question that almost made it onto the final (removed on Tuesday since the final was getting long)

# Example Problem: Dot Product over Finite Fields

Here's a question that almost made it onto the final (removed on Tuesday since the final was getting long)

Let $A_1,\ldots,A_n,B_1,\ldots,B_n$ be numbers in $\{0,\ldots,p-1\}$ for some prime number $p$. At least one of them is not zero. We pick $w_1,\ldots,w_n$, where each $w_i$ is picked from the set $\{0,\ldots,p-1\}$ uniformly at random. Let $\alpha = \sum_i w_i A_i$ and $\beta = \sum_i w_i B_i$ . You may assume at least one of the $A_i$s and at least one of the $B_i$s are nonzero.

1. **(11 points)** What is the probability that $\alpha = 0 \pmod{p}$?
2. **(11 points)** Give a strictly positive (non zero) lower bound to the probability that $\alpha \cdot \beta$ is not equal to zero. (Hint: union bound)

# Dot Product over Finite Fields, Solution

Part 1:

- Case 1: *Two or more $A_i$'s are non-zero.* Look at the coefficient $i$ of one of the non-zero ones. In order to make the sum non-zero, $w_i A_i$ must be equal to $S = \sum_{j \neq i} w_j A_j$. Therefore, we are asking for the probability that $w_i A_i = S$, which is $1/p$.

# Dot Product over Finite Fields, Solution

Part 1:

- Case 1: *Two or more $A_i$'s are non-zero.* Look at the coefficient $i$ of one of the non-zero ones. In order to make the sum non-zero, $w_i A_i$ must be equal to $S = \sum_{j \neq i} w_j A_j$. Therefore, we are asking for the probability that $w_i A_i = S$, which is $1/p$.
- Case 2: *Exactly one $A_i$ is non-zero.* Make its coefficient zero.

# Dot Product over Finite Fields, Solution

Part 1:

- Case 1: *Two or more $A_i$'s are non-zero.* Look at the coefficient $i$ of one of the non-zero ones. In order to make the sum non-zero, $w_iA_i$ must be equal to $S = \sum_{j \neq i} w_jA_j$. Therefore, we are asking for the probability that $w_iA_i = S$, which is $1/p$.
- Case 2: *Exactly one $A_i$ is non-zero.* Make its coefficient zero.

Probability for part 1: $1/p$.

# Dot Product over Finite Fields, Solution

Part 1:

- Case 1: *Two or more $A_i$'s are non-zero.* Look at the coefficient $i$ of one of the non-zero ones. In order to make the sum non-zero, $w_i A_i$ must be equal to $S = \sum_{j \neq i} w_j A_j$. Therefore, we are asking for the probability that $w_i A_i = S$, which is $1/p$.
- Case 2: *Exactly one $A_i$ is non-zero.* Make its coefficient zero.

Probability for part 1: $1/p$.

Part 2:

# Dot Product over Finite Fields, Solution

Part 1:

- Case 1: *Two or more $A_i$'s are non-zero.* Look at the coefficient $i$ of one of the non-zero ones. In order to make the sum non-zero, $w_iA_i$ must be equal to $S = \sum_{j \neq i} w_jA_j$. Therefore, we are asking for the probability that $w_iA_i = S$, which is $1/p$.
- Case 2: *Exactly one $A_i$ is non-zero.* Make its coefficient zero.

Probability for part 1: $1/p$.

Part 2:

$$Pr[\alpha\beta \neq 0] = 1 - \Pr[\alpha\beta = 0]$$

# Dot Product over Finite Fields, Solution

Part 1:

- Case 1: *Two or more $A_i$'s are non-zero.* Look at the coefficient $i$ of one of the non-zero ones. In order to make the sum non-zero, $w_iA_i$ must be equal to $S = \sum_{j \neq i} w_jA_j$. Therefore, we are asking for the probability that $w_iA_i = S$, which is $1/p$.
- Case 2: *Exactly one $A_i$ is non-zero.* Make its coefficient zero.

Probability for part 1: $1/p$.

Part 2:

$$Pr[\alpha\beta \neq 0] = 1 - \Pr[\alpha\beta = 0]$$

$$\Pr[\alpha\beta = 0] = Pr[\alpha = 0 \cup \beta = 0] \leq Pr[\alpha = 0] + Pr[\beta = 0] = \frac{2}{p}$$

# Cryptography

Simple private-key scheme: encrypt the message by bitwise XOR-ing with plaintext. Problems: huge key size, reliance on a shared secret, one-time key.

# Cryptography

Simple private-key scheme: encrypt the message by bitwise XOR-ing with plaintext. Problems: huge key size, reliance on a shared secret, one-time key.

RSA:

- **Key generation:** Recipient: compute $p$ and $q$, let $N = pq$. Choose some $e$ relatively prime to $(p-1)(q-1)$ (normally small, say, 3), and then computes $d = e^{-1} \mod (p-1)(q-1)$. Public key: $(N, e)$. Private key: $(N, d)$.

# Cryptography

Simple private-key scheme: encrypt the message by bitwise XOR-ing with plaintext. Problems: huge key size, reliance on a shared secret, one-time key.

RSA:

- **Key generation:** Recipient: compute $p$ and $q$, let $N = pq$. Choose some $e$ relatively prime to $(p-1)(q-1)$ (normally small, say, 3), and then computes $d = e^{-1} \mod (p-1)(q-1)$. Public key: $(N, e)$. Private key: $(N, d)$.
- **Encrypt:** Given plaintext $x$, sender computes ciphertext $c = E(x) = mod(x^e, N)$.

Simple private-key scheme: encrypt the message by bitwise XOR-ing with plaintext. Problems: huge key size, reliance on a shared secret, one-time key.

RSA:

- **Key generation:** Recipient: compute $p$ and $q$, let $N = pq$. Choose some $e$ relatively prime to $(p-1)(q-1)$ (normally small, say, 3), and then computes $d = e^{-1} \mod (p-1)(q-1)$. Public key: $(N, e)$. Private key: $(N, d)$.
- **Encrypt:** Given plaintext $x$, sender computes ciphertext $c = E(x) = mod(x^e, N)$.
- **Decrypt:** Recipient computes $x = D(c) = mod(c^d, N)$.

# Cryptography

Simple private-key scheme: encrypt the message by bitwise XOR-ing with plaintext. Problems: huge key size, reliance on a shared secret, one-time key.

RSA:

- **Key generation:** Recipient: compute $p$ and $q$, let $N = pq$. Choose some $e$ relatively prime to $(p-1)(q-1)$ (normally small, say, 3), and then computes $d = e^{-1} \mod (p-1)(q-1)$. Public key: $(N, e)$. Private key: $(N, d)$.
- **Encrypt:** Given plaintext $x$, sender computes ciphertext $c = E(x) = mod(x^e, N)$.
- **Decrypt:** Recipient computes $x = D(c) = mod(c^d, N)$.

How did we find primes? Random sampling primes around $x$ gives around $1/\ln x$ of finding primes. Test with Fermat's primality test.

# Cryptography

Simple private-key scheme: encrypt the message by bitwise XOR-ing with plaintext. Problems: huge key size, reliance on a shared secret, one-time key.

RSA:

- **Key generation:** Recipient: compute $p$ and $q$, let $N = pq$. Choose some $e$ relatively prime to $(p-1)(q-1)$ (normally small, say, 3), and then computes $d = e^{-1} \mod (p-1)(q-1)$. Public key: $(N, e)$. Private key: $(N, d)$.
- **Encrypt:** Given plaintext $x$, sender computes ciphertext $c = E(x) = mod(x^e, N)$.
- **Decrypt:** Recipient computes $x = D(c) = mod(c^d, N)$.

How did we find primes? Random sampling primes around $x$ gives around $1/\ln x$ of finding primes. Test with Fermat's primality test.

Pick random $a$. Check if $a^{p-1} \equiv 1 \pmod{p}$.

# Cryptography

Simple private-key scheme: encrypt the message by bitwise XOR-ing with plaintext. Problems: huge key size, reliance on a shared secret, one-time key.

RSA:

- **Key generation:** Recipient: compute $p$ and $q$, let $N = pq$. Choose some $e$ relatively prime to $(p-1)(q-1)$ (normally small, say, 3), and then computes $d = e^{-1} \mod (p-1)(q-1)$. Public key: $(N, e)$. Private key: $(N, d)$.
- **Encrypt:** Given plaintext $x$, sender computes ciphertext $c = E(x) = mod(x^e, N)$.
- **Decrypt:** Recipient computes $x = D(c) = mod(c^d, N)$.

How did we find primes? Random sampling primes around $x$ gives around $1/\ln x$ of finding primes. Test with Fermat's primality test.

Pick random $a$. Check if $a^{p-1} \equiv 1 \pmod{p}$. No? then composite. Yes? Prime or Carmichael w.p. at least 1/2.

# Public Key Encryption, In General...

Security rests on difficulty of integer factorization. Are there other hard

What about other hardness assumptions?

Security rests on difficulty of integer factorization. Are there other hard

What about other hardness assumptions?

Discrete log! Make cryptosystems based on the (widely believed) hardness of solving $b^k = g$ in some finite group. ElGamal, Diffie-Hellman, elliptic curves.

# Public Key Encryption, In General...

Security rests on difficulty of integer factorization. Are there other hard

What about other hardness assumptions?

Discrete log! Make cryptosystems based on the (widely believed) hardness of solving $b^k = g$ in some finite group. ElGamal, Diffie-Hellman, elliptic curves.

Sometimes private key encryption isn't safe for small, easily recognizable plaintexts... what if you try to encrypt 0 as a ciphertext? Or if you're trying to send something like a social security number (only 9 digits - easily brute-forced).

# Public Key Encryption, In General...

Security rests on difficulty of integer factorization. Are there other hard

What about other hardness assumptions?

Discrete log! Make cryptosystems based on the (widely believed) hardness of solving $b^k = g$ in some finite group. ElGamal, Diffie-Hellman, elliptic curves.

Sometimes private key encryption isn't safe for small, easily recognizable plaintexts... what if you try to encrypt 0 as a ciphertext? Or if you're trying to send something like a social security number (only 9 digits - easily brute-forced). Padding and hybrid encryption.

# Public Key Encryption, In General...

Security rests on difficulty of integer factorization. Are there other hard

What about other hardness assumptions?

Discrete log! Make cryptosystems based on the (widely believed) hardness of solving $b^k = g$ in some finite group. ElGamal, Diffie-Hellman, elliptic curves.

Sometimes private key encryption isn't safe for small, easily recognizable plaintexts... what if you try to encrypt 0 as a ciphertext? Or if you're trying to send something like a social security number (only 9 digits - easily brute-forced). Padding and hybrid encryption.

Like this stuff? Want to learn more? CS276.

# Chinese Remainder Theorem

For two congruences: Suppose $\gcd(m, n) = 1$. Then the two equations $x \equiv a \pmod{m}$ and $x \equiv b \pmod{n}$ have a unique solution mod $mn$

# Chinese Remainder Theorem

For two congruences: Suppose $\gcd(m,n) = 1$. Then the two equations $x \equiv a \pmod{m}$ and $x \equiv b \pmod{n}$ have a unique solution mod $mn$

How did we find a solution? Find $c \equiv m^{-1}(b-a) \pmod{n}$. Then $x \equiv a + mc \pmod{mn}$.

# Chinese Remainder Theorem

For two congruences: Suppose $\gcd(m, n) = 1$. Then the two equations $x \equiv a \pmod{m}$ and $x \equiv b \pmod{n}$ have a unique solution mod $mn$

How did we find a solution? Find $c \equiv m^{-1}(b - a) \pmod{n}$. Then $x \equiv a + mc \pmod{mn}$.

Expand to more congruences to get CRT! Let $m_1, \ldots, m_k$ be relatively prime numbers. Then the $k$ equations $x \equiv a_1 \pmod{m_1}$, ..., $x \equiv a_k \pmod{m_k}$ have a unique solution mod $m_1 m_2 \ldots m_k$.

# Euler's Criterion and Square Roots

**Theorem (Euler's Criterion):** Suppose $p$ is an odd prime and $a$ is some integer relatively prime to $p$. Then $a^{(p-1)/2}$ is 1 (mod $p$) if and only if there exists some integer $x$ such that $a \equiv x^2 \pmod{p}$ and $-1$ otherwise.

# Euler's Criterion and Square Roots

**Theorem (Euler's Criterion):** Suppose $p$ is an odd prime and $a$ is some integer relatively prime to $p$. Then $a^{(p-1)/2}$ is 1 (mod $p$) if and only if there exists some integer $x$ such that $a \equiv x^2 \pmod{p}$ and $-1$ otherwise.

How to find the square root? If $p \equiv 3 \pmod{4}$, and the square roots exist, then square roots of $a$ mod $p$ are given by $\pm a^{(p+1)/4}$.

# Blum Coin Flipping

How to flip a coin over the phone?

1. Alex chooses distinct primes $p, q$ congruent to $3 \pmod 4$, and computes $n = pq$. He sends $n$ (but not $p$ and $q$) to David.

# Blum Coin Flipping

How to flip a coin over the phone?

1. Alex chooses distinct primes $p$, $q$ congruent to $3 \pmod 4$, and computes $n = pq$. He sends $n$ (but not $p$ and $q$) to David.
2. David chooses $x \in (0, n)$ relatively prime to $n$ and sends $a = x^2 \pmod n$ to Alex.

# Blum Coin Flipping

How to flip a coin over the phone?

1. Alex chooses distinct primes $p, q$ congruent to $3 \pmod 4$, and computes $n = pq$. He sends $n$ (but not $p$ and $q$) to David.
2. David chooses $x \in (0, n)$ relatively prime to $n$ and sends $a = x^2 \pmod n$ to Alex.
3. Alex, armed with knowledge of $p, q$, computes the square roots $\pm x, \pm y$ of $a$, mod $n$, and sends one to David.

# Blum Coin Flipping

How to flip a coin over the phone?

1. Alex chooses distinct primes $p$, $q$ congruent to $3 \pmod 4$, and computes $n = pq$. He sends $n$ (but not $p$ and $q$) to David.
2. David chooses $x \in (0, n)$ relatively prime to $n$ and sends $a = x^2 \pmod n$ to Alex.
3. Alex, armed with knowledge of $p$, $q$, computes the square roots $\pm x, \pm y$ of $a$, mod $n$, and sends one to David.
4. If David got $\pm x$, then he says Alex guessed correctly. Otherwise, if he gets $\pm y$, he can factor $n$ (since $pq|(x+y)(x-y)$) and use that to prove that he won.

# Algebraic Structures

**Group:** $(G, +)$ with $+$ having the properties of closure, associativity, existence of identity, existence of inverse.

**Group:** $(G, +)$ with $+$ having the properties of closure, associativity, existence of identity, existence of inverse.

**Abelian group:** add commutativity of $+$.

# Algebraic Structures

**Group:** $(G, +)$ with $+$ having the properties of closure, associativity, existence of identity, existence of inverse.

**Abelian group:** add commutativity of $+$.

**Ring:** add $\times$ with closure, associativity, existence of identity, and left/right distributivity over $+$.

# Algebraic Structures

**Group:** $(G, +)$ with $+$ having the properties of closure, associativity, existence of identity, existence of inverse.

**Abelian group:** add commutativity of $+$.

**Ring:** add $\times$ with closure, associativity, existence of identity, and left/right distributivity over $+$.

**Field:** add existence of inverse of $\times$ for all elements except additive identity.

# Algebraic Structures

**Group:** $(G, +)$ with $+$ having the properties of closure, associativity, existence of identity, existence of inverse.

**Abelian group:** add commutativity of $+$.

**Ring:** add $\times$ with closure, associativity, existence of identity, and left/right distributivity over $+$.

**Field:** add existence of inverse of $\times$ for all elements except additive identity.

**Galois field:** field with finitely many elements. In this class we look at prime fields: $(\mathbb{Z}_p, +, \times)$ where arithmetic is done mod $p$.

**Group:** $(G, +)$ with $+$ having the properties of closure, associativity, existence of identity, existence of inverse.

**Abelian group:** add commutativity of $+$.

**Ring:** add $\times$ with closure, associativity, existence of identity, and left/right distributivity over $+$.

**Field:** add existence of inverse of $\times$ for all elements except additive identity.

**Galois field:** field with finitely many elements. In this class we look at prime fields: $(\mathbb{Z}_p, +, \times)$ where arithmetic is done mod $p$.

This material is covered in much greater depth in M113.

# Polynomials

Uniquely specify by coefficients: $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_d x^d$...

# Polynomials

Uniquely specify by coefficients: $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_d x^d$...

... or by $d+1$ points.

# Polynomials

Uniquely specify by coefficients: $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_d x^d$...

... or by $d+1$ points.

Coefficients to points: just evaluate!

# Polynomials

Uniquely specify by coefficients: $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_d x^d$...

... or by $d+1$ points.

Coefficients to points: just evaluate!

Points to coefficients? Lagrange interpolation:

$$\Delta_i(x) \coloneqq \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)}$$

Sum these for all $i$.

# Polynomials

Uniquely specify by coefficients: $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_d x^d$...

... or by $d+1$ points.

Coefficients to points: just evaluate!

Points to coefficients? Lagrange interpolation:

$$\Delta_i(x) := \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)}$$

Sum these for all $i$.

Or set up the Vandermonde matrix and solve.

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^d \\ 1 & x_2 & x_2^2 & \dots & x_2^d \\ 1 & x_3 & x_3^2 & \dots & x_3^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{d+1} & x_{d+1}^2 & \dots & x_{d+1}^d \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_d \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{d+1} \end{bmatrix}$$

# Secret Sharing

1. Pick some prime $q > s, n$. We will operate in $GF(q)$.
2. Pick a degree-$k-1$ polynomial $P$ such that $P(0) = s$, i.e. $P(x) = s + a_1 x + a_2 x^2 + ... + a_{k-1} x^{k-1}$, where $a_1, ..., a_{k-1}$ are chosen randomly.
3. Give $P(i)$ to the $i$th official.
4. To recover the secret: have $k$ people get together and interpolate to find $P(0)$.

No information can be recovered with less than $k$ people if done over a prime field!

Take original message $(1, m_1), (2, m_2), ..., (n, m_n)$ in $GF(q)$ and then interpolate a polynomial.

Take original message $(1, m_1), (2, m_2), \ldots, (n, m_n)$ in $GF(q)$ and then interpolate a polynomial.

Send $k$ extra points. If $k$ drop, it's ok! Just interpolate and evaluate.

# Berlekamp-Welch

For corruption errors. $k$ packets corrupted. How many packets to send if message is $n$ packets long?

# Berlekamp-Welch

For corruption errors. $k$ packets corrupted. How many packets to send if message is $n$ packets long? $n+2k$.

# Berlekamp-Welch

For corruption errors. $k$ packets corrupted. How many packets to send if message is $n$ packets long? $n+2k$.

1. Alex interpolates a degree $n-1$ polynomial $P(x)$ over the messages, like for erasure codes.

# Berlekamp-Welch

For corruption errors. $k$ packets corrupted. How many packets to send if message is $n$ packets long? $n+2k$.

1. Alex interpolates a degree $n-1$ polynomial $P(x)$ over the messages, like for erasure codes.
2. Alex sends $n+2k$ points to David: $(1,P(1)),(2,P(2)),\ldots,(n+2k,P(n+2k))$.

# Berlekamp-Welch

For corruption errors. $k$ packets corrupted. How many packets to send if message is $n$ packets long? $n+2k$.

1. Alex interpolates a degree $n-1$ polynomial $P(x)$ over the messages, like for erasure codes.
2. Alex sends $n+2k$ points to David: $(1, P(1)), (2, P(2)), \ldots, (n+2k, P(n+2k))$.
3. David receieves $n+2k$ points $(1, r_1), (2, r_2), \ldots, (n+2k, r_{n+2k})$.

# Berlekamp-Welch

For corruption errors. $k$ packets corrupted. How many packets to send if message is $n$ packets long? $n+2k$.

1. Alex interpolates a degree $n-1$ polynomial $P(x)$ over the messages, like for erasure codes.
2. Alex sends $n+2k$ points to David: $(1, P(1)), (2, P(2)), \ldots, (n+2k, P(n+2k))$.
3. David receieves $n+2k$ points $(1, r_1), (2, r_2), \ldots, (n+2k, r_{n+2k})$.
4. David writes down a system of equations:

   $$q_{n+k-1}x_i^{n+k-1} + \cdots + q_2 x_i^2 + q_1 x_i + q_0 = r_i(x_i^k + b_{k-1}x_i^{k-1} + \cdots + b_1 x_i + b_0)$$

   for each $x_i$.

# Berlekamp-Welch

For corruption errors. $k$ packets corrupted. How many packets to send if message is $n$ packets long? $n+2k$.

1. Alex interpolates a degree $n-1$ polynomial $P(x)$ over the messages, like for erasure codes.
2. Alex sends $n+2k$ points to David: $(1,P(1)),(2,P(2)),\dots,(n+2k,P(n+2k))$.
3. David receieves $n+2k$ points $(1,r_1),(2,r_2),\dots,(n+2k,r_{n+2k})$.
4. David writes down a system of equations:

   $$q_{n+k-1}x_i^{n+k-1}+\cdots+q_2x_i^2+q_1x_i+q_0=r_i(x_i^k+b_{k-1}x_i^{k-1}+\cdots+b_1x_i+b_0)$$

   for each $x_i$.
5. David solves the equations for the coefficients for $Q$ and $E$.

# Berlekamp-Welch

For corruption errors. $k$ packets corrupted. How many packets to send if message is $n$ packets long? $n+2k$.

1. Alex interpolates a degree $n-1$ polynomial $P(x)$ over the messages, like for erasure codes.
2. Alex sends $n+2k$ points to David: $(1,P(1)),(2,P(2)),\ldots,(n+2k,P(n+2k))$.
3. David receieves $n+2k$ points $(1,r_1),(2,r_2),\ldots,(n+2k,r_{n+2k})$.
4. David writes down a system of equations:

   $$q_{n+k-1}x_i^{n+k-1}+\cdots+q_2x_i^2+q_1x_i+q_0=r_i(x_i^k+b_{k-1}x_i^{k-1}+\cdots+b_1x_i+b_0)$$

   for each $x_i$.
5. David solves the equations for the coefficients for $Q$ and $E$.
6. David recovers $P(x)=Q(x)/E(x)$ by polynomial division.

# Berlekamp-Welch

For corruption errors. $k$ packets corrupted. How many packets to send if message is $n$ packets long? $n+2k$.

1. Alex interpolates a degree $n-1$ polynomial $P(x)$ over the messages, like for erasure codes.
2. Alex sends $n+2k$ points to David: $(1,P(1)),(2,P(2)),\ldots,(n+2k,P(n+2k))$.
3. David receieves $n+2k$ points $(1,r_1),(2,r_2),\ldots,(n+2k,r_{n+2k})$.
4. David writes down a system of equations:
$$q_{n+k-1}x_i^{n+k-1}+\cdots+q_2x_i^2+q_1x_i+q_0=r_i(x_i^k+b_{k-1}x_i^{k-1}+\cdots+b_1x_i+b_0)$$
for each $x_i$.
5. David solves the equations for the coefficients for $Q$ and $E$.
6. David recovers $P(x)=Q(x)/E(x)$ by polynomial division.

More on codes: EE121, EE229AB.

# Application/Research: PIT and Schwartz-Zippel*

**Theorem (Schwartz-Zippel Lemma)** : Let $Q(x_1, \ldots, x_n)$ be a *multivariate* polynomial of *total degree* $d$ (i.e. the sum of the powers of all the variables in a term are at most $d$) over some field $F$.

# Application/Research: PIT and Schwartz-Zippel*

**Theorem (Schwartz-Zippel Lemma)** : Let $Q(x_1, \ldots, x_n)$ be a *multivariate* polynomial of *total degree* $d$ (i.e. the sum of the powers of all the variables in a term are at most $d$) over some field $F$. Fix a finite set $S \subseteq F$, and let $r_1, r_2, \ldots, r_n$ be chosen independently and uniformly at random from $S$.

# Application/Research: PIT and Schwartz-Zippel*

**Theorem (Schwartz-Zippel Lemma)** : Let $Q(x_1, ..., x_n)$ be a *multivariate* polynomial of *total degree* $d$ (i.e. the sum of the powers of all the variables in a term are at most $d$) over some field $F$. Fix a finite set $S \subseteq F$, and let $r_1, r_2, ..., r_n$ be chosen independently and uniformly at random from $S$. Then $\Pr[Q(r_1, ..., r_n) = 0 | Q(x_1, ..., x_n) \not\equiv 0] \leq d/|S|$.

# Proof of SZ*

By induction on $n$.

# Proof of SZ*

By induction on $n$.

Base case: $n = 1$. Single variable polynomial.

# Proof of SZ*

By induction on $n$.

Base case: $n = 1$. Single variable polynomial. At most $d$ roots, so probability of getting a zero is at most $d/|S|$.

# Proof of SZ*

By induction on $n$.

Base case: $n = 1$. Single variable polynomial. At most $d$ roots, so probability of getting a zero is at most $d/|S|$.

Inductive step: assume SZ works up to $n - 1$ variable polynomials.

# Proof of SZ*

By induction on $n$.

Base case: $n = 1$. Single variable polynomial. At most $d$ roots, so probability of getting a zero is at most $d/|S|$.

Inductive step: assume SZ works up to $n - 1$ variable polynomials. Suppose $Q$ is not actually the zero polynomial (i.e. doesn't evaluate to 0 everywhere).

# Proof of SZ*

By induction on $n$.

Base case: $n = 1$. Single variable polynomial. At most $d$ roots, so probability of getting a zero is at most $d/|S|$.

Inductive step: assume SZ works up to $n-1$ variable polynomials. Suppose $Q$ is not actually the zero polynomial (i.e. doesn't evaluate to 0 everywhere). Group terms based on $x_1$:
$Q(x_1, \ldots, x_n) = \sum_{i=0}^{k} x_1^i Q_i(x_2, \ldots, x_n)$ where $k$ is the largest exponent of $x_1$ in $Q$, and each $Q_i$ is nonzero.

# Proof of SZ*

By induction on $n$.

Base case: $n = 1$. Single variable polynomial. At most $d$ roots, so probability of getting a zero is at most $d/|S|$.

Inductive step: assume SZ works up to $n-1$ variable polynomials. Suppose $Q$ is not actually the zero polynomial (i.e. doesn't evaluate to 0 everywhere). Group terms based on $x_1$:
$Q(x_1, \ldots, x_n) = \sum_{i=0}^{k} x_1^i Q_i(x_2, \ldots, x_n)$ where $k$ is the largest exponent of $x_1$ in $Q$, and each $Q_i$ is nonzero.

Condition on $x_2 = r_2, \ldots, x_n = r_n$.

# Proof of SZ*

By induction on $n$.

Base case: $n = 1$. Single variable polynomial. At most $d$ roots, so probability of getting a zero is at most $d/|S|$.

Inductive step: assume SZ works up to $n-1$ variable polynomials. Suppose $Q$ is not actually the zero polynomial (i.e. doesn't evaluate to 0 everywhere). Group terms based on $x_1$: $Q(x_1, ..., x_n) = \sum_{i=0}^{k} x_1^i Q_i(x_2, ..., x_n)$ where $k$ is the largest exponent of $x_1$ in $Q$, and each $Q_i$ is nonzero.

Condition on $x_2 = r_2, ..., x_n = r_n$.

By inductive step, we know that $Q_k(r_2, ..., r_n) = 0$ w.p. at most $(d-k)/|S|$ since total degree of $Q_k$ is at most $d-k$.

# Proof of SZ*

By induction on $n$.

Base case: $n = 1$. Single variable polynomial. At most $d$ roots, so probability of getting a zero is at most $d/|S|$.

Inductive step: assume SZ works up to $n-1$ variable polynomials. Suppose $Q$ is not actually the zero polynomial (i.e. doesn't evaluate to 0 everywhere). Group terms based on $x_1$: $Q(x_1,...,x_n) = \sum_{i=0}^{k} x_1^i Q_i(x_2,...,x_n)$ where $k$ is the largest exponent of $x_1$ in $Q$, and each $Q_i$ is nonzero.

Condition on $x_2 = r_2, ..., x_n = r_n$.

By inductive step, we know that $Q_k(r_2,...,r_n) = 0$ w.p. at most $(d-k)/|S|$ since total degree of $Q_k$ is at most $d-k$.

Now suppose $Q_k(r_2,...,r_n) \neq 0$. Then $q(x_1) = Q(x_1, r_2, ..., r_n)$ is a nonzero single-variable polynomial, so $q(r_1)$ is zero w.p. at most $k/|S|$.

# Proof of SZ*

By induction on $n$.

Base case: $n = 1$. Single variable polynomial. At most $d$ roots, so probability of getting a zero is at most $d/|S|$.

Inductive step: assume SZ works up to $n-1$ variable polynomials. Suppose $Q$ is not actually the zero polynomial (i.e. doesn't evaluate to 0 everywhere). Group terms based on $x_1$: $Q(x_1,...,x_n) = \sum_{i=0}^{k} x_1^i Q_i(x_2,...,x_n)$ where $k$ is the largest exponent of $x_1$ in $Q$, and each $Q_i$ is nonzero.

Condition on $x_2 = r_2, ..., x_n = r_n$.

By inductive step, we know that $Q_k(r_2,...,r_n) = 0$ w.p. at most $(d-k)/|S|$ since total degree of $Q_k$ is at most $d-k$.

Now suppose $Q_k(r_2,...,r_n) \neq 0$. Then $q(x_1) = Q(x_1, r_2, ..., r_n)$ is a nonzero single-variable polynomial, so $q(r_1)$ is zero w.p. at most $k/|S|$.

# Proof of SZ, II*

So:

$$\begin{aligned}\Pr(Q(r_1,\ldots,r_n)=0) &= \Pr(Q=0|Q_k=0)Pr(Q_k=0)+\\ &\quad \Pr(Q=0|Q_k\neq 0)\Pr(Q_k\neq 0)\end{aligned}$$

# Proof of SZ, II*

So:

$$
\begin{aligned}
\Pr(Q(r_1, \ldots, r_n) = 0) &= \Pr(Q = 0 | Q_k = 0) Pr(Q_k = 0) + \\
&\quad \Pr(Q = 0 | Q_k \neq 0) \Pr(Q_k \neq 0) \\
&\leq 1 \left( \frac{d-k}{|S|} \right) + \left( \frac{k}{|S|} \right) 1
\end{aligned}
$$

# Proof of SZ, II*

So:

$$\begin{aligned}
\Pr(Q(r_1,\ldots,r_n)=0) &= \Pr(Q=0|Q_k=0)Pr(Q_k=0)+ \\
&\quad \Pr(Q=0|Q_k\neq 0)\Pr(Q_k\neq 0) \\
&\leq 1\left(\frac{d-k}{|S|}\right)+\left(\frac{k}{|S|}\right)1 \\
&= \frac{d}{|S|}
\end{aligned}$$

$\square$

# Application: Finding Perfect Matchings*

Remember definition of perfect matching from MT1?

# Application: Finding Perfect Matchings*

Remember definition of perfect matching from MT1?

Bipartite graph. Each node on left matched with exactly one node on right by an edge.

# Application: Finding Perfect Matchings*

Remember definition of perfect matching from MT1?

Bipartite graph. Each node on left matched with exactly one node on right by an edge.

**Theorem** (Edmonds): Let $A$ be the matrix obtained from a bipartite graph $G = (U, V, E)$ as follows:

$$A_{ij} = \begin{cases} x_{ij} & \text{if } u_i, v_j \in E \\ 0 & \text{otherwise} \end{cases}$$

Then $G$ has a perfect matching if and only if $\det A \not\equiv 0$.

# Application: Finding Perfect Matchings*

Remember definition of perfect matching from MT1?

Bipartite graph. Each node on left matched with exactly one node on right by an edge.

**Theorem** (Edmonds): Let $A$ be the matrix obtained from a bipartite graph $G = (U, V, E)$ as follows:

$$A_{ij} = \begin{cases} x_{ij} & \text{if } u_i, v_j \in E \\ 0 & \text{otherwise} \end{cases}$$

Then $G$ has a perfect matching if and only if $\det A \not\equiv 0$.

**Proof sketch:** based on definition of determinant:

$$\det A = \sum_{\text{permutations } \pi} \operatorname{sign}(\pi) A_{1,\pi(1)} A_{2,\pi(2)}, \ldots, A_{n,\pi(n)}$$

Zero in each term if there is no perfect matching (missing edge), nonzero otherwise. No cancellations because no two terms have same set of variables. $\square$

# Perfect Matchings II

Determinant is just a polynomial! Use Schwartz-Zippel to test by plugging random values into the matrix.

Interested in this topic? CS270.

# Perfect Matchings II

Determinant is just a polynomial! Use Schwartz-Zippel to test by plugging random values into the matrix.

Interested in this topic? CS270.

Can we do this without randomness? Hot research topic! Derandomization has a lot of consequences in complexity theory.

# Perfect Matchings II

Determinant is just a polynomial! Use Schwartz-Zippel to test by plugging random values into the matrix.

Interested in this topic? CS270.

Can we do this without randomness? Hot research topic! Derandomization has a lot of consequences in complexity theory.

Hardness $\iff$ derandomization.

# Perfect Matchings II

Determinant is just a polynomial! Use Schwartz-Zippel to test by plugging random values into the matrix.

Interested in this topic? CS270.

Can we do this without randomness? Hot research topic! Derandomization has a lot of consequences in complexity theory.

Hardness $\iff$ derandomization.

# Conclusion

We hope you've enjoyed this semester and learned a lot.

Before CS70:

After CS70:

Thanks for taking CS70!