ASSIGNMENT FINAL REPORT

Qualification	**Pearson BTEC Level 5 Higher National Diploma in Computing**		
Unit number and title	**Unit 18: Discrete Maths**		
Submission date	9/12/2025	**Date Received 1st Submission**	
Re-submission Date		**Date Received 2nd Submission**	
Student Name	Phu Tuong Long	**Student ID**	BD00772
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	LONG

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	12/10/2025	**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
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.

P5	P6	P7	P8	M3	M4	D3	D4

OBSERVATION RECORD

Student	**Phu Tuong Long**

Description of activity undertaken

For P5: I diagrammed binary problems by creating logic gate schematics for specific domains, including a 'Smart Home Security System' and 'Agricultural Irrigation Control'.

For P6: I analyzed scenarios regarding a 'Secure Facility Door' and 'Computer Login' to produce truth tables and derive their corresponding Boolean equations from the logic requirements.

For M3: I utilized algebraic methods and Boolean laws (such as Distributive, Idempotent, and Complement laws) to simplify complex Boolean equations into their most efficient forms."

Assessment & grading criteria

How the activity meets the requirements of the criteria

Student signature:	**LONG**	**Date:**	**12/10/2025**
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

Project Overview and Main Objectives This report serves as the comprehensive submission for Assignment 1 of Unit 18: Discrete Maths. The primary objective of this project is to examine and apply fundamental discrete mathematical concepts that are essential to Software Engineering. This includes demonstrating a theoretical understanding and practical application of set theory, graph theory, Boolean algebra, and abstract algebra.

Problem Statement and Proposed Solutions In the field of computing, professionals frequently encounter complex logical and structural problems, such as optimizing network routing, managing data collections, and designing efficient logic circuits. This report addresses these issues by modeling them using discrete structures. The proposed solutions involve using Dijkstra's algorithm to find shortest paths in weighted graphs, utilizing Boolean algebra to simplify logic gates for security systems, and applying group theory to validate algebraic structures.

Summary of Key Sections The report is organized into four main chapters:

- Chapter 1 examines set theory, functions, and multisets (bags), including algebraic operations and cardinality calculations.
- Chapter 2 analyzes mathematical structures using graph theory, focusing on binary trees, Dijkstra's algorithm, and Eulerian/Hamiltonian paths.
- Chapter 3 investigates Boolean algebra, detailing the creation of truth tables, logic gate diagrams, and equation simplification.
- Chapter 4 explores abstract algebra, specifically the characteristics of binary operations and the determination of group and subgroup orders.

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

A set is defined as an unordered collection of elements. The concept of a set is fundamental to modern mathematics and much of computer science. The elements contained within a set can be various objects, such such as numbers, points in space, or even other sets. (Haggard et al,)

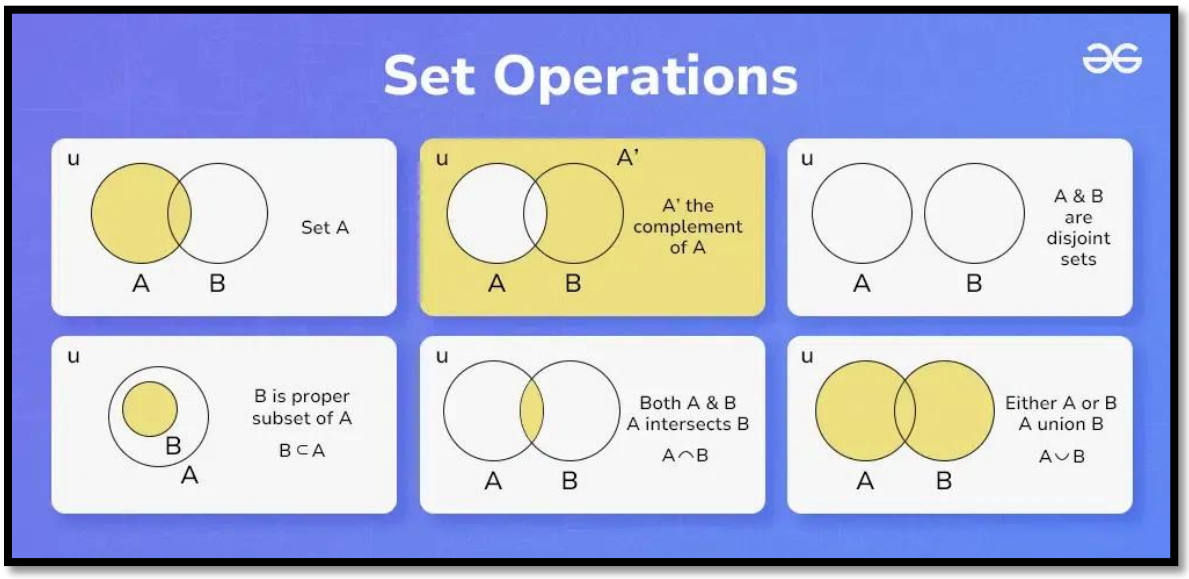

Figure 1 - 1: Set Operations

Methods of Representing Three primary methods to describe a set:

A. Listing Elements

Sets can be described by listing all of their elements inside curly braces { }. For larger finite sets, the list can be abbreviated for example:

- The set K of positive integers less than 100 can be denoted by K ={1,2,3,...,99}.
- The set O of odd positive integers less than 10 can be expressed by O ={1,3,5,7,9}

B. Describing by a Property (Set Builder Notation)

A set can be described in terms of some property $P(x)$ that its elements satisfy.

Notation Example:

- The set O of all odd positive integers less than 10 can be written as O ={ x | x is an odd positive integer less than 10}
- The set Q+ of all positive rational numbers can be written as Q+ ={x ∈ R | x= p q , for some positive integers p and q}

C. Describing Elements in a Context (Preferred Set Builder Notation)

This method describes the elements as belonging to some other specified set D while also satisfying a property. This form is generally preferred.

- Notation Example: $\{x \in \mathrm{Z} : x > -1/2 \text{ and } x < 19/2\}$.
- The symbol ∈ indicates membership in the larger set D (the context).

Key Notation and Terminology

The following fundamental symbols are used in set-theoretic notation:

Symbol	Meaning
∈	is an element of A (or is in A)
∉	is not an element of A
$A=B$	Sets A and B are equal
$A \subseteq B$	A is a subset of B (every element of A is also an element of B)
$A \subset B$	A is a proper subset of B ($A \subseteq B$ and $A \neq B$)
∅	The empty set (contains no elements)
$\mathrm{P}(A)$	The power set of A (the set of all subsets of A)

Table 1 - 1: Symbol & Meaning

Special Sets of Numbers

Mathematicians commonly use special symbols to represent certain sets of numbers:

Symbol	Meaning	Elements
N	Natural numbers (non-negative integers)	{0,1,2,3,…}
Z	Integers	{…,−2,−1, 0,1,2,3,…}
Q	Rational numbers	(Fractions of integers)
R	Real numbers	(π,e,2, etc.)
C	Complex numbers	(π,e,1,2,-1,-2, 5/2 etc.)

Table 1 - 2: Commonly use symbol

1.1.2 Basic Set Operations and Symbols

Union

The union of two sets is an operation that combines them into a single set containing elements from either or both.

Term	Notation	Description
Union	A∪B	A union B.
Definition	{x:x ∈ A or x ∈ B}	The set containing every element that is in *A*, or *B*, or both.
Logical Form	x∈A∪B⟺(x∈A)∨(x∈B)	An element *x* is in the union if *x* belongs to *A* or *x* belongs to *B* (or both).

Table 1 - 3: Union

Key Properties of Union:

- The Union operation is commutative, meaning the order does not matter: $A \cup B = B \cup A$.
- It is associative: $A \cup (B \cup C) = (A \cup B) \cup C$. This property allows the union of three or more sets to be defined without ambiguity, as $A \cup B \cup C$ is equivalent to either way of grouping the terms.
- The empty set (∅) has no effect on the union: $A \cup \emptyset = A$.
- Any set *A* is a subset of the union $A \cup B$.

Intersection

The intersection of two sets is the set formed by the elements common to both sets.

Term	Notation	Description
Intersection	A∩B	A intersect B.
Definition	{x:x∈A and x∈B}	The set containing all elements that are elements of both A and B.
Logical Form	x∈A∩B⟺(x∈A)∧(x∈B)	An element x is a member of A∩B if x belongs to A and x belongs to B.

Table 1 - 4: Intersection

Key Properties of Intersection:

- The Intersection operation is commutative: $A \cap B = B \cap A$.
- It is associative: $A \cap (B \cap C) = (A \cap B) \cap C$.
- The intersection of a set with itself is the set itself: $A \cap A = A$.
- The intersection of any set with the empty set (∅) is the empty set: $A \cap \emptyset = \emptyset$.

Disjoint Sets:

Two sets A and B are called disjoint if their intersection is the empty set ($A \cap B = \emptyset$). This means they have no elements in common.

Set Difference

The set difference operation identifies elements belonging exclusively to the first set listed.

- Set Difference A−B (or A\B) he set of elements of A not in B.
- Definition: The set cotainning all element of A that are not element of B Example $\{ x : x \in A \text{ and } x \notin B \}$

Relationship to Complement: If U is the universal set and B denotes the complement of B (elements not in B relative to U or the domain of discourse), the set difference can be expressed using intersection: $A - B = A \cap B$

1.1.3 Venn Diagrams for Set Operations

Venn diagrams are a very convenient type of diagram frequently used to illustrate set-theoretic relationships. (Haggard et al)

Components of a Venn Diagram

- Universal Set: Discussions regarding sets are often restricted to elements and subsets of a fixed set, called the universal set (*U*) or universe of discourse. In a Venn diagram, the universal set is represented by the bounding rectangle.
- Subsets: Subsets of the universal set are represented by circles or ovals within the rectangle.

Set operations are illustrated by shading the specific region(s) within these circles and the bounding rectangle that correspond to the resulting set.

Illustration of Basic Set Operations

Union

The union of two sets, *A* and *B*, denoted *A*∪*B*, is the set containing all elements that are members of *A* or *B* or both.

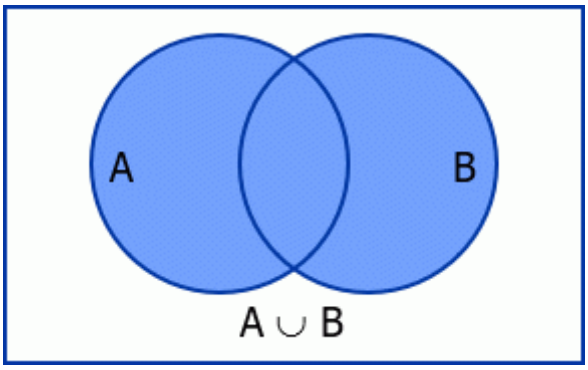

Figure 1 - 2: Union Venn Diagram

Venn Diagram Illustration: The shaded region for *A*∪*B* encompasses the entire area covered by both circles *A* and *B*.

Intersection

The intersection of two sets, *A* and *B*, denoted *A*∩*B*, is the set consisting of all elements that are common to both *A* and *B*.

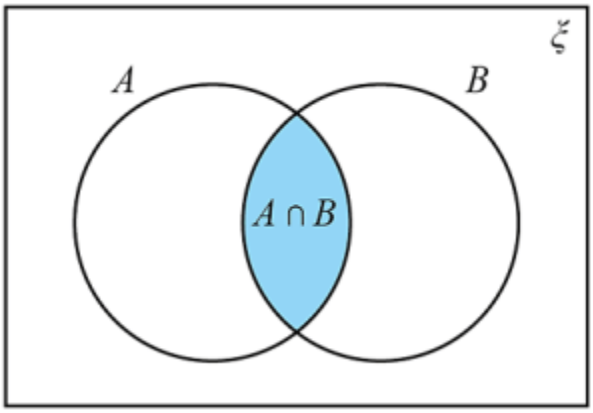

Figure 1 - 3: Intersection Venn Diagram

Venn Diagram Illustration: The shaded region for $A \cap B$ is the area of overlap where the circles representing A and B intersect.

Set Difference

The set difference of A and B, denoted $A - B$ (sometimes written $A \setminus B$), is the set of all elements that are in A but are not in B. This operation is also sometimes called the relative difference.

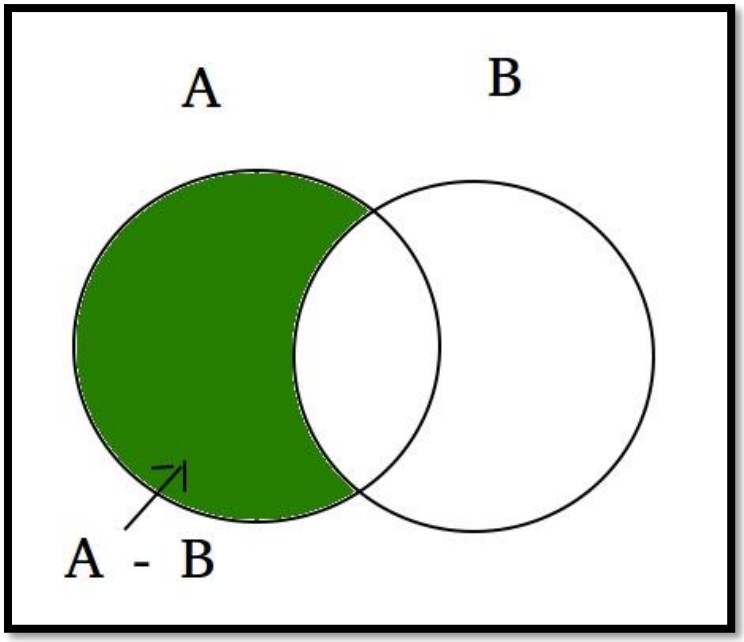

Figure 1 - 4: Set Difference Venn Diagram

Venn Diagram Illustration: The shaded area corresponds exclusively to the part of the circle A that lies outside the boundary of circle B.

1.1.4 Cartesian Product of Sets

Definition of Cartesian Product

The Cartesian product is an operation that creates a new set from two (or more) given sets, consisting of all possible combinations of elements from those sets arranged as ordered pairs or tuples.

For any sets *X* and *Y*, the product (also referred to as the Cartesian product or cross-product) is the set of all ordered pairs (*a*,*b*) such that the first component *a* is an element of *X* and the second component *b* is an element of *Y*.

- Notation: $A \times B = \{(a,b) : a \in A \text{ and } b \in B\}$.
- Ordered Pairs: The pairs within the product are ordered. This means that the pair $\langle a,b \rangle$ is notational distinct from the pair $\langle b,a \rangle$, unless $a=b$. Two ordered pairs $\langle a1,b1 \rangle$ and $\langle a2,b2 \rangle$ are equal if and only if $a1=a2$ and $b1=b2$.

Example 1: Product of two finite sets Let $X=\{1,2\}$ and $C=\{a,b\}$. The Cartesian product $X \times C$ is: $X \times C = \{(1,a),(1,b),(2,a),(2,b)\}$.

Example 2: Product of a set with itself Let $C=\{a,b\}$. The product $C \times C$ (or C^2) is: $C \times C = \{(a,a),(a,b),(b,a),(b,b)\}$.

Example 3: Product involving infinite sets Let N be the set of natural numbers (nonnegative integers). Let $S=\{a,b\}$. The product $\mathrm{N} \times S$ consists of all pairs whose first element is a nonnegative integer and whose second element is a or b: $\mathrm{N} \times \{a,b\} = \{(0,a),(0,b),(1,a),(1,b),(2,a),(2,b),\ldots\}$.

Example 4: Cardinality of the Product For finite sets, the cardinality (size) of the Cartesian product is the product of the cardinalities of the individual sets: $|P1 \times P2 \times \ldots \times Pn| = |P1| \mathrm{x} |P2| \ldots |Pn|$. For instance, if $A=\{1,2\}$ (so $|A|=2$) and $B=\{3,4,5\}$ (so $|B|=3$), the size of their product $|A \times B|$ is $2 \mathrm{x} 3=6$.

1.1.5 Applications in Computing Context

Figure 1 - 5: Set application

Foundational Structures and Modeling

Set theory serves as the foundation for defining critical mathematical structures frequently used in modeling and implementing solutions to problems in computer science.

- Relations and Functions: Sets are the basis for defining relations and functions.
  - A function, which formalizes the relationship between the input and the output for a computation, is defined as a special kind of set or relation.
  - A binary relation R between two sets A and B is formally defined as any subset of the Cartesian product A×B. Relations are used to formalize the familiar notion of a relationship between or among objects.
- Combinatorial Circuits and Boolean Algebra: Set operations are used as examples of operations that define Boolean algebras and lattices.

+ A Boolean algebra based on the elements {0,1} and operations like ∨ (join/OR) and ∧ (meet/AND) is used by computer scientists to model combinatorial circuits.

+ Functions used to describe the behavior of a circuit for each possible input are often Boolean functions realized by combinatorial networks.

Application in Databases

The entire concept of a relational database system is built upon set theory and relations.

- Data Structure: A relational database consists of a number of relations, which are sets of tuples (ordered n-tuples of data).
- Query Processing (Relational Algebra): Queries are processed using relational algebra, which utilizes set-theoretic operations such as selection, projection, join, and equijoin. The answer to any query is itself a relation (a set).

1.1.6 Activity 1 – Part 1

Requirements of Activity 1 – Part 1

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

1. Let A and B be two non-empty finite sets. Assume that cardinalities of the sets A, 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 of Activity 1 – Part 1

My student ID is: BD00772, therefore, according to the requirement that "a < b represents the largest digits in my ID", we have a = 2, b = 7

Exercise 1

Let A and B be two non-empty finite sets. Assume that cardinalities of the sets A B, and A B ∩ are 9 , 2 b a and a b + , respectively. Determine the cardinality of the set A ∪ B .

We are given two non-empty finite sets A and B with the following cardinalities:

$$|A| = \overline{9b} = 97$$

$$|B| = \overline{2a} = 22$$

$$|A \cap B| = a + b = 9$$

To determine the cardinality of the union $|A \cup B|$ we apply the principle of inclusion–exclusion, which states:

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

Substituting the given values:

$$A \cup B = 97 + 22 - 9$$

Final Answer:

$$A \cup B = 110$$

Exercise 2

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

We have:

$$|A - B| = \overline{3a} = 32$$

$$|A \cup B| = \overline{11b} = 117$$

$$|A \cap B| = \overline{1a} = 12$$

In addition,

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

By substituting the known values, we obtain:

$$|A| = 32 + 12 = 44$$

Next, we apply the principle of inclusion–exclusion to the union of two sets:

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

Substituting the given values, we have:

$$117 = 44 + |B| - 12$$

Simplifying this equation gives:

Finally, we find:

$$=> B = 117 - 44 + 12 = 85$$

Final Answer:

$$|B| = 85$$

Exercise 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?

We have:

Let U be the set of all customers at a local market,$|U| = \overline{35b} = 357$

Let F be the set of all customers have purchased fruits,$|F| = \overline{11a} = 112$

Let V be the set of all customers have purchased vegetables,$|V| = \overline{9b} = 97$

Let B be the set of all customers have purchased bakery, $|B| = \overline{\overline{8a}} = 84|B| = \overline{8a} = 82$

$F \cap V$ be the set of all customers have purchased both fruits and vegetables, $|F \cap V| = \overline{4B} = 47$

$V \cap B$ be the set of all customers have purchased both vegetables and bakery,$|V \cap B| = \overline{3b} = 37$

B ∩ F be the set of all customers have purchased fruits and bakery$|B \cap F| = \overline{2a} = 22$

$F \cap V \cap B$ be the set of all customers have purchased all three categories,$|F \cap V \cap B| = \overline{1a} = 12$

Use the inclusion–exclusion formula for three sets to find the number of customers who purchased at least one item

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

$$|\overline{F \cup B \cup V}| = 357 - (|F| + |B| + |V| - |F \cap B| - |F \cap V| - |B \cap V| + |F \cap B \cap V|)$$

Substitute the numbers:

$$|\overline{F \cup B \cup V}| = 357 - (112 + 82 + 97 - 22 - 47 - 37 + 12)$$

Customers have not purchased anything:

$$|\overline{F \cup B \cup V}| = 357 - 197 = 160$$

Answer: There are 160 customers who purchased nothing.

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

1.2.1 Definition of bag

Definition of Bag (Multiset)

A multiset is formally defined as an unordered collection of elements, where each element is allowed to appear any number of times. (Haggard, 2006)

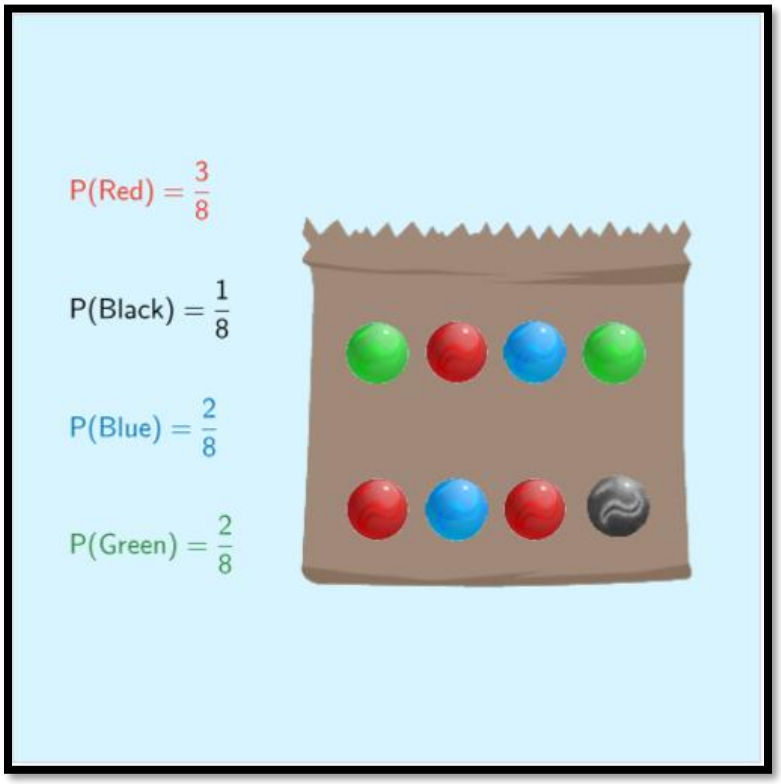

Figure 1 - 6: Bag example

Key characteristics include:

- Unordered: Like a standard set, the order of the elements does not matter.
- Repetition Allowed: Unlike a standard set, a multiset permits repeated elements.
- Multiplicity: The number of times an element appears in the collection is called its multiplicity.

Examples of Bags (Multisets)

- General Notation: The notation {1,2,2,5} represents a multiset. The cardinality of $\{a,a,a,b,c,c\}$ is 3+1+2=6.
- Counting Problems (Jelly Beans): If you grab a handful of ten jelly beans from a bag containing six flavors, the outcome can be represented as a multiset of flavors. A possible outcome expressed as a multiset is: $\{R,R,G,G,G,B,B,B,B,Y\}$ In this example:
  - The flavor R (red) has a multiplicity of 2.
  - The flavor B (blue) has a multiplicity of 4.
  - The flavor G (green) has a multiplicity of 3.
  - The flavor Y (yellow) has a multiplicity of 1.
  - The total number of elements (jelly beans) is 10.
- Cookie Distribution: The distribution of seven cookies to four kids can be represented by a multiset where the multiplicity of a child's name in the multiset is the number of cookies that child receives. For instance, the multiset $\{A,A,A,B,C,D,D\}$ represents an outcome where A gets 3 cookies, B gets 1, C gets 1, and D gets 2.

1.2.2 Operations on Bags

A bag, also known as a multiset, is an unordered collection of elements where each element is permitted to appear any number of times. The number of times an element appears is called its multiplicity. (Levin)

Formally, a multiset *M* over a set *S* is defined as a function $M:S\rightarrow$N (natural numbers). The value *M*(*a*) represents the multiplicity of element *a* in the multiset *M*.

Union of Bags

Given the two bags as follows:

$M1=\{a,a,a,b,c,c,c,d\}$ $M2=\{a,c,c,d\}$,

The union of two multisets, *M*1 and *M*2, is a multiset $M1\cup M2$ where the multiplicity of any element *a* is the sum of its multiplicities in *M*1 and *M*2.

Definition: For all elements *a* in the underlying set *S*, the multiplicity is calculated as: $(M1\cup M2)(a)=M1(a)+M2(a)$.

Example: If $M1=\{a,a,a,b,c,c,c,d\}$ and $M2=\{a,c,c,d\}$, then the union $M1 \cup M2$ is $\{a,a,a,a,b,c,c,c,c,c,d,d\}$. or : {a:4,b:1,c:5;d:2}

Intersections of Bags

The intersection of two bags A and B, denoted as A∩B, is a bag such that the multiplicity of an element is equal to the minimum of the multiplicity of an element in A and B.

Example Given the two bags as follows:

- A={|a,m,m,m,n|} = {|a : 1,m : 3,n : 1|}
- B ={|a,a,m,m,n,n,n,n,r|} = {|a : 2,m : 2,n : 4,r : 1|}.
- We have A∩B={|a,m,m,n|} = {|a : 1,m : 2,n : 1|}

Difference of Bags

The difference of multisets M_1 and M_2 results in a multiset $M_1 - M_2$ where the multiplicity of an element a is the difference between its multiplicity in M_1 and M_2, provided this difference is non-negative.

Example: If $M1=\{a:3,b:1,c:3,d:1\}$ and $M2=\{a:1,c:2,d:1\}$, then the difference $M1-M2$ is $\{a:2,b:1,c:1\}$.

Sums of Bags

The sum of two bags A and B, denoted as A+B, is a bag such that the multiplicity of an element is equal to the sum of the multiplicity of an element in A and B.

Given the two bags as follows

- A={|a,m,m,m,n|} = {|a : 1,m : 3,n : 1|}
- B ={|a,a,m,m,n,n,n,n,r|} = {|a : 2,m : 2,n : 4,r : 1|}.
- We have A+B={|a,a,a,m,m,m,m,m,n,n,n,n,n,r|} = {|a : 3,m : 5,n : 5,r : 1|}

1.2.3 Cardinality of Bags

The cardinality of a bag A, denoted as |A|, is the sum of the multiplicities of all its elements.

Example: For the multiset $M=\{a,a,a,b,c,c\}$:

- Element a has a multiplicity of 3.

- Element *b* has a multiplicity of 1.
- Element *c* has a multiplicity of 2.

The cardinality |*M*| is the sum of these multiplicities: |*M*|=3+1+2=6

The cardinality (or size) of a finite multiset *M*, denoted by , is the total number of elements it contains, which is calculated as the sum of the multiplicities of all its elements.

1.2.4 Applications in Computing Context

Applications in Counting and Combinatorics

The primary explicit application of multisets in the sources lies within counting and combinatorics, which is crucial for analyzing algorithms and complexity:

Modeling Counting Problems with Repetition: Multisets are used to represent problems where the resulting collection requires counting repeated items. For instance, counting the number of ways to distribute identical objects (like cookies or books) to distinct recipients (like children or shelves) is naturally modeled as a multiset problem, where the multiplicity of a recipient's name is the number of items they receive.

Translating Complex Counts (Sticks and Stones): Multisets can be translated into specialized binary strings using a technique sometimes referred to as sticks and stones (or stars and bars). This transformation allows complex counting problems (where repeats are allowed) to be solved using familiar techniques from Pascal's triangle and counting combinations.

Algorithm Analysis Foundation: Determining the running time or storage requirements of a computational problem often depends on solving underlying counting problems. The capability of multisets to precisely model outcomes involving repetition makes them foundational for certain complex combinatorial arguments used in the analysis of algorithms (Attenborough, 2003)

1.2.5 Activity 1 – Part 2

Requirements of 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 provided numbers.

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 of Activity 1 – Part 2

My student ID is: BD00772, therefore, according to the requirement that "a < b represents the last digits in my ID", we have a = 2, b = 7

Exercise 1

a)

We have: $\overline{1a2} = 122$

We have: The prime factorization of 122 is:

$$122 = 2 \times 61$$

The bag of prime factor is:

$$A = \{|2 : 1, 61 : 1|\}$$

b)

We have: $\overline{2b0}$ = 270

We have: The prime factorization of 270 is:

$$270 = 2 \times 3^3 \times 5$$

The bag of prime factor is:

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

Exercise 2

We have:

$$A = \{|2 : 1, 61 : 1|\}$$

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

a) Find the cardinalities of each of the aforementioned bags

The cardinalities of A and B one:

$$|A| = 1 + 1 = 2$$

$$|B| = 1 + 3 + 1 = 5$$

b) Find the cardinalities of the intersection of the aforementioned bags

We have:

$$A = \{|2 : 1, 61: 1|\}$$

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

The intersection of sets A and B is:

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

The cardinalities of $A \cap B$ is:

$$|A \cap B| = 1$$

c) Find the cardinalities of the union of the aforementioned bags

We have:

$$A = \{|2 : 1, 61: 1|\}$$

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

The union of sets A and B is:

$$A \cup B = \{2: 1, 3: 3, 5: 1, 61: 1\}$$

The Cardinalities of $A \cup B$ is:

$$|A \cup B| = 1 + 3 + 1 + 1 = 6$$

d) Find the cardinalities of the difference of the aforementioned bags

We have:

$$A = \{|2 : 1, 61: 1|\}$$

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

The difference of the aforementioned bags.

The different of bags $A - B$ is:

$$A - B = \{|61: 1|\}$$

The different of bags $B - A$ is:

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

The cardinality of bags $A - B$ is:

$$|A - B| = 1$$

The cardinality of bags $B - A$ is:

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

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

1.3.1 Definition of a Function

Function

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. We write f(a) = b if b is the unique element of B assigned by the function f to the element a of A. If f is a function from A to B, we write f : A → B (Levin, 2025)

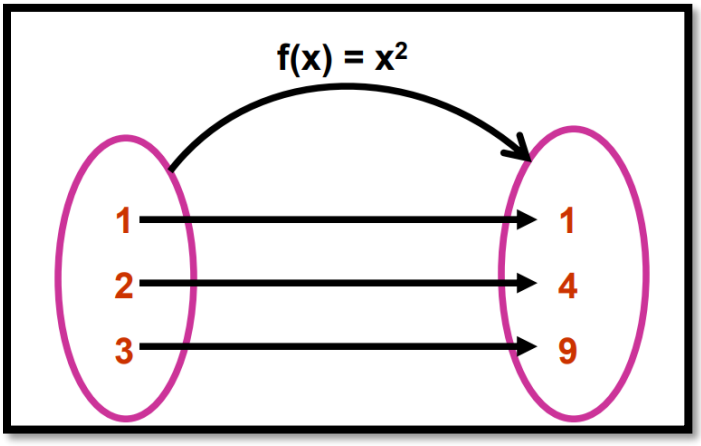

Figure 1 - 7: Relation and Functions

Domain & Range

If f is a function from A to B, we say that A is the domain of f and B is the codomain of f. If f(a) = b, we say that b is the image of a and a is a preimage of b. The range, or image, of f is the set of all images of elements of A. Also, if f is a function from A to B, we say that f maps A to B

The domain of a function $F:X \to Y$ is the set X.

The domain is defined as the set of all things that may be input to produce some output.

- When a function is considered a *total function* (the typical meaning of "function" unless specified otherwise), it is defined on every element of its domain.
- For a partial function, the input domain is X, but the *domain of the partial function* itself is the subset of X for which $F(x)$ is defined.

The range of a function is the set of all things that are actually output.

- The range is always a subset of the codomain.
- The range of F is also referred to as the image of F ($\mathrm{Im}(F)$).

1.3.2 Definition and Existence of Inverse Functions

Let f be a one-to-one correspondence from the set A to the set B. The inverse function of f, denoted by f^{-1}, is the function that assigns to an element b belonging to B the unique element a in A such that f(a) = b. Hence, f^{-1} (b) = a when f(a) = b.

The inverse relation f^{-1} is a function (specifically, a total function) if and only if the original function F is a correspondence.

A 1-1 correspondence is also known as a bijective function, or simply a bijection.

Theorem Summary: Let $f:X \to Y$ be a function.

- f^{-1} is a function from Y to X if and only if f is a One-to-One function correspondence (bijection).
- f^{-1} is a partial function from Y to X if and only if f is (injective).

Explanation of the Bijective Condition

A function $F:X \to Y$ must satisfy two independent properties simultaneously to be a bijection, ensuring that its inverse is also a function:

Injective (One-to-One function): A function F is injective if, for each $y \in Y$, there is at most one $x \in X$ such that $F(x)=y$.

The function is One-to-One function if $F(a)=F(b)$ implies $a=b$.

Why this is necessary: If a function were *not* injective (meaning two distinct inputs $x1$ and $x2$ map to the same output y), the inverse relation f^{-1} would contain the pairs $(y,x1)$ and $(y,x2)$. This violates the definition of a function, which requires that each input (in this case, y) maps to exactly one output. If a function is injective, its inverse satisfies the uniqueness condition (at most one output) required for a partial function.

Surjective (Onto): A function F is surjective (or onto) if, for each $y \in Y$, there is at least one $x \in X$ such that $F(x)=y$. This is equivalent to saying that the range of equals the codomain .

Why this is necessary: If the function were *not* surjective, there would be elements in the codomain Y (say y') that are not mapped to by any element in the domain X. The inverse relation f^{-1} would thus be undefined for y' (meaning y' has no corresponding output x). This violates the requirement that an inverse *function* (a total function) must be defined for every element of its domain (which is Y in the case of f^{-1}).

Conclusion: A function F is a bijection when it is both injective and surjective. This dual requirement ensures that every element in the codomain (Y) corresponds to exactly one element in the domain (X). When this condition holds, the inverse relation f^{-1} automatically satisfies the requirements of a total function from Y to X.

1.3.3 Finding the Inverse Algebraically

Steps for Finding the Inverse Algebraically

Step 1: Verify Invertibility Before proceeding, confirm that the function *F* is a 1−1 correspondence (a bijection), meaning it is both injective (1-1) and surjective (onto). If *F* is not a bijection, its inverse relation f^{-1} will not be a total function.

Step 2: Express the function using and Replace the function notation *F*(*x*) with the variable *y*: *y*=*F*(*x*)

Step 3: Swap the roles of and To find the inverse relation, swap the positions of *x* and *y* in the equation. This reflects the definition of the inverse relation f^{-1} containing pairs (*y*,*x*) where *f* contains pairs (*x*,*y*): *x*=*F*(*y*) *(This equation implicitly defines the inverse function)*.

Step 4: Solve the new equation for in terms of Use algebraic manipulation to isolate the variable *y* on one side of the equation. This isolates the output of the inverse function. This step leverages general

algebraic principles of solving equations for an unknown variable (as seen in mathematical examples used for computation, e.g.,).

Example: If the original function was $f(x)=2x+8$, we start with $y=2x+8$.

Swap: $x=2y+8$.

Solve for y: $2y=x-8$, leading to $y = \frac{x-8}{2}$

Thus the inverse function is :

$$f^{-1}(x) = \frac{x-8}{2}$$

Step 5: Write the result using inverse function notation Replace y with $F-1(x)$. $F-1(x)=$ (The resulting expression in terms of x)

The domain of the resulting inverse function $F-1$ is the range of the original function F (which is the codomain Y if F is surjective).

1.3.4 Applications of Inverse Functions in Computing

Encryption and Decryption (Cryptography)

The most direct application of inverse functions is found in ensuring data security through cryptographic systems.

Encoding/Decoding Schemes: To secure a transmission, the original message must be encoded or encrypted using a function (F) that transforms it into a non-recognizable form (cyphertext). The receiver must then use the inverse of the coding function () to decrypt the message and return it to its original form (plain text).

- Bijective Requirement: For a message to be uniquely recovered, the encoding scheme must use a function that is a bijection (a 1-1 correspondence) from the symbol set to itself. If the function were not injective or surjective, unique decryption would be impossible.
- Modular Arithmetic and RSA: Modern public key cryptosystems like RSA rely heavily on number theory and inverse operations in modular arithmetic.
- The encryption function (EA) and the decryption function (DA) used by the receiver are mutual inverses.

- The core difficulty that secures RSA is that the decrypting function, *DA*, is practically impossible to compute without specific "trapdoor" information (the secret key *dA*), even though the encryption function *EA* is easy to compute.

Data Mapping and Storage (Hashing)

In database systems and internal computer memory management, functions are used to map conceptual data to physical locations.

- Hashing Functions: A hashing function is a function that takes information (like an ATM card magnetic stripe value) as input and outputs a storage address.
- Injective Mapping: While real-world hashing functions deal with "collisions," an ideal or simple hashing function is often assumed to be 1-1 (injective) to ensure that distinct inputs map to distinct storage locations, thereby facilitating efficient retrieval.
- Domain and Image Size: The properties of functions, including injectivity and subjectivity, are used to relate the sizes of domains (input data) and codomains (potential memory addresses)

1.3.5 Activity 1 – Part 3

Requirements of 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 of Activity 1 – Part 3

My student ID is: BD00772, therefore, according to the requirement that “a < b represents the largest digits in my ID”, we have a = 2, b = 7

Exercise 1

a) We have:

$f = \mathbb{R} \to \mathbb{R}$ with $f(x) = bx + a = 7x + 2$

The function f has an inverse because it is a one-to-one correspondence. Suppose that y is the image of x, so that

$$f(x) = 7x + 2$$

$$y = 7x + 2$$

$$7x = y - 2$$

$$x = \frac{y-2}{7}$$

We conclude that the inverse function of the given function f is:

$$f^{-1}(y) = \frac{y-2}{7}$$

b) We have:

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

The function f has an inverse because it is a one-to-one correspondence Suppose that y is the image of x, so that:

$$f(x) = \sqrt{x+7}$$

$$y = \sqrt{x+7}$$

$$y^2 = x + 7$$

$$x = y^2 - 7$$

We conclude that the inverse function of the given function f is:

$$f^{-1}(x) = y^2 - 7$$

Exercise 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 $g \circ f.$

We have $f(x) = \begin{cases} 2x + a, x < 0 \\ x^3 + b, x \geq 0 \end{cases}$ And $g(x) = 7x - 2$

$$g \circ f(x) = g\big(f(x)\big) = \begin{cases} 7(2x+2) - 2, x < 0 \\ 7(x^3+7) - 2, x \geq 0 \end{cases}$$

$$g \circ f(x) = g\big(f(x)\big) = \begin{cases} 14x + 12, x < 0 \\ 7x^3 + 47, x \geq 0 \end{cases}$$

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

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

2.1.1 Definition (Binary Tree)

A binary tree is defined as either a tree with no vertices (the empty tree) or a rooted tree in which every vertex has a maximum of two children. This definition allows for the convenient concept of an empty tree, which is useful in many applications. (Rosen, 2011)

Key characteristics include:

Ordering: The children of a vertex in a binary tree are explicitly ordered by being designated as either the left child or the right child. Because of this ordering, two rooted trees can be isomorphic as general rooted trees but still represent different binary trees if the order of their children is switched.

Leaves: The leaves of a binary tree are conventionally considered to have empty subtrees for both the left and right children. This convention provides a straightforward way to determine when a leaf has been reached during traversal or search.

Inductive Definition: An Ordered Binary Tree (OBT) can be defined recursively (inductively): The empty tree () is an OBT, and if and are OBTs, then is an OBT with root , left subtree , and right subtree .

Binary Search Trees (BST):

- Definition: A BST is a rooted ordered binary tree where the nodes are labeled with elements (keys) from a totally ordered set.
- Property (Binary-Search-Tree Property): For every node *v*, the key of any vertex in its left subtree must be less than the key at *v*, and the key of any vertex in its right subtree must be greater than the key at *v* .
- Application: Used for searching an ordered set for a specific element, as the comparison at each node restricts the search to one of the two subtrees.
- Complete Binary Trees:
- Definition (Height-based): A binary tree is defined as complete if all its leaves occur at the lowest level.

- Definition (Inductive/OBT): A non-empty OBT *T* is complete if *T* is a single node (a root with two empty children), or if *T*=*u*(*T*1,*T*2) where both subtrees and are complete OBTs of *T*1 and are complete OBTof the same height.
- Properties: A complete OBT of height achieves the maximum possible number of nodes ($2^{h+1} - 1$)and maximum leaves (2^h) for that height. All non-leaf nodes have exactly two children.

Balanced Binary Trees (e.g., A-V-L Trees):

- Need: To ensure efficiency in BST operations, the height of the tree must be kept as small as possible; this is known as "balancing".
- A-V-L Trees: One technique for dynamic balancing involves rotations. An A-V-L tree performs a rebalancing action (such as a rotation) when a vertex's left subtree has a height two greater than its right subtree.
- Efficiency: If a BST is built from a list of randomly ordered elements, its height, on average, is *O*(log2(*n*)) , which is generally considered efficient.

Binary Heaps:

- Definition: An ordered binary tree that satisfies the heap property instead of the binary-search-tree property.
- Min-heap-property: The key of every descendant of a node is greater than the key of that node. The smallest key is at the root.
- Max-heap-property: The key of every descendant of a node is smaller than the key of that node. The largest key is at the root.
- Structure: A binary heap must be well-balanced, meaning its height is bounded logarithmically by the number of nodes *n* (specifically, at $[log2n]$ most is implied by the structural requirements).

2.1.2 Qualitative Modeling

Qualitative Analysis (Modeling): BSTs model the problem of efficiently locating a specific item within a dynamically changing, ordered data set. The underlying principle is one of division and elimination. When searching for an item, the process begins at the root and recursively decides which direction (left or right) to proceed based on a single comparison with the current node's key. The structure ensures that at

each step, roughly half of the remaining data is eliminated from consideration, making the search fast. (Gallier, Jean, and Jocelyn Quaintance)

Quantitative Analysis (Efficiency): The search algorithm's efficiency relies directly on the height of the tree.

Average Case: If the BST is reasonably balanced (e.g., built from randomly ordered elements), the height h is approximately O(log2n) , where is the number of elements. In this case, searching requires O(log2n)comparisons

Worst Case: If the tree becomes highly unbalanced (e.g., if elements are inserted in sorted order, creating a linear chain), the height can grow to n-1 , resembling a simple list traversal. In this scenario, the search requires O(n) comparisons, making it inefficient.

Qualitative modeling uses trees to represent structure, hierarchies, relationships, and logic. The goal is to clarify the organizational form or relational flow without necessarily focusing on numerical values or optimization.

Representing Hierarchical Structures: Rooted trees are an ideal model for hierarchical structures.

- File Systems: The file structure of a computer system can be represented by a rooted tree, with the root directory at the top.
- Genealogy: Family trees are typical examples of rooted trees, where ancestor and descendant relationships are clearly defined through the unique path from the root.

Modeling Expressions (Expression Trees): Expression trees are a visual representation for formal logical formulas or Boolean expressions.

- Logical Structure: The structure of the tree provides precise information about the order of operations (similar to parentheses). Manipulating expression trees can replace induction on formulas when writing formal proofs.
- Combinatorial Circuits: Trees and related logical representations (like normal forms) are used in designing combinatorial networks or circuits.

Modeling Decision Processes:

- Game Trees: Trees are used to model two-person terminating games of perfect information, such as Tic-Tac-Toe, where each node represents a board state or position.

Quantitative Analysis (Efficiency): The search algorithm's efficiency relies directly on the height of the tree.

Average Case: If the BST is reasonably balanced (e.g., built from randomly ordered elements), the height *h* is approximately O(log2n) , where is the number of elements. In this case, searching requires O(log2n) comparisons.

Worst Case: If the tree becomes highly unbalanced (e.g., if elements are inserted in sorted order, creating a linear chain), the height can grow to n-1 resembling a simple list traversal. In this scenario, the search requires O(n) comparisons, making it inefficient. (Haggard, 2006)

2.1.3 Quantitative Modeling

Qualitative Analysis (Modeling): A decision tree models the execution of any comparison-based algorithm, such as sorting, to determine the fundamental limit of its efficiency.

Structure: This rooted binary tree uses internal nodes to represent the comparison of two elements (a:b). The edges emanating from a node represent the two possible outcomes of the comparison (e.g., left edge if a < b , right edge if b < a).

Conclusion: Every leaf of the tree corresponds to a state where the relative ordering of all input elements has been completely determined, resulting in the sorted output.

Quantitative Analysis (Lower Bound): Since there are possible permutations (orderings) of input n elements, the decision tree must contain at least n! leaves.

Height and Comparisons: The minimum number of comparisons S(n) required by *any* such algorithm is equal to the minimum possible height of the decision tree. The level of any vertex represents the total number of comparisons performed to reach that state.

Mathematical Proof: Because a binary tree of height S(n) can have at most leaves, the following inequality must hold:

$$2^{S(n)} \geq n!$$

Solving this mathematically shows that $S(n) \geq log2(n!)$. Using approximation techniques,Log2(n!) is shown to be $O(nlog2n)$. This rigorously proves that $O(nlog2n)$ is a lower bound on the worst-case complexity for all sorting algorithms based on pairwise comparison.

2.1.4 Applications in Computing Context

Quantitative modeling uses trees to solve problems related to computation, counting, cost optimization, and algorithm performance analysis.

Optimization and Networks:

- Minimum Cost Spanning Tree (MCST): In a weighted graph, the Minimum Cost Spanning Tree is used to model the problem of connecting a number of specific locations most efficiently (e.g., computer networks or roads). The edge weights represent cost or distance.
- Historical Model: Gustav Kirchoff used trees in 1847 to model electrical networks, proving that all the information needed to solve for the currents lies in any spanning tree of the relevant graph.

Counting and Probability:

- Tree Diagrams: Trees are used to visualize and apply the Multiplication Principle in counting problems. Each leaf of the tree lists all possible outcomes of a multi-step process.
- Probability Trees: In probability theory, tree diagrams (also called probability trees or trees of possibilities) are essential graphical tools for modeling experiments involving sequential or dependent trials. By multiplying the probabilities along the path from the root to a leaf, one can calculate the probability of a specific outcome.

Algorithm Complexity Analysis:

- Decision Trees: These are rooted binary trees used to model comparison-based algorithms (like sorting). By analyzing the minimum height of this tree, one can find a lower bound on the complexity of that algorithm.
- Computation Trees: Trees are used to model the execution process of specific algorithms, such as Quicksort, helping to analyze performance and the number of comparisons needed based on pivot selection.

Data Structures: Trees like B-trees and B+ trees are used to index data in database systems, allowing for fast queries. Heaps (a type of binary tree) are used to implement priority queues.

Network Routing: The Spanning Tree Protocol (STP) is used in Ethernet networks to prevent broadcast storms by logically disabling loops, effectively creating a tree structure from the network graph.

Compilers: Abstract Syntax Trees (ASTs) are used by compilers to represent the structure of source code (e.g., the order of operations in a mathematical formula) before it's converted to machine code.

2.1.5 Example: Modeling Contextualized Problems

Tree Terminologies

Parent, Child, Sibling Ancestor, Descendant, Leaf, Internal Vertex, Subtree

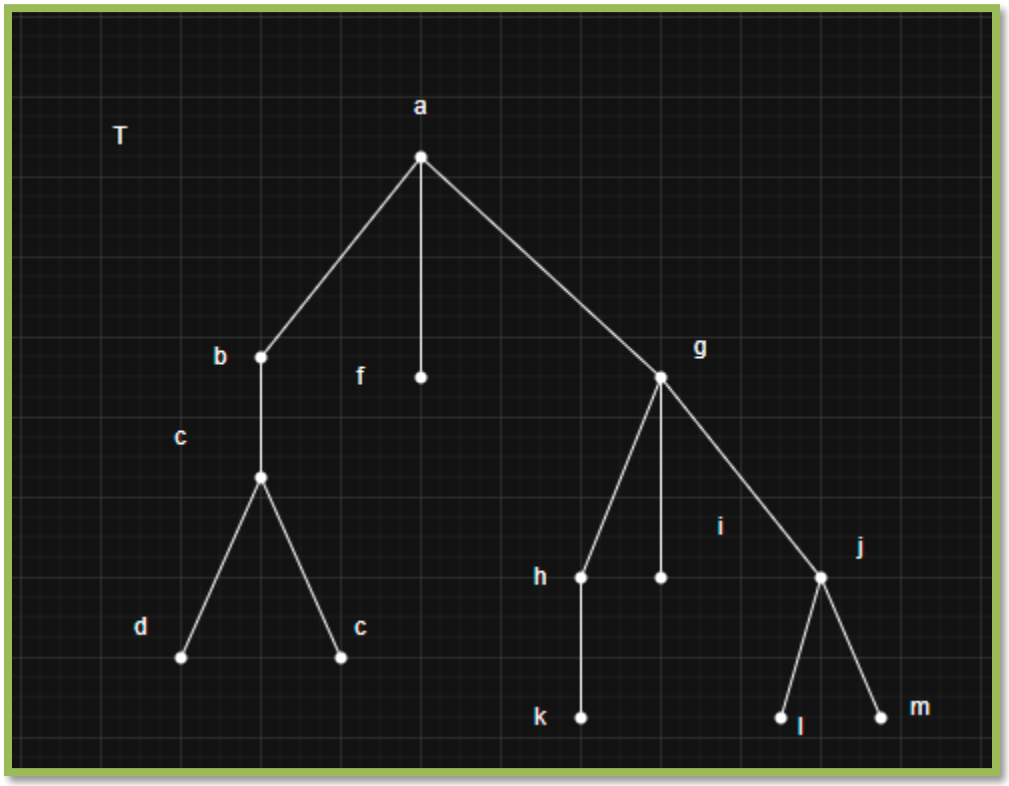

Figure 2 - 1: Tree Terminologies

Consider the rooted tree defined by the following edges (where the first vertex is the parent):

$a \rightarrow b, a \rightarrow f, a \rightarrow g$

$b \rightarrow c$

$c \rightarrow d, c \rightarrow e$

$g \rightarrow h, g \rightarrow i, g \rightarrow j$

$h \rightarrow k$

$j \rightarrow l, j \rightarrow m$

Using the definitions above, determine the following for the tree T:

- a) The parent of c.
- b) The children of g.
- c) The siblings of h.
- d) The ancestors of e.
- e) The descendants of b.
- f) The leaves of the tree.
- g) The internal vertices of the tree.

Definitions

- Parent: If v is a vertex in T other than the root, the parent of v is the unique vertex u such that there is a directed edge from u to v.
- Child: When u is the parent of v, v is called a child of u.
- Sibling: Vertices with the same parent are called siblings.
- **Ancestor:** The ancestors of a vertex (other than the root) are the vertices in the path from the root to this vertex, excluding the vertex itself and including the root.
- **Descendant:** The descendants of a vertex v are those vertices that have v as an ancestor.
- **Leaf:** A vertex of a rooted tree is called a leaf if it has no children.
- **Internal Vertex:** Vertices that have children are called internal vertices. (The root is an internal vertex unless it is the only vertex in the graph).
- **Subtree:** If a is a vertex in a tree, the subtree with a as its root is the subgraph of the tree consisting of a, its descendants, and all edges incident to these descendants.

Application Results Based on the tree structure described:

- a) The parent of c: The vertex c has an incoming edge from b. Therefore, the parent is b.
- b) The children of g: There are edges from g to h, i, and j. Therefore, the children are h, i, j.
- c) The siblings of h: The parent of h is g. The other children of g are i and j. Therefore, the siblings are i, j.

- d) The ancestors of e: The path from the root a to e is a → b →c → e. Excluding e itself, the ancestors are c, b, a.
- e) The descendants of b: The vertices that have b as an ancestor are c (child of b) and d, e (children of c). Therefore, the descendants are c, d, e.
- f) The leaves: The vertices with no outgoing edges (no children) are d, e, f, k, i, l, m.
- g) The internal vertices: The vertices that have at least one child are a, b, c, g, h, j.

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

2.2.1 Definition of Dijkstra's algorithm

In Dijkstra's Algorithm, the goal is to find the shortest distance from a given source node to all other nodes in the graph. As the source node is the starting point, its distance is initialized to zero. From there, we iteratively pick the unprocessed node with the minimum distance from the source, this is where a min-heap (priority queue) or a set is typically used for efficiency (Robinson, 1995)

- Shortest Paths: The distance between two vertices and is defined as the length of a shortest path joining to . If no path exists, the distance is commonly defined as infinity or left undefined.
- Optimal Paths: The context of finding an optimal path (like shortest path) between two vertices is a typical problem in graph algorithms, and the sources mention that a search method like Breadth First Search (BFS) is used for problems involving finding an optimal path between two vertices.
- Weighted Graphs: The idea of weighting edges to represent costs or values is fundamental to finding a shortest path spanning tree. A weighted graph is a graph G along with a function F : E -> R which assigns a weight, or cost, or value F to each edge . The goal in weighted graphs is often to find a minimum cost structure, such as a Minimum Cost Spanning Tree (MCST), using algorithms like Kruskal's or Prim's.

Applications in computing context of Dijkstra's algorithm

Communication Networks and Routing

Dijkstra's algorithm addresses the core need to find optimal routes in networks, which are frequently modeled as graphs:

- Network Structure: Computer scientists use graphs to model communication networks where vertices represent components such as computers, processors, or switches, and directed edges represent transmission lines or wires.
- Minimizing Delay (Latency): In communication networks, the delay experienced by a data packet is proportional to the length of the path it follows. Packets are generally routed along the shortest path possible. The maximum delay in a network, known as the diameter, is determined by the length of the shortest path between the farthest input and output.
- Weighted Graphs: The problem of finding a shortest path spanning tree is solved in the context of weighted graphs. In this model, edges are assigned a weight, cost, or value, representing metrics like cost or distance, making the algorithm suitable for finding structures of minimum cost.

General Graph Optimization

The algorithm is key to structural analysis of graphs in general computing contexts:

- Distance Calculation: The objective of finding a shortest path relates directly to calculating the distance between two vertices, defined as the length of the shortest path joining them.
- Optimal Path Finding: Finding an "optimal path between two vertices" is a typical algorithmic problem in graph analysis.
- Minimum Cost Structures: Closely related graph algorithms, such as Kruskal's and Prim's, are used to find a Minimum Cost Spanning Tree (MCST). While Dijkstra's focuses on shortest paths, these minimum cost problems share the foundational principle of optimizing a graph structure based on edge weights.

2.2.2 Activity 2 – Part 1

Requirements of Activity 2 – Part 1

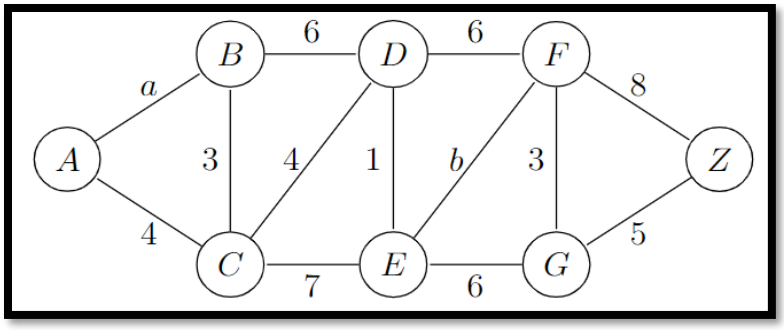

Figure 2 - 2: Activity 2 – Part 1

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

State Dijkstra's Algorithm in an undirected graph.

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

Solution of Activity 2 – Part 1

My student ID is: BD00772, therefore, according to the requirement that "a < b represents the largest digits in my ID", we have a = 2, b = 7

- We are finding the shortest path from vertex A to vertex Z.
- The distance to A is 0.
- The set of visited vertices is empty.

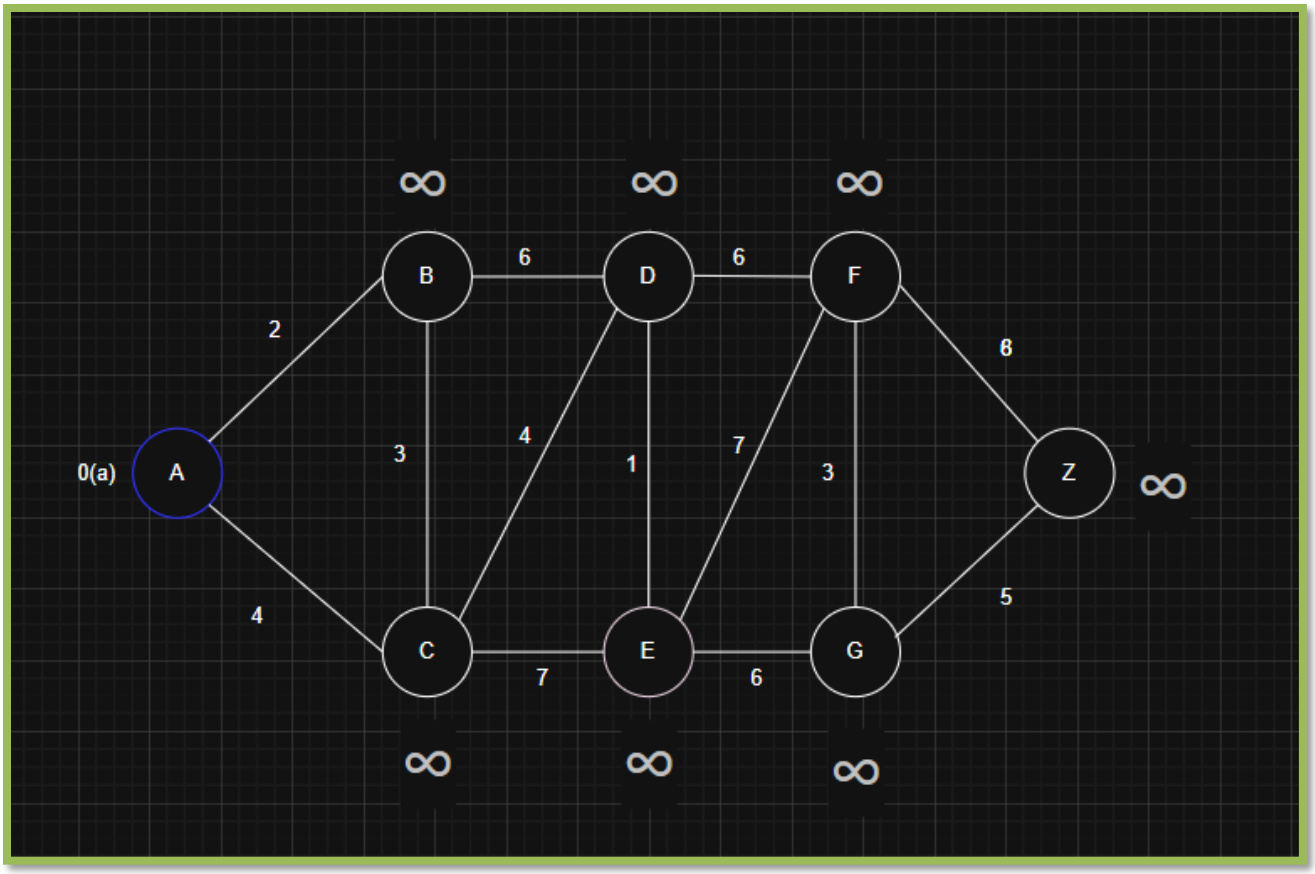

Figure 2 - 3: Activity 2 – Part 1

Step-by-Step Execution

Step 1:

- Select Vertex: A (distance 0).
- Add to Visited: {A}
- Update Neighbors:
  - Path to B: 0 + 2 = 2. (Update B's distance)
  - Path to C: 0 + 4 = 4. (Update C's distance)
- Current Shortest Distances: {A: 0, B: 2, C: 4}

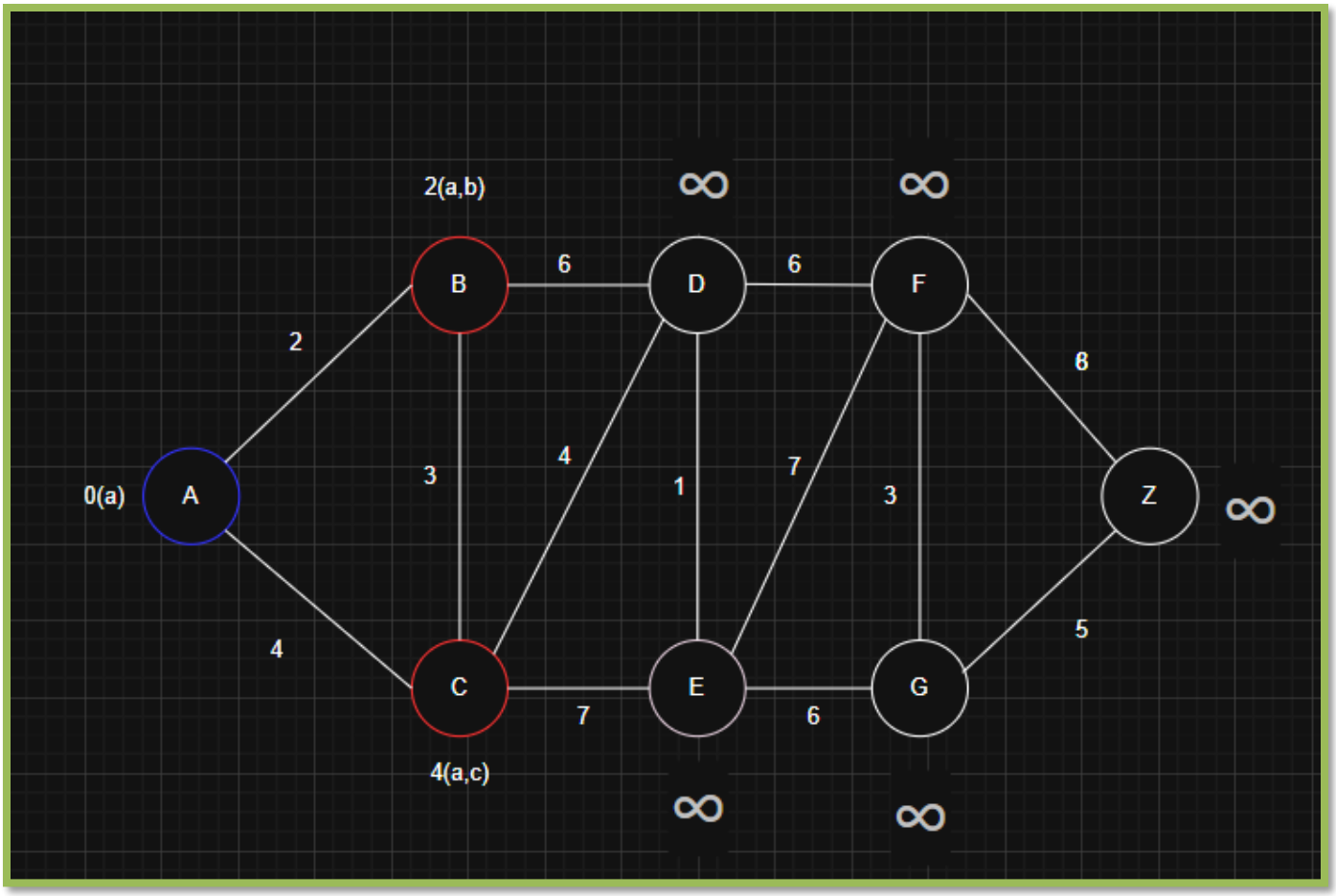

Figure 2 - 4: Activity 2 – Part 1

Step 2:

- Select Vertex: B (distance 2).
- Add to Visited: {A, B}
- Update Neighbors:
  - Path to C: 2 (to B) + 3 = 5. This is not less than C's current distance of 4. (No change)
  - Path to D: 2 (to B) + 6 = 8. (Update D's distance)
- Current Shortest Distances: {A: 0, B: 2, C: 4, D: 8}

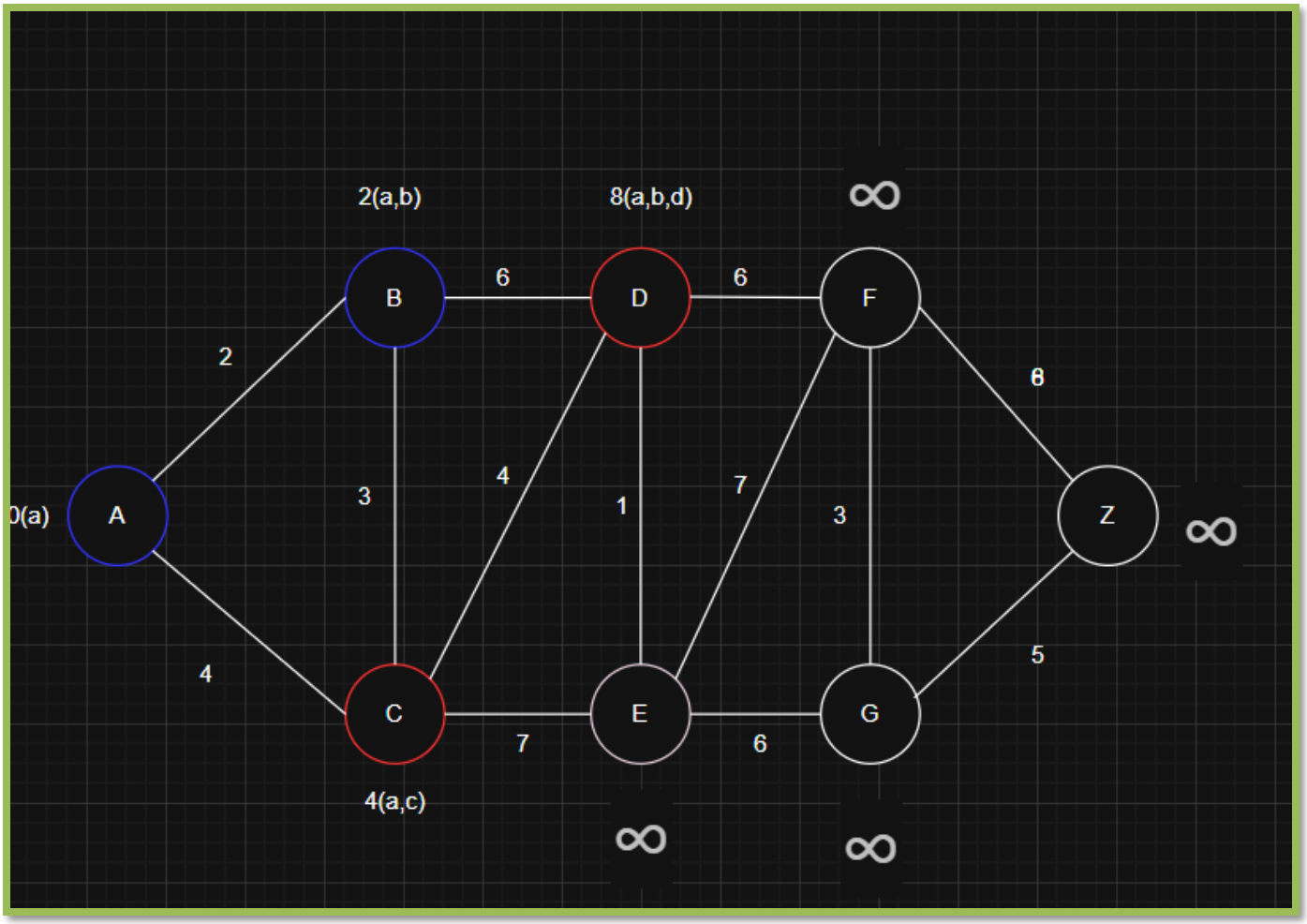

Figure 2 - 5: Activity 2 – Part 1

Step 3:

- Select Vertex: C (distance 4).
- Add to Visited: {A, B, C}
- Update Neighbors:
  - Path to D: 4 (to C) + 4 = 8. This is not less than D's current distance of 8. (No change)
  - Path to E: 4 (to C) + 7 = 11. (Update E's distance)
- Current Shortest Distances: {A: 0, B: 2, C: 4, D: 8, E: 11}

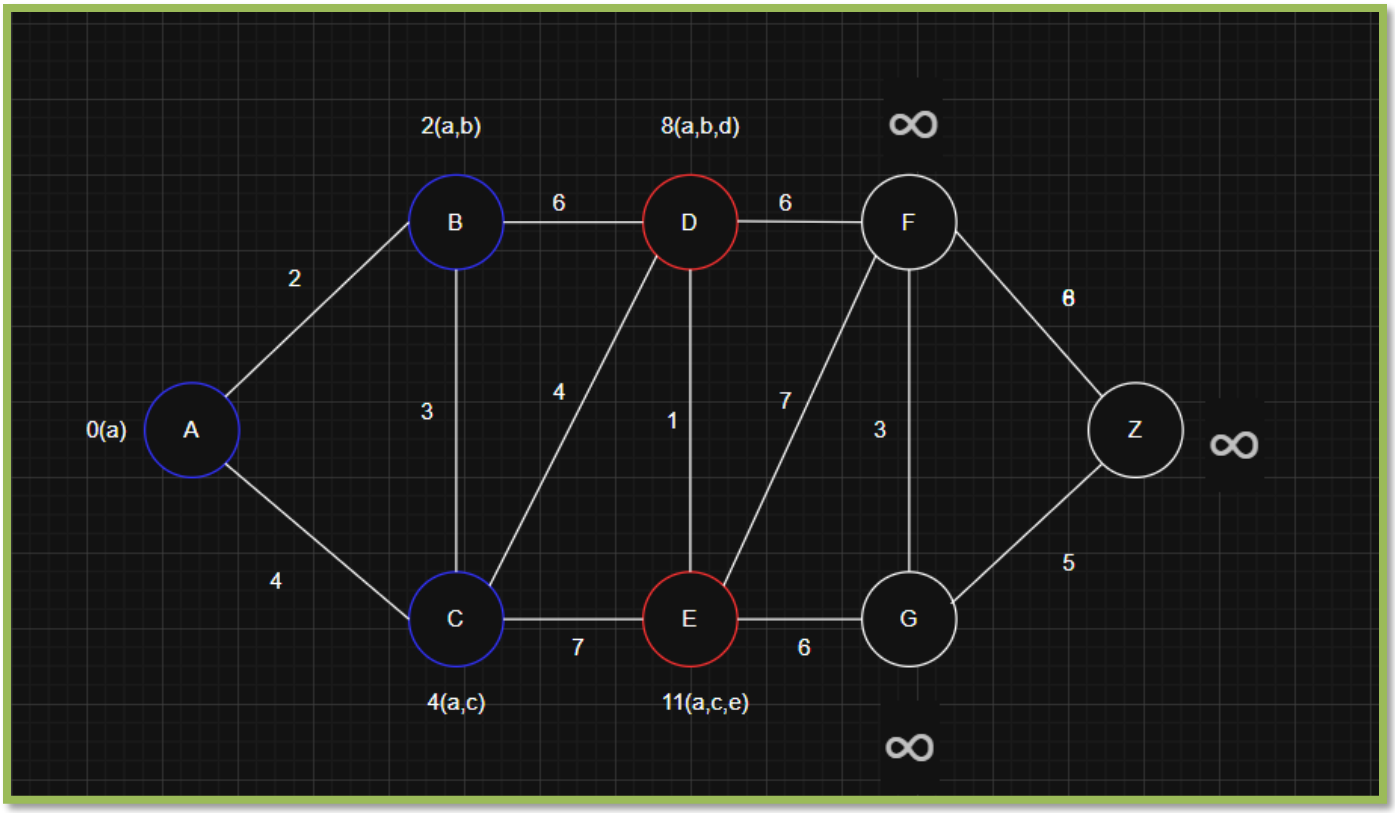

Figure 2 - 6: Activity 2 – Part 1

Step 4:

- Select Vertex: D (distance 8).
- Add to Visited: {A, B, C, D}
- Update Neighbors:
  - Path to E: 8 (to D) + 1 = 9. This is less than E's current distance of 11. (Update E's distance to 9)
  - Path to F: 8 (to D) + 6 = 14. (Update F's distance to 14)
- Current Shortest Distances: {A: 0, B: 2, C: 4, D: 8, E: 9, F: 14}

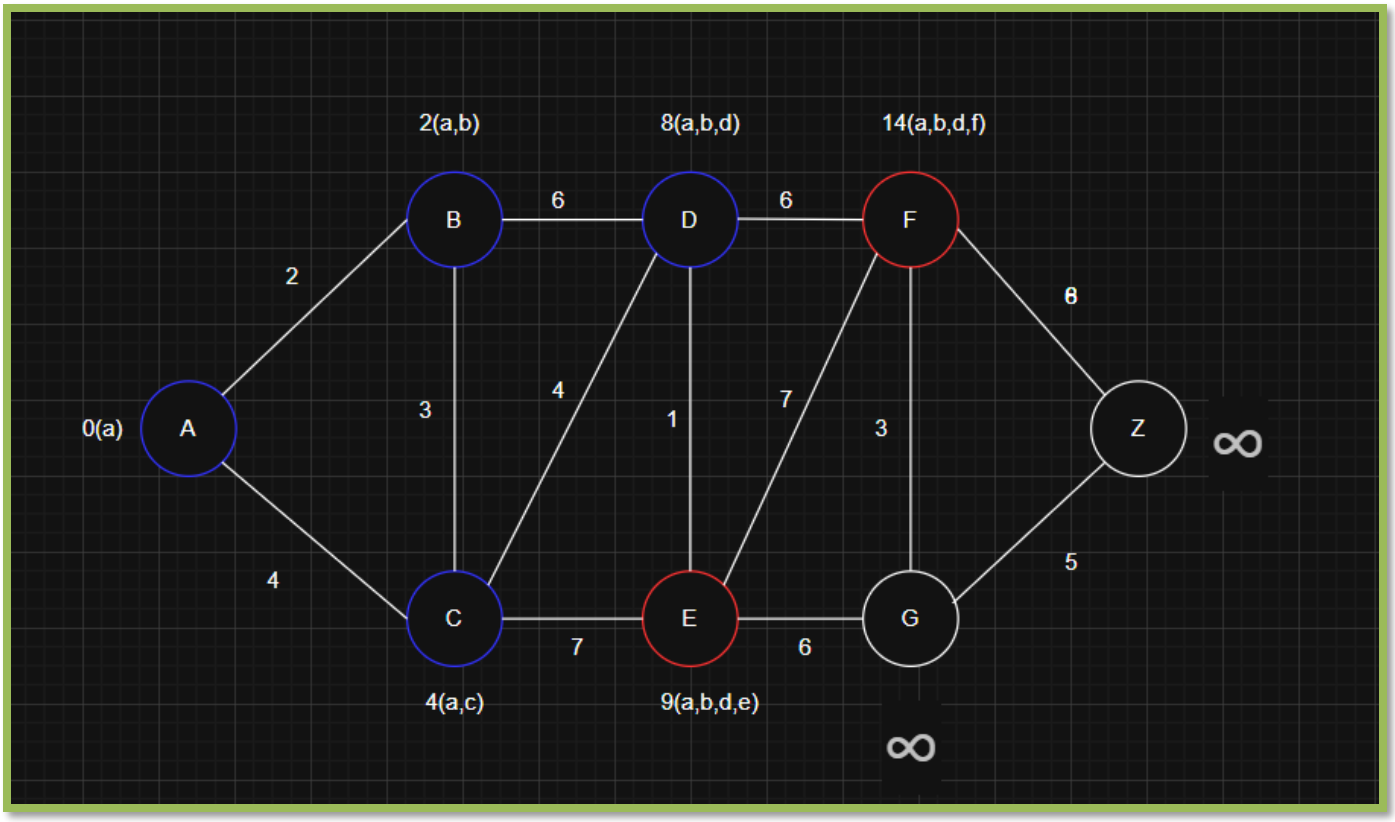

Figure 2 - 7: Activity 2 – Part 1

Step 5:

- Select Vertex: E (distance 9).
- Add to Visited: {A, B, C, D, E}
- Update Neighbors:
  - Path to F: 9 (to E) + 7 (your b value) = 16. This is not less than F's current distance of 14. (No change)
  - Path to G: 9 (to E) + 6 = 15. (Update G's distance to 15)
- Current Shortest Distances: {A: 0, B: 2, C: 4, D: 8, E: 9, F: 14, G: 15}

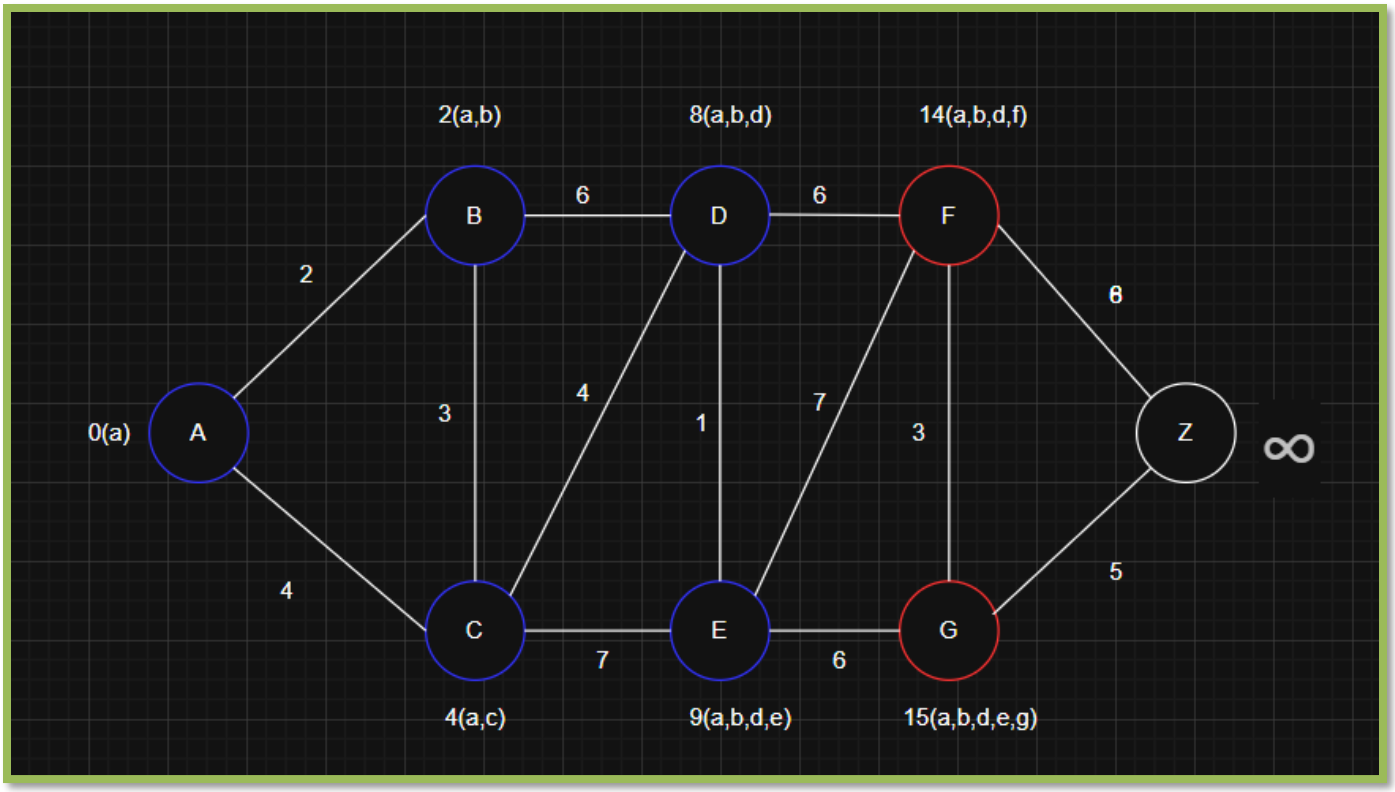

Figure 2 - 1: Activity 2 – Part 1

Step 6:

- Select Vertex: F (distance 14).
- Add to Visited: {A, B, C, D, E, F}
- Update Neighbors:
  - Path to G: 14 (to F) + 3 = 17. This is not less than G's current distance of 15. (No change)
  - Path to Z: 14 (to F) + 8 = 22. (Update Z's distance to 22)
- Current Shortest Distances: {A: 0, B: 2, C: 4, D: 8, E: 9, F: 14, G: 15, Z: 22}

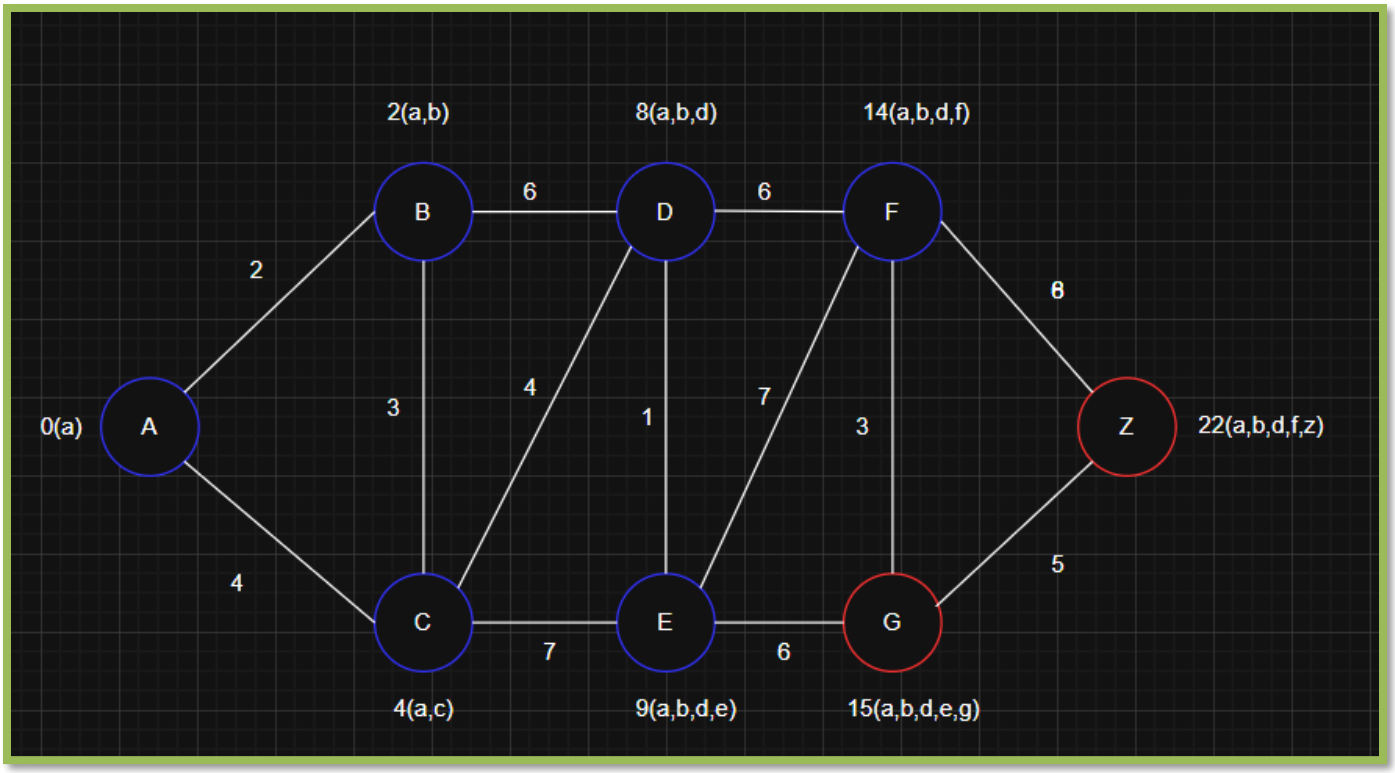

Figure 2 - 8: Activity 2 – Part 1

Step 7:

- Select Vertex: G (distance 15).
- Add to Visited: {A, B, C, D, E, F, G}
- Update Neighbors:
  - Path to Z: 15 (to G) + 5 = 20. This is less than Z's current distance of 22. (Update Z's distance to 20)
- Current Shortest Distances: {A: 0, B: 2, C: 4, D: 8, E: 9, F: 14, G: 15, Z: 20}

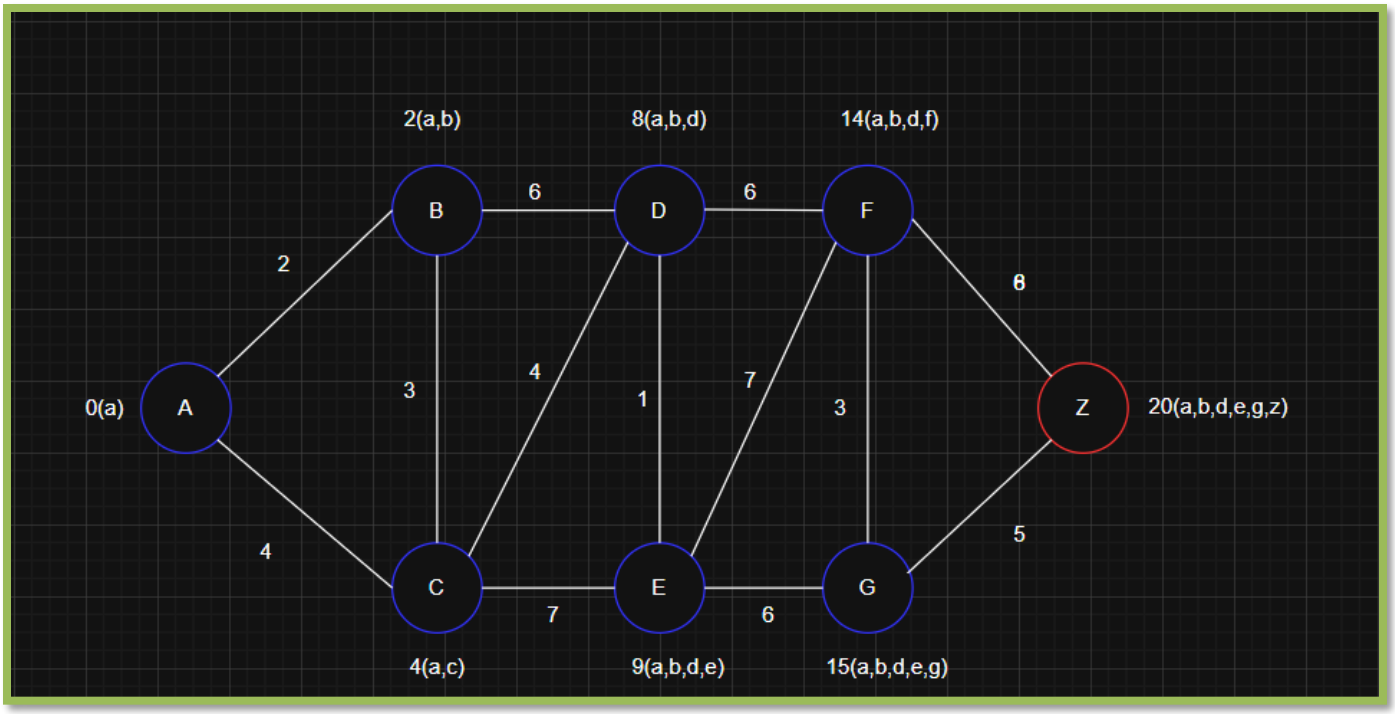

Figure 2 - 9: Activity 2 – Part 1

Step 8:

- Select Vertex: Z (distance 20).
- Add to Visited: {A, B, C, D, E, F, G, Z}
- The algorithm is complete.

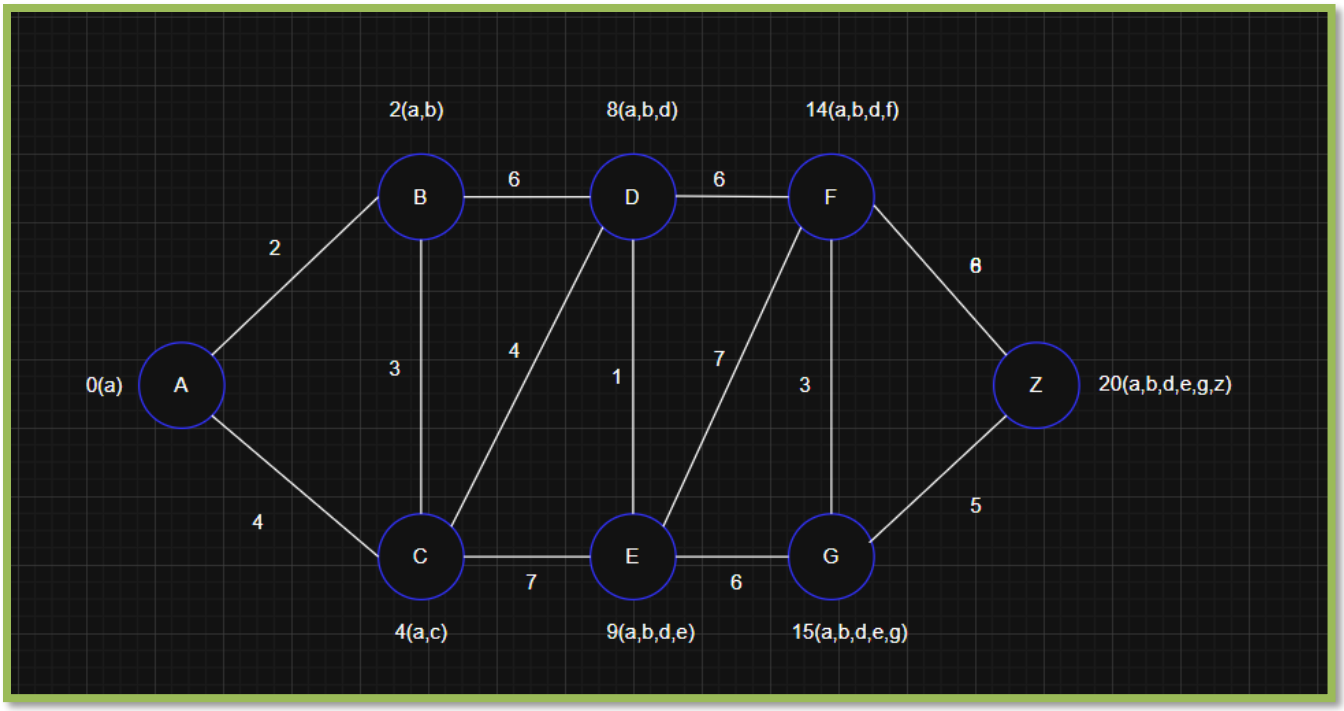

Figure 2 - 10: Activity 2 – Part 1

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

2.3.1 Eulerian Circuit & Path

Definition:

- Eulerian Circuit: A trail in a graph that traverses every edge exactly once and starts and ends at the same vertex.
- Eulerian Path: A trail that traverses every edge exactly once but starts and ends at different vertices.
- Condition: A connected graph has an Eulerian circuit if and only if every vertex has an even degree.

What Can It Do? (Applications)

Eulerian paths are primarily used for Coverage Problems where the goal is to pass through every connection or road efficiently.

- Route Optimization (The "Chinese Postman Problem"): It is used to plan optimal routes for services that need to traverse every street in a neighborhood, such as snow plows, street sweepers, garbage collection trucks, and mail delivery.
- DNA Sequencing: In bioinformatics, Eulerian paths are used in de Bruijn graphs to reconstruct long DNA sequences from short, overlapping fragments (reads). This is faster than Hamiltonian approaches for large genomes.
- Circuit Testing: It helps in designing test patterns for identifying faults in digital circuits by ensuring every connection (wire/transistor) is tested exactly once.

2.3.2 Hamiltonian Circuit & Path

Definition Used

- Hamiltonian Circuit: A loop in a graph that visits every vertex exactly once and returns to the start.
- Hamiltonian Path: A path that visits every vertex exactly once but does not necessarily return to the start.
- Condition: Unlike Eulerian graphs, there is no simple rule (like "even degrees") to quickly check for Hamiltonian paths. However, if a graph is Bipartite with an odd number of vertices, it cannot have a Hamiltonian circuit (as used in my previous proof).

What Can It Do? (Applications)

Hamiltonian paths are used for Ordering and Logistics Problems where the "destinations" matter more than the routes between them.

- Logistics & The Traveling Salesman Problem (TSP): This is the most famous application. It calculates the most efficient route for a delivery driver who needs to drop off packages at 10 specific locations (vertices) and return to the depot, visiting each location only once.
- Network Topology: In computer networks, Hamiltonian cycles are used to model "Token Ring" networks where data is passed sequentially from one computer to the next without skipping any node.
- Computer Graphics: It is used in "Triangle Stripification," a technique to render 3D meshes faster by ordering the vertices so the graphics card processes them in a single continuous stream rather than independent triangles.

2.3.3 Activity 2 -Part 3

Definition: An Eulerian circuit exists in a connected graph if and only if every vertex has an even degree.

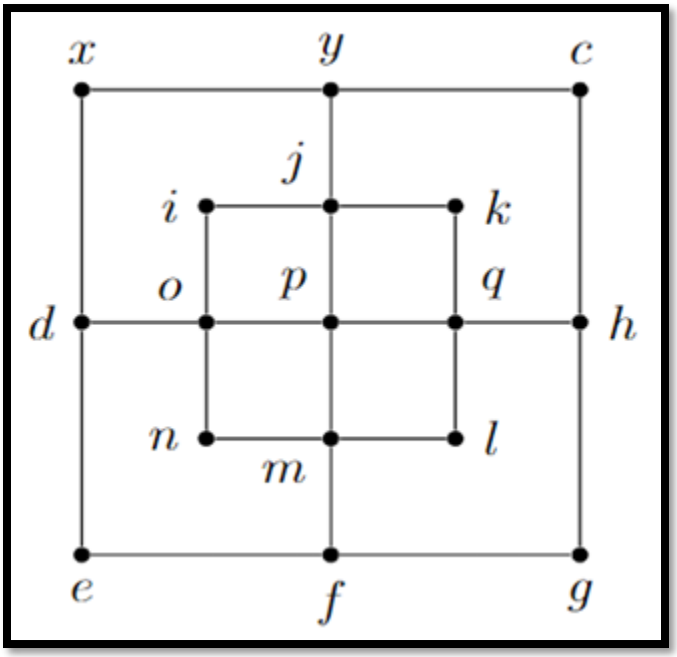

Figure 2 - 11: Grid graph for the Hamiltonian path problem

- Analysis of Degrees:
  - Let's look at the vertices on the "middle" of the outer sides: y, h, f, d.
  - Vertex y is connected to x, c, j. Degree = 3 (Odd).
  - Vertex h is connected to c, g, q. Degree = 3 (Odd).
  - Vertex f is connected to e, g, m. Degree = 3 (Odd).
  - Vertex d is connected to x, e, o. Degree = 3 (Odd).

Conclusion: Since there are vertices with odd degrees, no Eulerian Circuit exists. (Additionally, because there are more than 2 odd vertices, no Eulerian *Path* exists either).

2.3.4 Assessment of Hamiltonian Circuit

Definition: A Hamiltonian circuit is a closed loop that visits every vertex exactly once.

Bipartite Check:

- A graph is Bipartite if vertices can be divided into two disjoint sets (U and V) such that every edge connects a vertex in U to one in V.
- We can color this grid graph like a chessboard (alternating colors):
  - Set A (9 vertices): y, h, f, d, i, k, l, n, p
  - Set B (8 vertices): x, c, g, e, j, q, m, o
  - The Odd Vertex Count: The total number of vertices is 17 (9 + 8).
  - Theorem: A bipartite graph with an odd number of vertices cannot have a Hamiltonian Circuit. This is because any cycle in a bipartite graph must have an even length (alternating between Set A and Set B to return to the start).

Conclusion: The graph has 17 vertices (odd) and is bipartite; therefore, no Hamiltonian Circuit exists.

Does the graph have a Hamilton Path?

Answer: No, the graph does not have a Hamilton path.

Argument (Proof by Contradiction):

Step 1: Determine Start/End Constraints based on Bipartite Sets

We established the graph is bipartite with two sets:

- Set A (Size 9): y, h, f, d, i, k, l, n, p
- Set B (Size 8): x, c, g, e, j, q, m, o

For a Hamiltonian path to exist in a bipartite graph where the partition sizes differ by 1 (|A| = |B| + 1), the path must start and end at a vertex in the larger set (Set A).

Step 2: Analyze Structural Constraints (The "Isolation" Trap)

Let's look at the Outer Corners (x, c, g, e).

These four vertices belong to Set B.

Each has exactly degree 2. (e.g., x connects only to y and d).

If a vertex with degree 2 is *not* a start or end point of the path, the path must enter through one edge and exit through the other. This means both edges connected to that vertex must be used.

Hypothesis: Assume a Hamiltonian path exists. Based on Step 1, it must start and end in Set A. Therefore, no vertex in Set B can be an endpoint.

- This implies all Outer Corners (x, c, g, e) are internal nodes in the path.
- Consequently, we must use all edges connecting to these corners:
  - Edges (d, x) and (x, y) are used.
  - Edges (y, c) and (c, h) are used.
  - Edges (h, g) and (g, f) are used.
  - Edges (f, e) and (e, d) are used.

Step 3: Identify the Contradiction

If we use all the edges listed above to satisfy the flow through the corners, we form a closed cycle around the outer ring: x-y-c-h-g-f-e-d-x.

Now, look at the connecting vertices y, h, f, d (Set A):

- Vertex y has degree 3. Two of its edges (y-x and y-c) are now used by the outer ring path.
- This leaves y with no available edges to connect to the inner part of the graph (edge y-j cannot be used).
- The same applies to h, f, and d.

Result: The outer ring becomes isolated from the inner graph. A Hamiltonian path must visit *all* vertices, which is impossible if the graph is split into disconnected sections by the path logic.

Conclusion:

To prevent the outer ring from isolating itself, at least one Outer Corner (Set B) must be an endpoint. However, the Bipartite property dictates that endpoints must be in Set A. This is a contradiction. 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:

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).

Table 3 - 1: Symbol Meaning

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

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

Table 3 - 2: Trush Table For KBS

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:

$$Z = A \wedge (M \vee V)$$

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.

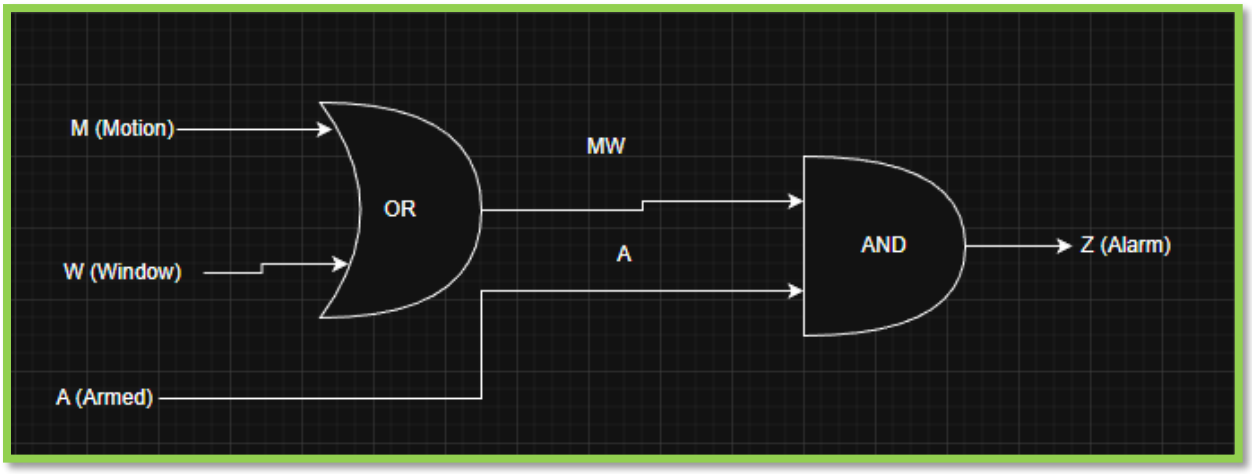

Figure 3 - 1: Logic Gate 1

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 & 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:

$$V = S \wedge \neg R$$

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.

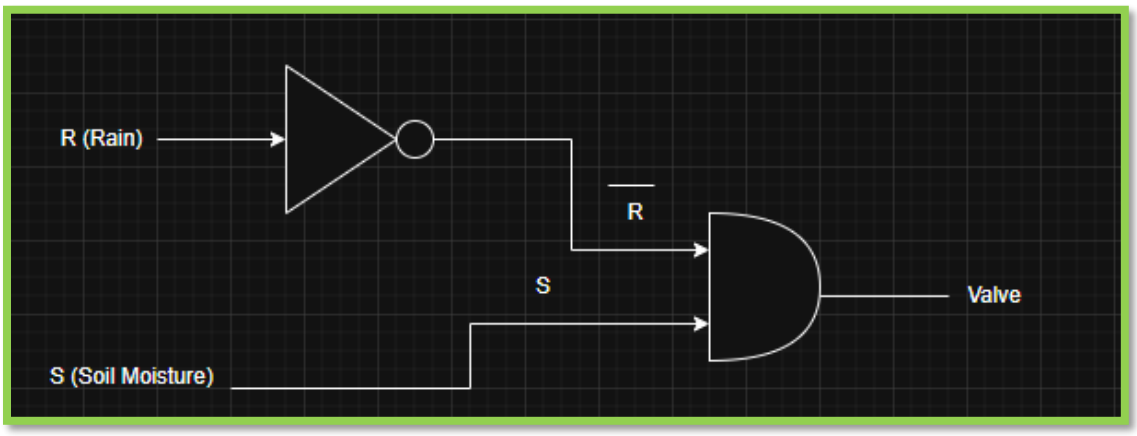

Figure 3 - 2: Logic Gate 2

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."

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

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: . $\overline{M}$

Boolean Equation

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

Truth Table

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

Row	Maintenance (M)	Card (C)	PIN (P)	OR Check (C+P)	NOT M ($\overline{M}$)	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

Table 3 - 3: Truth Table Of The Secure Facility Door

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

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).

Table 3 - 4: Truth Table Of Computer Login

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:

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

Table 3 - 5: Truth Table For Expression

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 & 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:

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:

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

$$x + y + z$$

Equation 3

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 S a 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:

$$a \circ e = a \; and \; e \circ a = a$$

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

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 +)

Table 4 - 1: Comparison of algebraic on the set of Integers Z

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

$$*: S \times S \to S$$

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

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

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:

$$| H \mid\mid G |$$

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 Gwithout overlap

- Therefore: | G |= $(number\ of\ cosets)$ ·| 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 Gmust satisfy.

$$| H \mid\mid G |$$

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:

$$G = \{e\}$$

$\circ$	e
e	e

Table 4 - 2: Group of Order 1 (Cayley)

Group order: $| G |= 1$

Every element has order 1

Group of Order 2

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

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

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

Table 4 - 3: Group of Order 2 (Cayley)

$$| G |= 2$$

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

Subgroups: $\{e\}$ and G

Group of Order 3

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

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

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

Table 4 - 4: Group of Order 3 (Cayley)

$$| G |= 3$$

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

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

$\circ$	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

Table 4 - 5: Cyclic group C_4

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

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

Table 4 - 6: Klein four-group V_4

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

∘	e
e	e

Table 4 - 7: Order 1

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., $a \circ a = e$).

∘	e	a
e	e	a
a	a	e

Table 4 - 8: Order 2

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

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

∘	e	a	b
e	e	a	b
a	a	b	e
b	b	e	a

Table 4 - 9: Order 3

- 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 $a^4 = e$.

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

Table 4 - 10: Cyclic Group Z_4

Case 2: Klein Four-Group V_4

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

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

Table 4 - 11: Klein Four-Group V_4

- 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 $S = \mathbb{R}\backslash\{-1\}$ 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 $e = 0$)

$$a + x - ax = 0$$
$$x(1 - a) = -a$$
$$x = -\frac{a}{1 - a} = \frac{a}{a - 1}$$

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

Summary of Main Content This report has comprehensively examined the fundamental concepts of discrete mathematics and their direct applications within software engineering. The study was structured around four key learning outcomes:

- Set Theory and Functions: Explored the algebraic operations of sets, the properties of multisets (bags), and the mechanics of inverse functions.
- Graph Theory: Analyzed mathematical structures using binary trees and utilized Dijkstra's algorithm to model pathfinding problems, while also assessing Eulerian and Hamiltonian circuits.
- Boolean Algebra: Investigated logic optimization by producing truth tables, diagramming binary problems, and simplifying Boolean equations using algebraic methods.
- Abstract Algebra: Explored the characteristics of binary operations and validated specific sets against the axioms required to form mathematical groups.

Final Results and Conclusion

The practical activities conducted in this assignment yielded specific, verified results based on the unique student identifier. Key findings include:

- Optimization:
- Logic Simplification:
- Structural Validation:

In conclusion, this report confirms that discrete mathematical structures provide the essential theoretical framework necessary for solving complex computational problems, ranging from network routing and circuit design to data organization and algorithmic analysis.

EVALUATION

Analysis of Solution Effectiveness The solutions provided are effective because they bridge theoretical definitions with practical application. For instance, the report does not just define Boolean algebra but applies it to concrete scenarios like "Smart Home Security" and "Agricultural Irrigation," demonstrating how abstract math translates to hardware logic. However, a limitation of the manual approach (as seen in the Cayley tables) is that it is only feasible for small datasets (Order 1–4); larger systems would require computational automation.

SWOT Analysis The following SWOT model provides a comprehensive evaluation of the methods and results presented in this assignment:

STRENGTHS (Internal)	WEAKNESSES (Internal)
Rigorous Methodology: The report strictly adheres to formal definitions (e.g., Rosen, Levin) ensuring theoretical soundness Detailed Process: The breakdown of Dijkstra's algorithm into 8 distinct steps allows for easy verification and error checking . Practical Relevance: The use of real-world examples (e.g., database hashing, logic circuits) strengthens the understanding of *why* these concepts matter.	Manual Computation: Solving complex graph problems or large Boolean equations by hand is time-consuming and prone to human error compared to algorithmic implementation. Abstract Complexity: Concepts like "Group Theory" and "Isomorphism" are highly abstract, making them difficult to visualize compared to the tangible nature of graph theory.
OPPORTUNITIES (External)	THREATS (External)
Software Application: The logic gates designed in Chapter 3 can be directly implemented in microcontroller programming (e.g., Arduino) for home automation projects .	Scalability Issues: While these methods work for small assignments, real-world networks (like the Internet) are too large for these manual spanning tree calculations without software aid.

Algorithm Optimization: Understanding the "Worst Case" scenarios in binary trees allows for better selection of sorting algorithms in future software development.	Strict Conditions: In Abstract Algebra, a single failing condition (like the lack of an inverse for one element) invalidates the entire structure, which can be a "pitfall" in system modeling.

REFERENCES

Attenborough, M., 2003. *Mathematics for Electrical Engineering and Computing.* 1st ed. Newnes.

Gallier, J. a. Q. J., 2025. *Mathematical Foundations And Aspects of Discrete Mathematics.* s.n.

Haggard, G. S. J. a. W. S., 2006. *Discrete Mathematics for Computer Science..* s.n.

Levin, O., 2025. *Discrete Mathematics: An Open Introduction..* s.n.

Meyer, A., 2010. *Mathematics for Computer Science..* s.n.

Piff, M., 1991. *Discrete Maths Software Engineers: An Introduction for Software Engineers.* 1st ed. Cambridge University Press.

Robinson, D., 1995. *A Course in the Theory of Groups.* 2nd ed. Springer.

Rosen, K. H., 2011. *Discrete Mathematics and Its Application.* 7th ed. McGraw-Hill.