Discrete Structures 2 [1]

A Study Guide for Students of Sorsogon State University - Bulan Campus[2]

Jarrian Vince G. Gojar[3]

April 27, 2025

[1]A course in the Bachelor of Science in Computer Science
[2]This book is a study guide for students of Sorsogon State University - Bulan Campus taking up the course Discrete Structures 2.
[3]https://github.com/godkingjay

Sorsogon State University - Bulan Campus

Contents

List of Figures

List of Tables

Preface

"If people do not believe that mathematics is simple, it is only because they do not realize how complicated life is."

– John von Neumann

Jarrian Vince G. Gojar
https://github.com/godkingjay

1

Boolean Algebra

1.1 Introduction

Circuits in computers are made up of millions of tiny switches that can be in one of two states: on or off. These switches are controlled by electrical signals that represent logical values. The behavior of these switches can be described using a mathematical system called Boolean Algebra. **Boolean Algebra** is a branch of mathematics that deals with logical values and operations on these values. It is widely used in computer science and engineering to design and analyze digital circuits. Computers uses the binary number system, which has only two digits: 0 and 1 which means "low voltage" and "high volt" respectively. These digits correspond to the logical values FALSE and TRUE, respectively.

1.2 History of Boolean Algebra

George Boole was an English mathematician and logician who lived in the 19th century. He was born in 1815 and died in 1864. Boole is best known for his work in the field of logic, which laid the foundation for modern computer science.

Boole's most famous work is his book *The Laws of Thought*, which was published in 1854. In this book, Boole introduced the concept of Boolean Algebra, which is a mathematical system for dealing with logical values. Boolean Algebra is based on the idea that logical values can be represented as either TRUE or FALSE.

In 1938, **Claude Shannon** showed that the two-valued Boolean Algebra could be used to describe the operation of electrical switches. This discovery laid the foundation for the design of digital circuits and computers.

1.3 Fundamental Operations

The three fundamental operations of Boolean Algebra are:

- **AND** - The AND operation takes two or more inputs and produces a 1 output only if all inputs are 1.
- **OR** - The OR operation takes two or more inputs and produces a 1 output if at least one input is 1.
- **NOT** - The NOT operation takes a single input and produces the opposite value. If the input is 1, the output is 0, and vice versa.

	Formal Logic	**Set Theory**	**Boolean Algebra**
Variables	p, q, r, ...	A, B, C, ...	x, y, z, ...
Operations	$\land$, $\lor$, $\neg$	$\cap$, $\cup$, $-$	$\bullet$, +, '
Special Elements	F, T	$\emptyset$, U	0, 1

Table 1.1: Comparison of Formal Logic, Set Theory, and Boolean Algebra

Table 1.1 shows a comparison of the notation used in formal logic, set theory, and Boolean Algebra. In formal logic, variables are represented by letters such as p, q, r, etc., and the logical operations are represented by symbols such as $\land$, $\lor$, and $\neg$. In set theory, variables are represented by capital letters such as A, B, C, etc., and the set operations are represented by symbols such as $\cap$, $\cup$, and $-$. In Boolean Algebra, variables are represented by letters such as x, y, z, etc., and the Boolean operations are represented by symbols such as $\cdot$, $+$, and $'$. The special elements in each system are F and T in formal logic, $\emptyset$ and U in set theory, and 0 and 1 in Boolean Algebra.

Though the notation used in each system is different, the underlying concepts are the same. For example, the AND operation in Boolean Algebra is similar to the logical conjunction operation in formal logic, where the output is TRUE only if all inputs are TRUE. Similarly, the OR operation in Boolean Algebra is similar to the logical disjunction operation in formal logic, where the output is TRUE if at least one input is TRUE. Compared to set theory, the AND operation in Boolean Algebra is similar to the intersection operation, where the output is the set of elements that are common to all input sets.

1.3.1 AND Operation

The AND operation is denoted by the symbol $\cdot$ or juxtaposition. The output of the AND operation is 1 only if all inputs are 1. In Boolean Algebra, the AND operation is represented by the multiplication symbol $\cdot$ or by juxtaposition. **Juxtaposition** is the act of placing two or more things side by side or close together.

Input 1	**Input 2**	**Output**
0	0	0
0	1	0
1	0	0
1	1	1

Table 1.2: Truth Table for the AND Operation

Table 1.2 shows the truth table for the AND operation. The output is 1 only if both inputs are 1; otherwise, the output is 0.

Suppose we have the variables x and y, and we want to represent the AND operation between them. We can write this as $x \cdot y$ or xy via juxtaposition. The output of this operation is 1 only if both x and y are 1.

Input 1	**Input 2**	**Output**
x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

Table 1.3: Truth Table for the AND Operation with Variables

Table 1.3 shows the truth table for the AND operation with variables x and y. The output is 1 only if both x and y are 1; otherwise, the output is 0.

Exercise

1. Consider the three input AND operation xyz. Write the truth table for this operation and determine the output for each combination of inputs.

1.3.2 OR Operation

The OR operation is denoted by the symbol +. The output of the OR operation is 1 if at least one input is 1. In Boolean Algebra, the OR operation is represented by the addition symbol +.

Input 1	**Input 2**	**Output**
0	0	0
0	1	1
1	0	1
1	1	1

Table 1.4: Truth Table for the OR Operation

Table 1.4 shows the truth table for the OR operation. The output is 1 if at least one input is 1; otherwise, the output is 0.

Suppose we have the variables x and y, and we want to represent the OR operation between them. We can write this as $x + y$. The output of this operation is 1 if at least one of x and y is 1.

Input 1	**Input 2**	**Output**
x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

Table 1.5: Truth Table for the OR Operation with Variables

Table 1.5 shows the truth table for the OR operation with variables x and y. The output is 1 if at least one of x and y is 1; otherwise, the output is 0.

Exercise
Write the truth table for the following operations:

1. $f(x,y,z) = x + y + z$
2. $f(x,y,z) = (x + y)z$
3. $f(x,y,z) = x + yz$
4. $f(x,y,z) = (x + y)(y + z)$
5. $f(x,y,z) = x(xy + yz)$

1.3.3 NOT Operation

The NOT operation is denoted by the symbol $'$. The output of the NOT operation is the opposite of the input. If the input is 1, the output is 0, and vice versa. In Boolean Algebra, the NOT operation is represented by the prime symbol $'$ or by an overline.

Input	**Output**
0	1
1	0

Table 1.6: Truth Table for the NOT Operation

Table 1.6 shows the truth table for the NOT operation. The output is the opposite of the input. If the input is 1, the output is 0, and vice versa.

Suppose we have the variable x, and we want to represent the NOT operation on it. We can write this as x' or $\overline{x}$. The output of this operation is the opposite or complement of x.

Input	**Output**
x	x'
0	1
1	0

Table 1.7: Truth Table for the NOT Operation with Variables

Table 1.7 shows the truth table for the NOT operation with variable x. The output is the opposite of x. If x is 1, the output is 0; if x is 0, the output is 1.

Exercise
Write the truth table for the following operations:

1. $f(x) = (x')'$
2. $f(x,y) = (x + y)'$
3. $f(x,y) = (x \cdot y)'$
4. $f(x,y,z) = (x + yz)'$
5. $f(x,y,z) = (x \cdot y + z)'$
6. $f(x,y,z) = (x + y)'z$
7. $f(x,y,z) = x' + y' + z'$
8. $f(x,y,z) = (x + y + z)'$
9. $f(x,y,z) = x'yz$
10. $f(x,y) = x' + y'(xy + x')$

1.4 Other Operations

In addition to the fundamental operations of AND, OR, and NOT, there are several other operations in Boolean Algebra that are commonly used. These operations include:

- **XOR** - The XOR operation takes two inputs and produces a 1 output if the inputs are different.
- **NAND** - The NAND operation is the complement of the AND operation. The output of the NAND operation is 0 only if all inputs are 1.
- **NOR** - The NOR operation is the complement of the OR operation. The output of the NOR operation is 0 if at least one input is 1.
- **XNOR** - The XNOR operation is the complement of the XOR operation. The output of the XNOR operation is 1 if the inputs are the same.

1.4.1 XOR Operation

The XOR operation is denoted by the symbol $\oplus$. The output of the XOR operation is 1 if the inputs are different. In Boolean Algebra, the XOR operation is represented by the symbol $\oplus$. If the XOR operation takes more than two inputs, it is called the **parity function**. A parity function is a function that determines whether the number of inputs that are 1 is even or odd.

Input 1	**Input 2**	**Output**
0	0	0
0	1	1
1	0	1
1	1	0

Table 1.8: Truth Table for the XOR Operation

Table 1.8 shows the truth table for the XOR operation. The output is 1 if the inputs are different; otherwise, the output is 0.

Suppose we have the variables x and y, and we want to represent the XOR operation between them. We can write this as $x \oplus y$. The output of this operation is 1 if x and y are different.

Input 1	**Input 2**	**Output**
x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Table 1.9: Truth Table for the XOR Operation with Variables

Table 1.9 shows the truth table for the XOR operation with variables x and y. The output is 1 if x and y are different; otherwise, the output is 0.

Exercise
Write the truth table for the following operations:
1. $f(x, y, z) = (x \oplus y)z$
2. $f(x, y, z) = x \oplus yz$
3. $f(x, y, z) = (x \oplus y)(y \oplus z)(z \oplus x)$
4. $f(x, y, z) = x' \oplus y \oplus z'$

1.4.2 NAND Operation

The NAND operation is simply the complement of the AND operation. The output of the NAND operation is 0 only if all inputs are 1. In Boolean Algebra, the NAND operation is represented by putting an overline or $'$ over the AND operation.

Input 1	**Input 2**	**Output**
x	y	$(xy)'$
0	0	1
0	1	1
1	0	1
1	1	0

Table 1.10: Truth Table for the NAND Operation with Variables

Table 1.10 shows the truth table for the NAND operation with variables x and y. The output is 0 only if both x and y are 1; otherwise, the output is 1.

1.4.3 NOR Operation

The NOR operation is simply the complement of the OR operation. The output of the NOR operation is 0 if at least one input is 1. In Boolean Algebra, the NOR operation is represented by putting an overline or $'$ over the OR operation.

Input 1	**Input 2**	**Output**
x	y	$(x+y)'$
0	0	1
0	1	0
1	0	0
1	1	0

Table 1.11: Truth Table for the NOR Operation with Variables

Table 1.11 shows the truth table for the NOR operation with variables x and y. The output is 0 if at least one of x and y is 1; otherwise, the output is 1.

1.4.4 XNOR Operation

The XNOR operation is simply the complement of the XOR operation. The output of the XNOR operation is 1 if the inputs are the same. In Boolean Algebra, the XNOR operation is represented by putting an overline or $'$ over the XOR operation.

Input 1	**Input 2**	**Output**
x	y	$(x \oplus y)'$
0	0	1
0	1	0
1	0	0
1	1	1

Table 1.12: Truth Table for the XNOR Operation with Variables

Table 1.12 shows the truth table for the XNOR operation with variables x and y. The output is 1 if x and y are the same; otherwise, the output is 0.

1.5 Tautology and Fallacy

In Boolean Algebra, a **tautology** is a statement that is always TRUE, regardless of the values of its variables. A **fallacy** is a statement that is always FALSE, regardless of the values of its variables.

x	x'	$x+x'$	xx'
0	1	1	0
1	0	1	0

Table 1.13: Examples of Tautologies and Fallacies

Table 1.13 shows examples of tautologies and fallacies. The expression $x + x'$ is a tautology because it is always TRUE, regardless of the value of x. The expression xx' is a fallacy because it is always FALSE, regardless of the value of x.

1.6 Boolean Functions

A **Boolean function** is a function that takes one or more Boolean variables as input and produces a Boolean output. Boolean functions are used to represent logical operations in Boolean Algebra. The output of a Boolean function is determined by the values of its input variables and the logical operations applied to them. It is also known as a **switching function**. Boolean variables are variables that can take on one of two values: 0 or 1. In Boolean Algebra, variables are typically denoted by letters such as x, y, z, etc. The values of these variables represent logical values: 0 corresponds to FALSE, and 1 corresponds to TRUE.

A Boolean function is a function in the form $f : B^n \to B$, where $B = \{0, 1\}$ is the set of Boolean values, and n is the number of input variables and is called the **arity** of the function.

A **literal** is a variable or its complement. For example, x and x' are literals. In the boolean function $f = (x + yz) + x'$, there are three variables: x, y, and z. The literals in this function are x, y, z, and x' which are the variables and their complements.

Exercises
Write the truth table for the following Boolean functions:

1. $f(x, y) = [xy + (x + y)']'$
2. $f(x, y) = (x + y) \oplus (xy)'$
3. $f(x, y, z) = x(y + z')$
4. $f(x, y, z) = (x + y)(y + z)(z + x)$
5. $f(x, y, z) = x \oplus y \oplus z$

1.7 Laws of Boolean Algebra

The laws of Boolean Algebra are a set of rules that define the properties of logical operations in Boolean Algebra. These laws are used to simplify Boolean expressions and to prove the equivalence of different expressions. The laws of Boolean Algebra are based on the properties of logical operations such as AND, OR, and NOT.

Boolean algebra's AND operations associates to multiplication in arithmetic, while OR operations associates to addition. While the NOT operation is simply the complement of the input.

Laws	**AND**	**OR**
Identity	$x(1) = x$	$x + 0 = x$
Domination	$x(0) = 0$	$x + 1 = 1$
Commutative	$xy = yx$	$x + y = y + x$
Associative	$x(yz) = (xy)z$	$x + (y + z) = (x + y) + z$
Distributive	$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$
Inverse	$xx' = 0$	$x + x' = 1$
Idempotent	$xx = x$	$x + x = x$
Absorption	$x(x + y) = x$	$x + xy = x$
	$x(x' + y) = xy$	$x + x'y = x + y$
De Morgan's Theorem	$(xy)' = x' + y'$	$(x + y)' = x'y'$
Consensus	$xy + x'z + yz = xy + x'z$	$(x + y)(x' + z)(y + z) = (x + y)(x' + z)$
Involution (Double Negation)	$(x')' = x$	

Table 1.14: Laws of Boolean Algebra

Table 1.14 shows the laws of Boolean Algebra. These laws are used to simplify Boolean expressions and to prove the equivalence of different expressions. The laws are based on the properties of logical operations such as AND, OR, and NOT.

Exercises

1. Verify the following laws of Boolean Algebra:
   - (a) Identity Law: $x + 0 = x$
   - (b) Commutative Law: $xy = yx$
   - (c) Associative Law: $x(yz) = (xy)z$
   - (d) Distributive Law: $x(y + z) = xy + xz$
   - (e) Inverse Law: $xx' = 0$
   - (f) Involution Law: $(x')' = x$
   - (g) De Morgan's Theorem: $(xy)' = x' + y'$
2. Prove the following laws of Boolean Algebra:
   - (a) Domination Law: $x + 1 = 1$
   - (b) Idempotent Law: $x + x = x$
   - (c) Absorption Law: $x(x + y) = x$

1.8 Simplifying Boolean Expressions

Boolean expressions can be simplified using the laws of Boolean Algebra. Simplifying a Boolean expression involves applying the laws of Boolean Algebra to reduce the expression to its simplest form. This process involves combining terms, eliminating redundant terms, and applying the laws of Boolean Algebra to simplify the expression.

Consider the Boolean expression $f(x, y) = x + x'y$. We can simplify this expression using the laws of Boolean Algebra as follows:

$$\begin{aligned}
f(x,y) &= x + x'y \\
&= (x+x')(x+y) && \text{Distributive Law} \\
&= (1)(x+y) && \text{Inverse Law} \\
&= x+y && \text{Identity Law} f(x,y) && = x + x'y = x + y
\end{aligned}$$

The expression $f(x,y) = x + x'y$ can be simplified to $f(x,y) = x + y$ using the laws of Boolean Algebra.

Consider the Boolean expression $f(x,y,z) = (x + yz') + (xy)'$. We can simplify this expression using the laws of Boolean Algebra as follows:

$$\begin{aligned}
f(x,y,z) &= (x + yz') + (xy)' \\
&= (x + yz') + x' + y' && \text{De Morgan's Theorem} \\
&= (x + x') + yz' + y' && \text{Commutative Law} \\
&= 1 + yz' + y' && \text{Inverse Law} \\
&= 1 && \text{Domination Law}
\end{aligned}$$

The expression $f(x,y,z) = (x + yz') + (xy)'$ can be simplified to $f(x,y,z) = 1$ using the laws of Boolean Algebra.

Consider the Boolean expression $f(x,y) = (x + x'y) + (x + y')$. We can simplify this expression using the laws of Boolean Algebra as follows:

$$\begin{aligned}
f(x,y) &= (x + x'y) + (x + y') \\
&= [(x + x')(x + y)] + (x + y') && \text{Distributive Law} \\
&= [1(x + y)] + (x + y') && \text{Inverse Law} \\
&= (x + y) + (x + y') && \text{Identity Law} \\
&= (y + x) + (x + y') && \text{Commutative Law} \\
&= y + (x + x) + y' && \text{Associative Law} \\
&= y + x + y' && \text{Idempotent Law} \\
&= x + y + y' && \text{Commutative Law} \\
&= x + 1 && \text{Inverse Law} \\
&= 1 && \text{Domination Law}
\end{aligned}$$

The expression $f(x,y) = (x + x'y) + (x + y')$ can be simplified to $f(x,y) = 1$ using the laws of Boolean Algebra.

Exercises
Simplify the following Boolean expressions using the laws of Boolean Algebra:

1. $f(x, y) = (x + y')(x + y)$
2. $f(w, x) = w + [w + (wx)]$
3. $f(x) = (x' + x')'$
4. $f(x) = (x + x')'$
5. $f(w, x, y, z) = w + (wx'yz)$
6. $f(w, x, y, z) = w'(wxyz)'$
7. $f(x, y, z) = [y + x'y + (x + y')]y'$
8. $f(w, x, y, z) = wx + (x'z') + (y + z')$
9. $f(x, y, z) = (x + y)(x + z)$
10. $f(x, y) = [xy + (x + y)']'$
11. $f(w, x, y, z) = (w + x' + y + z')y$
12. $f(x, y, z) = x + y + (x' + y + z)'$
13. $f(x, y, z) = xz + x'y + zy$
14. $f(x, y, z) = (x + z)(x' + y)(z + y)$
15. $f(x, y, z) = x' + y' + xyz'$

1.9 Principle of Duality

The **principle of duality** states that any theorem or identity in Boolean Algebra remains valid if we interchange the AND and OR operations and the constants 0 and 1. In other words, if we replace each AND operation with an OR operation, each OR operation with an AND operation, each 0 with a 1, and each 1 with a 0, the resulting expression is also valid.

Expression	**Dual Expression**
$x + y$	xy
$x(y + z)$	$x + yz$
$x(y + 0)$	$x + y \cdot 1$
$x + 1 = 1$	$x \cdot 0 = 0$
$x + x = x$	$x \cdot x = x$

Table 1.15: Examples of the Principle of Duality

Table 1.15 shows examples of the principle of duality. The dual expression of $x + y$ is xy, the dual expression of $x(y + z)$ is $x + yz$, and so on. The principle of duality states that any theorem or identity in Boolean Algebra remains valid if we interchange the AND and OR operations and the constants 0 and 1.

Exercises
Write the dual expression for the following Boolean expressions:

1. $x + yz$
2. $x(y + z)$
3. $x + y + z$
4. $x(y + z) + x'$
5. $x + y + z + 1$
6. $x + x'$
7. $x + y + z + 0$
8. $x(y + z) + 1$
9. $x + y + z + x'$
10. $x + y + z + x$

2

Logic Gates and Circuits

2.1 Introduction

A **logic gate** is an electronic device that performs a logical operation on one or more binary inputs and produces a single binary output. Logic gates are the building blocks of digital circuits and are used to implement Boolean functions.

Say for example, we have two binary inputs x and y, and we want to perform the AND operation on them. We can use an AND gate to perform this operation. The AND gate takes two binary inputs x and y and produces a single binary output that is the result of the AND operation on x and y.

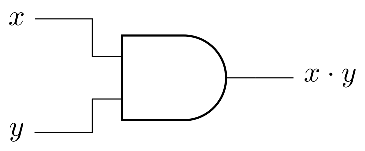


Figure 1: Example of a Logic Circuit for the AND Operation

Figure 1 shows an example of a logic circuit for the AND operation. The circuit takes two binary inputs x and y and produces a single binary output that is the result of the AND operation on x and y. This logic circuit is actually equivalent to the boolean expression $f(x, y) = x \cdot y$.

2.2 Logic Gates and Circuits

There are several types of logic gates that perform different logical operations. The most common logic gates are:

2.2.1 Buffer Gate

A **buffer gate** is a logic gate that takes a single input and produces a single output that is the same as the input.

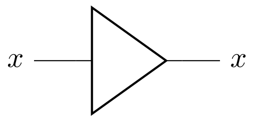


Figure 2: Logic Circuit for the Buffer Gate

Figure 2 shows a logic circuit for the buffer gate. The buffer gate takes a single binary input x and produces a single binary output that is the same as the input.

2.2.2 NOT Gate

A **NOT gate** is a logic gate that takes a single input and produces a single output that is the complement of the input. In boolean algebra, the NOT operation is represented by the prime symbol $'$ or by an overline.

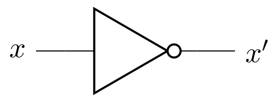


Figure 3: Logic Circuit for the NOT Gate

Figure 3 shows a logic circuit for the NOT gate. The NOT gate takes a single binary input x and produces a single binary output that is the complement of the input.

Exercises
Draw the logic circuits for the following boolean expressions:

1. $f(x) = (x')'$
2. $f(y) = ((y')')'$

2.2.3 AND Gate

An **AND gate** is a logic gate that takes two or more inputs and produces a single output that is the result of the AND operation on the inputs.

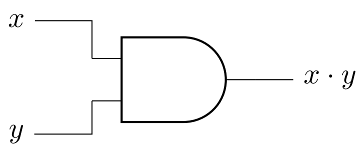


Figure 4: Logic Circuit for the AND Gate

Figure 4 shows a logic circuit for the AND gate. The AND gate takes two binary inputs x and y and produces a single binary output that is the result of the AND operation on x and y. This logic circuit is actually equivalent to the boolean expression $f(x, y) = x \cdot y$. The AND gate can take more than two inputs, and the output is TRUE only if all inputs are TRUE. Otherwise, the output is FALSE.

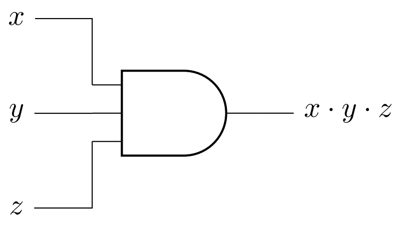


Figure 5: Logic Circuit for the AND Gate with 3 Inputs

Figure 5 shows a logic circuit for the AND gate with three inputs or the boolean expression $f(x, y, z) = x \cdot y \cdot z$. The AND gate takes three binary inputs x, y, and z and produces a single binary output that is the result of the AND operation on x, y, and z.

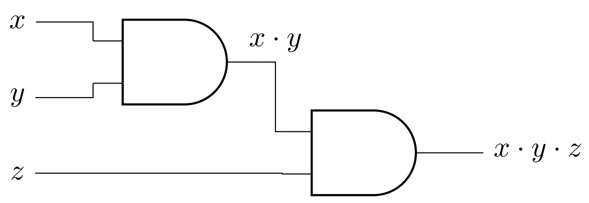


Figure 6: Logic Circuit for the AND Operation $x \cdot y \cdot z$

Figure 6 shows another implementation of the AND operation $x \cdot y \cdot z$ using two AND gates. The first AND gate takes two inputs x and y and produces the output $x \cdot y$. The second AND gate takes the output of the first AND gate and the third input z and produces the output $x \cdot y \cdot z$.

Exercises
Draw the logic circuits for the following boolean expressions:

1. $f(x, y) = xy'$
2. $f(x, y, z) = x'y'$
3. $f(x, y, z) = (xy)'z$

2.2.4 OR Gate

An **OR gate** is a logic gate that takes two or more inputs and produces a single output that is the result of the OR operation on the inputs.

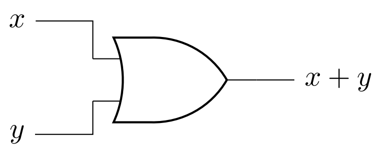


Figure 7: Logic Circuit for the OR Gate

Figure 7 shows a logic circuit for the OR gate for the boolean expression $f(x, y) = x + y$. The OR gate takes two binary inputs x and y and produces a single binary output that is the result of the OR operation on x and y.

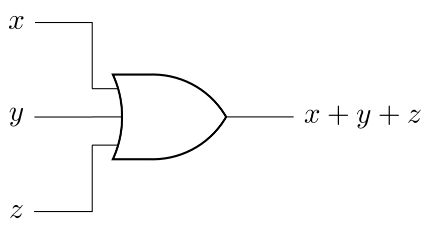


Figure 8: Logic Circuit for the OR Gate with 3 Inputs

Figure 8 shows a logic circuit for the OR gate with three inputs for the boolean expression $f(x, y, z) = x + y + z$. The OR gate takes three binary inputs x, y, and z and produces a single binary output that is the result of the OR operation on x, y, and z.

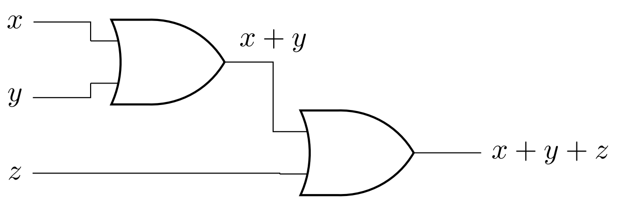


Figure 9: Logic Circuit for the OR Operation $x+y+z$

Figure 9 shows another implementation of the OR operation $x+y+z$ using two OR gates. The first OR gate takes two inputs x and y and produces the output $x+y$. The second OR gate takes the output of the first OR gate and the third input z and produces the output $x+y+z$.

Exercises
Draw the logic circuits for the following boolean expressions:

1. $f(x,y)=x+y'$
2. $f(x,y,z)=(x+y)'z$
3. $f(x,y,z)=xy+xz$
4. $f(x,y,z)=(x+y)(x+z)$

2.2.5 NAND Gate

A **NAND gate** is a logic gate that takes two or more inputs and produces a single output that is the complement of the result of the AND operation on the inputs.

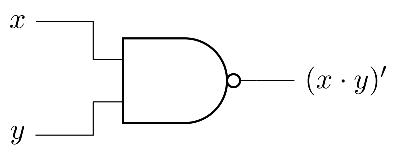


Figure 10: Logic Circuit for the NAND Gate

Figure 10 shows a logic circuit for the NAND gate for the boolean expression $f(x,y)=(x\cdot y)'$. The NAND gate takes two binary inputs x and y and produces a single binary output that is the complement of the result of the AND operation on x and y.

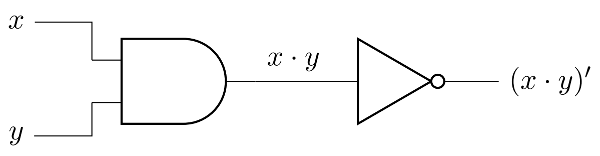


Figure 11: Logic Circuit for the NAND Gate using AND and NOT Gates

Figure 11 shows another implementation of the NAND gate using an AND gate and a NOT gate. The AND gate takes two inputs x and y and produces the output $x\cdot y$. The NOT gate takes the output of the AND gate and produces the complement of the result of the AND operation on x and y. This logic circuit is equivalent to the boolean expression $f(x,y)=(x\cdot y)'$.

Exercises
Draw the logic circuits for the following boolean expressions:

1. $f(x,y,z)=(xyz)'$
2. $f(x,y,z)=(xy)'+(xz)'$

2.2.6 NOR Gate

A **NOR gate** is a logic gate that takes two or more inputs and produces a single output that is the complement of the result of the OR operation on the inputs.

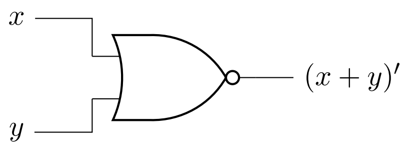


Figure 12: Logic Circuit for the NOR Gate

Figure 12 shows a logic circuit for the NOR gate for the boolean expression $f(x,y) = (x+y)'$. The NOR gate takes two binary inputs x and y and produces a single binary output that is the complement of the result of the OR operation on x and y.

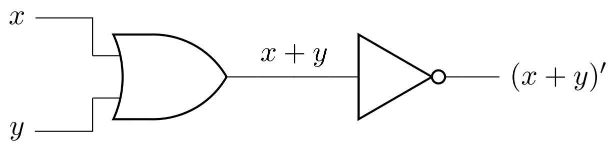


Figure 13: Logic Circuit for the NOR Gate using OR and NOT Gates

Figure 13 shows another implementation of the NOR gate using an OR gate and a NOT gate. The OR gate takes two inputs x and y and produces the output $x+y$. The NOT gate takes the output of the OR gate and produces the complement of the result of the OR operation on x and y. This logic circuit is equivalent to the boolean expression $f(x,y) = (x+y)'$.

Exercises
Draw the logic circuits for the following boolean expressions:

1. $f(x,y) = (x+y)'$
2. $f(x,y,z) = (x+y+z)'$
3. $f(x,y,z) = (x+y)(x+z)'$
4. $f(x,y,z) = (xy)'(x+z)'$

2.2.7 XOR Gate

An **XOR gate** is a logic gate that takes two or more inputs and produces a single output that is the result of the exclusive OR operation on the inputs.

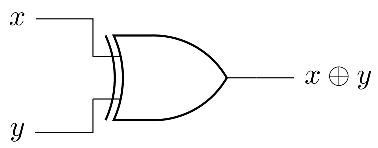


Figure 14: Logic Circuit for the XOR Gate

Figure 14 shows a logic circuit for the XOR gate for the boolean expression $f(x,y) = x \oplus y$. The XOR gate takes two binary inputs x and y and produces a single binary output that is the result of the exclusive OR operation on x and y. The output of the XOR gate is TRUE if the inputs are different and FALSE if the inputs are the same.

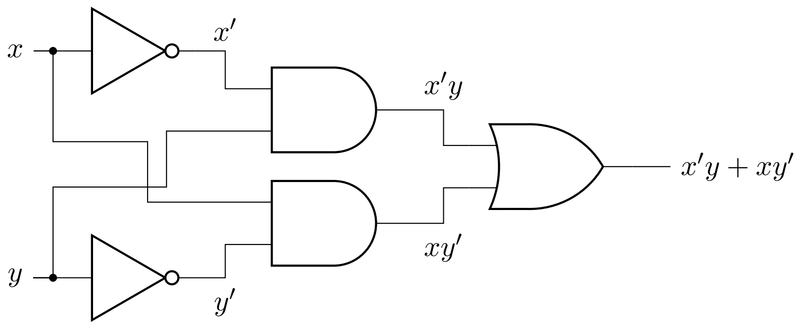


Figure 15: Logic Circuit for the XOR Gate using OR, AND, and NOT Gates

Figure 15 shows another implementation of the XOR gate using OR, AND, and NOT gates. The NOT gates take the inputs x and y and produce the complements x' and y'. The AND gates take the inputs x, y, x', and y' and produce the outputs $x'y$ and xy'. The OR gate takes the outputs of the AND gates and produces the output $x'y + xy'$. This logic circuit is equivalent to the boolean expression $f(x, y) = x \oplus y$. Figure 15 is a more complex implementation of the XOR gate using OR, AND, and NOT gates, but this circuit is equivalent to the XOR gate in Figure 14.

Exercises
Draw the logic circuits for the following boolean expressions:

1. $f(x, y) = x \oplus y'$
2. $f(x, y, z) = x \oplus y \oplus z$

2.2.8 XNOR Gate

An **XNOR gate** is a logic gate that takes two or more inputs and produces a single output that is the complement of the result of the exclusive NOR operation on the inputs.

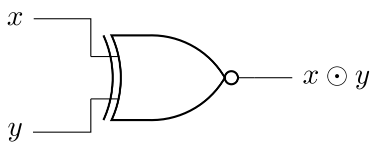


Figure 16: Logic Circuit for the XNOR Gate

Figure 16 shows a logic circuit for the XNOR gate for the boolean expression $f(x, y) = x \odot y$ or $f(x, y) = (x \oplus y)'$. The XNOR gate takes two binary inputs x and y and produces a single binary output that is the complement of the result of the exclusive NOR operation on x and y. The output of the XNOR gate is TRUE if the inputs are the same and FALSE if the inputs are different.

Exercises
Draw the logic circuits for the following boolean expressions:

1. $f(x, y) = x \odot y'$
2. $f(x, y, z) = x \odot y \odot z$
3. $f(x, y, z) = (x \oplus y) \odot z$
4. $f(x, y, z) = [x' + (y \odot z)]'$
5. $f(x, y, z) = (x \odot y) + (x' \odot z)$

2.2.9 Exercises

Exercises

Draw the logic circuits for the following boolean expressions:

1. $f(x, y) = (x + y')(x + y)$
2. $f(w, x) = w + [w + (wx)]$
3. $f(x) = (x' + x')'$
4. $f(x) = (x + x')'$
5. $f(w, x, y, z) = w + (wx'yz)$
6. $f(w, x, y, z) = w'(wxyz)'$
7. $f(x, y, z) = [y + x'y + (x + y')]y'$
8. $f(w, x, y, z) = wx + (x'z') + (y + z')$
9. $f(x, y, z) = (x \odot y)(x \oplus z)$
10. $f(x, y) = [xy + (x \oplus y)']'$
11. $f(w, x, y, z) = (w + x' + y + z')y$
12. $f(x, y, z) = x + y + (x' + y + z)'$
13. $f(x, y, z) = xz + x'y + zy$
14. $f(x, y, z) = (x \odot z)(x' + y)(z \oplus y)$
15. $f(x, y, z) = x' + y' + xyz'$

2.3 Logic Minimization

Logic minimization is the process of simplifying boolean expressions or logic circuits to reduce the number of gates and inputs. The goal of logic minimization is to reduce the cost and complexity of logic circuits while maintaining the same functionality. There are several methods for logic minimization, such as the Karnaugh map method.

2.3.1 Sum of Products and Product of Sums

A boolean expression can be represented in two forms: **sum of products** (SOP) and **product of sums** (POS). The sum of products form is the sum of several products of literals, while the product of sums form is the product of several sums of literals.

2.3.1.1 Sum of Products (SOP)

The sum of products form is the sum of several products of literals. The SOP form is also known as the **disjunctive normal form** (DNF). The general form of a boolean expression in SOP form is:

$$f(x_1, x_2, \ldots, x_n) = \sum m_i$$

where m_i is a product term of literals. The SOP form is the OR operation of several AND operations. m is a product term of literals, it is most commonly known as a **minterm**. A **minterm** is a product term that contains all the variables in the boolean expression.

An example of a boolean expression in Standard SOP form is:

$$f(x, y, z) = x'y'z + x'yz' + xy'z + xyz$$

The expression $f(x,y,z) = x'y'z + x'yz' + xy'z + xyz$ is a boolean expression in SOP form. The expression is the sum of four product terms of literals.

2.3.1.2 Product of Sums (POS)

The product of sums form is the product of several sums of literals. The POS form is also known as the **conjunctive normal form** (CNF). The general form of a boolean expression in POS form is:

$$f(x_1, x_2, \ldots, x_n) = \prod M_i$$

where M_i is a sum term of literals. The POS form is the AND operation of several OR operations. M is a sum term of literals, it is most commonly known as a **maxterm**. A **maxterm** is a sum term that contains all the variables in the boolean expression.

An example of a boolean expression in Standard POS form is:

$$f(x,y,z) = (x+y+z)(x+y+z')(x+y'+z)(x'+y+z)$$

The expression $f(x,y,z) = (x+y+z)(x+y+z')(x+y'+z)(x'+y+z)$ is a boolean expression in POS form. The expression is the product of four sum terms of literals.

Exercise
Determine whether the following boolean expressions are in SOP, POS, Both, or Neither:
1. $f(x,y,z) = x'y'z + x'yz' + xy'z + xyz$
2. $f(x,y,z) = (x+y+z)(x+y+z')(x+y'+z)(x'+y+z)$
3. $f(x,y,z) = xy'z + xyz + z'$
4. $f(x,y,z) = (x+y)(x+z)(y+z)$
5. $f(x,y,z) = x' + y + z'$
6. $f(w,x,y,z) = wx'y'z$
7. $f(x,y,z) = z + x(y+z)$

2.3.1.3 Conversion to SOP and POS Using Truth Tables

A boolean expression can be converted to SOP and POS forms using truth tables. The truth table lists all possible combinations of inputs and the corresponding output of the boolean expression. The SOP form is the sum of **minterms** where the output is TRUE, while the POS form is the product of **maxterms** where the output is FALSE. For the **minterm**, a variable uncomplemented if the input is TRUE and complemented if the input is FALSE. For the **maxterm**, a variable uncomplemented if the input is FALSE and complemented if the input is TRUE.

x	y	$f(x,y)$	Minterm	Maxterm
0	0	0	$m_0 = x'y'$	$M_0 = (x+y)$
0	1	1	$m_1 = x'y$	$M_1 = (x+y')$
1	0	1	$m_2 = xy'$	$M_2 = (x'+y)$
1	1	0	$m_3 = xy$	$M_3 = (x'+y')$

Table 2.1: Truth Table for the Boolean Expression $f(x,y) = x \oplus y$

Table 2.1 shows the truth table for the boolean expression $f(x, y) = x \oplus y$. The SOP form is the sum of minterms, while the POS form is the product of maxterms. The minterms are the product terms where the output is TRUE, while the maxterms are the sum terms where the output is FALSE.

In the truth table, the minterms are the product terms $x'y$ and xy' where the output is TRUE. The maxterms are the sum terms $(x + y)$ and $(x' + y')$ where the output is FALSE.

In this case the SOP form is $f(x, y) = x'y + xy'$ and the POS form is $f(x, y) = (x + y)(x' + y')$. The boolean expression $f(x, y) = x \oplus y$ can be converted to SOP and POS forms using the truth table. Another way to represent the expression is as follows:

$$\begin{aligned} f(x, y) &= \sum m(1, 2) && \text{Sum of Product} \\ f(x, y) &= \prod M(0, 3) && \text{Product of Sum} \end{aligned}$$

The function $f(x, y) = \sum m(1, 2)$ is the sum of the minterms $m_1 = x'y$ and $m_2 = xy'$ where the output is TRUE. The function $f(x, y) = \prod M(0, 3)$ is the product of the maxterms $M_0 = (x + y)$ and $M_3 = (x' + y')$ where the output is FALSE.

x	y	z	$f(x, y, z)$	Minterm	Maxterm
0	0	0	0	$m_0 = x'y'z'$	$M_0 = (x + y + z)$
0	0	1	1	$m_1 = x'y'z$	$M_1 = (x + y + z')$
0	1	0	1	$m_2 = x'yz'$	$M_2 = (x + y' + z)$
0	1	1	0	$m_3 = x'yz$	$M_3 = (x + y' + z')$
1	0	0	1	$m_4 = xy'z'$	$M_4 = (x' + y + z)$
1	0	1	0	$m_5 = xy'z$	$M_5 = (x' + y + z')$
1	1	0	0	$m_6 = xyz'$	$M_6 = (x' + y' + z)$
1	1	1	1	$m_7 = xyz$	$M_7 = (x' + y' + z')$

Table 2.2: Truth Table for the Boolean Expression $f(x, y, z) = x \oplus y \oplus z$

Table 2.2 shows the truth table for the boolean expression $f(x, y, z) = x \oplus y \oplus z$. The SOP form is the sum of minterms, while the POS form is the product of maxterms. The minterms are the product terms $x'y'z$, $x'yz'$, $xy'z$, and xyz where the output is TRUE. The maxterms are the sum terms $(x + y + z)$, $(x + y + z')$, $(x + y' + z)$, and $(x' + y' + z')$ where the output is FALSE.

In this case the SOP form is $f(x, y, z) = x'y'z + x'yz' + xy'z + xyz$ and the POS form is $f(x, y, z) = (x+y+z)(x+y+z')(x+y'+z)(x'+y'+z')$. The boolean expression $f(x, y, z) = x \oplus y \oplus z$ can be converted to SOP and POS forms using the truth table. Another way to represent the expression is as follows:

$$\begin{aligned} f(x, y, z) &= \sum m(1, 2, 4, 7) && \text{Sum of Product} \\ f(x, y, z) &= \prod M(0, 3, 5, 6) && \text{Product of Sum} \end{aligned}$$

The function $f(x, y, z) = \sum m(1, 2, 4, 7)$ is the sum of the minterms $m_1 = x'y'z$, $m_2 = x'yz'$, $m_4 = xy'z$, and $m_7 = xyz$ where the output is TRUE. The function $f(x, y, z) = \prod M(0, 3, 5, 6)$

is the product of the maxterms $M_0 = (x + y + z)$, $M_3 = (x + y + z')$, $M_5 = (x + y' + z)$, and $M_6 = (x' + y' + z')$ where the output is FALSE.

Exercises

1. Convert the following boolean expressions to standard SOP and POS forms using truth tables:
   (a) $f(x, y) = x + y$
   (b) $f(x, y, z) = x + y + z$
   (c) $f(x, y, z) = x' + y + z'$
2. Find the standard SOP and POS forms of the following functions:
   (a) $f(x, y) = \sum m(0, 3)$
   (b) $f(x, y, z) = \sum m(1, 2, 4, 7)$
   (c) $f(x, y) = \prod M(1, 2)$
   (d) $f(x, y, z) = \prod M(0, 3, 5, 6)$

2.3.1.4 Standard and Minimal SOP and POS Forms

The **standard form** of a boolean expression is the SOP or POS where its terms are all **minterms** or **maxterms** of the boolean expression. The **minimal form** of a boolean expression is the SOP or POS form that contains the fewest number of **sum terms** or **product terms** of the boolean expression.

The expression $f(x, y) = x'y + xy' + xy$ is a boolean expression in SOP form. This expression is called the **Standard SOP form** because all of its product terms are **minterms** of the boolean expression. When simplified further using *laws of boolean algebra*, the expression $f(x, y) = x'y + xy' + xy$ can be simplified into $f(x, y) = x + y$ which is the **Minimal SOP form** of the boolean expression.

$$\begin{aligned}
f(x, y) &= x'y + xy' + xy \\
&= x'y + x(y' + y) && \text{Distributive Law} \\
&= x'y + x && \text{Inverse Law} \\
&= x + x'y && \text{Commutative Law} \\
&= x + y && \text{Absorption Law} \\
&= x + y, \text{ Minimal SOP form}
\end{aligned}$$

The expression $f(x, y) = (x' + y)(x + y')(x + y)$ is a boolean expression in POS form. This expression is called the **Standard POS form** because all of its sum terms are **maxterms** of the boolean expression. When simplified further using *laws of boolean algebra*, the expression $f(x, y) = (x' + y)(x + y')(x + y)$ can be simplified into $f(x, y) = xy$ which is the **Minimal POS form** of the boolean expression.

$$
\begin{aligned}
f(x,y) &= (x'+y)(x+y')(x+y) & \\
&= (x'+y)(x+yy') & \text{Distributive Law}\\
&= (x'+y)x & \text{Inverse Law}\\
&= x(x'+y) & \text{Commutative Law}\\
&= xy & \text{Absorption Law}\\
&= xy, \text{ Minimal POS form} &
\end{aligned}
$$

Exercises

1. Find the standard SOP and POS forms of the following boolean expressions:
   (a) $f(x,y) = x + y + xy$
   (b) $f(x,y,z) = (x+y)(x'+y)(x+z)$
   (c) $f(x,y,z) = x' + y + z'$
   (d) $f(w,x,y,z) = wx + wyz + wxz$
   (e) $f(x,y,z) = xy + x'z + yz$
2. Find the minimal SOP and POS forms of the following functions:
   (a) $f(x,y) = \sum m(0,1,2)$
   (b) $f(x,y,z) = \sum m(0,1,2,3)$
   (c) $f(x,y) = \prod M(0,1)$
   (d) $f(x,y,z) = \prod M(0,1,2,3)$
   (e) $f(x,y,z) = \sum m(0,2,4)$

2.3.2 Karnaugh Map

Karnaugh map (K-map) is a graphical method for simplifying boolean expressions. K-maps are used to minimize the number of gates and inputs in a logic. It is best use for boolean expressions with 2 to 6 variables. K-maps are a visual representation of the truth table of a boolean expression. The K-map is a grid-like structure that is divided into cells, where each cell represents a unique combination of input variables. The K-map is used to group adjacent cells that contain 1s or 0s, which represent the **minterms** or **maxterms** of the boolean expression. The number of cells in a K-map is equal to 2^n, where n is the number of input variables. The K-map is divided into groups of adjacent cells, where each group represents a **minterm** or **maxterm**. The groups can be of size 1, 2, 4, 8, or 16 cells, depending on the number of input variables. The groups can overlap, and the goal is to cover all the cells with the fewest number of groups. A cell with a value of TRUE is represented by a 1, while a cell with a value of FALSE is represented by a 0. Cells with a value of 1 are the **minterms** of the boolean expression, while cells with a value of 0 are the **maxterms** of the boolean expression.

When grouping cells in a K-map, the following rules should be followed:

- Groups must be rectangular and contain 1, 2, 4, 8, 16, etc. cells.
- Groups can overlap, but each cell must be covered by at least one group.
- The groups should be as large as possible to minimize the boolean expression.
- Groups can wrap around the edges of the K-map.
- Corner cells can be also be grouped together.
- Grouping is only possible horizontally and vertically, not diagonally.
- **Redundant Groups** should be ignored. A redundant group is a group that is covered by

another group. Redundant groups do not contribute to the simplification of the boolean expression.

2.3.2.1 2-Variable K-map

A K-map for 2 variables has 4 cells, each representing a unique combination of the input variables. The K-map is divided into 2 rows and 2 columns, where each cell corresponds to a **minterm** or **maxterm**. Given a K-map for 2 variables with 1s and 0s, we can formulate the boolean expression in SOP or POS form as well as the minimal SOP or POS form.

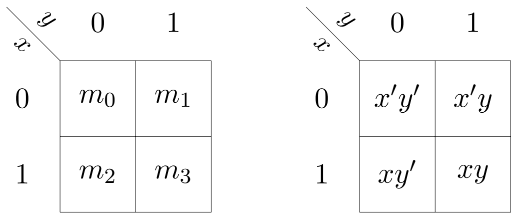


Figure 17: 2-Variable K-map with Minterms

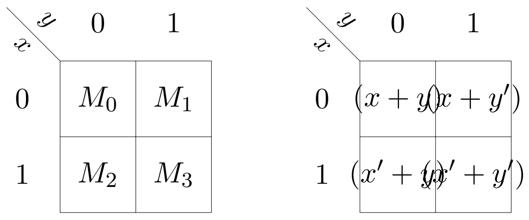


Figure 18: 2-Variable K-map with Maxterms

Figure 17 shows as to which cells in the K-map are the **minterms** of a 2-variable boolean expression. Figure 18 shows as to which cells in are the **maxterms** of a 2-variable boolean expression. To find the SOP form of a boolean expression using a K-map, we find the cells where the output is TRUE (1). The SOP form is the sum of of the **minterms** corresponding to those cells. To find the POS form of a boolean expression using a K-map, we find the cells where the output is FALSE (0). The POS form is the product of the **maxterms** corresponding to those cells. The K-map can also be used to find the minimal SOP by grouping the adjacent cells representing the **minterms** and finding the common literals. The minimal POS can be found by grouping the adjacent cells representing the **maxterms** and finding the common literals.

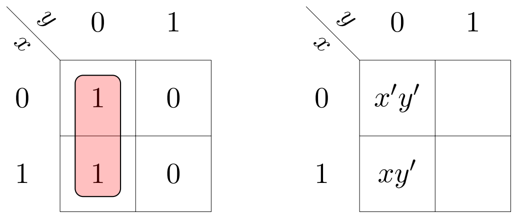


Figure 19: K-map for $x'y' + xy'$ or y'

$$G_1 = m_0 + m_2 = x'y' + xy' = y'$$

$$\text{Standard SOP} = x'y' + xy'$$

$$\text{Minimal SOP} = y'$$

Figure 19 shows the K-map for the boolean expression $f(x, y) = x'y' + xy'$. The boolean expression $f(x, y) = x'y' + xy'$ is the standard SOP form of the boolean expression. It was obtained by extracting the cells representing the **minterms** $m_0 = x'y'$ and $m_2 = xy'$. The sum of these **minterms** is $f(x, y) = x'y' + xy'$ which corresponds to the standard SOP form of the boolean expression. It can further be simplified to $f(x, y) = y'$ which is the minimal SOP form of the boolean expression. The minimal SOP form is obtained by grouping the adjacent cells and finding the common literals. For the K-map in Figure 19, the common literal is y' between m_0 and m_2. For the standard POS form, the boolean expression is obtained by extracting the cells representing the **maxterms** $M_1 = (x + y')$ and $M_4 = (x' + y')$ which gives the standard POS form $f(x, y) = (x + y')(x' + y')$. The minimal POS form is obtained by grouping the adjacent cells and finding the common literals. For the K-map in Figure 19, the common literal is y' between M_1 and M_3. Thus the K-Map can be written as follows:

$f(x, y) = \sum m(0, 2)$	Summation Notation
$f(x, y) = \prod M(1, 3)$	Product Notation
$f(x, y) = x'y' + xy'$	Standard SOP form
$f(x, y) = (x + y')(x' + y')$	Standard POS form
$f(x, y) = y'$	Minimal SOP form
$f(x, y) = y'$	Minimal POS form

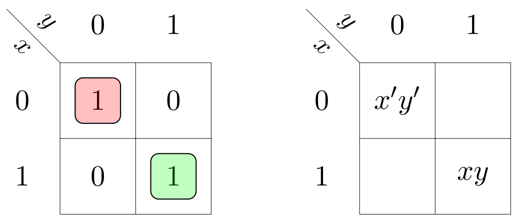


Figure 20: K-map for $\sum m(0, 3)$

$$G_1 = m_0 = x'y' = x'y'$$
$$G_2 = m_3 = xy = xy$$

$$\text{Standard SOP} = x'y' + xy$$

$$\text{Minimal SOP} = x'y' + xy$$

Figure 20 shows the K-map for the boolean expression $f(x, y) = \sum m(0, 3)$. The boolean expression $f(x, y) = \sum m(0, 3)$ is the sum of the **minterms** $m_0 = x'y'$ and $m_3 = xy$. The sum of these **minterms** is $f(x, y) = x'y' + xy$ which corresponds to the standard SOP form of the boolean expression. Since the K-map does not have any adjacent cells, there are no common literals to be found to simplify the expression. Therefore, the standard SOP form is also the minimal SOP form. The expression can be written as follows:

$f(x, y) = \sum m(0, 3)$	Summation Notation
$f(x, y) = \prod M(1, 2)$	Product Notation
$f(x, y) = x'y' + xy$	Standard or Minimal SOP form
$f(x, y) = (x + y)(x' + y')$	Standard or Minimal POS form
$f(x, y) = x \odot y$	Expression as XNOR Operation

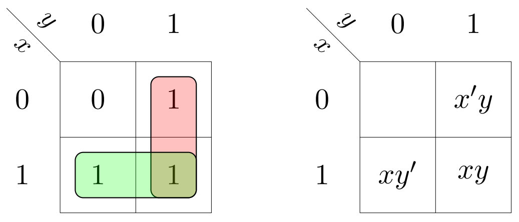


Figure 21: K-map for $\sum m(1, 2, 3)$

$$G_1 = m_1 + m_3 = x'y + xy = y$$
$$G_2 = m_2 + m_3 = xy' + xy = x$$

$$\text{Standard SOP} = x'y + xy' + xy$$

$$\text{Minimal SOP} = x + y$$

Figure 21 shows the K-map for the boolean expression f$(x, y) = \sum m(1, 2, 3)$. The boolean expression $f(x, y) = \sum m(1, 2, 3)$ is the sum of the **minterms** $m_1 = x'y$, $m_2 = xy'$, and $m_3 = xy$.

The sum of these **minterms** is $f(x,y) = x'y + xy' + xy$ which corresponds to the standard SOP form of the boolean expression. The K-map has adjacent cells m_1 and m_3 then m_2 and m_3 which can be grouped together. The common literal between m_1 and m_3 is y and the common literal between m_2 and m_3 is x. Therefore, the standard SOP form can be further simplified to $f(x,y) = x + y$ which is the minimal SOP form of the expression. The expression can be written as follows:

$$\begin{aligned}
f(x,y) &= \sum m(1,2,3) && \text{Summation Notation}\\
f(x,y) &= \prod M(0) && \text{Product Notation}\\
f(x,y) &= x'y + xy' + xy && \text{Standard SOP form}\\
f(x,y) &= (x+y) && \text{Standard POS form}\\
f(x,y) &= x + y && \text{Minimal SOP form}\\
f(x,y) &= (x+y) && \text{Minimal POS form}
\end{aligned}$$

Exercises
Convert the following boolean expressions to standard SOP and POS forms using K-maps and find the minimal SOP and POS forms:

1. $f(x,y) = x + xy'$
2. $f(x,y) = x \oplus y$
3. $f(x,y) = \sum m(0,1,2)$
4. $f(x,y) = \prod M(0,1)$
5. $f(x,y) = \sum m(1,2)$

2.3.3 3-Variable K-map

In a 3-variable K-map, there are 8 cells. Unlike the 2-variable K-map, the 3-variable is composed of 2 rows and 4 columns, where each cell corresponds to a **minterm** or **maxterm**. The K-map is divided into groups of adjacent cells, corner cells, and edge cells.

$x \backslash yz$	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

$x \backslash yz$	00	01	11	10
0	$x'y'z'$	$x'y'z$	$x'yz$	$x'yz'$
1	$xy'z'$	$xy'z$	xyz	xyz'

Figure 22: 3-Variable K-map with Minterms

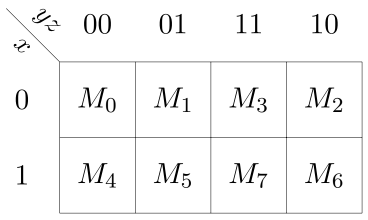


Figure 23: 3-Variable K-map with Maxterms

Figure 22 and 23 shows the K-map for a 3-variable boolean expression with 8 cells. Unlike the 2-variable K-map, where the arrangement of the cells is in a 2x2 grid, the 3-variable K-map is arranged in a 4x2 grid. Also take note of the arrangement of the cells in the K-map, a K-map is arranged in a way that the adjancent cells differ by only one bit. Figure 22 and 23 shows the position of the **minterms** and **maxterms** in the K-map.

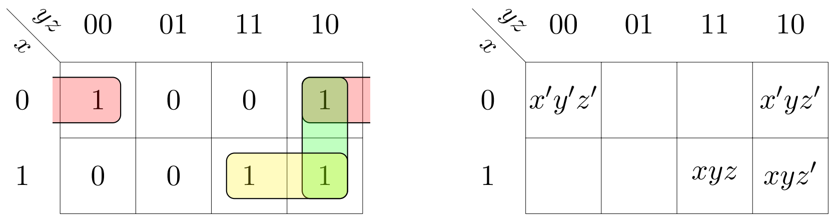


Figure 24: K-map for $x'y'z' + x'yz' + xyz' + xyz$

$$
\begin{aligned}
G_1 &= m_0 + m_2 = x'y'z' + x'yz' = x'z' \\
G_2 &= m_6 + m_7 = xyz' + xyz = xy \\
G_3 &= m_2 + m_6 = x'yz' + xyz' = yz' && \text{Redundant Group, thus ignored}
\end{aligned}
$$

$$\text{Standard SOP} = x'y'z' + x'yz' + xyz' + xyz$$

$$\text{Minimal SOP} = x'z' + xy$$

Figure 24 shows the K-map for the boolean expression $f(x, y, z) = x'y'z' + x'yz' + xyz' + xyz$. The boolean expression $f(x, y, z) = x'y'z' + x'yz' + xyz' + xyz$ is the standard SOP form. It was obtained by extracting the cells representing the **minterms** $m_0 = x'y'z'$, $m_2 = x'yz'$, $m_6 = xyz'$, and $m_7 = xyz$. The sum of these **minterms** is $f(x, y, z) = x'y'z' + x'yz' + xyz' + xyz$ which corresponds to the standard SOP form of the boolean expression. Since the K-map has adjacent cells m_0 and m_2, and m_6 and m_7 which can be grouped together. The common literal between m_0 and m_2 is $x'z'$ and the common literal between m_6 and m_7 is xy. With these common literals the standard SOP form can be further simplified into its minimal SOP form $f(x, y, z) = x'z' + xy$. The group of m_2 and m_6 was ignored because it is a **redundant group** which is a group that is covered by other groups. The expression can be written as follows:

$$\begin{aligned}
&f(x,y,z) = \sum m(0,2,6,7) && \text{Summation Notation}\\
&f(x,y,z) = \prod M(1,3,4,5) && \text{Product Notation}\\
&f(x,y,z) = x'y'z' + x'yz' + xyz' + xyz && \text{Standard SOP form}\\
&f(x,y,z) = (x+y+z')(x+y'+z')(x'+y+z)(x'+y+z') && \text{Standard POS form}\\
&f(x,y,z) = x'z' + xy && \text{Minimal SOP form}\\
&f(x,y,z) = (x+z')(x'+y) && \text{Minimal POS form}
\end{aligned}$$

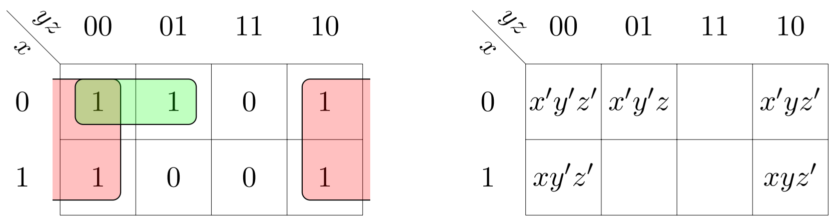


Figure 25: K-map for $x'y'z' + x'y'z + x'yz' + xy'z' + xyz'$

$$\begin{aligned}
G_1 &= m_0 + m_2 + m_4 + m_6 = x'y'z' + x'yz' + xy'z' + xyz' = z'\\
G_2 &= m_0 + m_1 = x'y'z' + x'y'z = x'y'
\end{aligned}$$

$$\text{Standard SOP} = x'y'z' + x'y'z + x'yz' + xy'z' + xyz'$$

$$\text{Minimal SOP} = z' + x'y'$$

Figure 25 shows the K-map for the boolean expression $f(x,y,z) = x'y'z' + x'y'z + x'yz' + xy'z' + xyz'$. The boolean expression $f(x,y,z) = x'y'z' + x'y'z + x'yz' + xy'z' + xyz'$ is the standard SOP form. It was obtained by extracting the cells representing the **minterms** $m_0 = x'y'z'$, $m_1 = x'y'z$, $m_2 = x'yz'$, $m_4 = xy'z'$, and $m_6 = xyz'$. The sum of these **minterms** is $f(x,y,z) = x'y'z' + x'y'z + x'yz' + xy'z' + xyz'$ which corresponds to the standard SOP m_4 and m_6, and m_0 and m_4 which can be grouped together. The common literal between m_0, m_2, m_4, and m_6 is z', and the common literal between m_0 and m_1 is $x'y'$. With these common literals the standard SOP form can be further simplified to $f(x,y,z) = z' + x'y'$. The expression can be written as follows:

$f(x,y,z) = \sum m(0,1,2,4,6)$	Summation Notation
$f(x,y,z) = \prod M(3,5,7)$	Product Notation
$f(x,y,z) = x'y'z' + x'y'z + x'yz' + xy'z' + xyz'$	Standard SOP form
$f(x,y,z) = (x+y'+z')(x'+y+z')(x'+y'+z')$	Standard POS form
$f(x,y,z) = x'y' + z'$	Minimal SOP form
$f(x,y,z) = (y'+z')(x'+z')$	Minimal POS form

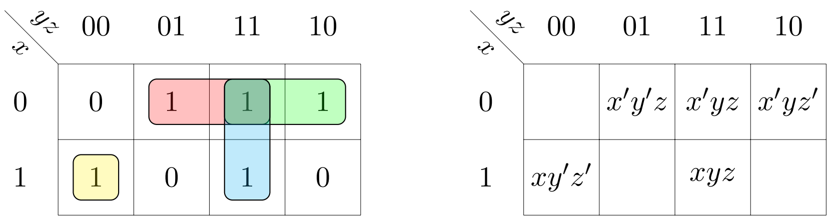


Figure 26: K-map for $x'y'z + x'yz' + x'yz + xy'z' + xyz$

$$\begin{aligned}
G_1 &= m_1 + m_3 = x'y'z + x'yz = x'z\\
G_2 &= m_2 + m_3 = x'yz' + x'yz = x'y\\
G_3 &= m_4 = xy'z' = xy'z'\\
G_4 &= m_3 + m_7 = x'yz + xyz = yz
\end{aligned}$$

$$\text{Standard SOP} = x'y'z + x'yz' + x'yz + xy'z' + xyz$$

$$\text{Minimal SOP} = x'z + x'y + yz + xy'z'$$

Figure 26 shows the K-map for the boolean expression $f(x,y,z) = x'y'z + x'yz' + x'yz + xy'z' + xyz$. The boolean expression $f(x,y,z) = x'y'z + x'yz' + x'yz + xy'z' + xyz$ is the standard SOP form. It was obtained by extracting the cells representing the **minterms** $m_1 = x'y'z$, $m_2 = x'yz'$, $m_3 = x'yz$, $m_4 = xy'z'$, and $m_7 = xyz$. The sum of these **minterms** is $f(x,y,z) = x'y'z + x'yz' + x'yz + xy'z' + xyz$ which corresponds to the standard SOP form of the boolean expression. The K-map has adjacent cells m_1 and m_3, m_2 and m_3, m_4, and m_3 and m_7 which can be grouped together. The common literal between m_1 and m_3 is $x'z$, the common literal between m_2 and m_3 is $x'y$, the common literal between m_4 is $xy'z'$, and the common literal between m_3 and m_7 is yz. With these common literals the standard SOP form can be further simplified to $f(x,y,z) = x'z + x'y + yz + xy'z'$. The expression can be written as follows:

$f(x,y,z) = \sum m(1,2,3,4,7)$	Summation Notation
$f(x,y,z) = \prod M(0,5,6)$	Product Notation
$f(x,y,z) = x'y'z + x'yz' + x'yz + xy'z' + xyz$	Standard SOP form
$f(x,y,z) = (x+y+z)(x'+y+z')(x'+y'+z)$	Standard POS form
$f(x,y,z) = x'z + x'y + yz + xy'z'$	Minimal SOP form
$f(x,y,z) = (x+y+z)(x'+y+z')(x'+y'+z)$	Minimal POS form

Exercises
Convert the following boolean expressions to standard SOP and POS forms using K-maps and find the minimal SOP and POS forms:

1. $f(x,y,z) = x + xy'z'$
2. $f(x,y,z) = x \oplus y \oplus z$
3. $f(x,y,z) = \sum m(0,1,2,5)$
4. $f(x,y,z) = \prod M(0,1)$
5. $f(x,y,z) = \sum m(1,2,4,5,6,7)$

2.3.4 4-Variable K-map

A K-map for 4 variables has 16 cells. The K-map is composed of 4 rows and 4 columns. Similar to the 3-variable K-map, the arrangement of the cells in the K-map is in a way that the adjacent cells differ by only one bit. The K-map is divided into groups of adjacent cells, corner cells, and edge cells.

$wx \backslash yz$	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

$wx \backslash yz$	00	01	11	10
00	M_0	M_1	M_3	M_2
01	M_4	M_5	M_7	M_6
11	M_{12}	M_{13}	M_{15}	M_{14}
10	M_8	M_9	M_{11}	M_{10}

Figure 27: 4-Variable K-map with Minterms and Maxterms

Figure 27 shows the K-map for a 4-variable boolean expression with 16 cells. The arrangement of the cells in the K-map is in a 4x4 grid.

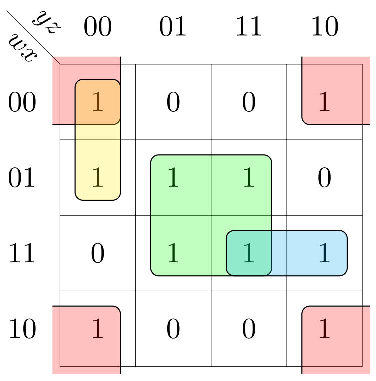


Figure 28: K-map for $m_0 + m_2 + m_4 + m_5 + m_7 + m_8 + m_{10} + m_{13} + m_{14} + m_{15}$

$$
\begin{aligned}
G_1 &= m_0 + m_2 + m_8 + m_{10} = w'x'y'z' + w'x'yz' + wx'y'z' + wx'yz' = x'z' \\
G_2 &= m_0 + m_4 = w'x'y'z' + w'xy'z' = w'y'z' \\
G_3 &= m_5 + m_7 + m_{13} + m_{15} = w'xy'z + w'xyz + wxy'z + wxyz = xz \\
G_4 &= m_{14} + m_{15} = wxyz' + wxyz = wxy
\end{aligned}
$$

$$
\begin{aligned}
\text{Standard SOP} = {} & w'x'y'z' + w'x'yz' + w'xy'z' + w'xy'z + w'xyz + wx'y'z' + wx'yz' + \\
& wxy'z + wxyz' + wxyz
\end{aligned}
$$

$$\text{Minimal SOP} = x'z' + w'y'z' + xz + wxy$$

Figure 28 shows the K-map for the boolean expression $f(x, y, z, w) = m_0 + m_2 + m_4 + m_5 + m_7 + m_8 + m_{10} + m_{13} + m_{14} + m_{15}$. The boolean expression $f(x, y, z, w) = m_0 + m_2 + m_4 + m_5 + m_7 + m_8 + m_{10} + m_{13} + m_{14} + m_{15}$ is the standard SOP form. It was obtained by extracting the cells representing the **minterms** $m_0 = w'x'y'z'$, $m_2 = w'x'yz'$, $m_4 = w'xy'z'$, $m_5 = w'xy'z$, $m_7 = w'xyz$, $m_8 = wx'y'z'$, $m_{10} = wx'yz'$, $m_{13} = wxy'z$, $m_{14} = wxyz'$, and $m_{15} = wxyz$. The sum of these **minterms** is $f(x, y, z, w) = w'x'y'z' + w'x'yz' + w'xy'z' + w'xy'z + w'xyz + wx'y'z' + wx'yz' + wxy'z + wxyz' + wxyz$ which corresponds to the standard SOP form of the boolean expression. The K-map has the groups $G_1 = m_0 + m_2 + m_8 + m_{10}$, $G_2 = m_0 + m_4$, $G_3 = m_5 + m_7 + m_{13} + m_{15}$, and $G_4 = m_{14} + m_{15}$. Other groups can also be formed but they are **redundant groups** as they are already covered by other groups. The common literal between m_0, m_2, m_8, and m_{10} is $x'z'$; the common literal between m_0 and m_4 is $w'y'z'$; the common literal between m_5, m_7, m_{13}, and m_{15} is xz; and the common literal between m_{14} and m_{15} is wxy. With these common literals the standard SOP form can be further simplified to $f(x, y, z, w) = x'z' + w'y'z' + xz + wxy$ which is the minimal SOP form of the expression.

Exercises
Convert the following boolean expressions to standard SOP and POS forms using K-maps and find the minimal SOP and POS forms:

1. $f(w, x, y, z) = w + x'y'z'$
2. $f(w, x, y, z) = w \oplus x \oplus y \oplus z$
3. $f(w, x, y, z) = \sum m(0, 1, 2, 3, 4, 8, 9, 10, 11, 12)$
4. $f(w, x, y, z) = \sum m(0, 4, 5, 10, 11, 13, 15)$
5. $f(w, x, y, z) = \prod M(1, 5, 7, 9, 12, 13, 14, 15)$

3

Graphs

3.1 Introduction to Graphs

Graphs are discrete structures that consist of a set of vertices and a set of edges. A graph is defined as $G = (V, E)$ where V is the set of vertices and E is the set of edges. The edges can be directed or undirected, and they can also have weights associated with them.

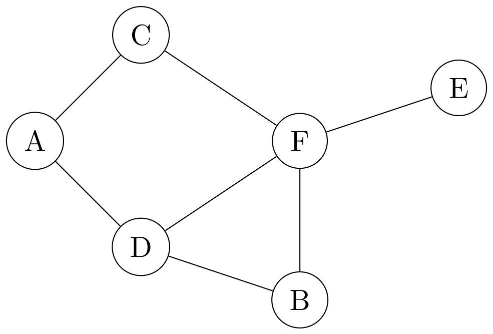


Figure 29: Undirected Graph

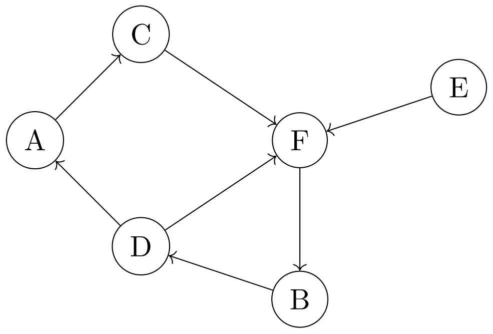


Figure 30: Directed Graph

Figure 29 shows an undirected graph where the edges do not have a direction. The edges can be traversed in both directions. Figure 30 shows a directed graph where the edges have a direction. The edges can only be traversed in the direction of the arrow.

3.1.1 Applications of Graphs

Graphs are used in various applications such as:

- Computer Networks: Graphs are used to represent the connections between computers in a network.
- Social Networks: Graphs are used to represent the relationships between people in a social network.
- Transportation Networks: Graphs are used to represent the routes and connections between different locations in a transportation network.
- Circuit Design: Graphs are used to represent the connections between components in a circuit.

3.2 Graph Terminology

A **graph** is defined as $G = (V, E)$ where V is the set of **vertices** and E is the set of **edges**. An edge $e \in E$ is denoted in the form $e = \{x, y\}$, where the vertices $x, y \in V$ is the **endpoints** of the edge. Two vertices x and y connected by the edge $e = \{x, y\}$, are said to be **adjacent** to each other.

A **simple graph** is a graph that does not contain any loops or multiple edges. All **simple graphs** are undirected graphs. An **undirected graph** is a graph where the edges do not have a direction.

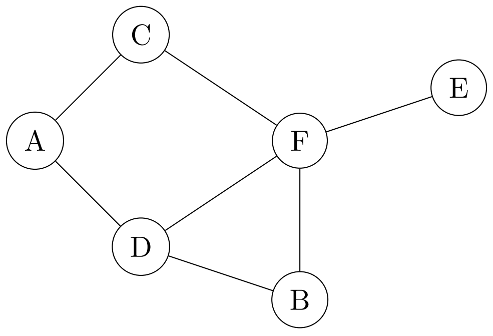


Figure 31: Undirected Graph

Figure 31 shows an undirected graph where the edges do not have a direction. The graph shown has a vertex set:

$$V = \{A, B, C, D, E, F\}$$

and an edge set:

$$E = \{\{A, C\}, \{A, D\}, \{B, D\}, \{B, F\}, \{C, F\}, \{D, F\}, \{F, E\}\}$$

A **directed graph** is a graph where the edges have a direction. It is also called as **digraph**. When a **directed graph** has no loops or multiple edges, it is called a **simple directed graph**. To denote a directed graph, the edge set are ordered.

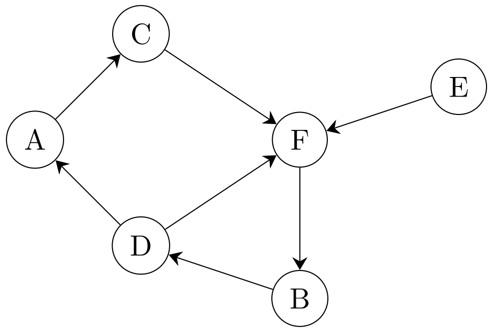


Figure 32: Directed Graph

Figure 32 shows a directed graph where the edges have a direction. The graph shown has a vertex set:

$$V = \{A, B, C, D, E, F\}$$

and an edge set where the vertex pairs are ordered to indicate the direction:

$$E = \{\{A, C\}, \{B, D\}, \{C, F\}, \{D, A\}, \{D, F\}, \{E, F\}, \{F, B\}\}$$

The **degree** of a vertex is the number of edges incident to it and is denoted by $deg(v)$. In figure 31, the degrees are as follows:

$$deg(A) = 2$$
$$deg(B) = 2$$
$$deg(C) = 2$$
$$deg(D) = 3$$
$$deg(E) = 1$$
$$deg(F) = 4$$

If the sum of the degrees $deg(A) + deg(B) + deg(C) + deg(D) + deg(E) + deg(F) = 14$ is twice the number of edges $|E| = 7$, then we state this result as **Handshaking Theorem** or **Handshaking Lemma**. The **Handshaking Theorem** states that the sum of the degrees of the vertices in a graph $G = (V, E)$ is equal to twice the number of edges in the graph.

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

In **directed graphs**, the degree of a vertex is defined as the sum of the **in-degree** and **out-degree**. The **in-degree** $deg^{-}(v)$ of a vertex is the number of edges directed towards the vertex or the number of edges that goes into the vertex. The **out-degree** $deg^{+}(v)$ of a vertex is the number of edges directed away from the vertex or the number of edges that goes out of the vertex. In figure 32, the **in-degrees** and **out-degrees** are as follows:

$Degree$	$In - degree$	$Out - degree$
$deg(A) = deg^{-}(A) + deg^{+}(A)$	$deg^{-}(A) = 1$	$deg^{+}(A) = 1$
$deg(B) = deg^{-}(B) + deg^{+}(B)$	$deg^{-}(B) = 1$	$deg^{+}(B) = 1$
$deg(C) = deg^{-}(C) + deg^{+}(C)$	$deg^{-}(C) = 1$	$deg^{+}(C) = 1$
$deg(D) = deg^{-}(D) + deg^{+}(D)$	$deg^{-}(D) = 1$	$deg^{+}(D) = 2$
$deg(E) = deg^{-}(E) + deg^{+}(E)$	$deg^{-}(E) = 0$	$deg^{+}(E) = 1$
$deg(F) = deg^{-}(F) + deg^{+}(F)$	$deg^{-}(F) = 3$	$deg^{+}(F) = 1$

A graph can also have **loops**. A **loop** is an edge that connects a vertex to itself. The degree of a vertex with a loop is increased by 2. For example, in figure 33, the vertex A has a loop. The degree of vertex A is $deg(A) = 2 + 2 = 4$.

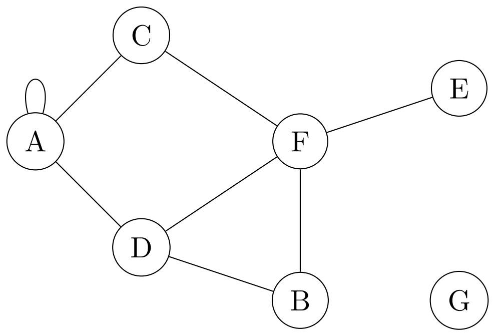


Figure 33: Undirected Graph with Loop and Isolated Vertex

Figure 33 shows an undirected graph with a loop on vertex A and an isolated vertex G. An **isolated vertex** is a vertex that has no edges incident to it or has a degree of 0.

Exercises

1. Draw the undirected graph for the following vertex and edge sets and determine the degree of each vertex:

   (a)

$$V = \{A, B, C, D, E, F\}$$
$$E = \{\{A, B\}, \{A, C\}, \{B, C\}, \{C, D\}, \{C, E\}, \{D, E\}, \{E, F\}\}$$

   (b)

$$V = \{A, B, C, D, E, F\}$$
$$E = \{\{A, B\}, \{B, C\}, \{B, D\}, \{C, D\}, \{C, E\}, \{D, E\}, \{E, F\}\}$$

2. Draw the directed graph for the following vertex and edge sets, determine the in-degree and out-degree of each vertex:

   (a)

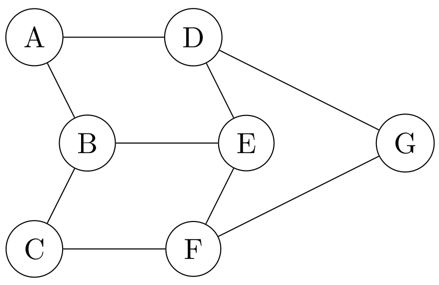
$$V = \{A, B, C, D, E, F\}$$
$$E = \{\{A, B\}, \{B, D\}, \{C, B\}, \{D, E\}, \{E, F\}, \{F, D\}\}$$

   (b)

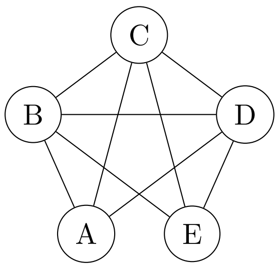
$$V = \{A, B, C, D, E, F, G\}$$
$$E = \{\{A, B\}, \{A, C\}, \{B, D\}, \{C, D\}, \{E, D\}, \{F, D\}, \{G, E\}, \{G, F\}\}$$

3. Find the vertex set and edge set of the following graphs:

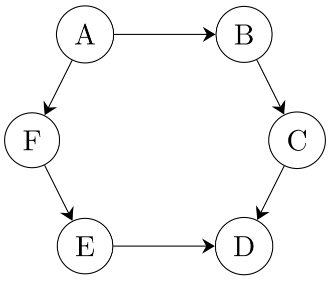
   (a)



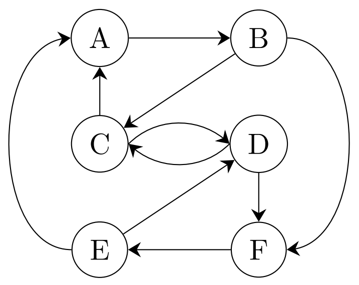
   (b)



   (c)



(d)



3.3 Graph Representation

There are several ways to represent a graph. The most common representations are the **adjacency list**, **adjacency matrix**, and **incidence matrix**. In this section, we will focus on the representation of graphs using **adjacency matrix**, and **incidence matrix**. We will not cover the **adjacency list** representation as it is not commonly used for graph representation.

3.3.1 Adjacency Matrix

An **adjacency matrix** is a square matrix used to represent a graph. The rows and columns of the matrix represent the vertices of the graph. The values in the matrix indicate whether pairs of vertices are adjacent or not in the graph. The adjacency matrix is a symmetric matrix for undirected graphs and a non-symmetric matrix for directed graphs. The adjacency matrix is defined as:

$$A_{ij} = \begin{cases} 1 & \text{if } e = \{v_i, v_j\} \text{ is an edge of } G, e \in E \\ 0 & \text{otherwise} \end{cases}$$

where A_{ij} is the value in the i^{th} row and j^{th} column of the **adjacency matrix** A of the graph G. The adjacency matrix is a square matrix of size $|V| \times |V|$ where $|V|$ is the number of vertices in the graph.

3.3.1.1 Adjacency Matrix for Undirected Graphs

The adjacency matrix for undirected graphs is symmetric. The value in the (i, j) position of the matrix is equal to the value in the (j, i) position of the matrix. This is because the edges in undirected graphs do not have a direction. Say we have an undirected graph G with the following vertex set and edge set:

$$V = \{a, b, c, d\}$$
$$E = \{\{a, b\}, \{a, c\}, \{b, c\}, \{c, d\}\}$$

We can represent the graph G using an adjacency matrix as follows:

$$\begin{array}{c|cccc|} & a & b & c & d \\ a & 0 & 1 & 1 & 0 \\ b & 1 & 0 & 1 & 0 \\ c & 1 & 1 & 0 & 1 \\ d & 0 & 0 & 1 & 0 \end{array}$$

Figure 34: Adjacency Matrix

Figure 34 shows the adjacency matrix for the graph G. Here, the adjacency matrix is a 4×4 matrix as there are 4 vertices in the graph. The rows and columns of the matrix represent the vertices of the graph in which in this case the vertices are represented as a, b, c, d. The value 1 in the matrix indicates that there is an edge between the vertices v_i and v_j. The example above is the **adjacency matrix** of the graph:

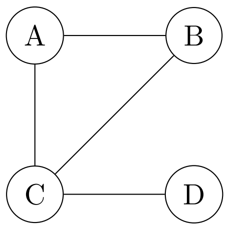


Figure 35: Graph G of the Adjacency Matrix

Figure 35 shows the graph G with the vertex set $V = \{a, b, c, d\}$ and edge set $E = \{\{a, b\}, \{a, c\}, \{b, c\}, \{c, d\}\}$. This is the graph for Figure 34.

3.3.1.2 Adjacency Matrix for Directed Graphs

In directed graphs, the adjacency matrix is not symmetric. The value in the (i, j) position of the matrix is not equal to the value in the (j, i) position of the matrix. This is because the edges in directed graphs have a direction. Say we have a directed graph G with the following vertex set and edge set:

$$V = \{a, b, c, d, e\}$$
$$E = \{\{a, b\}, \{a, c\}, \{b, c\}, \{b, e\}, \{c, d\}, \{d, a\}, \{d, e\}\}$$

We can represent the graph G using an adjacency matrix as follows:

$$\begin{array}{c|ccccc|} & a & b & c & d & e \\ a & 0 & 1 & 1 & 0 & 0 \\ b & 0 & 0 & 1 & 0 & 1 \\ c & 0 & 0 & 0 & 1 & 0 \\ d & 1 & 0 & 0 & 0 & 1 \\ e & 0 & 0 & 0 & 0 & 0 \end{array}$$

Figure 36: Adjacency Matrix for Directed Graph

Figure 36 shows the adjacency matrix for the directed graph G. Here, the adjacency matrix is a 5×5 matrix as there are 5 vertices in the graph. The rows and columns of the matrix represent the vertices of the graph in which in this case the vertices are represented as a, b, c, d, e. The value 1 in the matrix indicates that there is an edge between the vertices v_i and v_j. The example above is the **adjacency matrix** of the graph:

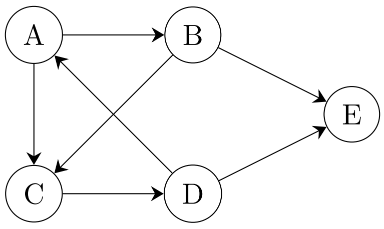


Figure 37: Graph G of the Adjacency Matrix for Directed Graph

Figure 37 shows the graph G with the vertex set $V = \{a, b, c, d, e\}$ and edge set $E = \{\{a, b\}, \{a, c\}, \{b, c\}, \{b, e\}, \{c, d\}, \{d, a\}, \{d, e\}\}$. This is the graph for Figure 36.

Exercises

1. Draw the undirected graph for the following vertex and edge sets and represent the graph using adjacency matrix:

   (a)

$$V = \{a, b, c, d, e, f\}$$
$$E = \{\{a, b\}, \{a, c\}, \{a, d\}, \{a, e\}, \{b, c\}, \{c, f\}, \{d, e\}, \{e, f\}, \{f, f\}\}$$

   (b)

$$V = \{a, b, c, d, e, f, g\}$$
$$E = \{\{a, a\}, \{a, b\}, \{a, c\}, \{a, e\}, \{a, g\}, \{c, d\}, \{d, e\}, \{e, f\}, \{f, g\}\}$$

2. Draw the directed graph for the following vertex and edge sets, and represent the graph using adjacency matrix:

   (a)

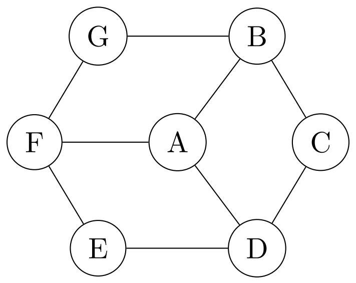
$$V = \{a, b, c, d, e, f\}$$
$$E = \{\{a, b\}, \{a, d\}, \{b, c\}, \{c, a\}, \{c, a\}, \{c, c\}, \{d, c\}, \{e, d\}, \{f, b\}\}$$

   (b)

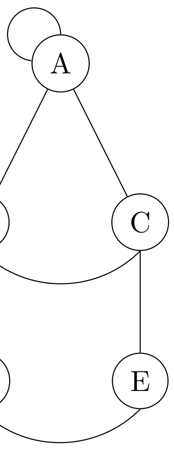
$$V = \{a, b, c, d, e, f, g\}$$
$$E = \{\{a, c\}, \{b, a\}, \{c, b\}, \{c, d\}, \{d, e\}, \{e, c\}, \{e, f\}, \{f, c\}, \{g, e\}\}$$

3. Represent the following graphs using adjacency matrix:

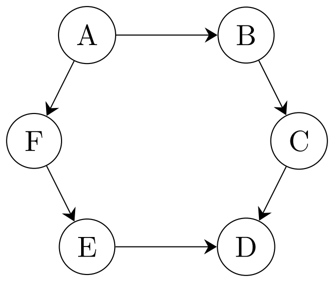
   (a)



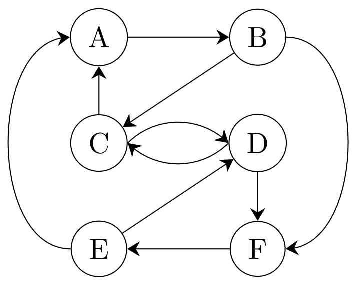
   (b)



   (c)



   (d)



4

Trees

5

Network Models and Petri Nets

6

Automata, Grammars and Languages

6.1 Languages and Grammars

6.2 Finite State Automata

6.3 Regular Expressions

7

Computational Geometry

8

References

A. **Books**

-

B. **Other Sources**

-