Discrete Mathematics Review

- logic, proofs
- induction
- graph theory
- modular arithmetic, RSA
- polynomials, error correction
- countability, computability
- counting

Set Notation

Basic notation: $\in$, $\subseteq$, $\cup$, $\cap$.

Set Notation

Basic notation: $\in$, $\subseteq$, $\cup$, $\cap$.

- To prove set equality $A = B$, show $A \subseteq B$ and $B \subseteq A$.

Set Notation

Basic notation: $\in$, $\subseteq$, $\cup$, $\cap$.

- To prove set equality $A = B$, show $A \subseteq B$ and $B \subseteq A$.
- To prove $A \subseteq B$, show that for each $a \in A$, then $a \in B$ also.

Set Notation

Basic notation: $\in$, $\subseteq$, $\cup$, $\cap$.

- To prove set equality $A = B$, show $A \subseteq B$ and $B \subseteq A$.
- To prove $A \subseteq B$, show that for each $a \in A$, then $a \in B$ also.
- $\{0, 1\}$ is the set containing the two elements 0 and 1.

Set Notation

Basic notation: $\in$, $\subseteq$, $\cup$, $\cap$.

- To prove set equality $A = B$, show $A \subseteq B$ and $B \subseteq A$.
- To prove $A \subseteq B$, show that for each $a \in A$, then $a \in B$ also.
- $\{0, 1\}$ is the set containing the two elements 0 and 1.
- $[0, 1]$ is the closed interval containing all x with $0 \leq x \leq 1$.

Set Notation

Basic notation: $\in$, $\subseteq$, $\cup$, $\cap$.

- To prove set equality $A = B$, show $A \subseteq B$ and $B \subseteq A$.
- To prove $A \subseteq B$, show that for each $a \in A$, then $a \in B$ also.
- $\{0, 1\}$ is the set containing the two elements 0 and 1.
- $[0, 1]$ is the closed interval containing all x with $0 \leq x \leq 1$.
- $(0, 1)$ is the open interval containing all x with $0 < x < 1$, *or* it is the ordered tuple containing 0 and 1 (context).

Set Notation

Basic notation: $\in$, $\subseteq$, $\cup$, $\cap$.

- To prove set equality $A = B$, show $A \subseteq B$ and $B \subseteq A$.
- To prove $A \subseteq B$, show that for each $a \in A$, then $a \in B$ also.
- $\{0, 1\}$ is the set containing the two elements 0 and 1.
- $[0, 1]$ is the closed interval containing all x with $0 \leq x \leq 1$.
- $(0, 1)$ is the open interval containing all x with $0 < x < 1$, *or* it is the ordered tuple containing 0 and 1 (context).
- Cartesian product: $A \times B$ is the set of all pairs (a, b) where $a \in A$ and $b \in B$.

Set Notation

Basic notation: $\in$, $\subseteq$, $\cup$, $\cap$.

- To prove set equality $A = B$, show $A \subseteq B$ and $B \subseteq A$.
- To prove $A \subseteq B$, show that for each $a \in A$, then $a \in B$ also.
- $\{0, 1\}$ is the set containing the two elements 0 and 1.
- $[0, 1]$ is the closed interval containing all x with $0 \leq x \leq 1$.
- $(0, 1)$ is the open interval containing all x with $0 < x < 1$, *or* it is the ordered tuple containing 0 and 1 (context).
- Cartesian product: $A \times B$ is the set of all pairs (a, b) where $a \in A$ and $b \in B$.
  $\{0, 1\} \times \{A, B\} = \{(0, A), (0, B), (1, A), (1, B)\}$.

Set Notation

Basic notation: $\in$, $\subseteq$, $\cup$, $\cap$.

- To prove set equality $A = B$, show $A \subseteq B$ and $B \subseteq A$.
- To prove $A \subseteq B$, show that for each $a \in A$, then $a \in B$ also.
- $\{0, 1\}$ is the set containing the two elements 0 and 1.
- $[0, 1]$ is the closed interval containing all x with $0 \leq x \leq 1$.
- $(0, 1)$ is the open interval containing all x with $0 < x < 1$, *or* it is the ordered tuple containing 0 and 1 (context).
- Cartesian product: $A \times B$ is the set of all pairs (a, b) where $a \in A$ and $b \in B$.
  $\{0, 1\} \times \{A, B\} = \{(0, A), (0, B), (1, A), (1, B)\}$.
- We define sets like so: $\{x \in S : \text{conditions on } x\}$.

Set Notation

Basic notation: $\in$, $\subseteq$, $\cup$, $\cap$.

- To prove set equality $A = B$, show $A \subseteq B$ and $B \subseteq A$.
- To prove $A \subseteq B$, show that for each $a \in A$, then $a \in B$ also.
- $\{0,1\}$ is the set containing the two elements 0 and 1.
- $[0,1]$ is the closed interval containing all x with $0 \leq x \leq 1$.
- $(0,1)$ is the open interval containing all x with $0 < x < 1$, *or* it is the ordered tuple containing 0 and 1 (context).
- Cartesian product: $A \times B$ is the set of all pairs (a,b) where $a \in A$ and $b \in B$.
  $\{0,1\} \times \{A,B\} = \{(0,A),(0,B),(1,A),(1,B)\}$.
- We define sets like so: $\{x \in S : \text{conditions on } x\}$. This is the set of all elements in S satisfying the stated conditions.

Set Notation

Basic notation: $\in$, $\subseteq$, $\cup$, $\cap$.

- To prove set equality $A = B$, show $A \subseteq B$ and $B \subseteq A$.
- To prove $A \subseteq B$, show that for each $a \in A$, then $a \in B$ also.
- $\{0, 1\}$ is the set containing the two elements 0 and 1.
- $[0, 1]$ is the closed interval containing all x with $0 \leq x \leq 1$.
- $(0, 1)$ is the open interval containing all x with $0 < x < 1$, *or* it is the ordered tuple containing 0 and 1 (context).
- Cartesian product: $A \times B$ is the set of all pairs (a, b) where $a \in A$ and $b \in B$.
  $\{0, 1\} \times \{A, B\} = \{(0, A), (0, B), (1, A), (1, B)\}$.
- We define sets like so: $\{x \in S : \text{conditions on } x\}$. This is the set of all elements in S satisfying the stated conditions.

$$\{x \in \mathbb{N} : 2 \leq x \leq 7\} = \{2, 3, 4, 5, 6, 7\}.$$

Propositional Logic

Language of propositional logic: given propositions P, Q,

Propositional Logic

Language of propositional logic: given propositions P, Q,

- negate a proposition: $\neg P$;

Propositional Logic

Language of propositional logic: given propositions P, Q,

- negate a proposition: $\neg P$;
- combine propositions: $P \vee Q$, $P \wedge Q$, $P \implies Q$, $P \iff Q$.

Propositional Logic

Language of propositional logic: given propositions P, Q,

- negate a proposition: $\neg P$;
- combine propositions: $P \vee Q$, $P \wedge Q$, $P \implies Q$, $P \iff Q$.

To answer questions in propositional logic, use *truth tables*.

Propositional Logic

Language of propositional logic: given propositions P, Q,

- negate a proposition: $\neg P$;
- combine propositions: $P \vee Q$, $P \wedge Q$, $P \implies Q$, $P \iff Q$.

To answer questions in propositional logic, use *truth tables*. Or, use logical equivalences (e.g., De Morgan).

Propositional Logic

Language of propositional logic: given propositions P, Q,

- negate a proposition: $\neg P$;
- combine propositions: $P \vee Q$, $P \wedge Q$, $P \implies Q$, $P \iff Q$.

To answer questions in propositional logic, use *truth tables*. Or, use logical equivalences (e.g., De Morgan).

Midterm question: given a truth table

P	Q	$P \oplus Q$
T	T	F
T	F	T
F	T	T
F	F	F

can you write an equivalent sentence using P, Q, $\neg$, $\wedge$, $\vee$?

Propositional Logic

Language of propositional logic: given propositions P, Q,

- negate a proposition: $\neg P$;
- combine propositions: $P \vee Q$, $P \wedge Q$, $P \implies Q$, $P \iff Q$.

To answer questions in propositional logic, use *truth tables*. Or, use logical equivalences (e.g., De Morgan).

Midterm question: given a truth table

P	Q	$P \oplus Q$
T	T	F
T	F	T
F	T	T
F	F	F

can you write an equivalent sentence using P, Q, $\neg$, $\wedge$, $\vee$?

- Answer: $(P \wedge \neg Q) \vee (\neg P \wedge Q)$.

First-Order Logic

First-order logic introduces quantifiers: $\forall$, $\exists$.

First-Order Logic

First-order logic introduces quantifiers: $\forall$, $\exists$. Now we need more than truth tables; we need semantic proofs.

First-Order Logic

First-order logic introduces quantifiers: $\forall$, $\exists$. Now we need more than truth tables; we need semantic proofs.

Recall the intuition:

First-Order Logic

First-order logic introduces quantifiers: $\forall$, $\exists$. Now we need more than truth tables; we need semantic proofs.

Recall the intuition:

- $\forall$ is a way to write infinite "AND"s;

First-Order Logic

First-order logic introduces quantifiers: $\forall$, $\exists$. Now we need more than truth tables; we need semantic proofs.

Recall the intuition:

- $\forall$ is a way to write infinite "AND"s;
- $\exists$ is a way to write infinite "OR"s.

First-Order Logic

First-order logic introduces quantifiers: $\forall$, $\exists$. Now we need more than truth tables; we need semantic proofs.

Recall the intuition:

- $\forall$ is a way to write infinite "AND"s;
- $\exists$ is a way to write infinite "OR"s.

Recall De Morgan: $\neg\forall x\, P(x) \equiv \exists x\, \neg P(x)$ and $\neg\exists x\, P(x) \equiv \forall x\, \neg P(x)$.

First-Order Logic

First-order logic introduces quantifiers: $\forall$, $\exists$. Now we need more than truth tables; we need semantic proofs.

Recall the intuition:

- $\forall$ is a way to write infinite "AND"s;
- $\exists$ is a way to write infinite "OR"s.

Recall De Morgan: $\neg\forall x\, P(x) \equiv \exists x\, \neg P(x)$ and $\neg\exists x\, P(x) \equiv \forall x\, \neg P(x)$.

Question for review: is $\forall x\, \exists y\, P(x, y) \equiv \exists y\, \forall x\, P(x, y)$?

First-Order Logic

First-order logic introduces quantifiers: $\forall$, $\exists$. Now we need more than truth tables; we need semantic proofs.

Recall the intuition:

- $\forall$ is a way to write infinite "AND"s;
- $\exists$ is a way to write infinite "OR"s.

Recall De Morgan: $\neg\forall x\ P(x) \equiv \exists x\ \neg P(x)$ and $\neg\exists x\ P(x) \equiv \forall x\ \neg P(x)$.

Question for review: is $\forall x\ \exists y\ P(x,y) \equiv \exists y\ \forall x\ P(x,y)$?

- No; $P(x,y) =$ "x loves y".

Induction

Principle of induction: To prove a statement $\forall n \in \mathbb{N},\ P(n)$,

- (base case) prove $P(0)$;
- (inductive step) prove $\forall n \in \mathbb{N},\ P(n) \implies P(n+1)$.

Induction

Principle of induction: To prove a statement $\forall n \in \mathbb{N},\ P(n)$,

- (base case) prove $P(0)$;
- (inductive step) prove $\forall n \in \mathbb{N},\ P(n) \implies P(n+1)$.

Union bound: for events A, B, $\mathbb{P}(A \cup B) \leq \mathbb{P}(A) + \mathbb{P}(B)$.

Induction

Principle of induction: To prove a statement $\forall n \in \mathbb{N},\ P(n)$,

- (base case) prove $P(0)$;
- (inductive step) prove $\forall n \in \mathbb{N},\ P(n) \implies P(n+1)$.

Union bound: for events A, B, $\mathbb{P}(A \cup B) \leq \mathbb{P}(A) + \mathbb{P}(B)$.

For positive integers n and events $A_1, \ldots, A_n$, prove $\mathbb{P}(\bigcup_{i=1}^{n} A_i) \leq \sum_{i=1}^{n} \mathbb{P}(A_i)$?

Induction

Principle of induction: To prove a statement $\forall n \in \mathbb{N},\ P(n)$,

- (base case) prove $P(0)$;
- (inductive step) prove $\forall n \in \mathbb{N},\ P(n) \implies P(n+1)$.

Union bound: for events A, B, $\mathbb{P}(A \cup B) \leq \mathbb{P}(A) + \mathbb{P}(B)$.

For positive integers n and events $A_1, \ldots, A_n$, prove $\mathbb{P}(\bigcup_{i=1}^n A_i) \leq \sum_{i=1}^n \mathbb{P}(A_i)$?

- Base cases: $n = 1$ obvious; $n = 2$ is given above.

Induction

Principle of induction: To prove a statement $\forall n \in \mathbb{N},\ P(n)$,

- (base case) prove $P(0)$;
- (inductive step) prove $\forall n \in \mathbb{N},\ P(n) \implies P(n+1)$.

Union bound: for events A, B, $\mathbb{P}(A \cup B) \le \mathbb{P}(A) + \mathbb{P}(B)$.

For positive integers n and events $A_1, \ldots, A_n$, prove $\mathbb{P}(\bigcup_{i=1}^{n} A_i) \le \sum_{i=1}^{n} \mathbb{P}(A_i)$?

- Base cases: $n = 1$ obvious; $n = 2$ is given above.
- Inductive step: Assume $P(n)$.

Induction

Principle of induction: To prove a statement $\forall n \in \mathbb{N},\ P(n)$,

- (base case) prove $P(0)$;
- (inductive step) prove $\forall n \in \mathbb{N},\ P(n) \implies P(n+1)$.

Union bound: for events A, B, $\mathbb{P}(A \cup B) \leq \mathbb{P}(A) + \mathbb{P}(B)$.

For positive integers n and events $A_1, \ldots, A_n$, prove $\mathbb{P}(\bigcup_{i=1}^{n} A_i) \leq \sum_{i=1}^{n} \mathbb{P}(A_i)$?

- Base cases: $n = 1$ obvious; $n = 2$ is given above.
- Inductive step: Assume $P(n)$. Prove $P(n+1)$.

Induction

Principle of induction: To prove a statement $\forall n \in \mathbb{N},\ P(n)$,

- (base case) prove $P(0)$;
- (inductive step) prove $\forall n \in \mathbb{N},\ P(n) \implies P(n+1)$.

Union bound: for events A, B, $\mathbb{P}(A \cup B) \leq \mathbb{P}(A) + \mathbb{P}(B)$.

For positive integers n and events $A_1, \ldots, A_n$, prove $\mathbb{P}(\bigcup_{i=1}^{n} A_i) \leq \sum_{i=1}^{n} \mathbb{P}(A_i)$?

- Base cases: $n = 1$ obvious; $n = 2$ is given above.
- Inductive step: Assume $P(n)$. Prove $P(n+1)$.
- Let $A_1, \ldots, A_{n+1}$ be events.

Induction

Principle of induction: To prove a statement $\forall n \in \mathbb{N},\ P(n)$,

- (base case) prove $P(0)$;
- (inductive step) prove $\forall n \in \mathbb{N},\ P(n) \implies P(n+1)$.

Union bound: for events A, B, $\mathbb{P}(A \cup B) \le \mathbb{P}(A) + \mathbb{P}(B)$.

For positive integers n and events $A_1, \dots, A_n$, prove $\mathbb{P}(\bigcup_{i=1}^{n} A_i) \le \sum_{i=1}^{n} \mathbb{P}(A_i)$?

- Base cases: $n = 1$ obvious; $n = 2$ is given above.
- Inductive step: Assume $P(n)$. Prove $P(n+1)$.
- Let $A_1, \dots, A_{n+1}$ be events. Then, $\mathbb{P}(\bigcup_{i=1}^{n+1} A_i) = \mathbb{P}((\bigcup_{i=1}^{n} A_i) \cup A_{n+1}) \le \mathbb{P}(\bigcup_{i=1}^{n} A_i) + \mathbb{P}(A_{n+1})$.

Induction

Principle of induction: To prove a statement $\forall n \in \mathbb{N},\ P(n)$,

- (base case) prove $P(0)$;
- (inductive step) prove $\forall n \in \mathbb{N},\ P(n) \implies P(n+1)$.

Union bound: for events A, B, $\mathbb{P}(A \cup B) \leq \mathbb{P}(A) + \mathbb{P}(B)$.

For positive integers n and events $A_1, \ldots, A_n$, prove $\mathbb{P}(\bigcup_{i=1}^{n} A_i) \leq \sum_{i=1}^{n} \mathbb{P}(A_i)$?

- Base cases: $n = 1$ obvious; $n = 2$ is given above.
- Inductive step: Assume $P(n)$. Prove $P(n+1)$.
- Let $A_1, \ldots, A_{n+1}$ be events. Then, $\mathbb{P}(\bigcup_{i=1}^{n+1} A_i) = \mathbb{P}((\bigcup_{i=1}^{n} A_i) \cup A_{n+1}) \leq \mathbb{P}(\bigcup_{i=1}^{n} A_i) + \mathbb{P}(A_{n+1})$.
- Apply inductive hypothesis.

Induction

Principle of induction: To prove a statement $\forall n \in \mathbb{N},\ P(n)$,

- (base case) prove $P(0)$;
- (inductive step) prove $\forall n \in \mathbb{N},\ P(n) \implies P(n+1)$.

Union bound: for events A, B, $\mathbb{P}(A \cup B) \leq \mathbb{P}(A) + \mathbb{P}(B)$.

For positive integers n and events $A_1, \ldots, A_n$, prove $\mathbb{P}(\bigcup_{i=1}^{n} A_i) \leq \sum_{i=1}^{n} \mathbb{P}(A_i)$?

- Base cases: $n = 1$ obvious; $n = 2$ is given above.
- Inductive step: Assume $P(n)$. Prove $P(n+1)$.
- Let $A_1, \ldots, A_{n+1}$ be events. Then, $\mathbb{P}(\bigcup_{i=1}^{n+1} A_i) = \mathbb{P}((\bigcup_{i=1}^{n} A_i) \cup A_{n+1}) \leq \mathbb{P}(\bigcup_{i=1}^{n} A_i) + \mathbb{P}(A_{n+1})$.
- Apply inductive hypothesis. $\mathbb{P}(\bigcup_{i=1}^{n} A_i) \leq \sum_{i=1}^{n} \mathbb{P}(A_i)$.

Induction

Principle of induction: To prove a statement $\forall n \in \mathbb{N},\ P(n)$,

- (base case) prove $P(0)$;
- (inductive step) prove $\forall n \in \mathbb{N},\ P(n) \implies P(n+1)$.

Union bound: for events A, B, $\mathbb{P}(A \cup B) \leq \mathbb{P}(A) + \mathbb{P}(B)$.

For positive integers n and events $A_1, \ldots, A_n$, prove $\mathbb{P}(\bigcup_{i=1}^{n} A_i) \leq \sum_{i=1}^{n} \mathbb{P}(A_i)$?

- Base cases: $n = 1$ obvious; $n = 2$ is given above.
- Inductive step: Assume $P(n)$. Prove $P(n+1)$.
- Let $A_1, \ldots, A_{n+1}$ be events. Then, $\mathbb{P}(\bigcup_{i=1}^{n+1} A_i) = \mathbb{P}((\bigcup_{i=1}^{n} A_i) \cup A_{n+1}) \leq \mathbb{P}(\bigcup_{i=1}^{n} A_i) + \mathbb{P}(A_{n+1})$.
- Apply inductive hypothesis. $\mathbb{P}(\bigcup_{i=1}^{n} A_i) \leq \sum_{i=1}^{n} \mathbb{P}(A_i)$.
- So, $\mathbb{P}(\bigcup_{i=1}^{n+1} A_i) \leq \sum_{i=1}^{n+1} \mathbb{P}(A_i)$.

Other Forms of Induction

Strengthening the inductive hypothesis: Instead of proving $\forall n \in \mathbb{N},\ P(n)$, prove $\forall n \in \mathbb{N},\ Q(n)$, where $Q(n)$ implies $P(n)$.

Other Forms of Induction

Strengthening the inductive hypothesis: Instead of proving $\forall n \in \mathbb{N},\ P(n)$, prove $\forall n \in \mathbb{N},\ Q(n)$, where $Q(n)$ implies $P(n)$.

- Try tiling a $2^n \times 2^n$ grid with the upper right corner missing using L-shaped tiles.

Other Forms of Induction

Strengthening the inductive hypothesis: Instead of proving $\forall n \in \mathbb{N},\ P(n)$, prove $\forall n \in \mathbb{N},\ Q(n)$, where $Q(n)$ implies $P(n)$.

- Try tiling a $2^n \times 2^n$ grid with the upper right corner missing using L-shaped tiles. Get stuck at the inductive step!

Other Forms of Induction

Strengthening the inductive hypothesis: Instead of proving $\forall n \in \mathbb{N},\ P(n)$, prove $\forall n \in \mathbb{N},\ Q(n)$, where $Q(n)$ implies $P(n)$.

- Try tiling a $2^n \times 2^n$ grid with the upper right corner missing using L-shaped tiles. Get stuck at the inductive step!
- Instead, tile a $2^n \times 2^n$ grid with *any* square missing.

Other Forms of Induction

Strengthening the inductive hypothesis: Instead of proving $\forall n \in \mathbb{N},\ P(n)$, prove $\forall n \in \mathbb{N},\ Q(n)$, where $Q(n)$ implies $P(n)$.

- Try tiling a $2^n \times 2^n$ grid with the upper right corner missing using L-shaped tiles. Get stuck at the inductive step!
- Instead, tile a $2^n \times 2^n$ grid with *any* square missing.
- Use this when your inductive hypothesis does not give you enough information.

Other Forms of Induction

Strengthening the inductive hypothesis: Instead of proving $\forall n \in \mathbb{N},\ P(n)$, prove $\forall n \in \mathbb{N},\ Q(n)$, where $Q(n)$ implies $P(n)$.

- Try tiling a $2^n \times 2^n$ grid with the upper right corner missing using L-shaped tiles. Get stuck at the inductive step!
- Instead, tile a $2^n \times 2^n$ grid with *any* square missing.
- Use this when your inductive hypothesis does not give you enough information.

Strong induction: During inductive step, you can use $P(0), P(1), \ldots, P(n)$ to help you prove $P(n+1)$.

Other Forms of Induction

Strengthening the inductive hypothesis: Instead of proving $\forall n \in \mathbb{N},\ P(n)$, prove $\forall n \in \mathbb{N},\ Q(n)$, where $Q(n)$ implies $P(n)$.

- Try tiling a $2^n \times 2^n$ grid with the upper right corner missing using L-shaped tiles. Get stuck at the inductive step!
- Instead, tile a $2^n \times 2^n$ grid with *any* square missing.
- Use this when your inductive hypothesis does not give you enough information.

Strong induction: During inductive step, you can use $P(0), P(1), \ldots, P(n)$ to help you prove $P(n+1)$.

- This is needed when you reduce, not just to the *previous* case $P(n)$, but to an even smaller case.

Other Forms of Induction

Strengthening the inductive hypothesis: Instead of proving $\forall n \in \mathbb{N},\ P(n)$, prove $\forall n \in \mathbb{N},\ Q(n)$, where $Q(n)$ implies $P(n)$.

- Try tiling a $2^n \times 2^n$ grid with the upper right corner missing using L-shaped tiles. Get stuck at the inductive step!
- Instead, tile a $2^n \times 2^n$ grid with *any* square missing.
- Use this when your inductive hypothesis does not give you enough information.

Strong induction: During inductive step, you can use $P(0), P(1), \ldots, P(n)$ to help you prove $P(n+1)$.

- This is needed when you reduce, not just to the *previous* case $P(n)$, but to an even smaller case.

Well ordering principle: Every non-empty subset of $\mathbb{N}$ has a least element.

Other Forms of Induction

Strengthening the inductive hypothesis: Instead of proving $\forall n \in \mathbb{N},\ P(n)$, prove $\forall n \in \mathbb{N},\ Q(n)$, where $Q(n)$ implies $P(n)$.

- Try tiling a $2^n \times 2^n$ grid with the upper right corner missing using L-shaped tiles. Get stuck at the inductive step!
- Instead, tile a $2^n \times 2^n$ grid with *any* square missing.
- Use this when your inductive hypothesis does not give you enough information.

Strong induction: During inductive step, you can use $P(0), P(1), \ldots, P(n)$ to help you prove $P(n+1)$.

- This is needed when you reduce, not just to the *previous* case $P(n)$, but to an even smaller case.

Well ordering principle: Every non-empty subset of $\mathbb{N}$ has a least element.

- Consider the *least counterexample*;

Other Forms of Induction

Strengthening the inductive hypothesis: Instead of proving $\forall n \in \mathbb{N},\ P(n)$, prove $\forall n \in \mathbb{N},\ Q(n)$, where $Q(n)$ implies $P(n)$.

- Try tiling a $2^n \times 2^n$ grid with the upper right corner missing using L-shaped tiles. Get stuck at the inductive step!
- Instead, tile a $2^n \times 2^n$ grid with *any* square missing.
- Use this when your inductive hypothesis does not give you enough information.

Strong induction: During inductive step, you can use $P(0), P(1), \ldots, P(n)$ to help you prove $P(n+1)$.

- This is needed when you reduce, not just to the *previous* case $P(n)$, but to an even smaller case.

Well ordering principle: Every non-empty subset of $\mathbb{N}$ has a least element.

- Consider the *least counterexample*; prove there is an even smaller counterexample!

Graph Theory

A graph is a set of vertices V and a set of edges E.

Graph Theory

A graph is a set of vertices V and a set of edges E.

Recall definitions: degree, connectedness.

Graph Theory

A graph is a set of vertices V and a set of edges E.

Recall definitions: degree, connectedness. Types of graphs: trees, forests, planar, bipartite, complete, hypercubes.

Graph Theory

A graph is a set of vertices V and a set of edges E.

Recall definitions: degree, connectedness. Types of graphs: trees, forests, planar, bipartite, complete, hypercubes.

Confusing terminology: paths, walks, cycles, tours?

Graph Theory

A graph is a set of vertices V and a set of edges E.

Recall definitions: degree, connectedness. Types of graphs: trees, forests, planar, bipartite, complete, hypercubes.

Confusing terminology: paths, walks, cycles, tours?

	repeats vertices/edges?	must return to start?
path	no	no
walk	possibly	no
cycle	no	yes
tour	possibly	yes

Graph Theory Results

Handshaking Lemma: $\sum_{v \in V} \deg v = 2|E|$.

Graph Theory Results

Handshaking Lemma: $\sum_{v \in V} \deg v = 2|E|$.

- Example: For K_n, $n(n-1) = 2|E|$, so $|E| = n(n-1)/2 = \binom{n}{2}$.

Graph Theory Results

Handshaking Lemma: $\sum_{v \in V} \deg v = 2|E|$.

- Example: For K_n, $n(n-1) = 2|E|$, so $|E| = n(n-1)/2 = \binom{n}{2}$.

Eulerian tours: Use every edge exactly once.

Graph Theory Results

Handshaking Lemma: $\sum_{v \in V} \deg v = 2|E|$.

- Example: For K_n, $n(n-1) = 2|E|$, so $|E| = n(n-1)/2 = \binom{n}{2}$.

Eulerian tours: Use every edge exactly once.

- An Eulerian tour exists if and only if the graph is connected and every vertex has even degree.

Graph Theory Results

Handshaking Lemma: $\sum_{v \in V} \deg v = 2|E|$.

- Example: For K_n, $n(n-1) = 2|E|$, so $|E| = n(n-1)/2 = \binom{n}{2}$.

Eulerian tours: Use every edge exactly once.

- An Eulerian tour exists if and only if the graph is connected and every vertex has even degree.

Trees:

- Connected and acyclic;

Graph Theory Results

Handshaking Lemma: $\sum_{v \in V} \deg v = 2|E|$.

- Example: For K_n, $n(n-1) = 2|E|$, so $|E| = n(n-1)/2 = \binom{n}{2}$.

Eulerian tours: Use every edge exactly once.

- An Eulerian tour exists if and only if the graph is connected and every vertex has even degree.

Trees:

- Connected and acyclic; equivalently, connected and has $|V| - 1$ edges.

Graph Theory Results

Handshaking Lemma: $\sum_{v \in V} \deg v = 2|E|$.

- Example: For K_n, $n(n-1) = 2|E|$, so $|E| = n(n-1)/2 = \binom{n}{2}$.

Eulerian tours: Use every edge exactly once.

- An Eulerian tour exists if and only if the graph is connected and every vertex has even degree.

Trees:

- Connected and acyclic; equivalently, connected and has $|V| - 1$ edges. Smallest connected graphs!

Graph Theory Results

Handshaking Lemma: $\sum_{v \in V} \deg v = 2|E|$.

- Example: For K_n, $n(n-1) = 2|E|$, so $|E| = n(n-1)/2 = \binom{n}{2}$.

Eulerian tours: Use every edge exactly once.

- An Eulerian tour exists if and only if the graph is connected and every vertex has even degree.

Trees:

- Connected and acyclic; equivalently, connected and has $|V|-1$ edges. Smallest connected graphs!
- Trees are planar.

Graph Theory Results

Handshaking Lemma: $\sum_{v \in V} \deg v = 2|E|$.

- Example: For K_n, $n(n-1) = 2|E|$, so $|E| = n(n-1)/2 = \binom{n}{2}$.

Eulerian tours: Use every edge exactly once.

- An Eulerian tour exists if and only if the graph is connected and every vertex has even degree.

Trees:

- Connected and acyclic; equivalently, connected and has $|V|-1$ edges. Smallest connected graphs!
- Trees are planar.

Hypercubes:

- Vertices consist of length-d bit strings; two vertices are adjacent iff they differ in one bit.

Graph Theory Results

Handshaking Lemma: $\sum_{v \in V} \deg v = 2|E|$.

- Example: For K_n, $n(n-1) = 2|E|$, so $|E| = n(n-1)/2 = \binom{n}{2}$.

Eulerian tours: Use every edge exactly once.

- An Eulerian tour exists if and only if the graph is connected and every vertex has even degree.

Trees:

- Connected and acyclic; equivalently, connected and has $|V|-1$ edges. Smallest connected graphs!
- Trees are planar.

Hypercubes:

- Vertices consist of length-d bit strings; two vertices are adjacent iff they differ in one bit.
- Hypercubes are bipartite and have Hamiltonian cycles (visit each vertex exactly once).

Planarity

Planarity: can be drawn on a plane without edge crossings.

Planarity

Planarity: can be drawn on a plane without edge crossings.

- We only discussed connected planar graphs.

Planarity

Planarity: can be drawn on a plane without edge crossings.

- We only discussed connected planar graphs.
- Euler's formula: $v + f = e + 2$.

Planarity

Planarity: can be drawn on a plane without edge crossings.

- We only discussed connected planar graphs.
- Euler's formula: $v + f = e + 2$.
- For $|V| \geq 3$, this gives $e \leq 3v - 6$.

Planarity

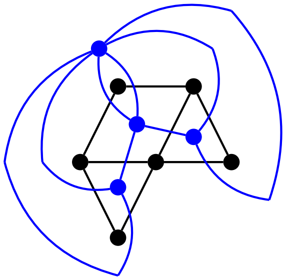
Planarity: can be drawn on a plane without edge crossings.

- We only discussed connected planar graphs.
- Euler's formula: $v + f = e + 2$.
- For $|V| \geq 3$, this gives $e \leq 3v - 6$.
- Important non-planar graphs: $K_{3,3}$, K_5.

Planarity

Planarity: can be drawn on a plane without edge crossings.

- We only discussed connected planar graphs.
- Euler's formula: $v + f = e + 2$.
- For $|V| \geq 3$, this gives $e \leq 3v - 6$.
- Important non-planar graphs: $K_{3,3}$, K_5.
- Every planar graph has a dual planar graph.



Graph Induction

A graph can be colored with $d_{\max} + 1$ colors, where $d_{\max}$ is the maximum degree of the graph.

Graph Induction

A graph can be colored with $d_{\max} + 1$ colors, where $d_{\max}$ is the maximum degree of the graph.

- Use induction on the number of vertices.

Graph Induction

A graph can be colored with $d_{\max} + 1$ colors, where $d_{\max}$ is the maximum degree of the graph.

- Use induction on the number of vertices.
- Base case: A graph with one vertex only needs one color.

Graph Induction

A graph can be colored with $d_{\max} + 1$ colors, where $d_{\max}$ is the maximum degree of the graph.

- Use induction on the number of vertices.
- Base case: A graph with one vertex only needs one color.
- Inductive hypothesis: Any graph H with n vertices can be colored with $d_{\max}(H) + 1$ colors.

Graph Induction

A graph can be colored with $d_{\max} + 1$ colors, where $d_{\max}$ is the maximum degree of the graph.

- Use induction on the number of vertices.
- Base case: A graph with one vertex only needs one color.
- Inductive hypothesis: Any graph H with n vertices can be colored with $d_{\max}(H) + 1$ colors.
- Consider a graph G with $n + 1$ vertices.

Graph Induction

A graph can be colored with $d_{\max}+1$ colors, where $d_{\max}$ is the maximum degree of the graph.

- Use induction on the number of vertices.
- Base case: A graph with one vertex only needs one color.
- Inductive hypothesis: Any graph H with n vertices can be colored with $d_{\max}(H)+1$ colors.
- Consider a graph G with $n+1$ vertices. Remove a vertex and its associated edges from G to form a graph G'.

Graph Induction

A graph can be colored with $d_{\max} + 1$ colors, where $d_{\max}$ is the maximum degree of the graph.

- Use induction on the number of vertices.
- Base case: A graph with one vertex only needs one color.
- Inductive hypothesis: Any graph H with n vertices can be colored with $d_{\max}(H) + 1$ colors.
- Consider a graph G with $n + 1$ vertices. Remove a vertex and its associated edges from G to form a graph G'.
- G' has n vertices, and $d_{\max}(G') \leq d_{\max}(G)$.

Graph Induction

A graph can be colored with $d_{\max} + 1$ colors, where $d_{\max}$ is the maximum degree of the graph.

- Use induction on the number of vertices.
- Base case: A graph with one vertex only needs one color.
- Inductive hypothesis: Any graph H with n vertices can be colored with $d_{\max}(H) + 1$ colors.
- Consider a graph G with $n + 1$ vertices. Remove a vertex and its associated edges from G to form a graph G'.
- G' has n vertices, and $d_{\max}(G') \leq d_{\max}(G)$. Apply inductive hypothesis to color G' with $\leq d_{\max}(G) + 1$ colors.

Graph Induction

A graph can be colored with $d_{\max}+1$ colors, where $d_{\max}$ is the maximum degree of the graph.

- Use induction on the number of vertices.
- Base case: A graph with one vertex only needs one color.
- Inductive hypothesis: Any graph H with n vertices can be colored with $d_{\max}(H)+1$ colors.
- Consider a graph G with $n+1$ vertices. Remove a vertex and its associated edges from G to form a graph G'.
- G' has n vertices, and $d_{\max}(G') \leq d_{\max}(G)$. Apply inductive hypothesis to color G' with $\leq d_{\max}(G)+1$ colors.
- Add the vertex and edges back to G' to form G.

Graph Induction

A graph can be colored with $d_{\max} + 1$ colors, where $d_{\max}$ is the maximum degree of the graph.

- Use induction on the number of vertices.
- Base case: A graph with one vertex only needs one color.
- Inductive hypothesis: Any graph H with n vertices can be colored with $d_{\max}(H) + 1$ colors.
- Consider a graph G with $n + 1$ vertices. Remove a vertex and its associated edges from G to form a graph G'.
- G' has n vertices, and $d_{\max}(G') \leq d_{\max}(G)$. Apply inductive hypothesis to color G' with $\leq d_{\max}(G) + 1$ colors.
- Add the vertex and edges back to G' to form G.
- Since the vertex has $\leq d_{\max}(G)$ neighbors, color it using color $d_{\max}(G) + 1$.

Modular Arithmetic

For a positive integer $m \geq 2$, say two numbers $x, y \in \mathbb{Z}$ are equivalent modulo m, $x \equiv y \pmod{m}$, if $m \mid x - y$.

Modular Arithmetic

For a positive integer $m \geq 2$, say two numbers $x, y \in \mathbb{Z}$ are equivalent modulo m, $x \equiv y \pmod{m}$, if $m \mid x - y$.

If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, then we can add and multiply these equations as normal:

$$a + c \equiv b + d \pmod{m}, \qquad ac \equiv bd \pmod{m}.$$

Modular Arithmetic

For a positive integer $m \geq 2$, say two numbers $x, y \in \mathbb{Z}$ are equivalent modulo m, $x \equiv y \pmod{m}$, if $m \mid x - y$.

If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, then we can add and multiply these equations as normal:

$$a + c \equiv b + d \pmod{m}, \qquad ac \equiv bd \pmod{m}.$$

Every $x \in \mathbb{Z}$ is equivalent to exactly one of $\{0, 1, \ldots, m-1\}$.

Modular Arithmetic

For a positive integer $m \geq 2$, say two numbers $x, y \in \mathbb{Z}$ are equivalent modulo m, $x \equiv y \pmod{m}$, if $m \mid x - y$.

If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, then we can add and multiply these equations as normal:

$$a + c \equiv b + d \pmod{m}, \qquad ac \equiv bd \pmod{m}.$$

Every $x \in \mathbb{Z}$ is equivalent to exactly one of $\{0, 1, \ldots, m-1\}$.
So, we let $\mathbb{Z}/m\mathbb{Z} = \{0, 1, \ldots, m-1\}$ be its own number system, with addition and multiplication defined modulo m.

Multiplicative Inverses

For $a \in \mathbb{Z}/m\mathbb{Z}$, the following are equivalent:

Multiplicative Inverses

For $a \in \mathbb{Z}/m\mathbb{Z}$, the following are equivalent:

- a has a multiplicative inverse in $\mathbb{Z}/m\mathbb{Z}$, i.e., there exists $x \in \mathbb{Z}/m\mathbb{Z}$ so that $ax = 1$.

Multiplicative Inverses

For $a \in \mathbb{Z}/m\mathbb{Z}$, the following are equivalent:

- a has a multiplicative inverse in $\mathbb{Z}/m\mathbb{Z}$, i.e., there exists $x \in \mathbb{Z}/m\mathbb{Z}$ so that $ax = 1$.
- $f : \mathbb{Z}/m\mathbb{Z} \to \mathbb{Z}/m\mathbb{Z}$ defined by $f(x) := ax$ is a bijection.

Multiplicative Inverses

For $a \in \mathbb{Z}/m\mathbb{Z}$, the following are equivalent:

- a has a multiplicative inverse in $\mathbb{Z}/m\mathbb{Z}$, i.e., there exists $x \in \mathbb{Z}/m\mathbb{Z}$ so that $ax = 1$.
- $f : \mathbb{Z}/m\mathbb{Z} \to \mathbb{Z}/m\mathbb{Z}$ defined by $f(x) := ax$ is a bijection.
- $\gcd(a, m) = 1$.

Multiplicative Inverses

For $a \in \mathbb{Z}/m\mathbb{Z}$, the following are equivalent:

- a has a multiplicative inverse in $\mathbb{Z}/m\mathbb{Z}$, i.e., there exists $x \in \mathbb{Z}/m\mathbb{Z}$ so that $ax = 1$.
- $f : \mathbb{Z}/m\mathbb{Z} \to \mathbb{Z}/m\mathbb{Z}$ defined by $f(x) := ax$ is a bijection.
- $\gcd(a, m) = 1$.

If a satisfies the three statements above, then we say $a \in (\mathbb{Z}/m\mathbb{Z})^{\times}$.

Multiplicative Inverses

For $a \in \mathbb{Z}/m\mathbb{Z}$, the following are equivalent:

- a has a multiplicative inverse in $\mathbb{Z}/m\mathbb{Z}$, i.e., there exists $x \in \mathbb{Z}/m\mathbb{Z}$ so that $ax = 1$.
- $f : \mathbb{Z}/m\mathbb{Z} \to \mathbb{Z}/m\mathbb{Z}$ defined by $f(x) := ax$ is a bijection.
- $\gcd(a, m) = 1$.

If a satisfies the three statements above, then we say $a \in (\mathbb{Z}/m\mathbb{Z})^\times$.

When p is prime, then $(\mathbb{Z}/p\mathbb{Z})^\times = \{1, \ldots, p-1\}$.

Multiplicative Inverses

For $a \in \mathbb{Z}/m\mathbb{Z}$, the following are equivalent:

- a has a multiplicative inverse in $\mathbb{Z}/m\mathbb{Z}$, i.e., there exists $x \in \mathbb{Z}/m\mathbb{Z}$ so that $ax = 1$.
- $f : \mathbb{Z}/m\mathbb{Z} \to \mathbb{Z}/m\mathbb{Z}$ defined by $f(x) := ax$ is a bijection.
- $\gcd(a, m) = 1$.

If a satisfies the three statements above, then we say $a \in (\mathbb{Z}/m\mathbb{Z})^\times$.

When p is prime, then $(\mathbb{Z}/p\mathbb{Z})^\times = \{1, \ldots, p-1\}$. Every non-zero element has a multiplicative inverse.

Multiplicative Inverses

For $a \in \mathbb{Z}/m\mathbb{Z}$, the following are equivalent:

- a has a multiplicative inverse in $\mathbb{Z}/m\mathbb{Z}$, i.e., there exists $x \in \mathbb{Z}/m\mathbb{Z}$ so that $ax = 1$.
- $f : \mathbb{Z}/m\mathbb{Z} \to \mathbb{Z}/m\mathbb{Z}$ defined by $f(x) := ax$ is a bijection.
- $\gcd(a, m) = 1$.

If a satisfies the three statements above, then we say $a \in (\mathbb{Z}/m\mathbb{Z})^\times$.

When p is prime, then $(\mathbb{Z}/p\mathbb{Z})^\times = \{1, \ldots, p-1\}$. Every non-zero element has a multiplicative inverse.

Extended Euclid's algorithm: given $a, m \in \mathbb{Z}$, $m \neq 0$, output $x, y \in \mathbb{Z}$ such that $ax + my = \gcd(a, m)$.

Multiplicative Inverses

For $a \in \mathbb{Z}/m\mathbb{Z}$, the following are equivalent:

- a has a multiplicative inverse in $\mathbb{Z}/m\mathbb{Z}$, i.e., there exists $x \in \mathbb{Z}/m\mathbb{Z}$ so that $ax = 1$.
- $f : \mathbb{Z}/m\mathbb{Z} \to \mathbb{Z}/m\mathbb{Z}$ defined by $f(x) := ax$ is a bijection.
- $\gcd(a, m) = 1$.

If a satisfies the three statements above, then we say $a \in (\mathbb{Z}/m\mathbb{Z})^\times$.

When p is prime, then $(\mathbb{Z}/p\mathbb{Z})^\times = \{1, \ldots, p-1\}$. Every non-zero element has a multiplicative inverse.

Extended Euclid's algorithm: given $a, m \in \mathbb{Z}$, $m \neq 0$, output $x, y \in \mathbb{Z}$ such that $ax + my = \gcd(a, m)$.

- For $a \in (\mathbb{Z}/m\mathbb{Z})^\times$, this gives $ax + my = 1$.

Multiplicative Inverses

For $a \in \mathbb{Z}/m\mathbb{Z}$, the following are equivalent:

- a has a multiplicative inverse in $\mathbb{Z}/m\mathbb{Z}$, i.e., there exists $x \in \mathbb{Z}/m\mathbb{Z}$ so that $ax = 1$.
- $f : \mathbb{Z}/m\mathbb{Z} \to \mathbb{Z}/m\mathbb{Z}$ defined by $f(x) := ax$ is a bijection.
- $\gcd(a, m) = 1$.

If a satisfies the three statements above, then we say $a \in (\mathbb{Z}/m\mathbb{Z})^\times$.

When p is prime, then $(\mathbb{Z}/p\mathbb{Z})^\times = \{1, \ldots, p-1\}$. Every non-zero element has a multiplicative inverse.

Extended Euclid's algorithm: given $a, m \in \mathbb{Z}$, $m \neq 0$, output $x, y \in \mathbb{Z}$ such that $ax + my = \gcd(a, m)$.

- For $a \in (\mathbb{Z}/m\mathbb{Z})^\times$, this gives $ax + my = 1$. So, x is the multiplicative inverse of a in $\mathbb{Z}/m\mathbb{Z}$.

Modular Arithmetic Results

Repeated squaring (or fast modular exponentiation): Calculate $a^b \bmod m$ fast!

Modular Arithmetic Results

Repeated squaring (or fast modular exponentiation): Calculate $a^b \bmod m$ fast!

- Try $3^{60} \bmod 13$.

Modular Arithmetic Results

Repeated squaring (or fast modular exponentiation): Calculate $a^b \bmod m$ fast!

- Try $3^{60} \bmod 13$.
- Square the base, halve the exponent.

Modular Arithmetic Results

Repeated squaring (or fast modular exponentiation): Calculate $a^b \bmod m$ fast!

- Try $3^{60} \bmod 13$.
- Square the base, halve the exponent. $3^{60} = 9^{30} = 81^{15}$.

Modular Arithmetic Results

Repeated squaring (or fast modular exponentiation): Calculate $a^b \bmod m$ fast!

- Try $3^{60} \bmod 13$.
- Square the base, halve the exponent. $3^{60} = 9^{30} = 81^{15}$.
- Reduce the base: $81^{15} = 3^{15}$.

Modular Arithmetic Results

Repeated squaring (or fast modular exponentiation): Calculate $a^b \bmod m$ fast!

- Try $3^{60} \bmod 13$.
- Square the base, halve the exponent. $3^{60} = 9^{30} = 81^{15}$.
- Reduce the base: $81^{15} = 3^{15}$.
- For an odd exponent, pull out one power.

Modular Arithmetic Results

Repeated squaring (or fast modular exponentiation): Calculate $a^b \bmod m$ fast!

- Try $3^{60} \bmod 13$.
- Square the base, halve the exponent. $3^{60} = 9^{30} = 81^{15}$.
- Reduce the base: $81^{15} = 3^{15}$.
- For an odd exponent, pull out one power.
  $3^{15} = 3 \cdot 3^{14} = 3 \cdot 9^7 = \cdots$

Modular Arithmetic Results

Repeated squaring (or fast modular exponentiation): Calculate $a^b \bmod m$ fast!

- Try $3^{60} \bmod 13$.
- Square the base, halve the exponent. $3^{60} = 9^{30} = 81^{15}$.
- Reduce the base: $81^{15} = 3^{15}$.
- For an odd exponent, pull out one power. $3^{15} = 3 \cdot 3^{14} = 3 \cdot 9^7 = \cdots$

Fermat's Little Theorem: For p prime and $a \in (\mathbb{Z}/p\mathbb{Z})^\times$, one has $a^{p-1} \equiv 1 \pmod{p}$.

Modular Arithmetic Results

Repeated squaring (or fast modular exponentiation): Calculate $a^b \bmod m$ fast!

- Try $3^{60} \bmod 13$.
- Square the base, halve the exponent. $3^{60} = 9^{30} = 81^{15}$.
- Reduce the base: $81^{15} = 3^{15}$.
- For an odd exponent, pull out one power. $3^{15} = 3 \cdot 3^{14} = 3 \cdot 9^7 = \cdots$

Fermat's Little Theorem: For p prime and $a \in (\mathbb{Z}/p\mathbb{Z})^\times$, one has $a^{p-1} \equiv 1 \pmod p$.

- Or, for all $a \in \mathbb{Z}/p\mathbb{Z}$, $a^p \equiv a \pmod p$.

Modular Arithmetic Results

Repeated squaring (or fast modular exponentiation): Calculate $a^b \bmod m$ fast!

- Try $3^{60} \bmod 13$.
- Square the base, halve the exponent. $3^{60} = 9^{30} = 81^{15}$.
- Reduce the base: $81^{15} = 3^{15}$.
- For an odd exponent, pull out one power.
  $3^{15} = 3 \cdot 3^{14} = 3 \cdot 9^7 = \cdots$

Fermat's Little Theorem: For p prime and $a \in (\mathbb{Z}/p\mathbb{Z})^\times$, one has $a^{p-1} \equiv 1 \pmod{p}$.

- Or, for all $a \in \mathbb{Z}/p\mathbb{Z}$, $a^p \equiv a \pmod{p}$.

Chinese Remainder Theorem: For pairwise coprime moduli $m_1, \ldots, m_n$ and fixed $a_1, \ldots, a_n$, the equations $x \equiv a_i \pmod{m_i}$ for $i = 1, \ldots, n$ has a unique solution $x \in \mathbb{Z}/m_1 \cdots m_n\mathbb{Z}$.

RSA

RSA public-key cryptosystem:

RSA

RSA public-key cryptosystem:

- Generate two distinct large primes, p and q.

RSA

RSA public-key cryptosystem:

- Generate two distinct large primes, p and q. Let $N := pq$.

RSA

RSA public-key cryptosystem:

- Generate two distinct large primes, p and q. Let $N := pq$.
- Pick a public key $e \in (\mathbb{Z}/(p-1)(q-1)\mathbb{Z})^\times$.

RSA

RSA public-key cryptosystem:

- Generate two distinct large primes, p and q. Let $N := pq$.
- Pick a public key $e \in (\mathbb{Z}/(p-1)(q-1)\mathbb{Z})^\times$. The private key d is the inverse of e in $\mathbb{Z}/(p-1)(q-1)\mathbb{Z}$.

RSA

RSA public-key cryptosystem:

- Generate two distinct large primes, p and q. Let $N := pq$.
- Pick a public key $e \in (\mathbb{Z}/(p-1)(q-1)\mathbb{Z})^\times$. The private key d is the inverse of e in $\mathbb{Z}/(p-1)(q-1)\mathbb{Z}$.
- Public information: (N, e).

RSA

RSA public-key cryptosystem:

- Generate two distinct large primes, p and q. Let $N := pq$.
- Pick a public key $e \in (\mathbb{Z}/(p-1)(q-1)\mathbb{Z})^\times$. The private key d is the inverse of e in $\mathbb{Z}/(p-1)(q-1)\mathbb{Z}$.
- Public information: (N, e). Only the receiver knows d.

RSA

RSA public-key cryptosystem:

- Generate two distinct large primes, p and q. Let $N := pq$.
- Pick a public key $e \in (\mathbb{Z}/(p-1)(q-1)\mathbb{Z})^\times$. The private key d is the inverse of e in $\mathbb{Z}/(p-1)(q-1)\mathbb{Z}$.
- Public information: (N, e). Only the receiver knows d.
- For a message m, encrypt using $E(m) = m^e \pmod{N}$ and then send.

RSA

RSA public-key cryptosystem:

- Generate two distinct large primes, p and q. Let $N := pq$.
- Pick a public key $e \in (\mathbb{Z}/(p-1)(q-1)\mathbb{Z})^\times$. The private key d is the inverse of e in $\mathbb{Z}/(p-1)(q-1)\mathbb{Z}$.
- Public information: (N, e). Only the receiver knows d.
- For a message m, encrypt using $E(m) = m^e \pmod{N}$ and then send. Receiver decrypts using $D(c) = c^d \pmod{N}$.

RSA

RSA public-key cryptosystem:

- Generate two distinct large primes, p and q. Let $N := pq$.
- Pick a public key $e \in (\mathbb{Z}/(p-1)(q-1)\mathbb{Z})^\times$. The private key d is the inverse of e in $\mathbb{Z}/(p-1)(q-1)\mathbb{Z}$.
- Public information: (N, e). Only the receiver knows d.
- For a message m, encrypt using $E(m) = m^e \pmod{N}$ and then send. Receiver decrypts using $D(c) = c^d \pmod{N}$.

RSA details:

RSA

RSA public-key cryptosystem:

- Generate two distinct large primes, p and q. Let $N := pq$.
- Pick a public key $e \in (\mathbb{Z}/(p-1)(q-1)\mathbb{Z})^{\times}$. The private key d is the inverse of e in $\mathbb{Z}/(p-1)(q-1)\mathbb{Z}$.
- Public information: (N, e). Only the receiver knows d.
- For a message m, encrypt using $E(m) = m^e \pmod{N}$ and then send. Receiver decrypts using $D(c) = c^d \pmod{N}$.

RSA details:

- Correctness: Proof uses Fermat's Little Theorem.

RSA

RSA public-key cryptosystem:

- Generate two distinct large primes, p and q. Let $N := pq$.
- Pick a public key $e \in (\mathbb{Z}/(p-1)(q-1)\mathbb{Z})^{\times}$. The private key d is the inverse of e in $\mathbb{Z}/(p-1)(q-1)\mathbb{Z}$.
- Public information: (N, e). Only the receiver knows d.
- For a message m, encrypt using $E(m) = m^e \pmod{N}$ and then send. Receiver decrypts using $D(c) = c^d \pmod{N}$.

RSA details:

- Correctness: Proof uses Fermat's Little Theorem.
- Efficiency: Repeated squaring, extended Euclid, Prime Number Theorem, primality tests.

RSA

RSA public-key cryptosystem:

- Generate two distinct large primes, p and q. Let $N := pq$.
- Pick a public key $e \in (\mathbb{Z}/(p-1)(q-1)\mathbb{Z})^\times$. The private key d is the inverse of e in $\mathbb{Z}/(p-1)(q-1)\mathbb{Z}$.
- Public information: (N, e). Only the receiver knows d.
- For a message m, encrypt using $E(m) = m^e \pmod{N}$ and then send. Receiver decrypts using $D(c) = c^d \pmod{N}$.

RSA details:

- Correctness: Proof uses Fermat's Little Theorem.
- Efficiency: Repeated squaring, extended Euclid, Prime Number Theorem, primality tests.
- Security: *Conjectured* to be secure.

Polynomials

A polynomial is of the form $P(x) = a_d x^d + \cdots + a_1 x + a_0$, where $d \in \mathbb{N}$ is the degree and $a_0, a_1, \ldots, a_d$ are the coefficients.

Polynomials

A polynomial is of the form $P(x) = a_d x^d + \cdots + a_1 x + a_0$, where $d \in \mathbb{N}$ is the degree and $a_0, a_1, \ldots, a_d$ are the coefficients.

We look at polynomials over *fields*.

Polynomials

A polynomial is of the form $P(x) = a_d x^d + \cdots + a_1 x + a_0$, where $d \in \mathbb{N}$ is the degree and $a_0, a_1, \ldots, a_d$ are the coefficients.

We look at polynomials over *fields*. Here are fields we care about: $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, $\mathbb{Z}/p\mathbb{Z}$ for p prime.

Polynomials

A polynomial is of the form $P(x) = a_d x^d + \cdots + a_1 x + a_0$, where $d \in \mathbb{N}$ is the degree and $a_0, a_1, \ldots, a_d$ are the coefficients.

We look at polynomials over *fields*. Here are fields we care about: $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, $\mathbb{Z}/p\mathbb{Z}$ for p prime.

Facts about polynomials in fields:

Polynomials

A polynomial is of the form $P(x) = a_d x^d + \cdots + a_1 x + a_0$, where $d \in \mathbb{N}$ is the degree and $a_0, a_1, \ldots, a_d$ are the coefficients.

We look at polynomials over *fields*. Here are fields we care about: $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, $\mathbb{Z}/p\mathbb{Z}$ for p prime.

Facts about polynomials in fields:

- A degree d polynomial has $\leq d$ roots.

Polynomials

A polynomial is of the form $P(x) = a_d x^d + \cdots + a_1 x + a_0$, where $d \in \mathbb{N}$ is the degree and $a_0, a_1, \ldots, a_d$ are the coefficients.

We look at polynomials over *fields*. Here are fields we care about: $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, $\mathbb{Z}/p\mathbb{Z}$ for p prime.

Facts about polynomials in fields:

- A degree d polynomial has $\leq d$ roots.
- There is a unique degree $\leq d$ polynomial which passes through any specified $d+1$ distinct points.

Polynomials

A polynomial is of the form $P(x) = a_d x^d + \cdots + a_1 x + a_0$, where $d \in \mathbb{N}$ is the degree and $a_0, a_1, \ldots, a_d$ are the coefficients.

We look at polynomials over *fields*. Here are fields we care about: $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, $\mathbb{Z}/p\mathbb{Z}$ for p prime.

Facts about polynomials in fields:

- A degree d polynomial has $\leq d$ roots.
- There is a unique degree $\leq d$ polynomial which passes through any specified $d+1$ distinct points.
- Lagrange interpolation: given distinct $(x_1, y_1), \ldots, (x_{d+1}, y_{d+1})$, then $P(x) := \sum_{i=1}^{d+1} y_i \Delta_i(x)$, where

$$\Delta_i(x) := \frac{\prod_{j \in \{1,\ldots,d+1\} \setminus \{i\}} (x - x_j)}{\prod_{j \in \{1,\ldots,d+1\} \setminus \{i\}} (x_i - x_j)},$$

  is the unique degree $\leq d$ interpolating polynomial.

Midterm Question

Polynomials P and Q (over $\mathbb{Z}/p\mathbb{Z}$) are equivalent modulo x^2+1 if $P(x)-Q(x) = K(x)(x^2+1)$ for some polynomial K.

Midterm Question

Polynomials P and Q (over $\mathbb{Z}/p\mathbb{Z}$) are equivalent modulo x^2+1 if $P(x)-Q(x) = K(x)(x^2+1)$ for some polynomial K.

- Similar to the definition of modular equivalence!

Midterm Question

Polynomials P and Q (over $\mathbb{Z}/p\mathbb{Z}$) are equivalent modulo x^2+1 if $P(x)-Q(x)=K(x)(x^2+1)$ for some polynomial K.

- Similar to the definition of modular equivalence!

How many polynomials can you put into a set so that no two of them are equivalent modulo x^2+1?

Midterm Question

Polynomials P and Q (over $\mathbb{Z}/p\mathbb{Z}$) are equivalent modulo x^2+1 if $P(x) - Q(x) = K(x)(x^2+1)$ for some polynomial K.

- Similar to the definition of modular equivalence!

How many polynomials can you put into a set so that no two of them are equivalent modulo x^2+1?

- First step: How many numbers are in $\mathbb{Z}/m\mathbb{Z}$?

Midterm Question

Polynomials P and Q (over $\mathbb{Z}/p\mathbb{Z}$) are equivalent modulo x^2+1 if $P(x)-Q(x)=K(x)(x^2+1)$ for some polynomial K.

- Similar to the definition of modular equivalence!

How many polynomials can you put into a set so that no two of them are equivalent modulo x^2+1?

- First step: How many numbers are in $\mathbb{Z}/m\mathbb{Z}$?
- For $x\in\mathbb{Z}$, Division Algorithm gives $x=qm+r$ where $q\in\mathbb{Z}$ and $r\in\{0,1,\dots,m-1\}$.

Midterm Question

Polynomials P and Q (over $\mathbb{Z}/p\mathbb{Z}$) are equivalent modulo x^2+1 if $P(x)-Q(x) = K(x)(x^2+1)$ for some polynomial K.

- Similar to the definition of modular equivalence!

How many polynomials can you put into a set so that no two of them are equivalent modulo x^2+1?

- First step: How many numbers are in $\mathbb{Z}/m\mathbb{Z}$?
- For $x \in \mathbb{Z}$, Division Algorithm gives $x = qm + r$ where $q \in \mathbb{Z}$ and $r \in \{0, 1, \ldots, m-1\}$. So, $x \equiv r \pmod{m}$.

Midterm Question

Polynomials P and Q (over $\mathbb{Z}/p\mathbb{Z}$) are equivalent modulo x^2+1 if $P(x)-Q(x)=K(x)(x^2+1)$ for some polynomial K.

- Similar to the definition of modular equivalence!

How many polynomials can you put into a set so that no two of them are equivalent modulo x^2+1?

- First step: How many numbers are in $\mathbb{Z}/m\mathbb{Z}$?
- For $x \in \mathbb{Z}$, Division Algorithm gives $x = qm + r$ where $q \in \mathbb{Z}$ and $r \in \{0, 1, \ldots, m-1\}$. So, $x \equiv r \pmod{m}$.
- Similarly, $P(x) = Q(x)(x^2+1) + R(x)$ for polynomials Q and R, where $\deg R < 2$.

Midterm Question

Polynomials P and Q (over $\mathbb{Z}/p\mathbb{Z}$) are equivalent modulo x^2+1 if $P(x)-Q(x)=K(x)(x^2+1)$ for some polynomial K.

- Similar to the definition of modular equivalence!

How many polynomials can you put into a set so that no two of them are equivalent modulo x^2+1?

- First step: How many numbers are in $\mathbb{Z}/m\mathbb{Z}$?
- For $x\in\mathbb{Z}$, Division Algorithm gives $x=qm+r$ where $q\in\mathbb{Z}$ and $r\in\{0,1,\ldots,m-1\}$. So, $x\equiv r \pmod{m}$.
- Similarly, $P(x)=Q(x)(x^2+1)+R(x)$ for polynomials Q and R, where $\deg R<2$.
- So, $R(x)=r_1x+r_0$ for some r_0, r_1.

Midterm Question

Polynomials P and Q (over $\mathbb{Z}/p\mathbb{Z}$) are equivalent modulo x^2+1 if $P(x)-Q(x)=K(x)(x^2+1)$ for some polynomial K.

- Similar to the definition of modular equivalence!

How many polynomials can you put into a set so that no two of them are equivalent modulo x^2+1?

- First step: How many numbers are in $\mathbb{Z}/m\mathbb{Z}$?
- For $x \in \mathbb{Z}$, Division Algorithm gives $x = qm + r$ where $q \in \mathbb{Z}$ and $r \in \{0, 1, \ldots, m-1\}$. So, $x \equiv r \pmod{m}$.
- Similarly, $P(x) = Q(x)(x^2+1) + R(x)$ for polynomials Q and R, where $\deg R < 2$.
- So, $R(x) = r_1 x + r_0$ for some r_0, r_1.
- Since we are in $\mathbb{Z}/p\mathbb{Z}$, there are p choices for r_0 and r_1, so there are p^2 different non-equivalent polynomials.

Applications of Polynomials

Shamir's secret sharing:

Applications of Polynomials

Shamir's secret sharing:

- If k officers get together, they know the secret $s \in \mathbb{Z}/p\mathbb{Z}$.

Applications of Polynomials

Shamir's secret sharing:

- If k officers get together, they know the secret $s \in \mathbb{Z}/p\mathbb{Z}$. If $\leq k-1$ officers get together, they learn nothing.

Applications of Polynomials

Shamir's secret sharing:

- If k officers get together, they know the secret $s \in \mathbb{Z}/p\mathbb{Z}$. If $\leq k-1$ officers get together, they learn nothing.
- Define $P(x) := s_{k-1}x^{k-1} + \cdots + s_1 x + s$, where $s_1, \ldots, s_{k-1}$ are chosen randomly.

Applications of Polynomials

Shamir's secret sharing:

- If k officers get together, they know the secret $s \in \mathbb{Z}/p\mathbb{Z}$. If $\leq k-1$ officers get together, they learn nothing.
- Define $P(x) := s_{k-1}x^{k-1} + \cdots + s_1 x + s$, where $s_1, \ldots, s_{k-1}$ are chosen randomly.
- Give each officer an evaluation of the polynomial.

Applications of Polynomials

Shamir's secret sharing:

- If k officers get together, they know the secret $s \in \mathbb{Z}/p\mathbb{Z}$. If $\leq k-1$ officers get together, they learn nothing.
- Define $P(x) := s_{k-1}x^{k-1} + \cdots + s_1 x + s$, where $s_1, \ldots, s_{k-1}$ are chosen randomly.
- Give each officer an evaluation of the polynomial.

Reed-Solomon codes:

Applications of Polynomials

Shamir's secret sharing:

- If k officers get together, they know the secret $s \in \mathbb{Z}/p\mathbb{Z}$. If $\leq k-1$ officers get together, they learn nothing.
- Define $P(x) := s_{k-1}x^{k-1} + \cdots + s_1 x + s$, where $s_1, \ldots, s_{k-1}$ are chosen randomly.
- Give each officer an evaluation of the polynomial.

Reed-Solomon codes:

- Given a message $(m_0, m_1, \ldots, m_{n-1})$, encode it as a polynomial $P(x) = m_{n-1}x^{n-1} + \cdots + m_1 x + m_0$.

Applications of Polynomials

Shamir's secret sharing:

- If k officers get together, they know the secret $s \in \mathbb{Z}/p\mathbb{Z}$. If $\leq k-1$ officers get together, they learn nothing.
- Define $P(x) := s_{k-1}x^{k-1} + \cdots + s_1 x + s$, where $s_1, \ldots, s_{k-1}$ are chosen randomly.
- Give each officer an evaluation of the polynomial.

Reed-Solomon codes:

- Given a message $(m_0, m_1, \ldots, m_{n-1})$, encode it as a polynomial $P(x) = m_{n-1}x^{n-1} + \cdots + m_1 x + m_0$.
- Encode the message as a codeword of length ℓ.

Applications of Polynomials

Shamir's secret sharing:

- If k officers get together, they know the secret $s \in \mathbb{Z}/p\mathbb{Z}$. If $\leq k-1$ officers get together, they learn nothing.
- Define $P(x) := s_{k-1}x^{k-1} + \cdots + s_1 x + s$, where $s_1, \ldots, s_{k-1}$ are chosen randomly.
- Give each officer an evaluation of the polynomial.

Reed-Solomon codes:

- Given a message $(m_0, m_1, \ldots, m_{n-1})$, encode it as a polynomial $P(x) = m_{n-1}x^{n-1} + \cdots + m_1 x + m_0$.
- Encode the message as a codeword of length ℓ.
- The codeword for the message is $(0, P(0)), (1, P(1)), \ldots, (\ell - 1, P(\ell - 1))$.

Reedand-Solomon Error Correction

- A Reed-Solomon code with codeword length $\ell = n + k$ can recover the message if $\leq k$ packets are *erased*.

Reed-Solomon Error Correction

- A Reed-Solomon code with codeword length $\ell = n + k$ can recover the message if $\leq k$ packets are *erased*.
- A Reed-Solomon code with codeword length $\ell = n + 2k$ can recover the message if $\leq k$ packets are *corrupted*.

Reed-Solomon Error Correction

- A Reed-Solomon code with codeword length $\ell = n + k$ can recover the message if $\leq k$ packets are *erased*.
- A Reed-Solomon code with codeword length $\ell = n + 2k$ can recover the message if $\leq k$ packets are *corrupted*.
  - This code has minimum pairwise Hamming distance $2k + 1$ $\implies$ correct k general errors.

Reed-Solomon Error Correction

- A Reed-Solomon code with codeword length $\ell = n + k$ can recover the message if $\leq k$ packets are *erased*.
- A Reed-Solomon code with codeword length $\ell = n + 2k$ can recover the message if $\leq k$ packets are *corrupted*.
  - This code has minimum pairwise Hamming distance $2k + 1$ $\implies$ correct k general errors.
- Berlekamp-Welch: An efficient decoding scheme for Reed-Solomon codes under corruption errors.

Reed-Solomon Error Correction

- A Reed-Solomon code with codeword length $\ell = n + k$ can recover the message if $\leq k$ packets are *erased*.
- A Reed-Solomon code with codeword length $\ell = n + 2k$ can recover the message if $\leq k$ packets are *corrupted*.
  - This code has minimum pairwise Hamming distance $2k + 1$ $\implies$ correct k general errors.
- Berlekamp-Welch: An efficient decoding scheme for Reed-Solomon codes under corruption errors.
  - If errors are at $e_1, \ldots, e_k$, define $E(x) = \prod_{i=1}^{k}(x - e_i) = x^k + a_{k-1}x^{k-1} + \cdots + a_1 x + a_0$.

Reed-Solomon Error Correction

- A Reed-Solomon code with codeword length $\ell = n + k$ can recover the message if $\leq k$ packets are *erased*.
- A Reed-Solomon code with codeword length $\ell = n + 2k$ can recover the message if $\leq k$ packets are *corrupted*.
  - This code has minimum pairwise Hamming distance $2k + 1$ $\implies$ correct k general errors.
- Berlekamp-Welch: An efficient decoding scheme for Reed-Solomon codes under corruption errors.
  - If errors are at $e_1, \ldots, e_k$, define $E(x) = \prod_{i=1}^{k}(x - e_i) = x^k + a_{k-1}x^{k-1} + \cdots + a_1 x + a_0$.
  - Define $Q(x) = P(x)E(x) = b_{n+k-1}x^{n+k-1} + \cdots + b_1 x + b_0$.

Reed-Solomon Error Correction

- A Reed-Solomon code with codeword length $\ell = n + k$ can recover the message if $\leq k$ packets are *erased*.
- A Reed-Solomon code with codeword length $\ell = n + 2k$ can recover the message if $\leq k$ packets are *corrupted*.
  - This code has minimum pairwise Hamming distance $2k + 1$ $\implies$ correct k general errors.
- Berlekamp-Welch: An efficient decoding scheme for Reed-Solomon codes under corruption errors.
  - If errors are at $e_1, \ldots, e_k$, define $E(x) = \prod_{i=1}^{k}(x - e_i) = x^k + a_{k-1}x^{k-1} + \cdots + a_1 x + a_0$.
  - Define $Q(x) = P(x)E(x) = b_{n+k-1}x^{n+k-1} + \cdots + b_1 x + b_0$.
  - Key Lemma: If $R_0, R_1, \ldots, R_{n+2k-1}$ are the received packets, then $R_i E(i) = P(i)E(i)$ for $i = 0, 1, \ldots, n + 2k - 1$.

Reed-Solomon Error Correction

- A Reed-Solomon code with codeword length $\ell = n + k$ can recover the message if $\leq k$ packets are *erased*.
- A Reed-Solomon code with codeword length $\ell = n + 2k$ can recover the message if $\leq k$ packets are *corrupted*.
  - This code has minimum pairwise Hamming distance $2k + 1$ $\implies$ correct k general errors.
- Berlekamp-Welch: An efficient decoding scheme for Reed-Solomon codes under corruption errors.
  - If errors are at $e_1, \ldots, e_k$, define $E(x) = \prod_{i=1}^{k}(x - e_i) = x^k + a_{k-1}x^{k-1} + \cdots + a_1 x + a_0$.
  - Define $Q(x) = P(x)E(x) = b_{n+k-1}x^{n+k-1} + \cdots + b_1 x + b_0$.
  - Key Lemma: If $R_0, R_1, \ldots, R_{n+2k-1}$ are the received packets, then $R_i E(i) = P(i)E(i)$ for $i = 0, 1, \ldots, n + 2k - 1$.
  - This is a system of $n + 2k$ linear equations in the $n + 2k$ unknowns $a_0, a_1, \ldots, a_{k-1}, b_0, b_1, \ldots, b_{n+k-1}$.

Countability

A set S is countable if there is an injection $S \to \mathbb{N}$.

Countability

A set S is countable if there is an injection $S \to \mathbb{N}$.

- Countable sets: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{N} \times \mathbb{N}$, $\mathbb{Q}$.

Countability

A set S is countable if there is an injection $S \to \mathbb{N}$.

- Countable sets: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{N} \times \mathbb{N}$, $\mathbb{Q}$. All finite-length strings from a countably infinite alphabet.

Countability

A set S is countable if there is an injection $S \to \mathbb{N}$.

- Countable sets: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{N} \times \mathbb{N}$, $\mathbb{Q}$. All finite-length strings from a countably infinite alphabet.
- How to show a set is countable: put its elements into a list!

Countability

A set S is countable if there is an injection $S \to \mathbb{N}$.

- Countable sets: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{N} \times \mathbb{N}$, $\mathbb{Q}$. All finite-length strings from a countably infinite alphabet.
- How to show a set is countable: put its elements into a list!

A set S is uncountable if it is not countable.

Countability

A set S is countable if there is an injection $S \to \mathbb{N}$.

- Countable sets: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{N} \times \mathbb{N}$, $\mathbb{Q}$. All finite-length strings from a countably infinite alphabet.
- How to show a set is countable: put its elements into a list!

A set S is uncountable if it is not countable.

- Examples: $\mathbb{R}$, infinite-length bit strings.

Countability

A set S is countable if there is an injection $S \to \mathbb{N}$.

- Countable sets: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{N} \times \mathbb{N}$, $\mathbb{Q}$. All finite-length strings from a countably infinite alphabet.
- How to show a set is countable: put its elements into a list!

A set S is uncountable if it is not countable.

- Examples: $\mathbb{R}$, infinite-length bit strings.
- How to show a set is uncountable: Cantor diagonalization.

Countability

A set S is countable if there is an injection $S \to \mathbb{N}$.

- Countable sets: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{N} \times \mathbb{N}$, $\mathbb{Q}$. All finite-length strings from a countably infinite alphabet.
- How to show a set is countable: put its elements into a list!

A set S is uncountable if it is not countable.

- Examples: $\mathbb{R}$, infinite-length bit strings.
- How to show a set is uncountable: Cantor diagonalization.

$$\begin{array}{cccc} 0 & 0 & 0 & \cdots \\ 0 & 1 & 1 & \cdots \\ 1 & 1 & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{array}$$

What infinite-length bit string is not in the list?

Countability

A set S is countable if there is an injection $S \to \mathbb{N}$.

- Countable sets: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{N} \times \mathbb{N}$, $\mathbb{Q}$. All finite-length strings from a countably infinite alphabet.
- How to show a set is countable: put its elements into a list!

A set S is uncountable if it is not countable.

- Examples: $\mathbb{R}$, infinite-length bit strings.
- How to show a set is uncountable: Cantor diagonalization.

$$\begin{matrix} 0 & 0 & 0 & \cdots \\ 0 & 1 & 1 & \cdots \\ 1 & 1 & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{matrix}$$

What infinite-length bit string is not in the list? $101\ldots$

Countability

A set S is countable if there is an injection $S \to \mathbb{N}$.

- Countable sets: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{N} \times \mathbb{N}$, $\mathbb{Q}$. All finite-length strings from a countably infinite alphabet.
- How to show a set is countable: put its elements into a list!

A set S is uncountable if it is not countable.

- Examples: $\mathbb{R}$, infinite-length bit strings.
- How to show a set is uncountable: Cantor diagonalization.

$$\begin{matrix} 0 & 0 & 0 & \cdots \\ 0 & 1 & 1 & \cdots \\ 1 & 1 & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{matrix}$$

  What infinite-length bit string is not in the list? 101...

- Alternatively, find an *injection* from an uncountable set (such as $\mathbb{R}$) into the set.

Computability

Not all functions can be computed.

Computability

Not all functions can be computed.

- Link to countability: computer programs are countably infinite, but functions $\mathbb{N} \to \{0, 1\}$ are uncountable.

Computability

Not all functions can be computed.

- Link to countability: computer programs are countably infinite, but functions $\mathbb{N} \to \{0,1\}$ are uncountable.
- `TestHalt` takes two arguments, a program P and an input x, and returns 1 iff $P(x)$ halts; 0 otherwise.

Computability

Not all functions can be computed.

- Link to countability: computer programs are countably infinite, but functions $\mathbb{N} \to \{0,1\}$ are uncountable.
- `TestHalt` takes two arguments, a program P and an input x, and returns 1 iff $P(x)$ halts; 0 otherwise.
- Then, $\texttt{TestHalt} : \mathbb{N} \times \mathbb{N} \to \{0,1\}$ is an *explicit* function which cannot be computed.

Computability

Not all functions can be computed.

- Link to countability: computer programs are countably infinite, but functions $\mathbb{N} \to \{0,1\}$ are uncountable.
- `TestHalt` takes two arguments, a program P and an input x, and returns 1 iff $P(x)$ halts; 0 otherwise.
- Then, $\texttt{TestHalt} : \mathbb{N} \times \mathbb{N} \to \{0,1\}$ is an *explicit* function which cannot be computed.

Reductions:

Computability

Not all functions can be computed.

- Link to countability: computer programs are countably infinite, but functions $\mathbb{N} \to \{0,1\}$ are uncountable.
- `TestHalt` takes two arguments, a program P and an input x, and returns 1 iff $P(x)$ halts; 0 otherwise.
- Then, $\texttt{TestHalt} : \mathbb{N} \times \mathbb{N} \to \{0,1\}$ is an *explicit* function which cannot be computed.

Reductions:

- To show that P is uncomputable, *assume P exists*.

Computability

Not all functions can be computed.

- Link to countability: computer programs are countably infinite, but functions $\mathbb{N} \to \{0,1\}$ are uncountable.
- `TestHalt` takes two arguments, a program P and an input x, and returns 1 iff $P(x)$ halts; 0 otherwise.
- Then, $\texttt{TestHalt} : \mathbb{N} \times \mathbb{N} \to \{0,1\}$ is an *explicit* function which cannot be computed.

Reductions:

- To show that P is uncomputable, *assume P exists*. But, *do not assume how P is implemented*.

Computability

Not all functions can be computed.

- Link to countability: computer programs are countably infinite, but functions $\mathbb{N} \to \{0,1\}$ are uncountable.
- `TestHalt` takes two arguments, a program P and an input x, and returns 1 iff $P(x)$ halts; 0 otherwise.
- Then, $\texttt{TestHalt} : \mathbb{N} \times \mathbb{N} \to \{0,1\}$ is an *explicit* function which cannot be computed.

Reductions:

- To show that P is uncomputable, *assume P exists*. But, *do not assume how P is implemented*.
- Example: To show that `TestHalt` is uncomputable, *do not assume that `TestHalt` must actually run $P(x)$*.

Computability

Not all functions can be computed.

- Link to countability: computer programs are countably infinite, but functions $\mathbb{N} \to \{0,1\}$ are uncountable.
- `TestHalt` takes two arguments, a program P and an input x, and returns 1 iff $P(x)$ halts; 0 otherwise.
- Then, $\texttt{TestHalt} : \mathbb{N} \times \mathbb{N} \to \{0,1\}$ is an *explicit* function which cannot be computed.

Reductions:

- To show that P is uncomputable, *assume P exists*. But, *do not assume how P is implemented*.
- Example: To show that `TestHalt` is uncomputable, *do not assume that `TestHalt` must actually run $P(x)$*.
- Then, use the power of P to define `TestHalt`, which you know is impossible.

Computability

Not all functions can be computed.

- Link to countability: computer programs are countably infinite, but functions $\mathbb{N} \to \{0,1\}$ are uncountable.
- `TestHalt` takes two arguments, a program P and an input x, and returns 1 iff $P(x)$ halts; 0 otherwise.
- Then, $\texttt{TestHalt} : \mathbb{N} \times \mathbb{N} \to \{0,1\}$ is an *explicit* function which cannot be computed.

Reductions:

- To show that P is uncomputable, *assume P exists*. But, *do not assume how P is implemented*.
- Example: To show that `TestHalt` is uncomputable, *do not assume that `TestHalt` must actually run $P(x)$*.
- Then, use the power of P to define `TestHalt`, which you know is impossible.
- Therefore, P cannot exist.

Counting

- Number of subsets of an n-element set?

Counting

- Number of subsets of an n-element set? 2^n.

Counting

- Number of subsets of an n-element set? 2^n. Same as the number of length-n bit strings.

Counting

- Number of subsets of an n-element set? 2^n. Same as the number of length-n bit strings.
- Number of ways to rearrange $\{1, \ldots, n\}$?

Counting

- Number of subsets of an n-element set? 2^n. Same as the number of length-n bit strings.
- Number of ways to rearrange $\{1, \ldots, n\}$? $n! = \prod_{i=1}^{n} i$.

Counting

- Number of subsets of an n-element set? 2^n. Same as the number of length-n bit strings.
- Number of ways to rearrange $\{1, \ldots, n\}$? $n! = \prod_{i=1}^{n} i$.
- Number of k-element subsets of an n-element set?

Counting

- Number of subsets of an n-element set? 2^n. Same as the number of length-n bit strings.
- Number of ways to rearrange $\{1, \ldots, n\}$? $n! = \prod_{i=1}^{n} i$.
- Number of k-element subsets of an n-element set? $\binom{n}{k} = \binom{n}{n-k} = n!/[k!(n-k)!]$.

Counting

- Number of subsets of an n-element set? 2^n. Same as the number of length-n bit strings.
- Number of ways to rearrange $\{1, \dots, n\}$? $n! = \prod_{i=1}^{n} i$.
- Number of k-element subsets of an n-element set? $\binom{n}{k} = \binom{n}{n-k} = n!/[k!(n-k)!]$.
- Number of solutions to $x_1 + \cdots + x_n = k$ in the natural numbers? Throw k unlabeled balls into n labeled bins.

Counting

- Number of subsets of an n-element set? 2^n. Same as the number of length-n bit strings.
- Number of ways to rearrange $\{1,\ldots,n\}$? $n! = \prod_{i=1}^n i$.
- Number of k-element subsets of an n-element set? $\binom{n}{k} = \binom{n}{n-k} = n!/[k!(n-k)!]$.
- Number of solutions to $x_1 + \cdots + x_n = k$ in the natural numbers? Throw k unlabeled balls into n labeled bins.
- Stars and bars: the n bins can be represented as $n-1$ "dividers" or bars.

Counting

- Number of subsets of an n-element set? 2^n. Same as the number of length-n bit strings.
- Number of ways to rearrange $\{1, \ldots, n\}$? $n! = \prod_{i=1}^{n} i$.
- Number of k-element subsets of an n-element set? $\binom{n}{k} = \binom{n}{n-k} = n!/[k!(n-k)!]$.
- Number of solutions to $x_1 + \cdots + x_n = k$ in the natural numbers? Throw k unlabeled balls into n labeled bins.
- Stars and bars: the n bins can be represented as $n-1$ "dividers" or bars. Answer: $\binom{n+k-1}{k}$.

Counting

- Number of subsets of an n-element set? 2^n. Same as the number of length-n bit strings.
- Number of ways to rearrange $\{1, \ldots, n\}$? $n! = \prod_{i=1}^{n} i$.
- Number of k-element subsets of an n-element set? $\binom{n}{k} = \binom{n}{n-k} = n!/[k!(n-k)!]$.
- Number of solutions to $x_1 + \cdots + x_n = k$ in the natural numbers? Throw k unlabeled balls into n labeled bins.
- Stars and bars: the n bins can be represented as $n-1$ "dividers" or bars. Answer: $\binom{n+k-1}{k}$.
- If A and B are disjoint, what is $|A \cup B|$?

Counting

- Number of subsets of an n-element set? 2^n. Same as the number of length-n bit strings.
- Number of ways to rearrange $\{1, \ldots, n\}$? $n! = \prod_{i=1}^{n} i$.
- Number of k-element subsets of an n-element set? $\binom{n}{k} = \binom{n}{n-k} = n!/[k!(n-k)!]$.
- Number of solutions to $x_1 + \cdots + x_n = k$ in the natural numbers? Throw k unlabeled balls into n labeled bins.
- Stars and bars: the n bins can be represented as $n-1$ "dividers" or bars. Answer: $\binom{n+k-1}{k}$.
- If A and B are disjoint, what is $|A \cup B|$? Answer: $|A| + |B|$.

Counting

- Number of subsets of an n-element set? 2^n. Same as the number of length-n bit strings.
- Number of ways to rearrange $\{1, \ldots, n\}$? $n! = \prod_{i=1}^{n} i$.
- Number of k-element subsets of an n-element set? $\binom{n}{k} = \binom{n}{n-k} = n!/[k!(n-k)!]$.
- Number of solutions to $x_1 + \cdots + x_n = k$ in the natural numbers? Throw k unlabeled balls into n labeled bins.
- Stars and bars: the n bins can be represented as $n-1$ "dividers" or bars. Answer: $\binom{n+k-1}{k}$.
- If A and B are disjoint, what is $|A \cup B|$? Answer: $|A| + |B|$.
- What if A and B are not disjoint?

Counting

- Number of subsets of an n-element set? 2^n. Same as the number of length-n bit strings.
- Number of ways to rearrange $\{1, \ldots, n\}$? $n! = \prod_{i=1}^{n} i$.
- Number of k-element subsets of an n-element set? $\binom{n}{k} = \binom{n}{n-k} = n!/[k!(n-k)!]$.
- Number of solutions to $x_1 + \cdots + x_n = k$ in the natural numbers? Throw k unlabeled balls into n labeled bins.
- Stars and bars: the n bins can be represented as $n-1$ "dividers" or bars. Answer: $\binom{n+k-1}{k}$.
- If A and B are disjoint, what is $|A \cup B|$? Answer: $|A| + |B|$.
- What if A and B are not disjoint? Inclusion-Exclusion: $|A| + |B| - |A \cap B|$.

Counting

- Number of subsets of an n-element set? 2^n. Same as the number of length-n bit strings.
- Number of ways to rearrange $\{1, \ldots, n\}$? $n! = \prod_{i=1}^{n} i$.
- Number of k-element subsets of an n-element set? $\binom{n}{k} = \binom{n}{n-k} = n!/[k!(n-k)!]$.
- Number of solutions to $x_1 + \cdots + x_n = k$ in the natural numbers? Throw k unlabeled balls into n labeled bins.
- Stars and bars: the n bins can be represented as $n-1$ "dividers" or bars. Answer: $\binom{n+k-1}{k}$.
- If A and B are disjoint, what is $|A \cup B|$? Answer: $|A| + |B|$.
- What if A and B are not disjoint? Inclusion-Exclusion: $|A| + |B| - |A \cap B|$.
- Binomial Theorem: $(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}$.

Combinatorial Proofs

Prove an equation involving combinatorial terms by showing that both sides count the same objects.

Combinatorial Proofs

Prove an equation involving combinatorial terms by showing that both sides count the same objects.

Midterm: For $k \geq n$,
$\binom{k-1}{n-1} = |\{(x_1, \ldots, x_n) \in \mathbb{N}^+ : x_1 + \cdots + x_n = k\}|$.

Combinatorial Proofs

Prove an equation involving combinatorial terms by showing that both sides count the same objects.

Midterm: For $k \geq n$,
$\binom{k-1}{n-1} = |\{(x_1, \ldots, x_n) \in \mathbb{N}^+ : x_1 + \cdots + x_n = k\}|$.

- On the RHS, since $x_1 + \cdots + x_n = k$, think of splitting up k things into n chunks of size ≥ 1 each.

Combinatorial Proofs

Prove an equation involving combinatorial terms by showing that both sides count the same objects.

Midterm: For $k \geq n$,
$\binom{k-1}{n-1} = |\{(x_1, \ldots, x_n) \in \mathbb{N}^+ : x_1 + \cdots + x_n = k\}|$.

- On the RHS, since $x_1 + \cdots + x_n = k$, think of splitting up k things into n chunks of size ≥ 1 each.
- How many ways are there to create these partitions?

Combinatorial Proofs

Prove an equation involving combinatorial terms by showing that both sides count the same objects.

Midterm: For $k \geq n$,
$\binom{k-1}{n-1} = |\{(x_1, \ldots, x_n) \in \mathbb{N}^+ : x_1 + \cdots + x_n = k\}|$.

- On the RHS, since $x_1 + \cdots + x_n = k$, think of splitting up k things into n chunks of size ≥ 1 each.
- How many ways are there to create these partitions?
- This is like placing $n-1$ dividers among the k objects...

Combinatorial Proofs

Prove an equation involving combinatorial terms by showing that both sides count the same objects.

Midterm: For $k \geq n$,
$\binom{k-1}{n-1} = |\{(x_1, \ldots, x_n) \in \mathbb{N}^+ : x_1 + \cdots + x_n = k\}|$.

- On the RHS, since $x_1 + \cdots + x_n = k$, think of splitting up k things into n chunks of size ≥ 1 each.
- How many ways are there to create these partitions?
- This is like placing $n-1$ dividers among the k objects...
- So it equals $\binom{k-1}{n-1}$.

Tomorrow

Review of probability.