Discrete Mathematics(H)
Southern University of Science and Technology
Mengxuan Wu
12212006

Final Review

Mengxuan Wu

1 Logic

1.1 Propositional Logic

1.1.1 Propositions

A proposition is a **declarative** sentence that is **either true or false**, but not both. For example, "*SUSTech is in Shenzhen*" is a proposition, while "*No parking on campus*" is not a proposition.

1.1.2 Logical Connectives

There are six logical connectives in propositional logic, which are **negation**($\neg$), **conjunction**($\wedge$), **disjunction**($\vee$), **exclusive or**($\oplus$), **implication**($\rightarrow$), and **biconditional**($\leftrightarrow$).

For implication $p \rightarrow q$, we call p the **hypothesis** and q the **conclusion**.

The **converse** of $p \rightarrow q$ is $q \rightarrow p$. The **inverse** of $p \rightarrow q$ is $\neg p \rightarrow \neg q$. The **contrapositive** of $p \rightarrow q$ is $\neg q \rightarrow \neg p$.

1.1.3 Tautologies and Contradictions

A **tautology** is a proposition that is always true, regardless of the truth values of its individual components. A **contradiction** is a proposition that is always false. A **contingency** is a proposition that is neither a tautology nor a contradiction.

1.1.4 Logical Equivalences

Two propositions are **logically equivalent** if they have the same truth values for all possible combinations of truth values of their component propositions.

The propositions p and q are logically equivalent if and only if $p \leftrightarrow q$ is a tautology, denoted by $p \equiv q$ or $p \Leftrightarrow q$.

1.1.5 Important Logical Equivalences

- **Identity laws**

$$p \wedge T \equiv p$$
$$p \vee F \equiv p$$

- **Domination laws**

$$p \vee T \equiv T$$
$$p \wedge F \equiv F$$

- **Idempotent laws**

$$p \vee p \equiv p$$
$$p \wedge p \equiv p$$

- **Double negation laws**

$$\neg(\neg p) \equiv p$$

- **Commutative laws**

$$p \vee q \equiv q \vee p$$
$$p \wedge q \equiv q \wedge p$$

- **Associative laws**

$$(p \vee q) \vee r \equiv p \vee (q \vee r)$$
$$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$$

- **Distributive laws**

$$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$$
$$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$$

- **De Morgan's laws**

$$\neg(p \wedge q) \equiv \neg p \vee \neg q$$
$$\neg(p \vee q) \equiv \neg p \wedge \neg q$$

- **Absorption laws**

$$p \vee (p \wedge q) \equiv p$$
$$p \wedge (p \vee q) \equiv p$$

- **Negation laws**

$$p \vee \neg p \equiv T$$
$$p \wedge \neg p \equiv F$$

- **Useful law**

$$p \to q \equiv \neg p \vee q$$

1.2 Predicate Logic

1.2.1 Predicates and Quantifiers

A **constant** models a specific object. A **variable** represents objects of specific type. A **predicate** represents properties or relations among objects.

However, a predicate is not a proposition, because it is not a declarative sentence. It becomes a proposition when the variable is assigned a value. Additionally, the universal quantification and existential quantification of a predicate is a proposition. For example, Prime(x) **is not** a proposition, while Prime(3) and $\exists x$Prime(x) **are** propositions.

The **universe(domain)** D of a predicate is the set of all objects that can be substituted for the variables in a predicate. The **truth set** of a predicate $P(x)$ is the set of objects in the universe that can be substituted for x in $P(x)$ to make the resulting proposition true.

The truth values of $\forall x P(x)$ and $\exists x P(x)$ depend on both the **universe** and the **predicate** $P(x)$.

1.2.2 Precedence of Quantifiers

The quantifiers $\forall$ and $\exists$ have higher precedence than all the logical operators. For example, $\forall x P(x) \wedge Q(x)$ means $(\forall x P(x)) \wedge Q(x)$, not $\forall x(P(x) \wedge Q(x))$.

1.2.3 Translation with Quantifiers

Universal quantification

Sentence: All SUSTech students are smart.

Universe: all students

Translation: $\forall x(\text{At}(x, \text{SUSTech}) \rightarrow \text{Smart}(x))$

Typical error: $\forall x(\text{At}(x, \text{SUSTech}) \wedge \text{Smart}(x))$, which means all students are at SUSTech and smart.

Existential quantification

Sentence: Some SUSTech students are smart.

Universe: all students

Translation: $\exists x(\text{At}(x, \text{SUSTech}) \wedge \text{Smart}(x))$

Typical error: $\exists x(\text{At}(x, \text{SUSTech}) \rightarrow \text{Smart}(x))$, this is true if there is anyone who is not at SUSTech.

1.2.4 Nested Quantifiers

The order of nested quantifiers is important. For example, let $L(x, y)$ denotes "x loves y", then $\forall x \exists y L(x, y)$ means "Everyone loves someone", while $\exists y \forall x L(x, y)$ means "There is someone whom everyone loves". In general, $\exists x \forall y P(x, y) \rightarrow \forall y \exists x P(x, y)$ is a tautology, but $\forall y \exists x P(x, y) \rightarrow \exists x \forall y P(x, y)$ is not always true.

However, the order of nested quantifiers does not matter if quantifiers are of the same type.

1.2.5 Negating Quantifiers

- $\neg \forall x P(x) \equiv \exists x \neg P(x)$
- $\neg \exists x P(x) \equiv \forall x \neg P(x)$

- $\neg\forall x\exists y P(x,y) \equiv \exists x\forall y\neg P(x,y)$

2 Mathematical Proofs

2.1 Theorems and Proofs

2.1.1 Definitions

An **axiom** or **postulate** is a statement or proposition that is regarded as being established, accepted, or self-evidently true. A **theorem** is a statement or proposition that can be proved to be true. A **lemma** is a statement that can be proved to be true, and is used in proving a theorem.

In **formal proofs**, steps follow logically from the set of premises, axioms, lemmas, and other previously proved theorems.

2.1.2 Rules of inference

- **Modus Ponens**: $((p \to q) \wedge p) \to q$
- **Modus Tollens**: $((p \to q) \wedge \neg q) \to \neg p$
- **Hypothetical Syllogism**: $((p \to q) \wedge (q \to r)) \to (p \to r)$
- **Disjunctive Syllogism**: $((p \vee q) \wedge \neg p) \to q$
- **Addition**: $p \to (p \vee q)$
- **Simplification**: $(p \wedge q) \to p$
- **Conjunction**: $((p) \wedge (q)) \to (p \wedge q)$
- **Resolution**: $((p \vee q) \wedge (\neg p \vee r)) \to (q \vee r)$
- **Universal instantiation**: $\forall x P(x) \to P(c)$
- **Universal generalization**: $P(c)$ for an arbitrary $c \to \forall x P(x)$
- **Existential instantiation**: $\exists x P(x) \to P(c)$ for some element c
- **Existential generalization**: $P(c) \to \exists x P(x)$

2.1.3 Methods of Proof

To proof $p \to q$:

- **Direct proof:** Show that if p is true then q follows.
- **Proof by contrapositive:** Show that $\neg q \to \neg p$.
- **Proof by contradiction:** Show that $p \wedge \neg q$ contradicts the assumptions.
- **Proof by cases:** Give proofs for all possible cases.
- **Proof of equivalence** $p \leftrightarrow q$: Prove $p \to q$ and $q \to p$.

3 Sets and Functions

3.1 Sets

3.1.1 Definitions

A **set** is an unordered collection of objects, called **elements** or **members** of the set. We can represent a set by listing its elements between braces, or defining a property that its elements satisfy.

3.1.2 Important Sets

- $\mathbb{N}$ is the set of natural numbers.
- $\mathbb{Z}$ is the set of integers. $\mathbb{Z}^+$ is the set of positive integers.
- $\mathbb{Q}$ is the set of rational numbers.
- $\mathbb{R}$ is the set of real numbers.
- $\mathbb{C}$ is the set of complex numbers.
- $\mathbb{U}$ is the set of all objects under consideration.
- $\emptyset$ is the empty set. *Note:* $\emptyset \neq \{\emptyset\}$
- $P(S)$ is the power set of S, which is the set of all subsets of S.
- Disjoint sets A and B are sets that have no elements in common, i.e., $A \cap B = \emptyset$.

3.1.3 Set Operations

- **Union:** $A \cup B = \{x | x \in A \text{ or } x \in B\}$
- **Intersection:** $A \cap B = \{x | x \in A \text{ and } x \in B\}$
- **Difference:** $A - B = \{x | x \in A \text{ and } x \notin B\}$
- **Complement:** $\overline{A} = \{x | x \in U \text{ and } x \notin A\}$
- **Cartesian product:** $A \times B = \{(a, b) | a \in A \text{ and } b \in B\}$

3.1.4 Cardinality

The **cardinality** of a set S, denoted by $|S|$, is the number of distinct elements in the set. The sets A and B have the same cardinality if there is a one-to-one correspondence between them. If there is a one-to-one function from A to B, the cardinality of A is less than or equal to the cardinality of B, denoted by $|A| \leq |B|$.

A set that is either finite or has the same cardinality as the set of positive integers $\mathbb{Z}^+$ is called **countable**. A set that is not countable is called **uncountable**.

Here are some examples of countable and uncountable sets:

- **Countable Sets:** $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{Z}^+$, the set of finite strings S over a finite alphabet A
- **Uncountable Sets:** $\mathbb{R}$, $P(\mathbb{N})$

The subset of a countable set is still countable.

To prove a set is **countable**, we can use Schröder-Bernstein theorem. If there are one-to-one functions $f : A \to B$ and $g : B \to A$, then there is a one-to-one correspondence between A and B, and $|A| = |B|$.

To prove a set is **uncountable**, we can use Cantor's diagonalization method. Assume that S is countable, then we can list all elements of S in a table. Then we can construct a new element that is not in the table, which contradicts the assumption that S is countable.

For power set, we have the following formula:

$$|P(S)| = 2^{|S|}$$

For union and intersection, we have the following formulas:

$$|A \cup B| = |A| + |B| - |A \cap B|$$

For Cartesian product, we have the following formula:

$$|A \times B| = |A| \times |B|$$

Note: $|\emptyset| = 0$ *and* $|\{\emptyset\}| = 1$

3.1.5 Computable vs. Uncomputable

We say a function is **computable** if there is an algorithm that can compute the function's value for any input in a finite amount of time.

There are functions that are not computable. This is true because the set of computer programs is countable, while the set of functions from $\mathbb{N}$ to the set $\{0, 1, 2, ..., 9\}$ is uncountable. (Cantor's diagonalization method)

3.1.6 Cantor's Theorem

For any set S, $|S| < |P(S)|$.

This is obviously true for finite sets, because $|P(S)| = 2^{|S|}$. (Note that $|\emptyset| = 0, |P(\emptyset)| = 1$)

For infinite sets, we can prove this by contradiction. Assume that $|S| = |P(S)|$, then there is a one-to-one correspondence between S and $P(S)$. Let f be a function from S to $P(S)$, then we can construct a set $T = \{s \in S | x \notin f(s)\}$. Since f is a one-to-one correspondence, there must be an element $s_0 \in S$ such that $f(s_0) = T$. However, $s_0 \in T$ implies $s_0 \notin T$, which is a contradiction.

To build a one-to-one function from $P(S)$ to S is trivial, because we can just map each subset of S to its smallest element.

Hence, we know that $|S| \neq |P(S)|$ and $|S| \leq |P(S)|$. Therefore, $|S| < |P(S)|$.

3.1.7 Set Identities

- **Identity laws**

$$A \cup \emptyset = A$$
$$A \cap U = A$$

- **Domination laws**

$$A \cup U = U$$
$$A \cap \emptyset = \emptyset$$

- **Idempotent laws**

$$A \cup A = A$$
$$A \cap A = A$$

- **Complementation laws**

$$\overline{\overline{A}} = A$$

- **Commutative laws**

$$A \cup B = B \cup A$$
$$A \cap B = B \cap A$$

- **Associative laws**

$$(A \cup B) \cup C = A \cup (B \cup C)$$
$$(A \cap B) \cap C = A \cap (B \cap C)$$

- **Distributive laws**

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$
$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

- **De Morgan's laws**

$$\overline{A \cup B} = \overline{A} \cap \overline{B}$$
$$\overline{A \cap B} = \overline{A} \cup \overline{B}$$

- **Absorption laws**

$$A \cup (A \cap B) = A$$
$$A \cap (A \cup B) = A$$

- **Complement laws**

$$A \cup \overline{A} = U$$
$$A \cap \overline{A} = \emptyset$$

3.2 Tuples

An **ordered n-tuple** is a sequence of n elements, where n is a positive integer.

3.3 Functions

3.3.1 Definitions

Let A and B be sets. A **function** f from A to B, denoted by $f : A \to B$, is an assignment of exactly one element of B to each element of A.

We represent a function by a formula or explicitly state the assignments between elements of A and B. For example, let $A = \{1, 2, 3\}$ and $B = \{a, b, c, d\}$, then $f : A \to B$ can be represented by $1 \mapsto a, 2 \mapsto b, 3 \mapsto c$.

Let $f : A \to B$ be a function. We say that A is the **domain** of f and B is the **codomain** of f. If $f(a) = b$, b is the **image** of a under f, and a is a **preimage** of b. The **range of** f is the set of all images of elements of A, denoted by $f(A)$.

3.3.2 Injective, Surjective, and Bijective Functions

A function $f : A \to B$ is **injective (one-to-one)** if and only if $f(a_1) = f(a_2)$ implies $a_1 = a_2$ for all $a_1, a_2 \in A$. A function $f : A \to B$ is **surjective (onto)** if and only if $f(A) = B$. A function $f : A \to B$ is **bijective (one-to-one correspond)** if and only if f is both injective and surjective.

The inverse of a function, denoted by $f^{-1}(x)$, is only defined for bijective functions.

3.3.3 Sequences

A **sequence** is a function from a subset of integers (typically $\{0, 1, 2, ...\}$ or $\{1, 2, 3, ...\}$) to a set S. We use the notation a_n to denote the image of n under the function, and we call a_n the nth term of the sequence.

4 Complexity of Algorithms

4.1 Big-O Notation

We say that $f(n) = O(g(n))$ (reads as "$f(n)$ is big-O of $g(n)$") if there are positive constants c and n_0 such that $|f(n)| \leq |c \cdot g(n)|$ for all $n \geq n_0$.

Important big-O estimation:

- $n! = O(n^n)$
- $\log n! = O(n \log n)$ (Actually we can prove $\log n! < n \log n < 2 \log n!$)
- $\log_a n = O(n)$ for any $a \geq 2$
- $n^k = O(a^n)$ for any k and $a > 1$

Similarly, we can define big-Ω and big-Θ.

4.2 Algorithms

An **algorithm** is a finite set of precise instructions for performing a computation or for solving a problem. A **computational problem** is a specification of the desired input-output relationship. An **instance** of a computational problem is a specific input. A **correct algorithm** halts with the correct output for every instance of the problem, and we can say that the algorithm **solves** the problem.

4.2.1 Time and Space Complexity

The **time complexity** of an algorithm is the number of machine operations it performs on an instance. The **space complexity** of an algorithm is the number of cells of memory it uses on an instance.

4.2.2 Input Size

The input size is the number of bits needed to represent the input. For an integer n, the input size actually is $\lceil \log_2(n+1) \rceil$ (or just $\log_2 n$). Therefore, an algorithm that runs in $\Theta(n)$ seems to be linear and efficient, but it is actually exponential ($\Theta(n) = \Theta(2^{size(n)})$).

We say two positive functions $f(n)$ and $g(n)$ are of the same type if and only If

$$c_1 g(n^{a_1})^{b_1} \leq f(n) \leq c_2 g(n^{a_2})^{b_2}$$

for all large n and some positive constants $c_1, c_2, a_1, a_2, b_1, b_2$.

Therefore, all polynomial functions are of the same type, and all exponential functions are of the same type. But polynomial functions are not of the same type as exponential functions.

4.2.3 Decision Problems and Optimization Problems

A **decision problem** is a question that has two possible answers: yes or no. An **optimization problem** is a question that requires an answer that is an optimal configuration.

If L is a decision problem and x is the input, we often write $x \in L$ to mean that the answer to the decision problem is yes, and $x \notin L$ to mean that the answer is no.

An optimization problem usually has a corresponding decision problem.

4.2.4 Complexity Classes

We divide the set of all decision problems into three complexity classes.

A problem is **solvable (tractable)** in **polynomial time** if there is an algorithm that solves the problem and the number of steps required by the algorithm on any instance of size n is $O(n^k)$ for some constant k.

The class P is the set of all decision problems that are solvable in polynomial time. The class NP is the set of all decision problems for which there exists a certificate for each yes-input that can be verified in polynomial time.

4.2.5 NP Completeness

Reduction is a relationship between two problems. We say Q can be reduced to Q' if every instance of Q can be transformed into an instance of Q' such that the answer to the transformed instance is yes if and only if the answer to the original instance is yes.

A polynomial-time reduction from Q to Q' is a reduction that can be performed in polynomial time, denoted by $Q \leq_p Q'$. Intuitively, this means Q is no harder than Q'.

A problem Q is **NP-complete** if and only if

- $Q \in NP$
- Every problem $L \in NP$, $L \leq_p Q$

Therefore, if we can find a polynomial-time algorithm for an NP-complete problem, then we can solve all NP problems in polynomial time.

5 Number Theory

5.1 Divisibility and Modular Arithmetic

5.1.1 Divisibility

Let a, b, c be integers. Then the following holds:

- If $a|b$ and $a|c$, then $a|(b+c)$ and $a|(b-c)$.
- If $a|b$, then $a|bc$.
- If $a|b$ and $b|c$, then $a|c$.
- Corollary: If a, b, c are integers, where $a \neq 0$, such that $a|b$ and $a|c$, then $a|(mb+nc)$ for any integers m and n.

5.1.2 Modular Arithmetic

Let a, b, n be integers, where $n > 0$. We say that a is **congruent to** b **modulo** n, denoted by $a \equiv b \pmod{n}$, if and only if $n|(a-b)$. This is called **congruence** and n is called the **modulus**.

The following properties hold:

- If $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then $a + c \equiv b + d \pmod{n}$.
- If $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then $ac \equiv bd \pmod{n}$.
- Corollary: $(a+b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m$, and $(a \cdot b) \bmod m = ((a \bmod m) \cdot (b \bmod m)) \bmod m$.

5.2 Prime

An integer $p > 1$ is **prime** if and only if its only positive divisors are 1 and p. An integer $n > 1$ is **composite** if and only if it is not prime.

The **fundamental theorem of arithmetic** states that every integer greater than 1 is either prime or can be written as a unique product of prime numbers.

GCD of two integers a and b, denoted by $\gcd(a, b)$, is the largest integer that divides both a and b. If we factorize a and b into prime numbers, then $\gcd(a, b) = p_1^{\min(e_1, f_1)} \cdot p_2^{\min(e_2, f_2)} \cdot \ldots \cdot p_k^{\min(e_k, f_k)}$, where e_i and f_i are the exponents of p_i in the factorization of a and b. Two integers are **relatively prime** if and only if their GCD is 1.

Similarly, we can define **LCM** of two integers a and b, denoted by $\text{lcm}(a, b)$, is the smallest positive integer that is divisible by both a and b.

5.3 Eculidean Algorithm

The **Eculidean algorithm** is an efficient method for computing the GCD of two integers. It can solve the problem in $O(\log n)$ time, where n is the smaller of the two integers.

The central idea is that if $a = bq + r$, then $\gcd(a, b) = \gcd(b, r)$. Here is the proof: If d is a common divisor of a and b, we can write $d \mid a$ and $d \mid b$. By the corollary, we know that $d \mid (a - bq)$, which implies $d \mid r$. Therefore, d is a common divisor of b and r.

For example, to compute $\gcd(287, 91)$:

$$\begin{aligned} 287 \bmod 91 &= 14 \\ 91 \bmod 14 &= 7 \\ 14 \bmod 7 &= 0 \end{aligned}$$

Therefore, $\gcd(287, 91) = 7$.

5.4 Bezout's Theorem

Let a and b be integers, not both zero. Then there exist integers x and y such that $\gcd(a, b) = ax + by$.

We can use the extended Eculidean algorithm to find x and y.

For example, since $\gcd(503, 286) = 1$, we can find x and y such that $503x + 286y = 1$.

$$\begin{aligned} 503 &= 1 \cdot 286 + 217 \\ 286 &= 1 \cdot 217 + 69 \\ 217 &= 3 \cdot 69 + 10 \\ 69 &= 6 \cdot 10 + 9 \\ 10 &= 1 \cdot 9 + 1 \\ 9 &= 9 \cdot 1 + 0 \end{aligned}$$

Eculidean algorithm

$$\begin{aligned} 1 &= 10 - 1 \cdot 9 \\ &= 10 - 1 \cdot (69 - 6 \cdot 10) = 7 \cdot 10 - 1 \cdot 69 \\ &= 7 \cdot (217 - 3 \cdot 69) - 1 \cdot 69 = 7 \cdot 217 - 22 \cdot 69 \\ &= 7 \cdot 217 - 22 \cdot (286 - 1 \cdot 217) = 29 \cdot 217 - 22 \cdot 286 \\ &= 29 \cdot (503 - 1 \cdot 286) - 22 \cdot 286 = 29 \cdot 503 - 51 \cdot 286 \end{aligned}$$

Extended Eculidean algorithm

Therefore, $x = 29$ and $y = -51$.

The corollaries of Bezout's theorem are:

- If $1 = \gcd(a, b)$ and $a|bc$, then $a|c$.
- If p is a prime and $p|a_1a_2...a_n$, then $p|a_i$ for some i.
- If $ac \equiv bc \pmod{n}$ and $\gcd(c, n) = 1$, then $a \equiv b \pmod{n}$.

The proof of the first corollary is as follows: Since $\gcd(a, b) = 1$, we can find x and y such that $ax + by = 1$. Then $c = cax + cby$. Since $a|bc$, we know that $a|cby$. Therefore, $a|(cax + cby) = c$.

5.5 Linear Congruence

A **linear congruence** is an equation of the form $ax \equiv b \pmod{n}$, where a, b, n are integers and $n > 0$.

5.5.1 Modular Inverse

Let a and n be integers, where $n > 0$. If there exists an integer $\overline{a}$ such that $a \cdot \overline{a} \equiv 1 \pmod{n}$, then $\overline{a}$ is called the **modular inverse** of a modulo n.

If a and n are relatively prime, then a has a modular inverse modulo n. Furthermore, the modular inverse of a modulo n is unique modulo n.

We can use the extended Eculidean algorithm to find the modular inverse of a modulo n.

5.5.2 Chinese Remainder Theorem

Let $n_1, n_2, ..., n_k$ be positive integers that are pairwise relatively prime, and let $a_1, a_2, ..., a_k$ be any integers. Then the system of linear congruences:

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\vdots \\ x &\equiv a_k \pmod{n_k} \end{aligned}$$

has a solution, and any two solutions are congruent modulo $n_1 n_2 ... n_k$.

Let $n = n_1 n_2 ... n_k$. Then the solution is $x = a_1 y_1 \frac{n}{n_1} + a_2 y_2 \frac{n}{n_2} + ... + a_k y_k \frac{n}{n_k}$, where y_i is the modular inverse of $\frac{n}{n_i}$ modulo n_i.

For example, to solve the system of linear congruences:

$$\begin{aligned} x &\equiv 2 \pmod{3} \\ x &\equiv 3 \pmod{5} \\ x &\equiv 2 \pmod{7} \end{aligned}$$

We can find $y_1 = 2$, $y_2 = 1$, and $y_3 = 1$. Therefore, $x = 2 \cdot 35 \cdot 2 + 3 \cdot 21 \cdot 1 + 2 \cdot 15 \cdot 1 = 233$ is a solution.

5.6 Fermat's Little Theorem

Let p be a prime and a be an integer such that $a \not\equiv 0 \pmod{p}$. Then $a^{p-1} \equiv 1 \pmod{p}$.

5.7 Euler's Theorem

Euler's function $\phi(n)$ is the number of positive integers less than or equal to n that are relatively prime to n. The function has the following properties:

- If p is prime, then $\phi(p) = p - 1$.
- If p is prime and $k \geq 1$, then $\phi(p^k) = p^k - p^{k-1}$.
- If m and n are relatively prime, then $\phi(mn) = \phi(m)\phi(n)$.

Euler's theorem states that if n is a positive integer and a is an integer such that a and n are relatively prime, then $a^{\phi(n)} \equiv 1 \pmod{n}$.

It is worth noting that $\phi(n)$ might not be the smallest positive integer k such that $a^k \equiv 1 \pmod{n}$.

5.8 Primitive Roots

A **primitive root** modulo a prime p is an integer g such that every nonzero integer modulo p is congruent to a power of g modulo p.

6 Groups, Rings, and Fields

6.1 Groups

A **group** is a set G together with a binary operation $*$ on G such that the following axioms hold:

- **Closure:** For all $a, b \in G$, $a * b \in G$.
- **Associativity:** For all $a, b, c \in G$, $(a * b) * c = a * (b * c)$.
- **Identity:** There exists a **unique** element $1_e \in G$ such that for all $a \in G$, $a * 1_e = a$.
- **Inverse:** For each $a \in G$, there exists an element $a^{-1} \in G$ such that $a * a^{-1} = 1_e$.

6.1.1 Permutation Groups

A permutation group is a group whose elements are permutations of a given set S and whose operation is composition of permutations in S. Let $s_n =< 1, 2, ..., n >$ denotes a sequence of n elements, and P_n denotes the set of all permutations of s_n. Then for any two elements π and ρ, $\pi \circ \rho$ is also a permutation of s_n.

For example, $s_3 =< 1, 2, 3 >$, and $P_3 = \{< 1, 2, 3 >, < 1, 3, 2 >, < 2, 1, 3 >, < 2, 3, 1 >, < 3, 1, 2 >, < 3, 2, 1 >\}$. If $\pi =< 3, 2, 1 >$ and $\rho =< 1, 3, 2 >$, then $\pi \circ \rho =< 2, 1, 3 >$. The operation is $\pi \circ \rho[i] = \rho[\pi[i]]$.

Therefore, $(P_n, \circ)$ is a group.

6.1.2 Abeilian Groups

A group G is **abeilian** if and only if for all $a, b \in G$, $a * b = b * a$.

6.2 Rings

If $(R, +)$ is an abeilian group, we define a second binary operation $*$ on R such that the following axioms hold:

- **Closure:** For all $a, b \in R$, $a * b \in R$.
- **Associativity:** For all $a, b, c \in R$, $(a * b) * c = a * (b * c)$.
- **Distributivity:** For all $a, b, c \in R$, $a*(b+c) = a*b+a*c$ and $(a+b)*c = a*c+b*c$.

6.2.1 Commutative Ring

A ring R is **commutative** if and only if for all $a, b \in R$, $a * b = b * a$.

6.2.2 Integral Domain

A commutative ring R is an **integral domain** if the following axiom holds:

- **Identity:** There exists a **unique** element $1_m \in R$ such that for all $a \in R$, $a * 1_m = 1_m * a = a$.
- **Nonzero product:** For all $a, b \in R$, if $a * b = 0$, then $a = 0$ or $b = 0$.

6.3 Fields

A commutative ring F is a **field** if the following axiom holds:

- **Inverse:** For each $a \in F$, there exists an element $a^{-1} \in F$ such that $a * a^{-1} = a^{-1} * a = 1_m$.

6.4 Other Facts

- Z_m, the set of integers modulo m, is a commutative ring.
- $(GL(n), \cdot)$ is a group but not an abeilian group. (The set of all invertible $n \times n$ matrices)
- $(\mathbb{M}_{n\times n}, +, \cdot)$ is a ring but not a commutative ring.
- $(Z_m, +_m, \cdot_m)$ is a commutative ring but not an integral domain.
- $(\mathbb{Z}, +, \cdot)$ is an integral domain but not a field.

7 Cryptography

7.1 Public Key Cryptography

7.1.1 RSA Cryptosystem

The RSA public key cryptosystem works as follows:

1. Choose two large primes p and q.
2. Compute $n = pq$ and $\phi(n) = (p-1)(q-1)$.
3. Choose e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$.
4. Compute d such that $ed \equiv 1 \pmod{\phi(n)}$.
5. Publish the public key (n, e) and keep the private key d secret.

To encrypt a message m, compute $c \equiv m^e \pmod{n}$. To decrypt a ciphertext c, compute $m \equiv c^d \pmod{n}$.

The correctness of the algorithm is as follows:

$$\begin{aligned} c^d &\equiv (m^e)^d && \pmod{n} \\ &\equiv m^{ed} && \pmod{n} \\ &\equiv m^{k\phi(n)+1} && \pmod{n} \\ &\equiv m \cdot (m^{\phi(n)})^k && \pmod{n} \\ &\equiv m \cdot 1^k && \pmod{n} \\ &\equiv m && \pmod{n} \end{aligned}$$

To leave a RSA signature, compute $s \equiv m^d \pmod{n}$. To verify a RSA signature, compute $m \equiv s^e \pmod{n}$.(d and e are interchangeable)

7.1.2 El Gamal Cryptosystem

The El Gamal public key cryptosystem works as follows:

1. Choose a large prime p and a primitive root g modulo p.
2. Choose x such that $1 < x < p - 2$.
3. Compute $y \equiv g^x \pmod{p}$.
4. Publish the public key (p, g, y) and keep the private key x secret.

To encrypt a message m, choose k such that $1 < k < p - 1$ and $\gcd(k, p - 1) = 1$, then compute $c_1 \equiv g^k \pmod{p}$ and $c_2 \equiv m \cdot y^k \pmod{p}$. To decrypt a ciphertext (c_1, c_2), compute $m \equiv c_2 \cdot (c_1^x)^{-1} \pmod{p}$.

The correctness of the algorithm is as follows:

$$\begin{aligned} c_2 \cdot (c_1^x)^{-1} &\equiv m \cdot y^k \cdot (g^{kx})^{-1} && \pmod{p} \\ &\equiv m \cdot g^{kx} \cdot g^{-kx} && \pmod{p} \\ &\equiv m && \pmod{p} \end{aligned}$$

7.2 Diffie-Hellman Key Exchange

The Diffie-Hellman key exchange works as follows:

1. Choose a large prime p and a primitive root g modulo p.
2. Alice chooses x such that $1 < x < p - 2$ and sends $y \equiv g^x \pmod{p}$ to Bob.
3. Bob chooses z such that $1 < z < p - 2$ and sends $w \equiv g^z \pmod{p}$ to Alice.
4. Alice computes $w^x \equiv (g^z)^x \equiv g^{zx} \pmod{p}$.
5. Bob computes $y^z \equiv (g^x)^z \equiv g^{xz} \pmod{p}$.
6. Now Alice and Bob share the secret key $g^{xz} \pmod{p}$.

8 Mathematical Induction

8.1 Proof by Smallest Counterexample

To prove a statement $P(n)$ for all $n \geq n_0$, we can use proof by smallest counterexample. We assume that $P(n)$ is false for some $n > 0$. Then there must be a smallest integer m such that $P(m)$ is false. Since $P(n_0)$ is true, $m > n_0$. Then we use the fact that $P(m')$ the for all $0 \leq m' < m$ is true to show that $P(m)$ is true, which is a contradiction.

8.2 Direct Proof

8.2.1 Weak Principle of Mathematical Induction

If the statement $P(b)$ is true, and the statement $P(n-1) \to P(n)$ is true for all integers $n > b$, then the statement $P(n)$ is true for all integers $n \geq b$.

We call $P(b)$ the **basis step (inductive hypothesis)** and $P(n-1) \to P(n)$ the **inductive step (inductive conclusion)**.

8.2.2 Strong Principle of Mathematical Induction

If the statement $P(b)$ is true, and the statement $P(b) \wedge P(b+1) \wedge \cdots \wedge P(n-1) \to P(n)$ is true for all integers $n > b$, then the statement $P(n)$ is true for all integers $n \geq b$.

We can see that the weak form is a special case of the strong form, and the strong form can be derived from the weak form.

8.3 Recursion

To prove a recursive algorithm is correct (for example the tower of Hanoi problem), we can use the mathematical induction. Also, the proof of runtime of recursive algorithms can also be proved by mathematical induction.

To proof:

$$M(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2M(n-1) + 1 & \text{if } n > 1 \end{cases}$$

Base case: When $n = 1$, $M(1) = 1$ is true.

Inductive hypothesis: Assume that $M(k) = 2^k - 1$ is true for all $k < n$. Then for n, we have $M(n) = 2M(n-1) + 1 = 2(2^{n-1} - 1) + 1 = 2^n - 1$.

8.3.1 Iteration

For a recursive equation in the form of $T(n) = rT(n-1) + a$, we can use iteration to solve it.

$$\begin{aligned}
T(n) =& rT(n-1) + a \\
=& r^2T(n-2) + ra + a \\
=& r^3T(n-3) + r^2a + ra + a \\
& \vdots \\
=& r^kT(n-k) + r^{k-1}a + r^{k-2}a + \cdots + ra + a \\
=& r^nT(0) + a\sum_{i=0}^{n-1} r^i
\end{aligned}$$

This is called the "**Top-down**" method.
The "**Bottom-up**" method is to start from $T(0)$ and compute $T(1), T(2), ..., T(n)$.

$$\begin{aligned}
T(0) =& b \\
T(1) =& rT(0) + a = rb + a \\
T(2) =& rT(1) + a = r(rb + a) + a = r^2b + ra + a \\
& \vdots \\
T(n) =& rT(n-1) + a = r(r^{n-1}b + a\sum_{i=0}^{n-2} r^i) + a \\
=& r^nb + a\sum_{i=0}^{n-1} r^i
\end{aligned}$$

Formula of Recursive Equations: If $T(n) = rT(n-1) + a$ and $T(0) = b$, then we have

$$T(n) = r^nb + a\sum_{i=0}^{n-1} r^i = r^nb + a\frac{r^n - 1}{r - 1}$$

8.3.2 First Order Linear Recurrence

A **first order linear recurrence** is a recurrence of the form $T(n) = rT(n-1) + a$. **First order** means that the recurrence is defined by $T(n)$ and $T(n-1)$, or to say it only goes back one step. **Linear** means that the power of $T(n-1)$ is 1.

For a first order linear recurrence that $f(n)$ is a constant, we have:

$$T(n) = \begin{cases} a & \text{if } n = 0 \\ rT(n-1) + g(n) & \text{if } n > 0 \end{cases}$$

Then we have

$$T(n) = r^na + \sum_{i=1}^{n} r^{n-i}g(i)$$

This equation can often be solved by extracting the constant factor r^n.

$$T(n) = r^n a + r^n \sum_{i=1}^{n} \frac{g(i)}{r^i}$$

We can use the theorem that combines the geometric series and the arithmetic series to solve the equation.

$$\sum_{i=1}^{n} ix^i = \frac{x - (n+1)x^{n+1} + nx^{n+2}}{(1-x)^2}$$

8.3.3 Divide and Conquer Recurrence

A **divide and conquer recurrence** is a recurrence of the form $T(n) = aT(n/b) + f(n)$. This can be solved by the regular "**Top-down**" method.

$$\begin{aligned}
T(n) =& 2T(n/2) + n(\text{ assume } n = 2^k) \\
=& 4T(n/4) + 2n \\
=& 8T(n/8) + 3n \\
& \vdots \\
=& 2^i T(n/2^i) + in \\
=& 2^{\log_2 n} T(n/2^{\log_2 n}) + n\log_2 n(\text{ ends when } n/2^{\log_2 n} = 1) \\
=& nT(1) + n\log_2 n
\end{aligned}$$

8.3.4 Master Theorem

Suppose that $T(n) = aT(n/b) + cn^d$, where a is a positive integer, $b \geq 1$ and c and d are constants. Then $T(n)$ has the following asymptotic bounds:

1. If $a < b^d$, then $T(n) = \Theta(n^d)$.
2. If $a = b^d$, then $T(n) = \Theta(n^d \log n)$.
3. If $a > b^d$, then $T(n) = \Theta(n^{\log_b a})$.

9 Counting

9.1 Basic Counting Principles

9.1.1 Product Rule

If there are n_1 ways to perform action 1, and for each of these ways of performing action 1, there are n_2 ways to perform action 2, then there are $n_1 n_2$ ways to perform action 1 and then action 2.

9.1.2 Sum Rule

If there are n_1 ways to perform action 1, and n_2 ways to perform action 2, and the two actions cannot be performed at the same time, then there are $n_1 + n_2$ ways to perform either action 1 or action 2.

9.2 Pigeonhole Principle

If there are $k+1$ objects to be placed into k boxes, then there is at least one box containing two or more objects.

Generalized version: If there are N objects to be placed into k boxes, then there is at least one box containing at least $\lceil N/k \rceil$ objects.

9.3 Permutations and Bijection

A bijection that maps a set S to itself is called a **permutation** of S.

In counting we usually use bijection to show two sets have the same number of elements, so that we can count one set and get the number of elements of the other set. And it is often done implicitly. For example, if we want to count the number of increasing tuple of size n, we can define a bijection as follows:

$$f((a_1, a_2, ..., a_n)) = \{a_1, a_2, ..., a_n\}$$

This means the number of increasing tuple of size n is the same as the number of subsets of size n of a set with n elements, which is $\binom{n}{k}$.

9.4 Inclusion-Exclusion Principle

If S is a finite set and $A_1, A_2, ..., A_n$ are its subsets, then

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{k=1}^{n}(-1)^{k+1} \sum_{1\le i_1<i_2<\cdots<i_k\le n} |A_{i_1} \cap A_{i_2} \cap \cdots \cap A_{i_k}|$$

This can be proved by mathematical induction.

Base case: When $n = 2$, we have

$$|E \cup F| = |E| + |F| - |E \cap F|$$

Inductive hypothesis: Assume that the equation is true for $n = k$. Then for $n = k+1$, we have

$$\left|\bigcup_{i=1}^{k+1} A_i\right| = \left|\bigcup_{i=1}^{k} A_i\right| + |A_{k+1}| - \left|\left(\bigcup_{i=1}^{k} A_i\right) \cap A_{k+1}\right|$$

Notice that the right most term $\left|\left(\bigcup_{i=1}^{k} A_i\right) \cap A_{k+1}\right| = \left|\bigcup_{i=1}^{k}(A_i \cap A_{k+1})\right|$. Then we can use the inductive hypothesis on this term.

This principle can be used to count the number of onto functions from a set A with m elements to a set B with n elements. Firstly, the number of all possible functions is n^m. Then we try to exclude the functions that are not onto. Let the set E_i be the set of functions that map nothing to element i. Then we have

$$\begin{aligned}
\#(\text{onto functions}) =& n^m - |E_1 \cup E_2 \cup \cdots \cup E_n| \\
=& n^m - \sum_{k=1}^{n}(-1)^{k+1} \sum_{1\le i_1<i_2<\cdots<i_k\le n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}| \\
=& n^m - \sum_{k=1}^{n}(-1)^{k+1} \binom{n}{k} (n-k)^m
\end{aligned}$$

9.5 k-Element Permutations

An ordered tuple of k distinct elements taken from set N is called a **k-element permutation** of N.

If N is a positive integer and k is an integer that satisfies $1 \leq k \leq N$, then there are

$$\begin{aligned} P(n,k) =& n(n-1)(n-2)\cdots(n-k+1) \\ =& \frac{n!}{(n-k)!} \end{aligned}$$

9.6 Binomial Coefficients

For integers n and k with $0 \leq k \leq n$, the number of k-element subsets of an n-element set is denoted by $\binom{n}{k}$ and is called a **binomial coefficient**.

$$\binom{n}{k} = C(n,k) = \frac{P(n,k)}{k!} = \frac{n!}{k!(n-k)!}$$

9.6.1 Properties of Binomial Coefficients

1. $\binom{n}{0} = \binom{n}{n} = 1$
2. $\binom{n}{k} = \binom{n}{n-k}$
3. $\sum_{k=0}^{n} \binom{n}{k} = 2^n$ This can be interpreted as: the number of subsets of an n-element set is equal to the sum of the number of k-element subsets of an n-element set for $k = 0, 1, ..., n$.
4. Pascal's identity: $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$ This can be interpreted as: the number of k-element subsets of an n-element set is equal to the sum of the number of subsets that contain element n and the number of subsets that do not contain element n.

9.6.2 Binomial Theorem

For any real numbers x and y and any non-negative integer n, we have

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$$

This can be proved by induction and Pascal's identity.
Base case: When $n = 0$, $(x+y)^0 = 1 = \binom{0}{0} x^0 y^0$.
Inductive hypothesis: Assume that the equation is true for $n = k$. Then for

$n = k + 1$, we have

$$\begin{aligned}
(x+y)^{k+1} =& (x+y)^k(x+y) \\
=& \sum_{i=0}^{k} \binom{k}{i} x^{k-i}y^i(x+y) \\
=& \sum_{i=0}^{k} \binom{k}{i} x^{k-i+1}y^i + \sum_{i=0}^{k} \binom{k}{i} x^{k-i}y^{i+1} \\
=& \sum_{i=0}^{k} \binom{k}{i} x^{k-i+1}y^i + \sum_{i=1}^{k+1} \binom{k}{i-1} x^{k-i+1}y^i \\
=& \binom{k}{0} x^{k+1} + \sum_{i=1}^{k} \left(\binom{k}{i} + \binom{k}{i-1} \right) x^{k-i+1}y^i + \binom{k}{k} y^{k+1} \\
=& \binom{k+1}{0} x^{k+1} + \sum_{i=1}^{k} \binom{k+1}{i} x^{k-i+1}y^i + \binom{k+1}{k+1} y^{k+1}
\end{aligned}$$

9.6.3 Labeling and Trinomial Coefficients

When $k_1 + k_2 + \cdots + k_r = n$, the number of ways to label n distinct objects with r distinct labels so that there are k_i objects with label i is

$$\binom{n}{k_1, k_2, ..., k_r} = \frac{n!}{k_1!k_2! \cdots k_r!}$$

9.7 Combinatorial Proof and Arithmetic Proof

A **combinatorial proof** of an identity is a proof that there is a bijection between the two sides of the identity. An **arithmetic proof** of an identity is a proof that the two sides of the identity are equal by algebraic manipulation.

For example, we can use combinatorial proof for Pascal's identity. The term $\binom{n}{k}$ is the number of k-element subsets of an n-element set. The term $\binom{n-1}{k-1}$ is the number of $k-1$-element subsets of an $(n-1)$-element set. The term $\binom{n-1}{k}$ is the number of k-element subsets of an $(n-1)$-element set. We can make a bijection between the two sides: On the right side, the first term represent the number of k-element subsets of an n-element set that contains element n. The second term represent the number of k-element subsets of an n-element set that does not contain element n. Therefore, the two sides are equal.

9.8 Birthday Attack

The probablity that at least two people in a group of n people have the same birthday is

$$\begin{aligned}
P(n) =& 1 - P(\text{all people have different birthdays}) \\
=& 1 - \frac{365}{365} \cdot \frac{364}{365} \cdot \frac{363}{365} \cdots \frac{365-n+1}{365} \\
=& 1 - \prod_{i=0}^{n-1} \left(1 - \frac{i}{365}\right)
\end{aligned}$$

We can estimate this number using the Taylor series of e^x. Since $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$, for $|x| << 1$, we have $e^x \approx 1 + x$. Thus, $e^{-i/H} \approx 1 - \frac{i}{H}$.

$$\begin{aligned} P(n) =& 1 - \prod_{i=0}^{n-1} \left(1 - \frac{i}{365}\right) \\ \approx& 1 - \prod_{i=0}^{n-1} e^{-i/365} \\ =& 1 - e^{-\sum_{i=0}^{n-1} i/365} \end{aligned}$$

10 Linear Recurrence

A linear homogeneous recurrence of degree k with constant coefficients is a recurrence of the form

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$$

where $c_1, c_2, ..., c_k$ are constants and $c_k \neq 0$.

"Linear" means that the power of each term is 1. **"Homogeneous"** all terms are a multiple of a_n. **"Degree k"** means that a_n is defined by previous k terms. **"Constant coefficients"** means that the coefficients $c_1, c_2, ..., c_k$ are constants.

By induction, we know that such a relation is uniquely determined by the initial values $a_0, a_1, ..., a_{k-1}$.

10.1 Solving Linear Recurrence

To solve a linear homogeneous recurrence of degree k with constant coefficients, we first find the roots of the characteristic equation $x^k - c_1 x^{k-1} - c_2 x^{k-2} - \cdots - c_k = 0$. Then we can find the general solution of the recurrence. (Since it is linear and homogeneous, any linear combination of solutions is also a solution) Finally, we can find the coefficients of the general solution by using the initial values.

For example, to solve the Fibonacci sequence $F_n = F_{n-1} + F_{n-2}$:

1. Find the roots of $x^2 - x - 1 = 0$, which are $\frac{1 \pm \sqrt{5}}{2}$.
2. The general solution is $F_n = \alpha \left(\frac{1+\sqrt{5}}{2}\right)^n + \beta \left(\frac{1-\sqrt{5}}{2}\right)^n$.
3. Since $F_0 = 0$ and $F_1 = 1$, we have $\alpha = \frac{1}{\sqrt{5}}$ and $\beta = -\frac{1}{\sqrt{5}}$.

Theorem: If the characteristic equation has k distinct real roots $r_1, r_2, ..., r_k$, then $\{a_n\}$ is a solution of the recurrence if and only if

$$a_n = \alpha_1 r_1^n + \alpha_2 r_2^n + \cdots + \alpha_k r_k^n$$

To prove this, we need to prove two statements:

1. $\{a_n\}$ in the form of $\alpha_1 r_1^n + \alpha_2 r_2^n + \cdots + \alpha_k r_k^n$ is a solution of the recurrence.
2. Any solution of the recurrence is in the form of $\alpha_1 r_1^n + \alpha_2 r_2^n + \cdots + \alpha_k r_k^n$.

Statement 1: Let's take an example that degree is 2. Since r_1 and r_2 are roots of the characteristic equation, we have

$$\begin{aligned} r_1^2 - c_1 r_1 - c_2 &= 0 \\ r_2^2 - c_1 r_2 - c_2 &= 0 \end{aligned}$$

Then, we can find that

$$\begin{aligned} c_1 a_{n-1} + c_2 a_{n-2} &= c_1(\alpha_1 r_1^{n-1} + \alpha_2 r_2^{n-1}) + c_2(\alpha_1 r_1^{n-2} + \alpha_2 r_2^{n-2}) \\ &= \alpha_1 r_1^{n-2}(c_1 r_1 + c_2) + \alpha_2 r_2^{n-2}(c_1 r_2 + c_2) \\ &= \alpha_1 r_1^{n-2}(r_1^2) + \alpha_2 r_2^{n-2}(r_2^2) \\ &= \alpha_1 r_1^n + \alpha_2 r_2^n \\ &= a_n \end{aligned}$$

Hence, $\{a_n\}$ in the form of $\alpha_1 r_1^n + \alpha_2 r_2^n$ is a solution of the recurrence.

Statement 2:

With the initial values, we can find all the constant $\alpha_1, \alpha_2, ..., \alpha_k$, thus giving us the general solution. Since the solution of the recurrence is unique when the initial values are given, the general solution is the only solution. Hence, the sequence $\{a_n\}$ is the same as $\alpha_1 r_1^n + \alpha_2 r_2^n + \cdots + \alpha_k r_k^n$.

10.1.1 Degenerate Roots

For a characteristic equation of degree 2, if the two roots are the same, then the general solution is

$$a_n = \alpha_1 r_1^n + \alpha_2 n r_2^n$$

Theorem: Suppose there are t roots $r_1, r_2, ..., r_t$ of multiplicity $m_1, m_2, ..., m_t$ respectively. Then the general solution of the recurrence is

$$a_n = \sum_{i=1}^{t} \sum_{j=1}^{m_i} \alpha_{ij} n^{j-1} r_i^n$$

10.2 Linear Nonhomogeneous Recurrence

A linear nonhomogeneous recurrence of degree k with constant coefficients is a recurrence of the form

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} + f(n)$$

The associated homogeneous recurrence is $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$.

However, there is no general method to solve the particular solution $f(n)$. We can only solve it by guessing.

11 Generating Functions

The generating function for a sequence $a_0, a_1, a_2, ...$ is the formal power series

$$G(x) = \sum_{i=0}^{\infty} a_i x^i$$

A finite sequence $a_0, a_1, ..., a_n$ can be represented by the generating function by adding 0s to the end of the sequence.

11.1 Operations on Generating Functions

Let $G(x) = \sum_{i=0}^{\infty} a_i x^i$ and $H(x) = \sum_{i=0}^{\infty} b_i x^i$ be two generating functions. Then we have

$$G(x) + H(x) = \sum_{i=0}^{\infty} (a_i + b_i) x^i$$

$$G(x)H(x) = \sum_{i=0}^{\infty} \left(\sum_{j=0}^{i} a_j b_{i-j} \right) x^i$$

11.2 Useful Generating Functions

- $\frac{1}{1-x} = 1 + x + x^2 + x^3 + \cdots$
- $\frac{1}{1-ax} = 1 + ax + a^2x^2 + a^3x^3 + \cdots$
- $\frac{1}{1-x^r} = 1 + x^r + x^{2r} + x^{3r} + \cdots$
- $\frac{1}{(1-x)^2} = 1 + 2x + 3x^2 + 4x^3 + \cdots$
- $\frac{1}{(1-x)^n} = \sum_{k=0}^{\infty} \binom{n+k-1}{k} x^k$
- $\frac{1}{(1+x)^n} = \sum_{k=0}^{\infty} \binom{-n}{k} x^k$
- $\frac{1}{(1-ax)^n} = \sum_{k=0}^{\infty} \binom{n+k-1}{k} a^k x^k$
- $\frac{1-x^{n+1}}{1-x} = 1 + x + x^2 + \cdots + x^n$
- $(1+x)^n = \binom{n}{0} + \binom{n}{1}x + \binom{n}{2}x^2 + \cdots + \binom{n}{n}x^n$
- $(1+ax)^n = \binom{n}{0} + \binom{n}{1}ax + \binom{n}{2}a^2x^2 + \cdots + \binom{n}{n}a^n x^n$
- $(1+x^r)^n = \binom{n}{0} + \binom{n}{1}x^r + \binom{n}{2}x^{2r} + \cdots + \binom{n}{n}x^{nr}$
- $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$
- $\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots$

Here is a way to derive the generating function if we forget it: Suppose $G(x) = 1 + x + x^2 + x^3 + \cdots$. Then $xG(x) = x + x^2 + x^3 + x^4 + \cdots$. Then $G(x) - xG(x) = 1$, thus $G(x) = \frac{1}{1-x}$. The other generating functions can be derived similarly, or by operations on the generating functions (for example, $(1+x)^2 = (1+x)(1+x)$).

Another way is to use the MacLaurin series of the function. Here is the general formula:

$$f(x) = f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \cdots = \sum_{i=0}^{\infty} \frac{f^{(i)}(0)}{i!}x^i$$

11.3 Counting with Generating Functions

Convolution:

Let $G(x) = \sum_{i=0}^{\infty} a_i x^i$ and $H(x) = \sum_{i=0}^{\infty} b_i x^i$ be two generating functions. Then the coefficient of x^k in $G(x)H(x)$ is $\sum_{i=0}^{k} a_i b_{k-i}$.

11.3.1 k-Combination with Repetition

The number of k-combinations with repetition allowed means to choose k elements from a set S and repetition is allowed. The generating function for this is

$$\left(\frac{1}{1-x}\right)^n = \sum_{k=0}^{\infty} \binom{n+k-1}{k} x^k$$

11.3.2 Extended Binomial Theorem

The extended binomial theorem is

$$(1+x)^n = \sum_{k=0}^{\infty} \binom{n}{k} x^k$$

where n is a real number and $|x| < 1$.

Corollary:

$$\begin{aligned}\binom{-n}{k} =& \frac{(-n)(-n-1)(-n-2)\cdots(-n-k+1)}{k!} \\ =& \frac{(-1)^k n(n+1)(n+2)\cdots(n+k-1)}{k!} \\ =& (-1)^k \binom{n+k-1}{k}\end{aligned}$$

12 Relation

Let A and B be two sets. A binary **relation** R from A to B is a subset of $A \times B$. Note that order matters in a relation, since it is a Cartesian product.

Typical representations of relations can use a table or a directed graph.

A relation **on a set** A is a relation from A to A. The number of relation on a set A is $2^{|A|^2}$.

12.1 Properties of Relations

12.1.1 Reflexive

A relation R on a set A is **reflexive** if $(a, a) \in R$ for all $a \in A$. In the matrix representation, the diagonal elements are all 1 if the relation is reflexive. For example, the divisible relation is reflexive, since $a|a$ for all $a \in \mathbb{Z}$.

12.1.2 Irrreflexive

A relation R on a set A is **irreflexive** if $(a, a) \notin R$ for all $a \in A$. In the matrix representation, the diagonal elements are all 0 if the relation is irreflexive. For example, the not equal relation is irreflexive, since $a = a$ for all $a \in \mathbb{Z}$.

12.1.3 Symmetric

A relation R on a set A is **symmetric** if $(a, b) \in R$ implies $(b, a) \in R$ for all $a, b \in A$. In the matrix representation, the matrix is symmetric if the relation is symmetric. For example, the equal relation is symmetric, since $a = b$ implies $b = a$ for all $a, b \in \mathbb{Z}$.

12.1.4 Antisymmetric

A relation R on a set A is **antisymmetric** if $(a, b) \in R$ and $(b, a) \in R$ implies $a = b$ for all $a, b \in A$. In the matrix representation, the matrix can have at most one 1 on the corresponding positions of the upper triangle and lower triangle if the relation is antisymmetric. But the diagonal elements can be whatever. For example, the divisible relation is antisymmetric, since $a|b$ and $b|a$ implies $a = b$ for all $a, b \in \mathbb{Z}$.

12.1.5 Transitive

A relation R on a set A is **transitive** if $(a, b) \in R$ and $(b, c) \in R$ implies $(a, c) \in R$ for all $a, b, c \in A$. For example, the divisible relation is transitive, since $a|b$ and $b|c$ implies $a|c$ for all $a, b, c \in \mathbb{Z}$.

Theorem: The relation is transitive if and only if $R^n \subseteq R$ for all $n \geq 1$.

The "if" part: If $R^n \subseteq R$ for all $n \geq 1$, then specifically $R^2 \subseteq R$. Then $(a, b) \in R$ and $(b, c) \in R$ implies $(a, c) \in R$.

The "only if" part: Proof by induction. The base case is trivial. Suppose $R^n \subseteq R$ for all $n \geq 1$. Then we need to prove $R^{n+1} = R^n \circ R$. Since $R^n \subseteq R$, every element in R^n in also in R. Since R is transitive, every element in $R \circ R$ is also in R. Therefore, $R^{n+1} \subseteq R$.

12.2 Composite Relation

Let R be a relation from A to B and S be a relation from B to C. Then the **composite relation** $S \circ R$ from A to C is defined as

$$S \circ R = \{(a, c) \in A \times C | \text{ there exists } b \in B \text{ such that } (a, b) \in R \text{ and } (b, c) \in S\}$$

12.3 Closure of a Relation

The reflexive closure of a relation R is a set S that contains all elements of R, is reflexive and is the smallest set that satisfies these conditions. Similarly, we can define the symmetric closure, transitive closure, etc.

The generalized definition of closure is as follows: Let R be a relation on a set A. A relation S on A with property P is called the **closure** of R with respect to P if S is subset of all relations on A with property P and $R \subseteq S$. In other words, S is the smallest relation on A with property P that contains R.

How to find transitive closure: Find all pairs that are connected in the graph.

12.4 Path and Circuit

A **path** from a to b in a relation R is a finite sequence of elements $a_0, a_1, ..., a_n$ such that $a_0 = a$, $a_n = b$ and $(a_i, a_{i+1}) \in R$ for $i = 0, 1, ..., n - 1$. A **circuit** is a path that starts and ends at the same element.

Theorem: There is a path of length n from a to b in R if and only if $(a, b) \in R^n$. This can be proved by induction.

12.4.1 Connectivity Relation

The **connectivity relation** R on a set A that contains all pairs (a, b) such that there is a path from a to b in R is an equivalence relation.

Or more formally,

$$R = \bigcup_{n=1}^{\infty} R^n$$

Since a path that has no loop has at most $|A|$ elements, we can simplify the above equation to

$$R = \bigcup_{n=1}^{|A|} R^n$$

The transitive closure of R is the connectivity relation. To prove this, we need to prove two statements:

1. The connectivity relation is transitive.
2. The connectivity relation is the smallest transitive relation that contains R.

Statement 1: If there is a path from a to b and a path from b to c, then there is a path from a to c.

Statement 2: Let S be a transitive relation that contains R. Then $R* \subseteq S* \subseteq S$, where $R*$ is the transitive closure of R and $S*$ is the transitive closure of S.

The transitive closure can be found by running a Dijkstra algorithm on the adjacency matrix of the graph.

12.5 Equivalence Relation

An **equivalence relation** is a relation that is reflexive, symmetric and transitive. The equivalence relation partitions the set into disjoint subsets, called **equivalence classes**, denoted by $[a]$ or $[a]_R$.

The following statements are equivalent:

1. aRb
2. $[a] = [b]$
3. $[a] \cap [b] \neq \emptyset$

Proof:

1. (1) $\Rightarrow$ (2): $[a] \subseteq [b]$ and $[b] \subseteq [a]$.
2. (2) $\Rightarrow$ (3): $[a] = [b]$ implies $[a] \cap [b] \neq \emptyset$, since there is at least one element in $[a]$ and $[b]$.
3. (3) $\Rightarrow$ (1): $[a] \cap [b] \neq \emptyset$ implies there is an element c such that $c \in [a]$ and $c \in [b]$. Then aRc and cRb implies aRb.

A **partition** of a set A is a collection of nonempty subsets of A such that every element of A is in exactly one of these subsets.

$$A_i \cap A_j = \emptyset \text{ for } i \neq j \text{ and } \bigcup_{i=1}^{\infty} A_i = A$$

The equivalence classes of an equivalence relation on a set A form a partition of A.

12.6 Partial Order

A **partial order** is a relation that is reflexive, antisymmetric and transitive. A set A with a partial order is called a **partially ordered set** or **poset**.

12.6.1 Comparable

Two elements a and b in a poset are **comparable** if either $a \leq b$ or $b \leq a$.

12.6.2 Total Order

A **total order** is a partial order in which every two elements are comparable.

12.6.3 Lexicographic Order

The **lexicographic order** on $A \times B$ is defined as

$$(a, b) \leq (c, d) \text{ if and only if } a < c \text{ or } (a = c \text{ and } b \leq d)$$

12.6.4 Hasse Diagram

A **Hasse diagram** is a directed graph that represents a partial order. It leaves out the edges that can be inferred from the transitive or reflexive property.

12.6.5 Maximal and Minimal

An element a in a poset is **maximal** if there is no element b such that $a < b$. An element a in a poset is **minimal** if there is no element b such that $b < a$.

The **greatest element** a in a poset is an element such that $a \geq b$ for all $b \in A$, which sometimes does not exist. We can define the **least element** similarly.

12.6.6 Upper and Lower Bound

An element a in a poset is an **upper bound** of a set S if $a \geq b$ for all $b \in S$. The **least upper bound** of a set S is an element a such that a is an upper bound of S and $a \leq b$ for all upper bounds b of S. We can define the **lower bound** and **greatest lower bound** similarly.

12.6.7 Well Ordering

A poset is **well ordered** if every nonempty subset has a least element.

12.6.8 Lattice

A **lattice** is a poset in which every two elements have a least upper bound and a greatest lower bound.

13 Graph

A **graph** G is an ordered pair (V, E), where V is a finite set and E is a set of unordered pairs of distinct elements of V. Each edge joins two endpoints, the two endpoints are **adjacent** to each other, the endpoint and the edge are **incident** to each other.

13.1 Simple Graph

A **simple graph** is a graph with no loops and no multiple edges.

13.2 Complete Graph

A **complete graph** is a simple graph in which every pair of distinct vertices is joined by exactly one edge, denoted by K_n.

13.3 Undirected Graph

An **undirected graph** is a graph in which the edges are not ordered pairs. The two endpoints of an edge are **adjacent** to each other, or **neighbors** of each other. We denote the set of neighbors of a vertex v by $N(v)$.

13.4 Directed Graph

A **directed graph** is a graph in which the edges are ordered pairs. If (u, v) is an edge in a directed graph, then we say that u is **adjacent to** v and v is **adjacent from** u.

13.5 Degree

The **degree** of a vertex v in an undirected graph is the number of edges incident to v, denoted by $deg(v)$. The **in-degree** of a vertex v in a directed graph is the number of edges that end at v, denoted by $deg^-(v)$. The **out-degree** of a vertex v in a directed graph is the number of edges that start at v, denoted by $deg^+(v)$.

13.5.1 Handshaking Theorem

In an undirected graph, the sum of the degrees of all vertices is twice the number of edges.

$$\sum_{v \in V} deg(v) = 2|E|$$

In a directed graph, the sum of the in-degrees of all vertices is equal to the sum of the out-degrees of all vertices.

$$\sum_{v \in V} deg^-(v) = \sum_{v \in V} deg^+(v) = |E|$$

13.6 Cycle

A **cycle** is a simple graph in which all vertices have degree 2, denoted by C_n.

13.7 Wheel

A **wheel** can be obtained by adding a vertex to a cycle and connecting it to all vertices of the cycle, denoted by W_n.

13.8 N-Dimensional Hypercube

An N-dimensional hypercube is a graph with 2^N vertices, each of which is labeled by an N-bit string. Two vertices are adjacent if and only if their labels differ in exactly one bit. An N-dimensional hypercube is denoted by Q_N, has $N2^{N-1}$ edges and $N2^{N-1}$ vertices of degree N. An N-dimensional hypercube is always bipartite.

13.9 Bipartite Graph

A **bipartite graph** is a graph whose vertices can be partitioned into two sets V_1 and V_2 such that every edge has one endpoint in V_1 and the other endpoint in V_2. An equivalent definition is that a graph is possible to be colored with two colors such that no two adjacent vertices have the same color. A bipartite graph is denoted by $K_{m,n}$, where m is the number of vertices in V_1 and n is the number of vertices in V_2. A bipartite graph has no odd cycles.

13.9.1 Complete Bipartite Graph

A **complete bipartite graph** is a bipartite graph in which every vertex in V_1 is adjacent to every vertex in V_2, denoted by $K_{m,n}$.

13.9.2 Bipartite Matching

A **bipartite matching** is a set of edges in a bipartite graph such that no two edges share a common endpoint. A **maximum bipartite matching** is a bipartite matching with the maximum number of edges. A matching is **complete** if every vertex in V_1 is incident to an edge in the matching, or $|M| = |V_1|$.

Hall's Marriage Theorem: A bipartite graph G with bipartition (V_1, V_2) has a complete matching from V_1 to V_2 if and only if $|N(S)| \geq |S|$ for all $S \subseteq V_1$.

13.10 Union of Graphs

The **union** of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is the graph $G = (V_1 \cup V_2, E_1 \cup E_2)$.

13.11 Representation of Graph

13.11.1 Adjacency Matrix

The **adjacency matrix** of a graph G is a $|V| \times |V|$ matrix A such that

$$A_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

It can be modified to have multiple edges by using the number of edges instead of 1, or to have loops.

13.11.2 Incidence Matrix

The **incidence matrix** of a graph G is a $|V| \times |E|$ matrix B such that

$$B_{ij} = \begin{cases} 1 & \text{if vertex } i \text{ is incident to edge } j \\ 0 & \text{otherwise} \end{cases}$$

13.11.3 Adjacency List

The **adjacency list** of a graph G is a list of vertices such that each vertex is followed by a list of vertices that are adjacent to it. This representation does not allow multiple edges, but allows loops.

13.12 Isoomorphism

Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, an **isomorphism** from G_1 to G_2 is a bijection $f : V_1 \to V_2$ such that $(u, v) \in E_1$ if and only if $(f(u), f(v)) \in E_2$. There are some useful graph invariants that can be used to determine whether two graphs are isomorphic:

- Number of vertices
- Number of edges
- Degree sequence
- The existence of a simple circuit of length k for $k = 3, 4, 5, ...$

13.13 Path

A **path** from u to v in a graph G is a sequence of n edges $e_1, e_2, ..., e_n$ such that $e_i = (x_i, x_{i+1})$ for $i = 1, 2, ..., n-1$ and $x_1 = u$ and $x_n = v$. The **length** of a path is the number of edges in the path. The path is a **cycle** if $u = v$. A path or cycle is **simple** if it does not contain the same edge more than once.

Lemma: If there is a path from u to v in a graph G, then there is a simple path from u to v in G.

13.13.1 Connected

A graph G is **connected** if there is a path from u to v for every pair of vertices u and v in G.

13.13.2 Connected Component

A **connected component** of a graph G is a maximal connected subgraph of G. A directed graph is **strongly connected** if there is a path from u to v and a path from v to u for every pair of vertices u and v in G. A directed graph is **weakly connected** if the underlying undirected graph is connected.

13.13.3 Cut Vertex and Cut Edge

A **cut vertex** is a vertex whose removal disconnects the graph. A **cut edge** is an edge whose removal disconnects the graph.

A set of edges E' is an **edge cut** if $G - E'$ is disconnected. The **edge connectivity** of a graph G is the minimum size of an edge cut of G, denoted by $\lambda(G)$.

13.13.4 Counting Paths

The number of paths of length k from u to v in a graph G is the (u, v)-entry of A^k, where A is the adjacency matrix of G.

13.14 Eular Path and Circuit

An **Eular path** in a graph G is a simple path that contains every edge of G. An **Eular circuit** in a graph G is a simple circuit that contains every edge of G.

The Eular path exists if and only if the graph is connected and has exactly two vertices of odd degree. The Eular circuit exists if and only if the graph is connected and every vertex has even degree.

13.15 Hamilton Path and Circuit

A **Hamilton path** in a graph G is a simple path that passes through every vertex of G exactly once. A **Hamilton circuit** in a graph G is a simple circuit that passes through every vertex of G exactly once.

Dirac's Theorem: If G is a simple graph with n vertices such that $n \geq 3$ and $deg(v) \geq \frac{n}{2}$ for every vertex v of G, then G has a Hamilton circuit.

Ore's Theorem: If G is a simple graph with n vertices such that $n \geq 3$ and $deg(u) + deg(v) \geq n$ for every pair of nonadjacent vertices u and v of G, then G has a Hamilton circuit.

13.16 Shortest Path

If G is a weighted graph, the shortest path from u to v is the path from u to v with the smallest total weight.

13.16.1 Dijkstra's Algorithm

Dijkstra's algorithm can be used to find the shortest path from a vertex u to every other vertex in a weighted graph G. The algorithm is as follows:

1. Let $S = \{u\}$ and $d(u) = 0$.
2. For every vertex $v \in V - S$, let $d(v)$ be the length of the shortest path from u to v.
3. Choose a vertex $w \in V - S$ such that $d(w)$ is minimum and add w to S.
4. For every vertex $v \in V - S$, if $d(w) + l(w, v) < d(v)$, then let $d(v) = d(w) + l(w, v)$.
5. Repeat steps 3 and 4 until $S = V$.

The runtime of Dijkstra's algorithm is $O(|V|^2)$, but can be improved to $O(|E| + |V| \log |V|)$ using a priority queue. This algorithm can only work on graphs with non-negative weights. If there are negative weights, we can use the Bellman-Ford algorithm, which runs in $O(|V||E|)$ time.

13.17 Planar Graph

A **planar graph** is a graph that can be drawn in the plane without any edges crossing. An n-dimensional hypercube is planar if and only if $n \leq 3$. A complete graph K_n is planar if and only if $n \leq 4$.

13.17.1 Euler's Formula

If G is a connected planar graph with v vertices, e edges and r regions, then

$$v - e + r = 2$$

This can be proved by induction on the number of edges. The inductive step is to add an edge to the graph. There are two cases: the edge creates a new region or the edge does not create a new region.

The **degree of a region** is the number of edges that bound the region. When an edge occur twice in the boundary of a region, it is counted twice. By this we can prove the corollaries:

- If G is a connected planar simple graph with v vertices, e edges and $v \geq 3$, then $e \leq 3v - 6$. This is because each region is bounded by at least 3 edges and each edge is counted twice. Hence $2e = \sum_{r \in R} deg(r) \geq 3r$. Then we substituted r in the Euler's formula and get $e \leq 3v - 6$.

- If G is a connected planar simple graph, there exists a vertex of degree at most 5. Proof by contradiction. Suppose every vertex has degree at least 6. Then by Handshaking Theorem, $2e \geq 6v$. Then $e \geq 3v$. Then by the corollary above, $3v \leq e \leq 3v - 6$, which is a contradiction.

- If G is a connected planar simple graph with v vertices, e edges and $v \geq 3$, but has no circuits of length 3, then $e \leq 2v - 4$. Similar to the first corollary, each region is bounded by at least 4 edges and each edge is counted twice. Hence $2e = \sum_{r \in R} deg(r) \geq 4r$. Then we substituted r in the Euler's formula and get $e \leq 2v - 4$.

13.17.2 Kuratowski's Theorem

If a graph G is planar, so will be any graph obtained from G by a sequence of elementary subdivisions. An elementary subdivision is the replacement of an edge by a path of length 2 or more. Two graphs are called **homeomorphic** if both can be obtained from the same graph by a sequence of elementary subdivisions.

The more useful version of Kuratowski's theorem is that a graph is planar if and only if it does not contain a subgraph that is homeomorphic to K_5 or $K_{3,3}$.

13.17.3 Platonic Solids

There are only 5 platonic solids:

- Tetrahedron ($\{3, 3\}$)
- Cube ($\{4, 3\}$)
- Octahedron ($\{3, 4\}$)
- Dodecahedron ($\{5, 3\}$)
- Icosahedron ($\{3, 5\}$)

where the first number is the number of sides of each face and the second number is the number of faces that meet at each vertex.

Notice that in the planar graph representation of a platonic solid of n faces:

- $r = n$
- $pr = qv = 2e$

Combining this two equations with Euler's formula, we get

$$\frac{1}{p} + \frac{1}{q} = \frac{1}{2} + \frac{1}{r} > \frac{1}{2}$$

Iterating through all possible values of p, q and r, we get the five platonic solids.

13.18 Graph Coloring

A **coloring** of a graph G is an assignment of a color to each vertex of G such that no two adjacent vertices have the same color. The **chromatic number** of a graph G, denoted by $\chi(G)$, is the minimum number of colors needed to color G.

By the **Four Color Theorem**, every planar graph can be colored with at most 4 colors, or $\chi(G) \leq 4$. We now give proof to the weaker version of the theorem, which states that every planar graph can be colored with at most 6 colors, or $\chi(G) \leq 6$. Then another proof is given to the stronger version of the theorem, which states that every planar graph can be colored with at most 5 colors, or $\chi(G) \leq 5$.

Six Color Theorem: By induction on the number of vertices. Base case: $|V| = 1$. Inductive step: Suppose every planar graph with n vertices can be colored with at most 6 colors. By the previous corollary, there exists a vertex v of degree at most 5. We remove the vertex v and by inductive hypothesis, the remaining graph can be colored with at most 6 colors. Then we put back the vertex v, since it has at most 5 neighbors, there is at least one color that is not used by its neighbors. Then we can color v with that color.

Five Color Theorem: By induction on the number of vertices. The base case is the same as above. The inductive step is the same as above when we can find a vertex of degree 4 or less, or when we can find a vertex of degree 5 but adjacent vertices only use 4 colors. Then we need to proof the case when the vertex has degree 5 and adjacent vertices use all 5 colors. Let the 5 adjacent vertices be v_1, v_2, v_3, v_4, v_5, and the color of v_i be c_i. Then there are 2 cases:

1. There is no path from v_1 to v_3, then we can use the color c_1 to color v_3. (If there is a vertex adjacent to v_3 and colored c_1, we use c_3 to color it. Basically we flip the color of a chain that c_3 and c_1 are used repeatedly) Then c_3 is never used and can be used to color the vertex v.

2. There is a path from v_1 to v_3, since this will create a cycle, then there must be no path from v_2 to v_4. Then we repeat the same process as above.

13.19 Tree

A **tree** is a connected undirected graph with no simple circuits. An undirected graph is a tree if and only if there is a unique simple path between every pair of vertices.

Proof: "⇒": Easy since a tree is connected and has no simple circuits. "⇐": Easy to find it is connected. Suppose there is a simple circuit in the graph. Then there are two simple paths between two vertices in the circuit, which is a contradiction.

A **rooted tree** is a tree in which one vertex has been designated as the root and every edge is directed away from the root. A rooted tree is an m**-ary tree** if every vertex has at most m children. A **full** m**-ary tree** is an m-ary tree in which every internal vertex has exactly m children. A full m-ary tree with n vertices, i of which are internal, l leaves satisfies the following equations:

$$n = mi + 1, \quad n = i + l$$

The **level** of a vertex in a rooted tree is the length of the path from the root to the vertex. The **height** of a rooted tree is the maximum level of any vertex in the tree. A rooted tree is **balanced** if all leaves are at level h or $h - 1$. There are at most m^h leaves in an m-ary tree of height h. The equation $h \geq \lceil \log_m l \rceil$ holds for any m-ary tree with l leaves.

13.19.1 Polish Notation and Expression Tree

The **Polish notation** of an expression is an expression in which the operator is written before its operands. For example, 3+4 is written as +34. This corresponds to the preorder traversal of the expression tree.

Similarly, the **reverse Polish notation** of an expression is an expression in which the operator is written after its operands. For example, $3 + 4$ is written as 34+. This corresponds to the postorder traversal of the expression tree.

13.20 Catalan Number

The **Catalan number** C_n is the number of different binary trees with n vertices. Since a binary tree can be partitioned into a root, a left subtree and a right subtree, we have the following recurrence relation:

$$C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}$$

Let $f(x) = \sum_{i=0}^{\infty} C_i x^i$. Then we have

$$\begin{aligned} f(x)^2 &= \left(\sum_{i=0}^{\infty} C_i x^i\right)\left(\sum_{j=0}^{\infty} C_j x^j\right) \\ &= \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} C_i C_j x^{i+j} \\ &= \sum_{n=0}^{\infty}\sum_{i=0}^{n} C_i C_{n-i} x^n \\ &= \sum_{n=0}^{\infty} C_{n+1} x^n \end{aligned}$$

Then we have

$$x f(x)^2 + 1 = f(x)$$

Solving the quadratic equation, we get

$$f(x) = \frac{1 \pm \sqrt{1-4x}}{2x}$$

Since $f(0) = 1$, we have

$$f(x) = \frac{1 - \sqrt{1-4x}}{2x}$$

Then, using the extended binomial theorem, we have

$$\sqrt{1-4x} = \sum_{i=0}^{\infty} \binom{\frac{1}{2}}{i} (-4x)^i$$

Substituting this into the equation above, we get

$$\begin{aligned} \frac{1-\sqrt{1-4x}}{2x} &= \frac{1 - \sum_{i=0}^{\infty}\binom{\frac{1}{2}}{i}(-4x)^i}{2x} \\ &= \frac{-\sum_{i=1}^{\infty}\binom{\frac{1}{2}}{i}(-4x)^i}{2x} \\ &= \sum_{i=1}^{\infty} -\frac{1}{2}\binom{\frac{1}{2}}{i}(-4)^i x^{i-1} \\ &= \sum_{i=0}^{\infty} -\frac{1}{2}\binom{\frac{1}{2}}{i+1}(-4)^{i+1} x^i \end{aligned}$$

Then we have

$$\begin{aligned}
C_n =& -\frac{1}{2}\binom{\frac{1}{2}}{n+1}(-4)^{n+1} \\
=& -\frac{1}{2}\frac{\frac{1}{2}(\frac{1}{2}-1)(\frac{1}{2}-2)...(\frac{1}{2}-n)}{(n+1)!}(-4)^{n+1} \\
=& -\frac{1}{2}\frac{(-1)^n\frac{1}{2}^{n+1}1\cdot 3\cdot 5\cdot ...\cdot(2n-1)}{(n+1)!}(-4)^{n+1} \\
=& \frac{2^n 1\cdot 3\cdot 5\cdot ...\cdot(2n-1)}{(n+1)!} \\
=& \frac{1}{n+1}\frac{1\cdot 2\cdot 3\cdot 4\cdot ...\cdot(2n-1)\cdot 2n}{n!n!} \\
=& \frac{1}{n+1}\binom{2n}{n}
\end{aligned}$$

13.21 Spanning Tree

A simple graph G is **connected** if and only if it has a spanning tree.

13.21.1 Prim's Algorithm

Prim's algorithm can be used to find the minimum spanning tree of a weighted graph G.

runtime: $O(e \log v)$

correctness: By induction on the number of edges. Suppose the current tree is T and it is a subgraph of some minimum spanning tree M. When adding an edge e to the tree T, we want to prove that $T \cup \{e\}$ is still a subgraph of some minimum spanning tree M. If e is in M, then $T \cup \{e\}$ is a subgraph of M. If e is not in M, then $T \cup \{e\}$ contains a cycle. Then there must be an edge f in the cycle that is not in T. Then $M \cup \{e\} - \{f\}$ is another minimum spanning tree that contains $T \cup \{e\}$.

13.21.2 Kruskal's Algorithm

Very similar to Prim's algorithm, but instead of adding edges to the tree, we add edges to the forest.

runtime: $O(e \log e)$

correctness: Same as Prim's algorithm.

13.22 NP Complete and SAT

A given boolean expression is **satisfiable** if there is an assignment of truth values to the variables that makes the expression true. A **k-CNF** expression is a boolean expression which is $f_1 \wedge f_2 \wedge ... \wedge f_k$, where each f_i is $l_{i1} \vee l_{i2} \vee ... \vee l_{ik}$, where each l_{ij} is a literal.

13.22.1 2-CNF

The 2-CNF expression is satisfiable, and is polynomial time decidable. For each disjunction $l_{i1} \vee l_{i2}$, we add an edge between $\neg l_{i1}$ and l_{i2} and $\neg l_{i2}$ and l_{i1}. Then if there is exists a path from l to $\neg l$ and $\neg l$ to l, then the expression is not satisfiable.

Proof:

- Claim 1: If there is a path from x to y, then there is no path from $\neg y$ to $\neg x$.
- Claim 2: The expression is unsatisfiable if and only if there is a variable x such that there is a path from x to $\neg x$ and $\neg x$ to x. This is because, one of these two path must begin with $\top$ and end with $\bot$, then somewhere in the middle, we can find $\top \to \bot$, which is a contradiction. If there are no such literal, then we do as follows:
  - Find an unassigned literal x, with no path from x to $\neg x$.
  - Assign x to $\top$ and all its reachable literals to $\top$.
  - Assign $\neg x$ to $\bot$ and all its reachable literals to $\bot$.
  - Repeat until all literals are assigned.

  This assignment is well defined because there is no path from x to y and $\neg y$ at the same time. If so, by claim 1 we know there is a path from y to $\neg x$, then concatenating the two paths, we get a path from x to $\neg x$, which is a contradiction.

13.22.2 SAT and Other NP Complete Problems

By **Cook's Theorem**, SAT is NP complete. Then we prove DCLIQUE is NP complete by reducing SAT to DCLIQUE. Then we prove DVC is NP complete by reducing DCLIQUE to DVC.

A **clique** in an undirected graph G is a subset S of vertices such that every two vertices in S are adjacent. The **CLIQUE** problem is to find the maximum clique in a graph. The **DCLIQUE** problem is to determine whether a graph has a clique of size k. We can reduce a k-SAT problem to a DCLIQUE problem by constructing a graph G with vertices representing the literals and edges representing the clauses. Then we can reduce a DCLIQUE problem to a DVC problem by constructing a graph G with vertices representing every literals (allow duplicates) and edges across clauses (but not from x to $\neg x$). A clique of size k in G corresponds to a set of literals that satisfies k clauses.

A vertex color of a graph G is a set of vertices that every edge is incident to at least one vertex in the set. The **vertex coloring** problem is to find the minimum size of such a set. The **DVC** problem is to determine whether a graph has a vertex color of size k. We can reduce a DVC problem to a vertex coloring problem by constructing a complement graph G' of G. If there is a vertex color of size k in G', then there is a clique of size $|V|-k$ in G.

We have a 2-approximation algorithm for vertex coloring.

1. Choose an arbitrary edge $e = (u, v)$ in E.
2. Remove all edges incident to u and v.
3. Repeat until all edges are removed.

This is because the algorithm gives a maximum matching for G. The optimal vertex coloring is at least the size of the maximum matching. And the vertex coloring given by the algorithm is twice the size of the maximum matching. Hence we have

$$|C| = 2|M| \leq 2|C^*|, \frac{|C|}{|C^*|} \leq 2$$