# A Random Walk through CS70

CS70 Summer 2016 - Lecture 8B

David Dinh

09 August 2016

UC Berkeley

# Today (and tomorrow, and Wednesday)

Review: what have we done in class?

Future classes: where do you go next?

Applications: how is the stuff you learned in 70 useful in the real world?

Research frontiers: what are people in academia working on (related to 70) right now?

Gigs: interesting stuff with material for fun and practice!

**Announcement:** No scantron HKN surveys now (or ever again!). Everything's online now. You should have received an email about this sometime in the previous week or two.

# Propositional Logic

We started with propositions...

# Propositional Logic

We started with propositions...Either true or false (forget about Godel for now).

# Propositional Logic

We started with propositions...Either true or false (forget about Godel for now).

Quantifiers: forall, exists.

# Propositional Logic

We started with propositions...Either true or false (forget about Godel for now).

Quantifiers: forall, exists.

Statements: "$9 = 1$". "David had two pieces of bread for breakfast.". "$\forall q \in \mathbb{R} : |q| \geq q$".

We started with propositions...Either true or false (forget about Godel for now).

Quantifiers: forall, exists.

Statements: “$9 = 1$”. “David had two pieces of bread for breakfast.”. “$\forall q \in \mathbb{R} : |q| \geq q$”.

Not statements: “$x = 5$”. “42”. “$\forall x \in \mathbb{R} : xy = 0$”.

# Propositional Logic

We started with propositions...Either true or false (forget about Godel for now).

Quantifiers: forall, exists.

Statements: "$9 = 1$". "David had two pieces of bread for breakfast.". "$\forall q \in \mathbb{R} : |q| \geq q$".

Not statements: "$x = 5$". "42". "$\forall x \in \mathbb{R} : xy = 0$".

Sanity check: Why is "$\forall q \in \mathbb{R} : |q| \geq q$" a statement but "$\forall x \in \mathbb{R} : xy = 0$" not a statement?

## Combining Statements

Boolean operators: and ($\wedge$), or ($\vee$), not ($\neg$ or a line over your expression, e.g. $\bar{x}$), conditional ($\implies$), biconditional ($\iff$).

# Combining Statements

Boolean operators: and ($\wedge$), or ($\vee$), not ($\neg$ or a line over your expression, e.g. $\bar{x}$), conditional ( $\implies$ ), biconditional ( $\iff$ ).

Prove things are equivalent by simplifying one (or both) sides, or with truth tables.

# Combining Statements

Boolean operators: and ($\wedge$), or ($\vee$), not ($\neg$ or a line over your expression, e.g. $\bar{x}$), conditional ( $\implies$ ), biconditional ( $\iff$ ).

Prove things are equivalent by simplifying one (or both) sides, or with truth tables.

Disprove things with counterexamples.

# Combining Statements

Boolean operators: and ($\wedge$), or ($\vee$), not ($\neg$ or a line over your expression, e.g. $\bar{x}$), conditional ( $\implies$ ), biconditional ( $\iff$ ).

Prove things are equivalent by simplifying one (or both) sides, or with truth tables.

Disprove things with counterexamples.

Example (SU14 MT1): True or false? $P \implies (Q \implies R) \equiv (P \wedge Q) \implies R$.

# Combining Statements

Boolean operators: and ($\wedge$), or ($\vee$), not ($\neg$ or a line over your expression, e.g. $\bar{x}$), conditional ( $\implies$ ), biconditional ( $\iff$ ).

Prove things are equivalent by simplifying one (or both) sides, or with truth tables.

Disprove things with counterexamples.

Example (SU14 MT1): True or false? $P \implies (Q \implies R) \equiv (P \wedge Q) \implies R$.

Intuitive method: guess whether or not this seems right or not.

# Combining Statements

Boolean operators: and ($\wedge$), or ($\vee$), not ($\neg$ or a line over your expression, e.g. $\bar{x}$), conditional ( $\implies$ ), biconditional ( $\iff$ ).

Prove things are equivalent by simplifying one (or both) sides, or with truth tables.

Disprove things with counterexamples.

Example (SU14 MT1): True or false? $P \implies (Q \implies R) \equiv (P \wedge Q) \implies R$.

Intuitive method: guess whether or not this seems right or not. If $P$ true? Then $Q$ true, then $R$ true. If $P$ not true, then both sides reduce to $Q \implies R$ so they're equivalent.

# Combining Statements

Boolean operators: and ($\wedge$), or ($\vee$), not ($\neg$ or a line over your expression, e.g. $\bar{x}$), conditional ( $\implies$ ), biconditional ( $\iff$ ).

Prove things are equivalent by simplifying one (or both) sides, or with truth tables.

Disprove things with counterexamples.

Example (SU14 MT1): True or false? $P \implies (Q \implies R) \equiv (P \wedge Q) \implies R$.

Intuitive method: guess whether or not this seems right or not. If $P$ true? Then $Q$ true, then $R$ true. If $P$ not true, then both sides reduce to $Q \implies R$ so they're equivalent.

Symbolic manipulation approach:

$$P \implies (Q \implies R)$$

# Combining Statements

Boolean operators: and ($\wedge$), or ($\vee$), not ($\neg$ or a line over your expression, e.g. $\bar{x}$), conditional ( $\implies$ ), biconditional ( $\iff$ ).

Prove things are equivalent by simplifying one (or both) sides, or with truth tables.

Disprove things with counterexamples.

Example (SU14 MT1): True or false? $P \implies (Q \implies R) \equiv (P \wedge Q) \implies R$.

Intuitive method: guess whether or not this seems right or not. If $P$ true? Then $Q$ true, then $R$ true. If $P$ not true, then both sides reduce to $Q \implies R$ so they're equivalent.

Symbolic manipulation approach:

$$P \implies (Q \implies R) \quad \equiv \quad \overline{P} \vee (Q \implies R)$$

# Combining Statements

Boolean operators: and ($\wedge$), or ($\vee$), not ($\neg$ or a line over your expression, e.g. $\bar{x}$), conditional ( $\implies$ ), biconditional ( $\iff$ ).

Prove things are equivalent by simplifying one (or both) sides, or with truth tables.

Disprove things with counterexamples.

Example (SU14 MT1): True or false? $P \implies (Q \implies R) \equiv (P \wedge Q) \implies R$.

Intuitive method: guess whether or not this seems right or not. If $P$ true? Then $Q$ true, then $R$ true. If $P$ not true, then both sides reduce to $Q \implies R$ so they're equivalent.

Symbolic manipulation approach:

$$P \implies (Q \implies R) \quad \equiv \quad \overline{P} \vee (Q \implies R) \equiv \overline{P} \vee (\overline{Q} \vee R)$$

# Combining Statements

Boolean operators: and ($\wedge$), or ($\vee$), not ($\neg$ or a line over your expression, e.g. $\bar{x}$), conditional ( $\implies$ ), biconditional ( $\iff$ ).

Prove things are equivalent by simplifying one (or both) sides, or with truth tables.

Disprove things with counterexamples.

Example (SU14 MT1): True or false? $P \implies (Q \implies R) \equiv (P \wedge Q) \implies R$.

Intuitive method: guess whether or not this seems right or not. If $P$ true? Then $Q$ true, then $R$ true. If $P$ not true, then both sides reduce to $Q \implies R$ so they're equivalent.

Symbolic manipulation approach:

$$P \implies (Q \implies R) \quad \equiv \quad \overline{P} \vee (Q \implies R) \equiv \overline{P} \vee (\overline{Q} \vee R) \equiv (\overline{P} \vee \overline{Q}) \vee R$$

# Combining Statements

Boolean operators: and ($\wedge$), or ($\vee$), not ($\neg$ or a line over your expression, e.g. $\bar{x}$), conditional ($\implies$), biconditional ($\iff$).

Prove things are equivalent by simplifying one (or both) sides, or with truth tables.

Disprove things with counterexamples.

Example (SU14 MT1): True or false? $P \implies (Q \implies R) \equiv (P \wedge Q) \implies R$.

Intuitive method: guess whether or not this seems right or not. If $P$ true? Then $Q$ true, then $R$ true. If $P$ not true, then both sides reduce to $Q \implies R$ so they're equivalent.

Symbolic manipulation approach:

$$
\begin{aligned}
P \implies (Q \implies R) &\equiv \overline{P} \vee (Q \implies R) \equiv \overline{P} \vee (\overline{Q} \vee R) \equiv (\overline{P} \vee \overline{Q}) \vee R \\
&\equiv \overline{(P \wedge Q)} \vee R
\end{aligned}
$$

# Combining Statements

Boolean operators: and ($\wedge$), or ($\vee$), not ($\neg$ or a line over your expression, e.g. $\bar{x}$), conditional ( $\implies$ ), biconditional ( $\iff$ ).

Prove things are equivalent by simplifying one (or both) sides, or with truth tables.

Disprove things with counterexamples.

Example (SU14 MT1): True or false? $P \implies (Q \implies R) \equiv (P \wedge Q) \implies R$.

Intuitive method: guess whether or not this seems right or not. If $P$ true? Then $Q$ true, then $R$ true. If $P$ not true, then both sides reduce to $Q \implies R$ so they're equivalent.

Symbolic manipulation approach:

$$\begin{aligned} P \implies (Q \implies R) &\equiv \overline{P} \vee (Q \implies R) \equiv \overline{P} \vee (\overline{Q} \vee R) \equiv (\overline{P} \vee \overline{Q}) \vee R \\ &\equiv \overline{(P \wedge Q)} \vee R \equiv (P \wedge Q) \implies R \end{aligned}$$

# Applications: Circuits

Boolean logic encompasses a lot of computation...

# Applications: Circuits

Boolean logic encompasses a lot of computation...

Circuits!

# Applications: Circuits

Boolean logic encompasses a lot of computation...

Circuits! Boolean circuits are in your computers. And gates, or gates, NAND gates, etc. Take boolean inputs and give some output. CS61C.

# Applications: Circuits

Boolean logic encompasses a lot of computation...

Circuits! Boolean circuits are in your computers. And gates, or gates, NAND gates, etc. Take boolean inputs and give some output. CS61C.

Want to squeeze more processing power onto my chip using less gates. Can I take a Boolean circuit and simplify it?

# Applications: Circuits

Boolean logic encompasses a lot of computation...

Circuits! Boolean circuits are in your computers. And gates, or gates, NAND gates, etc. Take boolean inputs and give some output. CS61C.

Want to squeeze more processing power onto my chip using less gates. Can I take a Boolean circuit and simplify it? Estimates? K-maps (also 61C).

# Applications: Circuits

Boolean logic encompasses a lot of computation...

Circuits! Boolean circuits are in your computers. And gates, or gates, NAND gates, etc. Take boolean inputs and give some output. CS61C.

Want to squeeze more processing power onto my chip using less gates. Can I take a Boolean circuit and simplify it? Estimates? K-maps (also 61C). Best way to do it? CIRCUIT-MIN problem. *Really* hard problem ("$\Sigma_2^P$ complete" - take CS172 to find out what that means.)

# Applications: Circuits

Boolean logic encompasses a lot of computation...

Circuits! Boolean circuits are in your computers. And gates, or gates, NAND gates, etc. Take boolean inputs and give some output. CS61C.

Want to squeeze more processing power onto my chip using less gates. Can I take a Boolean circuit and simplify it? Estimates? K-maps (also 61C). Best way to do it? CIRCUIT-MIN problem. *Really* hard problem ("$\Sigma_2^P$ complete" - take CS172 to find out what that means.)

How big do circuits need to be in order to compute some function that you're interested in?

# Applications: Circuits

Boolean logic encompasses a lot of computation...

Circuits! Boolean circuits are in your computers. And gates, or gates, NAND gates, etc. Take boolean inputs and give some output. CS61C.

Want to squeeze more processing power onto my chip using less gates. Can I take a Boolean circuit and simplify it? Estimates? K-maps (also 61C). Best way to do it? CIRCUIT-MIN problem. *Really* hard problem ("$\Sigma_2^P$ complete" - take CS172 to find out what that means.)

How big do circuits need to be in order to compute some function that you're interested in? *Circuit lower bounds.* Hard problem. Lots of research going on about this. Absolute lower bounds that don't depend on unproven assumptions are pretty primitive (think lower bounds for computing whether you have an even number of 1s as input).

# Applications: Satisfiability

Motivating example: $\exists x, y, z : (x \vee y \vee \bar{z}) \wedge (\bar{x} \vee y \vee z)$?

# Applications: Satisfiability

Motivating example: $\exists x, y, z : (x \vee y \vee \bar{z}) \wedge (\bar{x} \vee y \vee z)$? Yes, let $x$ be true, $y$ be true, and $z$ be anything.

# Applications: Satisfiability

Motivating example: $\exists x, y, z : (x \vee y \vee \bar{z}) \wedge (\bar{x} \vee y \vee z)$? Yes, let $x$ be true, $y$ be true, and $z$ be anything.

Generally: given some Boolean formula...

# Applications: Satisfiability

Motivating example: $\exists x, y, z : (x \vee y \vee \bar{z}) \wedge (\bar{x} \vee y \vee z)$? Yes, let $x$ be true, $y$ be true, and $z$ be anything.

Generally: given some Boolean formula… Is there some assignment of variables that makes it true? Do *all* assignments of variables make it true?

# Applications: Satisfiability

Motivating example: $\exists x, y, z : (x \vee y \vee \bar{z}) \wedge (\bar{x} \vee y \vee z)$? Yes, let $x$ be true, $y$ be true, and $z$ be anything.

Generally: given some Boolean formula… Is there some assignment of variables that makes it true? Do *all* assignments of variables make it true?

*Satisfiability* and *tautology* problems. Also well studied!

# Applications: Satisfiability

Motivating example: $\exists x, y, z : (x \vee y \vee \bar{z}) \wedge (\bar{x} \vee y \vee z)$? Yes, let $x$ be true, $y$ be true, and $z$ be anything.

Generally: given some Boolean formula... Is there some assignment of variables that makes it true? Do *all* assignments of variables make it true?

*Satisfiability* and *tautology* problems. Also well studied! Satisfiability in the general case actually models almost any computation we care about.

# Applications: Satisfiability

Motivating example: $\exists x, y, z : (x \vee y \vee \bar{z}) \wedge (\bar{x} \vee y \vee z)$? Yes, let $x$ be true, $y$ be true, and $z$ be anything.

Generally: given some Boolean formula... Is there some assignment of variables that makes it true? Do *all* assignments of variables make it true?

*Satisfiability* and *tautology* problems. Also well studied! Satisfiability in the general case actually models almost any computation we care about. Hard to solve in the general case though.

CS170, CS172 for info on why this is hard.

# Applications: Satisfiability

Motivating example: $\exists x, y, z : (x \vee y \vee \bar{z}) \wedge (\bar{x} \vee y \vee z)$? Yes, let $x$ be true, $y$ be true, and $z$ be anything.

Generally: given some Boolean formula... Is there some assignment of variables that makes it true? Do *all* assignments of variables make it true?

*Satisfiability* and *tautology* problems. Also well studied! Satisfiability in the general case actually models almost any computation we care about. Hard to solve in the general case though.

CS170, CS172 for info on why this is hard.

Sometimes we can estimate. Or we can satisfy some portion of the formula but not all.

# Applications: Satisfiability

Motivating example: $\exists x, y, z : (x \vee y \vee \bar{z}) \wedge (\bar{x} \vee y \vee z)$? Yes, let $x$ be true, $y$ be true, and $z$ be anything.

Generally: given some Boolean formula... Is there some assignment of variables that makes it true? Do *all* assignments of variables make it true?

*Satisfiability* and *tautology* problems. Also well studied! Satisfiability in the general case actually models almost any computation we care about. Hard to solve in the general case though.

CS170, CS172 for info on why this is hard.

Sometimes we can estimate. Or we can satisfy some portion of the formula but not all. CS174.

# Applications: Satisfiability

Motivating example: $\exists x, y, z : (x \vee y \vee \bar{z}) \wedge (\bar{x} \vee y \vee z)$? Yes, let $x$ be true, $y$ be true, and $z$ be anything.

Generally: given some Boolean formula... Is there some assignment of variables that makes it true? Do *all* assignments of variables make it true?

*Satisfiability* and *tautology* problems. Also well studied! Satisfiability in the general case actually models almost any computation we care about. Hard to solve in the general case though.

CS170, CS172 for info on why this is hard.

Sometimes we can estimate. Or we can satisfy some portion of the formula but not all. CS174.

What's the probability that some formula has a satisfying assignment? Counting/probabilistic arguments. Interesting results. Phase transitions.

Direct proof. Just go and prove it!

# Proofs: Techniques

Direct proof. Just go and prove it!

Contraposition. To prove $p \implies q$, prove that $\overline{q} \implies \overline{p}$.

# Proofs: Techniques

Direct proof. Just go and prove it!

Contraposition. To prove $p \implies q$, prove that $\overline{q} \implies \overline{p}$.

Contradiction. Suppose that what you're trying to prove is wrong. Prove that the universe implodes.

# Proofs: Techniques

Direct proof. Just go and prove it!

Contraposition. To prove $p \implies q$, prove that $\overline{q} \implies \overline{p}$.

Contradiction. Suppose that what you're trying to prove is wrong. Prove that the universe implodes.

Casewise. Split into cases. Make sure one of those cases always applies!

# Proofs: Techniques

Direct proof. Just go and prove it!

Contraposition. To prove $p \implies q$, prove that $\overline{q} \implies \overline{p}$.

Contradiction. Suppose that what you're trying to prove is wrong. Prove that the universe implodes.

Casewise. Split into cases. Make sure one of those cases always applies!

Combinatorial Proofs. Count something two ways. Those two quantities must be the same.

# Proofs: Techniques

Direct proof. Just go and prove it!

Contraposition. To prove $p \implies q$, prove that $\overline{q} \implies \overline{p}$.

Contradiction. Suppose that what you're trying to prove is wrong. Prove that the universe implodes.

Casewise. Split into cases. Make sure one of those cases always applies!

Combinatorial Proofs. Count something two ways. Those two quantities must be the same.

Probabilistic Method (not on exam). If you do something randomly, and you have some chance of encountering something… that something must exist!

# Proofs: Techniques

Direct proof. Just go and prove it!

Contraposition. To prove $p \implies q$, prove that $\overline{q} \implies \overline{p}$.

Contradiction. Suppose that what you're trying to prove is wrong. Prove that the universe implodes.

Casewise. Split into cases. Make sure one of those cases always applies!

Combinatorial Proofs. Count something two ways. Those two quantities must be the same.

Probabilistic Method (not on exam). If you do something randomly, and you have some chance of encountering something... that something must exist!

Induction. Start from base case and expand to entire (countable) set with an inductive step.

# Proofs: Techniques

Direct proof. Just go and prove it!

Contraposition. To prove $p \implies q$, prove that $\overline{q} \implies \overline{p}$.

Contradiction. Suppose that what you're trying to prove is wrong. Prove that the universe implodes.

Casewise. Split into cases. Make sure one of those cases always applies!

Combinatorial Proofs. Count something two ways. Those two quantities must be the same.

Probabilistic Method (not on exam). If you do something randomly, and you have some chance of encountering something... that something must exist!

Induction. Start from base case and expand to entire (countable) set with an inductive step.

**These are all techniques you can compose!**

# Example: A Combinatorial Proof

**Claim**: for $0 \leq k \leq n$, $k\binom{n}{k} = n\binom{n-1}{k-1}$.

# Example: A Combinatorial Proof

**Claim**: for $0 \leq k \leq n$, $k\binom{n}{k} = n\binom{n-1}{k-1}$.

How to prove the claim?

# Example: A Combinatorial Proof

**Claim**: for $0 \leq k \leq n$, $k\binom{n}{k} = n\binom{n-1}{k-1}$.

How to prove the claim? Combinatorial proof.

# Example: A Combinatorial Proof

**Claim**: for $0 \leq k \leq n$, $k\binom{n}{k} = n\binom{n-1}{k-1}$.

How to prove the claim? Combinatorial proof. At the sandwich shop. They have $n$ sandwiches. I want to buy $k$, and eat one right now. How many ways?

# Example: A Combinatorial Proof

**Claim**: for $0 \leq k \leq n$, $k\binom{n}{k} = n\binom{n-1}{k-1}$.

How to prove the claim? Combinatorial proof. At the sandwich shop. They have $n$ sandwiches. I want to buy $k$, and eat one right now. How many ways?

Answer 1: pick $k$ sandwiches to buy: $\binom{n}{k}$ ways. Pick one of them to eat: $k$ ways. Total: $k\binom{n}{k}$ ways.

# Example: A Combinatorial Proof

**Claim**: for $0 \leq k \leq n$, $k\binom{n}{k} = n\binom{n-1}{k-1}$.

How to prove the claim? Combinatorial proof. At the sandwich shop. They have $n$ sandwiches. I want to buy $k$, and eat one right now. How many ways?

Answer 1: pick $k$ sandwiches to buy: $\binom{n}{k}$ ways. Pick one of them to eat: $k$ ways. Total: $k\binom{n}{k}$ ways.

Answer 2: pick sandwich to eat right now first: $n$ ways. Now pick $k-1$ sandwiches out of the remaining $n-1$ at the shop to take home: $\binom{n-1}{k-1}$ ways. Total: $n\binom{n-1}{k-1}$ ways.

# Example: A Combinatorial Proof

**Claim**: for $0 \le k \le n$, $k\binom{n}{k} = n\binom{n-1}{k-1}$.

How to prove the claim? Combinatorial proof. At the sandwich shop. They have $n$ sandwiches. I want to buy $k$, and eat one right now. How many ways?

Answer 1: pick $k$ sandwiches to buy: $\binom{n}{k}$ ways. Pick one of them to eat: $k$ ways. Total: $k\binom{n}{k}$ ways.

Answer 2: pick sandwich to eat right now first: $n$ ways. Now pick $k-1$ sandwiches out of the remaining $n-1$ at the shop to take home: $\binom{n-1}{k-1}$ ways. Total: $n\binom{n-1}{k-1}$ ways.

Two quantities have to be the same. So we have proved the claim.

# Example: FLT

**Fermat's little theorem:** For all $p$ prime, $p|(a^p - a)$.

# Example: FLT

**Fermat's little theorem:** For all $p$ prime, $p|(a^p - a)$.

**Proof:** by induction on $a$.

# Example: FLT

**Fermat's little theorem:** For all $p$ prime, $p|(a^p - a)$.

**Proof:** by induction on $a$. Base case: obviously $p$ must divide $0 = (0^p - 0) = (1^p - 1)$.

# Example: FLT

**Fermat's little theorem:** For all $p$ prime, $p|(a^p - a)$.

**Proof:** by induction on $a$. Base case: obviously $p$ must divide $0 = (0^p - 0) = (1^p - 1)$.

Suppose for induction that $p|(a^p - a)$. It suffices to show that

# Example: FLT

**Fermat's little theorem:** For all $p$ prime, $p|(a^p - a)$.

**Proof:** by induction on $a$. Base case: obviously $p$ must divide $0 = (0^p - 0) = (1^p - 1)$.

Suppose for induction that $p|(a^p - a)$. It suffices to show that $p|((a+1)^p - (a+1))$.

# Example: FLT

**Fermat's little theorem:** For all $p$ prime, $p|(a^p - a)$.

**Proof:** by induction on $a$. Base case: obviously $p$ must divide $0 = (0^p - 0) = (1^p - 1)$.

Suppose for induction that $p|(a^p - a)$. It suffices to show that $p|((a+1)^p - (a+1))$. Expand the first term (binomial theorem): $(a+1)^p$

# Example: FLT

**Fermat's little theorem:** For all $p$ prime, $p|(a^p - a)$.

**Proof:** by induction on $a$. Base case: obviously $p$ must divide $0 = (0^p - 0) = (1^p - 1)$.

Suppose for induction that $p|(a^p - a)$. It suffices to show that $p|((a+1)^p - (a+1))$. Expand the first term (binomial theorem): $(a+1)^p = \sum_{k=0}^{p} \binom{p}{k} a^k = 1 + a^p + \sum_{k=1}^{p-1} \binom{p}{k} a^k$.

# Example: FLT

**Fermat's little theorem:** For all $p$ prime, $p|(a^p - a)$.

**Proof:** by induction on $a$. Base case: obviously $p$ must divide $0 = (0^p - 0) = (1^p - 1)$.

Suppose for induction that $p|(a^p - a)$. It suffices to show that $p|((a+1)^p - (a+1))$. Expand the first term (binomial theorem): $(a+1)^p = \sum_{k=0}^{p} \binom{p}{k} a^k = 1 + a^p + \sum_{k=1}^{p-1} \binom{p}{k} a^k$.

Notice that since $p$ is prime, $p|\binom{p}{k}$ for $k \in (0, p)$. Why?

# Example: FLT

**Fermat's little theorem:** For all $p$ prime, $p|(a^p - a)$.

**Proof:** by induction on $a$. Base case: obviously $p$ must divide $0 = (0^p - 0) = (1^p - 1)$.

Suppose for induction that $p|(a^p - a)$. It suffices to show that $p|((a+1)^p - (a+1))$. Expand the first term (binomial theorem): $(a+1)^p = \sum_{k=0}^{p} \binom{p}{k} a^k = 1 + a^p + \sum_{k=1}^{p-1} \binom{p}{k} a^k$.

Notice that since $p$ is prime, $p|\binom{p}{k}$ for $k \in (0, p)$. Why? From previous page: $k\binom{p}{k} = p\binom{p-1}{k-1}$. We know $p$ and $k$ have no common factors since $p$ is prime. So $p|\binom{p}{k}$.

# Example: FLT

**Fermat's little theorem:** For all $p$ prime, $p|(a^p - a)$.

**Proof:** by induction on $a$. Base case: obviously $p$ must divide $0 = (0^p - 0) = (1^p - 1)$.

Suppose for induction that $p|(a^p - a)$. It suffices to show that $p|((a+1)^p - (a+1))$. Expand the first term (binomial theorem): $(a+1)^p = \sum_{k=0}^{p} \binom{p}{k} a^k = 1 + a^p + \sum_{k=1}^{p-1} \binom{p}{k} a^k$.

Notice that since $p$ is prime, $p|\binom{p}{k}$ for $k \in (0,p)$. Why? From previous page: $k\binom{p}{k} = p\binom{p-1}{k-1}$. We know $p$ and $k$ have no common factors since $p$ is prime. So $p|\binom{p}{k}$.

Also $a^p \equiv a \pmod{p}$ by inductive hypothesis.

So
$$(a+1)^p - (a+1) = 1 + a^p + \sum_{k=1}^{p-1} \binom{p}{k} a^k - (a+1)$$

# Example: FLT

**Fermat's little theorem:** For all $p$ prime, $p|(a^p - a)$.

**Proof:** by induction on $a$. Base case: obviously $p$ must divide $0 = (0^p - 0) = (1^p - 1)$.

Suppose for induction that $p|(a^p - a)$. It suffices to show that $p|((a+1)^p - (a+1))$. Expand the first term (binomial theorem): $(a+1)^p = \sum_{k=0}^{p} \binom{p}{k} a^k = 1 + a^p + \sum_{k=1}^{p-1} \binom{p}{k} a^k$.

Notice that since $p$ is prime, $p|\binom{p}{k}$ for $k \in (0, p)$. Why? From previous page: $k\binom{p}{k} = p\binom{p-1}{k-1}$. We know $p$ and $k$ have no common factors since $p$ is prime. So $p|\binom{p}{k}$.

Also $a^p \equiv a \pmod{p}$ by inductive hypothesis.

So
$$\begin{aligned}(a+1)^p - (a+1) &= 1 + a^p + \sum_{k=1}^{p-1} \binom{p}{k} a^k - (a+1) \\ &\equiv 1 + a + 0 - (a+1) \pmod{p}\end{aligned}$$

# Example: FLT

**Fermat's little theorem:** For all $p$ prime, $p|(a^p - a)$.

**Proof:** by induction on $a$. Base case: obviously $p$ must divide $0 = (0^p - 0) = (1^p - 1)$.

Suppose for induction that $p|(a^p - a)$. It suffices to show that $p|((a+1)^p - (a+1))$. Expand the first term (binomial theorem): $(a+1)^p = \sum_{k=0}^{p} \binom{p}{k} a^k = 1 + a^p + \sum_{k=1}^{p-1} \binom{p}{k} a^k$.

Notice that since $p$ is prime, $p|\binom{p}{k}$ for $k \in (0,p)$. Why? From previous page: $k\binom{p}{k} = p\binom{p-1}{k-1}$. We know $p$ and $k$ have no common factors since $p$ is prime. So $p|\binom{p}{k}$.

Also $a^p \equiv a \pmod{p}$ by inductive hypothesis.

So

$$\begin{aligned} (a+1)^p - (a+1) &= 1 + a^p + \sum_{k=1}^{p-1} \binom{p}{k} a^k - (a+1) \\ &\equiv 1 + a + 0 - (a+1) \pmod{p} \\ &\equiv 0 \pmod{p} \end{aligned}$$

as desired. $\square$

# Graphs

$G = (V, E)$. Collection of **vertices (or nodes)** and **edges = pairs of vertices**.

## Graphs

$G = (V, E)$. Collection of **vertices (or nodes)** and **edges = pairs of vertices**.Unordered or ordered pairs? Depends on whether the graph is directed.

# Graphs

$G = (V, E)$. Collection of **vertices (or nodes)** and **edges = pairs of vertices**.Unordered or ordered pairs? Depends on whether the graph is directed.

**Degree** of a vertex: number of edges touching the vertex.

# Graphs

$G = (V, E)$. Collection of **vertices (or nodes)** and **edges = pairs of vertices**.Unordered or ordered pairs? Depends on whether the graph is directed.

**Degree** of a vertex: number of edges touching the vertex.

Paths in graphs. Can you traverse edges from vertex $v$ to vertex $u$? Then there is a path from $v$ to $u$.

# Graphs

$G = (V, E)$. Collection of **vertices (or nodes)** and **edges = pairs of vertices**.Unordered or ordered pairs? Depends on whether the graph is directed.

**Degree** of a vertex: number of edges touching the vertex.

Paths in graphs. Can you traverse edges from vertex $v$ to vertex $u$? Then there is a path from $v$ to $u$.

Connectivity. An undirected graph is connected if you can reach every vertex from every other vertex (always exists a path between any two vertices). Directed graph is strongly connected if you can reach every vertex from every other vertex by following edges in the correct direction.

# Graphs

$G = (V, E)$. Collection of **vertices (or nodes)** and **edges = pairs of vertices**.Unordered or ordered pairs? Depends on whether the graph is directed.

**Degree** of a vertex: number of edges touching the vertex.

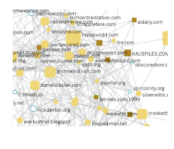
Paths in graphs. Can you traverse edges from vertex $v$ to vertex $u$? Then there is a path from $v$ to $u$.

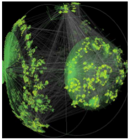
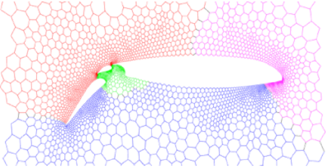
Connectivity. An undirected graph is connected if you can reach every vertex from every other vertex (always exists a path between any two vertices). Directed graph is strongly connected if you can reach every vertex from every other vertex by following edges in the correct direction.

(remember that a Markov chain represented by a strongly connected graph is irreducible).

# Aside: Interesting Applications of Graphs

Web hyperlinks and social networks. Meshes in simulations and scientific computing.[1]



Maps and grids. Games.

Finding paths in graphs is really useful. How does Google maps find a route to your destination? Finding paths in graphs! CS170, CS188.

[1] Images from Aydin Buluc's CS267 slides,
`https://people.eecs.berkeley.edu/~demmel/cs267_Spr16/Lectures/CS267_March17_Buluc_2016_4pp.pdf`

# Walks and Tours

If all degrees are even: Eulerian tour - can walk around the graph so we touch every edge exactly once and return to where we started.

# Walks and Tours

If all degrees are even: **Eulerian tour** - can walk around the graph so we touch every edge exactly once and return to where we started.

If either all degrees or even, or there are exactly two vertices with odd degree: **Eulerian walk** - same as above except we do not necessarily return to the same point.

# Walks and Tours

If all degrees are even: **Eulerian tour** - can walk around the graph so we touch every edge exactly once and return to where we started.

If either all degrees or even, or there are exactly two vertices with odd degree: **Eulerian walk** - same as above except we do not necessarily return to the same point.

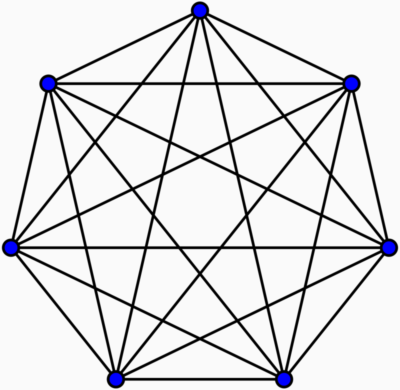
What about if we say that we want to touch every vertex? Interesting question... and hard. CS170.

# Complete graphs.

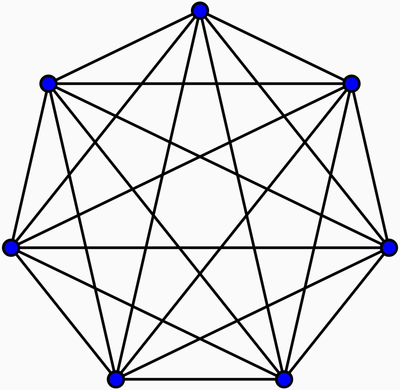
Complete graphs.

Complete graphs.

Complete graphs.

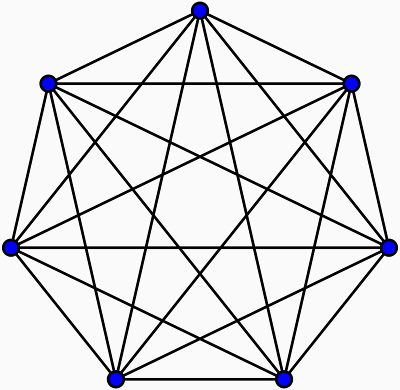



Complete graphs.

$K_n$. $n$ vertices. How many edges? $\binom{n}{2}$.

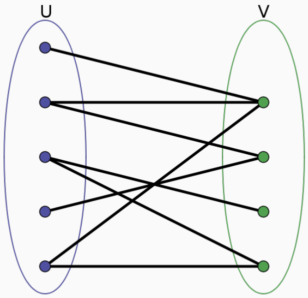
Complete graphs.

$K_n$. $n$ vertices. How many edges? $\binom{n}{2}$.

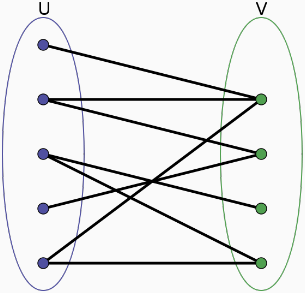
# Bipartite Graphs

Bipartite graphs. Vertices can be partitioned into two sets such that there are no edges between edges in the same set.



# Bipartite Graphs

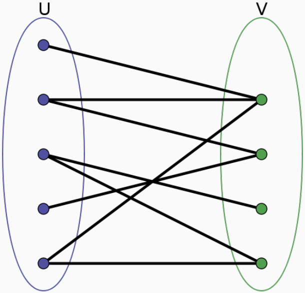
Bipartite graphs. Vertices can be partitioned into two sets such that there are no edges between edges in the same set.




Can represent matchings. Remember stable matchings problem from MT1?

# Bipartite Graphs

Bipartite graphs. Vertices can be partitioned into two sets such that there are no edges between edges in the same set.

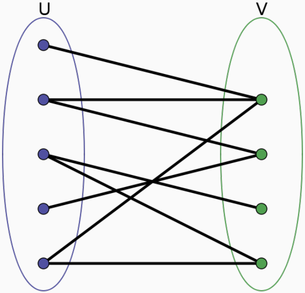



Can represent matchings. Remember stable matchings problem from MT1?

No odd length cycles.

# Bipartite Graphs

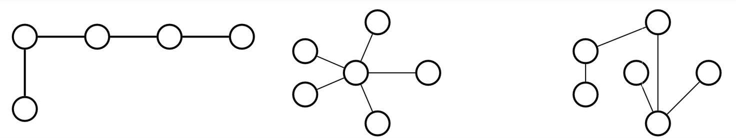
Bipartite graphs. Vertices can be partitioned into two sets such that there are no edges between edges in the same set.




Can represent matchings. Remember stable matchings problem from MT1?

No odd length cycles. Random walk on a bipartite graph is periodic.

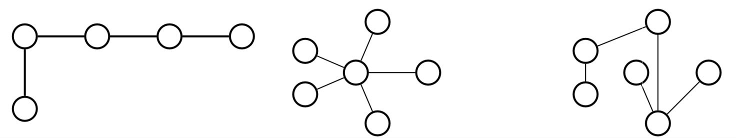
# Trees



How do you define a tree?

- Connected acyclic graph.

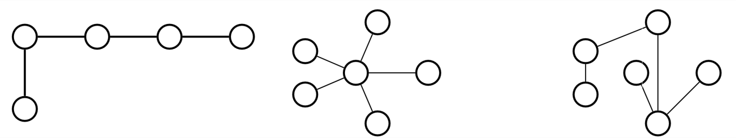
# Trees



How do you define a tree?

- Connected acyclic graph.
- Connected graph with $|V| - 1$ edges.

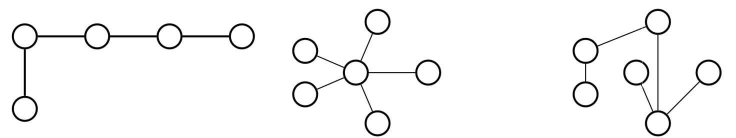
# Trees



How do you define a tree?

- Connected acyclic graph.
- Connected graph with $|V|-1$ edges.
- Connected graph that can be disconnected by removing any edge.

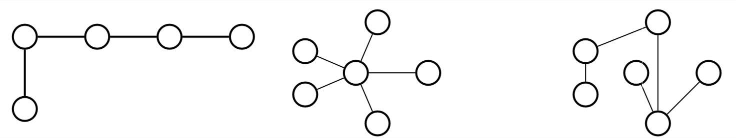
# Trees



How do you define a tree?

- Connected acyclic graph.
- Connected graph with $|V| - 1$ edges.
- Connected graph that can be disconnected by removing any edge.
- Acyclic graph where any edge addition creates a cycle.

# Trees

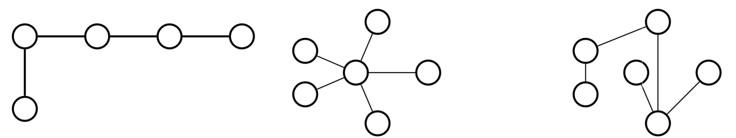


How do you define a tree?

- Connected acyclic graph.
- Connected graph with $|V|-1$ edges.
- Connected graph that can be disconnected by removing any edge.
- Acyclic graph where any edge addition creates a cycle.

Which definition is correct?

# Trees



How do you define a tree?

- Connected acyclic graph.
- Connected graph with $|V| - 1$ edges.
- Connected graph that can be disconnected by removing any edge.
- Acyclic graph where any edge addition creates a cycle.

Which definition is correct? All of them are equivalent. Good practice exercise: prove it!

# Trees are really useful!

Data structures! Binary search trees, heaps, red-black trees, B-trees, etc. Great for storing data.CS61B.

# Trees are really useful!

Data structures! Binary search trees, heaps, red-black trees, B-trees, etc. Great for storing data.CS61B.

Spanning trees. Given a graph, take a subset of edges that makes a tree touching all the vertices. We used this to prove the $4|V||E|$ bound on the cover time of a graph.

# Trees are really useful!

Data structures! Binary search trees, heaps, red-black trees, B-trees, etc. Great for storing data.CS61B.

Spanning trees. Given a graph, take a subset of edges that makes a tree touching all the vertices. We used this to prove the $4|V||E|$ bound on the cover time of a graph. Also really useful in a lot of applications, including fast solvers for systems of linear equations.

# Trees are really useful!

Data structures! Binary search trees, heaps, red-black trees, B-trees, etc. Great for storing data.CS61B.

Spanning trees. Given a graph, take a subset of edges that makes a tree touching all the vertices. We used this to prove the $4|V||E|$ bound on the cover time of a graph. Also really useful in a lot of applications, including fast solvers for systems of linear equations.

Spawn trees. Represent function calls, etc.

# Trees are really useful!

Data structures! Binary search trees, heaps, red-black trees, B-trees, etc. Great for storing data.CS61B.

Spanning trees. Given a graph, take a subset of edges that makes a tree touching all the vertices. We used this to prove the $4|V||E|$ bound on the cover time of a graph. Also really useful in a lot of applications, including fast solvers for systems of linear equations.

Spawn trees. Represent function calls, etc.

Decision trees. Every vertex represents a decision you can make. CS188

Can you draw the graph on paper so that no edges cross? If so, it's a planar graph.

Can you draw the graph on paper so that no edges cross? If so, it's a planar graph.

Euler's formula: $v+f=e+2$. (how did we prove this? Induction on $e$. Remove cycle by removing edge.)

# Planar Graphs

Can you draw the graph on paper so that no edges cross? If so, it's a planar graph.

Euler's formula: $v+f=e+2$. (how did we prove this? Induction on $e$. Remove cycle by removing edge.)

Four color theorem: any planar graph can be colored with four colors so that no edge is monochromatic (same color on both endpoints). You can color a map with four colors. Proof? 400 pages long. Too long for this course... or the exam.

We proved a six color theorem for by induction on $v$.

# Planar Graphs

Can you draw the graph on paper so that no edges cross? If so, it's a planar graph.

Euler's formula: $v + f = e + 2$. (how did we prove this? Induction on $e$. Remove cycle by removing edge.)

Four color theorem: any planar graph can be colored with four colors so that no edge is monochromatic (same color on both endpoints). You can color a map with four colors. Proof? 400 pages long. Too long for this course... or the exam.

We proved a six color theorem for by induction on $v$.

**Practice problems:** try doing these proofs yourself (without looking at the old slides).

# On coloring...

Can you color a graph with *three* colors? Hard problem. Really hard! See why in CS170.

# On coloring...

Can you color a graph with *three* colors? Hard problem. Really hard! See why in CS170.

People do try to approximate good colorings (for general graphs, not just planar) though. It's useful.

# On coloring...

Can you color a graph with *three* colors? Hard problem. Really hard! See why in CS170.

People do try to approximate good colorings (for general graphs, not just planar) though. It's useful.

For instance: if a program I'm trying to compile that looks like this:

```
a <- b + c;     c <- a + b;     z <- b - a;
```

How many registers (memory) do I need to run this program?

# On coloring...

Can you color a graph with *three* colors? Hard problem. Really hard! See why in CS170.

People do try to approximate good colorings (for general graphs, not just planar) though. It's useful.

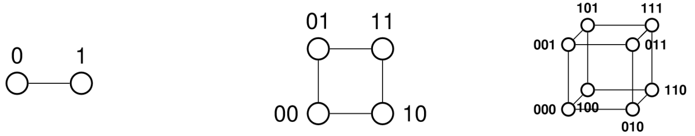
For instance: if a program I'm trying to compile that looks like this:

```
a <- b + c;      c <- a + b;      z <- b - a;
```

How many registers (memory) do I need to run this program? Draw a graph and try to approximate the optimal coloring! Each color is a register.

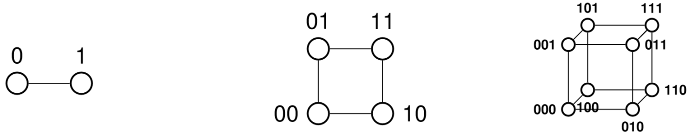
Register optimization! Touched on in CS164.

# Hypercubes




Hypercubes. $G = \{V, E\}$. $V = \{0,1\}^n$.
$E = \{(u,v) | u$ and $v$ differ by exactly one bit position$\}$.

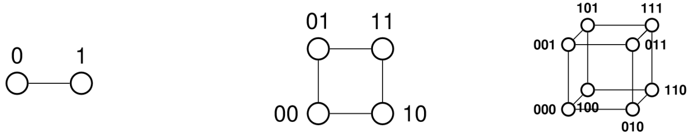
# Hypercubes




Hypercubes. $G = \{V, E\}$. $V = \{0,1\}^n$.
$E = \{(u,v) | u$ and $v$ differ by exactly one bit position$\}$.

Number of vertices? $2^n$.

# Hypercubes

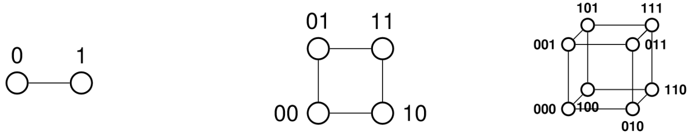



Hypercubes. $G = \{V, E\}$. $V = \{0,1\}^n$.
$E = \{(u,v) | u$ and $v$ differ by exactly one bit position$\}$.

Number of vertices? $2^n$. Number of edges? $n2^{n-1} = O(|V| \log |V|)$.

Recursively: Start with a point. Dimension 0 hypercube.

# Hypercubes

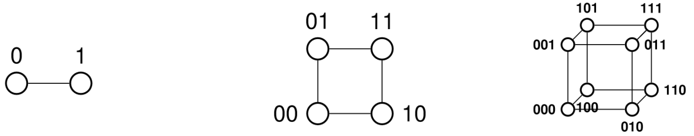



Hypercubes. $G = \{V, E\}$. $V = \{0,1\}^n$.
$E = \{(u,v) | u$ and $v$ differ by exactly one bit position$\}$.

Number of vertices? $2^n$. Number of edges? $n2^{n-1} = O(|V| \log |V|)$.

Recursively: Start with a point. Dimension 0 hypercube.Get a dimension 1 hypercube by taking two points and connecting them.

# Hypercubes

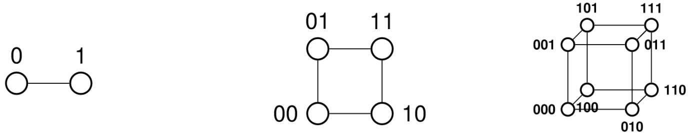



Hypercubes. $G = \{V, E\}$. $V = \{0,1\}^n$.
$E = \{(u,v) | u$ and $v$ differ by exactly one bit position$\}$.

Number of vertices? $2^n$. Number of edges? $n2^{n-1} = O(|V| \log |V|)$.

Recursively: Start with a point. Dimension 0 hypercube.Get a dimension 1 hypercube by taking two points and connecting them. Get dimension 2 hypercube by taking two dimension 1 hypercubes and connecting corresponding vertices.

# Hypercubes

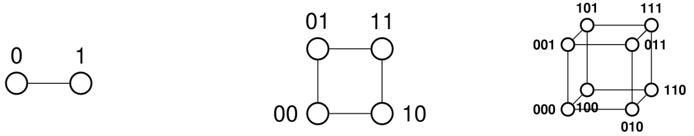



Hypercubes. $G = \{V, E\}$. $V = \{0,1\}^n$.
$E = \{(u,v) | u$ and $v$ differ by exactly one bit position$\}$.

Number of vertices? $2^n$. Number of edges? $n2^{n-1} = O(|V| \log |V|)$.

Recursively: Start with a point. Dimension 0 hypercube.Get a dimension 1 hypercube by taking two points and connecting them. Get dimension 2 hypercube by taking two dimension 1 hypercubes and connecting corresponding vertices. Get dimension $n+1$ hypercube by taking two dimension $n$ hypercubes and connecting corresponding edges.

# Hypercubes




Hypercubes. $G = \{V, E\}$. $V = \{0,1\}^n$.
$E = \{(u,v) | u$ and $v$ differ by exactly one bit position$\}$.

Number of vertices? $2^n$. Number of edges? $n2^{n-1} = O(|V| \log |V|)$.

Recursively: Start with a point. Dimension 0 hypercube.Get a dimension 1 hypercube by taking two points and connecting them. Get dimension 2 hypercube by taking two dimension 1 hypercubes and connecting corresponding vertices. Get dimension $n+1$ hypercube by taking two dimension $n$ hypercubes and connecting corresponding edges.

# Hypercubes: Properties and Applications

Shortest path from $u$ to $v$? How many bits do they differ in? "hamming distance". Each edge traversal is a bit flip.

[2] `https://blog.ethereum.org/2014/10/21/scalability-part-2-hypercubes/`

# Hypercubes: Properties and Applications

Shortest path from $u$ to $v$? How many bits do they differ in? "hamming distance". Each edge traversal is a bit flip.

Dense cuts. If you want to cut off $k$ vertices, you need to cut $k$ edges.

[2] `https://blog.ethereum.org/2014/10/21/scalability-part-2-hypercubes/`

# Hypercubes: Properties and Applications

Shortest path from $u$ to $v$? How many bits do they differ in? "hamming distance". Each edge traversal is a bit flip.

Dense cuts. If you want to cut off $k$ vertices, you need to cut $k$ edges.Proof was by induction on the dimension. Also a good practice problem to go over this proof again.

[2] `https://blog.ethereum.org/2014/10/21/scalability-part-2-hypercubes/`

# Hypercubes: Properties and Applications

Shortest path from $u$ to $v$? How many bits do they differ in? "hamming distance". Each edge traversal is a bit flip.

Dense cuts. If you want to cut off $k$ vertices, you need to cut $k$ edges.Proof was by induction on the dimension. Also a good practice problem to go over this proof again.

Ease of routing and difficulty to cut make hypercubes really useful for distributed systems. Hypercube topology to be very common in supercomputers: Intel iPSC, nCube.

Now being used for routing messages in the Ethereum network.[2]

[2] https://blog.ethereum.org/2014/10/21/scalability-part-2-hypercubes/

# Randomness and Graphs

Random graphs. $G_{n,p}$. $n$ vertices. Each edge exists with probability $p$.

Random graphs. $G_{n,p}$. $n$ vertices. Each edge exists with probability $p$. Expected number of edges? $\binom{n}{2}p$. We used this for probabilistic method proofs.

# Randomness and Graphs

Random graphs. $G_{n,p}$. $n$ vertices. Each edge exists with probability $p$. Expected number of edges? $\binom{n}{2}p$. We used this for probabilistic method proofs.

Random walks on undirected graphs. Take a graph and turn it into a Markov chain. Vertices are states. Transition to any neighboring state with uniform probability.

# Randomness and Graphs

Random graphs. $G_{n,p}$. $n$ vertices. Each edge exists with probability $p$. Expected number of edges? $\binom{n}{2}p$. We used this for probabilistic method proofs.

Random walks on undirected graphs. Take a graph and turn it into a Markov chain. Vertices are states. Transition to any neighboring state with uniform probability. Irreducible if graph is connected. Aperiodic if graph isn't bipartite.

# Randomness and Graphs

Random graphs. $G_{n,p}$. $n$ vertices. Each edge exists with probability $p$. Expected number of edges? $\binom{n}{2}p$. We used this for probabilistic method proofs.

Random walks on undirected graphs. Take a graph and turn it into a Markov chain. Vertices are states. Transition to any neighboring state with uniform probability. Irreducible if graph is connected. Aperiodic if graph isn't bipartite.

Stationary distribution? $\deg(v)/2|E|$.

Cover time? bounded by $4|V||E|$.

# Randomness and Graphs

Random graphs. $G_{n,p}$. $n$ vertices. Each edge exists with probability $p$. Expected number of edges? $\binom{n}{2}p$. We used this for probabilistic method proofs.

Random walks on undirected graphs. Take a graph and turn it into a Markov chain. Vertices are states. Transition to any neighboring state with uniform probability. Irreducible if graph is connected. Aperiodic if graph isn't bipartite.

Stationary distribution? $\deg(v)/2|E|$.

Cover time? bounded by $4|V||E|$.

More review on this tomorrow.

# Stable Marriage and Matchings

Want to match men and women in a stable manner. No rogue couples (people who would both rather be with each other than with their current partners).

# Stable Marriage and Matchings

Want to match men and women in a stable manner. No rogue couples (people who would both rather be with each other than with their current partners).

Each day: Each man proposes to the highest woman on his list who hasn't rejected him yet. Each woman rejects all but the best, whom she "keeps on a string".

# Stable Marriage and Matchings

Want to match men and women in a stable manner. No rogue couples (people who would both rather be with each other than with their current partners).

Each day: Each man proposes to the highest woman on his list who hasn't rejected him yet. Each woman rejects all but the best, whom she "keeps on a string".

**Improvement lemma:** If a woman has a man on a string: any future man who proposes to her is going to be better.

# Stable Marriage and Matchings

Want to match men and women in a stable manner. No rogue couples (people who would both rather be with each other than with their current partners).

Each day: Each man proposes to the highest woman on his list who hasn't rejected him yet. Each woman rejects all but the best, whom she "keeps on a string".

**Improvement lemma:** If a woman has a man on a string: any future man who proposes to her is going to be better.

TMA produces *male optimal* stable matching! Male optimal = best pairing for men among all possible matchings.

**Theorem:** male optimal = female pessimal.

# Stable Marriage and Matchings

Want to match men and women in a stable manner. No rogue couples (people who would both rather be with each other than with their current partners).

Each day: Each man proposes to the highest woman on his list who hasn't rejected him yet. Each woman rejects all but the best, whom she "keeps on a string".

**Improvement lemma:** If a woman has a man on a string: any future man who proposes to her is going to be better.

TMA produces *male optimal* stable matching! Male optimal = best pairing for men among all possible matchings.

**Theorem:** male optimal = female pessimal.

What about stable roommates? Nope. No stable pairing necessarily exists.

# Stable Marriage and Matchings

Want to match men and women in a stable manner. No rogue couples (people who would both rather be with each other than with their current partners).

Each day: Each man proposes to the highest woman on his list who hasn't rejected him yet. Each woman rejects all but the best, whom she "keeps on a string".

**Improvement lemma:** If a woman has a man on a string: any future man who proposes to her is going to be better.

TMA produces *male optimal* stable matching! Male optimal = best pairing for men among all possible matchings.

**Theorem:** male optimal = female pessimal.

What about stable roommates? Nope. No stable pairing necessarily exists.

Used in hospital residency matching systems. Matching in general is a well studied problem. Used in programs like kidney exchanges.

# Counting

| | With replacement | Without replacement |
|---|---|---|
| Order matters | $n^k$ | $\frac{n!}{(n-k)!}$ |
| Order doesn't matter | $\binom{n+k-1}{n-1}$ | $\binom{n}{k}$ |

Selecting $k$ TAs out of $n$ to run a review session?

# Counting

| | With replacement | Without replacement |
|---|---|---|
| Order matters | $n^k$ | $\frac{n!}{(n-k)!}$ |
| Order doesn't matter | $\binom{n+k-1}{n-1}$ | $\binom{n}{k}$ |

Selecting $k$ TAs out of $n$ to run a review session? Order doesn't matter, without replacement.

# Counting

| | With replacement | Without replacement |
|---|---|---|
| Order matters | $n^k$ | $\frac{n!}{(n-k)!}$ |
| Order doesn't matter | $\binom{n+k-1}{n-1}$ | $\binom{n}{k}$ |

Selecting $k$ TAs out of $n$ to run a review session? Order doesn't matter, without replacement.

Polling $k$ random people on the street in a city of population $n$?

# Counting

| | With replacement | Without replacement |
|---|---|---|
| Order matters | $n^k$ | $\frac{n!}{(n-k)!}$ |
| Order doesn't matter | $\binom{n+k-1}{n-1}$ | $\binom{n}{k}$ |

Selecting $k$ TAs out of $n$ to run a review session? Order doesn't matter, without replacement.

Polling $k$ random people on the street in a city of population $n$? Order doesn't matter, with replacement.

# Counting

| | With replacement | Without replacement |
|---|---|---|
| Order matters | $n^k$ | $\frac{n!}{(n-k)!}$ |
| Order doesn't matter | $\binom{n+k-1}{n-1}$ | $\binom{n}{k}$ |

Selecting $k$ TAs out of $n$ to run a review session? Order doesn't matter, without replacement.

Polling $k$ random people on the street in a city of population $n$? Order doesn't matter, with replacement.

Making a homework by selecting $k$ problems from a set of $n$?

# Counting

| | With replacement | Without replacement |
|---|---|---|
| Order matters | $n^k$ | $\frac{n!}{(n-k)!}$ |
| Order doesn't matter | $\binom{n+k-1}{n-1}$ | $\binom{n}{k}$ |

Selecting $k$ TAs out of $n$ to run a review session? Order doesn't matter, without replacement.

Polling $k$ random people on the street in a city of population $n$? Order doesn't matter, with replacement.

Making a homework by selecting $k$ problems from a set of $n$? Order matters, without replacement.

# Counting

| | With replacement | Without replacement |
|---|---|---|
| Order matters | $n^k$ | $\frac{n!}{(n-k)!}$ |
| Order doesn't matter | $\binom{n+k-1}{n-1}$ | $\binom{n}{k}$ |

Selecting $k$ TAs out of $n$ to run a review session? Order doesn't matter, without replacement.

Polling $k$ random people on the street in a city of population $n$? Order doesn't matter, with replacement.

Making a homework by selecting $k$ problems from a set of $n$? Order matters, without replacement.

Answering $n$ multiple-choice questions, each with $k$ options?

# Counting

| | With replacement | Without replacement |
|---|---|---|
| Order matters | $n^k$ | $\frac{n!}{(n-k)!}$ |
| Order doesn't matter | $\binom{n+k-1}{n-1}$ | $\binom{n}{k}$ |

Selecting $k$ TAs out of $n$ to run a review session? Order doesn't matter, without replacement.

Polling $k$ random people on the street in a city of population $n$? Order doesn't matter, with replacement.

Making a homework by selecting $k$ problems from a set of $n$? Order matters, without replacement.

Answering $n$ multiple-choice questions, each with $k$ options? Order matters, with replacement.

# Counting

| | With replacement | Without replacement |
|---|---|---|
| Order matters | $n^k$ | $\frac{n!}{(n-k)!}$ |
| Order doesn't matter | $\binom{n+k-1}{n-1}$ | $\binom{n}{k}$ |

Selecting $k$ TAs out of $n$ to run a review session? Order doesn't matter, without replacement.

Polling $k$ random people on the street in a city of population $n$? Order doesn't matter, with replacement.

Making a homework by selecting $k$ problems from a set of $n$? Order matters, without replacement.

Answering $n$ multiple-choice questions, each with $k$ options? Order matters, with replacement.

If you like counting, take Math 172.

# Inclusion-Exclusion

Number of things in $A_1 \cup A_2 \cup \ldots \cup A_n$?

# Inclusion-Exclusion

Number of things in $A_1 \cup A_2 \cup \ldots \cup A_n$?

Add the cardinalities of all the sets...

# Inclusion-Exclusion

Number of things in $A_1 \cup A_2 \cup \ldots \cup A_n$?

Add the cardinalities of all the sets…

… and then subtract size of the unions of every pair of sets…

# Inclusion-Exclusion

Number of things in $A_1 \cup A_2 \cup \ldots \cup A_n$?

Add the cardinalities of all the sets...

... and then subtract size of the unions of every pair of sets...

... and then add the size of the unions of every triplet of sets...

# Inclusion-Exclusion

Number of things in $A_1 \cup A_2 \cup \ldots \cup A_n$?

Add the cardinalities of all the sets...

... and then subtract size of the unions of every pair of sets...

... and then add the size of the unions of every triplet of sets...

And so on!

# Uncountability

What's countable and what's not?

# Uncountability

What's countable and what's not?

Countability: mappable to the natural numbers. Not everything is countable.

# Uncountability

What's countable and what's not?

Countability: mappable to the natural numbers. Not everything is countable.

Proof of uncountability: bijection with an uncountable set. Diagonalization.

# Uncountability

What's countable and what's not?

Countability: mappable to the natural numbers. Not everything is countable.

Proof of uncountability: bijection with an uncountable set. Diagonalization.

Proof of countability: bijection with a known countable set. Enumeration.

# Uncountability

What's countable and what's not?

Countability: mappable to the natural numbers. Not everything is countable.

Proof of uncountability: bijection with an uncountable set. Diagonalization.

Proof of countability: bijection with a known countable set. Enumeration.

- Finite bitstrings.

# Uncountability

What's countable and what's not?

Countability: mappable to the natural numbers. Not everything is countable.

Proof of uncountability: bijection with an uncountable set. Diagonalization.

Proof of countability: bijection with a known countable set. Enumeration.

- Finite bitstrings. countable

# Uncountability

What’s countable and what’s not?

Countability: mappable to the natural numbers. Not everything is countable.

Proof of uncountability: bijection with an uncountable set. Diagonalization.

Proof of countability: bijection with a known countable set. Enumeration.

- Finite bitstrings. countable
- Programs that print “CS70”.

# Uncountability

What's countable and what's not?

Countability: mappable to the natural numbers. Not everything is countable.

Proof of uncountability: bijection with an uncountable set. Diagonalization.

Proof of countability: bijection with a known countable set. Enumeration.

- Finite bitstrings. countable
- Programs that print "CS70". countable

# Uncountability

What’s countable and what’s not?

Countability: mappable to the natural numbers. Not everything is countable.

Proof of uncountability: bijection with an uncountable set. Diagonalization.

Proof of countability: bijection with a known countable set. Enumeration.

- Finite bitstrings. countable
- Programs that print “CS70”. countable
- Programs that loop infinitely.

# Uncountability

What's countable and what's not?

Countability: mappable to the natural numbers. Not everything is countable.

Proof of uncountability: bijection with an uncountable set. Diagonalization.

Proof of countability: bijection with a known countable set. Enumeration.

- Finite bitstrings. countable
- Programs that print "CS70". countable
- Programs that loop infinitely. countable

# Uncountability

What's countable and what's not?

Countability: mappable to the natural numbers. Not everything is countable.

Proof of uncountability: bijection with an uncountable set. Diagonalization.

Proof of countability: bijection with a known countable set. Enumeration.

- Finite bitstrings. countable
- Programs that print "CS70". countable
- Programs that loop infinitely. countable
- Countable sets of countable sets.

# Uncountability

What's countable and what's not?

Countability: mappable to the natural numbers. Not everything is countable.

Proof of uncountability: bijection with an uncountable set. Diagonalization.

Proof of countability: bijection with a known countable set. Enumeration.

- Finite bitstrings. countable
- Programs that print "CS70". countable
- Programs that loop infinitely. countable
- Countable sets of countable sets. countable

# Uncountability

What's countable and what's not?

Countability: mappable to the natural numbers. Not everything is countable.

Proof of uncountability: bijection with an uncountable set. Diagonalization.

Proof of countability: bijection with a known countable set. Enumeration.

- Finite bitstrings. countable
- Programs that print "CS70". countable
- Programs that loop infinitely. countable
- Countable sets of countable sets. countable
- Pairs of real numbers.

# Uncountability

What's countable and what's not?

Countability: mappable to the natural numbers. Not everything is countable.

Proof of uncountability: bijection with an uncountable set. Diagonalization.

Proof of countability: bijection with a known countable set. Enumeration.

- Finite bitstrings. countable
- Programs that print "CS70". countable
- Programs that loop infinitely. countable
- Countable sets of countable sets. countable
- Pairs of real numbers. uncountable

# Uncountability

What's countable and what's not?

Countability: mappable to the natural numbers. Not everything is countable.

Proof of uncountability: bijection with an uncountable set. Diagonalization.

Proof of countability: bijection with a known countable set. Enumeration.

- Finite bitstrings. countable
- Programs that print "CS70". countable
- Programs that loop infinitely. countable
- Countable sets of countable sets. countable
- Pairs of real numbers. uncountable
- Graphs.

# Uncountability

What's countable and what's not?

Countability: mappable to the natural numbers. Not everything is countable.

Proof of uncountability: bijection with an uncountable set. Diagonalization.

Proof of countability: bijection with a known countable set. Enumeration.

- Finite bitstrings. countable
- Programs that print "CS70". countable
- Programs that loop infinitely. countable
- Countable sets of countable sets. countable
- Pairs of real numbers. uncountable
- Graphs. countable

# Uncountability

What's countable and what's not?

Countability: mappable to the natural numbers. Not everything is countable.

Proof of uncountability: bijection with an uncountable set. Diagonalization.

Proof of countability: bijection with a known countable set. Enumeration.

- Finite bitstrings. countable
- Programs that print "CS70". countable
- Programs that loop infinitely. countable
- Countable sets of countable sets. countable
- Pairs of real numbers. uncountable
- Graphs. countable
- Markov chains.

# Uncountability

What’s countable and what’s not?

Countability: mappable to the natural numbers. Not everything is countable.

Proof of uncountability: bijection with an uncountable set. Diagonalization.

Proof of countability: bijection with a known countable set. Enumeration.

- Finite bitstrings. countable
- Programs that print “CS70”. countable
- Programs that loop infinitely. countable
- Countable sets of countable sets. countable
- Pairs of real numbers. uncountable
- Graphs. countable
- Markov chains. uncountable

# Uncountability

What's countable and what's not?

Countability: mappable to the natural numbers. Not everything is countable.

Proof of uncountability: bijection with an uncountable set. Diagonalization.

Proof of countability: bijection with a known countable set. Enumeration.

- Finite bitstrings. countable
- Programs that print "CS70". countable
- Programs that loop infinitely. countable
- Countable sets of countable sets. countable
- Pairs of real numbers. uncountable
- Graphs. countable
- Markov chains. uncountable
- Distributions.

# Uncountability

What's countable and what's not?

Countability: mappable to the natural numbers. Not everything is countable.

Proof of uncountability: bijection with an uncountable set. Diagonalization.

Proof of countability: bijection with a known countable set. Enumeration.

- Finite bitstrings. countable
- Programs that print "CS70". countable
- Programs that loop infinitely. countable
- Countable sets of countable sets. countable
- Pairs of real numbers. uncountable
- Graphs. countable
- Markov chains. uncountable
- Distributions. uncountable

# Uncountability

What's countable and what's not?

Countability: mappable to the natural numbers. Not everything is countable.

Proof of uncountability: bijection with an uncountable set. Diagonalization.

Proof of countability: bijection with a known countable set. Enumeration.

- Finite bitstrings. countable
- Programs that print "CS70". countable
- Programs that loop infinitely. countable
- Countable sets of countable sets. countable
- Pairs of real numbers. uncountable
- Graphs. countable
- Markov chains. uncountable
- Distributions. uncountable

# Undecidability

Can we tell if a program halts? **No!** Undecidable problem.

Why not? self-reference!

# Undecidability

Can we tell if a program halts? **No!** Undecidable problem.

Why not? self-reference!

Suppose for contradiction that we have some oracle that can tell us if a program halts. Then we can make some program called "breakUniverse":

# Undecidability

Can we tell if a program halts? **No!** Undecidable problem.

Why not? self-reference!

Suppose for contradiction that we have some oracle that can tell us if a program halts. Then we can make some program called "breakUniverse":

Input: some program $P$.

Feed the text representation of $P$ into $P$. Ask our oracle if it halts. If it does: infinite loop. Otherwise: stop.

# Undecidability

Can we tell if a program halts? **No!** Undecidable problem.

Why not? self-reference!

Suppose for contradiction that we have some oracle that can tell us if a program halts. Then we can make some program called "breakUniverse":

Input: some program $P$.

Feed the text representation of $P$ into $P$. Ask our oracle if it halts. If it does: infinite loop. Otherwise: stop.

Feed "breakUniverse" into "breakUniverse".

# Undecidability

Can we tell if a program halts? **No!** Undecidable problem.

Why not? self-reference!

Suppose for contradiction that we have some oracle that can tell us if a program halts. Then we can make some program called "breakUniverse":

Input: some program $P$.

Feed the text representation of $P$ into $P$. Ask our oracle if it halts. If it does: infinite loop. Otherwise: stop.

Feed "breakUniverse" into "breakUniverse".

Halts if and only if it doesn't halt. AAAAAAAAAAHHHHHH!!!!!!11!!!1111!!
Contradiction.

# Undecidability

Can we tell if a program halts? **No!** Undecidable problem.

Why not? self-reference!

Suppose for contradiction that we have some oracle that can tell us if a program halts. Then we can make some program called "breakUniverse":

Input: some program $P$.

Feed the text representation of $P$ into $P$. Ask our oracle if it halts. If it does: infinite loop. Otherwise: stop.

Feed "breakUniverse" into "breakUniverse".

Halts if and only if it doesn't halt. AAAAAAAAAHHHHHH!!!!!!11!!!1111!! Contradiction.

So halting is undecidable!

Questions?