ASSIGNMENT FINAL REPORT

Qualification	**Pearson BTEC Level 5 Higher National Diploma in Computing**		
Unit number and title	**Unit 18: Discrete Maths**		
Submission date	16/12/2025	**Date Received 1st Submission**	
Re-submission Date		**Date Received 2nd Submission**	
Student Name	HO DUC DUONG	**Student ID**	BD00535
Class	SE07202	**Assessor name**	Nguyen Thi Su

Plagiarism

Plagiarism is a particular form of cheating. Plagiarism must be avoided at all costs and students who break the rules, however innocently, may be penalised. It is your responsibility to ensure that you understand correct referencing practices. As a university level student, you are expected to use appropriate references throughout and keep carefully detailed notes of all your sources of materials for material you have used in your work, including any material downloaded from the Internet. Please consult the relevant unit lecturer or your course tutor if you need any further advice.

Student Declaration

I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I declare that the work submitted for assessment has been carried out without assistance other than that which is acceptable according to the rules of the specification. I certify I have clearly referenced any sources and any artificial intelligence (AI) tools used in the work. I understand that making a false declaration is a form of malpractice.

Student's signature	DUONG

Grading grid

P1	P2	P3	P4	M1	M2	D1	D2

ASSIGNMENT GROUP WORK

Qualification	**Pearson BTEC Level 5 Higher National Diploma in Computing**		
Unit number and title	**Unit 18: Discrete Maths**		
Submission date	16/12/2025	**Date Received 1st submission**	
Re-submission Date		**Date Received 2nd submission**	
Group number:	**Student names & codes**	**Final scores**	**Signatures**
	Ho Duc Duong – BD00535		DUONG
	Phu Tuong Long – BD00772		LONG
	Huynh Nguyen Ngoc Hoang – BD00698		HOANG

Class	SE07202	**Assessor name**	Nguyen Thi Su

P5	P6	P7	P8	M3	M4	D3	D4

OBSERVATION RECORD

Student	**Ho Duc Duong – BD00535**

Description of activity undertaken

Completed all tasks related to Chapter 4 (LO4): Explore applicable concepts within abstract algebra. Specific activities included

(P7): Described the distinguishing characteristics of binary operations (Closure, Associativity, Commutativity, Identity, Inverse) and verified whether specific operations on sets $\mathbb{N}$ and qualify as $\mathbb{Z}$ binary operations

(P8): Determined the order of groups and subgroups. Constructed Cayley tables for groups of orders 1, 2, 3, and 4. Applied Lagrange's Theorem to discuss the possibility of a subgroup of order 4 existing within a group of order 9

(M4): Validated whether the set $S = \mathbb{R}\backslash\{-1\}$ under a specific operation constitutes a group by verifying group axioms and calculated the total number of possible binary operations on a finite set

Assessment & grading criteria

How the activity meets the requirements of the criteria

Student signature:	**DUONG**	**Date:**	**16/12/2025**

Assessor signature:		**Date:**	
Assessor name:			

❐ **Summative Feedback:** ❐ **Resubmission Feedback:**

Grade:	**Assessor Signature:**	**Date:**

Internal Verifier's Comments:

Signature & Date:

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

In the rapidly advancing realm of technology, discrete mathematics serves as the fundamental backbone of computer science, playing a crucial role in algorithm design, data structures, and system logic. This report explores the essential components of mathematical modeling and analysis, with a particular focus on applying these abstract concepts to solve practical problems in software engineering

The report is structured as follows

Chapter 1: Examine set theory and functions applicable to software engineering This chapter serves as the foundation of our mathematical exploration. Key discussions include

- Performing algebraic set operations and utilizing Venn diagrams to visualize relationships

- Determining the cardinality of bags (multisets) and analyzing the inverse of functions using appropriate techniques

- Formulating corresponding proof principles to validate properties about defined sets

Chapter 2: Analyse mathematical structures of objects using graph theory The focus shifts to structural modeling and optimization

- Modeling contextualised problems using trees, both quantitatively and qualitatively

- Using Dijkstra's algorithm to find a shortest path spanning tree in a weighted graph

- Assessing whether Eulerian and Hamiltonian circuits exist within undirected graphs and constructing proofs for graph coloring

Chapter 3: Investigate solutions to problem situations using the application of Boolean algebra To validate logical decision-making in electronic systems

- Diagramming binary problems and designing complex systems using logic gates

- Producing truth tables and deriving corresponding Boolean equations from applicable scenarios

- Simplifying complex Boolean equations using algebraic methods to optimize circuit design

Chapter 4: Explore applicable concepts within abstract algebra Analyzing fundamental algebraic structures and operations

- Describing the distinguishing characteristics of different binary operations performed on the same set

- Determining the order of a group and subgroup, and applying Lagrange's Theorem

- Validating whether a given set with a binary operation qualifies as a group structure

The aim of this report is to bridge theoretical discrete mathematical concepts with practical computing application, providing insights for building efficient algorithms and logical systems

CHAPTER 1 – LO1: EXAMINE SET THEORY AND FUNCTIONS APPLICABLE TO SOFTWARE ENGINEERING

1.1 (P1) Perform algebraic set operations in a formulated mathematical problem

1.1.1 Definition of Sets and Set Notation

In mathematics, a set is defined as an unordered collection of distinct objects. These objects are called the elements or members of the set. (Rosen, 2019)

The core characteristics of a set are "distinctness" (elements are not repeated) and "unordered" (the arrangement of elements does not change the set).

Representation and Notation

There are two common ways to represent a set:

- Roster Method: This method involves listing all the elements of the set, enclosed in curly braces {} and separated by commas.

$A = \{1, 2, 3, 4, 5\}$ is the set of the first five positive integers

$V = \{a, e, i, o, u\}$ is the set of vowels in the English alphabet

- Set-Builder Notation: This method describes the elements of the set by stating one or more properties they must satisfy.

$B = \{x \mid x\ is\ an\ even\ integer\ and\ 0 < x < 10\ \}$ Read as: B is the set of all x such that x is an even integer and x is greater than 0 and less than 10. Thus, $B = \{2, 4, 6, 8\}$

$Q = \left\{\frac{p}{q}\middle|\, p\ \in Z, q\ \in Z, and\ q \neq 0\right\}$ is the set of rational numbers.

Basic symbols

$a\ \in A$: : "a is an element of set A"

$b\ \notin A$: "$b\ is\ not\ an\ element\ of\ set\ A$"

$\emptyset$ $(or\ \{\ \})$ $: The\ Empty\ set, which\ is\ the\ set\ containing\ no\ elements$

U: The Universal Set, which is the set containing all objects under consideration in a specific context (Epp, 2019)

$A\ \subseteq B$: Subset. A is a subset of B if every element of A is also an element of B

$A \subset B$**:** Proper Subset. A is a subset of B, but $A \neq B$

1.1.2 Basic Set Operations and Symbols

Let A and B be two subsets of the universal set U

Union: $A \cup B$ The set containing all elements that are in A, or in B, or in both.

Set-Builder Notation: $A \cup B = \{x \in U \mid x \in A \; or \; x \in B\}$

Example: If $A = \{1, 2, 3\} \; and \; B = \{3, 4, 5\}, \; then \; A \cup B = \{1, 2, 3, 4, 5\}$

Intersection: $A \cap B$ The set containing all elements that are in both A and B.

Set-Builder Notation: $A \cap B = \{x \in U \mid x \in A \; and \; x \in B\}$

Example: If $A = \{1, 2, 3\} \; and \; B = \{3, 4, 5\}, then \; A \cap B = \{3\}$

Note: If $A \cap B = \emptyset, A \; and \; B \; are \; called \; disjoint \; (Grimaldi, 2017)$

Difference: $A - B$ The set containing all elements that are in A but not in B.

Set-Builder Notation: $A - B = \{x \in U \mid x \in A \; and \; x \notin B\}$

Example: IF $A = \{1, 2, 3\} \; and \; B = \{3, 4, 5\}, \; then \; A - B = \{1, 2\}$

Complement: $\bar{A}$ $(A^c \; or \; A')$: The set containing all elements in the universal set U that are not in A. This is a special case of the difference $(U - A)$.

Set-Builder Notation: $\bar{A} = \{x \in U \mid x \notin A\}$

Example: If $U = \{1, 2, 3, 4, 5\} \; and \; A = \{1, 2, 3\}, then \; \bar{A} = \{4, 5\}$

1.1.3 Venn Diagrams for Set Operations

Venn Diagrams are powerful visualization tools for representing relationships and operations between sets. They typically use circles (representing sets) inside a rectangle (representing the universal set U) (Epp, 2019)

- Union $(A \cup B)$: The entire shaded area of both circles A and B

- Intersection $(A \cap B)$: The common area (the overlap) between circles A and B

- Difference $(A - B)$: The area that is only in circle A, not including the overlapping part with B

- Complement $(\bar{A})$: The entire area inside the rectangle U but outside of circle A

1.1.4 Cartesian Product of Sets

The Cartesian Product of two sets A and B, denoted $A \times B$, is the set of all ordered pairs (a, b) where $a \in A \; and \; b \in B$ (Rosen, 2019)

Set builder notation: $A \times B = \{(a, b) | a \in A \; and \; b \in B\}$

A key point is that order matters: $(a, b) \neq (b, a)$ unless $a = b$. Therefore, the Cartesian Product is not commutative, meaning $A \times B \neq B \times A$ (unless $A = B$ or one of the sets is empty)

Example: Suppose we have two sets

$A = \{Rice, Noodles\}$ (set of main courses)

$B = \{Tea, Coffee\}$ (set of drinks)

The Cartesian Product $A \times B$ represents all possible combinations of a main course and a drink: $A \times B = \{(Rice, Tea), (Rice, Coffee), (Noodles, Tea), (Noodles, Coffee)\}$

If $A = \{1, 2\} \; and \; B = \{a, b, c\}, then \; A \times B = \{(1, a), (1, b), (1, c), (2, a), (2, b), (2, c)\}$ Noete that the cardinality (size) is $|A \times B| = |A| \times |B| = 2 \times 3 = 6$

1.1.5 Applications in Computing Context

Set theory is a fundamental basis for many areas in computer science

Relational databases

- SQL (Structured Query Language) is built upon relational algebra, which is an extension of set theory (Epp, 2019)

- The “UNION” command in SQL directly corresponds to the set union operation

- The “INTERSECT” command corresponds to the intersection operation

- The “EXCEPT” (or “MINUS” in Oracle) command corresponds to the set difference

- A “JOIN” operation is essentially a Cartesian Product filtered by a predicate (a condition)

Data structures

Many programming languages (like Python, Java, C++) provide “Set” and “Map” (based on ordered pairs) data structures. These structures are optimized for fast operations like membership testing ($x \in A$), addition, deletion, and the operations of union, intersection, and difference (Grimaldi, 2017)

Hash Tables are a common implementation of the set data structure, allowing for an average time complexity of $O(1)$ for lookups

Formal Languages and Automata Theory

An alphabet is defined as a finite set of characters (e.g., $\sum = \{0, 1\}$)

A language is defined as a (possibly infinite) set of strings formed from that alphabet (Rosen, 2019). Operations like union, intersection, and a variation of the Cartesian product (called concatenation) are fundamental to building regular expressions and automata

Artificial Intelligence

In Fuzzy Logic, Fuzzy Sets are an extension of classical set theory, allowing an element to "belong" to a set with a certain degree of membership (from 0 to 1), rather than just the binary "belongs" (1) or "does not belong" (0)

1.1.6 Activity 1 – Part 1

Stick in mind that $a < b$represents the largest digits in your ID.

1. Let $A$and Bbe two non-empty finite sets. Assume that cardinalities of the setsA, B and $A \cap B$are $\overline{9b}, \overline{2a}$ and $a + b$, respectively. Determine the cardinality of the set $A \cup B$.

2. Suppose $|A - B| = \overline{3a}, |A \cup B| = \overline{11b}$ and $|A \cap B| = \overline{1a}$. Determine $|B|$.

3. At a local market, there are $\overline{35b}$ customers. Suppose $\overline{11a}$ have purchased fruits, $\overline{9b}$ have purchased vegetables, $\overline{8a}$ have purchased bakery items, $\overline{4b}$have purchased both fruits and vegetables, $\overline{3b}$ have purchased both vegetables and bakery items, $\overline{2a}$ have purchased both fruits and bakery items, and $\overline{1a}$ have purchased all three categories. How many customers have not purchased anything?

Solution

My student ID is BD00535, and since a < b, we have a = 3 and b = 5

1. Given information:

$$|A| = \overline{9b} = 95$$

$$|B| = \overline{2a} = 23$$

$$|A \cap B| = a + b = 3 + 5 = 8$$

We have:

$$|A \cup B| = |A| + |B| - |A| \cap |B| = 95 + 23 - 8 = 110$$

Result:

$$|A \cup B| = 110$$

2. Given information:

$$|A - B| = \overline{3a} = 33$$

$$|A \cup B| = \overline{11b} = 115$$

$$|A \cap B| = \overline{1a} = 13$$

$$|A| = |A - B| + |A \cap B| = 33 + 13 = 46$$

From the combined formula, we have:

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

$$115 = 46 + |B| - 13 \rightarrow |B| = 115 - 46 + 13 = 82$$

Result:

$$|B| = 82$$

3. Given information:

$$|B| = \overline{35b} = 355$$

$$|F| = \overline{11a} = 113$$

$$|V| = \overline{9b} = 95$$

$$|B| = \overline{8a} = 83$$

$$|V \cap F| = \overline{4b} = 45$$

$$|V \cap B| = \overline{3b} = 35$$

$$|B \cap F| = \overline{2a} = 23$$

$$|F \cap V \cap B| = \overline{1a} = 13$$

The number of customers who bought at least one type:

$$|\overline{F \cup B \cup V}| = |F| + |B| + |V| - |F \cap B| - |F \cap V| - |B \cap V| + |F \cap B \cap V|$$

Substituting the values, we have:

$$|\overline{F \cup B \cup V}| = 113 + 83 + 95 - 23 - 45 - 35 + 13 = 201$$

The number of customers who bought nothing is:

$$|U| - |F \cup B \cup V| = 355 - 201 = 154$$

Result: There are 154 customers who bought nothing

1.2 (P2) Determine the cardinality of a given bag (multiset)

1.2.1 Definition of bag

A "bag" (or more commonly, a multiset) is a collection of objects, similar to a set, but which allows for repeated elements (Grimaldi, 2017)

The key difference compared to a regular set is

- Set: Unordered, elements are distinct.

Example: $\{1, 2, 3\}$

- Bag (Multiset): Unordered, elements can be repeated

Example: $\{|1, 1, 2, 3, 3, 3|\}$

Each element in a bag has multiplicity, which indicates the number of times that element appears in the bag

Notation and Examples

Bags are often represented with curly braces {}, similar to sets, but the context will make it clear that repetition is important

Example 1: $A = \{|a, a, b, c, c, c|\}$

- The multiplicity of ‘a’ is 2

- The multiplicity of ‘b’ is 1

- The multiplicity of ‘c’ is 3

Example 2: $B = \{|1\ , 2, 3|\}$. This is both a set and a bag where each element has a multiplicity of 1

Example 3: $C = \{|apple, orange, apple, pcar|\}$

Sometimes, to explicitly represent the multiplicity, pairs of (element, multiplicity) are used. For example, bag A above can be written as $\{|a{:}\,2, b{:}\,1, c{:}\,3|\}$

1.2.2 Operations on Bags

Operations on sets can be extended to bags, but they must account for the multiplicity of the elements (Rosen, 2019). Let A and B be two bags. Let $M_A(x)$ be the multiplicity of x in A and $M_B(x)$ be the multiplicity of x in B

Union (or Sum): The union of two bags (sometimes called the sum, denoted ∪ or+) creates a new bag where the multiplicity of each element is the *sum* of its multiplicities in the original two bags $(M_{A \uplus B}(x) = M_A(x) + M_B(x))$ (Rosen, 2019)

Example:

$$A = \{|a, a, b, c|\}$$
$$B = \{|a, b, b, d|\}$$
$$A \cup B = \{|a, a, a, b, b, b, c, d|\}$$

Intersection: The intersection of two bags creates a new bag where the multiplicity of each element is the *minimum (min)* of its multiplicities in the original two bags $M_{A\,\cap B}(x) = (M_A(x), M_B(x))$ (Rosen, 2019)

Example:

$$A = \{|a, a, b, c|\}$$
$$B = \{|a, b, b, d|\}$$
$$A\ \cap B = \{|a\ , b|\}$$

Difference: The difference $A - B$ creates a new bag where the multiplicity of x is its multiplicity in A *minus* its multiplicity in B, but never less than zero $M_{A-B}(x) = (0, M_A(x) - M_B(x))$ (Rosen, 2019)

Example:

$$A = \{|a, a, b, c|\}$$
$$B = \{|a, b, b, d|\}$$
$$A - B = \{|a, c|\}\ because\ M_A(b) - M_B(b) = 1 - 2 = -1, we\ take\ 0$$

1.2.3 Cardinality of Bags

The Cardinality of a bag, denoted $|A|$, is the total number of elements in the bag, *including repetitions* (i.e., the sum of the multiplicities of all distinct elements) (Epp, 2019)

In other words, the cardinality of a bag is equal to the sum of the multiplicities of all its distinct elements

Formula

If $A = \{|x, x, \dots, x, \dots|\}\ and\ M_A(x_i)$ is the multiplicity of element x_i , then $|A| = \sum_{x_i \in A} M_A(x_i)$ (The sum is taken over all *distinct* elements $x_i\ in\ A$)

Example

- Bag $A = \{|a, a, b, c, c, c|\} \rightarrow M_A(a) = 2, M_A(b) = 1, M_A(c) = 3 \rightarrow |A| = 2 + 1 + 3 = 6$

- Bag $B = \{|apple, orange, apple, pear|\} \rightarrow M_B(apple) = 2, M_B(orange) = 1, M_B(pear) = 1$

$$|B| = 2 + 1 + 1 = 4$$

- Bag $C = \{|5, 5, 5, 5|\} \rightarrow M_C(5) = 4 \rightarrow |C| = 4$

1.2.4 Applications in Computing Context

The concept of a "bag" is very important in computer science because many real-world problems involve collections where repetition is significant

Databases

- The standard SQL language operates on a bag model, not a set model

- When you execute a "SELECT" query, the default result is a bag.

Example, "SELECT age FROM Employees" will return *all* age values, including duplicates (e.g., $\{|25, 30, 25, 40|\}$)

- To force the result to be a set (removing duplicates), the user must use the "SELECT DISTINCT" keyword (Grimaldi, 2017)

Data mining and text processing

- In Natural Language Processing (NLP), the "Bag-of-Words" (BoW) model is a common technique. A text document (like an email or an article) is represented as a bag of its words (Rosen, 2019)

- This model ignores grammar and word order but retains the *multiplicity* (frequency) of each word

Example: The sentence "the cat sat on the mat" is represented by the bag: $\{|the: 2, cat: 1, sat: 1, on: 1, mat: 1|\}$. The cardinality of this bag is 6. This representation is crucial for text classification algorithms (e.g., spam filtering)

Combinatorics and counting

- Many counting problems involve "combinations with repetition," which are essentially problems of counting bags

Example: "How many ways are there to choose 3 pieces of candy from 5 different types, if repetition is allowed?" (e.g., choosing 3 of the same type). This is precisely the problem of finding the number of bags of cardinality 3, drawn from a set of 5 elements

1.2.5 Activity 1 – Part 2

Keep in mind that $a < b$ represents the largest digits in your ID.

1. List the bag of prime factors for each of the numbers provided.

a) $\overline{1a2}$

b) $\overline{2b0}$

2. Find the cardinalities of

a) each of the aforementioned bags.

b) the intersection of the aforementioned bags.

c) the union of the aforementioned bags.

d) the difference of the aforementioned bags.

Solution

My student ID is BD00535, and since a < b, we have a = 3 and b = 5

1. a. With $a = 3$, we have:

$$\overline{1a2} = 100 + 10a + 2 \rightarrow 100 + 10 \times 3 + 2 = 132$$

With 132, we have the following prime factorization: $132 = 2^2 \times 3 \times 11$.

We have: $|A| = \{|2{:}\,2, 3{:}\,1, 11{:}\,1|\}$

b. With $b\ =\ 5$, we have:

$$\overline{2b0} = 200 + 10b + 0\ \rightarrow 200 + 10\ \times 5 + 0 = 250$$

For 250, we have the following prime factorization: $250 = 2\ \times 5^3$

We have: $|B| = \{|2{:}\,1{,}5{:}\,3|\}$

2. Find the cardinalities of the intersection of the bags

We have:

$$|A| = \{|2{:}\,2, 3{:}\,1, 11{:}\,1|\}\ và\ |B| = \{|2{:}\,1{,}5{:}\,3|\}$$

a. The cardinalities of A and B is

$$|A| = 2 + 1 + 1 = 4$$
$$|B| = 1 + 3 = 4$$

b. The intersection of A and B is

$$A\ \cap B = \{|2{:}\,1|\}$$

c. The Union of A and B is

$$A\ \cup B = \{|2{:}\,2, 3{:}\,1, 5{:}\,3, 11{:}\,1|\}$$

d. The difference between A and B is

$$A - B = \{|2{:}\,1, 3{:}\,1, 5{:}\,0, 11{:}\,1|\}$$

1.3 (M1) Determine the inverse of a function using appropriate mathematical techniques

1.3.1 Definition of a Function

A function f from a set A to a set B is an assignment of *exactly one* element of B to *each* element of A

- Notation: We write $f\ : A\ \rightarrow B$ to denote that f is a function from A to B

Definition (from Rosen): "Let A and B be nonempty sets. A function f from A to B is an assignment of exactly one element of B to each element of A" (Rosen, 2019, p. 139)

Key terminology: Assome $f(a) = b, where\ a\ \in A\ and\ b\ \in B$

- Domain: The set A, which is the set of all possible inputs

- Codomain: The set B, which is the set that contains all *possible* outputs

- Range: The set of all *actual* output values of f. The range is a subset of the codomain (Epp, 2019). Denoted as $f(A) = \{f(a) \mid a \in A\}$

- b is called the image of a under f

- a is called the pre-image of b

Example: Given the function f: $\mathbb{Z}$ *to* $\mathbb{Z}$ *(from the set of integers to the set of integers) with* $f(x) = x^2$

- Domain: $\mathbb{Z}$

- Codomain: $\mathbb{Z}$

- Range: $\{0, 1, 4, 9, 16, \dots\}$ (the set of perfect squares)

1.3.2 Definition and Existence of Inverse Functions

Let f be a function from set A to set B. The inverse function of f, denoted $f^{\{-1\}}$ (read "f-inverse"), is a function from B to A that satisfies the condition: If $f(a) = b, then\ f^{-1}(b) = a$ for all $a \in A\ and\ b \in B$

The inverse function $f^{\{-1\}}$ essentially "undoes" the operation of the function f (Rosen, 2019)

Condition for existence: Bijective functions

A function $f: A \rightarrow B$ has an inverse function $f^{-1}: B \rightarrow A$ if and only if f is a bijection (or a one-to-one correspondence) (Epp, 2019)

A function is called a bijection if it satisfies both of the following conditions

Injective (One-to-one)

A function f is injective if it never assigns distinct elements in its domain to the same element in its codomain. That is, if $f(a_1) = f(a_2), then\ a_1 = a_2$

If f *is not one-to-one (e.g.,* $f(1) = 5$ *and* $f(2) = 5$*), then when trying to find* $f^{-1}(5)$*, we cannot determine if the value is 1 or 2. This violates the definition of a function (must assign exactly one value)*

Surjective (Onto)

A function $f: A \rightarrow B$ is surjective if its range is equal to its codomain (i.e., $f(A) = B$). In other words, for every element $b \in B$, there exists *at least one* element $a \in A$ such that $f(a) = b$

If f is not surjective, there will be an element $b \in B$ that no $a \in A$ maps to. In that case, $f^{-1}(b)$ would be undefined. The domain of f^{-1} (which is B) would have an element that is not mapped, violating the definition of a function

Only when a function is both injective and surjective (bijective) can we guarantee that its inverse f^{-1} is also a valid function

1.3.3 Finding the Inverse Algebraically

To find the inverse of a function f (if it exists), we typically follow these algebraic steps (Epp, 2019)

Step 1: Write the equation $y = f(x)$

Step 2: Swap the roles of x and y in the equation. The new equation will be $x = f(y)$. (This step represents the switching of the domain and range)

Step 3: Solve the equation $x = f(y)$ for y in terms of x

Step 4: The new function of y is the inverse function $f^{-1}(x). Write\ y = f^{-1}(x)$

Example: Find the inverse of $f(x) = 3x - 5$ (this is a bijection from $\mathbb{R}\ to\ \mathbb{R}$)

- Step 1: $y = 3x - 5$

- Step 2: $x = 3y - 5$

- Step 3: Solve for y

$$x + 5 = 3y$$
$$y = \frac{x+5}{3}$$

- Step 4: Therefore: $f^{-1}(x) = \frac{x+5}{3}$ Check $f\left(f^{-1}(x)\right) = 3\left(\frac{x+5}{3}\right) - 5 = (x + 5\ ) - 5 = x$

1.3.4 Applications of Inverse Functions in Computing

The concept of inverse functions is fundamental to many processes and algorithms in computer science

Cryptography

This is the most direct application. An Encryption function, E, takes a plaintext (P) and transforms it into a ciphertext (C). $C = E(P)$

The Decryption function, D, is precisely the inverse function of E. It takes the ciphertext C and returns the original plaintext P. $P = D(C) = E^{-1}(C)$

For the system to work, the encryption function E must be invertible (bijective) to guarantee that decryption is unique (Rosen, 2019).

Example, shift ciphers (like the Caesar cipher) $E(x) = (x + k)$ mod n have the inverse $D(x) = (x - k)\ mod\ n$

Data mapping and transformations

In computer graphics, functions are used to transform coordinates from 3D space (model space) to 2D space (screen space) for display

The inverse function is necessary to perform the reverse process: for example, when a user clicks on a 2D point on the screen, the system needs the inverse function to determine which 3D object in the scene was clicked (a process called "picking")

Hashing and One-Way Functions

In contrast to cryptography, security applications (like password storage) use functions that are *not* invertible (or are computationally infeasible to invert)

One-way functions like SHA-256 are designed to be easy to compute $y = f(x)$ but practically impossible to compute $x = f^{-1}(y)$. The *absence* of a feasible inverse function is the key to their security (Grimaldi, 2017)

1.3.5 Activity 1 – Part 3

Bear in mind that $a < b$ represents the largest digits in your ID.

1. Ascertain whether the given functions are invertible. If they are, identify the rule for the inverse function f^{-1}.

a) $f: \mathbb{R} \to \mathbb{R}$ with $f(x) = bx + a$.

b) $f: [-b, +\infty) \to [0, +\infty)$ with $f(x) = \sqrt{x + b}$.

2. Let $f, g: \mathbb{R} \to \mathbb{R}$ be defined as $f(x) = \begin{cases} 2x + a, x < 0 \\ x^3 + b, x \geq 0 \end{cases}$ and $g(x) = bx - a$. Find $g \circ f$

Solution

My student ID is BD00535, and since a < b, we have a = 3 and b = 5

1. a. $f: \mathbb{R} \to \mathbb{R}$ with $f(x) = bx + a$.

Substituting a and b, we have:

$$f(x) = 5x + 3$$

Suppose y is the image of x, we have

b. $f: [-b, +\infty) \to [0, +\infty)$ with $f(x) = \sqrt{x + b}$.

By substituting a and b, we get:

$$f(x) = \sqrt{x + 5}$$

Assuming $y$is the image of x, we have:

$$y = \sqrt{x + 5}$$
$$\Leftrightarrow y^2 = x + 5$$
$$\Leftrightarrow x = y^2 - 5$$

Therefore, the inverse function is $f^{-1}(y) = y^2 - 5$, given that $x \geq 0$

2. We have: $f(x) = \begin{cases} 2x + a, x < 0 \\ x^3 + b, x \geq 0 \end{cases} = \begin{cases} 2x + 3, x < 0 \\ x^3 + 5, x \geq 0 \end{cases}$ *and* $g(x) = bx - a = 5x - 3$

$$\Rightarrow g \circ f(x) = g(f(x)) = \begin{cases} 5(2x + 3) - 3, x < 0 \\ 5(x^3 + 5) - 3, x \geq 0 \end{cases} = \begin{cases} 10x + 12, x < 0 \\ 5x^3 + 22, x \geq 0 \end{cases}$$

CHAPTER 2 – LO2: ANALYSE MATHEMATICAL STRUCTURES OF OBJECTS USING GRAPH THEORY

2.1 (P3) Model contextualised problems using trees, both quantitatively and qualitatively

2.1.1 Definition and Terminology of Trees

In graph theory, a tree is a connected, undirected graph that contains no simple circuits (Rosen, 2019). A key property of a tree is that there is a unique simple path between any two of its vertices

For modeling contextual problems, we often use a rooted tree. 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 (Epp, 2019). This structure imposes a natural hierarchy

Key terminology

- Root: The single designated starting node of the tree

- Parent: The node directly above a given node (connected by an edge moving *away* from the root). Every node except the root has exactly one parent

- Child: A node directly below a given node

- Leaf: A node with no children

- Internal Node: A node that has at least one child

- Subtree: A node and all its descendants (children, children's children, etc.) (Grimaldi, 2017)

- Level: The level of a node is the length of the path from the root. The root is at level 0

- Height: The maximum level of any node in the tree

2.1.2 Qualitative Modeling with Trees

Qualitative modeling involves using trees to represent structure, hierarchy, and relationships. The focus is on the *connections* and *organization* rather than on numerical values

Hierarchical Structures (Taxonomies): Trees are the natural way to model any strict hierarchy

- *Example: File Systems.* The root directory (e.g., / or C:\) is the root. Folders are internal nodes, and files are leaves. The tree structure perfectly represents the "folder-contains-subfolder" relationship (Epp, 2019)

- *Example: Organizational Charts.* The CEO is the root, VPs are children of the root, directors are their children, and so on. The tree qualitatively models the lines of authority

Syntactic Structure (Parse Trees): In linguistics and computer science, parse trees are used to model the grammatical structure of a sentence or a line of code

- *Example:* An Abstract Syntax Tree (AST) is used by a compiler to understand a line of code like $x = (a + b) * 5$. The tree's structure (with * as the root) defines the order of operations, qualitatively representing the code's logic (Grimaldi, 2017)

2.1.3 Quantitative Modeling with Trees

Quantitative modeling involves using trees to calculate a value, make an optimal decision, or measure outcomes. The nodes and edges often carry weights, probabilities, or costs

Decision Trees: These are widely used in machine learning and operations research

A tree where each internal node represents a "test" on an attribute (e.g., "Is Revenue > 1000?"), each edge represents an outcome of the test (e.g., "Yes" / "No"), and each leaf node represents a classification or a final decision (e.g., "Approve Loan") (Rosen, 2019)

Use: The tree provides a quantitative model for making a decision

Expression Trees: Used to model mathematical expressions

A tree where the leaves are operands (numbers or variables) and the internal nodes are operators (+, -, *, /) (Grimaldi, 2017)

Use: The tree can be evaluated quantitatively by traversing it. For example, a "post-order traversal" (Left-Right-Root) of the tree for $(a + b) * c$ yields $a\ b\ +\ c$ *, which is easily computed by a stack

Optimization (Minimum Spanning Trees)

Given a weighted, connected graph (e.g., cities with costs to build roads between them), a Minimum Spanning Tree (MST) is a subtree that connects all vertices (cities) with the minimum possible total edge weight (cost) (Rosen, 2019)

Use: Algorithms like Kruskal's or Prim's find this tree, providing a quantitative solution to an optimization problem

2.1.4 Tree Traversal

To use a tree model, we must "read" or traverse it. The traversal method is key to interpreting the model

Pre-order (Root-Left-Right): Useful for copying a tree or listing the contents of a file system (visit directory, then its contents)

In-order (Left-Root-Right): Used with Binary Search Trees (BSTs). This traversal visits all nodes in their sorted key order, providing a quantitative, sorted output from a qualitative structure (Epp, 2019)

Post-order (Left-Right-Root): Used for evaluating expression trees and for safely deleting all nodes in a tree (you must delete children before their parents)

2.1.5 Applications in Computing Context

The tree structure is one of the most fundamental and widely used data structures in computer science. The concepts of qualitative and quantitative modeling, along with traversal methods, are directly applied in many cores' technologies

Binary Search Trees (BSTs)

Model: A binary tree that adheres to a quantitative property: the value of each node is greater than all values in its left subtree and less than all values in its right subtree

Context: Balanced versions of BSTs (such as AVL trees or Red-Black trees) are the foundation for efficient implementations of lookup data structures (like "map" or "set") in many programming libraries. They allow for searching, inserting, and deleting data in $O(\log\ n)$ average time (Epp, 2019)

Link (from 2.1.4): An In-order (Left-Root-Right) traversal of a BST will return the elements in sorted order (Epp, 2019)

File Systems

Model: As described in 2.1.2, this is a qualitative hierarchical model. The root directory is the tree's root, folders are internal nodes, and files are leaves (Epp, 2019)

Context: The operating system uses this structure to organize, manage, and navigate billions of files on a disk

Link (from 2.1.4): A Pre-order (Root-Left-Right) traversal is the natural method for listing a directory tree (visit the directory, then visit its contents)

Document Object Model (DOM)

Model: A web browser uses a qualitative tree model to represent the structure of an HTML document. The <html> tag is the root, and nested tags (like <body>, <div>) are child nodes, creating a clear hierarchy

Context: JavaScript and CSS use this tree structure to query and manipulate elements on the webpage. CSS selectors are essentially patterns for traversing the DOM tree (Epp, 2019)

Compilers and Syntax Trees

Model: As mentioned in 2.1.2 (Parse Trees) and 2.1.3 (Expression Trees), a compiler parses source code into an Abstract Syntax Tree (AST). This serves as a qualitative model of the code's logic and a quantitative model for expressions (Grimaldi, 2017)

Context: The AST is the central data structure a compiler uses to optimize code and generate machine code or intermediate bytecode

Link (from 2.1.4): A Post-order (Left-Right-Root) traversal is used to correctly evaluate expression trees (Grimaldi, 2017)

Computer Networks

Model: Routing algorithms and network design often use quantitative tree models, especially the Minimum Spanning Tree (MST) (described in 2.1.3)

Context: The Spanning Tree Protocol (STP) is used in network switches to create a logical tree, eliminating loops that can cause broadcast storms. Furthermore, MST-finding algorithms (like Prim's or Kruskal's) are used to design physical networks (e.g., laying fiber) to connect all nodes with minimal total cost (Rosen, 2019)

Artificial Intelligence (AI) and Game Trees

Model: A game tree is a quantitative model used in AI to analyze games (like chess or checkers). The root is the current board state, child nodes are possible moves, and leaves are end-game states (win/lose/draw)

Context: Algorithms like Minimax traverse this tree, assigning "scores" (quantitative values) to states. By propagating these scores upward, the AI can determine the optimal move (the one leading to the subtree with the best value) (Rosen, 2019)

2.1.6 Activity 2 – Part 1

Discuss two notable instances of binary trees, providing both quantitative and qualitative analyses.

Solution

Instance 1: Expression Tree (Mathematical Expression Tree)

Modeling the mathematical expression

$$E = (a + b) \times (c - d)$$

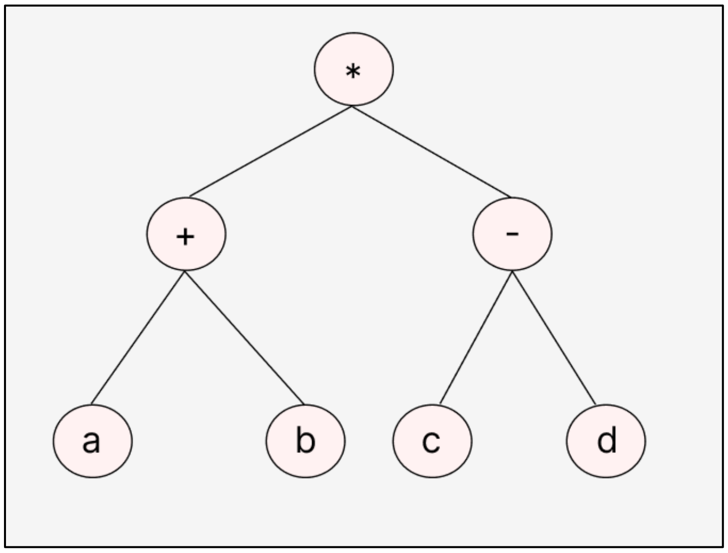


Figure 1: Expression Tree

Root: The multiplication operator ($*$)

Left child of the root: The addition operator $+$, connected to the leaves $a\ and\ b$

Right child of the root: The subtraction operator $-$, connected to the leaves $c\ and\ d$

Qualitative Analysis

Hierarchical structure: The tree represents the order in which the operations are performed. Operators located lower in the tree (closer to the leaves) have higher precedence or are enclosed in parentheses

Role of nodes

- Leaves: Represent operands, which are the variables a, b, c, d. They do not perform any operation

- Internal nodes: Represent operators such as $+$, $-$, and $*$.

- Root: Represents the final operation executed to produce the result of the entire expression

Quantitative Analysis

Tree type: This is a Full Binary Tree, because each binary operator has exactly two children

Number of nodes

- Number of leaves $l = 4$(nodes a, b, c, d)

- Using the full binary tree property (for $m = 2$)

$$i = l - 1 = 4 - 1 = 3$$

- The three internal nodes correspond to the operators $+, -, *$

- Total number of nodes: $n = 2i + 1 = 2(3) + 1 = 7$

Height

- Root * at level 0

- Operators + and - at level 1

- Leaves a, b, c, dat level 2

$$h = 2$$

The tree is balanced because all leaves are located at the same level (level 2)

Instance 2: Binary Decision Tree

A simple procedure to determine why a light bulb is not working

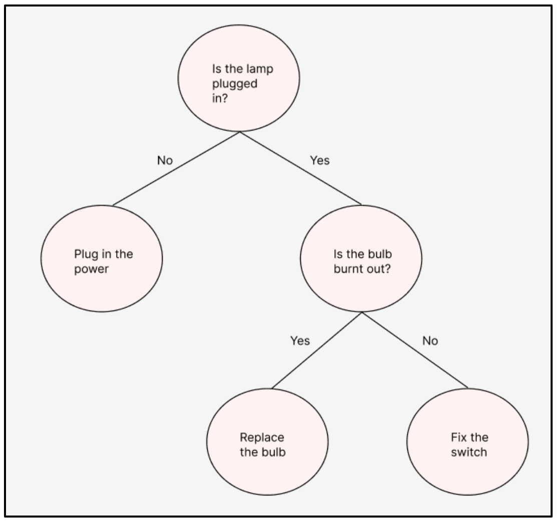


Figure 2: Binary Decision Tree

Root: The question “Is the lamp plugged in?”

No branch (left): Action – “Plug it in.” (Leaf)

Yes branch (right): Proceed to the next question – “Is the bulb burnt out?”

Yes branch (of the right child): Action – “Replace the bulb.” (Leaf)

No branch (of the right child): Action – “Fix the switch.” (Leaf)

1. Qualitative Analysis

Meaning of edges: Edges represent answers “Yes” or “No,” guiding the flow of the decision process.

Meaning of nodes:

- Internal nodes: Represent conditions or tests. A parent node corresponds to a prerequisite condition for its child nodes.

Leaves: Represent the final decisions or specific actions taken to resolve the problem.

Property: The tree helps isolate the problem by breaking it down into smaller possibilities (divide and conquer).

2. Quantitative Analysis

Tree type: This is not necessarily a full binary tree, but it is a binary tree because each question has at most two outcomes (Yes/No).

Number of nodes: Total nodes $n = 5$

- Internal nodes (questions): 2

- “Is the lamp plugged in?”

- “Is the bulb burnt out?”

Leaves (actions): 3

- “Plug it in”

- “Replace the bulb”

- “Fix the switch”

Height: The longest path is

$$\text{Root} \rightarrow \text{Is the bulb burnt out?} \rightarrow \text{Action}$$

This path has 2 edges, so

$$h = 2$$

Balanced: The tree is not balanced.

- Leaf “Plug it in” is at level 1

- Leaves “Replace the bulb” and “Fix the switch” are at level 2

According to the broad definition, a tree is balanced when leaves appear at levels $h$or $h - 1$. Here, leaves at levels 1 and 2 (with $h = 2$) technically satisfy this condition, but visually the tree is skewed to the rightd

2.2 (P4) Use Dijkstra’s algorithm to find a shortest path spanning tree in graph

2.2.1 Definitions and Terminology

To apply Dijkstra's algorithm, we first model the problem using a weighted graph

- Weighted Graph: A graph $G = (V, E)$ where each edge $e \in E$ is assigned a non-negative numerical value, called its weight or cost, denoted $w(e)$ or $w(u, v)$ for the edge connecting u and v (Rosen, 2019)

- Undirected Graph: A graph where edges have no direction. The edge (u, v) is the same as the edge (v, u). In a weighted context, this means the cost to go from u to v is the same as the cost from v to u

- Path: A sequence of vertices $v_0, v_1, \dots, v_k$ such that (v_{i-1}, v_i) is an edge for all $i = 1, \dots, k$

- Path Length: In a weighted graph, this is the sum of the weights of all edges in the path, not the number of edges (Epp, 2019)

- Shortest Path Problem: Given two vertices, a source s and a destination t, the goal is to find a path from s to t such that the sum of the weights of the edges on that path is minimized (Grimaldi, 2017)

Dijkstra's algorithm solves the single-source shortest path problem, meaning it finds the shortest path length from a source vertex s to *all* other vertices in the graph (Rosen, 2019). A critical prerequisite is that all edge weights must be non-negative**.**

2.2.2 Applications in Computing

Dijkstra's algorithm works by maintaining a set of "visited" vertices (S) for which we know the shortest path, and a set of "unvisited" vertices ($V - S$)

Input: A weighted, undirected graph $G = (V, E)$ with non-negative weights $w(u, v) \geq 0$, and a source vertex $s \in V$.

Output: An array $L[v]$ containing the shortest path length from s to every $v \in V$

Procedure

Initialization

- Create a set S (of visited vertices) and initialize $S = \emptyset$

- Create an array L for path lengths

- Set $L[s] = 0$ (the distance from the source to itself is 0)

- For every other vertex $v \neq s$, set $L[v] = \infty$ (representing that we have not yet found a path to v) (Rosen, 2019)

Main Loop

- While $S \neq V$

- Select: Choose the vertex u not in S (i.e., $u \in V - S$) that has the smallest $L[u]$ value

- Visit: Add u to the set S

- Relaxation: For each vertex v adjacent to u and not in S

- If $L[u] + w(u,v) < L[v]$, then update

$$L[v] = L[u] + w(u,v)$$

- This step is called "relaxation" of the edge (u,v). It checks if going through u provides a shorter path to v than any path currently known (Epp, 2019)

Termination

- When the loop finishes ($S = V$), the array L contains the shortest path lengths from s to every other vertex in the graph

2.2.3 Worked Example

Specific Case**:** Given the following weighted, undirected graph

- Vertices (V): {A, B, C, D, Z}

- Edges (E) and Weights (w)

$$\begin{aligned} w(A,B) &= 4 \\ w(A,C) &= 2 \\ w(B,C) &= 1 \\ w(B,D) &= 5 \\ w(C,D) &= 8 \\ w(C,Z) &= 10 \\ w(D,Z) &= 2 \end{aligned}$$

Objective: Apply Dijkstra's algorithm to find the shortest path length from A to Z

2.2.4 Applications in Computing

Network Routing: This is the most classic application. Routing protocols like OSPF (Open Shortest Path First) use Dijkstra's algorithm to determine the "least cost" path (where cost can be latency, bandwidth, or hop count) for sending data packets across the internet from one router to another (Rosen, 2019)

GPS and Mapping Systems: When you use Google Maps or Waze to find the "fastest" route, the system is solving the shortest path problem. Intersections are vertices, and roads are edges, with weights representing travel time or distance (Epp, 2019)

Logistics Planning: Shipping and delivery companies use these algorithms to find the most cost-effective or time-efficient routes for delivering goods from warehouses to customers

Social Networks: While often unweighted (using BFS), weighted versions can model the "strength" of a relationship, and Dijkstra could be used to find the "strongest" path of connection between two people (Grimaldi, 2017)

2.2.5 Activity 2 – Part 4

Stick in mind that a < b represents the largest digits in your ID.

1. State Dijkstra's Algorithm in an undirected graph.

2. Apply Dijkstra's algorithm to determine the shortest path length between vertices A and Z in the provided weighted graph

Solution

1. My student ID is BD00535, and since a < b, we have a = 3 and b = 5

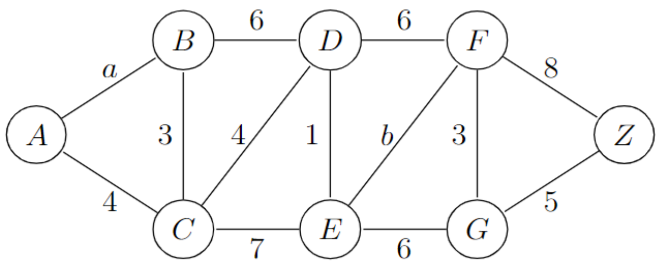


Figure 3: Algebra exercises

First, to determine the shortest path, I initialized vertex A with a value of 0, while all other vertices were assigned a value of infinity

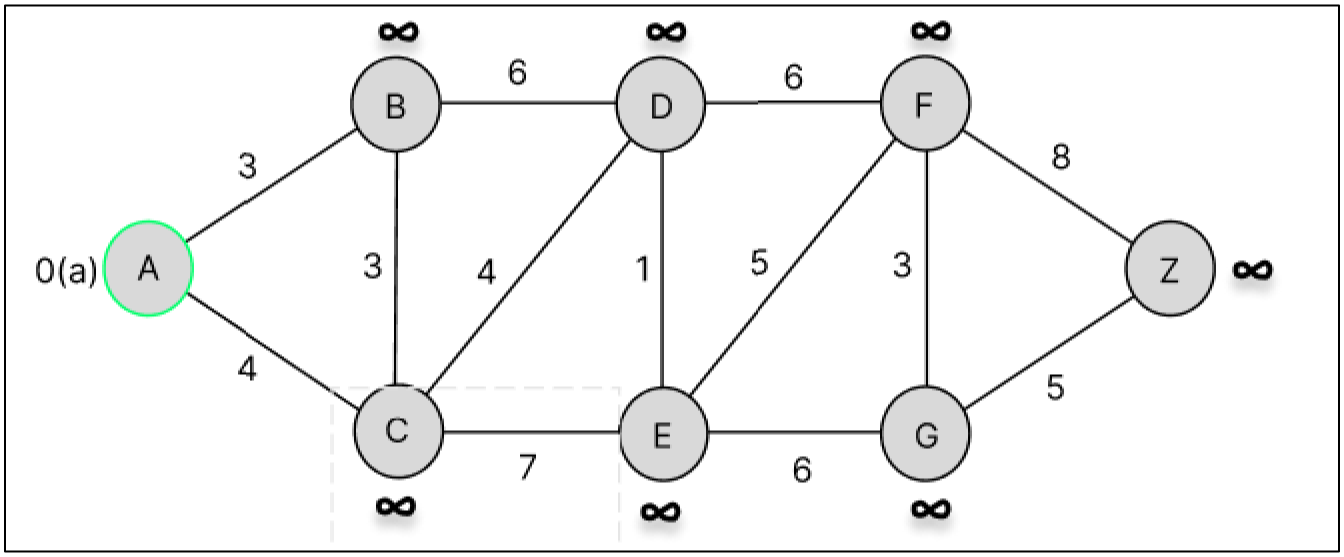


Figure 4: Initialization of Dijkstra's algorithm

Next, from vertex A, there are two possible vertices to move to: B with a cost of 3 and C with a cost of 4. Applying the shortest-path rule, I proceed to vertex B since its cost is 3 (3(a, b)). Similarly, for vertex C, the path has a cost of 4 (4(a, c))

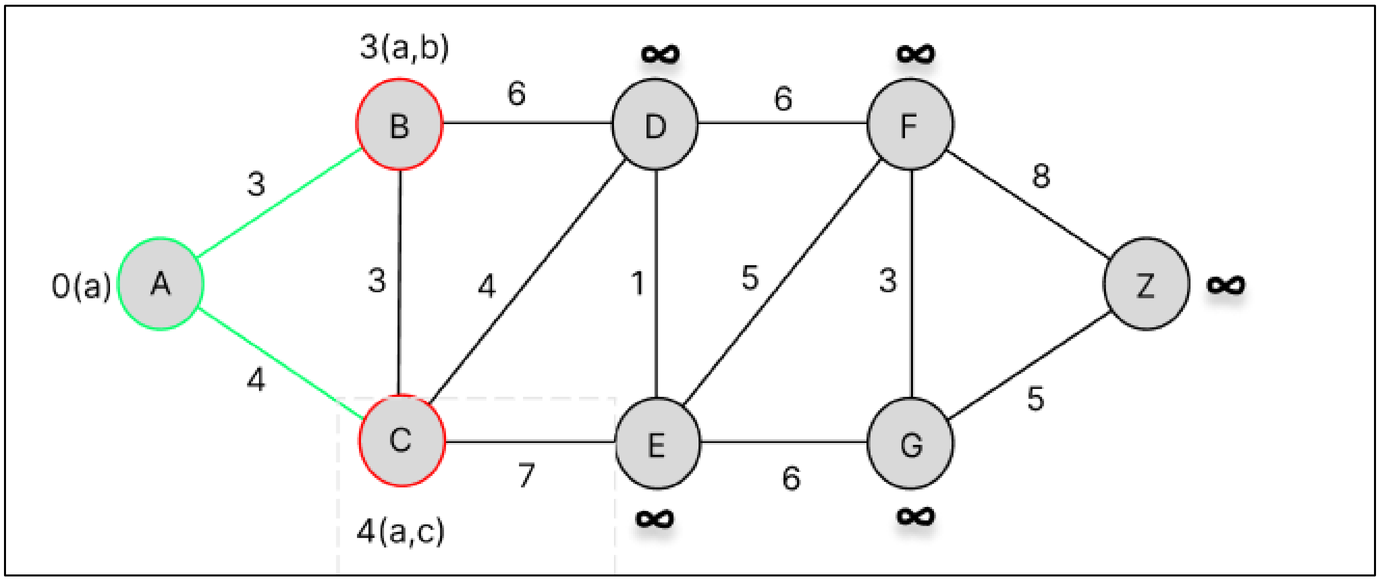


Figure 5: Step to examine the neighbors of A (B and C)

At this point, there are two possible cases

Case 1 (Starting from vertex B)

- If I continue from vertex B, there are two vertices to evaluate

- From B, the cost to reach C is 3, and the cost to reach D is 6

- Therefore, moving from B to C results in a total cost of 6 (a → b → c), while going from B to D results in 9 (a → b → d)

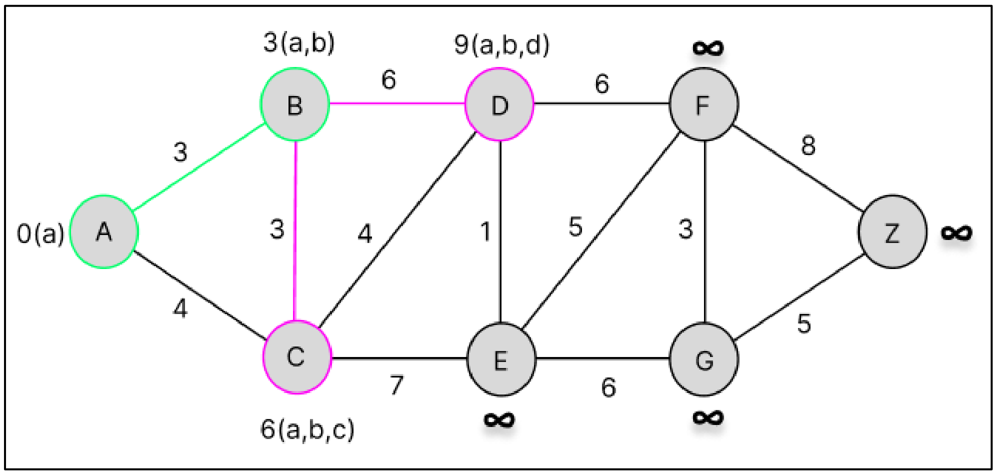


Figure 6: Next step examining neighbors from vertex B (to C and D)

Case 2 (Starting from vertex C)

If I proceed from vertex C, I need to consider three neighboring vertices

- Moving from C back to B results in 4 + 3 = 7 → 7(a, c, b), but this does not improve the existing path to B, and it also leads to a longer path to D compared to the route A → B → D (which is 9). Therefore, this path is discarded

- For the remaining vertices

+ From C to D: the total distance becomes 4 + 4 = 8 → 8(a, c, d)

+ From C to E: the total distance becomes 4 + 7 = 11 → 11(a, c, e)

Based on the current shortest distances, the optimal path at this stage is A → C → D

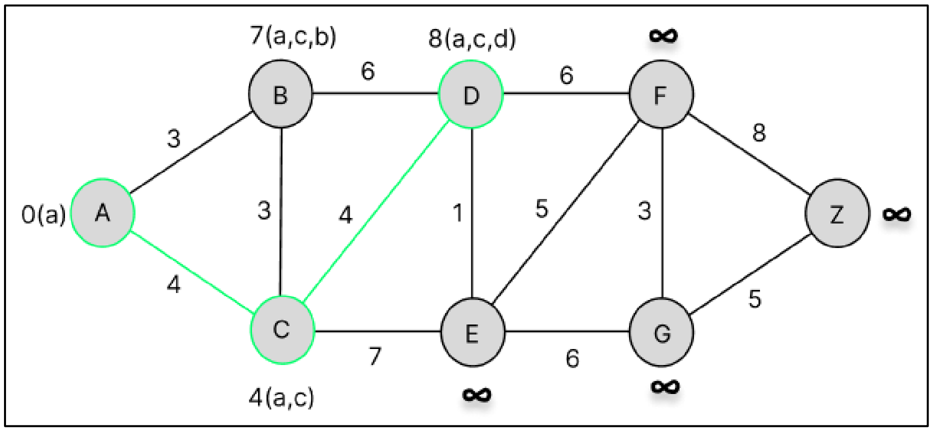


Figure 7: Updating shortest paths from C and selecting vertex D

From vertex D, there are two neighboring vertices to evaluate

Moving from D to E adds a cost of 1, resulting in a total distance of 8 + 1 = 9 (a → c → d → e)

Moving from D to F adds a cost of 6, resulting in a total distance of 8 + 6 = 14 (a → c → d → f)

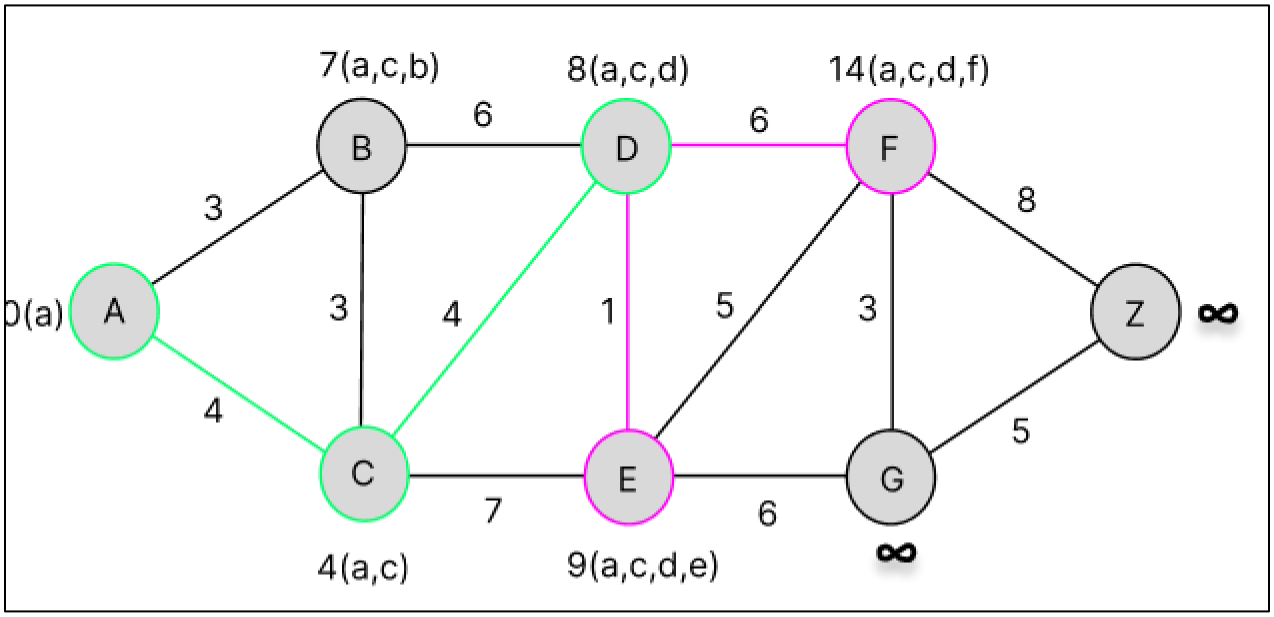


Figure 8: Examining neighbors from vertex D and updating shortest paths to E and F

At this point, comparing all currently known distances, the shortest available next step is the path leading to vertex E with a cost of 9, so the algorithm proceeds to vertex E

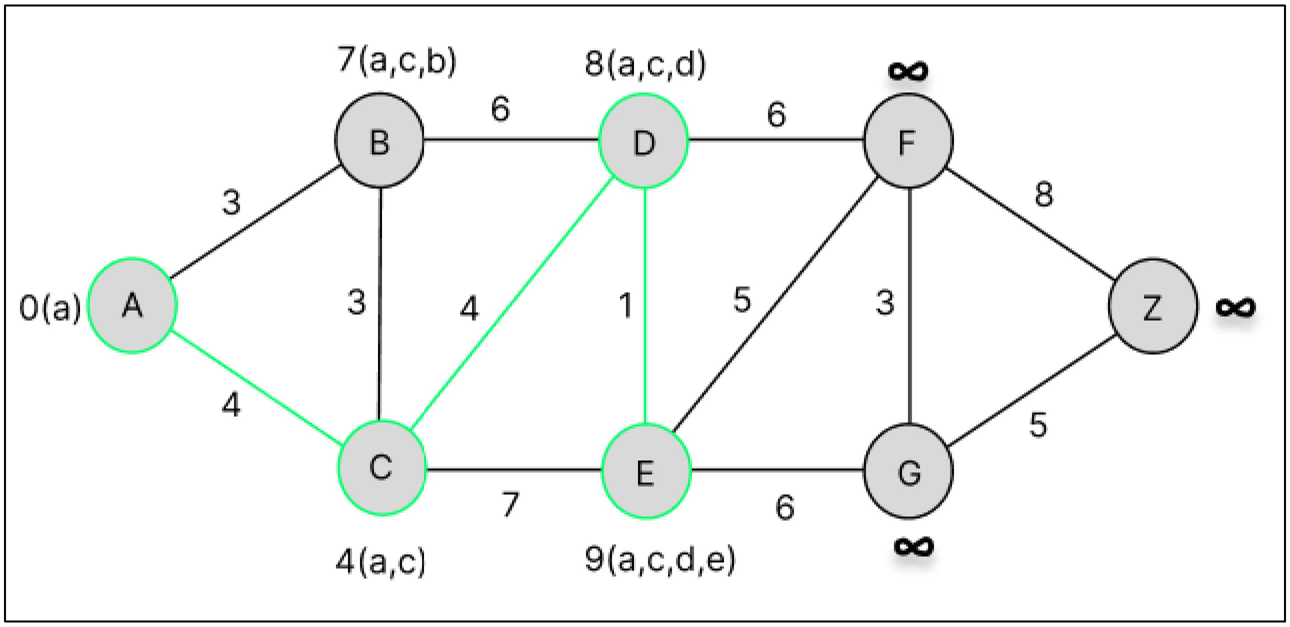


Figure 9: Examining neighbors from vertex D and updating shortest paths to E and F

From vertex E, there are two neighboring vertices:

- Moving from E to F adds a cost of 5 → cost becomes 9 + 5 = 14 (a → c → d → e → f)

- Moving from E to G adds a cost of 6 → cost becomes 9 + 6 = 15 (a → c → d → e → g)

- Comparing all current tentative distances, the next vertex with the smallest value is F = 14, so the algorithm proceeds to vertex F.

From vertex F, there is one relevant neighbor:

- Moving from F to Z adds a cost of 8 → cost becomes 14 + 8 = 22 (a → c → d → e → f → z)

- Since vertex G still has a tentative cost of 15, which is smaller than 22, the algorithm continues with vertex G.

From vertex G, evaluating its neighbor:

- Moving from G to Z adds a cost of 5 → cost becomes 15 + 5 = 20 (a → c → d → e → g → z)

- Since 20 is smaller than the previously discovered path (22 via F), the shortest distance to Z is updated to 20, and the optimal path is now:

A → C → D → E → G → Z

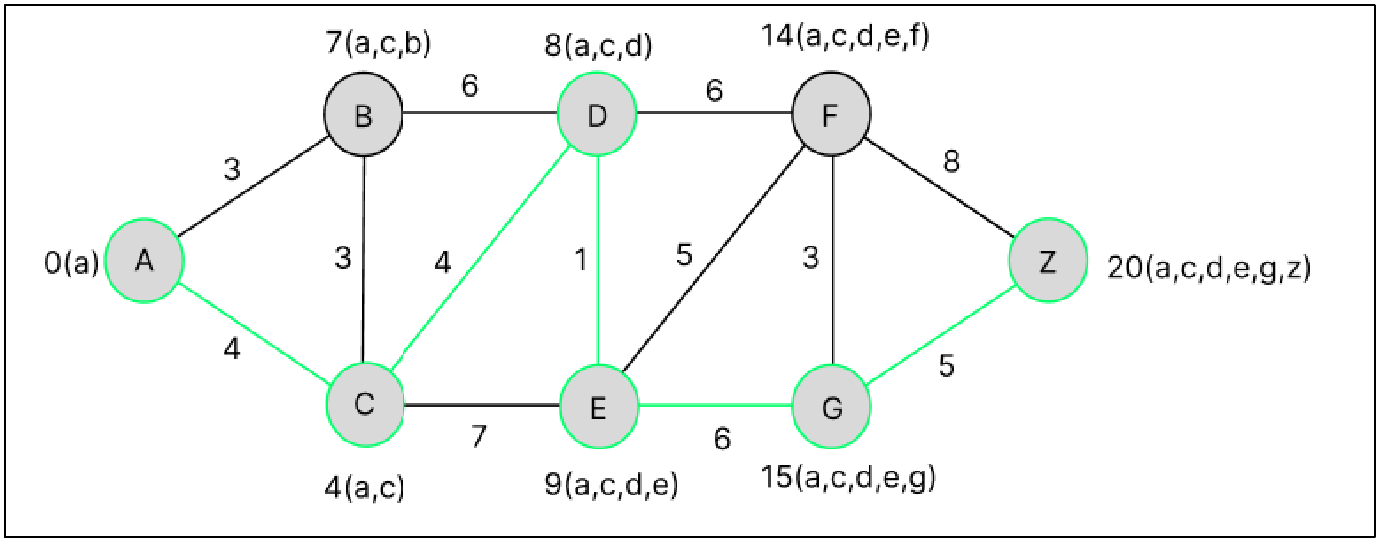


Figure 10: Result of the shortest path from A to Z (A → C → D → E → G → Z)

2.3 (M2) Assess whether a Eulerian and Hamiltonian circuit exists in an undirected graph

2.3.1 Definition and Conditions for Euler Circuits

Definition An Euler circuit in a graph G is a simple circuit that contains every edge of G. This means the circuit must start and end at the same vertex and traverse every single edge exactly once.

If a path traverses every edge exactly once but does not return to the starting vertex, it is called an Euler path.

Conditions for Existence To determine if an undirected graph contains an Euler circuit without having to manually trace it, we use the degree of the vertices:

- A connected multigraph has an Euler circuit if and only if every vertex has an even degree.

- If the graph has exactly two vertices of odd degree, it does not have an Euler circuit, but it has an Euler path.

- If the graph has more than two vertices of odd degree, it has neither

2.3.2 Definition and Conditions for Hamilton Circuits

Definition A Hamilton circuit is a simple circuit in a graph G that passes through every vertex exactly once. Unlike Euler circuits, which focus on edges, Hamilton circuits focus on visiting every node (vertex) and returning to the start.

Conditions for Existence Determining the existence of a Hamilton circuit is generally more difficult than for Euler circuits, as there is no simple "if and only if" criterion like vertex degrees. However, there are sufficient conditions:

- Dirac's Theorem: If G is a simple graph with n vertices ($n \geq 3$) and the degree of every vertex is at least $\frac{n}{2}$, then G has a Hamilton circuit.

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

If these conditions are not met, a Hamilton circuit might still exist, and we must attempt to construct one or prove it is impossible logically.

Problem Assess whether the graph used in the previous Dijkstra algorithm section (Section 2.2.5) contains

Analysis of the Graph Based on the graph provided in the Dijkstra solution, the set of vertices is $V = \{A, B, C, D, E, F, G, Z\}$

Assessment for Euler Circuit First, I determine the degree (number of connecting edges) for each vertex in the graph

- deg(A) = 2 (Connected to B, C)

- deg(B) = 3 (Connected to A, C, D) → Odd

- deg(C) = 4 (Connected to A, B, D, E)

- deg(D) = 4 (Connected to B, C, E, F)

- deg(E) = 4 (Connected to C, D, F, G)

- deg(F) = 4 (Connected to D, E, G, Z)

- deg(G) = 3 (Connected to E, F, Z) → Odd

- deg(Z) = 2 (Connected to F, G)

According to the condition for Euler circuits, every vertex must have an even degree. In this graph, vertices B and G have odd degrees (3). Therefore, an Euler Circuit does NOT exist in this graph.

Since there are exactly two vertices of odd degree, an Euler Path exists, but not a Circuit

Assessment for Hamilton Circuit I check if there is a path that visits vertices A, B, C, D, E, F, G, Z exactly once and returns to the start. Let's attempt to construct a path starting from A

- Start at A

- Move to B

- Move to D

- Move to F

- Move to Z

- Move to G

- Move to E

- Move to C

- From C, return to A

Path Verification: The sequence is: $A \rightarrow B \rightarrow D \rightarrow F \rightarrow Z \rightarrow G \rightarrow E \rightarrow C \rightarrow A$

Are all edges valid?

- (A,B): Yes

- (B,D): Yes

- (D,F): Yes

- (F,Z): Yes

- (Z,G): Yes

- (G,E): Yes

- (E,C): Yes

- (C,A): Yes

Did we visit every vertex? Yes (A, B, C, D, E, F, G, Z are all visited)

Did we visit any vertex twice? No (except the start/end)

Since we found a valid cycle that visits every vertex exactly once and returns to the start, a Hamilton Circuit DOES exist in this graph. The circuit is: A - B - D - F - Z - G - E - C – A

2.3.3. Activity 2 -Part 3

Does the following graph have a Hamilton path? If so, find such a path. If it does not, give an argument to show why no such path exists

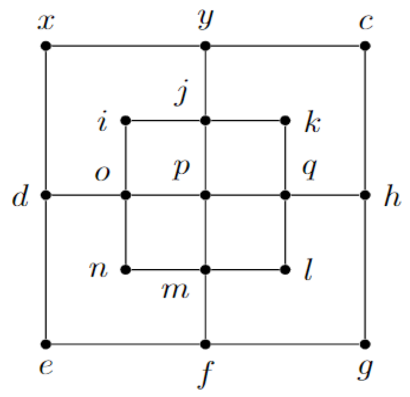


Figure 11: Grid graph for the Hamiltonian path problem

Solution

Assessment for Euler Path

Theoretical Basis: According to the conditions for the existence of Euler paths in a connected graph.

- A graph has an Euler circuit if and only if every vertex has an even degree.

- A graph has an Euler path if and only if it has exactly two vertices of odd degree.

- If the number of vertices with odd degrees is greater than 2, no Euler path exists.

Graph Analysis: We calculate the degree (number of incident edges) for the vertices in the given graph.

- Degree 2 (Even): Vertices x, c, g, e (outer corners) and all inner vertices $(i, j, k, \ldots)$.

- Degree 3 (Odd): Vertices y, h, f, d (the midpoints of the outer edges).

$$y \text{ connects to } x, c, j$$
$$h \text{ connects to } c, g, q$$
$$f \text{ connects to } g, e, m$$
$$d \text{ connects to } e, x, o$$

There are exactly 4 vertices with odd degrees (y, h, f, d). Since $4 > 2$, no Euler path exists in this graph.

Assessment for Hamilton Path

Theoretical Basis: A Hamilton path is a simple path that passes through every vertex of the graph exactly once. To prove non-existence, we can use the Bipartite Coloring Argument.

Proof

Step 1: Bipartite Coloring (Vertex Segmentation) The graph is bipartite. We can color the vertices with two colors, White (W) and Black (B), such that no two vertices of the same color are adjacent.

- Set W (White): Outer midpoints (y, h, f, d), Inner corners (i, k, l, n), Center (p) => Total count: $|W| = 9$.

- Set B (Black): Outer corners (x, c, g, e), Inner midpoints (j, q, m, o) => Total count: $|B| = 8$.

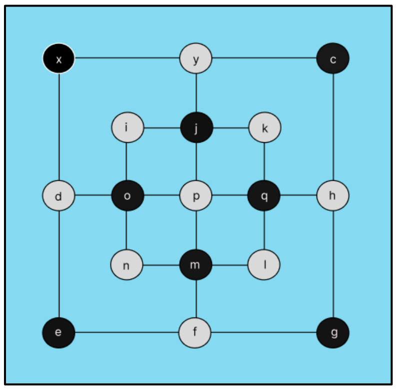


Figure 12: Illustration 1: Bipartite Coloring (White/Black) of the Grid Graph

Illustration 1: Vertex Coloring

Since $|W| = 9$ and $|B| = 8$, any Hamilton path must start and end at a White vertex.

Step 2: Endpoint Logic Any path in a bipartite graph alternates colors ($W \rightarrow B \rightarrow W \rightarrow \cdots$). Since there are 9 White and 8 Black vertices, the path must follow the pattern:

$$W_1 \rightarrow B_1 \rightarrow W_2 \rightarrow \cdots \rightarrow B_8 \rightarrow W_9$$

The path must start and end at White vertices. Consequently, no Black vertex can be an endpoint.

Step 3: The Contradiction at Outer Corners Consider the outer corner vertices: x, c, g, e

- They are all Black vertices. Therefore, they cannot be endpoints (from Step 2).

- They all have a degree of 2.

- If a vertex is not an endpoint, the path must pass *through* it. To pass through a vertex with a degree of 2, the path must enter via one edge and exit via the other.

- Implication: Both edges incident to every outer corner (x, c, g, e) must be included in the path.

Step 4: Cycle Isolation (The Trap) Forcing the usage of all edges connected to the outer corners (x, c, g, e) creates a closed loop on the perimeter.

- Using edges at x connects d and y

- Using edges at c connects y and h

- Using edges at g connects h and f

- Using edges at e connects f and d

Illustration 2: The Forced Outer Cycle

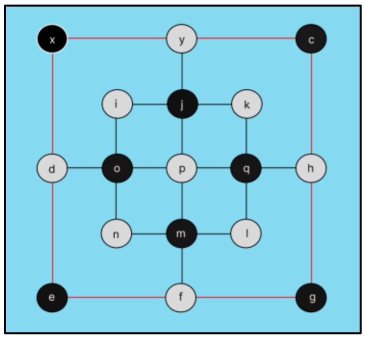


Figure 13: Forced Outer Cycle in the proof of non-existence of a Hamilton path

This logic forces the construction of the cycle $x - y - c - h - g - f - e - d - x$. Once this outer cycle is formed, the path has visited the vertices y, h, f, d. It cannot "turn inward" to visit the inner vertices $(j, i, k, \ldots)$ without traversing y, h, f or d a second time. (which violates the definition of a Hamilton path)

Therefore, no Hamilton path exists.

CHAPTER 3 – LO3: INVESTIGATE SOLUTIONS TO PROBLEM SITUATIONS USING THE APPLICATION OF BOOLEAN ALGEBRA

3.1 (P5) Diagram a binary problem in the application of Boolean algebra

3.1.1 What is Boolean Algebra

Boolean algebra is the mathematics of digital logic. It uses symbols to represent logical decisions rather than calculation

- In Regular Algebra: $1 + 1 = 2$

- In Boolean Algebra: $1 + 1 = 1$ (Because "True OR True" equals "True")

Boolean algebra is used to model the circuitry of electronic devices. Each input and each output of such a device can be thought of as a member of the set {0,1}. A computer, or other electronic device, is made up of a number of circuits. Each circuit can be designed using the rules of Boolean algebra. The basic elements of circuits are called gates. Each type of gate implements a Boolean operation.

It relies on three main operators which you will use to write your equations

Table 1: Basic Logic Gates (AND, OR, NOT) and Mathematical Equivalents

Logic Name	Symbol	Math Analogy	How it works
AND	. (dot)	Multiplication	Output is 1 only if ALL inputs are 1.
OR	+ (plus)	Addition	Output is 1 if ANY input is 1.
NOT	$\bar{A}(bar)$	Inversion	Output is the opposite (0 becomes 1).

3.1.2 How do you use it? (The 3-Step Process)

Step 1: Define your Variables

Assign a letter to every condition in your scenario

- *Example:* "If the key is turned (K) and the brake is pressed (B)…"

Step 2: Translate English to Math (The Equation)

Replace the words "AND", "OR", and "NOT" with symbols.

- English: The Car Starts (S) if Key is turned (K) AND Brake is pressed (B).

- Boolean Equation: S = K . B

Step 3: Calculate Outcomes (The Truth Table)

Because there are only 0 and 1, you can list every possible combination to prove how the system works.

Example Truth Table for S = K . B

Table 2: Truth Table for the Car Start System (AND Logic: $S = K \cdot B$)

K (Key)	B (Brake)	S (Start)	Logic
0	0	0	Key Off AND Brake Off = No Start
0	1	0	Key Off AND Brake On = No Start
1	0	0	Key On AND Brake Off = No Start
1	1	1	Key On AND Brake On = Start

3.1.3 Domain 1: Smart Home Security System

This example demonstrates the application of Boolean algebra in consumer electronics and safety systems.

The Problem Scenario

A home security system should trigger an alarm (Z) only if the System is Armed (A) AND either the Motion Sensor (M) detects movement OR the Window Sensor (W) is broken.

Binary Number Representation

In Boolean algebra, we map physical states to binary values (0 or 1):

- Input A (System Armed): 0 = Disarmed, 1 = Armed.

- Input M (Motion): 0 = No Motion, 1 = Motion Detected.
- Input W (Window): 0 = Closed/Safe, 1 = Open/Broken.
- Output Z (Alarm): 0 = Silent, 1 = Ringing.

Boolean Logic & Equation

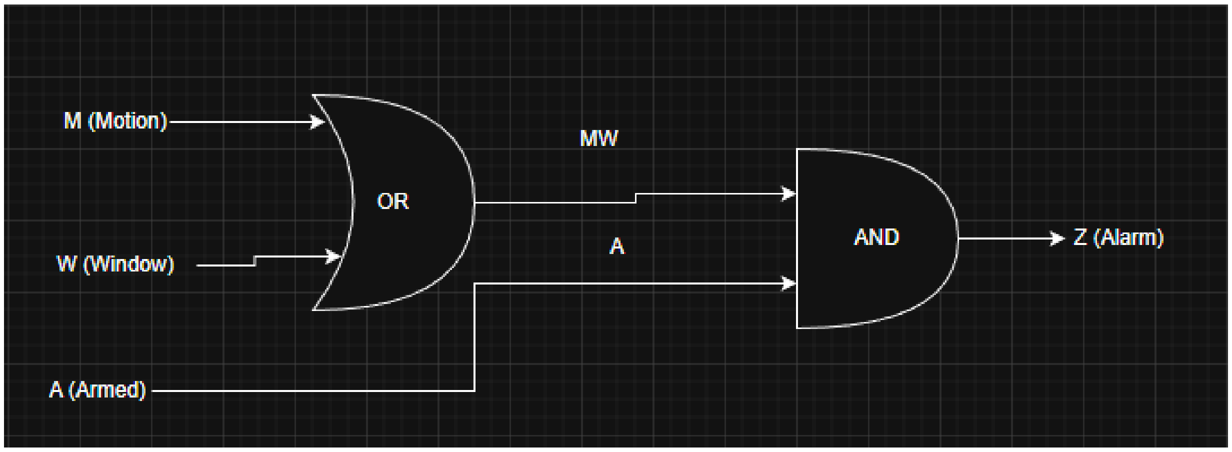
The logic requires an OR operation for the intrusion sensors (Motion or Window) and an AND operation to check if the master system is active.

Boolean Equation:

$$Z = A \land (M \lor\ V)$$

Logic Gate Diagram

To diagram this, you would connect inputs M and W into an OR gate. The output of that OR gate is then connected to one input of an AND gate. Input A connects to the second input of the AND gate. The final output is Z.



Practical Application

This logic is used in microcontroller programming (like Arduino or PLC) for residential security. It ensures that the alarm does not go off falsely when the homeowner is home (System Disarmed = 0), effectively managing false positives.

3.1.4 Domain 2: Agricultural Irrigation Control

This example demonstrates the application of Boolean algebra in environmental control and automation.

The Problem Scenario

An automatic irrigation valve (V) should open to water the crops ONLY IF the Soil is Dry (S) AND it is NOT Raining (R).

Binary Number Representation

- Input S (Soil Moisture Sensor): 0 = Wet (Do not water), 1 = Dry (Needs water).
- Input R (Rain Sensor): 0 = No Rain, 1 = Raining.
- Output V (Valve): 0 = Closed, 1 = Open (Watering).

Boolean Logic and Equation

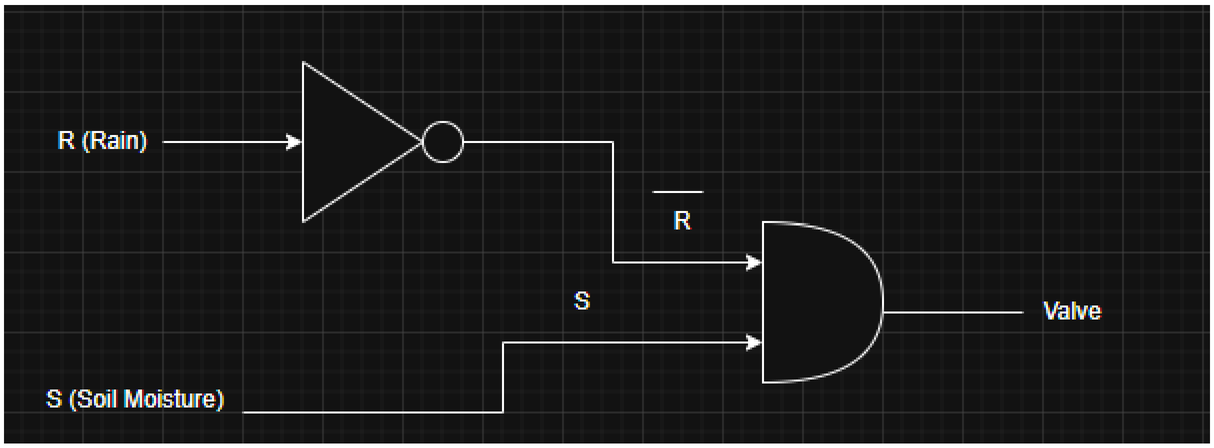
The system requires the soil to be dry (S=1) AND the rain sensor to be inactive (R=0). This requires a NOT operation on the rain input to invert the logic (we want the absence of rain).

Boolean Equation:

$$V = S \land \neg R$$

Logic Gate Diagram

To diagram this, Input R is passed through a NOT gate (inverter). The output of the NOT gate and Input S are then passed into an AND gate. The final output is V.



Practical Application

This is widely used in "Smart Farming" (Precision Agriculture). It conserves water resources by preventing the irrigation system from running during a rainstorm, optimizing resource usage based on binary sensor data

3.2 (P6) Produce a truth table and its corresponding Boolean equation from an applicable scenario

3.2.1 What is a Boolean Equation

A Boolean Equation is just a sentence written in math symbols instead of words. It describes the rule for when something should happen.

- In English: "The Alarm (Z) rings if the system is Armed (A) AND there is Motion (M)."

- In Boolean Algebra: $z = A \,.\, M$

It converts human logic into a format computers can understand using three main symbols:

- (.) (AND) means both must happen.

- $(+\,)$(OR) means either one can happen.

- $(\bar{A})$ (NOT) means the opposite must happen.

3.2.2 Scenario A: The Secure Facility Door

Scenario Description: "In a secure facility, if the access card is swiped OR the correct PIN is entered AND the security system is NOT in maintenance mode, then the door should unlock."

1. Problem Decomposition (Variables)

First, we assign binary variables to the physical conditions.

- Input C (Card): 0 = Not Swiped, 1 = Swiped

- Input P (PIN): 0 = Incorrect/No PIN, 1 = Correct PIN entered

- Input M (Maintenance): 0 = Normal Mode, 1 = Maintenance Mode

- Output D (Door): 0 = Locked, 1 = Unlocked

2.Logic Analysis

"Access card is swiped OR the correct PIN is entered": This represents a logical OR sum: (C + P).

"AND the security system is NOT in maintenance mode": This requires an AND operation with the inverse (NOT) of maintenance: $.\,\bar{M}$

3. Boolean Equation

$$D = (C + P) . \bar{M}$$

4. Truth Table

The system requires (C+P) to be True, AND M to be False (0) for the door to unlock.

Table 3: Truth Table for the Secure Facility Door System (Equation: $D = (C + P) \cdot \bar{M}$)

Row	Maintenance (M)	Card (C)	PIN (P)	OR Check (C+P)	NOT M ($\bar{M}$)	Door (D)
0	0	0	0	0	1	0
1	0	0	1	1	1	1
2	0	1	0	1	1	1
3	0	1	1	1	1	1
4	1	0	0	0	0	0
5	1	0	1	1	0	0
6	1	1	0	1	0	0
7	1	1	1	1	0	0

3.2.3 Scenario B: Computer Login (Exclusive OR)

Scenario Description: "For a computer to successfully log in, either a valid username and password combination must be entered OR a security token must be provided, but not both."

Problem Decomposition (Variables)

- Input C (Credentials): 0 = Invalid, 1 = Valid Username/Password

- Input T (Token): 0 = Not Provided, 1 = Provided Valid Token

- Output L (Login): 0 = Failed, 1 = Success

Logic Analysis

The phrase "either... or... but not both" describes the Exclusive OR (XOR) operation.

If both inputs are present (1, 1), the login fails (perhaps to prevent duplicate session errors or security conflicts).

If neither is present (0, 0), the login fails.

Boolean Equation

Using Basic Logic Gates (AND, OR, NOT):

$$L = (C.\bar{T}) + (\bar{C}.T)$$

Truth Table

Table 4: Truth Table for the Computer Login Scenario

Row	Credentials (C)	Token (T)	Login (L)	Explanation
0	0	0	0	No credentials or token provided.
1	0	1	1	Token provided without credentials. (Success)
2	1	0	1	Credentials provided without token. (Success)
3	1	1	0	Both provided ("but not both" rule violations).

3.2.4 Truth Table for Provided Expression

$$xyz + x\overline{yz} + \overline{x}y\overline{z} + \overline{xy}z.$$

Logic Analysis: The function F is True (1) if any of the four terms are True.

- xyz: All inputs are 1.

- $x\overline{yz}$: x is 1, y and z are 0

- $\overline{x}y\overline{z}$: y is 1, x and z are 0

- $\overline{xy}z$:z is 1, x and y are 0

Truth Table:

Table 5: Truth Table for the Boolean Function $F = xyz + x\overline{yz} + \overline{x}y\overline{z} + \overline{xy}z$

Row	x	y	z	xyz	$x\overline{yz}$	$\overline{x}y\overline{z}$	$\overline{xy}z$	F
0	0	0	0	0	0	0	0	0

1	0	0	1	0	0	0	1	1
2	0	1	0	0	0	1	0	1
3	0	1	1	0	0	0	0	0
4	1	0	0	0	1	0	0	1
5	1	0	1	0	0	0	0	0
6	1	1	0	0	0	0	0	0
7	1	1	1	1	0	0	0	1

3.3 (M3) Simplify a Boolean equation using algebraic methods

3.3.1 Definition: Boolean Simplification (Algebraic Methods)

Boolean Simplification is the mathematical process of reducing a complex Boolean expression to its shortest, simplest equivalent form without changing its logical output.

Instead of relying on truth tables (which list every possibility), this method uses specific Algebraic Laws to cancel out redundant terms.

What Can It Do? (Purpose & Utility)

Reduces Complexity

- It transforms long, difficult-to-read equations into concise logic.

- *Example from your text:* It takes a complex string like x(x+y) + y(y+z) +z(z+x) and reduces it down to simply $x + y + z$.

Optimizes Electronic Circuitry

- Boolean algebra is used to "model the circuitry of electronic devices" where the basic elements are "gates".

- Fewer Components: By simplifying the equation, you prove that you need fewer logic gates to build the machine. This means a complex circuit could effectively be replaced by a single wire (connection), saving massive amounts of hardware.

- Cost and Efficiency: Fewer gates mean the final device is cheaper to build, consumes less power, and processes data faster.

Improves Human Readability

- It converts raw logic into a format that is easier for engineers and computer scientists to understand and maintain. It creates a "Simplified Answer" that clarifies exactly what conditions are necessary for a result.

3.3.1 Activity 3 - Part 1

Equation 1

Original Expression:

$$x(x+y)+y(y+z)+z(z+x).$$

Step-by-Step Simplification:

Expand brackets (Distributive Law):

$$xx + xy + yy + yz + zz + zx$$

Apply Idempotent Law (AA = A):

$$x + xy + y + yz + z + zx$$

Group terms (Factor out x, y, z):

$$x(1 + y) + y(1 + z) + z(1 + x)$$

Apply Annulment Law (1 + A = 1):

$$x(1) + y(1) + z(1)$$

Final Simplified Answer:

$$x + y + z$$

Equation 2

Original Expression:

$$(x+\overline{y})(y+z)+(x+y)(z+\overline{x}).$$

Expand both sets of brackets (FOIL method):

$$(xy+xz+\bar{y}y+ \bar{y}z)+(xz+x\bar{x}+yz+y\bar{x})$$

Apply Complement Law ($A\bar{A}=0$):

$$xy + xz + \bar{y}z + xz + yz + \bar{x}y$$

Group common terms:

Group y: $xy + \bar{x}y = y\ (x + \bar{x})$ = y(1) = y

Group z: $\bar{y}z + yz = z(\bar{y} + y) = z(1) = z$

Group x: $xz + xz = xz$ (Idempotent Law) Result: $y + z + xz$

Factor out z:

$$y + z(1 + x)$$

Apply Annulment Law (1 + x = 1):

$$y + z(1)$$

$$\mathrm{y} + \mathrm{z}$$

Final Simplified Answer:

$$\mathrm{y} + \mathrm{z}$$

Equation Three

Original Expression:

$$(x + y) + (xz + x\bar{z}) + zx + x$$

Step-by-Step Simplification:

Simplify the inner bracket $(xz + x\bar{z})$Factor out x: $x(z + \bar{z})$ Apply Complement Law: $x(1) = x$ Substitute back into equation:

$$(x + y)(x) + zx + x$$

Expand brackets:

$$xx + xy + zx + x$$

Apply Idempotent Law $(xx = x)$

$$x + xy + zx + x$$

$$x + xy + zx$$

Factor out x:

$$x(1 + y + z)$$

Apply Annulment Law ($1 + anything = 1$)

$$x(1)$$

Final Simplified Answer:

$$x$$

Equation 4

Original Expression:

$$\bar{x}(x + y) + (x + y)(x + \bar{y})$$

Step-by-Step Simplification:

Expand first part:

$$\bar{x}x + \bar{x}y = 0 + \bar{x}y = \bar{x}y$$

Expand second part:

$$xx + x\bar{y} + yx + y\bar{y}$$

$$x + x\bar{y} + xy + 0$$

$$x(1 + \bar{y} + y)$$

$$x(1) = x$$

Combine parts:

$$\bar{x}y + x$$

Apply Redundancy Law ($A + \bar{A}B = A + B$)This law states that if you have a variable (x) and its inverse multiplied by another variable $\bar{x}y$ you can remove the inverse.

$$x + y$$

Final Simplified Answer:

$$x + y$$

CHAPTER 4 – LO4: EXPLORE APPLICABLE CONCEPTS WITHIN ABSTRACT ALGEBRA

4.1 (P7) Describe the distinguishing characteristics of different binary operations that are performed on the same set

4.1.1 Definition of Binary Operations

A binary operation on a set S is defined in Lecture 14 as a rule that assigns to each ordered pair (x, y)in Sa unique element in S. Formally

$$\alpha: S \times S \to S$$

It is commonly written in the form

$$x \circ y = \alpha(x, y)$$

Closure Property

For a rule to be considered a binary operation, the result must remain within the set

$$\forall x, y \in S, x \circ y \in S$$

Example: Addition on the set of integers: $+: \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$

Testing some values:

$3 + 4 = 7$→ still an integer

$(-2) + 5 = 3$→ still an integer

Addition is a binary operation because the result always lies in $\mathbb{Z}$

Example: Operation $a * b = a^2 + b$ on integers: $*: \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$

Testing:

$$2 * 3 = 2^2 + 3 = 7$$

$$(-1) * 5 = (-1)^2 + 5 = 6$$

This operation is also a binary operation because the results remain in $\mathbb{Z}$

4.1.2 Distinguishing Characteristics of Binary Operations on the Same Set

Even if multiple binary operations are defined on the same underlying set, the algebraic behavior of these operations can vary significantly because each operation interacts with the elements of the set in a different way. This difference arises from the specific algebraic properties such as closure, associativity, commutativity, the existence of an identity, and the existence of inverses that each operation satisfies or fails to satisfy.

For example, in Lecture 14, both addition $+$and the operation $a * b = a^2 + b$ are defined on the same set of integers $\mathbb{Z}$, and both are valid binary operations because they are closed on $\mathbb{Z}$. However, addition is associative, commutative, has an identity element (0), and provides inverses for all integers, making it the foundation of a group. In contrast, the operation $a * b = a^2 + b$ fails to be associative, fails to be commutative, has no identity element, and therefore cannot provide inverses. As a result, it does not form a semigroup, monoid, or group even though it is defined on the same set $\mathbb{Z}$.

Lecture 15 elaborates on why these differences matter: depending on which combination of properties an operation satisfies, we can classify the resulting structure as a semigroup, monoid, or group. Therefore, even when operations share the same domain, their distinct algebraic properties lead to fundamentally different structures and behaviors.

Closure Property

The Closure Property is one of the fundamental characteristics of binary operations in abstract algebra. A binary operation $\circ$ on a set A is said to be closed on A if, for every pair of elements $a, b \in A$, the result of the operation $a \circ b$ also belongs to A:

$$\forall a, b \in A, a \circ b \in A$$

This property ensures that applying the operation does not produce elements outside the original set, maintaining structural stability and allowing repeated applications of the operation without expanding the domain.

Example: Addition on $\mathbb{Z}$

For any integers a and b, their sum $a + b$ is always an integer. Thus:

$$a, b \in \mathbb{Z} \Rightarrow a + b \in \mathbb{Z}$$

Example: The operation $a * b = a^2 + b$ on $\mathbb{Z}$

Since a^2 is an integer and the sum $a^2 + b$ remains an integer for all $a, b \in \mathbb{Z}$, this operation also satisfies closure:

$$a, b \in \mathbb{Z} \Rightarrow a^2 + b \in \mathbb{Z}$$

Although both operations are closed, it is important to emphasize that closure alone does not provide sufficient insight into the deeper algebraic behavior of a binary operation. Additional properties—such as associativity, commutativity, and the existence of identity and inverse elements—are essential in determining whether a set and operation form more advanced algebraic structures such as semigroups, monoids, or groups.

Therefore, even though addition and the operation $a * b = a^2 + b$ are both closed on $\mathbb{Z}$, their algebraic characteristics differ significantly. Addition possesses the necessary properties to form a group, while the operation $a * b = a^2 + b$ fails to meet those requirements.

The Closure Property offers only a basic level of information about the stability of a binary operation, but it is not enough to determine the complexity or structural nature of the algebraic system built upon that operation.

Associative Property

The Associative Property is a key characteristic used to evaluate the behavior of binary operations in abstract algebra. A binary operation $\circ$ on a set A is said to be associative if, for all elements $a, b, c \in A$, the grouping of the operands does not affect the final result:

$$(a \circ b) \circ c = a \circ (b \circ c)$$

This property ensures that the order of applying the operation is structurally consistent, allowing expressions to be written without ambiguity regarding the placement of parentheses.

Example of an Associative Operation: Addition

Addition on the integers is a standard illustration of associativity:

$$(2 + 3) + 4 = 5 + 4 = 9$$

$$2 + (3 + 4) = 2 + 7 = 9$$

Since both expressions yield the same result, it follows that:

$$(2 + 3) + 4 = 2 + (3 + 4)$$

Addition is an associative operation

Example of a Non-Associative Operation: $a * b = a^2 + b$

Consider the operation defined on the integers by:

$$a * b = a^2 + b$$

To test associativity, compare the following computations:

Left grouped expression:

$$(1 * 2) * 3$$

First,

$$1 * 2 = 1^2 + 2 = 3$$

Then,

$$3 * 3 = 3^2 + 3 = 12$$

Right grouped expression

$$1 * (2 * 3)$$

First,

$$2 * 3 = 2^2 + 3 = 7$$

Then,

$$1 * 7 = 1^2 + 7 = 8$$

Since: $12 \neq 8$, the two groupings produce different results

The operation $a * b = a^2 + b$ is not associative

Commutative Property

The Commutative Property describes whether the order of applying a binary operation affects the resulting value. A binary operation $\circ$on a set $A$is said to be commutative if, for all $a, b \in A$:

$$a \circ b = b \circ a$$

This property indicates that swapping the operands does not alter the outcome.

Example of a Commutative Operation: Addition

Addition is a classic example of commutativity. For instance:

$$4 + 5 = 9 = 5 + 4$$

Because the order of the operands does not affect the result, addition is commutative.

Example of a Non-Commutative Operation: $a * b = a^2 + b$

Consider the operation defined as:

$$a * b = a^2 + b$$

Let $a = 2$ and $b = 3$:

$$2 * 3 = 2^2 + 3 = 7$$

$$3 * 2 = 3^2 + 2 = 11$$

Since: $7 \neq 11$, the operation is not commutative.

Identity Element

An element $e \in S$is called an identity element if it satisfies:

$$a \circ e = a \text{ and } e \circ a = a$$

for every $a \in S$. The identity element leaves other elements unchanged under the operation.

Example: Addition on $\mathbb{Z}$ and $\mathbb{N}$

For addition:

$$a + 0 = a, 0 + a = a$$

Thus, 0 is the identity element.

Example Without Identity: $a * b = a^2 + b$

Assume an identity element $e$exists for the operation:

$$a * e = a^2 + e = a$$

Solving for e:

$$e = a - a^2$$

However, $e$now depends on a, meaning it is not a universal identity element, which violates the definition

This operation does not have an identity.

Inverse Property

An element $b$is called an inverse of $a$under a binary operation $\circ$ if: $a \circ b = e$, where e is the identity element of the set.

Example: Addition on $\mathbb{Z}$

$$5 + (-5) = 0$$

$$-3 + 3 = 0$$

Every integer has an inverse under addition; therefore, $(\mathbb{Z}, +)$forms a group**.**

Cases without inverses

- Natural numbers $(\mathbb{N}, +)$: No inverse exists because natural numbers do not include negatives.

- Operation $a * b = a^2 + b$**:** Inverses cannot exist because the operation lacks an identity element, which is required before inverses can be defined.

Idempotence

A binary operation is called idempotent if:

$$a \circ a = a$$

Neither of the considered operations is idempotent:

Addition

$$a + a = 2a \neq a$$

Operation: $a * b = a^2 + b$

$$a * a = a^2 + a \neq a$$

Neither operation is idempotent.

Distributive Property

Unlike the previous properties which apply to a single binary operation, the Distributive Property describes how two binary operations interact with each other.

If we have two binary operations, denoted by $\circ$ and $\circ$, on a set A, we say that $\circ$ distributes over $\circ$ if for all $a, b, c \in A$

$$a \circ (b \circ c) = (a \circ b) \circ (a \circ c)$$

To assess the given examples, we test if the defined operation distributes over standard addition ($+$), or if standard multiplication distributes over the defined operation.

Example: Standard operations on $\mathbb{Z}$ In the set of integers, standard multiplication ($\times$) distributes over standard addition ($+$)

$$a \times (b + c) = (a \times b) + (a \times c)$$

For instance:

$$2 \times (3 + 4) = 2 \times 7 = 14$$

$$(2 \times 3) + (2 \times 4) = 6 + 8 = 14$$

Since $14 = 14$, the property holds. This is a fundamental property that allows $\mathbb{Z}$to form a Ring structure

Example 2: The operation $a * b = a^2 + b$

We test whether this operation $*$distributes over standard addition ($+$). That is, we check if:

$$a * (b + c) = (a * b) + (a * c)$$

Left Hand Side (LHS)

$$a * (b + c) = a^2 + (b + c) = a^2 + b + c$$

Right Hand Side (RHS)

$$(a * b) + (a * c) = (a^2 + b) + (a^2 + c) = 2a^2 + b + c$$

Since $a^2 + b + c \neq 2a^2 + b + c$ (unless $a = 0$), the operation $*$ is not distributive over addition.

4.1.3 Comparative Table

*Table 6: Comparison of algebraic properties for addition + and Operation * on the set of Integers Z*

Property	$+$ *on* $\mathbb{Z}$	$*$ *on* $\mathbb{Z}$
Closure	Yes	Yes
Associativity	Yes	No
Commutativity	Yes	No

Identity	0	None
Inverse	Yes	None
Structure formed	Group	None
Idempotence	No	No
Distributivity	N/A (Standard + is the base)	No (does not distribute over +)

Although two binary operations may both be defined on the same set, they can behave very differently.

- One operation may form a group

- Another may form only a monoid

- Another may not form any algebraic structure at all

Even though all of them operate on the same underlying set, such as $\mathbb{Z}$.

This demonstrates that the distinguishing characteristics of binary operations closure, associativity, commutativity, identity, and inverses play a crucial role in determining the algebraic structure formed by the operation.

4.1.4 Activity 2 Part 1

1. Explain the attributes of various binary operations executed within a common set.

2. Check whether the operations applied to pertinent sets qualify as binary operations.

a. Subtraction on set of natural numbers.

b. Exponential operation on set integers.

Solution

1. Attributes of Binary Operations on a Set

A binary operation on a set $S$is a rule that assigns to every ordered pair of elements (a, b) in $S \times S$ a unique element $c \in S$. In other words, it is a function

$$*: S \times S \rightarrow S$$

For a binary operation to be well-defined, the operation must satisfy certain key attributes.

Closure

Definition: A set S is closed under a binary operation $*$ if performing the operation on any two elements of the set always produces another element in the set.

$$\forall a, b \in S, \qquad a * b \in S$$

Example: Addition on integers ($\mathbb{Z}$) is closed because the sum of any two integers is always an integer.

Non-example: Subtraction on natural numbers ($\mathbb{N}$) is not closed, because $3 - 5 = -2 \notin \mathbb{N}$.

Associativity

Definition: A binary operation $*$ is associative if the grouping of elements does not affect the result.

$$(a * b) * c = a * (b * c) \ \ \forall a, b, c \in S$$

Example: Addition and multiplication on integers ($\mathbb{Z}$) are associative.

Non-example: Subtraction is not associative because $(5 - 3) - 2 = 0$, but $5 - (3 - 2) = 4$

Commutativity

Definition: A binary operation $*$ is commutative if the order of elements does not affect the result.

$$a * b = b * a \ \forall a, b \in S$$

Example: Addition and multiplication of integers are commutative

Non-example: Subtraction and division are not commutative $(5 - 3 \neq 3 - 5)$

Identity Element

Definition: An element $e \in S$ is an identity for a binary operation $*$ if it leaves every element unchanged when combined with it.

$$a * e = e * a = a, \qquad \forall a \in S$$

Example: 0 is the additive identity in $\mathbb{Z}$, and 1 is the multiplicative identity.

Inverse Element

Definition: For a given binary operation $*$, an element $b \in S$ is the inverse of $a \in S$ if $a * b = b * a = e$, where e is the identity element.

Example: For addition on $\mathbb{Z}$, the inverse of a is $-a$

Idempotence

Definition: An operation $*$is idempotent if applying it to two identical elements gives the same element

$$a * a = a, \forall a \in S$$

Example: The operation $\min(a, b)$ or $\max(a, b)$on numbers is idempotent

A binary operation on a set is characterized by these attributes, but not all operations have all properties.

Closure is essential: without it, the operation cannot even be considered a binary operation on the set.

Other attributes like associativity, commutativity, identity, inverse, and idempotence determine additional algebraic structures such as semigroups, monoids, and groups.

2. Check whether the operations applied to pertinent sets qualify as binary operations

a. Subtraction on the set of natural numbers

$$-: N \times N \to N$$

$$(x, y) \mapsto x - y$$

The operation of subtraction is not a binary operation on the set of natural numbers $\mathbb{N}$because it lacks closure.

Choose two elements $x, y \in \mathbb{N}$, for example, $x = 6$and $y = 8$

Apply the operation

$$x - y = 6 - 8 = -2$$

The result -2 is a negative integer, which is not in $\mathbb{N}$

Since there exists at least one pair of elements in $\mathbb{N}$whose difference is not in $\mathbb{N}$, the subtraction operation does not map $\mathbb{N} \times \mathbb{N}$to $\mathbb{N}$. Therefore, it does not qualify as a binary operation on $\mathbb{N}$.

b. Exponential operation on the set of integers

$$\hat{}: \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$$
$$(x, y) \mapsto x^y$$

The exponential operation is not a binary operation on the set of integers $\mathbb{Z}$because it lacks closure.

Counterexample 1 (Negative exponent)

Choose $x = 3$and $y = -2$, both in $\mathbb{Z}$

Apply the operation

$$x^y = 3^{-2} = \frac{1}{3^2} = \frac{1}{9}$$

The result $\frac{1}{9}$ is a fraction, not an integer, so it is not in $\mathbb{Z}$

Counterexample 2 (Zero base with negative exponent)

Choose $x = 0$and $y = -1$

Apply the operation

$$x^y = 0^{-1} = \frac{1}{0}, \, which \, is \, undefined$$

An undefined result cannot be an element of $\mathbb{Z}$

Since there exist pairs of integers whose exponentiation results in a non-integer fraction or undefined value, the operation does not map $\mathbb{Z} \times \mathbb{Z}$to $\mathbb{Z}$. Therefore, it does not qualify as a binary operation on $\mathbb{Z}$.

4.2 (P8) Determine the order of a group and the order of a subgroup in given examples

4.2.1 Definition: Orders in Group Theory

The order of a group G is the number of elements contained in G. If Ghas a finite number of elements, it is called a finite group, and its order $|\ G\ |$is a positive integer.

The order of an element $g \in G$is the smallest positive integer $n > 0$such that, $g^n = e$, where $e$is the identity element of the group. If no such integer exists, the element g is said to have infinite order.

The order of a subgroup $H \leq G$ is simply the number of elements in H. For a finite group G, the quantity $|H|$ must divide $|G|$. This divisibility condition is guaranteed by Lagrange's Theorem.

Lagrange's Theorem

If G is a finite group and $H \leq G$, then:

$$|H| | |G|$$

Moreover, the number of left cosets of H in G is $|G|/|H|$

Idea of the proof

- Every left coset $gH = \{gh: h \in H\}$ has exactly $|H|$ elements

- The cosets partition G without overlap

- Therefore: $|G| = (number\ of\ cosets) \cdot |H|$. Thus, $|H|$ divides $|G|$

4.2.2 Determining Orders: Principles and Methods

Several methods can be used to determine the order of a group, an element, or a subgroup.

Listing elements

If every element of G is explicitly known, the order of the group is obtained by counting them.

Constructing a Cayley

A Cayley table helps verify closure, associativity (if already known or assumed), the identity element, and inverses. It is especially effective for analyzing small groups.

Identifying the cyclic subgroup generated by an element

For an element $g \in G$:

$$\langle g \rangle = \{e, g, g2, \dots\}$$

The number of distinct elements in $\langle g \rangle$ is precisely the order of g.

Applying Lagrange's Theorem

Every subgroup H of a finite group G must satisfy.

$$|H| | |G|$$

This restricts the possible subgroup orders and is particularly useful for classification.

4.2.3 Cayley Tables for All Non-Isomorphic Groups of Order 1–4

Group of Order 1

The unique group of order 1 is the trivial group:

$$G = \{e\}$$

Table 7: Group of Order 1 (Cayley)

$\circ$	e
e	e

Group order: $\mid G \mid = 1$

Every element has order 1

Group of Order 2

Every group of order 2 is isomorphic to the cyclic group:

$$C_2 = \{e, a\}, a^2 = e$$

Table 8: Group of Order 2 (Cayley)

$\circ$	e	a
e	e	a
a	a	e

$$\mid G \mid = 2$$

$$o(e) = 1, \qquad o(a) = 2$$

Subgroups: $\{e\}$ and G

Group of Order 3

There is only one group of order 3, the cyclic group:

$$C_3 = \{e, a, a^2\}, a^3 = e$$

Table 9: Group of Order 3 (Cayley)

$\circ$	e	a	a^2
e	e	a	a^2
a	a	a^2	e
a^2	a^2	e	a

$$\mid G \mid = 3$$

Orders: $o(e) = 1, o(a) = 3, o(a^2) = 3$

Subgroups: $\{e\}, G$

Group of Order 4

There are exactly two non-isomorphic groups of order 4: the cyclic group $C_4$and the Klein four-group V_4.

Cyclic group C_4

$$C_4 = \{e, a, a^2, a^3\}, a^4 = e$$

Table 10: Cyclic group C_4

$\circ$	e	a	a^2	a^3
e	e	a	a^2	a^3
a	a	a^2	a^3	e
a^2	a^2	a^3	e	a
a^3	a^3	e	a	a^2

Orders: $o(e) = 1, o(a) = 4, o(a^2) = 2, o(a^3) = 4$

Subgroups: , $\{e, a^2\}$, C_4

Klein four-group V_4

$V_4 = \{e, u, v, w\}$, in which every non-identity element has order 2

Table 11: Klein four-group V_4

$\circ$	e	u	v	w
e	e	u	v	w
u	u	e	w	v
v	v	w	e	u
w	w	v	u	e

Orders: $o(e) = 1; o(u) = o(v) = o(w) = 2$

Subgroups: $\{e, u\}, \{e, v\}, \{e, w\}, V_4$

Although both groups have order 4, their structures differ significantly

In C_4, an element of order 4 exists; in V_4, no such element exists

4.2.4 Examples: Determining Group and Subgroup Orders

Example 1: The Group $\mathbb{Z}_6$

$$\mathbb{Z}_6 = \{0,1,2,3,4,5\}$$

Group order: $|\mathbb{Z}_6| = 6$

The subgroup generated by an element k is: $\langle k \rangle = \{0, k, 2k, 3k, \dots\}$

Subgroups:

- $\langle 0 \rangle = \{0\}$(order 1)

- $\langle 1 \rangle = \mathbb{Z}_6$(order 6)

- $\langle 2 \rangle = \{0, 2, 4\}$ (order 3)

- $\langle 3 \rangle = \{0, 3\}$ (order 2)

- $\langle 4 \rangle = \langle 2 \rangle$ (order 3)

- $\langle 5 \rangle = \mathbb{Z}_6$ (order 6)

Possible subgroup orders: 1,2,3,6, all of which divide 6

Example 2: The Symmetric Group S_3

Elements: $e, (12), (13), (23), (123), (132)$

Subgroups:

- Order 1: $\{e\}$

- Order 2: three subgroups, each generated by a transposition (e.g., $\{e, (12)\}$)

- Order 3: one cyclic subgroup generated by a 3-cycle

- Order 6: S_3 itself

No subgroups of order 4 or 5 exist since these do not divide 6

Applications of Lagrange's Theorem

Application A: Subgroups of S_3

Since $|S_3| = 6$, possible subgroup orders are: 1,2,3,6. This matches the actual subgroup classification

Application B: Checking for Subgroups of Order 4 in Groups of Order 8

If $|G| = 8$, subgroups of order 4 may exist, because $4 \mid 8$.

However, existence is not guaranteed.

Examples:

- $\mathbb{Z}_8$ has a cyclic subgroup of order 4

- The quaternion group Q_8 also contains subgroups of order 4

Lagrange provides a necessary condition (divisibility), not a sufficient one.

Application C: Subgroups of $\mathbb{Z}_6$

Since $|\mathbb{Z}_6| = 6$, only subgroup orders 1, 2, 3, 6 are possible no subgroup can have order 4 or 5.

The order of a group can be determined by direct enumeration or structural analysis.

The order of an element is the smallest exponent producing the identity.

Subgroups can be identified by evaluating $\langle g \rangle$ for various generators or by analyzing cosets.

Lagrange's Theorem is a powerful tool for eliminating impossible subgroup orders, though it does not guarantee existence for all divisors

Cayley tables are invaluable for studying small groups, confirming closure, and identifying identities and inverses.

4.2.2 Activity 2 Part 2

1. Construct the operation tables for group G with orders 1, 2, 3, and 4, utilizing the elements a, b, c and e as the identity element in a suitable manner.

Solution

The operation table (or Cayley table) displays the result of the binary operation between every pair of elements in a finite group G. Since these groups are small, they are generally cyclic or can be constructed based on the properties of a group (closure, identity element e, and inverses).

Group of Order 1: $G_1 = \{e\}$

This is the trivial group, containing only the identity element e

Table 12: Order 1

∘	e
e	e

Group of Order 2: $G_2 = \{e, a\}$

The identity is e. Since every element must have an inverse, and $a \neq e$, a must be its own inverse (i.e., $a \circ a = e$).

Table 13: Order 2

∘	e	a
e	e	a
a	a	e

Group of Order 3: $G_3 = \{e, a, b\}$

Any group of prime order is cyclic. Let a be the generator.

Table 14: Order 3

∘	e	a	b
e	e	a	b
a	a	b	e
b	b	e	a

– Explanation:

  + a∘a must be the only remaining element, b.
  + a ∘ b must be the identity e (since G_3 is cyclic, $a^3 = e$).
  + b ∘ a: Since the group is cyclic (and thus Abelian), b ∘ a = a ∘ b = e.
  + b ∘ b: Since $b = a^2, b \circ b = a^4 = a^3 \circ a = e \circ a = a$.

Group of Order 4: $G_4 = \{e, a, b, c\}$

There are two distinct group structures of order 4 (up to isomorphism):

– The Cyclic Group $\mathbb{Z}_4$(e.g., generated by a)

– The Klein Four-Group V_4 (every non-identity element is its own inverse).

Case 1: Cyclic Group $\mathbb{Z}_4$

$G_4 = \{e, a, a^2, a^3\}$. Let $b = a^2$ and $c = a^3$. The generator a has order 4, so $a^4 = e$.

Table 15: Cyclic Group $\mathbb{Z}_4$

∘	e	a	b	c
e	e	a	b	c
a	a	b	c	e
b	b	c	e	a
c	c	e	a	b

Case 2: Klein Four-Group V_4

In this group, every non-identity element has order 2, meaning $a \circ a = e, b \circ b = e, and\ c \circ c = e$

Table 16: Klein Four-Group V_4

∘	e	a	b	c
e	e	a	b	c
a	a	e	c	b
b	b	c	e	a
c	c	b	a	e

– Explanation: Due to closure, the product of any two distinct non-identity elements must be the third non-identity element. For instance:

+ a ∘ b cannot be a (since b ≠ e).

+ a ∘ b cannot be b (since a ≠ e).

+ a ∘ b cannot be e (since b is the inverse of b, and a ≠ b).

Therefore, a ∘ b = c.

2. State the Lagrange's theorem of group theory. Using this theorem to discuss whether a group K with order 4 can be a subgroup of a group G with order 9 or not. Provide a clear exposition of the reasons.

Solution

a) Statement of Lagrange's Theorem

Lagrange's Theorem states that for any finite group G, the order (the number of elements) of any subgroup H of G must be a divisor of the order of G.

Mathematically, if H is a subgroup of a finite group G, then:

$$|H| \text{ divides } |G|$$

where |G| is the order of group G and |H| is the order of subgroup H.

b) Discussion using Lagrange's Theorem

We are asked to discuss whether a group K with order 4 (i.e., |K|=4) can be a subgroup of a group G with order 9 (i.e., |G|=9).

Conclusion: No, the group K cannot be a subgroup of group G.

c) Clear Exposition of the Reasons:

- Identify the Orders:
  + The order of the potential subgroup K is $|K| = 4$.
  + The order of the group G is $|G| = 9$.
- Apply Lagrange's Theorem:
  + If K were a subgroup of G, then according to Lagrange's Theorem, the order of K (|K|=4) must divide the order of G (|G|=9).
- Check for Divisibility:
  + We check if 4 divides 9.
  + $\frac{9}{4} = 2.25$, which is not an integer.
  + Since 4 is not a divisor of 9, the condition required by Lagrange's Theorem is violated.

Therefore, because the order of group K (4) does not divide the order of group G (9), K cannot exist as a subgroup of G.

4.3 (M4) Validate whether a given set with a binary operation is indeed a group

To determine whether a mathematical structure consisting of a set S and a binary operation $\circ$, denoted as $(S,\circ)$, constitutes a Group, we must perform a rigorous verification process based on the group axioms. Furthermore, determining the number of possible binary operations that can be established on a finite set helps us better understand the scope of potential algebraic structures.

4.3.1 Group Validation Criteria

A set G combined with a binary operation $\circ$ is recognized as a group if and only if it fully satisfies the following four fundamental algebraic properties.

- Closure Property: The operation must preserve the nature of the set. Specifically, for any arbitrary pair of elements a, b belonging to G, the result of the operation $a \circ b$ must be a defined element that also lies within G. If the result falls outside the set, the structure is immediately invalidated.

- Associativity: The order in which the operation is performed must not alter the final result. For all $x, y, z \in G$, the following equality must always hold.

$$(x \circ y) \circ z = x \circ (y \circ z)$$

This property ensures consistency in long sequences of operations without concern for the placement of parentheses.

- Existence of Identity**:** There must exist a unique element e in the set (called the neutral or identity element) such that it does not change the value of other elements when the operation is performed. For all $x \in G$.

$$x \circ e = x \, and \, e \circ x = x$$

A classic example is the number 0 in addition or the number 1 in multiplication.

- Existence of Inverse: For every element $x \in G$, it is mandatory to find a corresponding element $y \in G$ (denoted as x^{-1}) such that when combined, they yield the identity element.

$$x \circ y = e \, and \, y \circ x = e$$

This ensures that every effect of the operation can be reversed.

If a set satisfies the first three properties but lacks the inverse property, it is considered only a Monoid, not a complete Group.

4.3.2 Formula for Binary Operations

Beyond verifying properties, we can determine the scale of operations that can be defined on a set. Since a binary operation is essentially a mapping from the Cartesian product $S \times S$ to S ($\alpha: S \times S \to S$), the quantity of operations depends on the order (number of elements) of that set.

If set S has an order of n (i.e., $|S| = n$), the formula for the total number of possible binary operations is

$$N = n^{(n^2)}$$

The domain $S \times S$ contains n^2 pairs of elements. Each of these pairs can map to one of the n values in S. Therefore, the total number of possibilities is $n \times n \times \ldots \times n$ (n^2 times)

4.3.3 Case Validation

Case A: The set of Integers $\mathbb{Z}$ with Addition $(+)$ To conclude that $(\mathbb{Z}, +)$ is a group, we verify step by step:

- Closure: The sum of any two integers is always an integer ($a + b \in \mathbb{Z}$). (Satisfied).

- Associativity: Addition is always associative: $(a + b) + c = a + (b + c)$. (Satisfied).

- Identity: The number 0 is the identity element because adding 0 does not change the value ($a + 0 = a$). (Satisfied).

- Inverse: Every integer a has an opposite $-a$ such that $a + (-a) = 0$. (Satisfied). → Conclusion: $(\mathbb{Z}, +)$ is a Group.

Case B: The set of Integers $\mathbb{Z}$ with the operation $*$ ($a * b = a^2 + b$) Let us consider the operation $a * b = a^2 + b$.

- Closure: The result $a^2 + b$ is always an integer. (Satisfied)

- Associativity: We compare $(a * b) * c$ and $a * (b * c)$ with test values $a = 1, b = 2, c = 3$:

Left-hand side: $(1 * 2) * 3 = (1^2 + 2) * 3 = 3 * 3 = 3^2 + 3 = 12$

Right-hand side: $1 * (2 * 3) = 1 * (2^2 + 3) = 1 * 7 = 1^2 + 7 = 8$

Since $12 \neq 8$, the operation is not associative. → Conclusion: Due to the violation of associativity, the structure $(\mathbb{Z}, *)$ is NOT a Group.

Case C: Calculation of Operation Quantity Consider a small set $A = \{0, 1\}$ with order $n = 2$. Applying the stated formula: The number of binary operations that can be defined on A is

$$2^{(2^2)} = 2^4 = 16$$

This result demonstrates that even with a set of only 2 elements, we can construct up to 16 different truth tables.

4.3.4 Activity 2 Part 3

1. Check whether the set $S = \mathbb{R}\setminus\{-1\}$ is a group under the binary operation defined as $a * b = a + b - ab$.

To determine if $(S,*)$ is a group, we must verify the four fundamental group axioms: closure, associativity, identity, and inverseness.

Closure: We must check if the result of $a * b$ is always in S for any $a, b \in S$. This means $a * b$ must never equal -1. Let's test if $a * b = -1$.

$$a + b - ab = -1$$

$$a + 1 + b - ab = 0$$

$$(a+1) + b(1-a) = 0$$

$$(a+1) - b(a-1) = 0$$

$$b = \frac{a+1}{a-1}$$

If we choose $a = 2$ (which is in S) and calculate b

$$b = \frac{2+1}{2-1} = 3$$

Since $b = 3$ is also in$S$, we have found two valid elements (2 and 3) where

$$2 * 3 = 2 + 3 - (2)(3) = 5 - 6 = -1$$

Since -1 is excluded from set S, the operation is not closed.

Associativity: We check whether $(a * b) * c = a * (b * c)$.

Compute the Left Side (LHS)

$$(a * b) * c = (a - b - ab) * c$$

$$= (a + b - ab) + c - (a + b - ab)c$$

$$= a + b + c - ab - ac - bc + abc$$

Compute the Right Side (RHS)

$$a * (b * c) = a * (b + c - bc)$$

$$= a + (b + c - bc) - a(b + c - bc)$$

$$= a + b + c - bc + ab + ac + abc$$

Since LHS = RHS, associativity holds

Identity Element: We need an element $e \in S$ such that $a * e = a$ for all a

$$a + e - ae = a$$

$$e - ae = 0$$

$$e(1 - a) = 0$$

Since this must hold for any a, we must have $e = 0$. Since $0 \in \mathbb{R}\setminus\{-1\}$, the identity exists.

Inverse Element: For every $a \in S$, we must find $x \in S$ such that $a * x = e$ (where $e = 0$)

$$\begin{aligned} &a + x - ax = 0 \\ &x(1 - a) = -a \\ &x = -\frac{a}{1-a} = \frac{a}{a-1} \end{aligned}$$

For the inverse to exist, the denominator must not be zero ($a \neq 1$). However, the element 1 is included in set S (since S only excludes -1). If $a = 1$, the inverse x is undefined. Therefore, the element 1 has no inverse. Inverseness fails.

The set $S = \mathbb{R}\setminus\{-1\}$ is not a group under the operation $a * b = a + b - ab$

2. Express the connection between the order of a group and the quantity of binary operations that can be defined on that set. Apply to answer the question: "What is the total number of binary operations that can be defined on a set containing 3 elements?"

Connection between Set Order and Binary Operations: To determine the number of possible binary operations, we must look at the definition of a binary operation relative to the size (order) of the set

A binary operation on a set S is formally defined as a function from the Cartesian product $S \times S$ to S

$$f: S \times S \rightarrow S$$

Input Space: If the order of set S is n (meaning it has n elements), the Cartesian product $S \times S$ contains $n \times n = n^2$ ordered pairs. These represent all the possible "cells" in an operation table (Cayley table).

Output Choices: For each of these n^2 pairs, the operation must assign exactly one result. This result can be any of the n elements in the set

Formula**:** Since there are n^2 independent choices to be made, and each choice has n possibilities, the total number of distinct binary operations is:

$$N = n^{(n^2)}$$

Application**:** We apply this formula to a set containing 3 elements ($n = 3$)

The number of input pairs is $3^2 = 9$

Each pair can result in one of 3 distinct values

Total operations:

$$3^{(3^2)} = 3^9$$

Calculating the value

$$3^9 = 19{,}683$$

Answer: There are 19,683 distinct binary operations that can be defined on a set containing 3 elements

CONCLUSION

This report has systematically explored the core concepts of discrete mathematics and their practical applications within computer science and software engineering. By addressing the four key problem areas outlined in the assignment, the following conclusions have been reached

Firstly, regarding Set Theory and Functions (Chapter 1), the report successfully addressed data processing challenges through set operations and Venn diagrams. The application of the Bag (multiset) concept allowed for accurate cardinality calculations in collections with duplicate data, while the analysis of inverse functions clarified the mechanisms behind data transformation and cryptography

Secondly, in Graph Theory (Chapter 2), modeling real-world problems using trees and weighted graphs provided optimal solutions for network routing. Specifically, Dijkstra's algorithm was effectively applied to determine the shortest path between nodes (from A to Z), and the analysis of Euler and Hamilton circuits verified the connectivity and traversability of the network structures

Thirdly, concerning Boolean Algebra (Chapter 3), the report demonstrated the essential role of mathematical logic in hardware design. Practical scenarios, such as security systems and automated irrigation, were successfully translated into Boolean equations and logic gate diagrams. Furthermore, algebraic methods were utilized to simplify complex expressions, proving that logical reduction leads to optimized system performance

Finally, in Abstract Algebra (Chapter 4), the report delved into the fundamental structures of mathematics. By rigorously verifying properties such as closure, associativity, commutativity, identity, and inverse, the distinguishing characteristics of binary operations were established. The application of Lagrange's Theorem further enabled the accurate determination of subgroup orders within finite group structures

In conclusion, this report affirms that discrete mathematics provides the necessary logical reasoning tools and rigorous structures required to analyze, model, and solve complex engineering problems, ranging from data management to algorithm optimization and circuit design

EVALUATION

Assessment of Accuracy and Effectiveness The results presented in this report are mathematically accurate and adhere to the fundamental theorems of discrete mathematics

- Accuracy: The application of Dijkstra's algorithm in Chapter 2 correctly identified the shortest path (Cost = 20) by systematically verifying all node weights. Similarly, the Boolean simplifications in Chapter 3 were verified using standard algebraic laws (Idempotent, Complement, Distributive), ensuring that the final logic gates produce the exact same output as the original complex equations

- Effectiveness: The solutions proposed demonstrate high effectiveness. Using "Bags" for data with duplicates is a more realistic approach for database management than standard sets. The simplification of Boolean functions significantly reduces the number of logic gates required, which directly translates to cost savings and increased processing speed in hardware design

Strengths	Weaknesses
Theoretical Solidity: The report is grounded in proven mathematical theories (Lagrange's Theorem, Graph Theory), ensuring the reliability of the conclusions	Manual Calculation Limits: The algorithms (like Dijkstra) and truth tables were executed manually. While accurate for small datasets, this method is prone to human error if scaled up
Practical Applicability: Concepts are not just abstract but applied to real-world scenarios like Smart Home Security (Boolean logic) and Network Routing (Dijkstra)	Static Analysis: The solutions provide a snapshot of optimal logic but do not account for dynamic changes in variables (e.g., changing edge weights in real-time)
Step-by-Step Clarity: Complex problems, such as finding the shortest path or verifying group axioms, are broken down into clear, verifiable steps	
Opportunities	Threats
Software Integration: The mathematical models developed here (e.g., the logic for the irrigation	Computational Complexity: In Graph Theory, as the number of nodes increases, finding

system) can be directly coded into software algorithms (using Python or C++) to automate decision-making	Hamiltonian circuits becomes NP-Complete. The current manual approach would fail in large-scale network analysis
Optimization: Further reduction of Boolean circuits using Karnaugh Maps (K-Map) could be explored for even greater efficiency than algebraic simplification	Abstract Complexity: Misinterpreting abstract algebraic structures (like distinguishing between a Monoid and a Group) can lead to fundamental errors in system architecture design

The approaches taken in this assignment provide robust and logically sound solutions. The bridge between abstract mathematics and software engineering applications has been successfully established, proving that discrete structures are essential for building efficient, secure, and optimized computing systems

REFERENCES

Epp, S. S. (2019). *Discrete mathematics with applications* (5th ed.). Boston, MA: Cengage Learning. [Accessed: 5 November 2025]

Grimaldi, R. P. (2017). *Discrete and combinatorial mathematics: An applied introduction* (5th ed.). Harlow: Pearson Education. [Accessed: 5 November 2025]

Rosen, K. H. (2019). *Discrete mathematics and its applications* (8th ed.). New York: McGraw-Hill Education [Accessed: 5 November 2025]