CM1020: Discrete Mathematics

Summary

Arjun Muralidharan

28th February 2020

Contents

List of Figures

List of Tables

1 Sets

Learning Outcomes

✓ Understand sets and power sets

✓ Learn the listing methods and rules of inclusion methods

✓ Learn to manipulate set operations

✓ Represent a set using Venn diagrams

✓ Understand and apply De Morgan's law

✓ Understand, and apply commutative, associative and distributive laws

1.1 Definition of a Set

A set is an *unordered collection* of *distinct objects*, called *elements*. We write $a \in A$ to denote that a is an element of the set A, and $a \notin A$ to denote that a is **not** an element of A.

We can describe a set using **roster method** as $A = \{a, b, c, d.\}$ or **set builder notation**:

$$A = \{x \in \mathbb{Z} \mid x \text{ is odd}\}$$

Two sets are considered equal if they have the same elements.

$$\forall x(x \in A \leftrightarrow x \in B)$$

An empty set is denoted as $\{\}$ or $\emptyset$. Do not confuse $\emptyset$ with $\{\emptyset\}$ (the set containing the empty set).

1.2 Subsets

A is a subset of B if and only if all elements of A are in B. B is a superset of A.

$$A \subseteq B$$

$$B \supseteq A$$

Further, we can show **proper subsets**, which imply that A $\neq$ B

$$A \subset B$$

$$B \supset A$$

Show that $A \subseteq B$ Show that if x belongs to A, then x belongs to B.

Show that $A \nsubseteq B$ Find a single $x \in A$ such that $x \notin B$

Show that $A = B$ Show that $A \subseteq B$ and $B \subseteq A$

Every nonempty set S has at least two subsets: the empty set $\emptyset$ and the set S itself.

1.3 Cardinality

The *cardinality* of a *finite* set S is denoted as $|S|$ and describes the number of distinct elements in a set. Because the empty set has no elements, it follows that $|\emptyset| = 0$.

1.4 Power Sets

The power set $\mathcal{P}(S)$ of S is the set of all subsets of the set S. For nonempty sets, the empty set and the set itself are members of the power set.

The empty set has exactly one subset, itself.

$$\mathcal{P}(\emptyset) = \{\emptyset\}$$

The set $\{\emptyset\}$ has two subsets.

$$\mathcal{P}(\{\emptyset\}) = \{\{\emptyset\}, \emptyset\}$$

If two sets have the same power set, the sets themselves are equivalent. That is:

$$\mathcal{P}(A) = \mathcal{P}(B) \leftrightarrow A = B$$

The cardinality of a power set is given as

$$|\mathcal{P}(S)| = 2^n \text{ where } n \text{ is the cardinality of } S$$

1.5 Set Operations & Identities

The Universal Set is denoted as U and contains everything.

1.5.1 Union

The union of A and B is denoted as $A \cup B$ and contains all elements that are in A or B, or both. This is shown in Figure 1.

$$A \cup B = \{x \mid x \in A \vee x \in B\}$$

The union of sets is **commutative**, **associative** and **distributive** over the intersection.

$$A \cup B = B \cap A \qquad \text{(Commutative Property)}$$

$$(A \cup B) \cup C = A \cup (B \cup C) \qquad \text{(Associative Property)}$$

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C) \qquad \text{(Distributive Property)}$$

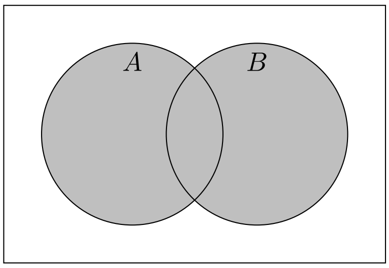


Figure 1. *Union of two sets*

1.5.2 Intersection

The intersection of A and B is denoted as $A \cap B$ and contains all elements that are in A and B. This is shown in Figure 2.

$$A \cap B = \{x \mid x \in A \wedge x \in B\}$$

The intersection of sets is **commutative**, **associative** and **distributive** over the union.

$$A \cap B = B \cap A \tag{Commutative Property}$$
$$(A \cap B) \cap C = A \cap (B \cap C) \tag{Associative Property}$$
$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \tag{Distributive Property}$$

Two sets are called *disjoint* if $A \cap B = \emptyset$.

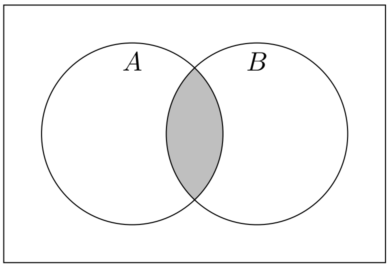


Figure 2. *Intersection of two sets*

1.5.3 Set Difference

The difference of A and B is denoted as $A - B$ and contains all elements that are in A but not B. This is shown in Figure 3.

$$A - B = \{x \mid x \in A \wedge x \notin B\}$$

The set difference of sets is **not commutative** and **not associative**.

$$A - B \neq B \cap B - A$$
$$(A - B) \cap C \neq B - A$$

An important identify for the set difference is

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

Important: The difference of two sets is not treated like an arithmetic subtraction. It only means taking all elements of a set, and removing any elements that overlap with the second set.

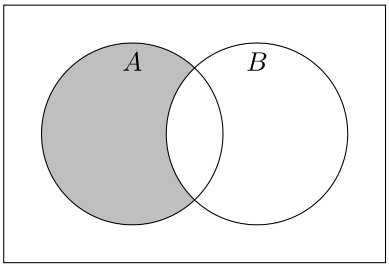


Figure 3. *Set difference of two sets*

1.5.4 Complement

$U$is the universal set comprising everything. The complement of A with respect to U is denoted as $\bar{A}$ and contains all elements that are in U but not A. This is equivalent to $U - A$ and is shown in Figure 4.

$$\bar{A} = \{x \mid x \in U \land x \notin A\}$$

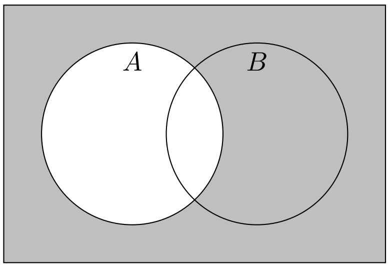


Figure 4. *Complement of a set*

1.5.5 Symmetric Difference

The symmetric difference of A and B is denoted as $A \oplus B$ and contains all elements that are in A or in B, but **not** in both. This is shown in Figure 5.

$$A \oplus B = \{x \mid (x \in A \lor x \notin B) \text{ and } x \notin A \cap B\}$$

In other words, $A \oplus B = A \cup B - A \cap B$.

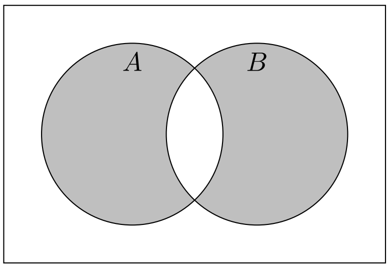


Figure 5. *Symmetric difference of two sets*

The symmetric difference of sets is **commutative** and **associative**. The intersection is **distributive** over the symmetric difference.

$$A \oplus B = B \oplus A \qquad \text{(Commutative Property)}$$

$$(A \oplus B) \oplus C = A \oplus (B \oplus C) \qquad \text{(Associative Property)}$$

$$A \cap (B \oplus C) = (A \cap C) \oplus (B \cap A) \qquad \text{(Distributive Property)}$$

1.5.6 Cartesian Products

The *ordered n-tuple* $(a_1, a_2, \ldots, a_n)$ is the ordered collection that has a_1 as its first element, a_2 as its second element, $\ldots$, and a_n as it nth element.

Let A and B be sets. The *Cartesian Product* of A and B, denoted by $A \times B$, is the set of all ordered pairs (a, b) where $a \in A$ and $b \in B$. Hence,

$$A \times B = (a, b) \mid a \in A \wedge b \in B$$

1.6 Principle of Inclusion-Exclusion

When operating on **union** of two or more sets, the cardinality of the union can be determined using the **principle of inclusion-exclusion**.

The cardinality of the union of two sets is given by adding the cardinalities of the two sets, and subtracting the cardinality of the intersection (as to not double count these elements).

$$|A \cup B| = |A| + |B| - |A \cap B| \qquad \text{(Principle of inclusion-exclusion)}$$

For three sets, the principle states that the “undercount” of elements must be added back.

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

More generally, the principle states that the cardinality of the union of n sets is given by adding the individual cardinalities, subtracting the cardinalities of the intersections of all pairs, adding the cardinalities of the intersection of all 3-tuple sets, subtracting the cardinalities of 4-tuple sets, and so on. Formally:

Let $A_1, A_2, \ldots, A_n$ be finite sets. Then

$$\begin{aligned}|A_1 \cup A_2 \cup \cdots \cup A_n| = &\sum_{1 \leq i \leq n} |A_i| - \sum_{1 \leq i \leq j \leq n} |A_i \cap A_j| \\ &+ \sum_{1 \leq i \leq j \leq k \leq n} |A_i \cap A_j \cap A_k| - \cdots + (-1)^{n+1} |A_1 \cap A_2 \cap \ldots \cap A_n|.\end{aligned}$$

1.7 De Morgan's Laws

De Morgan's laws describe how mathematical statements and concepts are related through their opposites.

First Law The complement of the union of two sets is equal to the intersection of their complements.

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

Second Law The complement of the intersection of two sets is equal to the union of their complements.

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

1.8 Summary of Identity Laws

A compiled overview of identity laws of sets are shown in Table 1.

Name	Identity
Identity Laws	$A \cap U = A$ $A \cup \emptyset = A$
Domination Laws	$A \cup U = U$ $A \cap \emptyset = \emptyset$
Idempotent Laws	$A \cup A = A$ $A \cap A = A$
Complementation Law	$\overline{\overline{A}} = A$
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	$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ $AA \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ $A \cap (B \oplus C) = (A \cap C) \oplus (B \cap A)$
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	$A \cup \overline{A} = U$ $A \cap \overline{A} = \emptyset$

Table 1. *Summary of Identities in Set Theory*

1.9 Partitions

A **partition** of a set S is a a set of *disjointed, non-empty* subsets of S that have S as their union. This is shown in Figure 6.

In other words, the set of subsets A_i is a partition of S if and only if

$$A_i \neq \emptyset$$

$$A_i \cap A_j = \emptyset when i \neq j$$

$$A_1 \cup A_2 \cup A_3 \ldots \cup A_i = S$$

2 Functions

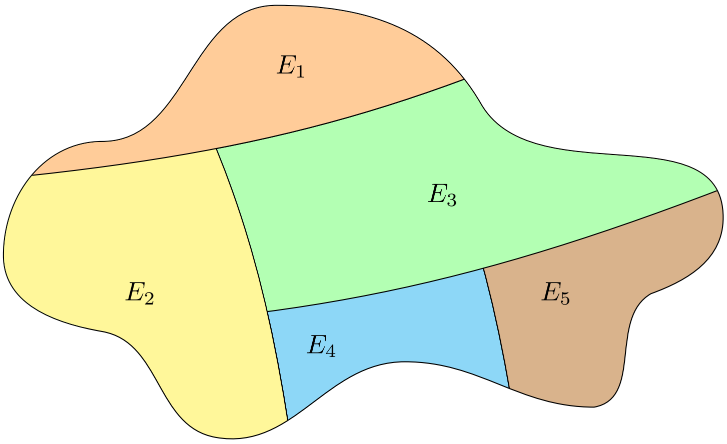


Figure 6. *Partitions of a set*

Learning Outcomes

- ✓ Understand what is a function and define its domain, and co-domain,
- ✓ Define the range of a function and the pre- image of an element
- ✓ Recognise linear, quadratic and exponential functions
- ✓ Define and learn examples of injective and surjective functions.
- ✓ Build functions compositions and a function's inverse.
- ✓ Recognise and define some particular functions such as absolute value, floor, ceiling

2.1 Definition of a function

Let A and B be nonempty sets. A *function* from A to B is an assignment of exactly one element of B to every element of A.

$$f\colon A \to B$$

A function is a subset of the cartesian product (see section 1.5.6) $A \times B$ that has only one ordered pair (a, b) for each $a \in A$.

2.2 Domain, Co-Domain and Range

In a function $f\colon A \to B$, A is the **domain** D_f, and B is the **co-domain** $Co\text{-}D_f$. When noted as $f(a) = b$, a is the *pre-image* and b is the *image*. The **range** R_f is the set of all images of A in B.

Two functions are *equal* if and only if they have the same domain, co-domain and map each element of the domain the same element in the co-domain.

A function is called *real-valued* if $D_f = \mathbb{R}$ and *integer-valued* if $D_f = \mathbb{Z}$.

2.3 Injective & Surjective Functions

2.3.1 Injective Functions

A function is **injective** (or one-to-one) if $f(a) = f(b) \to a = b$. In other words, if two images are the same, their pre-images are the same. **An injective function only assigns one image for each pre-image**. This is clarified by the contrapositive of the above implication: $f(a) \neq f(b) \to a \neq b$. In general:

$$\forall a \forall b(a = b \to f(a) = f(b))$$

If a function is **strictly increasing** or **decreasing** (i.e. $\forall a \forall b(a \leq b \to f(a) \leq f(b))$ or $\forall a \forall b(a \geq b \to f(a) \geq f(b))$), the function is injective.

Proof To prove a function is injective, show that for a fixed $x1, x2 \in D_f$ where $f(x_1) = f(x_2)$ it follows that $x_1 = x_2$. This can be done by transforming the equation $f(x_1) = f(x_2)$ to the form $x_1 = x_2$

2.3.2 Surjective Functions

A function is **surjective** (or onto) if $\forall y \exists x(f(x) = y)$. That is, for every image in $Co\text{-}D_f$ there exists a pre-image in D_f, thus $Co\text{-}D_f = R_f$.

Proof To prove a function is surjective, express x in terms of y and insert into the function $f(x)$. Then, transform this to $f(x) = y$ and confirm that there exists an $f(x)$ for all y in the Co-Domain.

2.4 Bijective Functions

A function is called *bijective* or a *one-to-one correspondence* if it is **both injective and surjective**.

2.5 Inverse of a function

The **inverse** f^{-1} of a *bijective* function $f : A \to B$ is the function that assigns a unique element of A to each element in B. Only injective functions are **invertible**.

To find the inverse, transform the function $f(x)$ to the form $f(y)$, isolating x to one side.

2.6 Composition

The function composition of two functions f and g is denoted as $f \circ g$. This is equivalent to first applying the right-most function (g), and subsequently applying the functions further upfront (f).

$$(f \circ g)(x) = f(g(a))$$

The domain of the composition $f \circ g$ is the domain of g, while the co-domain is the co-domain of f. Hence if $f : A \to B$ and $g : B \to C$ then $f \circ g : A \to C$. This is illustrated in Figure 7.

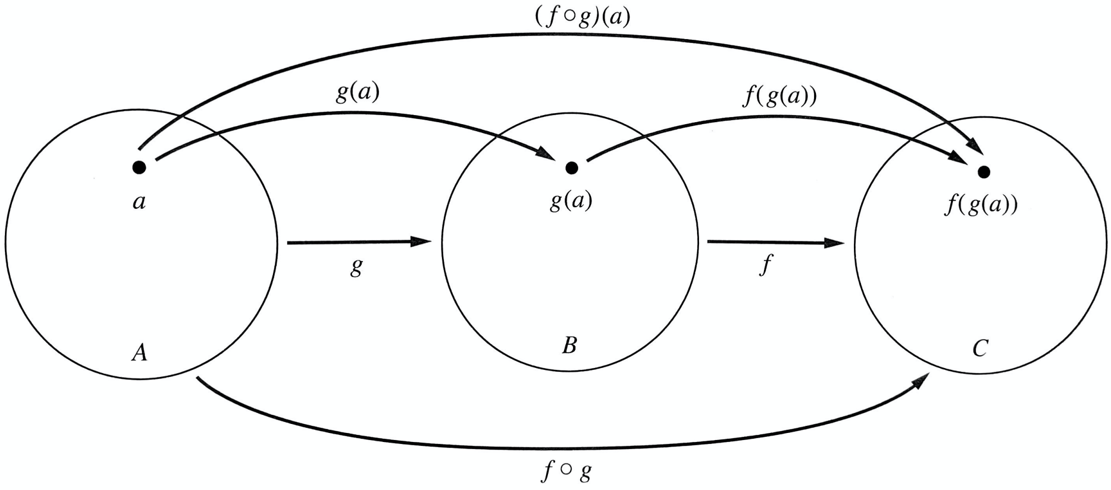


Figure 7. *The composition of functions f and g*

2.7 Floor and Ceiling Functions

The **floor function** $\lfloor x \rfloor$ assigns to the real number x the largest integer that is less than or equal to x.

The **ceiling function** $\lceil x \rceil$ assigns to the real number x the smallest integer that is greater than or equal to x.

3 Propositional Logic

Learning Outcomes

✓ Define propositional logic and learn some of its properties

✓ Give some examples of domains where propositional logic is used

✓ Defining a proposition in mathematical context and distinguish examples of sentences considered as propositions and other that are not propositions

✓ Learn how propositional variables help to simplify notations

✓ Learn how we can build a truth table and give an example of how it works

✓ Practice how to build compound statements using logical operators and take into consideration the order of precedence of the operators

✓ Define truth sets and learn some examples of truth sets for some compound propositions

✓ Define what is an implication and equivalence, what are their properties and build their truth table

✓ Learn some important laws of propositional logic including De Morgan's laws, and practice their use in building a reasoning, and proving equivalence

3.1 Connectives

A **proposition** is a declarative sentence that is either true or false, but not both. We use **propositional** or **sentential** variables such as $p, q, r, s \dots$ to represent propositions. The **truth value of a proposition** is denoted by **T** or **F**. The simplest form of a proposition is an **atomic proposition**. Propositions can be placed in relation to one another using **connectives**.

3.1.1 Negation

Let p be a proposition. The *negation* of p is stated as

$$\neg \mathbf{p}$$

and proposes that "It is not the case that p".

The proposition $\neg p$ is read "not p". The truth value of the negation of $p, \neg p$, is the opposite of the truth value of p.

3.1.2 Conjunction

Let p and q be propositions. The *conjunction* of p and q, is denoted as

$$\mathbf{p} \wedge \mathbf{q}$$

and proposes that "p and q".

The *conjunction* $p \wedge q$ is true when both p and q are true and is false otherwise.

3.1.3 Disjunction

Let p and q be propositions. The *disjunction* of p and q, denoted as

$$\mathbf{p} \vee \mathbf{q}$$

is the statement "p or q".

The *disjunction* $p \vee q$ is false when both p and q are false and is true otherwise. This is an **inclusive or**, which means that the disjunction is true if at least one of the propositions is true, or both are true.

3.1.4 Conditional Statements & Implication

Let p and q be propositions. The *conditional statement*

$$p \to q$$

is the proposition “if p, then q.” The conditional statement $p \to q$ is false when p is true and q is false, and true otherwise. In the implication $p \to q$, p is the *hypothesis* (or *antecedent* or *premise*) and q is called the *conclusion* (or *consequence*).

The implication does not express causality in discrete mathematics. It only covers the truth value of the statement. It can also be read as:

1. “p is sufficient for q ”
2. “a necessary condition for q is p ”
3. “q unless $\neg p$ ”
4. “q whenever p ”

3.1.5 Exclusive Or

Let p and q be propositions. The *exclusive or* of p and q, denoted as

$$\mathbf{p \oplus q}$$

is the statement “either p or q, but not both”.

The *exclusive or* $p \oplus q$ is true when exactly one of p and q is true and false otherwise. In other words, p and q may never have the same truth value.

3.1.6 Biconditional

Let p and q be propositions. The *biconditional statement*

$$\mathbf{p \leftrightarrow q}$$

is the proposition “p if and only if q”.

The biconditional statement $p \leftrightarrow q$ is true when p and q have the same truth values, and is false otherwise.

3.2 Truth Tables

A truth table evaluates the truth values for propositions and connectives systematically. When given a compound proposition, a truth table can be used to evaluate the atomic propositions step-by-step and arrive at the correct truth value for the compound proposition.

Table 2. *Truth table for basic connectives*

p	**q**	$p \wedge q$	$p \vee q$	$p \oplus q$	$p \rightarrow q$	$p \leftrightarrow q$
T	**T**	T	T	F	T	T
T	**F**	F	T	T	F	F
F	**T**	F	T	T	T	F
F	**F**	F	F	F	T	T

Truth values can also be denoted using binary bits, i.e. T is 1 and F is 0. When constructing a truth table, the number of rows is 2^n where n is the number of propositions involved. So compound proposition involving 3 atomic propositions p, q, r has $2^3 = 8$ rows. Further, The the first column (e.g. p) is constructed with all true values for the first half, and false values for the second half, the second (q) is true for the first 2 rows and false for next two rows, and the third is true for the first row and false for the second row of the table, and then this pattern repeats down the table for p, q and r.

The universal and existential quantifications $\forall$ and $\exists$ are described in subsection 4.1.

3.3 Precedence of Logical Operators

Logical operators have an order of precedence as follows.

$$\forall \quad \exists \quad \neg \quad \wedge \quad \vee \quad \rightarrow \quad \leftrightarrow$$

3.4 Propositional Equivalences

1. **Tautology:** A compound proposition that is always true, no matter the truth values of the individual propositions

2. **Contradiction:** A compound proposition that is always false, no matter the truth values of the individual propositions. This is called **inconsistent**.

3. **Contingency:** A compound proposition that is true for at least one scenario of truth values. This is called **consistent**. All tautologies are consistent by definition.

3.5 Logical Equivalence

Compound propositions that have the same truth value in all possible cases are called **logically equivalent**. In other words, if $p \leftrightarrow q$ is a tautology, then $p \equiv q$.

Logical equivalence can be proved using truth tables.

3.6 De Morgan's Laws

$$\neg(p \land q) \equiv \neg p \lor \neg q \qquad \text{(De Morgan's First Law)}$$

$$\neg(p \lor q) \equiv \neg p \land \neg q \qquad \text{(De Morgan's Second Law)}$$

3.7 Special equivalences

3.7.1 Conjunction-Disjunction

The **conditional-disjunction equivalence** allows us to replace conditional statements with disjunctions. This is especially useful to convert a conditional to a form that can then be transformed using De Morgan's laws.

$$p \to q \equiv \neg p \lor q$$

3.7.2 Contrapositive

The **contrapositive** of a conditional statement is equivalent to the original conditional statement and is defined as follows.

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

3.8 Summary of Logical Equivalences

Table 3. *Summary of Equivalence Laws*

Name	Equivalence
Identity Laws	$p \wedge \mathbf{T} \equiv p$ $p \vee \mathbf{F} \equiv p$
Domination Laws	$p \vee \mathbf{T} \equiv \mathbf{T}$ $p \wedge \mathbf{F} \equiv \mathbf{F}$
Idempotent Laws	$p \vee p \equiv p$ $p \wedge p \equiv p$
Double Negation Law	$\neg(\neg p) \equiv p$
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	$p \vee \neg p \equiv \mathbf{T}$ $p \wedge \neg p \equiv \mathbf{F}$

3.8.1 Logical Equivalences Involving Conditional Statements

$$\begin{aligned}
p \to q &\equiv \neg p \lor q \\
p \to q &\equiv \neg q \to \neg p \\
p \lor q &\equiv \neg p \to q \\
p \land q &\equiv \neg(p \to \neg q) \\
\neg(p \to q) &\equiv p \land \neg q \\
(p \to q) \land (p \to r) &\equiv p \to (q \land r) \\
(p \to q) \land (q \to r) &\equiv (p \lor q) \to r \\
(p \to q) \lor (p \to r) &\equiv p \to (q \lor r) \\
(p \to r) \lor (q \to r) &\equiv (p \land q) \to r \\
(p \lor \neg r) \to (\neg q \land r)
\end{aligned}$$

3.8.2 Logical Equivalences Involving Biconditional Statements

$$\begin{aligned}
p \leftrightarrow q &\equiv (p \to q) \land (q \to p) \\
p \leftrightarrow q &\equiv \neg p \leftrightarrow \neg q \\
p \leftrightarrow q &\equiv (p \land q) \lor (\neg p \land \neg q) \\
\neg(p \leftrightarrow q) &\equiv p \leftrightarrow \neg q
\end{aligned}$$

4 Predicate Logic

The **predicate** of a statement is the property assigned to a specific variable. In the statement "x is greater than 3", x is the variable and "is greater than 3" is the predicate. This statement can be noted as the **propositional function** $P(x)$ Predicate statements that describe valid input are **preconditions**, and statements describing valid output are **postconditions**.

4.1 Quantifiers

Quantification expresses to which extent a propositional function $P(x)$ is true over a range of elements.

4.1.1 Universal Quantification

The *universal quantification* is stated as

$$\forall x P(x)$$

and states "$P(x)$ for all values of x in the domain".

4.1.2 Existential Quantification

The *existential quantification* is stated as

$$\exists x P(x)$$

and states "There exists an element x in the domain such that $P(x)$".

The truth values of universal and existential quantifications are shown in Table 4.

4.1.3 Uniqueness Quantification

The *uniqueness quantifier* is stated as

$$\exists ! x P(x)$$

and states "There exists one and only one element x in the domain such that $P(x)$ "

Table 4. *Truth values of quantifiers*

Statement	*When true?*	*When false?*
$\forall x P(x)$	$P(x)$ is true for every x.	$P(x)$ is false for at least one x.
$\exists x P(x)$	There exists at least an x such that $P(x)$.	$P(x)$ is false for all x.

4.1.4 De Morgan's Laws for Quantifiers

The negation of a universal quantification of a statement $P(x)$ is equivalent to an existential quantification of the negation of $P(x)$. Likewise, the negation of an existential quantification of a statement $P(x)$ is equivalent to the universal quantification of the negation of $P(x)$.

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

4.2 Rules of Inference

An **argument** in propositional logic is a sequence of propositions. All but the final proposition in the argument are called *premises* and the final proposition is called the *conclusion*. An argument is **valid** if the truth of all its premises implies that the conclusion is true. That is,

$$(p_1 \wedge p_2 \wedge \ldots \wedge p_n) \rightarrow q$$

is a tautology where $p_1 \ldots p_n$ are the premises and q is the conclusion. A single premise can in itself be a conditional proposition.

An argument form in propositional logic is a sequence of compound propositions involving propositional variables. An **argument form** is valid no matter which particular propositions are substituted for the propositional variables in its premises, the conclusion is true if the premises are all true. The notation for two premises p and $p \rightarrow q$ and its conclusion q is shown below.

$$\begin{array}{ll} & p \\ & p \rightarrow q \\ \cline{2-2} \therefore & q \end{array}$$

Instead of using truth tables, an argument's truth value can be evaluated by using **rules of inference**. The most common rules for logical propositions are listed in Table 5, while rules for quantified statements are shown in Table 6.

Rule	Tautology	Name
$\begin{array}{ll} & p \\ & p \rightarrow q \\ \cline{2-2} \therefore & q \end{array}$	$(p \wedge (p \rightarrow q)) \rightarrow q$	Modus ponens
$\begin{array}{ll} & \neg q \\ & p \rightarrow q \\ \cline{2-2} \therefore & \neg p \end{array}$	$(\neg q \wedge (p \rightarrow q)) \rightarrow \neg p$	Modus tollens
$\begin{array}{ll} & p \rightarrow q \\ & q \rightarrow r \\ \cline{2-2} \therefore & p \rightarrow r \end{array}$	$((p \rightarrow q) \wedge (q \rightarrow p)) \rightarrow (p \rightarrow q)$	Hypothetical syllogism
$\begin{array}{ll} & p \vee q \\ & \neg p \\ \cline{2-2} \therefore & q \end{array}$	$((p \vee q) \wedge \neg p) \rightarrow q$	Disjunctive syllogism
$\begin{array}{ll} & p \\ \cline{2-2} \therefore & p \vee q \end{array}$	$p \rightarrow (p \vee q)$	Addition
$\begin{array}{ll} & p \wedge q \\ \cline{2-2} \therefore & p \end{array}$	$((p \wedge q) \rightarrow p)$	Simplification
$\begin{array}{ll} & p \\ & q \\ \cline{2-2} \therefore & p \wedge q \end{array}$	$(p) \wedge (q) \rightarrow (p \wedge q)$	Conjunction
$\begin{array}{ll} & p \vee q \\ & \neg p \vee r \\ \cline{2-2} \therefore & q \vee r \end{array}$	$(p \vee q) \wedge (\neg p \vee r) \rightarrow (q \vee r)$	Resolution

Table 5. *Rules of inference*

Rule	Name
$\forall x P(x)$ ——— $\therefore\ P(c)$	Universal Instantiation
$P(c)$ for an arbitrary c ——— $\therefore\ \forall x P(x)$	Universal Generalization
$\exists x P(x)$ ——— $\therefore\ P(c)$ for some element c	Existential Instantiation
$P(c)$ for some element c ——— $\therefore\ \exists x P(x)$	Existential Generalization

Table 6. *Rules of Inference for Quantified Statements*

5 Boolean Algebra

5.1 Boolean Functions

Boolean algebra provides operations for working with the set $\{0, 1\}$:

1. **Complementation** : $\overline{0} = 1$, corresponds to $\neg$
2. **Boolean Product**: $1 \cdot 1$, corresponds to $\wedge$
3. **Boolean Sum**: $1 + 1$, corresponds to $\vee$

The order of precedence is as listed above.

Boolean equalities can be converted to logical equivalences. For example, the boolean equality $1 \cdot 0 + \overline{0 + 1} = 0$ can be restated as $\mathbf{T} \wedge \mathbf{F} \vee \neg(\mathbf{T} \vee \mathbf{F}) \equiv \mathbf{F}$.

A Boolean variable is a variable that may take on values only from the set$B = \{0, 1\}$

A Boolean function of degree n or of order n is a function with domain

$$B^n = \{(x_1, x_2, \ldots, x_n)\,|x_i \in B\}$$

and codomain B. In other words, Boolean functions of degree n are functions of the form

$$F : B^n \to B$$

Two Boolean functions, F and G are equivalent if

$$F(x_1, x_2, x_3, \ldots, x_n) = G(x_1, x_2, x_3, \ldots, x_n)$$

for every ordered n-tuple $(x_1, x_2, x_3, \ldots, x_n) \in B^n$

5.2 Boolean identities

Table 7. *Boolean Identities*

Name	Equivalence
Identity Laws	$x + x = x$ $x \cdot x = x$
Domination Laws	$x + 1 = 1$ $x \cdot 0 = 0$
Idempotent Laws	$x + x = x$ $x \cdot x = x$
Double Negation Law	$\overline{\overline{x}} = x$
Commutative Laws	$x + y = y + x$ $xy = yx$
Associative Laws	$x + (y + z) = (x + y) + z$ $x(yz) = (xy)z$
Distributive Laws	$x + yz = (x + y)(x + z)$ $x(y + z) = xy + xz$
De Morgan's Laws	$\overline{(xy)} = \overline{x} + \overline{y}$ $\overline{(x + y)} = \overline{x}\overline{y}$
Absorption Laws	$x + xy = x$ $x(x + y) = x$
Unit Property	$x + \overline{x} = 1$
Zero Property	$x\overline{x} = 0$

5.2.1 Duality

The laws in Table 7 come in pairs. This is because they use the principle of a **dual**. A dual is a boolean expression obtained by interchanging Boolean sums and Boolean products, as well as 1s and 0s.

5.3 Sum-of-Products Expansion

A **minterm** is a Boolean product of literals with one variable for each literal. The **sum-of-products** form is a sum of minterms expression a Boolean function.

For example, if a function has the variables x, y and z, then a minterm includes all variables, e.g. $xy\overline{z}$.

A boolean sum of minterms has the value 1 when at least one of the minterms has the value 1. Hence when evaluating a Boolean function, all values of the variables that result in the function value being 1 can be converted to minterms.

Using Identities By using the boolean identities, any boolean function can be converted to the sum-of-products form by rearranging the expression to a sum of minterms.

$$\begin{aligned}
F(x,y,z) &= (x+y)\overline{z} \\
&= x\overline{z} + y\overline{z} && \text{(Distributive Law)} \\
&= x1\overline{z} + 1y\overline{z} && \text{(Identity Law)} \\
&= x(y+\overline{y})\overline{z} + (x+\overline{x})y\overline{z} && \text{(Unit Property)} \\
&= xy\overline{z} + x\overline{y}\overline{z} + xy\overline{z} + \overline{x}y\overline{z} && \text{(Distributive Law)} \\
&= xy\overline{z} + x\overline{y}\overline{z} + \overline{x}y\overline{z} && \text{(Idempotent Law)}
\end{aligned}$$

Using Truth Tables Building a truth table for all possible values of the variables will provide a way to see when the function value is 1. Since we know that in these cases the Boolean sum of minterms has to be 1, we can construct the minterms from the variables where the function value is 1.

x	y	z	$x+y$	$\overline{z}$	$(x+y)\overline{z}$
1	1	1	1	0	0
1	1	0	1	1	1
1	0	1	1	0	0
1	0	0	1	1	1
0	1	1	1	0	0
0	1	0	1	1	1
0	0	1	0	0	0
0	0	0	0	1	0

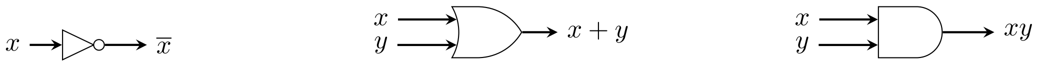
Table 8. *Example: Finding sum-of-products form using truth tables*

Based on the three rows on the example shown in Table 8, we can form the minterms $xy\overline{z}, x\overline{y}\overline{z}$ and $\overline{x}y\overline{z}$, the sum of which is a valid sum-of-products expansion.

5.4 Logic Gates

Boolean algebra is used to model electronic circuitry. The basic elements of circuits are **gates**. A **combinatorial circuit** is a circuit that depends only on the input, and not on the state of the circuit. It has no memory capabilities.

There are three basic types of gates shown in Figure 8.

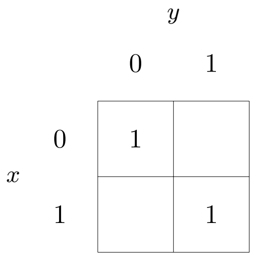


(a) *Inverter* **(b)** *OR gate* **(c)** *AND gate*

Figure 8. *Basic types of gates*

5.5 Karnaugh Maps

To reduce the number of terms in a Boolean expression representing a circuit, a **Karnaugh map** can be drawn. There are *four possible minterms* in the sum-of-products expansion of a Boolean function in the two variables x and y. In a K-Map, a 1 is placed in the cell representing the minterm if it is present. Cells are **adjacent** if the minterms they represent differ in exactly one literal.



		y	
		0	1
x	0	1	
	1		1

Figure 9. *Karnaugh map for a boolean function*

In a K-map with two variables, find the largest possible blocks of 2 adjacent cells containing a 1. These can be reduced to one variable.

In a K-map with three variables, find the largest possible blocks of 2 or 4 adjacent cells containing a 1, these can be reduced to two or 1 literal respectively.

6 Proof Techniques

6.1 Terminology

Theorem A *theorem* is a formal statement that can be shown to be true.

Axiom An *axiom* is a statement that we assume to be true to

6.2 Proof Methods

A proof is a **valid argument** that shows the truth of a mathematical statement. It uses the *premise*, *axioms*, *theorems* to prove a *conjecture* to be true or false. Most commonly, we prove a statement of the form

$$\forall x P(x) \rightarrow Q(x)$$

which is often simplified to omit the universal quantification as

$$p \rightarrow q$$

This statement can be proven with various methods.

6.2.1 Direct Proof

A **direct proof** is constructed by *assuming that p is true* and showing that if p is true, q is true. This is done by expanding the definition underlying p and applying it to q.

6.2.2 Proof by Contrapositive

A **proof by contrapositive** uses the fact that

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

to first assume that q is false. If the contrapositive can be shown to be true, then the original statement is also true.

Vacuous Proofs A proof is *vacuous* or *trivial* if e.g. in the statement $p \to q$ we can show that p is false, because $p \to q$ must be true if p is false.

6.3 Proof by Contradiction

We can show that p is true if

$$\neg p \to (r \land \neg r)$$

is true for some proposition r. Because the conclusion is false, the hypothesis $\neg p$ must be false for the conditional to be true. Therefore, p is true. This proof works by first assuming $\neg p$ and then constructing the conditional. We assume that p is false and then show that this assumption leads to a contradiction, therefore proving that p is true.

6.4 Proof by Induction

Proof by induction can be used to prove statements that assert that $P(n)$ is true for all positive integers n, where $P(n)$ is a propositional function. To prove this, there are two steps.

Basis Step We verify that $P(1)$ is true. Note that we cannot just assume that $P(1)$ is true, we need to show that it indeed is, by other proof methods. Note that the domain of the axiom to be proven may be restricted (e.g. $k > 3$). In this case, the basis step would be $P(4)$ instead of $P(1)$.

Inductive Step We show that the conditional statement $P(k) \to P(k+1)$ is true for all positive integers k. Here, we *assume* that $P(k)$ is true to show that $P(k+1)$ is true, leading the conditional to be true.

Proof by induction can be stated as a rule of inference:

$$(P(1) \land \forall k(P(k) \to P(k+1))) \to \forall n P(n)$$

The general template for proofs by induction is as follows.

1. Express the statement that is to be proved in the form "for all $n \geq b, P(n)$" for a fixed integer b. For statements of the form "$P(n)$ for all positive integers n", let $b = 1$, and for statements of the form "$P(n)$ for all non-negative integers n", let $b = 1$. For some statements of the form $P(n)$, such as inequalities, you may need to determine the appropriate value of b by checking the truth values of $P(n)$ for small values of n.

2. Write out the words “Basis Step”. Then show that $P(b)$ is true, taking care that the correct value of b is used. This completes the first part of the proof.

3. Write out the words “Inductive Step” and state, and clearly identify, the inductive hypothesis, in the form “Assume that $P(k)$ is true for an arbitrary fixed integer $k \geq b$.”

4. State what needs to be proved under the assumption that the inductive hypothesis is true. That is, write out what $P(k+1)$ says.

5. Prove the statement $P(k+1)$ making use of the assumption $P(k)$. (Generally, this is the most difficult part of a mathematical induction proof. Decide on the most promising proof strategy and look ahead to see how to use the induction hypothesis to build your proof of the inductive step. Also, be sure that your proof is valid for all integers k with $k \geq b$, taking care that the proof works for small values of k, including $k = b$.)

6. Clearly identify the conclusion of the inductive step, such as by saying “This completes the inductive step.” After completing the basis step and the inductive step, state the conclusion, namely, “By mathematical induction, $P(n)$ is true for all integers n with $n \geq b$”.

6.5 Strong Induction

When we cannot use induction to easily prove a result, we can consider using **strong induction**.

To prove that $P(n)$ is true for all positive integers n, wheren $P(n)$ is a propositional function, we complete two steps.

Basis Step We verify that the proposition $P(1)$ is true.

Inductive Step We show that the conditional statement $[P(1) \wedge P(2) \wedge \cdots \wedge P(k)] \rightarrow P(k+1)$ is true for all positive integers k.

7 Recursion & Recurrence Relations

7.1 Recursion

A recursively defined object is defined in terms of itself. When defining a *set* recursively, we specify some initial elements in a **basis step** and provide a rule for constructing new elements from those we already have in a **recursive step**.

7.2 Functions

We use two steps for the recursive definition of a function.

Basis Step Specify the value of the function at zero.

Recursive Step Give a rule for finding its value at an integer from its values at smaller integers. Example of a function:

$$f(0) = 3$$
$$f(n+1) = 2f(n) + 3$$

Recursively defined functions are **well-defined**, meaning for every positive integer the function value is determined unambiguously.

7.3 Sets

Just like with functions, sets are defined recursively by defining an initial collection of elements in the **basis step** and rules for forming new elements in the set from those already know to be in the set in the **recursive step**.

We tacitly assume an **exclusion rule**, which states that a recursively defined set contains nothing except the elements defined in the basis step or generated by using the rules in the recursive step.

Example of a set:

- **Basis Step:** $3 \in S$
- **Recursive Step:** If $x \in S$ and $y \in S$, then $x + y \in S$.

Given this example, if we inset 3 as the value for x and y, we see that $6 \in S$. Again by inserting 6 and 3, we get $9 \in S$, and so on.

Definition The set Σ^* of *strings* over the alphabet Σ is defined recursively by

- **Basis Step:** $\epsilon \in \Sigma^*$ (where ϵ is the empty string)
- **Recursive Step:** If $w \in \Sigma^*$ and $x \in \Sigma$, then $wx \in \Sigma^*$.

7.4 Recurrence Relations

A *recurrence relation* for the sequence $\{a_n\}$ is an equation that expresses a_n in terms of one or more of the previous terms of the sequence, namely $a_0, a_1, \cdots, a_{n-1}$, for all integers n with $n \geq n_0$, where n is a non-negative integer. A sequence is called a *solution* of a recurrence relation if its terms satisfy the recurrence relation. The sequence is recursively defined by the recurrence relation.

The **initial conditions** for a recursively defined sequence are the *basis step* for a recurrence relation and define the first terms of a sequence before the recurrence relation takes effect. In order for a recurrence relation to determine a unique solution, both initial conditions and the relation itself are necessary.

7.4.1 Solving Recurrence Relations

A solution to a recurrence relation is a **closed formula** that uniquely identifies any nth term of a recursively defined sequence.

Homogenous Recurrence Relations Let $c_1, c_2, \cdots, c_k$ be real numbers. Suppose that the *characteristic equation*

$$r^k - c_1 r^{k-1} - \cdots - c_k = 0$$

has k *distinct roots* $r_1, r_2, \cdots, r_k$. Then the sequence $\{a_n\}$ is a solution of the recurrence relation

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

if and only If

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

for $n = 0, 1, 2, \cdots$, where $\alpha_1, \alpha_2, \cdots, \alpha_k$ are constants.

Solution Procedure To apply the above theorem, the following steps are always to be followed when solving a homogenous recurrence relation.

1. Find the characteristic equation.
2. Find the distinct roots.
3. State the solution in the form using the constants $\alpha_1, \alpha_2, \cdots$
4. Solve for the constants by using the initial conditions and solving the equation system
5. Plug in the constants into the solution from Step 3.
6. If you find non-distinct roots, then multiply those roots with n , multiplying the first root with $n^0 = 1$, the second with $n^1 = n$, the third with n^2 and so on.

8 Graphs

8.1 Graph Theory

Graphs are discrete structures that connect vertices using edges. A **graph** $G = (V, E)$ consists of V, a nonempty set of *vertices* and E, a set of *edges*. Each edge has either **one or two egdes** called *endpoints*.

A *directed* graph (V, E) consists of a nonempty set of vertices V and a set of *directed edges E*. Each directed edge is associated with an **ordered pair** of vertices.

Walks A *walk* is any alternating sequence of vertices and edges in a graph, e.g. $v_o, e_1, v_1, e_2, \ldots$

Trails A *trail* is a walk with no repeated edges.

Paths A *path* is a trail with no repeated vertices or edges.

Circuits A *circuit* is a closed trail that can have repeated vertices only (but not edges).

Cycle A *cycle* is a closed path where a vertex is reachable from itself.

Euler circuits and paths

- An *Euler circuit* in a graph G is a simple circuit containing every **edge** of G exactly once (and reaching the original vertex).
- An *Euler path* is a path that follows every **edge** of G exactly once.

Hamilton circuits and paths

- A *Hamilton circuit* in a graph G is a simple circuit containing every **vertex** of G exactly once (except the start vertex, which is contained twice).
- A *Hamilton path* is a path that reaches every **vertex** of G exactly once.

8.2 Connectivity

- An **undirected graph** is called *connected* if there exists a path between any two nodes in the graph.
- A **directed graph** is *strongly connected* if there exists a path from a to b **and** b to a whenever a and b are vertices in the graph.
- A **directed graph** is *weakly connected* if the underlying undirected graph is connected.

8.3 Degree Sequence of a Graph

Two vertices are called *adjacent* in a graph of they are endpoints of a single edge. Such an edge is called *incident.* The *degree of a vertex in an undirected graph* is the number of edges incident with it, except that a loop is counted as two edges. The degree of a vertex is denoted as deg(v).

Handshaking Theorem Let $G = (V, E)$ be an undirected graph with m edges. Then

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

It states that the the number of edges is half the sum of all degree vertices. This theorem also shows that the number of edges in an undirected graph is **always even**.

In- and Out-Degree In a graph with *directed* edges the *in-degree* denoted by $deg^{-}(v)$ is the number of edges with v as the terminal vertex. The *out-degree* denoted by $deg^{+}(v)$ is the number of edges with v as the initial vertex. Because each edge is an outgoing or incoming edge of a vertex, the sum of all in-degrees and out-degrees is equal to the sum of all edges.

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

8.4 Properties of Graphs

A **simple graph** with n vertices can at most have vertices of degree $n-1$. For example, a graph with 4 vertices and a degree sequence `4,3,3,2` cannot be simple because it contains one vertex with degree `4` which is greater than $n-1=3$.

Regular Graphs A simple graph is called **regular** if every vertex of this graph has the same degree. A regular graph is called n-regular if every vertex has degree n.

Complete Graphs A complete graph K_n on n vertices is a simple graph that contains exactly one edge between each pair of distinct vertices.

8.5 Isomorphism

Two graphs $G = (V_1, E_1)$ and $H = V_2, E_2$ are *isomorphic* if there exists a one-to-one correspondence (bijective relationship) between their sets of vertices V_1 and V_2. Isomorphic graphs have the following properties, called **graph invariants**.

- The number of vertices is the same (by definition)
- The number of edges is the same
- The degrees of the vertices are the same
- The degrees of adjacent vertices must also be the same For example, if a vertex in V_1 with deg(2) is adjacent to a vertex with deg(3), then the corresponding vertex in V_2 must also have deg(2) and be adjacent to some vertex with deg(3).

8.5.1 Adjacency Matrices

Even if the invariants between two graphs seem to hold, isomorphism can be demonstrated using an adjacency matrix.

First, we need to establish the function f between two graphs and map each vertex in the first graph to a valid counterpart in the second graph (same degree, same degree in adjacent vertices).

Second, we write a matrix for all vertices in the first graph and denote with `1` and `0` whether they are adjacent.

We write the matrix for the second graph listing the corresponding vertices in the right order. For example, if the vertices u_1, u_2, u_3 corresponded to $f(u_1) = v_3, f(u_2) = v_1, f(u_3) = v_2$, the matrix structure would look as follows.

$$\begin{array}{c|ccc|} & u_1 & u_2 & u_3 \\ u_1 & 0 & 1 & 1 \\ u_2 & 1 & 0 & 0 \\ u_3 & 1 & 0 & 0 \end{array} \longrightarrow \begin{array}{c|ccc|} & v_3 & v_1 & v_2 \\ v_3 & 0 & 1 & 1 \\ v_1 & 1 & 0 & 0 \\ v_2 & 1 & 0 & 0 \end{array}$$

The sum of the degree sequence of an *undirected* graph is equal to the sum of the elements in the adjacency matrix. The sum of the edges of a *directed* graph is equal to the sum of the elements in the adjacency matrix.

8.6 Bipartite Graphs

A graph is a *bipartite graph* if its vertex set can be partitioned into two disjoint sets V_1 and V_2 such that every edge in the graph connects a vertex in V_1 to a vertex in V_2. A simple graph is bipartite if and only if it is possible to assign one of two colours to each vertex of the graph so that no two adjacent vertices are assigned the same colour. The number of vertices in V_1 and V_2 may differ, however the sum of degrees in V_1 is equal to the sum of degrees in V_2, i.e. $|V_1| = |V_2|$.

8.6.1 Matchings

A *matching* M in a simple graph $G = (V, E)$ is a subset of the set E of edges of the graph such that no two edges are incident with the same vertex. In other words, we pick the unique edges that exclusively link two vertices.

A *complete matching* from V_1 to V_2 is given if every vertex inV_1 is an endpoint of an edge in the matching, giving $|M| = |V_1|$. A *maximum matching* is the largest possible number of edges in the matching.

8.7 Dijkstra's Algorithm

A *weighted graph* is a graph that has a number assigned to each edge. This can signify the cost of an edge in a given path between two vertices. The *length* of a path is the sum of the weights of the edges in the path. When seeking the *shortest path*, one method is to use **Dijkstra's algorithm** as shown in Algorithm 1.

Algorithm 1 Dijkstra's Algorithm

```
function DIJKSTRA($G$)
    $L(v_i) \leftarrow \infty$
    $L(a) \leftarrow$ undefined
    $S \leftarrow \emptyset$
    while $z \notin S$ do
        $u \leftarrow$ a vertex $\notin S$ with $L(u)$ minimal
        $S \leftarrow S \cup \{u\}$
        for all vertices $v \notin S$ do
            if $L(u) + w(u, v) < L(v)$ then
                $L(v) \leftarrow L(u) + w(u, v)$
            end if
        end for
    end while
    return $L(z)$
end function
```

9 Trees

9.1 Trees and Properties of Trees

A connected, undirected graph that contains no simple circuits is called a **tree**. Graphs that have no circuits but are not connected are called **forests** where each unconnected component is a tree. An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.

A tree with n vertices has $n - 1$ edges.

9.2 Spanning Trees

A **spanning tree** of a graph is a subgraph that is a tree containing every vertex of G. A simple graph is connected if and only if it has a spanning tree. A **minimum spanning tree** in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges. Two algorithms are commonly used to find a minimum spanning tree.

9.2.1 Prim's Algorithm

Choose any edge with smallest weight and place it in the spanning tree. Add edges to the tree that are incident to a vertex already in the tree provided they do not form a simple circuit with edges already in the tree. Stop when n-1 edges have been added, meaning all vertices of a graph G are in the minimum spanning tree T.

Algorithm 2 Prim's Algorithm

```
function PRIM($G$: weighted connected undirected graph with $n$ vertices)
    $T \leftarrow$ a minimum-weight edge
    for $1 \leq i \leq n-2$ do (we loop through $n$ vertices and two vertices are already in $T$ )
        $e \leftarrow$ an edge of minimum weight incident to a vertex in $T$ and not forming a simple circuit in $T$ if added to $T$
        $T \leftarrow T$ with $e$ added
    end for
    return $T$ ($T$ is a minimum spanning tree of $G$ )
end function
```

9.2.2 Kruskal's Algorithm

An edge with smallest weight is chosen and added to the tree. Successively, further edges with smallest weight are added, and they may not be connected to vertices already in the tree. This algorithm uses the tree property that a tree with n vertices has $n-1$ edges . Therefore, once $n-1$ edges have been added, we know that all vertices have been included in the spanning tree.

Algorithm 3 Kruskal's Algorithm

```
function KRUSKAL($G$: weighted connected undirected graph with $n$ vertices)
    $T \leftarrow$ empty graph
    for $1 \leq i \leq n-1$ do
        $e \leftarrow$ any edge in $G$ with smallest weight that does not form a simple circuit when added to $T$
        $T \leftarrow T$ with $e$ added
    end for
    $T \leftarrow$ ($T$ is a minimumg spanning tree of $G$ )
end function
```

9.3 Rooted Trees

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 converts the tree from an undirected graph to a directed one.

- A vertex of a rooted tree is called a **leaf** if it has no children. Vertices that have children are called **internal vertices**.

- A rooted tree is called an *m-ary tree* if every internal vertex has no more than m children.

- The tree is called a *regular* or *full m-ary tree* if every internal vertex has exactly m children. A tree with $m = 2$ is a *binary tree.*

- A *regular m-ary tree* with i internal vertices contains $mi + 1$ total vertices.

- There are at most m^h leaves in an m-ary tree of height h.
- The *height* of a tree is the longest path from its root to a leaf. The height any single node is the longest path from that node to a leaf.
- The *depth* of a node is the number of edges a node is away from the root.

A *balanced tree* is a rooted m-ary tree of height h where all leaves are at levels h or $h - 1$.

9.3.1 Binary Search Trees

A *binary search tree* is a *balanced* binary tree that is used for performing the **binary search algorithm**. The height of a binary search tree with n leaves is given by

$$h = \lceil \log_2 l + 1 \rceil$$

The binary search algorithm is shown in Algorithm 4.

Algorithm 4 Binary Search

```
function BinarySearch($v$, $item$)
    $L \leftarrow 1$
    $R \leftarrow \text{LENGTH}[v]$
    while $L \leq R$ do
        $M \leftarrow \lfloor \frac{L+R}{2} \rfloor$
        if $v[M] < item$ then
            $L \leftarrow M + 1$
        else if $v[M] > item$ then
            $R \leftarrow M - 1$
        else
            return $M$
        end if
    end while
    return false
end function
```

10 Relations

A **binary relation** from a set A to a set B is the set R of ordered pairs, where the first element of each ordered pair comes from A and the second element comes from B. The relation is a subset of $A \times B$.

Functions as Relations In a function f from set A to B, the graph of f is a subset of $A \times B$ and can be viewed as a relation R where every element of A is the first element of exactly one tuple in R. A relation in general can describe a one-to-many relationship, but a relation as a function describes a one-to-one relationship (similar to an injective function).

Relation on a set A relation on a set A is a subset of $A \times A$. On a set with n elements, there are 2^{n^2} relations.

Matrix representation A relation can be represented as a matrix $\mathbf{M}_R = [m_{ij}]$ with the properties

$$m_{ij} = \begin{cases} 1 \text{ if } (a_i, b_j) \in R \\ 0 \text{ if } (a_i, b_j) \notin R \end{cases}$$

Digraph representation The relation R on a set A can be represented by the **directed graph** that has the elements of A as its vertices and the ordered pairs $(a, b) \in R$ as its edges.

10.1 Properties of Relations

10.1.1 Reflexivity

A relation R on a set A is called *reflexive* if $\forall a((a, a) \in R))$.

Matrix representation R is reflexive if all the elements on the main diagonal of $\mathbf{M}_R$ are equal to 1, i.e.

$$m_{ii} = 1 \text{ for } i = 1, 2, \ldots, n$$

The *join* of two matrices $\mathbf{A}$ and $\mathbf{B}$ is the zero-one matrix with (i, j)th entry $a_{ij} \vee b_{ij}$. The *meet* of two matrices $\mathbf{A}$ and $\mathbf{B}$ is the zero-one matrix with (i, j)th entry $a_{ij} \wedge b_{ij}$.

The intersection or union of two relations can be represented using the meet and join operations.

$$\mathbf{M}_{R_1 \cup R_2} = \mathbf{M}_{R_1} \vee \mathbf{M}_{R_2} \text{ and } \mathbf{M}_{R_1 \cap R_2} = \mathbf{M}_{R_1} \wedge \mathbf{M}_{R_2}$$

Digraph representation R is reflexive if its digraph has a **loop** at every vertex, so that every ordered pair (x, x) appears in the graph.

10.1.2 Symmetry

A relation R on a set A is *symmetric* if $\forall a \forall b((a, b) \in R \to (b, a) \in R)$.

Matrix representation R is symmetric if and only if

$$\mathbf{M}_R = (\mathbf{M}_R)^T$$

That is, a relation is symmetric if and only if it's matrix is symmetric and is equal to its own transposed form. It is *antisymmetric* if either $m_{ij} = 0$ or $m_{ji} = 0$ when $i \neq j$.

Digraph representation R is symmetric if and only if for every edge between distinct vertices in its digraph there is an edge in the opposite direction, so that (y, x) is in the relation whenever (x, y) is in the relation. It is *antisymmetric* if and only if there are never two edges in opposite directions between distinct vertices.

10.1.3 Transitivity

A relation R on a set A is *transitive* if $\forall a \forall b \forall c(((a, b) \in R \wedge (b, c) \in R) \rightarrow (a, c) \in R)$.

Digraph representation R is transitive if and only if whenever there is an edge from a vertex x to a vertex y and an edge from a vertex y to a vertex z , there is an edge from x to z.

10.2 Equivalence Relations

A relation on a set A is called an *equivalence relation* if it is reflexive, symmetric, and transitive. Two elements that are related through an equivalence relation are called *equivalent*, noted as a b.

Congruence Modulo m Let m be an integer with $m > 1$. The relation

$$R = \{(a, b) \mid a \equiv b \pmod{m}\}$$

is an equivalence relation on the set of integers with m equivalence classes.

10.2.1 Equivalence Classes

The set of all elements that are related to an element a of A is called the *equivalence class* of a, denoted as

$$[a]_R = \{s \mid (a, s) \in R\}$$

If R is an equivalence relation, then

$$(i)\ aRb \qquad (ii)\ [a] = [b] \qquad (iii)\ [a] \cap [b] \neq \emptyset$$

In other words, if a and b are in the equivalence relation, then their equivalence classes are identical and cannot be disjoint. If they are not related in the equivalence relation, then they have their own equivalence class and are a disjoint partition.

An equivalence relation partitions a set into equivalence classes. The union of all equivalence classes is the original set.

$$\bigcup_{a \in A} [a]_R = A$$

In a graph, you can identify equivalence classes by identifying subgraphs that are not connected to other graphs.

10.3 Partial Orderings

A relation R on a set S is called a *partial ordering* if it is reflexive, antisymmetric, and transitive. A set S together with a partial ordering R is called a *partially ordered set* denoted as (S, R). When every two elements of a set are comparable, the relation is called a **total ordering**.

11 Combinatorics

Product Rule Suppose that a procedure can be broken down into a sequence of two tasks. If there are n_1 ways to do the first task and for each of these ways of doing the first task, there are n_2 ways to do the second task; then there are $n_1 n_2$ ways to do the procedure.

Sum Rule If a task can be done either in one of n_1 ways or in one of n_2 ways, where none of the set of n_1 ways is the same as any of the set of n_2 ways, then there are $n_1 + n_2$ ways to do the task.

Subtraction Rule If a task can be done in either n_1 ways or n_2 ways, then the number of ways to do the task is $n_1 + n_2$ minus the number of ways to do the task that are common to the two different ways. This is the same as the **principle of inclusion-exclusion** (see Discrete Mathematics — Set Theory).

11.1 Pigeonhole Principle

If N objects are placed into k boxes, then there is **at least** one box with **at least** $\lceil N/k \rceil$ objects.

Important: When calculating the number of objects N required to satisfy a specific $\lceil N/k \rceil$ outcome, keep in mind that you are rounding up N/k. For example, if $\lceil N/k \rceil = 6$ and $k = 4$, then

$$\begin{aligned} \lceil N/4 \rceil = 6 \Leftrightarrow N/4 &> 5 \\ N &> 20 \\ N &= 21 \end{aligned}$$

Therefore, when solving the inequality, you can add one to the result for N to arrive at the smallest number that will satisfy the desired N/k (if that is what is asked).

11.2 Permutations

A permutation of n different elements is an ordering of the elements such that one element is first, one is second, one is third, and so on.

The number of permutations of n elements is

$$n \cdot (n-1) \cdot \cdot 4 \cdot 3 \cdot 2 \cdot 1 = n!$$

In other words, there are $n!$ different ways of ordering n elements.

The number of permutations of n elements taken r at a time **without repetition** is

$$ {}_nP_r = \frac{n!}{(n-r)!} = n(n-1)(n-2)\cdots(n-r+1) $$

The number or permutations of n elements taken r at a time **with repetition allowed** inclusion

$$ n^r $$

Consider a set of n objects that has n_1 of one kind of object, n_2 of a second kind, and so on. The number of **distinguishable permutations** of the n objects is

$$ \frac{n!}{n_1! \cdot n_2! \cdot n_3! \cdot \cdots \cdot n_k!} \tag{1} $$

This is equivalent to the number of ways n distinguishable objects can be placed into k boxes.

11.3 Combinations

Combinations consider only the possible sets of objects *regardless* of the order in which the members of the set are arranged.

The number of possible combinations of n elements taken r at a time **without repetition** is

$$ {}_nC_r = \frac{n!}{(n-r)!r!} = \frac{{}_nP_r}{r!} \tag{2} $$

The number of possible combinations of n elements taken r at a time **with repetition** is

$$ {}_nC_r = \frac{(n+r-1)!}{(n-1)!r!} $$

11.4 Binomial Theorem

The number of r-combinations from a set with n elements can be denoted as $\binom{n}{r}$. A **binomial** expression is the sum of two terms. The **binomial theorem** states that given two variables x and y, and a nonnegative integer n,

$$ (x+y)^n = \sum_{j=0}^{n} \binom{n}{j} x^{n-j}y^j = \binom{n}{0} x^n + \binom{n}{1} x^{n-1}y + \cdots + \binom{n}{n-1} xy^{n-1} + \binom{n}{n} y^n $$

Note that the exponent of n decreases with each term as the exponent of y increases with each term. Also note that if x or y is negative, then each alternating term will be negative by moving the negative sign from the negative term to the front (e.g. $x(-y)^3 = -xy^3$).