Discrete Mathematics

mathematical structures

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Discrete Mathematics is a branch of mathematics involving discrete elements that uses algebra and arithmetic. It is increasingly being applied in the practical fields of mathematics and computer science. It is a very good tool for improving reasoning and problem-solving capabilities.

This tutorial explains the fundamental concepts of Sets, Relations and Functions, Mathematical Logic, Group theory, Counting Theory, Probability, Mathematical Induction and Recurrence Relations, Graph Theory, Trees and Boolean Algebra.

Audience

This tutorial has been prepared for students pursuing a degree in any field of computer science and mathematics. It endeavors to help students grasp the essential concepts of discrete mathematics.

Prerequisites

This tutorial has an ample amount of both theory and mathematics. The readers are expected to have a reasonably good understanding of elementary algebra and arithmetic.

Copyright & Disclaimer

Table of Contents

1. Discrete Mathematics – Introduction

Mathematics can be broadly classified into two categories:

- **Continuous Mathematics** ─ It is based upon continuous number line or the real numbers. It is characterized by the fact that between any two numbers, there are almost always an infinite set of numbers. For example, a function in continuous mathematics can be plotted in a smooth curve without breaks.

- **Discrete Mathematics** ─ It involves distinct values; i.e. between any two points, there are a countable number of points. For example, if we have a finite set of objects, the function can be defined as a list of ordered pairs having these objects, and can be presented as a complete list of those pairs.

Topics in Discrete Mathematics

Though there cannot be a definite number of branches of Discrete Mathematics, the following topics are almost always covered in any study regarding this matter:

- Sets, Relations and Functions
- Mathematical Logic
- Group theory
- Counting Theory
- Probability
- Mathematical Induction and Recurrence Relations
- Graph Theory
- Trees
- Boolean Algebra

We will discuss each of these concepts in the subsequent chapters of this tutorial.

Part 1: Sets, Relations, and Functions

2. Sets

German mathematician **G. Cantor** introduced the concept of sets. He had defined a set as a collection of definite and distinguishable objects selected by the means of certain rules or description.

Set theory forms the basis of several other fields of study like counting theory, relations, graph theory and finite state machines. In this chapter, we will cover the different aspects of **Set Theory**.

Set – Definition

A set is an unordered collection of different elements. A set can be written explicitly by listing its elements using set bracket. If the order of the elements is changed or any element of a set is repeated, it does not make any changes in the set.

Some Example of Sets

- A set of all positive integers
- A set of all the planets in the solar system
- A set of all the states in India
- A set of all the lowercase letters of the alphabet

Representation of a Set

Sets can be represented in two ways:

- Roster or Tabular Form
- Set Builder Notation

Roster or Tabular Form

The set is represented by listing all the elements comprising it. The elements are enclosed within braces and separated by commas.

Example 1: Set of vowels in English alphabet, A = {a,e,i,o,u}

Example 2: Set of odd numbers less than 10, B = {1,3,5,7,9}

Set Builder Notation

The set is defined by specifying a property that elements of the set have in common. The set is described as A = { x : p(x)}

Example 1: The set {a,e,i,o,u} is written as:
A = { x : x is a vowel in English alphabet}

Example 2: The set {1,3,5,7,9} is written as:

$B = \{ x : 1 \le x < 10 \text{ and } (x\%2) \neq 0\}$

If an element x is a member of any set S, it is denoted by $x \in S$ and if an element y is not a member of set S, it is denoted by $y \notin S$.

Example: If $S = \{1, 1.2, 1.7, 2\}$, $1 \in S$ but $1.5 \notin S$

Some Important Sets

N: the set of all natural numbers = {1, 2, 3, 4, ...}

Z: the set of all integers = {....., -3, -2, -1, 0, 1, 2, 3, ...}

Z^+: the set of all positive integers

Q: the set of all rational numbers

R: the set of all real numbers

W: the set of all whole numbers

Cardinality of a Set

Cardinality of a set S, denoted by |S|, is the number of elements of the set. The number is also referred as the cardinal number. If a set has an infinite number of elements, its cardinality is ∞.

Example: $|\{1, 4, 3, 5\}| = 4$, $|\{1, 2, 3, 4, 5, \ldots\}| = \infty$

If there are two sets X and Y,

- $|X| = |Y|$ denotes two sets X and Y having same cardinality. It occurs when the number of elements in X is exactly equal to the number of elements in Y. In this case, there exists a bijective function 'f' from X to Y.

- $|X| \le |Y|$ denotes that set X's cardinality is less than or equal to set Y's cardinality. It occurs when number of elements in X is less than or equal to that of Y. Here, there exists an injective function 'f' from X to Y.

- $|X| < |Y|$ denotes that set X's cardinality is less than set Y's cardinality. It occurs when number of elements in X is less than that of Y. Here, the function 'f' from X to Y is injective function but not bijective.

- If $|X| \le |Y|$ and $|Y| \le |X|$ then $|X| = |Y|$. The sets X and Y are commonly referred as equivalent sets.

Types of Sets

Sets can be classified into many types. Some of which are finite, infinite, subset, universal, proper, singleton set, etc.

Finite Set

A set which contains a definite number of elements is called a finite set.

Example: S = {x | x ∈ N and 70 > x > 50}

Infinite Set

A set which contains infinite number of elements is called an infinite set.

Example: S = {x | x ∈ N and x > 10}

Subset

A set X is a subset of set Y (Written as X ⊆ Y) if every element of X is an element of set Y.

Example 1: Let, X = { 1, 2, 3, 4, 5, 6 } and Y = { 1, 2 }. Here set Y is a subset of set X as all the elements of set Y is in set X. Hence, we can write Y ⊆ X.

Example 2: Let, X = {1, 2, 3} and Y = {1, 2, 3}. Here set Y is a subset (Not a proper subset) of set X as all the elements of set Y is in set X. Hence, we can write Y ⊆ X.

Proper Subset

The term "proper subset" can be defined as "subset of but not equal to". A Set X is a proper subset of set Y (Written as X ⊂ Y) if every element of X is an element of set Y and |X| < |Y|.

Example: Let, X = {1, 2, 3, 4, 5, 6} and Y = {1, 2}. Here set Y ⊂ X since all elements in Y are contained in X too and X has at least one element is more than set Y.

Universal Set

It is a collection of all elements in a particular context or application. All the sets in that context or application are essentially subsets of this universal set. Universal sets are represented as U.

Example: We may define U as the set of all animals on earth. In this case, set of all mammals is a subset of U, set of all fishes is a subset of U, set of all insects is a subset of U, and so on.

Empty Set or Null Set

An empty set contains no elements. It is denoted by ∅. As the number of elements in an empty set is finite, empty set is a finite set. The cardinality of empty set or null set is zero.

Example: S = {x | x ∈ N and 7 < x < 8} = ∅

Singleton Set or Unit Set

Singleton set or unit set contains only one element. A singleton set is denoted by {s}.

Example: S = {x | x ∈ N, 7 < x < 9} = { 8 }

Equal Set

If two sets contain the same elements they are said to be equal.

Example: If A = {1, 2, 6} and B = {6, 1, 2}, they are equal as every element of set A is an element of set B and every element of set B is an element of set A.

Equivalent Set

If the cardinalities of two sets are same, they are called equivalent sets.

Example: If A = {1, 2, 6} and B = {16, 17, 22}, they are equivalent as cardinality of A is equal to the cardinality of B. i.e. |A|=|B|=3

Overlapping Set

Two sets that have at least one common element are called overlapping sets.

In case of overlapping sets:

- n(A ∪ B) = n(A) + n(B) - n(A ∩ B)
- n(A ∪ B) = n(A - B) + n(B - A) + n(A ∩ B)
- n(A) = n(A - B) + n(A ∩ B)
- n(B) = n(B - A) + n(A ∩ B)

Example: Let, A = {1, 2, 6} and B = {6, 12, 42}. There is a common element '6', hence these sets are overlapping sets.

Disjoint Set

Two sets A and B are called disjoint sets if they do not have even one element in common. Therefore, disjoint sets have the following properties:

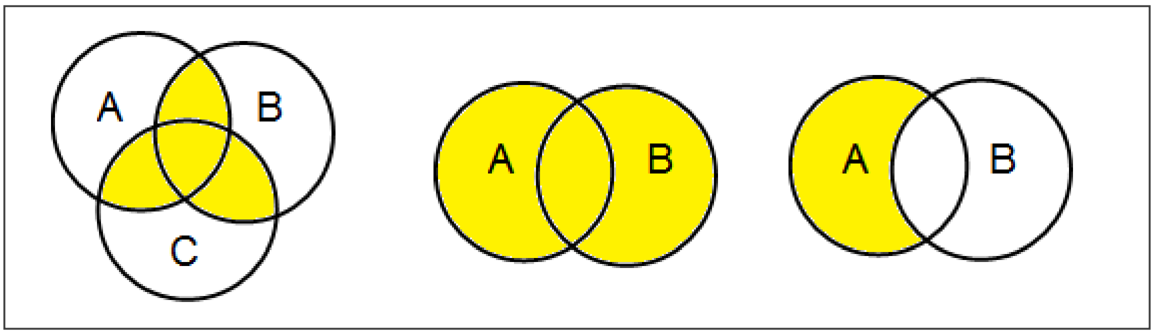
- n(A ∩ B) = ∅
- n(A ∪ B) = n(A) + n(B)

Example: Let, A = {1, 2, 6} and B = {7, 9, 14}; there is not a single common element, hence these sets are overlapping sets.

Venn Diagrams

Venn diagram, invented in1880 by John Venn, is a schematic diagram that shows all possible logical relations between different mathematical sets.

Examples



Set Operations

Set Operations include Set Union, Set Intersection, Set Difference, Complement of Set, and Cartesian Product.

Set Union

The union of sets A and B (denoted by A ∪ B) is the set of elements which are in A, in B, or in both A and B. Hence, A∪B = {x | x ∈A OR x ∈B}.

Example: If A = {10, 11, 12, 13} and B = {13, 14, 15}, then A ∪ B = {10, 11, 12, 13, 14, 15}. (The common element occurs only once)

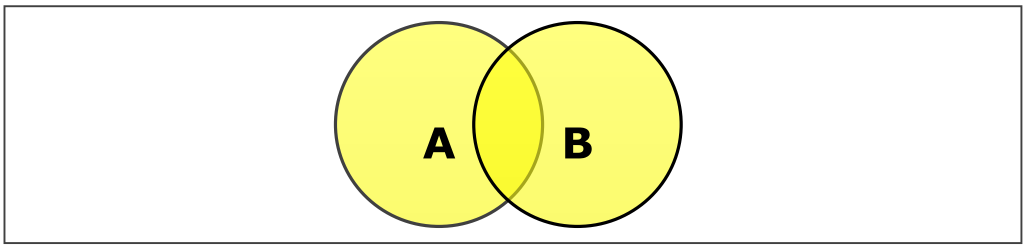


Figure: Venn Diagram of A ∪ B

Set Intersection

The intersection of sets A and B (denoted by A ∩ B) is the set of elements which are in both A and B. Hence, A∩B = {x | x ∈A AND x ∈B}.

Example: If A = {11, 12, 13} and B = {13, 14, 15}, then A∩B = {13}.

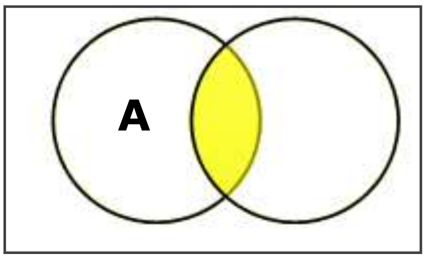


Figure: Venn Diagram of A ∩ B

Set Difference/ Relative Complement

The set difference of sets A and B (denoted by A–B) is the set of elements which are only in A but not in B. Hence, A−B = {x | x ∈A AND x ∉B}.

Example: If A = {10, 11, 12, 13} and B = {13, 14, 15}, then (A−B) = {10, 11, 12} and (B−A) = {14,15}. Here, we can see (A−B) ≠ (B−A)

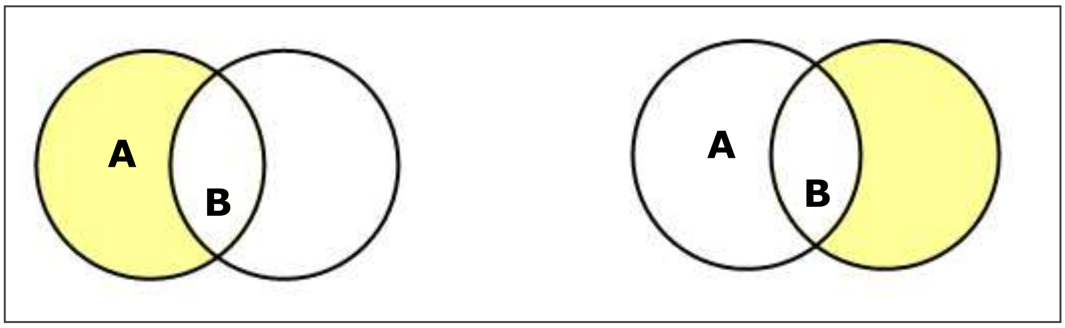


Figure: Venn Diagram of A – B and B – A

Complement of a Set

The complement of a set A (denoted by A') is the set of elements which are not in set A. Hence, A' = {x | x ∉A}.

More specifically, A'= (U–A) where U is a universal set which contains all objects.

Example: If A ={x | x belongs to set of odd integers} then A' ={y | y does not belong to set of odd integers}

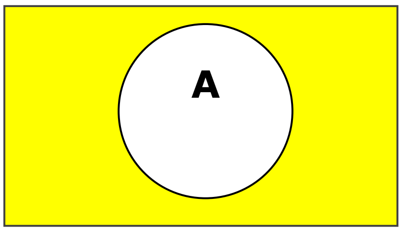


Figure: Venn Diagram of A'

Cartesian Product / Cross Product

The Cartesian product of n number of sets A_1, A_2.....A_n, denoted as $A_1 \times A_2 \times ..... \times A_n$, can be defined as all possible ordered pairs (x_1,x_2,....x_n) where $x_1 \in A_1$, $x_2 \in A_2$, ... $x_n \in A_n$

Example: If we take two sets A= {a, b} and B= {1, 2},

The Cartesian product of A and B is written as: A×B= {(a, 1), (a, 2), (b, 1), (b, 2)}

The Cartesian product of B and A is written as: B×A= {(1, a), (1, b), (2, a), (2, b)}

Power Set

Power set of a set S is the set of all subsets of S including the empty set. The cardinality of a power set of a set S of cardinality n is 2^n. Power set is denoted as P(S).

Example:

For a set S = {a, b, c, d} let us calculate the subsets:

- Subsets with 0 elements: {∅} (the empty set)
- Subsets with 1 element: {a}, {b}, {c}, {d}
- Subsets with 2 elements: {a,b}, {a,c}, {a,d}, {b,c}, {b,d},{c,d}
- Subsets with 3 elements: {a,b,c},{a,b,d},{a,c,d},{b,c,d}
- Subsets with 4 elements: {a,b,c,d}

Hence, P(S) =

{ {∅},{a}, {b}, {c}, {d},{a,b}, {a,c}, {a,d}, {b,c},
{b,d},{c,d},{a,b,c},{a,b,d},{a,c,d},{b,c,d},{a,b,c,d} **}**

$| P(S) | = 2^4 = 16$

Note: The power set of an empty set is also an empty set.
$| P (\{\emptyset\}) | = 2^0 = 1$

Partitioning of a Set

Partition of a set, say *S*, is a collection of *n* disjoint subsets, say $P_1, P_2, \ldots\ldots P_n$, that satisfies the following three conditions:

- P_i does not contain the empty set.
  $[\ P_i \neq \{\emptyset\}$ for all $0 < i \leq n]$
- The union of the subsets must equal the entire original set.
  $[P_1 \cup P_2 \cup \ldots\ldots \cup P_n = S]$
- The intersection of any two distinct sets is empty.
  $[P_a \cap P_b = \{\emptyset\}$, for $a \neq b$ where $n \geq a, b \geq 0\ ]$

Example

Let S = {a, b, c, d, e, f, g, h}

One probable partitioning is {a}, {b, c, d}, {e, f, g,h}

Another probable partitioning is {a,b}, { c, d}, {e, f, g,h}

Bell Numbers

Bell numbers give the count of the number of ways to partition a set. They are denoted by B_n where n is the cardinality of the set.

Example:

Let S = { 1, 2, 3}, n = |S| = 3

The alternate partitions are:

1. ∅, {1, 2, 3}
2. {1}, {2, 3}
3. {1, 2}, {3}
4. {1, 3}, {2}
5. {1}, {2},{3}

Hence B_3 = 5

3. Relations

Whenever sets are being discussed, the relationship between the elements of the sets is the next thing that comes up. **Relations** may exist between objects of the same set or between objects of two or more sets.

Definition and Properties

A binary relation R from set x to y (written as xRy or R(x,y)) is a subset of the Cartesian product x × y. If the ordered pair of G is reversed, the relation also changes.

Generally an n-ary relation R between sets A_1, ... , and A_n is a subset of the n-ary product $A_1 \times \ldots \times A_n$. The minimum cardinality of a relation R is Zero and maximum is n^2 in this case.

A binary relation R on a single set A is a subset of A × A.

For two distinct sets, A and B, having cardinalities *m* and *n* respectively, the maximum cardinality of a relation R from A to B is *mn*.

Domain and Range

If there are two sets A and B, and relation R have order pair (x, y), then:

- The **domain** of R, Dom(R), is the set { x | (x, y) ∈ R for some y in B }
- The **range** of R, Ran(R), is the set { y | (x, y) ∈ R for some x in A }

Examples

Let, A = {1,2,9} and B = {1,3,7}

- Case 1: If relation R is 'equal to' then R = {(1, 1), (3, 3)}

  Dom(R) = { 1, 3}, Ran(R) = { 1, 3}
- Case 2: If relation R is 'less than' then R = {(1, 3), (1, 7), (2, 3), (2, 7)}

  Dom(R) = { 1, 2}, Ran(R) = { 3, 7}
- Case 3: If relation R is 'greater than' then R = {(2, 1), (9, 1), (9, 3), (9, 7)}

  Dom(R) = { 2, 9}, Ran(R) = { 1, 3, 7}

Representation of Relations using Graph

A relation can be represented using a directed graph.

The number of vertices in the graph is equal to the number of elements in the set from which the relation has been defined. For each ordered pair (x, y) in the relation R, there will be a directed edge from the vertex 'x' to vertex 'y'. If there is an ordered pair (x, x), there will be self- loop on vertex 'x'.

Suppose, there is a relation R = {(1, 1), (1,2), (3, 2)} on set S = {1,2,3}, it can be represented by the following graph:

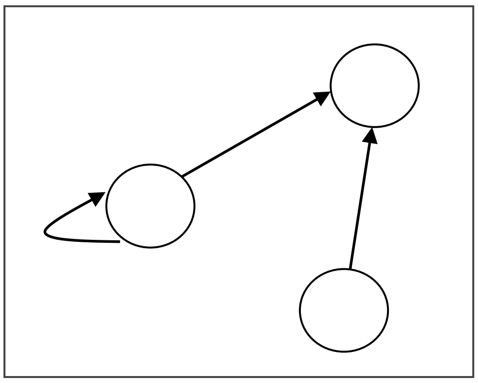


Figure: Representation of relation by directed graph

Types of Relations

1. The **Empty Relation** between sets X and Y, or on E, is the empty set ∅
2. The **Full Relation** between sets X and Y is the set X×Y
3. The **Identity Relation** on set X is the set {(x,x) | x ∈ X}
4. The Inverse Relation R' of a relation R is defined as: R'= {(b,a) | (a,b) ∈R}

   Example: If R = {(1, 2), (2,3)} then R' will be {(2,1), (3,2)}

5. A relation R on set A is called **Reflexive** if ∀a∈A is related to a (aRa holds).

   Example: The relation R = {(a,a), (b,b)} on set X={a,b} is reflexive

6. A relation R on set A is called **Irreflexive** if no a∈A is related to a (aRa does not hold).

   Example: The relation R = {(a,b), (b,a)} on set X={a,b} is irreflexive

7. A relation R on set A is called **Symmetric** if xRy implies yRx, ∀x∈A and ∀y∈A.

   Example: The relation R = {(1, 2), (2, 1), (3, 2), (2, 3)} on set A={1, 2, 3} is symmetric.

8. A relation R on set A is called **Anti-Symmetric** if xRy and yRx implies

   x=y ∀x ∈ A and ∀y ∈ A.

   Example: The relation R = { (x,y) ∈ N | x ≤ y } is anti-symmetric since x ≤ y and y ≤ x implies x = y.

9. A relation R on set A is called **Transitive** if xRy and yRz implies xRz, ∀x,y,z ∈ A.

   Example: The relation R = {(1, 2), (2, 3), (1, 3)} on set A= {1, 2, 3} is transitive.

10. A relation is an **Equivalence Relation** if it is reflexive, symmetric, and transitive.

Example: The relation R = {(1, 1), (2, 2), (3, 3), (1, 2),(2,1), (2,3), (3,2), (1,3), (3,1)} on set A= {1, 2, 3} is an equivalence relation since it is reflexive, symmetric, and transitive.

4. Functions

A **Function** assigns to each element of a set, exactly one element of a related set. Functions find their application in various fields like representation of the computational complexity of algorithms, counting objects, study of sequences and strings, to name a few. The third and final chapter of this part highlights the important aspects of functions.

Function – Definition

A function or mapping (Defined as $f: X \to Y$) is a relationship from elements of one set X to elements of another set Y (X and Y are non-empty sets). X is called Domain and Y is called Codomain of function 'f'.

Function 'f' is a relation on X and Y such that for each $x \in X$, there exists a unique $y \in Y$ such that $(x,y) \in R$. 'x' is called pre-image and 'y' is called image of function f.

A function can be one to one or many to one but not one to many.

Injective / One-to-one function

A function $f: A \to B$ is injective or one-to-one function if for every $b \in B$, there exists at most one $a \in A$ such that $f(s) = t$.

This means a function **f** is injective if $a_1 \neq a_2$ implies $f(a_1) \neq f(a_2)$.

Example

1. $f: N \to N, f(x) = 5x$ is injective.
2. $f: N \to N, f(x) = x^2$ is injective.
3. $f: R \to R, f(x) = x^2$ is not injective as $(-x)^2 = x^2$

Surjective / Onto function

A function $f: A \to B$ is surjective (onto) if the image of f equals its range. Equivalently, for every $b \in B$, there exists some $a \in A$ such that $f(a) = b$. This means that for any y in B, there exists some x in A such that $y = f(x)$.

Example

1. $f : N \to N, f(x) = x + 2$ is surjective.
2. $f : R \to R, f(x) = x^2$ is not surjective since we cannot find a real number whose square is negative.

Bijective / One-to-one Correspondent

A function $f: A \to B$ is bijective or one-to-one correspondent if and only if **f** is both injective and surjective.

Problem:

Prove that a function f: R→R defined by $f(x) = 2x - 3$ is a bijective function.

Explanation: We have to prove this function is both injective and surjective.

If $f(x_1) = f(x_2)$, then $2x_1 - 3 = 2x_2 - 3$ and it implies that $x_1 = x_2$.

Hence, f is **injective**.

Here, $2x - 3= y$

So, $x = (y+5)/3$ which belongs to R and $f(x) = y$.

Hence, f is **surjective**.

Since **f** is both **surjective** and **injective**, we can say **f** is **bijective**.

Inverse of a Function

The **inverse** of a one-to-one corresponding function $f : A \to B$, is the function $g : B \to A$, holding the following property:

$$f(x) = y \Leftrightarrow g(y) = x$$

The function f is called **invertible**, if its inverse function g exists.

Example:

- A function $f : Z \to Z$, $f(x) = x + 5$, is invertible since it has the inverse function $g : Z \to Z$, $g(x) = x - 5$
- A function $f : Z\to Z$, $f(x) = x^2$ is not invertible since this is not one-to-one as $(-x)^2 = x^2$.

Composition of Functions

Two functions f: A→B and g: B→C can be composed to give a composition g o f. This is a function from A to C defined by $(gof)(x) = g(f(x))$

Example

Let $f(x) = x + 2$ and $g(x) = 2x + 1$, find $(f \circ g)(x)$ and $(g \circ f)(x)$

Solution

$(f \circ g)(x) = f(g(x)) = f(2x + 1) = 2x + 1 + 2 = 2x + 3$

$(g \circ f)(x) = g(f(x)) = g(x + 2) = 2(x+2) + 1 = 2x + 5$

Hence, $(f \circ g)(x) \neq (g \circ f)(x)$

Some Facts about Composition

- If f and g are one-to-one then the function (g o f) is also one-to-one.
- If f and g are onto then the function (g o f) is also onto.

- Composition always holds associative property but does not hold commutative property.

Part 2: Mathematical Logic

5. Propositional Logic

The rules of mathematical logic specify methods of reasoning mathematical statements. Greek philosopher, Aristotle, was the pioneer of logical reasoning. Logical reasoning provides the theoretical base for many areas of mathematics and consequently computer science. It has many practical applications in computer science like design of computing machines, artificial intelligence, definition of data structures for programming languages etc.

Propositional Logic is concerned with statements to which the truth values, "true" and "false", can be assigned. The purpose is to analyze these statements either individually or in a composite manner.

Prepositional Logic – Definition

A proposition is a collection of declarative statements that has either a truth value "true" or a truth value "false". A propositional consists of propositional variables and connectives. We denote the propositional variables by capital letters (A, B, etc). The connectives connect the propositional variables.

Some examples of Propositions are given below:

- "Man is Mortal", it returns truth value "TRUE"
- "12 + 9 = 3 – 2", it returns truth value "FALSE"

The following is not a Proposition:

- "A is less than 2". It is because unless we give a specific value of A, we cannot say whether the statement is true or false.

Connectives

In propositional logic generally we use five connectives which are:

- OR (V)
- AND (Λ)
- Negation/ NOT (¬)
- Implication / if-then (→)
- If and only if (⇔).

OR (V): The OR operation of two propositions A and B (written as A V B) is true if at least any of the propositional variable A or B is true.

The truth table is as follows:

A	**B**	**A V B**
True	True	True
True	False	True
False	True	True
False	False	False

AND (∧): The AND operation of two propositions A and B (written as A ∧ B) is true if both the propositional variable A and B is true.

The truth table is as follows:

A	**B**	**A ∧ B**
True	True	True
True	False	False
False	True	False
False	False	False

Negation (¬): The negation of a proposition A (written as ¬A) is false when A is true and is true when A is false.

The truth table is as follows:

A	**¬A**
True	False
False	True

Implication / if-then (→): An implication A →B is the proposition "if A, then B". It is false if A is true and B is false. The rest cases are true.

The truth table is as follows:

A	**B**	**A → B**
True	True	True
True	False	False
False	True	True
False	False	True

If and only if (⇔): A ⇔B is bi-conditional logical connective which is true when p and q are same, i.e. both are false or both are true.

The truth table is as follows:

A	**B**	**A ⇔ B**
True	True	True
True	False	False
False	True	False
False	False	True

Tautologies

A Tautology is a formula which is always true for every value of its propositional variables.

Example: Prove [(A → B) Λ A] →B is a tautology

The truth table is as follows:

A	**B**	**A → B**	**(A → B) Λ A**	**[(A → B) Λ A] →B**
True	True	True	True	True
True	False	False	False	True
False	True	True	False	True
False	False	True	False	True

As we can see every value of [(A → B) Λ A] →B is "True", it is a tautology.

Contradictions

A Contradiction is a formula which is always false for every value of its propositional variables.

Example: Prove (A V B) Λ [(¬A) Λ (¬B)] is a contradiction

The truth table is as follows:

A	**B**	**A V B**	**¬A**	**¬B**	**(¬A) Λ (¬B)**	**(A V B) Λ [(¬A) Λ (¬B)]**
True	True	True	False	False	False	False
True	False	True	False	True	False	False
False	True	True	True	False	False	False
False	False	False	True	True	True	False

As we can see every value of (A V B) Λ [(¬A) Λ (¬B)] is "False", it is a contradiction.

Contingency

A Contingency is a formula which has both some true and some false values for every value of its propositional variables.

Example: Prove (A V B) Λ (¬A) a contingency

The truth table is as follows:

A	**B**	**A V B**	**¬A**	**(A V B) Λ (¬A)**
True	True	True	False	False
True	False	True	False	False
False	True	True	True	True
False	False	False	True	False

As we can see every value of (A V B) Λ (¬A) has both "True" and "False", it is a contingency.

Propositional Equivalences

Two statements X and Y are logically equivalent if any of the following two conditions hold:

- The truth tables of each statement have the same truth values.
- The bi-conditional statement X ⇔ Y is a tautology.

Example: Prove ¬ (A V B) and [(¬A) Λ (¬B)] are equivalent

Testing by 1st method (Matching truth table):

A	**B**	**A V B**	**¬ (A V B)**	**¬A**	**¬B**	**[(¬A) Λ (¬B)]**
True	True	True	False	False	False	False
True	False	True	False	False	True	False
False	True	True	False	True	False	False
False	False	False	True	True	True	True

Here, we can see the truth values of ¬ (A V B) and [(¬A) Λ (¬B)] are same, hence the statements are equivalent.

Testing by 2nd method (Bi-conditionality):

A	**B**	**¬ (A V B)**	**[(¬A) Λ (¬B)]**	**[¬ (A V B)] ⇔[(¬A) Λ (¬B)]**
True	True	False	False	True
True	False	False	False	True
False	True	False	False	True
False	False	True	True	True

As [¬ (A V B)] ⇔ [(¬A) Λ (¬B)] is a tautology, the statements are equivalent.

Inverse, Converse, and Contra-positive

Implication / if-then (→) is also called a conditional statement. It has two parts-

- Hypothesis , p
- Conclusion , q

As mentioned earlier, it is denoted as p → q.

Example of Conditional Statement: "If you do your homework, you will not be punished." Here, "you do your homework" is the hypothesis, p, and "you will not be punished" is the conclusion, q.

Inverse: An inverse of the conditional statement is the negation of both the hypothesis and the conclusion. If the statement is "If p, then q", the inverse will be "If not *p*, then not *q*". Thus the inverse of p→q is ¬p→¬q.

> **Example** : The inverse of "If you do your homework, you will not be punished" is "If you do not do your homework, you will be punished."

Converse: The converse of the conditional statement is computed by interchanging the hypothesis and the conclusion. If the statement is "If p, then q", the converse will be "If *q*, then *p*". The converse of p→q is q→p.

> **Example** : The converse of "If you do your homework, you will not be punished" is "If you will not be punished, you do not do your homework".

Contra-positive: The contra-positive of the conditional is computed by interchanging the hypothesis and the conclusion of the inverse statement. If the statement is "If p, then q", the contra-positive will be "If not *q*, then not *p*". The contra-positive of p→q is ¬q→¬p.

> **Example** : The Contra-positive of " If you do your homework, you will not be punished" is "If you are not punished, then you do not do your homework".

Duality Principle

Duality principle states that for any true statement, the dual statement obtained by interchanging unions into intersections (and vice versa) and interchanging Universal set into Null set (and vice versa) is also true. If dual of any statement is the statement itself, it is said **self-dual** statement.

Example: The dual of (A ∩ B) ∪ C is (A∪ B) ∩ C

Normal Forms

We can convert any proposition in two normal forms:

- Conjunctive normal form

- Disjunctive normal form

Conjunctive Normal Form

A compound statement is in conjunctive normal form if it is obtained by operating AND among variables (negation of variables included) connected with ORs. In terms of set operations, it is a compound statement obtained by Intersection among variables connected with Unions.

Examples

- (A V B) Λ (A V C) Λ (B V C V D)
- (P ∪Q) ∩ (Q ∪ R)

Disjunctive Normal Form

A compound statement is in conjunctive normal form if it is obtained by operating OR among variables (negation of variables included) connected with ANDs. In terms of set operations, it is a compound statement obtained by Union among variables connected with Intersections.

Examples

- (A Λ B) V (A Λ C) V (B Λ C Λ D)
- (P ∩ Q) ∪ (Q ∩ R)

6. Predicate Logic

Predicate Logic deals with predicates, which are propositions containing variables.

Predicate Logic – Definition

A predicate is an expression of one or more variables defined on some specific domain. A predicate with variables can be made a proposition by either assigning a value to the variable or by quantifying the variable.

The following are some examples of predicates:

- Let E(x, y) denote "x = y"
- Let X(a , b, c) denote "a + b + c = 0"
- Let M(x, y) denote "x is married to y"

Well Formed Formula

Well Formed Formula (wff) is a predicate holding any of the following -

- All propositional constants and propositional variables are wffs
- If x is a variable and Y is a wff, ∀x Y and ∃x Y are also wff
- Truth value and false values are wffs
- Each atomic formula is a wff
- All connectives connecting wffs are wffs

Quantifiers

The variable of predicates is quantified by quantifiers. There are two types of quantifier in predicate logic: Universal Quantifier and Existential Quantifier.

Universal Quantifier

Universal quantifier states that the statements within its scope are true for every value of the specific variable. It is denoted by the symbol ∀.

∀x P(x) is read as for every value of x, P(x) is true.

Example: "Man is mortal" can be transformed into the propositional form ∀x P(x) where P(x) is the predicate which denotes x is mortal and the universe of discourse is all men.

Existential Quantifier

Existential quantifier states that the statements within its scope are true for some values of the specific variable. It is denoted by the symbol ∃.

∃x P(x) is read as for some values of x, P(x) is true.

Example: "Some people are dishonest" can be transformed into the propositional form ∃x P(x) where P(x) is the predicate which denotes x is dishonest and the universe of discourse is some people.

Nested Quantifiers

If we use a quantifier that appears within the scope of another quantifier, it is called nested quantifier.

Examples

- ∀a ∃b P (x, y) where P (a, b) denotes a + b=0
- ∀a ∀b ∀c P (a, b, c) where P (a, b) denotes a + (b+c) = (a+b) +c

Note: ∀a ∃b P (x, y) ≠ ∃a ∀b P (x, y)

7. Rules of Inference

To deduce new statements from the statements whose truth that we already know, **Rules of Inference** are used.

What are Rules of Inference for?

Mathematical logic is often used for logical proofs. Proofs are valid arguments that determine the truth values of mathematical statements.

An argument is a sequence of statements. The last statement is the conclusion and all its preceding statements are called premises (or hypothesis). The symbol "∴", (read therefore) is placed before the conclusion. A valid argument is one where the conclusion follows from the truth values of the premises.

Rules of Inference provide the templates or guidelines for constructing valid arguments from the statements that we already have.

Table of Rules of Inference

Rule of Inference	Name	Rule of Inference	Name
P ---------- ∴ P V Q	Addition	P V Q ¬P ---------- ∴ Q	Disjunctive Syllogism
P Q ---------- ∴ P Λ Q	Conjunction	P → Q Q → R ---------- ∴ P → R	Hypothetical Syllogism
P Λ Q ---------- ∴ P	Simplification	(P → Q) Λ (R → S) P V R ---------- ∴ Q V S	Constructive Dilemma
P→Q P ---------- ∴ Q	Modus ponens	(P → Q) Λ (R → S) ¬Q V ¬S ---------- ∴ ¬P V ¬R	Destructive Dilemma
P→Q ¬Q ---------- ∴ ¬P	Modus Tollens		

Addition

If P is a premise, we can use Addiction rule to derive P ∨ Q.

$$\begin{array}{c} P \\ \text{----------} \\ \therefore P \lor Q \end{array}$$

Example

Let P be the proposition, "He studies very hard" is true

Therefore: "Either he studies very hard Or he is a very bad student." Here Q is the proposition "he is a very bad student".

Conjunction

If P and Q are two premises, we can use Conjunction rule to derive P ∧ Q.

$$\begin{array}{c} P \\ Q \\ \text{----------} \\ \therefore P \land Q \end{array}$$

Example

Let P: "He studies very hard"

Let Q: "He is the best boy in the class"

Therefore: "He studies very hard and he is the best boy in the class"

Simplification

If P ∧ Q is a premise, we can use Simplification rule to derive P.

$$\begin{array}{c} P \land Q \\ \text{----------} \\ \therefore P \end{array}$$

Example

"He studies very hard and he is the best boy in the class", P ∧ Q

Therefore: "He studies very hard"

Modus Ponens

If P and P→Q are two premises, we can use Modus Ponens to derive Q.

P→Q
P

∴ Q

Example

"If you have a password, then you can log on to facebook", P→Q

"You have a password", P

Therefore: "You can log on to facebook"

Modus Tollens

If P→Q and ¬Q are two premises, we can use Modus Tollens to derive ¬P.

P→Q
¬Q

∴ ¬P

Example

"If you have a password, then you can log on to facebook", P→Q

"You cannot log on to facebook", ¬Q

Therefore: "You do not have a password "

Disjunctive Syllogism

If ¬P and P V Q are two premises, we can use Disjunctive Syllogism to derive Q.

¬P
P V Q

∴ Q

Example

"The ice cream is not vanilla flavored", ¬P

"The ice cream is either vanilla flavored or chocolate flavored", P V Q

Therefore: "The ice cream is chocolate flavored"

Hypothetical Syllogism

If P → Q and Q → R are two premises, we can use Hypothetical Syllogism to derive P → R

P → Q
Q → R

$\therefore P \rightarrow R$

Example

"If it rains, I shall not go to school", $P \rightarrow Q$

"If I don't go to school, I won't need to do homework", $Q \rightarrow R$

Therefore: "If it rains, I won't need to do homework"

Constructive Dilemma

If $(P \rightarrow Q) \land (R \rightarrow S)$ and $P \lor R$ are two premises, we can use constructive dilemma to derive $Q \lor S$.

$(P \rightarrow Q) \land (R \rightarrow S)$
$P \lor R$

$\therefore Q \lor S$

Example

"If it rains, I will take a leave", $(P \rightarrow Q)$

"If it is hot outside, I will go for a shower", $(R \rightarrow S)$

"Either it will rain or it is hot outside", $P \lor R$

Therefore: "I will take a leave or I will go for a shower"

Destructive Dilemma

If $(P \rightarrow Q) \land (R \rightarrow S)$ and $\neg Q \lor \neg S$ are two premises, we can use destructive dilemma to derive $P \lor R$.

$(P \rightarrow Q) \land (R \rightarrow S)$
$\neg Q \lor \neg S$

$\therefore \neg P \lor \neg R$

Example

"If it rains, I will take a leave", $(P \rightarrow Q)$

"If it is hot outside, I will go for a shower", $(R \rightarrow S)$

"Either I will not take a leave or I will not go for a shower", $\neg Q \lor \neg S$

Therefore: "Either it rains or it is hot outside"

Part 3: Group Theory

8. Operators and Postulates

Group Theory is a branch of mathematics and abstract algebra that defines an algebraic structure named as **group**. Generally, a group comprises of a set of elements and an operation over any two elements on that set to form a third element also in that set.

In 1854, Arthur Cayley, the British Mathematician, gave the modern definition of group for the first time:

> *"A set of symbols all of them different, and such that the product of any two of them (no matter in what order), or the product of any one of them into itself, belongs to the set, is said to be a group. These symbols are not in general convertible [commutative], but are associative."*

In this chapter, we will know about **operators and postulates** that form the basics of set theory, group theory and Boolean algebra.

Any set of elements in a mathematical system may be defined with a set of operators and a number of postulates.

A **binary operator** defined on a set of elements is a rule that assigns to each pair of elements a unique element from that set. For example, given the set A={1,2,3,4,5}, we can say ⊗ is a binary operator for the operation $c = a \otimes b$, if it specifies a rule for finding c for the pair of (a,b), such that a,b,c ∈ A.

The **postulates** of a mathematical system form the basic assumptions from which rules can be deduced. The postulates are:

Closure

A set is closed with respect to a binary operator if for every pair of elements in the set, the operator finds a unique element from that set.

Example: Let A = { 0, 1, 2, 3, 4, 5, …………. }

This set is closed under binary operator into (*), because for the operation c = a * b, for any a, b ∈ A, the product c ∈ A.

The set is not closed under binary operator divide (÷), because, for the operation c = a ÷ b, for any a, b ∈ A, the product c may not be in the set A. If a = 7, b = 2, then c = 3.5. Here a,b ∈ A but c ∉ A.

Associative Laws

A binary operator ⊗ on a set A is associative when it holds the following property:

$(x \otimes y) \otimes z = x \otimes (y \otimes z)$, where x, y, z ∈ A

Example: Let A = { 1, 2, 3, 4 }

The operator plus (+) is associative because for any three elements, x,y,z ∈ A, the property (x + y) + z = x + (y + z) holds.

The operator minus (-) is not associative since

$(x - y) - z \neq x - (y - z)$

Commutative Laws

A binary operator $\otimes$ on a set A is commutative when it holds the following property:

$$x \otimes y = y \otimes x, \text{ where } x, y \in A$$

Example: Let A = { 1, 2, 3, 4 }

The operator plus (+) is commutative because for any two elements, x,y ∈ A, the property x + y = y + x holds.

The operator minus (-) is not associative since

$$x - y \neq y - x$$

Distributive Laws

Two binary operators $\otimes$ and $\circledast$ on a set A, are distributive over operator $\circledast$ when the following property holds:

$$x \otimes (y \circledast z) = (x \otimes y) \circledast (x \otimes z), \text{ where } x, y, z \in A$$

Example: Let A = { 1, 2, 3, 4 }

The operators into (*) and plus (+) are distributive over operator + because for any three elements, x,y,z ∈ A, the property x * (y + z) = (x * y) + (x * z) holds.

However, these operators are not distributive over * since

$$x + (y * z) \neq (x + y) * (x + z)$$

Identity Element

A set A has an identity element with respect to a binary operation $\otimes$ on A, if there exists an element e ∈ A, such that the following property holds:

$$e \otimes x = x \otimes e, \text{ where } x \in A$$

Example: Let Z = { 0, 1, 2, 3, 4, 5, ……………….. }

The element 1 is an identity element with respect to operation ***** since for any element x ∈ Z,

$1 * x = x * 1$

On the other hand, there is no identity element for the operation minus (-)

Inverse

If a set A has an identity element e with respect to a binary operator $\otimes$, it is said to have an inverse whenever for every element x ∈ A, there exists another element y ∈ A, such that the following property holds:

$$x \otimes y = e$$

Example: Let A = { ………….. -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, ………….. }

Given the operation plus (+) and e = 0, the inverse of any element x is (-x) since x + (-x) = 0

De Morgan's Law

De Morgan's Laws gives a pair of transformations between union and intersection of two (or more) sets in terms of their complements. The laws are:

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

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

Example: Let A = { 1, 2, 3, 4}, B = {1, 3, 5, 7}, and

Universal set U = { 1, 2, 3, ………, 9, 10 }

$A' = \{5, 6, 7, 8, 9, 10\}$

$B' = \{2, 4, 6, 8, 9, 10\}$

$A \cup B = \{1, 2, 3, 4, 5, 7\}$

$A \cap B = \{1, 3\}$

$(A \cup B)' = \{6, 8, 9, 10\}$

$A' \cap B' = \{6, 8, 9, 10\}$

Thus, we see that $(A \cup B)' = A' \cap B'$

$(A \cap B)' = \{2, 4, 5, 6, 7, 8, 9, 10\}$

$A' \cup B' = \{2, 4, 5, 6, 7, 8, 9, 10\}$

Thus, we see that $(A \cap B)' = A' \cup B'$

9. Group Theory

Semigroup

A finite or infinite set ‘S’ with a binary operation ‘0’ (Composition) is called semigroup if it holds following two conditions simultaneously:

- **Closure**: For every pair (a, b) ∈ S, (a 0 b) has to be present in the set S.
- **Associative**: For every element a, b, c ∈ S, (a 0 b) 0 c = a 0 (b 0 c) must hold.

Example:

The set of positive integers (excluding zero) with addition operation is a semigroup. For example, S = {1, 2, 3,...}

Here closure property holds as for every pair (a, b) ∈ S, (a + b) is present in the set S. For example, 1 +2 =3 ∈ S]

Associative property also holds for every element a, b, c ∈S, (a + b) + c = a + (b + c). For example, (1 +2) +3=1+ (2+3)=5

Monoid

A monoid is a semigroup with an identity element. The identity element (denoted by **e** or **E**) of a set S is an element such that (a 0 **e**) = a, for every element a ∈ S. An identity element is also called a **unit element**. So, a monoid holds three properties simultaneously: **Closure**, **Associative**, **Identity element**.

Example

The set of positive integers (excluding zero) with multiplication operation is a monoid. S = {1, 2, 3,...}

Here closure property holds as for every pair (a, b) ∈ S, (a × b) is present in the set S. [For example, 1 × 2 = 2 ∈ S and so on]

Associative property also holds for every element a, b, c ∈S, (a × b) × c = a × (b × c) [For example, (1 × 2) × 3 = 1 × (2 × 3) = 6 and so on]

Identity property also holds for every element a ∈ S, (a × e) = a [For example, (2 ×1) = 2, (3 ×1) =3 and so on]. Here identity element is 1.

Group

A group is a monoid with an inverse element. The inverse element (denoted by I) of a set S is an element such that (a 0 I) = (I 0 a) =a, for each element a ∈ S. So, a group holds four properties simultaneously - i) Closure, ii) Associative, iii) Identity element, iv) Inverse element. The order of a group G is the number of elements in G and the order of an element in a group is the least positive integer n such that a^n is the identity element of that group G.

Examples

The set of N×N non-singular matrices form a group under matrix multiplication operation.

The product of two N×N non-singular matrices is also an N×N non-singular matrix which holds closure property.

Matrix multiplication itself is associative. Hence, associative property holds.

The set of N×N non-singular matrices contains the identity matrix holding the identity element property.

As all the matrices are non-singular they all have inverse elements which are also non-singular matrices. Hence, inverse property also holds.

Abelian Group

An abelian group G is a group for which the element pair (a,b) ∈ G always holds commutative law. So, a group holds five properties simultaneously - i) Closure, ii) Associative, iii) Identity element, iv) Inverse element, v) Commutative.

Example

The set of positive integers (including zero) with addition operation is an abelian group. G = {0, 1, 2, 3,…}

Here closure property holds as for every pair (a, b) ∈ S, (a + b) is present in the set S. [For example, 1 +2 =2 ∈ S and so on]

Associative property also holds for every element a, b, c ∈S, (a + b) + c = a + (b + c) [For example, (1 +2) +3=1 + (2 +3) =6 and so on]

Identity property also holds for every element a ∈ S, (a × e) = a [For example, (2 ×1) =2, (3 ×1) =3 and so on]. Here, identity element is 1.

Commutative property also holds for every element a ∈S, (a × b) = (b × a) [For example, (2 ×3) = (3 ×2) =3 and so on]

Cyclic Group and Subgroup

A **cyclic group** is a group that can be generated by a single element. Every element of a cyclic group is a power of some specific element which is called a generator. A cyclic group can be generated by a generator ‘g’, such that every other element of the group can be written as a power of the generator ‘g’.

Example

The set of complex numbers {1,-1, i, -i} under multiplication operation is a cyclic group.

There are two generators: i and –i as $i^1=i$, $i^2=-1$, $i^3=-i$, $i^4=1$ and also $(-i)^1=-i$, $(-i)^2=-1$, $(-i)^3=i$, $(-i)^4=1$ which covers all the elements of the group. Hence, it is a cyclic group.

Note: A **cyclic group** is always an abelian group but not every abelian group is a cyclic group. The rational numbers under addition is not cyclic but is abelian.

A **subgroup** H is a subset of a group G (denoted by $H \leq G$) if it satisfies the four properties simultaneously: **Closure**, **Associative**, **Identity element**, and **Inverse**.

A subgroup H of a group G that does not include the whole group G is called a proper subgroup (Denoted by $H<G$). A subgroup of a cyclic group is cyclic and a abelian subgroup is also abelian.

Example

Let a group G = {1, i, -1, -i}

Then some subgroups are H_1= {1}, H_2= {1,-1},

This is not a subgroup: H_3= {1, i} because that $(i)^{-1} = -i$ is not in H_3

Partially Ordered Set (POSET)

A partially ordered set consists of a set with a binary relation which is reflexive, anti-symmetric and transitive. "Partially ordered set" is abbreviated as POSET.

Examples

1. The set of real numbers under binary operation less than or equal to (≤) is a poset.

   Let the set S = {1, 2, 3} and the operation is ≤

   The relations will be {(1, 1), (2, 2), (3, 3), (1, 2), (1, 3), (2, 3)}

   This relation R is reflexive as {(1, 1), (2, 2), (3, 3)} ∈ R

   This relation R is anti-symmetric, as

   {(1, 2), (1, 3), (2, 3)} ∈ R and {(2,1), (3,1), (3,2)} ∉ R

   This relation R is also transitive as {(1,2), (2,3), (1,3)} ∈ R.

   Hence, it is a **poset**.

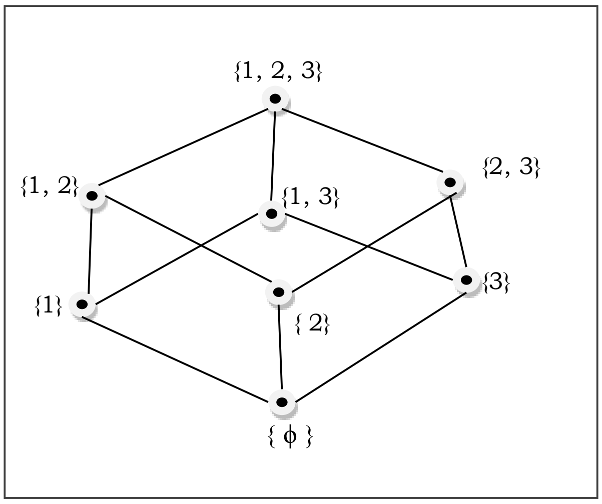
2. The vertex set of a directed acyclic graph under the operation 'reachability' is a poset.

Hasse Diagram

The Hasse diagram of a poset is the directed graph whose vertices are the element of that poset and the arcs covers the pairs (x, y) in the poset. If in the poset x<y, then the point x appears lower than the point y in the Hasse diagram. If x<y<z in the poset, then the arrow is not shown between x and z as it is implicit.

Example

The poset of subsets of {1, 2, 3} = {ϕ, {1}, {2}, {3}, {1, 2}, {1, 3}, {2, 3}, {1, 2, 3}} is shown by the following Hasse diagram:



Linearly Ordered Set

A Linearly ordered set or Total ordered set is a partial order set in which every pair of element is comparable. The elements a, b ∈S are said to be comparable if either a ≤ b or b ≤ a holds. Trichotomy law defines this total ordered set. A totally ordered set can be defined as a distributive lattice having the property {a V b, a Λ b} = {a, b} for all values of a and b in set S.

Example

The powerset of {a, b} ordered by ⊆ is a totally ordered set as all the elements of the power set P= {φ, {a}, {b}, {a, b}} are comparable.

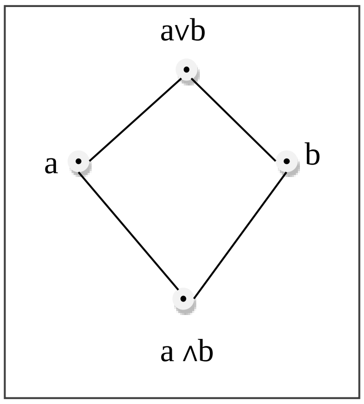
Example of non-total order set

A set S= {1, 2, 3, 4, 5, 6} under operation x divides y is not a total ordered set.

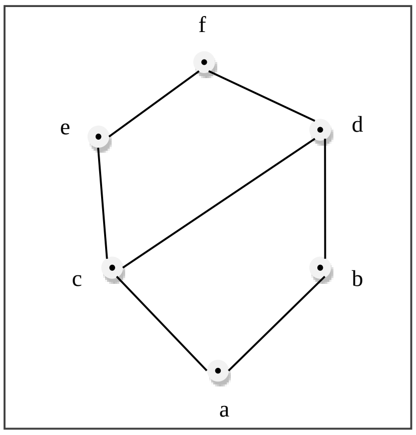
Here, for all (x, y) ∈ S, x | y have to hold but it is not true that 2 | 3, as 2 does not divide 3 or 3 does not divide 2. Hence, it is not a total ordered set.

Lattice

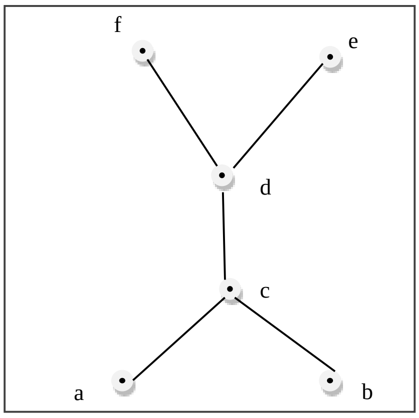
A lattice is a poset (L, ≤) for which every pair {a, b} ∈ L has a least upper bound (denoted by a V b) and a greatest lower bound (denoted by a Λ b). LUB ({a,b}) is called the join of a and b. GLB ({a,b}) is called the meet of a and b.



Example



This above figure is a lattice because for every pair {a, b} ∈ L, a GLB and a LUB exists.



This above figure is a not a lattice because GLB (a, b) and LUB (e, f) does not exist.

Some other lattices are discussed below:

Bounded Lattice

A lattice L becomes a bounded lattice if it has a greatest element 1 and a least element 0.

Complemented Lattice

A lattice L becomes a complemented lattice if it is a bounded lattice and if every element in the lattice has a complement. An element x has a complement x' if ∃x(x ∧x'=0 and x ∨ x' = 1)

Distributive Lattice

If a lattice satisfies the following two distribute properties, it is called a distributive lattice.

- a ∨ (b ∧ c) = (a ∨ b) ∧ (a ∨ c)
- a ∧ (b ∨ c) = (a ∧ b) ∨ (a ∧ c)

Modular Lattice

If a lattice satisfies the following property, it is called modular lattice.

a ∧(b ∨ (a ∧ d)) = (a ∧ b) ∨ (a ∧ d)

Properties of Lattices

Idempotent Properties

- a v a = a
- a ∧ a = a

Absorption Properties

- a v (a ∧ b) = a
- a ∧ (a v b) = a

Commutative Properties

- a v b = b v a
- a ∧ b = b ∧ a

Associative Properties

- a v (b v c)= (a v b) v c
- a ∧ (b ∧ c)= (a ∧ b) ∧ c

Dual of a Lattice

The dual of a lattice is obtained by interchanging the 'v' and '∧' operations.

Example

The dual of [a v (b ∧ c)] is [a ∧ (b v c)]

Part 4: Counting & Probability

10. Counting Theory

In daily lives, many a times one needs to find out the number of all possible outcomes for a series of events. For instance, in how many ways can a panel of judges comprising of 6 men and 4 women be chosen from among 50 men and 38 women? How many different 10 lettered PAN numbers can be generated such that the first five letters are capital alphabets, the next four are digits and the last is again a capital letter. For solving these problems, mathematical theory of counting are used. **Counting** mainly encompasses fundamental counting rule, the permutation rule, and the combination rule.

The Rules of Sum and Product

The **Rule of Sum** and **Rule of Product** are used to decompose difficult counting problems into simple problems.

- **The Rule of Sum:** If a sequence of tasks T_1, T_2, …, T_m can be done in w_1, w_2,… w_m ways respectively (the condition is that no tasks can be performed simultaneously), then the number of ways to do one of these tasks is $w_1 + w_2 + \ldots + w_m$. If we consider two tasks A and B which are disjoint (i.e. $A \cap B = \emptyset$), then mathematically $|A \cup B| = |A| + |B|$

- **The Rule of Product:** If a sequence of tasks T_1, T_2, …, T_m can be done in w_1, w_2,… w_m ways respectively and every task arrives after the occurrence of the previous task, then there are $w_1 \times w_2 \times \ldots \times w_m$ ways to perform the tasks. Mathematically, if a task B arrives after a task A, then $|A\times B| = |A|\times|B|$

Example

Question: A boy lives at X and wants to go to School at Z. From his home X he has to first reach Y and then Y to Z. He may go X to Y by either 3 bus routes or 2 train routes. From there, he can either choose 4 bus routes or 5 train routes to reach Z. How many ways are there to go from X to Z?

Solution: From X to Y, he can go in 3+2=5 ways (Rule of Sum). Thereafter, he can go Y to Z in 4+5 = 9 ways (Rule of Sum). Hence from X to Z he can go in 5×9 =45 ways (Rule of Product).

Permutations

A **permutation** is an arrangement of some elements in which order matters. In other words a Permutation is an ordered Combination of elements.

Examples

- From a set S ={x, y, z} by taking two at a time, all permutations are:
  xy, yx, xz, zx, yz, zy.

- We have to form a permutation of three digit numbers from a set of numbers S= {1, 2, 3}. Different three digit numbers will be formed when we arrange the digits. The permutation will be = 123, 132, 213, 231, 312, 321

Number of Permutations

The number of permutations of ‘n’ different things taken ‘r’ at a time is denoted by ^nP_r

$$^nP_r = \frac{n!}{(n-r)!}$$

where $n! = 1.2.3. \ldots\ldots (n-1).n$

Proof: Let there be ‘n’ different elements.

There are n number of ways to fill up the first place. After filling the first place (n-1) number of elements is left. Hence, there are (n-1) ways to fill up the second place. After filling the first and second place, (n-2) number of elements is left. Hence, there are (n-2) ways to fill up the third place. We can now generalize the number of ways to fill up r-th place as [n – (r–1)] = n–r+1

So, the total no. of ways to fill up from first place up to r-th-place:

$^nP_r = n\ (n-1)\ (n-2) \ldots\ldots (n-r+1)$

$= [n(n-1)(n-2) \ldots (n-r+1)]\ [(n-r)(n-r-1)\text{-----}3.2.1] / [(n-r)(n-r-1) .. 3.2.1]$

Hence,

$^nP_r = n!/(n-r)!$

Some important formulas of permutation

1. If there are n elements of which a_1 are alike of some kind, a_2 are alike of another kind; a_3 are alike of third kind and so on and a_r are of r^{th} kind, where $(a_1 + a_2 + \ldots a_r) = n$.
   Then, number of permutations of these n objects is = $n! / [\ (a_1!)\ (a_2!) \ldots\ldots (a_r!)]$.

2. Number of permutations of n distinct elements taking n elements at a time = $^nP_n = n!$

3. The number of permutations of n dissimilar elements taking r elements at a time, when x particular things always occupy definite places = $^{n-x}p_{r-x}$

4. The number of permutations of n dissimilar elements when r specified things always come together is: $r!\ (n-r+1)!$

5. The number of permutations of n dissimilar elements when r specified things never come together is: $n!-[r!\ (n-r+1)!]$

6. The number of circular permutations of n different elements taken x elements at time = $^nP_x /x$

7. The number of circular permutations of n different things = $^nP_n /n$

Some Problems

Problem 1: From a bunch of 6 different cards, how many ways we can permute it?

Solution: As we are taking 6 cards at a time from a deck of 6 cards, the permutation will be $^6P_6$ = 6! = 720

Problem 2: In how many ways can the letters of the word 'READER' be arranged?

Solution: There are 6 letters word (2 E, 1 A, 1D and 2R.) in the word 'READER'.

The permutation will be = 6! / [(2!) (1!)(1!)(2!)] = 180.

Problem 3: In how ways can the letters of the word 'ORANGE' be arranged so that the consonants occupy only the even positions?

Solution: There are 3 vowels and 3 consonants in the word 'ORANGE'. Number of ways of arranging the consonants among themselves= $^3P_3$ = 3! = 6. The remaining 3 vacant places will be filled up by 3 vowels in $^3P_3$ = 3! = 6 ways. Hence, the total number of permutation is $6\times6=36$

Combinations

A **combination** is selection of some given elements in which order does not matter.

The number of all combinations of n things, taken r at a time is:

$$^nC_r = \frac{n!}{r!\ \ (n-r)!}$$

Problem 1
Find the number of subsets of the set {1, 2, 3, 4, 5, 6} having 3 elements.

Solution
The cardinality of the set is 6 and we have to choose 3 elements from the set. Here, the ordering does not matter. Hence, the number of subsets will be $^6C_3=20$.

Problem 2
There are 6 men and 5 women in a room. In how many ways we can choose 3 men and 2 women from the room?

Solution
The number of ways to choose 3 men from 6 men is $^6C_3$ and the number of ways to choose 2 women from 5 women is $^5C_2$
Hence, the total number of ways is: $^6C_3 \times {}^5C_2=20\times10=200$

Problem 3
How many ways can you choose 3 distinct groups of 3 students from total 9 students?

Solution

Let us number the groups as 1, 2 and 3
For choosing 3 students for 1st group, the number of ways: $^9C_3$
The number of ways for choosing 3 students for 2nd group after choosing 1st group: $^6C_3$
The number of ways for choosing 3 students for 3rd group after choosing 1st and 2nd group: $^3C_3$
Hence, the total number of ways = $^9C_3 \times {}^6C_3 \times {}^3C_3 = 84\times20\times1 = 1680$

Pascal's Identity

Pascal's identity, first derived by Blaise Pascal in 19th century, states that the number of ways to choose k elements from n elements is equal to the summation of number of ways to choose (k-1) elements from (n-1) elements and the number of ways to choose k elements from n-1 elements.

Mathematically, for any positive integers k and n: $^nC_k = {}^{n-1}C_{k-1} + {}^{n-1}C_k$

Proof:

$$
\begin{aligned}
&{}^{n-1}C_{k-1} + {}^{n-1}C_k \\
&= \frac{(n-1)!}{(k-1)!\,(n-k)!} + \frac{(n-1)!}{k!\,(n-k-1)!} \\
&= (n-1)!\left(\frac{k}{k!\,(n-k)!} + \frac{n-k}{k!\,(n-k)!}\right) \\
&= (n-1)! \cdot \frac{n}{k!\,(n-k)!} \\
&= \frac{n!}{k!\,(n-k)!} \\
&= {}^nC_k
\end{aligned}
$$

Pigeonhole Principle

In 1834, German mathematician, Peter Gustav Lejeune Dirichlet, stated a principle which he called the drawer principle. Now, it is known as the pigeonhole principle.

Pigeonhole Principle states that if there are fewer pigeon holes than total number of pigeons and each pigeon is put in a pigeon hole, then there must be at least one pigeon hole with more than one pigeon. If n pigeons are put into m pigeonholes where n>m, there's a hole with more than one pigeon.

Examples

1. Ten men are in a room and they are taking part in handshakes. If each person shakes hands at least once and no man shakes the same man's hand more than once then two men took part in the same number of handshakes.

2. There must be at least two people in a class of 30 whose names start with the same alphabet.

The Inclusion-Exclusion principle

The **Inclusion-exclusion principle** computes the cardinal number of the union of multiple non-disjoint sets. For two sets A and B, the principle states:

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

For three sets A, B and C, the principle states: $|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$

The generalized formula:

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{1 \le i < j \le n} |A_i \cap A_j| + \sum_{1 \le i < j < k \le n} |A_i \cap A_j \cap A_k| - \ldots\ldots + (-1)^{n-1}|A_1 \cap \ldots \cap A_2|$$

Problem 1

How many integers from 1 to 50 are multiples of 2 or 3 but not both?

Solution

From 1 to 100, there are 50/2=25 numbers which are multiples of 2.

There are 50/3=16 numbers which are multiples of 3.

There are 50/6=8 numbers which are multiples of both 2 and 3.

So, $|A|=25$, $|B|=16$ and $|A \cap B|= 8$.

$|A \cup B| = |A| + |B| - |A \cap B| = 25 + 16 - 8 = 33$

Problem 2

In a group of 50 students 24 like cold drinks and 36 like hot drinks and each student likes at least one of the two drinks. How many like both coffee and tea?

Solution

Let X be the set of students who like cold drinks and Y be the set of people who like hot drinks.

So, $|X \cup Y| = 50$, $|X| = 24$, $|Y| = 36$

$|X \cap Y| = |X| + |Y| - |X \cup Y| = 24 + 36 - 50 = 60 - 50 = 10$

Hence, there are 10 students who like both tea and coffee.

11. Probability

Closely related to the concepts of counting is Probability. We often try to guess the results of games of chance, like card games, slot machines, and lotteries; i.e. we try to find the likelihood or probability that a particular result with be obtained.

Probability can be conceptualized as finding the chance of occurrence of an event. Mathematically, it is the study of random processes and their outcomes. The laws of probability have a wide applicability in a variety of fields like genetics, weather forecasting, opinion polls, stock markets etc.

Basic Concepts

Probability theory was invented in the 17th century by two French mathematicians, Blaise Pascal and Pierre de Fermat, who were dealing with mathematical problems regarding of chance.

Before proceeding to details of probability, let us get the concept of some definitions.

Random Experiment: An experiment in which all possible outcomes are known and the exact output cannot be predicted in advance is called a random experiment. Tossing a fair coin is an example of random experiment.

Sample Space: When we perform an experiment, then the set S of all possible outcomes is called the sample space. If we toss a coin, the sample space S = {H, T}

Event: Any subset of a sample space is called an event. After tossing a coin, getting Head on the top is an event.

The word "probability" means the chance of occurrence of a particular event. The best we can say is how likely they are to happen, using the idea of probability.

$$\text{Probability of occurence of an event} = \frac{\text{Total number of favourable outcome}}{\text{Total number of Outcomes}}$$

As the occurrence of any event varies between 0% and 100%, the probability varies between 0 and 1.

Steps to find the probability:

Step 1: Calculate all possible outcomes of the experiment.

Step 2: Calculate the number of favorable outcomes of the experiment.

Step 3: Apply the corresponding probability formula.

Tossing a Coin

If a coin is tossed, there are two possible outcomes: Heads (H) or Tails (T)

So, Total number of outcomes = 2

Hence, the probability of getting a Head (H) on top is ½ and the probability of getting a Tails (T) on top is ½

Throwing a Dice

When a dice is thrown, six possible outcomes can be on the top: 1, 2, 3, 4, 5, 6.

The probability of any one of the numbers is 1/6

The probability of getting even numbers is 3/6=1/3

The probability of getting odd numbers is 3/6=1/3

Taking Cards From a Deck

From a deck of 52 cards, if one card is picked find the probability of an ace being drawn and also find the probability of a diamond being drawn.

Total number of possible outcomes: 52

Outcomes of being an ace: 4

Probability of being an ace = 4/52 =1/13

Probability of being a diamond = 13/52 =1/4

Probability Axioms

- The probability of an event always varies from 0 to 1. $[0 \leq P(x) \leq 1]$
- For an impossible event the probability is 0 and for a certain event the probability is 1.
- If the occurrence of one event is not influenced by another event, they are called mutually exclusive or disjoint.

  If A_1, A_2....A_n are mutually exclusive/disjoint events, then
  $P(A_i \cap A_j) = \varphi$ for $i \neq j$ and $P(A_1 \cup A_2 \cup .... A_n) = P(A_1) + P(A_2) + ... P(A_n)$

Properties of Probability

1. If there are two events x and $\bar{x}$ which are complementary, then the probability of the complementary event is:

$$P(\bar{x}) = 1 - P(x)$$

2. For two non-disjoint events A and B, the probability of the union of two events:

$$P(A \cup B) = P(A) + P(B)$$

3. If an event A is a subset of another event B (i.e. $A \subset B$), then the probability of A is less than or equal to the probability of B. Hence, $A \subset B$ implies $P(A) \leq p(B)$

Conditional Probability

The conditional probability of an event B is the probability that the event will occur given an event A has already occurred. This is written as $P(B|A)$.

Mathematically: $P(B|A) = P(A \cap B) / P(A)$

If event A and B are mutually exclusive, then the conditional probability of event B after the event A will be the probability of event B that is P(B).

Problem 1

In a country 50% of all teenagers own a cycle and 30% of all teenagers own a bike and cycle. What is the probability that a teenager owns bike given that the teenager owns a cycle?

Solution

Let us assume A is the event of teenagers owning only a cycle and B is the event of teenagers owning only a bike.

So, $P(A) = 50/100 = 0.5$ and $P(A \cap B) = 30/100= 0.3$ from the given problem.

$P(B|A) = P(A \cap B) / P(A) = 0.3/0.5 = 0.6$

Hence, the probability that a teenager owns bike given that the teenager owns a cycle is 60%.

Problem 2

In a class, 50% of all students play cricket and 25% of all students play cricket and volleyball. What is the probability that a student plays volleyball given that the student plays cricket?

Solution

Let us assume A is the event of students playing only cricket and B is the event of students playing only volleyball.

So, $P(A) = 50/100=0.5$ and $P(A \cap B) = 25/100=0.25$ from the given problem.

$P(B|A) = P(A \cap B) / P(A) = 0.25/0.5 = 0.5$

Hence, the probability that a student plays volleyball given that the student plays cricket is 50%.

Problem 3

Six good laptops and three defective laptops are mixed up. To find the defective laptops all of them are tested one-by-one at random. What is the probability to find both of the defective laptops in the first two pick?

Solution

Let A be the event that we find a defective laptop in the first test and B be the event that we find a defective laptop in the second test.

Hence, $P(A \cap B) = P(A)P(B|A) = 3/9 \times 2/8 = 1/21$

Bayes' Theorem

Theorem: If A and B are two mutually exclusive events, where P(A) is the probability of A and P(B) is the probability of B, P(A | B) is the probability of A given that B is true. P(B | A) is the probability of B given that A is true, then Bayes' Theorem states:

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{\sum_{i=1}^{n} P(B \mid Ai) P(Ai)}$$

Application of Bayes' Theorem

- In situations where all the events of sample space are mutually exclusive events.
- In situations where either $P(A_i \cap B)$ for each A_i or $P(A_i)$ and $P(B|A_i)$ for each A_i is known.

Problem

Consider three pen-stands. The first pen-stand contains 2 red pens and 3 blue pens; the second one has 3 red pens and 2 blue pens; and the third one has 4 red pens and 1 blue pen. There is equal probability of each pen-stand to be selected. If one pen is drawn at random, what is the probability that it is a red pen?

Solution

Let A_i be the event that i^{th} pen-stand is selected.
Here, i = 1,2,3.
Since probability for choosing a pen-stand is equal, $P(A_i) = 1/3$

Let B be the event that a red pen is drawn.

The probability that a red pen is chosen among the five pens of the first pen-stand,

$P(B|A_1) = 2/5$

The probability that a red pen is chosen among the five pens of the second pen-stand,

$P(B|A_2) = 3/5$

The probability that a red pen is chosen among the five pens of the third pen-stand,

$P(B|A_3) = 4/5$

According to Bayes' Theorem,

$$\begin{aligned} P(B) &= P(A_1).P(B|A_1) + P(A_2).P(B|A_2) + P(A_3).P(B|A_3) \\ &= 1/3 \cdot 2/5 \ + \ 1/3 \cdot 3/5 \ + \ 1/3 \cdot 4/5 \\ &= 3/5 \end{aligned}$$

Part 5: Mathematical Induction & Recurrence Relations

12. Mathematical Induction

Mathematical induction, is a technique for proving results or establishing statements for natural numbers. This part illustrates the method through a variety of examples.

Definition

Mathematical Induction is a mathematical technique which is used to prove a statement, a formula or a theorem is true for every natural number.

The technique involves two steps to prove a statement, as stated below:

Step 1(Base step): It proves that a statement is true for the initial value.

Step 2(Inductive step): It proves that if the statement is true for the n^{th} iteration (or number n), then it is also true for $(n+1)^{th}$ iteration (or number $n+1$).

How to Do It

Step 1: Consider an initial value for which the statement is true. It is to be shown that the statement is true for n=initial value.

Step 2: Assume the statement is true for any value of $n=k$. Then prove the statement is true for $n=k+1$. We actually break $n=k+1$ into two parts, one part is n=k (which is already proved) and try to prove the other part.

Problem 1

3^n-1 is a multiple of 2 for n=1, 2, ...

Solution

Step 1: For n=1, 3^1-1 = 3-1 = 2 which is a multiple of 2

Step 2: Let us assume 3^n-1 is true for n=k, Hence, 3^k -1 is true (It is an assumption)

We have to prove that 3^{k+1}-1 is also a multiple of 2

$$3^{k+1} - 1 = 3 \times 3^k - 1 = (2 \times 3^k) + (3^k - 1)$$

The first part (2×3^k) is certain to be a multiple of 2 and the second part (3^k -1) is also true as our previous assumption.

Hence, $3^{k+1} - 1$ is a multiple of 2.

So, it is proved that $3^n - 1$ is a multiple of 2.

Problem 2

$1 + 3 + 5 + \ldots + (2n-1) = n^2$ for $n=1, 2, \ldots$

Solution

Step 1: For $n=1$, $1 = 1^2$, Hence, step 1 is satisfied.

Step 2: Let us assume the statement is true for $n=k$.

Hence, $1 + 3 + 5 + \ldots + (2k-1) = k^2$ is true (It is an assumption)

We have to prove that $1 + 3 + 5 + \ldots + (2(k+1)-1) = (k+1)^2$ also holds

$$
\begin{aligned}
& 1 + 3 + 5 + \ldots + (2(k+1) - 1) \\
&= 1 + 3 + 5 + \ldots + (2k+2 - 1) \\
&= 1 + 3 + 5 + \ldots + (2k + 1) \\
&= 1 + 3 + 5 + \ldots + (2k - 1) \mathbf{+} (2k + 1) \\
&= k^2 + (2k + 1) \\
&= (k + 1)^2
\end{aligned}
$$

So, $1 + 3 + 5 + \ldots + (2(k+1) - 1) = (k+1)^2$ hold which satisfies the step 2.

Hence, $1 + 3 + 5 + \ldots + (2n - 1) = n^2$ is proved.

Problem 3

Prove that $(ab)^n = a^n b^n$ is true for every natural number *n*

Solution

Step 1: For $n=1$, $(ab)^1 = a^1b^1 = ab$, Hence, step 1 is satisfied.

Step 2: Let us assume the statement is true for $n=k$, Hence, $(ab)^k = a^kb^k$ is true (It is an assumption).

We have to prove that $(ab)^{k+1} = a^{k+1}b^{k+1}$ also hold

Given, $(ab)^k = a^kb^k$

Or, $(ab)^k (ab)= (a^kb^k) (ab)$ [Multiplying both side by 'ab']

Or, $(ab)^{k+1} = (aa^k) (bb^k)$

Or, $(ab)^{k+1} = (a^{k+1}b^{k+1})$

Hence, step 2 is proved.
So, $(ab)^n = a^nb^n$ is true for every natural number n.

Strong Induction

Strong Induction is another form of mathematical induction. Through this induction technique, we can prove that a propositional function, *P(n)* is true for all positive integers, *n*, using the following steps:

- **Step 1(Base step):** It proves that the initial proposition *P(1)* true.
- **Step 2(Inductive step):** It proves that the conditional statement $[P(1) \land P(2) \land P(3) \land \ldots\ldots\ldots\ldots \land P(k)] \rightarrow P(k+1)$ is true for positive integers *k*.

13. Recurrence Relation

In this chapter, we will discuss how recursive techniques can derive sequences and be used for solving counting problems. The procedure for finding the terms of a sequence in a recursive manner is called **recurrence relation**. We study the theory of linear recurrence relations and their solutions. Finally, we introduce generating functions for solving recurrence relations.

Definition

A recurrence relation is an equation that recursively defines a sequence where the next term is a function of the previous terms (Expressing F_n as some combination of F_i with $i<n$).

Example: Fibonacci series: $F_n = F_{n-1} + F_{n-2}$, Tower of Hanoi: $F_n = 2F_{n-1} + 1$

Linear Recurrence Relations

A linear recurrence equation of degree **k** or order **k** is a recurrence equation which is in the format $x_n= A_1\ x_{n-1}+ A_2\ x_{n-1}+ A_3\ x_{n-1}+\dots A_k\ x_{n-k}$ (A_n is a constant and $A_k \neq 0$) on a sequence of numbers as a first-degree polynomial.

These are some examples of linear recurrence equations:

Recurrence relations	Initial values	Solutions
$F_n = F_{n-1} + F_{n-2}$	$a_1=a_2=1$	Fibonacci number
$F_n = F_{n-1} + F_{n-2}$	$a_1=1, a_2=3$	Lucas number
$F_n = F_{n-2} + F_{n-3}$	$a_1=a_2=a_3=1$	Padovan sequence
$F_n = 2F_{n-1} + F_{n-2}$	$a_1=0, a_2=1$	Pell number

How to solve linear recurrence relation

Suppose, a two ordered linear recurrence relation is: $F_n = AF_{n-1} + BF_{n-2}$ where A and B are real numbers.

The characteristic equation for the above recurrence relation is:

$$x^2 - Ax - B = 0$$

Three cases may occur while finding the roots:

Case 1: If this equation factors as $(x- x_1)(x- x_1) = 0$ and it produces two distinct real roots x_1 and x_2, then $F_n = ax_1^n+ bx_2^n$ is the solution. [Here, a and b are constants]

Case 2: If this equation factors as $(x- x_1)^2 = 0$ and it produces single real root x_1, then $F_n = a\ x_1^n+ bn\ x_1^n$ is the solution.

Case 3: If the equation produces two distinct complex roots, x_1 and x_2 in polar form $x_1 = r \angle \theta$ and $x_2 = r \angle(- \theta)$, then $F_n = r^n\ (a \cos(n\theta)+ b \sin(n\theta))$ is the solution.

Problem 1

Solve the recurrence relation $F_n = 5F_{n-1} - 6F_{n-2}$ where $F_0 = 1$ and $F_1 = 4$

Solution

The characteristic equation of the recurrence relation is:

$$x^2 - 5x + 6=0,$$

So, $(x-3)(x-2) = 0$

Hence, the roots are:

$$x_1 = 3 \text{ and } \quad x_2= 2$$

The roots are real and distinct. So, this is in the form of case 1

Hence, the solution is:

$$F_n = ax_1^n+ bx_2^n$$

Here, $F_n = a3^n+ b2^n$ (As $x_1 = 3$ and $x_2= 2$)

Therefore,

$1=F_0 = a3^0+ b2^0 = a+b$

$4=F_1 = a3^1+ b2^1 = 3a+2b$

Solving these two equations, we get $a = 2$ and $b = -1$

Hence, the final solution is:

$$F_n = 2.3^n + (-1) . 2^n= 2.3^n - 2^n$$

Problem 2

Solve the recurrence relation $F_n = 10F_{n-1} - 25F_{n-2}$ where $F_0 = 3$ and $F_1 = 17$

Solution

The characteristic equation of the recurrence relation is:

$x^2 -10x -25 =0,$

So, $(x - 5)^2 = 0$

Hence, there is single real root $x_1 = 5$

As there is single real valued root, this is in the form of case 2

Hence, the solution is:

$F_n = ax_1^n + bnx_1^n$

$3 = F_0= a.5^0+ b.0.5^0 = a$

$17 = F_1= a.5^1 + b.1.5^1 = 5a+5b$

Solving these two equations, we get $a = 3$ and $b = 2/5$

Hence, the final solution is:

$$F_n = 3.5^n + (2/5) .n.2^n$$

Problem 3

Solve the recurrence relation $F_n = 2F_{n-1} - 2F_{n-2}$ where $F_0 = 1$ and $F_1 = 3$

Solution

The characteristic equation of the recurrence relation is:

$$x^2 -2x -2 =0$$

Hence, the roots are:

$x_1 = 1+ i$ and $x_2= 1- i$

In polar form,

$x_1 = r \angle \theta$ and $x_2 = r \angle(- \theta)$, where $r= \sqrt{2}$ and $\theta= \pi / 4$

The roots are imaginary. So, this is in the form of case 3.

Hence, the solution is:

$$F_n = (\sqrt{2})^n (a \cos(n. \pi / 4) + b \sin(n. \pi / 4))$$

$$1 = F_0 = (\sqrt{2})^0 (a \cos(0. \pi / 4) + b \sin(0. \pi / 4)) = a$$

$$3 = F_1 = (\sqrt{2})^1 (a \cos(1. \pi / 4) + b \sin(1. \pi / 4)) = \sqrt{2} (a/\sqrt{2} + b/\sqrt{2})$$

Solving these two equations we get $a = 1$ and $b = 2$

Hence, the final solution is:

$$F_n = (\sqrt{2})^n (\cos(n. \pi / 4)+ 2 \sin(n. \pi / 4))$$

Non-Homogeneous Recurrence Relation and Particular Solutions

A recurrence relation is called non-homogeneous if it is in the form

$F_n = AF_{n-1} + BF_{n-2} + f(n)$ where $f(n) \neq 0$

Its associated homogeneous recurrence relation is $F_n = AF_{n-1} + BF_{n-2}$

The solution (a_n) of a non-homogeneous recurrence relation has two parts.

First part is the solution (a_h) of the associated homogeneous recurrence relation and the second part is the particular solution (a_t).

$$a_n= a_h + a_t$$

Solution to the first part is done using the procedures discussed in the previous section.

To find the particular solution, we find an appropriate trial solution.

Let $f(n) = cx^n$; let $x^2 = Ax + B$ be the characteristic equation of the associated homogeneous recurrence relation and let x_1 and x_2 be its roots.

- If $x \neq x_1$ and $x \neq x_2$, then $a_t = Ax^n$
- If $x = x_1$, $x \neq x_2$, then $a_t = Anx^n$
- If $x= x_1 = x_2$, then $a_t = An^2x^n$

Example

Let a non-homogeneous recurrence relation be $F_n = AF_{n-1} + BF_{n-2} + f(n)$ with characteristic roots $x_1 = 2$ and $x_2 = 5$. Trial solutions for different possible values of *f(n)* are as follows:

f(n)	**Trial solutions**
4	A
5.2^n	$An2^n$
8.5^n	$An5^n$
4^n	$A4^n$
$2n^2 + 3n + 1$	$An^2 + Bn + C$

Problem

Solve the recurrence relation $F_n = 3F_{n-1} + 10F_{n-2} + 7.5^n$ where $F_0 = 4$ and $F_1 = 3$

Solution

This is a linear non-homogeneous relation, where the associated homogeneous equation is $F_n = 3F_{n-1} + 10F_{n-2}$ and $f(n) = 7.5^n$.

The characteristic equation of its associated homogeneous relation is:

$x^2 - 3x - 10 = 0$

Or, $(x - 5)(x + 2) = 0$

Or, $x_1 = 5$ and $x_2 = -2$

Hence $a_h = a.5^n + b.(-2)^n$, where a and b are constants.

Since $f(n) = 7.5^n$, i.e. of the form $c.x^n$, a reasonable trial solution of a_t will be Anx^n

$a_t = Anx^n = An5^n$

After putting the solution in the recurrence relation, we get:

$An5^n = 3A(n - 1)5^{n-1} + 10A(n - 2)5^{n-2} + 7.5^n$

Dividing both sides by 5^{n-2}, we get:

$An5^2 = 3A(n - 1)5 + 10A(n - 2)5^0 + 7.5^2$

Or, $25An = 15An - 15A + 10An - 20A + 175$

Or, $35A = 175$

Or, $A = 5$

So, $F_n = An5^n = 5n5^n = n5^{n+1}$

The solution of the recurrence relation can be written as:

$F_n = a_h + a_t$

$= a.5^n + b.(-2)^n + n5^{n+1}$

Putting values of $F_0 = 4$ and $F_1 = 3$, in the above equation, we get $a = -2$ and $b = 6$

Hence, the solution is:

$F_n = n5^{n+1} + 6.(-2)^n - 2.5^n$

Generating Functions

Generating Functions represents sequences where each term of a sequence is expressed as a coefficient of a variable *x* in a formal power series.

Mathematically, for an infinite sequence, say $a_0, a_1, a_2, \dots\dots\dots, a_k, \dots\dots\dots$, the generating function will be:

$$G_x = a_0 + a_1 x + a_2 x^2 + \dots\dots\dots + a_k x^k + \dots\dots\dots = \sum_{k=0}^{\infty} a_k x^k$$

Some Areas of Application:

Generating functions can be used for the following purposes:

- For solving a variety of counting problems. For example, the number of ways to make change for a Rs. 100 note with the notes of denominations Rs.1, Rs.2, Rs.5, Rs.10, Rs.20 and Rs.50
- For solving recurrence relations
- For proving some of the combinatorial identities
- For finding asymptotic formulae for terms of sequences

Problem 1

What are the generating functions for the sequences $\{a_k\}$ with $a_k = 2$ and $a_k = 3k$?

Solution

When $a_k = 2$, generating function, G(x) = $\sum_{k=0}^{\infty} 2x^k = 2 + 2x + 2x^2 + 2x^3 + \dots\dots\dots$

When $a_k = 3k$, G(x) = $\sum_{k=0}^{\infty} 3kx^k = 0 + 3x + 6x^2 + 9x^3 + \dots\dots\dots$

Problem 2

What is the generating function of the infinite series; 1, 1, 1, 1, ……….?

Solution

Here, $a_k = 1, for\ 0 \le k \le \infty$.
Hence, G(x) = $1 + x + x^2 + x^3 + \dots\dots\dots = \frac{1}{(1-x)}$

Some Useful Generating Functions

- For $a_k = a^k$, G(x) = $\sum_{k=0}^{\infty} a^k x^k = 1 + ax + a^2x^2 + \dots\dots\dots = {}^{1}/_{(1-ax)}$
- For $a_k = (k+1)$, G(x) = $\sum_{k=0}^{\infty}(k+1)x^k = 1 + 2x + 3x^2 + \dots\dots\dots = \frac{1}{(1-x)^2}$
- For $a_k = C_k^n$, G(x) = $\sum_{k=0}^{\infty} C_k^n x^k = 1 + C_1^n x + C_2^n x^2 + \dots\dots\dots + x^2 = (1+x)^n$
- For $a_k = \frac{1}{k!}$, G(x) = $\sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} \dots\dots\dots = e^x$

Part 6: Discrete Structures

14. Graph and Graph Models

The previous part brought forth the different tools for reasoning, proofing and problem solving. In this part, we will study the discrete structures that form the basis of formulating many a real-life problem.

The two discrete structures that we will cover are graphs and trees. A graph is a set of points, called nodes or vertices, which are interconnected by a set of lines called edges. The study of graphs, or **graph theory** is an important part of a number of disciplines in the fields of mathematics, engineering and computer science.

What is a Graph?

Definition: A graph (denoted as G = (V, E)) consists of a non-empty set of vertices or nodes V and a set of edges E.

Example: Let us consider, a Graph is G = (V, E) where V = {a, b, c, d} and E = {{a, b}, {a, c}, {b, c},{c, d}}

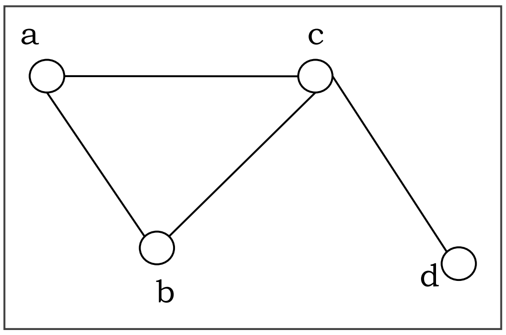


Figure: A graph with four vertices and four edges

Degree of a Vertex: The degree of a vertex V of a graph G (denoted by deg (V)) is the number of edges incident with the vertex V.

Vertex	Degree	Even / Odd
a	2	even
b	2	even
c	3	odd
d	1	odd

Even and Odd Vertex: If the degree of a vertex is even, the vertex is called an even vertex and if the degree of a vertex is odd, the vertex is called an odd vertex.

Degree of a Graph: The degree of a graph is the largest vertex degree of that graph. For the above graph the degree of the graph is 3.

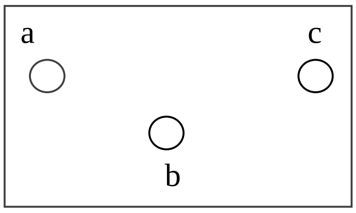
The Handshaking Lemma: In a graph, the sum of all the degrees of all the vertices is equal to twice the number of edges.

Types of Graphs

There are different types of graphs, which we will learn in the following section.

Null Graph

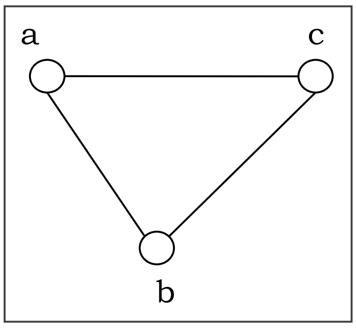
A null graph has no edges. The null graph of n vertices is denoted by N_n



Null graph of 3 vertices

Simple Graph

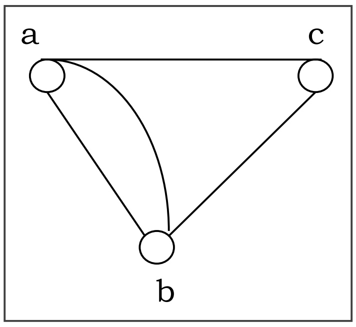
A graph is called simple graph/strict graph if the graph is undirected and does not contain any loops or multiple edges.



Simple graph

Multi-Graph

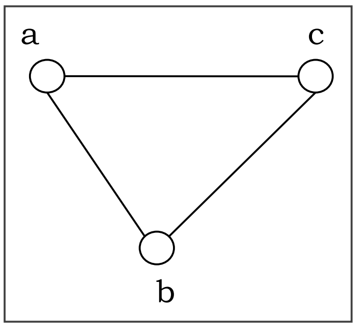
If in a graph multiple edges between the same set of vertices are allowed, it is called Multi-graph. In other words, it is a graph having at least one loop or multiple edges.



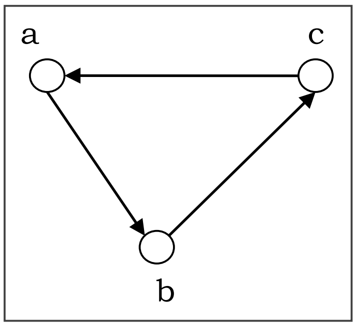
Multi-graph

Directed and Undirected Graph

A graph G = (V, E) is called a directed graph if the edge set is made of ordered vertex pair and a graph is called undirected if the edge set is made of unordered vertex pair.



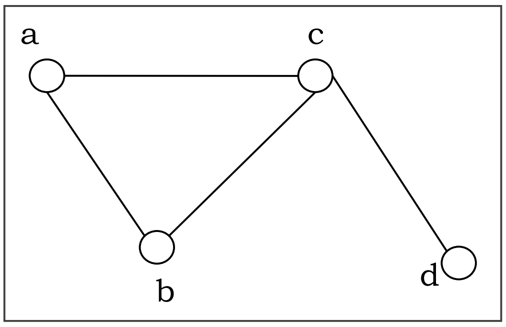
Undirected graph



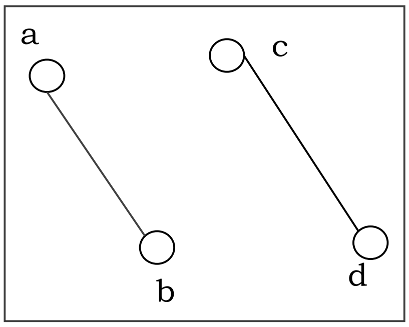
Directed graph

Connected and Disconnected Graph

A graph is connected if any two vertices of the graph are connected by a path; while a graph is disconnected if at least two vertices of the graph are not connected by a path. If a graph G is disconnected, then every maximal connected subgraph of G is called a connected component of the graph G.



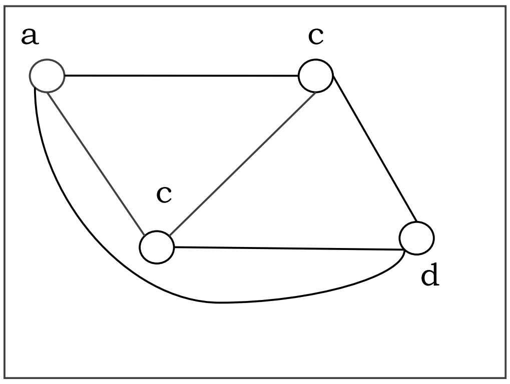
Connected graph



Unconnected graph

Regular Graph

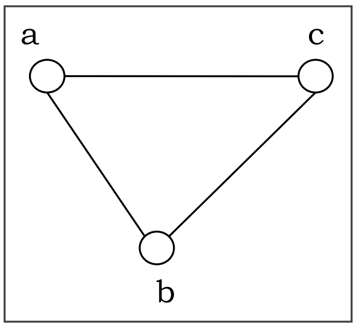
A graph is regular if all the vertices of the graph have the same degree. In a regular graph G of degree r, the degree of each vertex of G is r.



Regular graph of degree 3

Complete Graph

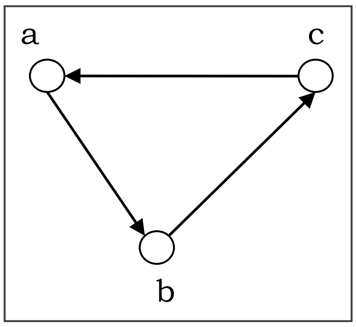
A graph is called complete graph if every two vertices pair are joined by exactly one edge. The complete graph with n vertices is denoted by K_n



Complete graph K_3

Cycle Graph

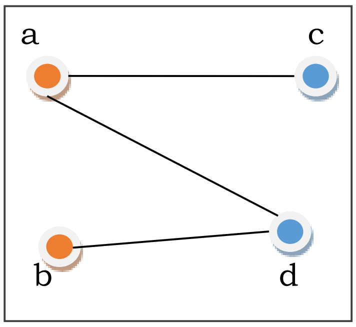
If a graph consists of a single cycle, it is called cycle graph. The cycle graph with n vertices is denoted by C_n



Cyclic graph C_3

Bipartite Graph

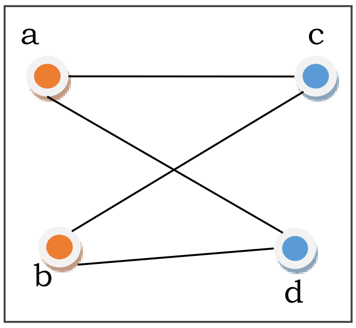
If the vertex-set of a graph G can be split into two disjoint sets, V_1 and V_2, in such a way that each edge in the graph joins a vertex in V_1 to a vertex in V_2, and there are no edges in G that connect two vertices in V_1 or two vertices in V_2, then the graph G is called a bipartite graph.



Bipartite graph

Complete Bipartite Graph

A complete bipartite graph is a bipartite graph in which each vertex in the first set is joined to every single vertex in the second set. The complete bipartite graph is denoted by $K_{x,y}$ where the graph G contains *x* vertices in the first set and *y* vertices in the second set.



Complete bipartite graph $K_{2,2}$

Representation of Graphs

There are mainly two ways to represent a graph:

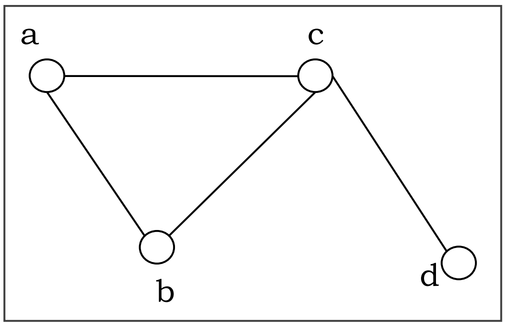
- Adjacency Matrix
- Adjacency List

Adjacency Matrix

An Adjacency Matrix A[V][V] is a 2D array of size V×V where V is the number of vertices in a undirected graph. If there is an edge between V_x to V_y then the value of A[V_x][V_y]=1 and A[V_y][V_x]=1, otherwise the value will be zero. And for a directed graph, if there is an edge between V_x to V_y, then the value of A[V_x][V_y]=1, otherwise the value will be zero.

Adjacency Matrix of an Undirected Graph

Let us consider the following undirected graph and construct the adjacency matrix:



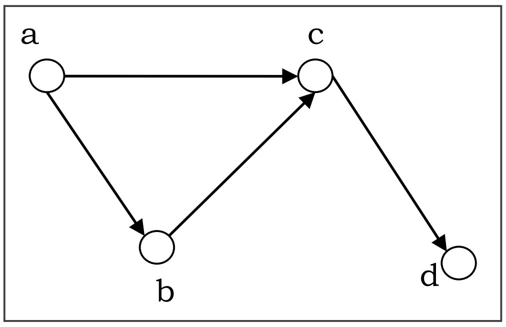
An undirected graph

Adjacency matrix of the above undirected graph will be:

	a	**b**	**c**	**d**
a	0	1	1	0
b	1	0	1	0
c	1	1	0	1
d	0	0	1	0

Adjacency Matrix of a Directed Graph

Let us consider the following directed graph and construct its adjacency matrix:



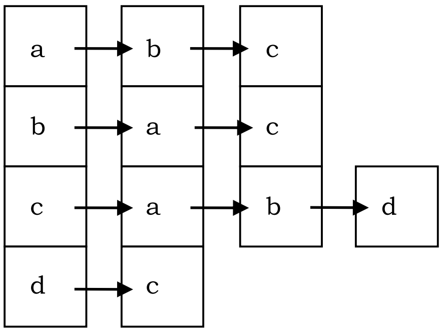
A directed graph

Adjacency matrix of the above directed graph will be:

	a	**b**	**c**	**d**
a	0	1	1	0
b	0	0	1	0
c	0	0	0	1
d	0	0	0	0

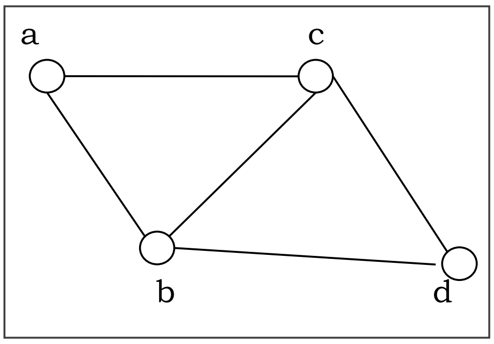
Adjacency List

In adjacency list, an array (A[V]) of linked lists is used to represent the graph G with V number of vertices. An entry A[V_x] represents the linked list of vertices adjacent to the V**x-**th vertex. The adjacency list of the undirected graph is as shown in the figure below:



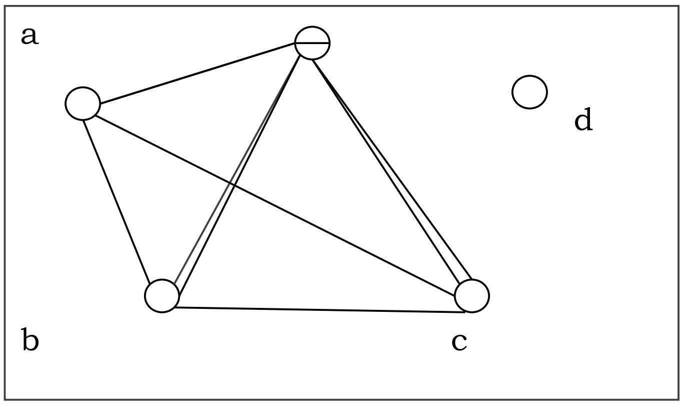
Planar vs. Non-planar Graph

Planar graph: A graph G is called a planar graph if it can be drawn in a plane without any edges crossed. If we draw graph in the plane without edge crossing, it is called embedding the graph in the plane.



Planar graph

Non-planar graph: A graph is non-planar if it cannot be drawn in a plane without graph edges crossing.



Non-planar graph

Isomorphism

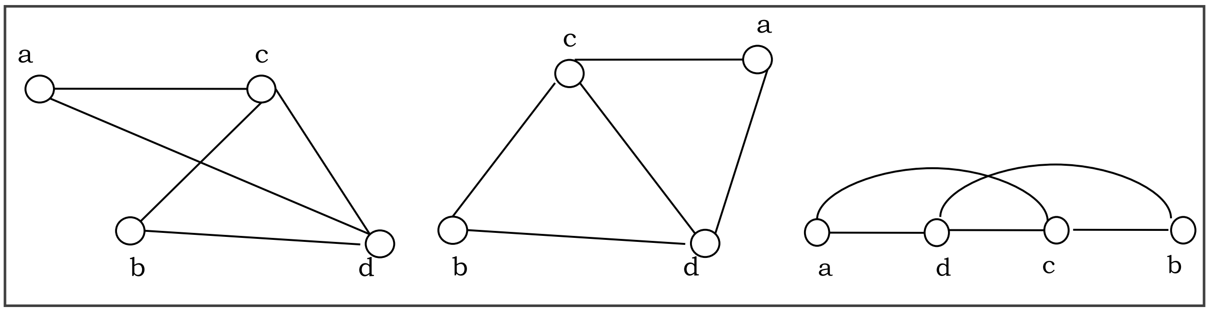
If two graphs G and H contain the same number of vertices connected in the same way, they are called isomorphic graphs (denoted by G≅H).

It is easier to check non-isomorphism than isomorphism. If any of these following conditions occurs, then two graphs are non-isomorphic:

- The number of connected components are different
- Vertex-set cardinalities are different
- Edge-set cardinalities are different
- Degree sequences are different

Example

The following graphs are isomorphic:



Three isomorphic graphs

Homomorphism

A homomorphism from a graph G to a graph H is a mapping (May not be a bijective mapping) h: G→H such that: (x, y) ∈ E(G) → (h(x), h(y)) ∈ E(H) . It maps adjacent vertices of graph G to the adjacent vertices of the graph H.

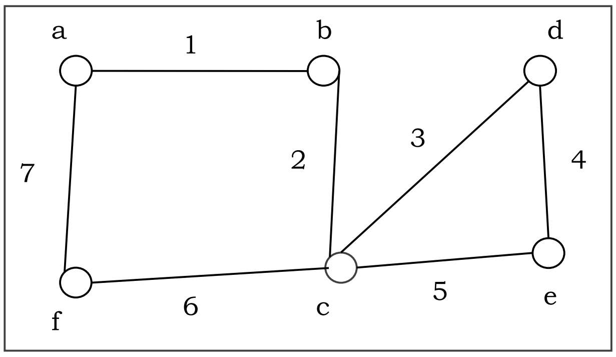
Properties of Homomorphisms:

- A homomorphism is an isomorphism if it is a bijective mapping.
- Homomorphism always preserves edges and connectedness of a graph.
- The compositions of homomorphisms are also homomorphisms.
- To find out if there exists any homomorphic graph of another graph is a NP-complete problem.

Euler Graphs

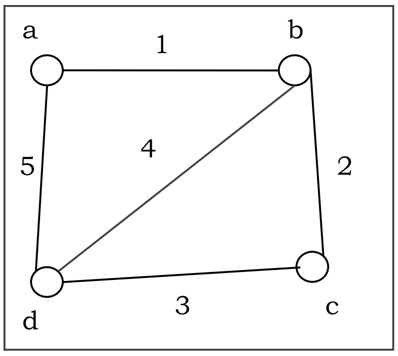
A connected graph G is called an Euler graph, if there is a closed trail which includes every edge of the graph G. An Euler path is a path that uses every edge of a graph exactly once. An Euler path starts and ends at different vertices.

An Euler circuit is a circuit that uses every edge of a graph exactly once. An Euler circuit always starts and ends at the same vertex. A connected graph G is an Euler graph if and only if all vertices of G are of even degree, and a connected graph G is Eulerian if and only if its edge set can be decomposed into cycles.



Euler graph

The above graph is an Euler graph as "a 1 b 2 c 3 d 4 e 5 c 6 f 7 g" covers all the edges of the graph.



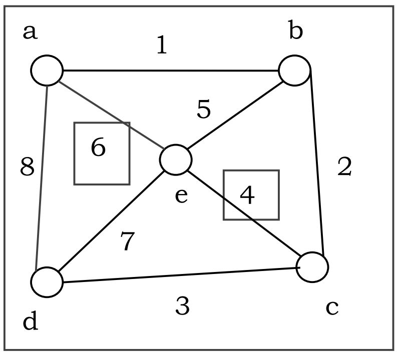
Non-Euler graph

Hamiltonian Graphs

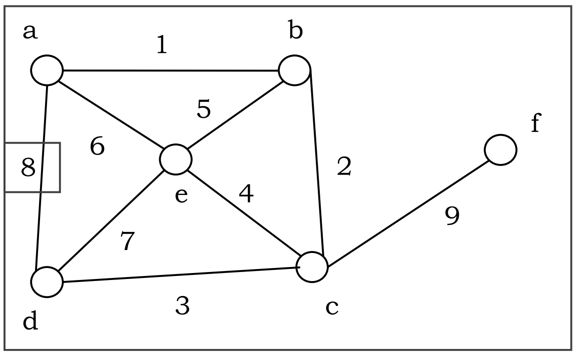
A connected graph G is called Hamiltonian graph if there is a cycle which includes every vertex of G and the cycle is called Hamiltonian cycle. Hamiltonian walk in graph G is a walk that passes through each vertex exactly once.

If G is a simple graph with n vertices, where n ≥ 3 If deg(v) ≥ n/2 for each vertex v, then the graph G is Hamiltonian graph. This is called **Dirac's Theorem**.

If G is a simple graph with n vertices, where n ≥ 2 if deg(x) + deg(y) ≥ n for each pair of non-adjacent vertices x and y, then the graph G is Hamiltonian graph. This is called **Ore's theorem**.



Hamiltonian graph



Non-Hamiltonian graph

15. More on Graphs

Graph Coloring

Graph coloring is the procedure of assignment of colors to each vertex of a graph G such that no adjacent vertices get same color. The objective is to minimize the number of colors while coloring a graph. The smallest number of colors required to color a graph G is called its chromatic number of that graph. Graph coloring problem is a NP Complete problem.

Method to Color a Graph

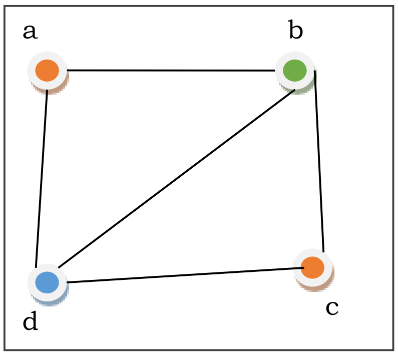
The steps required to color a graph G with n number of vertices are as follows:

Step 1. Arrange the vertices of the graph in some order.

Step 2. Choose the first vertex and color it with the first color.

Step 3. Choose the next vertex and color it with the lowest numbered color that has not been colored on any vertices adjacent to it. If all the adjacent vertices are colored with this color, assign a new color to it. Repeat this step until all the vertices are colored.

Example



Graph coloring

In the above figure, at first vertex **a** is colored red. As the adjacent vertices of vertex **a** are again adjacent, vertex **b** and vertex **d** are colored with different color, green and blue respectively. Then vertex **c** is colored as red as no adjacent vertex of **c** is colored red. Hence, we could color the graph by 3 colors. Hence, the chromatic number of the graph is 3.

Applications of Graph Coloring

Some applications of graph coloring include –

- Register Allocation
- Map Coloring
- Bipartite Graph Checking
- Mobile Radio Frequency Assignment
- Making time table, etc.

Graph Traversal

Graph traversal is the problem of visiting all the vertices of a graph in some systematic order. There are mainly two ways to traverse a graph.

- Breadth First Search
- Depth First Search

Breadth First Search

Breadth First Search (BFS) starts at starting level-0 vertex X of the graph G. Then we visit all the vertices that are the neighbors of X. After visiting, we mark the vertices as "visited," and place them into level-1. Then we start from the level-1 vertices and apply the same method on every level-1 vertex and so on. The BFS traversal terminates when every vertex of the graph has been visited.

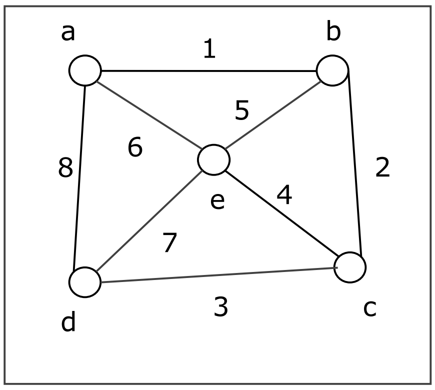
BFS Algorithm

The concept is to visit all the neighbor vertices before visiting other neighbor vertices of neighbor vertices.

- Initialize status of all nodes as "Ready"
- Put source vertex in a queue and change its status to "Waiting"
- Repeat the following two steps until queue is empty:
  - Remove the first vertex from the queue and mark it as "Visited"
  - Add to the rear of queue all neighbors of the removed vertex whose status is "Ready". Mark their status as "Waiting".

Problem

Let us take a graph (Source vertex is 'a') and apply the BFS algorithm to find out the traversal order.



A graph

Solution:

- Initialize status of all vertices to "Ready"
- Put *a* in queue and change its status to "Waiting"
- Remove *a* from queue, mark it as "Visited"
- Add *a*'s neighbors in "Ready" state *b*, *d* and *e* to end of queue and mark them as "Waiting"
- Remove *b* from queue, mark it as "Visited", put its "Ready" neighbor *c* at end of queue and mark *c* as "Waiting"
- Remove *d* from queue and mark it as "Visited". It has no neighbor in "Ready" state.
- Remove *e* from queue and mark it as "Visited". It has no neighbor in "Ready" state.
- Remove *c* from queue and mark it as "Visited". It has no neighbor in "Ready" state.
- Queue is empty so stop.

So the traversal order is:

a→b→d→e→c

The alternate orders of traversal are:

a→b→e→d→c

Or,

a→d→b→e→c

Or,

a→e→b→d→c

Or,

a→b→e→d→c

Or,

a→d→e→b→c

Application of BFS

- Finding the shortest path
- Minimum spanning tree for un-weighted graph
- GPS navigation system

- Detecting cycles in an undirected graph
- Finding all nodes within one connected component

Complexity Analysis

Let G(V, E) be a graph with |V| number of vertices and |E| number of edges. If breadth first search algorithm visits every vertex in the graph and checks every edge, then its time complexity would be:

$$O(|V| + |E|).\ O(|E|)$$

It may vary between $O(1)$ and $O(|V^2|)$

Depth First Search

Depth First Search (DFS) algorithm starts from a vertex v, then it traverses to its adjacent vertex (say x) that has not been visited before and mark as "visited" and goes on with the adjacent vertex of x and so on.

If at any vertex, it encounters that all the adjacent vertices are visited, then it backtracks until it finds the first vertex having an adjacent vertex that has not been traversed before. Then, it traverses that vertex, continues with its adjacent vertices until it traverses all visited vertices and has to backtrack again. In this way, it will traverse all the vertices reachable from the initial vertex v.

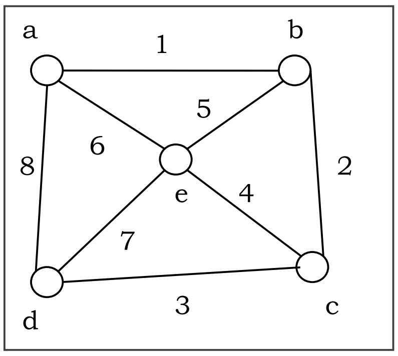
DFS Algorithm

The concept is to visit all the neighbor vertices of a neighbor vertex before visiting the other neighbor vertices.

- Initialize status of all nodes as "Ready"
- Put source vertex in a stack and change its status to "Waiting"
- Repeat the following two steps until stack is empty:
    - Pop the top vertex from the stack and mark it as "Visited"
    - Push onto the top of the stack all neighbors of the removed vertex whose status is "Ready". Mark their status as "Waiting".

Problem

Let us take a graph (Source vertex is 'a') and apply the DFS algorithm to find out the traversal order.



A graph

Solution

- Initialize status of all vertices to "Ready"
- Push *a* in stack and change its status to "Waiting"
- Pop *a* and mark it as "Visited"
- Push *a*'s neighbors in "Ready" state *e*, *d* and *b* to top of stack and mark them as "Waiting"

- Pop *b* from stack, mark it as "Visited", push its "Ready" neighbor *c* onto stack.
- Pop c from stack and mark it as "Visited". It has no "Ready" neighbor.
- Pop *d* from stack and mark it as "Visited". It has no "Ready" neighbor.
- Pop *e* from stack and mark it as "Visited". It has no "Ready" neighbor.
- Stack is empty. So stop.

So the traversal order is:

$a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$

The alternate orders of traversal are:

$a \rightarrow e \rightarrow b \rightarrow c \rightarrow d$

Or,

$a \rightarrow b \rightarrow e \rightarrow c \rightarrow d$

Or,

$a \rightarrow d \rightarrow e \rightarrow b \rightarrow c$

Or,

$a \rightarrow d \rightarrow c \rightarrow e \rightarrow b$

Or,

$a \rightarrow d \rightarrow c \rightarrow b \rightarrow e$

Complexity Analysis

Let G(V, E) be a graph with |V| number of vertices and |E| number of edges. If DFS algorithm visits every vertex in the graph and checks every edge, then the time complexity is:

$$\Theta\ (|V| + |E|)$$

Applications

- Detecting cycle in a graph
- To find topological sorting
- To test if a graph is bipartite
- Finding connected components
- Finding the bridges of a graph
- Finding bi-connectivity in graphs
- Solving the Knight’s Tour problem
- Solving puzzles with only one solution

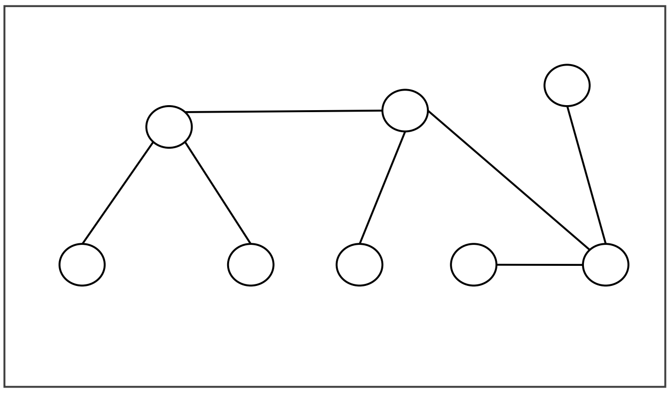
16. Introduction to Trees

Tree is a discrete structure that represents hierarchical relationships between individual elements or nodes. A tree in which a parent has no more than two children is called a binary tree.

Tree and its Properties

Definition: A Tree is a connected acyclic undirected graph. There is a unique path between every pair of vertices in G. A tree with N number of vertices contains (N-1) number of edges. The vertex which is of 0 degree is called root of the tree. The vertex which is of 1 degree is called leaf node of the tree and the degree of an internal node is at least 2.

Example: The following is an example of a tree:



A tree

Centers and Bi-Centers of a Tree

The center of a tree is a vertex with minimal eccentricity. The eccentricity of a vertex X in a tree G is the maximum distance between the vertex X and any other vertex of the tree. The maximum eccentricity is the tree diameter. If a tree has only one center, it is called Central Tree and if a tree has only more than one centers, it is called Bi-central Tree. Every tree is either central or bi-central.

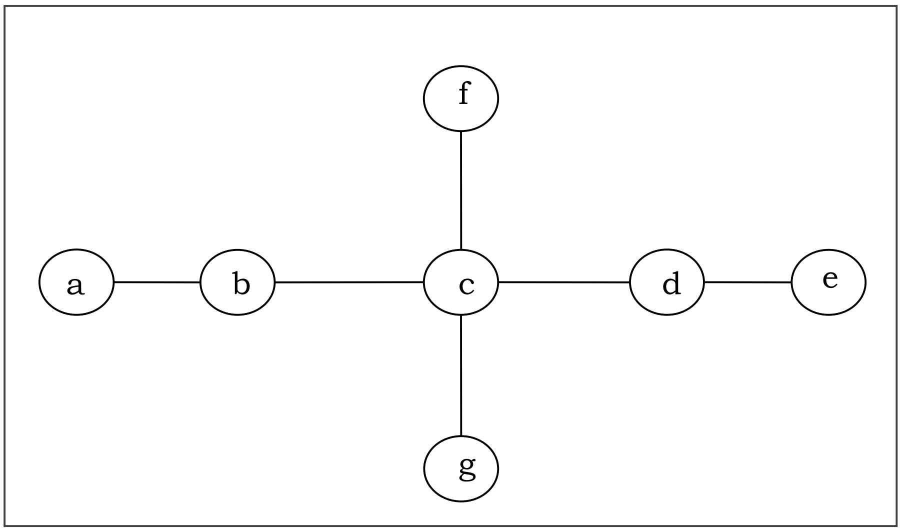
Algorithm to find centers and bi-centers of a tree

Step 1: Remove all the vertices of degree 1 from the given tree and also remove their incident edges.

Step 2: Repeat step 1 until either a single vertex or two vertices joined by an edge is left. If a single vertex is left then it is the center of the tree and if two vertices joined by an edge is left then it is the bi-center of the tree.

Problem 1

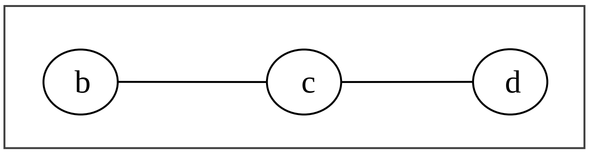
Find out the center/bi-center of the following tree:



Tree T1

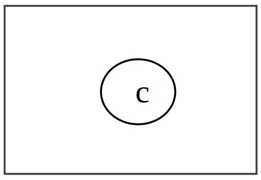
Solution

At first, we will remove all vertices of degree 1 and also remove their incident edges and get the following tree:



Tree after removing vertices of degree 1 from T1

Again, we will remove all vertices of degree 1 and also remove their incident edges and get the following tree:

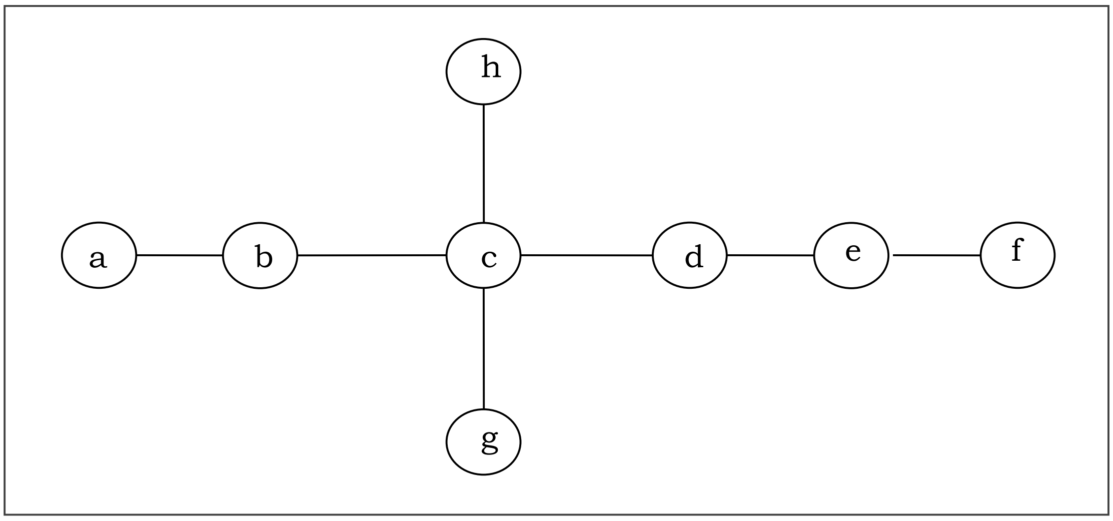


Tree after again removing vertices of degree 1

Finally we got a single vertex 'c' and we stop the algorithm. As there is single vertex, this tree has one center 'c' and the tree is a central tree.

Problem 2

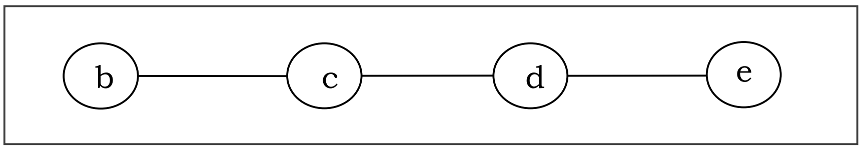
Find out the center/bi-center of the following tree:



A tree T2

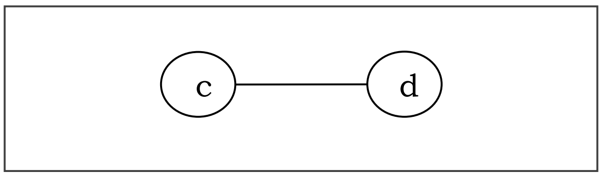
Solution

At first, we will remove all vertices of degree 1 and also remove their incident edges and get the following tree:



Tree after removing vertices of degree 1 from T2

Again, we will remove all vertices of degree 1 and also remove their incident edges and get the following tree:



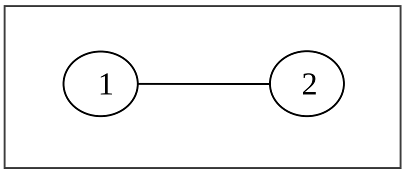
Tree after again removing vertices of degree 1

Finally, we got two vertices 'c' and 'd' left, hence we stop the algorithm. As two vertices joined by an edge is left, this tree has bi-center 'cd' and the tree is bi-central.

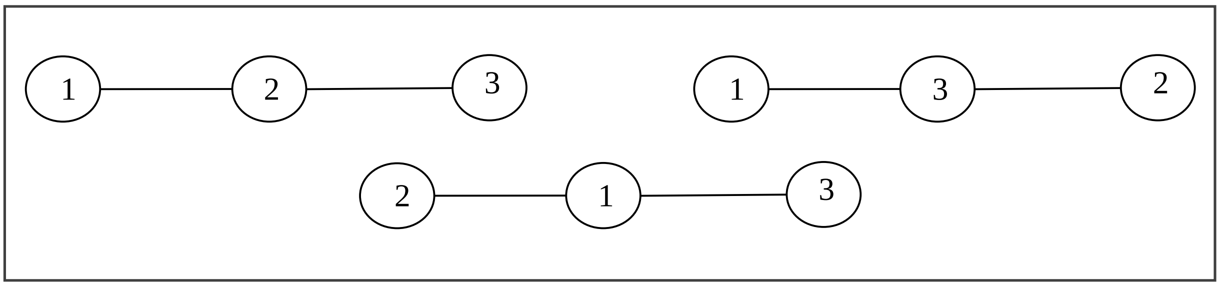
Labeled Trees

Definition: A labeled tree is a tree the vertices of which are assigned unique numbers from 1 to n. We can count such trees for small values of n by hand so as to conjecture a general formula. The number of labeled trees of n number of vertices is n^{n-2}. Two labeled trees are isomorphic if their graphs are isomorphic and the corresponding points of the two trees have the same labels.

Example



A labeled tree with two vertices

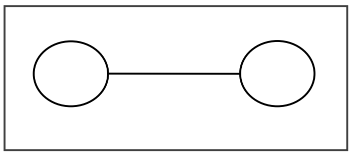


Three possible labeled tree with three vertices

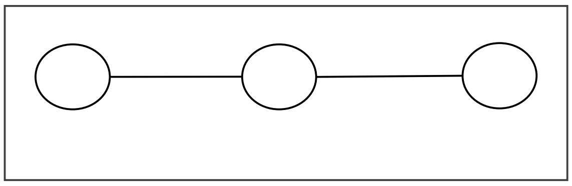
Unlabeled trees

Definition: An unlabeled tree is a tree the vertices of which are not assigned any numbers. The number of labeled trees of n number of vertices is $\frac{(2n)!}{(n+1)!n!}$ (nth Catalan number)

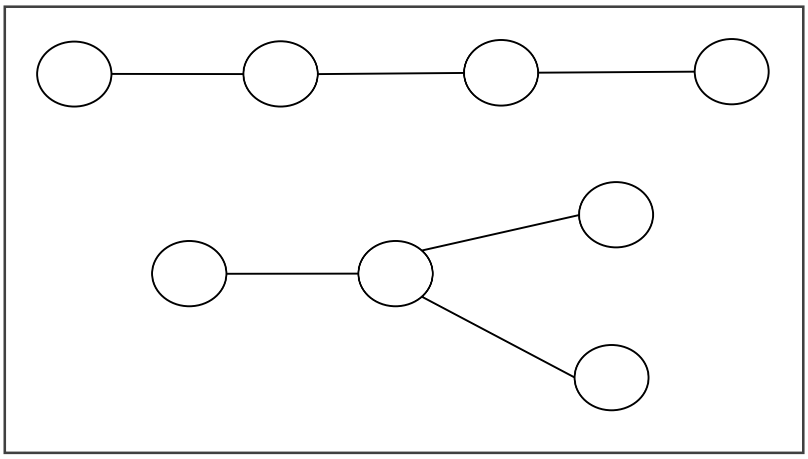
Example



An unlabeled tree with two vertices



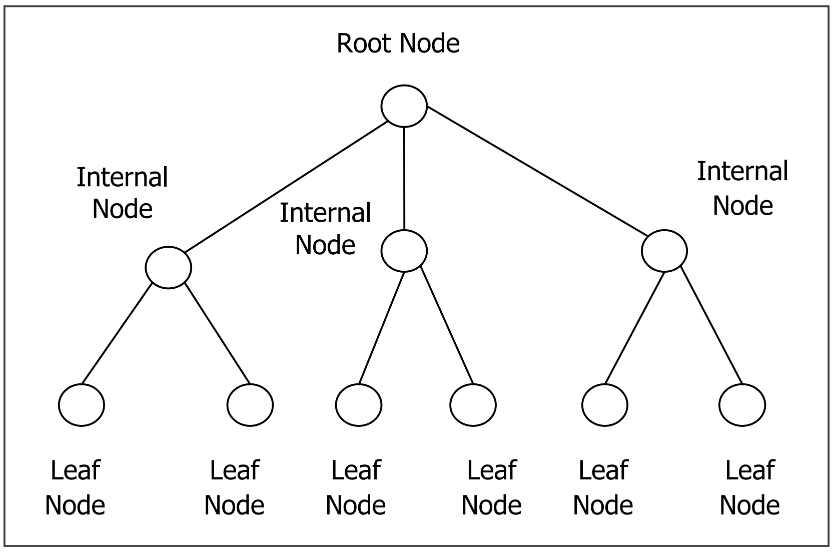
An unlabeled tree with three vertices



Two possible unlabeled trees with four vertices

Rooted Tree

A rooted tree G is a connected acyclic graph with a special node that is called the root of the tree and every edge directly or indirectly originates from the root. An ordered rooted tree is a rooted tree where the children of each internal vertex are ordered. If every internal vertex of a rooted tree has not more than m children, it is called an m-ary tree. If every internal vertex of a rooted tree has exactly m children, it is called a full m-ary tree. If m = 2, the rooted tree is called a binary tree.



A Rooted Tree

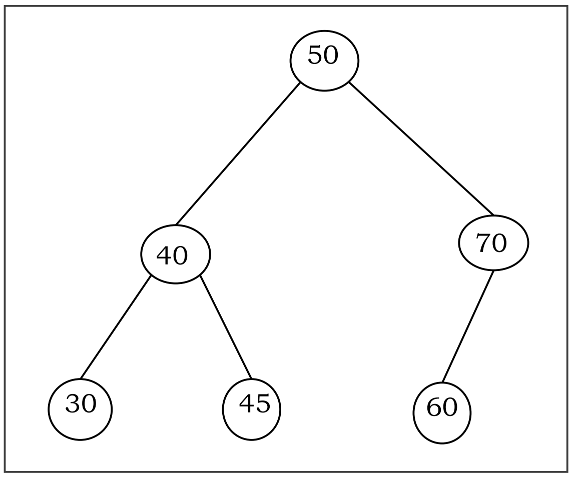
Binary Search Tree

Binary Search tree is a binary tree which satisfies the following property:

- X in left sub-tree of vertex V, Value(X) ≤ Value (V)
- Y in right sub-tree of vertex V, Value(Y) ≥ Value (V)

So, the value of all the vertices of the left sub-tree of an internal node V are less than or equal to V and the value of all the vertices of the right sub-tree of the internal node V are greater than or equal to V. The number of links from the root node to the deepest node is the height of the Binary Search Tree.

Example



A Binary Search Tree

Algorithm to search for a key in BST

```
BST_Search(x, k)
if ( x = NIL or k = Value[x] )
        return x;
if ( k < Value[x])
        return BST_Search (left[x], k);
else
        return BST_Search (right[x], k)
```

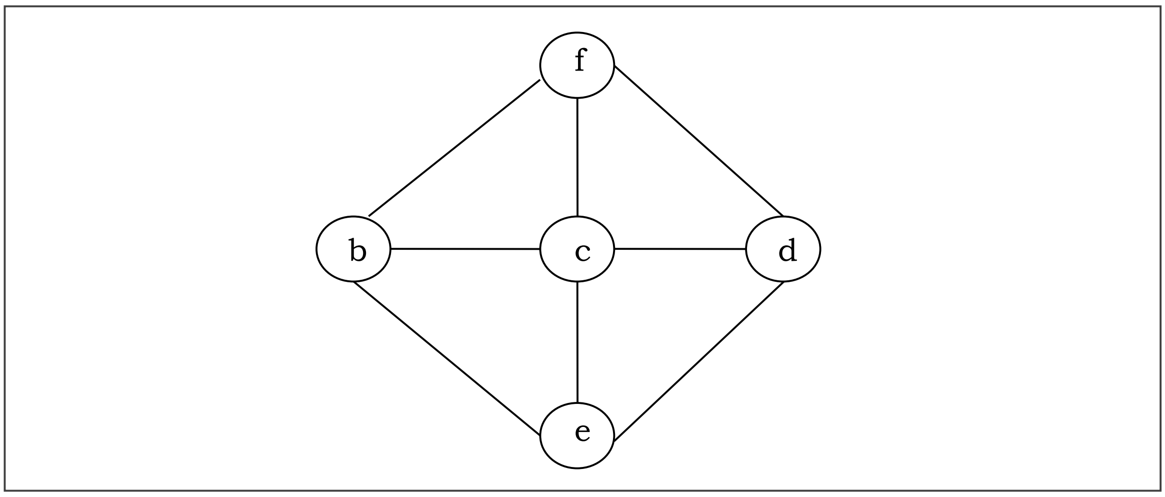
Complexity of Binary Search Tree

	Average Case	**Worst case**
Space Complexity	O(n)	O(n)
Search Complexity	O(log n)	O(n)
Insertion Complexity	O(log n)	O(n)
Deletion Complexity	O(log n)	O(n)

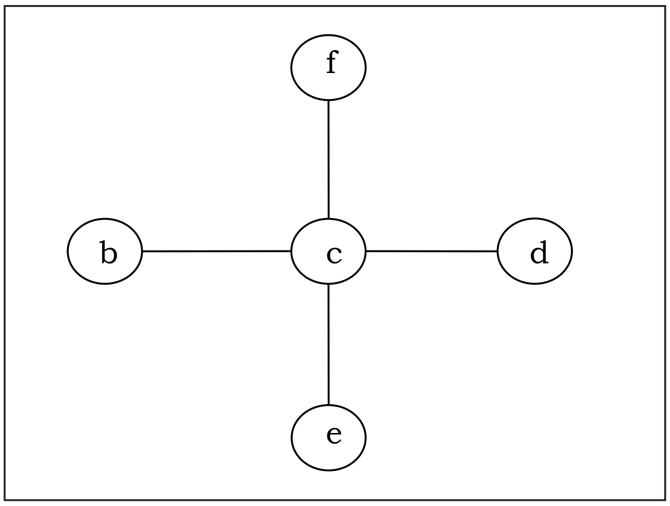
17. Spanning Trees

A spanning tree of a connected undirected graph G is a tree that minimally includes all of the vertices of G. A graph may have many spanning trees.

Example



A Graph G

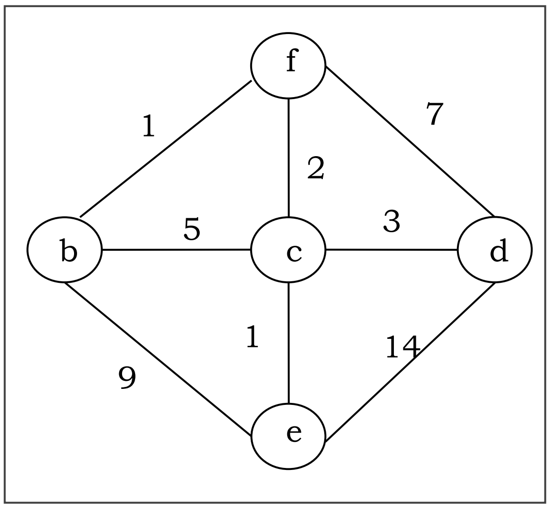


A Spanning Tree of Graph G

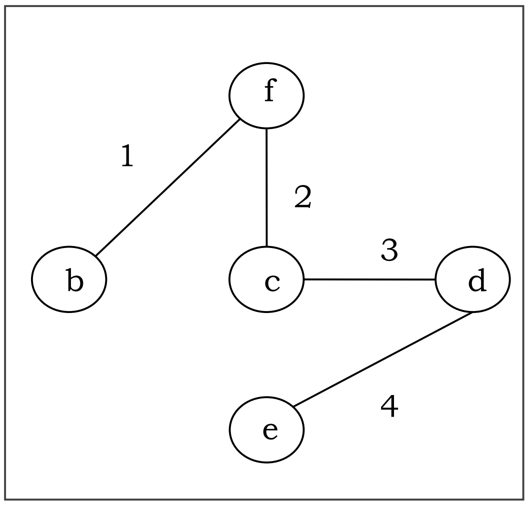
Minimum Spanning Tree

A spanning tree with assigned weight less than or equal to the weight of every possible spanning tree of a weighted, connected and undirected graph G, it is called minimum spanning tree (MST). The weight of a spanning tree is the sum of all the weights assigned to each edge of the spanning tree.

Example



Weighted Graph G



A Minimum Spanning Tree of Graph G

Kruskal's Algorithm

Kruskal's algorithm is a greedy algorithm that finds a minimum spanning tree for a connected weighted graph. It finds a tree of that graph which includes every vertex and the total weight of all the edges in the tree is less than or equal to every possible spanning tree.

Algorithm

Step 1: Arrange all the edges of the given graph G (V,E) in non-decreasing order as per their edge weight.

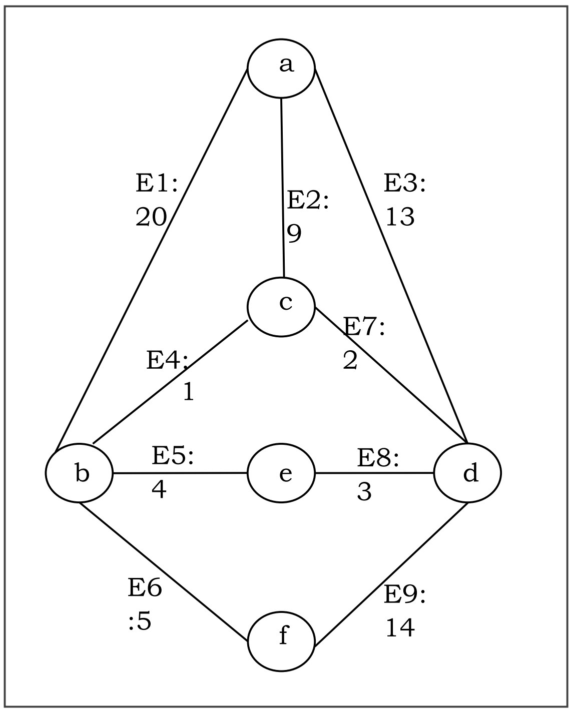
Step 2: Choose the smallest weighted edge from the graph and check if it forms a cycle with the spanning tree formed so far.

Step 3: If there is no cycle, include this edge to the spanning tree else discard it.

Step 4: Repeat Step 2 and Step 3 until (V-1) number of edges are left in the spanning tree.

Problem

Suppose we want to find minimum spanning tree for the following graph G using Kruskal's algorithm.



Weighted Graph G

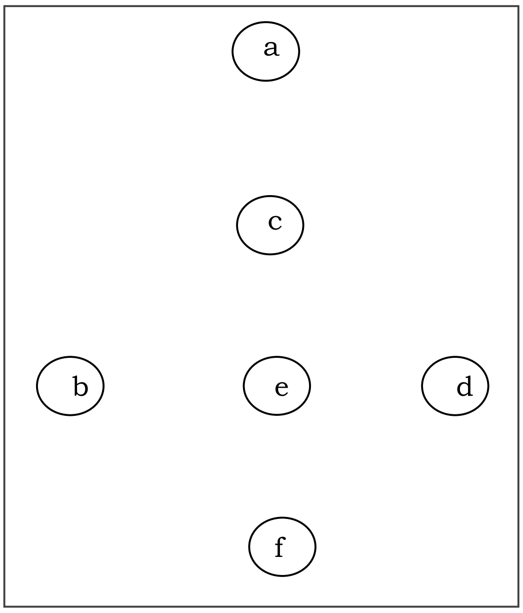
Solution

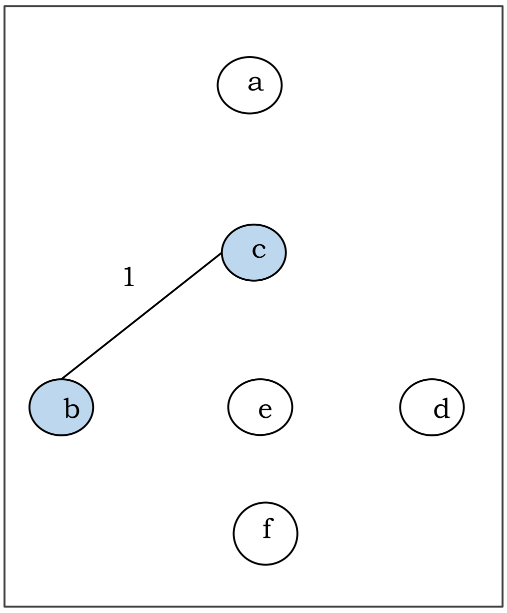
From the above graph we construct the following table:

Edge No.	Vertex Pair	Edge Weight
E1	(a, b)	20
E2	(a, c)	9
E3	(a, d)	13
E4	(b, c)	1
E5	(b, e)	4
E6	(b, f)	5
E7	(c, d)	2
E8	(d, e)	3
E9	(d, f)	14

Now we will rearrange the table in ascending order with respect to Edge weight:

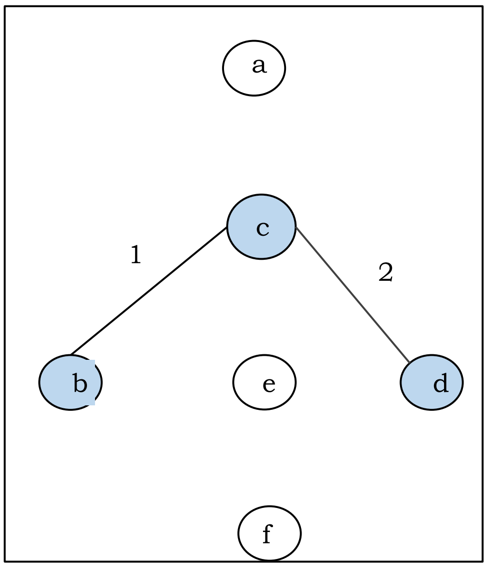
Edge No.	Vertex Pair	Edge Weight
E4	(b, c)	1
E7	(c, d)	2
E8	(d, e)	3
E5	(b, e)	4
E6	(b, f)	5
E2	(a, c)	9
E3	(a, d)	13
E9	(d, f)	14
E1	(a, b)	20



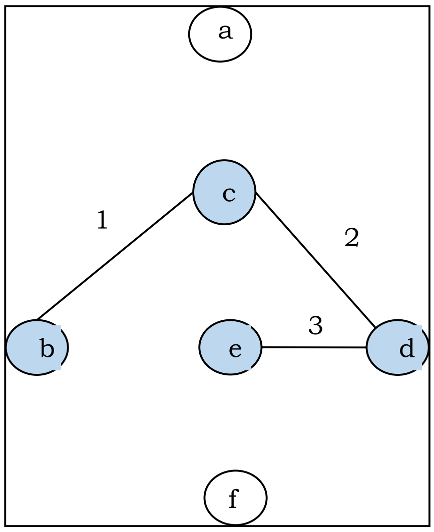


After adding vertices

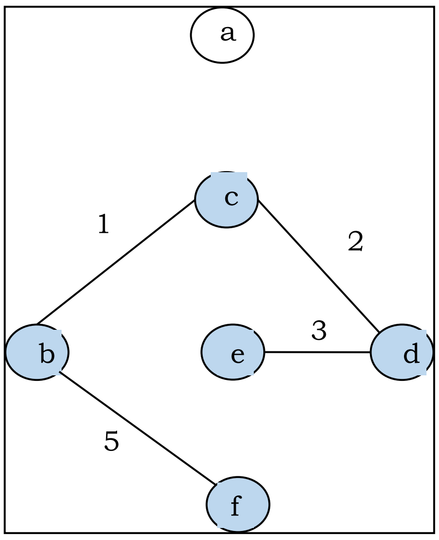
After adding edge E4



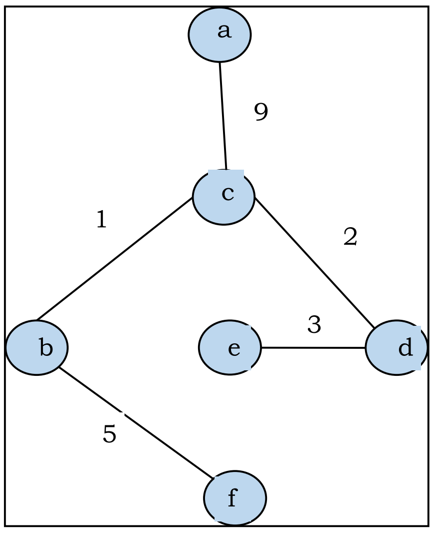
After adding edge E7



After adding edge E8



After adding edge E6
(don’t add E5 since it forms cycle)



After adding edge E2

Since we got all the 5 edges in the last figure, we stop the algorithm and this is the minimal spanning tree and its total weight is (1+2+3+5+9) = 20.

Prim's Algorithm

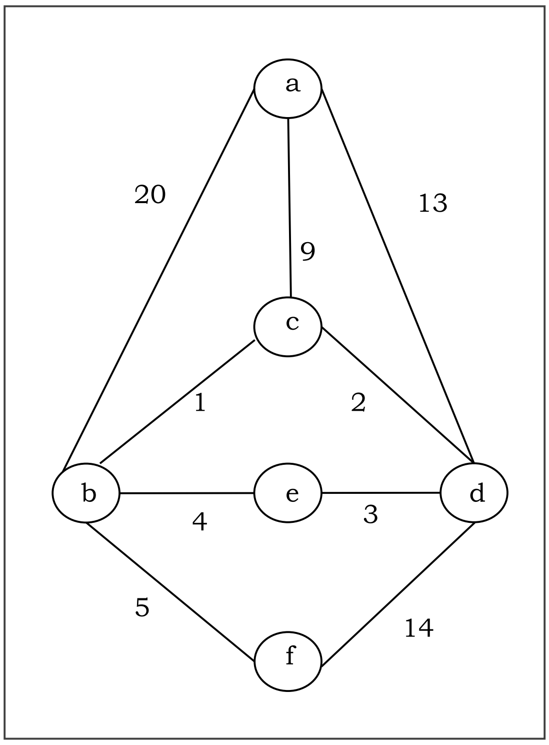
Prim's algorithm, discovered in 1930 by mathematicians, Vojtech Jarnik and Robert C. Prim, is a greedy algorithm that finds a minimum spanning tree for a connected weighted graph. It finds a tree of that graph which includes every vertex and the total weight of all the edges in the tree is less than or equal to every possible spanning tree. Prim's algorithm is faster on dense graphs.

Algorithm

1. Initialize the minimal spanning tree with a single vertex, randomly chosen from the graph.
2. Repeat steps 3 and 4 until all the vertices are included in the tree.
3. Select an edge that connects the tree with a vertex not yet in the tree, so that the weight of the edge is minimal and inclusion of the edge does not form a cycle.
4. Add the selected edge and the vertex that it connects to the tree.

Problem

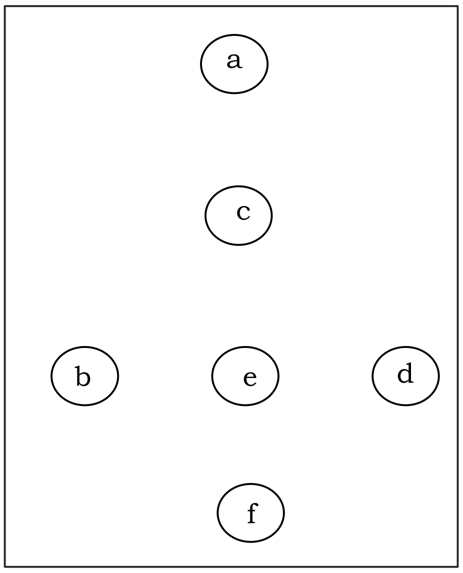
Suppose we want to find minimum spanning tree for the following graph G using Prim's algorithm.



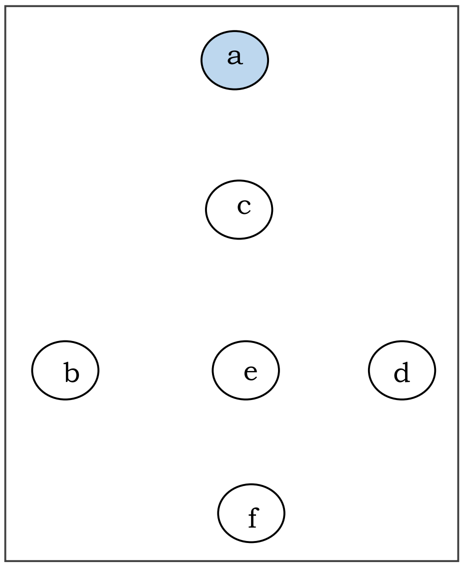
Weighted Graph G

Solution

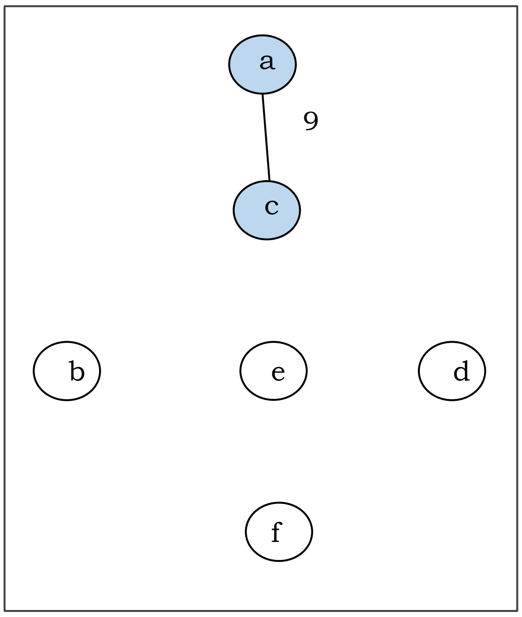
Here we start with the vertex 'a' and proceed.



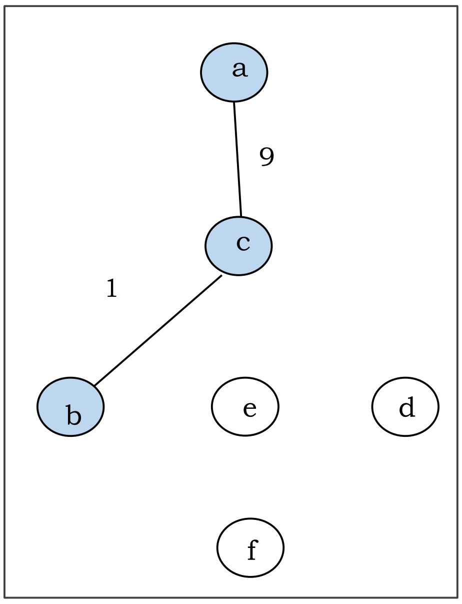
No vertices added



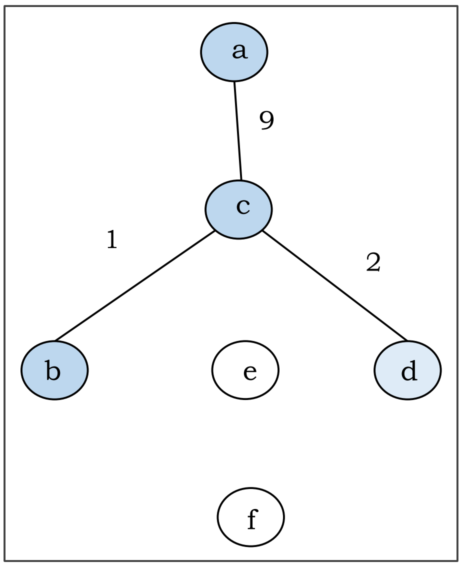
After adding vertex 'a'



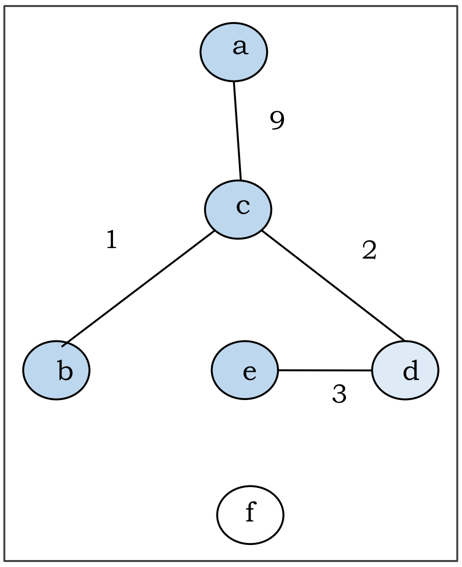
After adding vertex 'c'



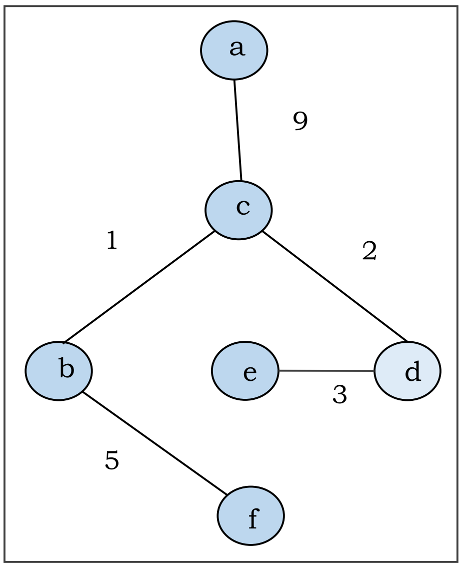
After adding vertex 'b'



After adding vertex 'd'



After adding vertex 'e'



After adding vertex 'f'

This is the minimal spanning tree and its total weight is (1+2+3+5+9) = 20.

Part 7: Boolean Algebra

18. Boolean Expressions and Functions

Boolean algebra is algebra of logic. It deals with variables that can have two discrete values, 0 (False) and 1 (True); and operations that have logical significance. The earliest method of manipulating symbolic logic was invented by George Boole and subsequently came to be known as Boolean Algebra.

Boolean algebra has now become an indispensable tool in computer science for its wide applicability in switching theory, building basic electronic circuits and design of digital computers.

Boolean Functions

A Boolean function is a special kind of mathematical function $f: X^n \rightarrow X$ of degree n, where X = {0, 1} is a Boolean domain and n is a non-negative integer. It describes the way how to derive Boolean output from Boolean inputs.

Example: Let, F(A, B) = A'B'. This is a function of degree 2 from the set of ordered pairs of Boolean variables to the set {0, 1} where F(0, 0) = 1, F(0, 1) = 0, F(1, 0) = 0 and F(1, 1) = 0

Boolean Expressions

A Boolean expression always produces a Boolean value. A Boolean expression is composed of a combination of the Boolean constants (True or False), Boolean variables and logical connectives. Each Boolean expression represents a Boolean function.

Example: AB'C is a Boolean expression.

Boolean Identities

Double Complement Law

~(~A) = A

Complement Law

A + ~A = 1 (OR Form)
A · ~A = 0 (AND Form)

Idempotent Law

A + A = A (OR Form)
A · A = A (AND Form)

Identity Law

A + 0 = A (OR Form)
A · 1 = A (AND Form)

Dominance Law

$A + 1 = 1$ (OR Form)
$A \cdot 0 = 0$ (AND Form)

Commutative Law

$A + B = B + A$ (OR Form)
$A \cdot B = B \cdot A$ (AND Form)

Associative Law

$A + (B + C) = (A + B) + C$ (OR Form)
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$ (AND Form)

Absorption Law

$A \cdot (A + B) = A$
$A + (A \cdot B) = A$

Simplification Law

$A \cdot (\sim A + B) = A \cdot B$
$A + (\sim A \cdot B) = A + B$

Distributive Law

$A + (B \cdot C) = (A + B) \cdot (A + C)$
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$

De-Morgan's Law

$\sim(A \cdot B) = \sim A + \sim B$
$\sim(A + B) = \sim A \cdot \sim B$

Canonical Forms

For a Boolean expression there are two kinds of canonical forms:

1. The sum of minterms (SOM) form
2. The product of maxterms (POM) form

The Sum of Minterms (SOM) or Sum of Products (SOP) form

A minterm is a product of all variables taken either in their direct or complemented form. Any Boolean function can be expressed as a sum of its 1-minterms and the inverse of the function can be expressed as a sum of its 0-minterms. Hence,

F (list of variables) = Σ (list of 1-minterm indices)

and

F′ (list of variables) = Σ (list of 0-minterm indices)

A	B	C	Term	Minterm
0	0	0	x′y′z′	m_0
0	0	1	x′y′z	m_1
0	1	0	x′yz′	m_2
0	1	1	x′yz	m_3
1	0	0	xy′z′	m_4
1	0	1	xy′z	m_5
1	1	0	xyz′	m_6
1	1	1	xyz	m_7

Example

Let, F(x, y, z) = x′ y′ z′ + x y′ z + x y z′ + x y z

Or, F(x, y, z) = $m_0 + m_5 + m_6 + m_7$

Hence,

F(x, y, z) = Σ (0, 5, 6, 7)

Now we will find the complement of F(x, y, z)

F′ (x, y, z) = x′ y z + x′ y′ z + x′ y z′ + x y′ z′

Or, F′(x, y, z) = $m_3 + m_1 + m_2 + m_4$

Hence,

F′(x, y, z) = Σ (3, 1, 2, 4) = Σ (1, 2, 3, 4)

The Product of Maxterms (POM) or Product of Sums (POS) form

A maxterm is addition of all variables taken either in their direct or complemented form. Any Boolean function can be expressed as a product of its 0-maxterms and the inverse of the function can be expressed as a product of its 1-maxterms. Hence,

F (list of variables) = π (list of 0-maxterm indices)

and

F′(list of variables) = π (list of 1-maxterm indices).

A	B	C	Term	Maxterm
0	0	0	x + y + z	M_0
0	0	1	x + y + z′	M_1
0	1	0	x + y′ + z	M_2
0	1	1	x + y′ + z′	M_3
1	0	0	x′ + y + z	M_4
1	0	1	x′ + y + z′	M_5
1	1	0	x′ + y′ + z	M_6
1	1	1	x′ + y′ + z′	M_7

Example

Let, $F(x, y, z) = (x+y+z) \bullet (x+y+z') \bullet (x+y'+z) \bullet (x'+y+z)$

Or, $F(x, y, z) = M_0 \bullet M_1 \bullet M_2 \bullet M_4$

Hence,

$$F(x, y, z) = \pi(0, 1, 2, 4)$$

$$F''(x, y, z) = (x+y'+z') \bullet (x'+y+z') \bullet (x'+y'+z) \bullet (x'+y'+z')$$

Or, $F(x, y, z) = M_3 \bullet M_5 \bullet M_6 \bullet M_7$

Hence,

$$F'(x, y, z) = \pi(3, 5, 6, 7)$$

Logic Gates

Boolean functions are implemented by using logic gates. The following are the logic gates:

NOT Gate

A NOT gate inverts a single bit input to a single bit of output.

A	~A
0	1
1	0

Truth table of NOT Gate

AND Gate

An AND gate is a logic gate that gives a high output only if all its inputs are high, otherwise it gives low output. A dot (.) is used to show the AND operation.

A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

Truth table of AND Gate

OR Gate

An OR gate is a logic gate that gives high output if at least one of the inputs is high. A plus (+) is used to show the OR operation.

A	**B**	**A+B**
0	0	0
0	1	1
1	0	1
1	1	1

Truth table of OR Gate

NAND Gate

A NAND gate is a logic gate that gives a low output only if all its inputs are high, otherwise it gives high output.

A	**B**	**~ (A.B)**
0	0	1
0	1	1
1	0	1
1	1	0

Truth table of NAND Gate

NOR Gate

An NOR gate is a logic gate that gives high output if both the inputs are low, otherwise it gives low output.

A	**B**	**~ (A+B)**
0	0	1
0	1	0
1	0	0
1	1	0

Truth table of NOR Gate

XOR (Exclusive OR) Gate

An XOR gate is a logic gate that gives high output if the inputs are different, otherwise it gives low output.

A	**B**	**A⊕B**
0	0	0
0	1	1
1	0	1
1	1	0

Truth table of XOR Gate

X-NOR (Exclusive NOR) Gate

An EX-NOR gate is a logic gate that gives high output if the inputs are same, otherwise it gives low output.

A	**B**	**A** X-NOR **B**
0	0	1
0	1	0
1	0	0
1	1	1

Truth table of X-NOR Gate

19. Simplification of Boolean Functions

Simplification Using Algebraic Functions

In this approach, one Boolean expression is minimized into an equivalent expression by applying Boolean identities.

Problem 1

Minimize the following Boolean expression using Boolean identities:

F (A, B, C) = A′B+ BC′+ BC+ AB′C′

Solution

Given, F (A, B, C) = A′B+ BC′+ BC+ AB′C′

Or, F (A, B, C) = A′B+ (BC′+ BC′) + BC+ AB′C′

[By idempotent law, BC′ = BC′ + BC′]

Or, F (A, B, C) = A′B+ (BC′+ BC) + (BC′+ AB′C′)

Or, F (A, B, C) = A′B+ B(C′+ C) + C′(B+ AB′)

[By distributive laws]

Or, F (A, B, C) = A′B+ B.1+ C′(B+ A)

[(C′+ C) =1 and absorption law (B+ AB′)= (B+ A)]

Or, F (A, B, C) = A′B+ B+ C′(B+ A)

[B.1 =B]

Or, F (A, B, C) = B(A′+ 1)+ C′(B+ A)

Or, F (A, B, C) = B.1+ C′(B+ A)

[(A′+ 1) =1]

Or, F (A, B, C) = B+ C′(B+ A)

[As, B.1 =B]

Or, F (A, B, C) = B+ BC′ + AC′

Or, F (A, B, C) = B(1+ C′) + AC′

Or, F (A, B, C) = B.1 + AC′

[As, (1+ C′) =1]

Or, F (A, B, C) = B + AC′

[As, B.1 =B]

So, F (A, B, C) = B + AC′ is the minimized form.

Problem 2

Minimize the following Boolean expression using Boolean identities:

F (A, B, C) = (A+B) (B+ C)

Solution

Given,	F (A, B, C) = (A+B) (A+ C)	
Or,	F (A, B, C) = A.A+A.C+B.A+ B.C	[Applying distributive Rule]
Or,	F (A, B, C) = A+A.C+B.A+ B.C	[Applying Idempotent Law]
Or,	F (A, B, C) = A(1+C)+B.A+ B.C	[Applying distributive Law]
Or,	F (A, B, C) = A+B.A+ B.C	[Applying dominance Law]
Or,	F (A, B, C) = (A+1).A+ B.C	[Applying distributive Law]
Or,	F (A, B, C) = 1.A+ B.C	[Applying dominance Law]
Or,	F (A, B, C) = A+ B.C	[Applying dominance Law]

So, F (A, B, C) = A+ BC is the minimized form.

Karnaugh Maps

The Karnaugh map (K–map), introduced by Maurice Karnaughin in 1953, is a grid-like representation of a truth table which is used to simplify boolean algebra expressions. A Karnaugh map has zero and one entries at different positions. It provides grouping together Boolean expressions with common factors and eliminates unwanted variables from the expression. In a K-map, crossing a vertical or horizontal cell boundary is always a change of only one variable.

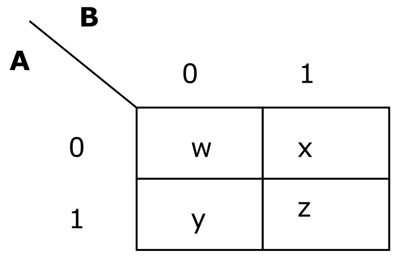
Example 1

An arbitrary truth table is taken below:

A	**B**	**A**operation**B**
0	0	w
0	1	x
1	0	y
1	1	z

Truth table

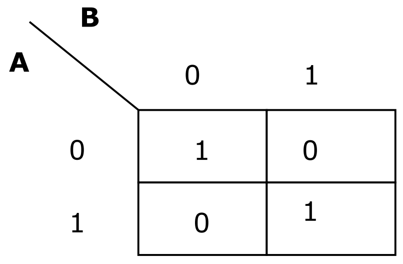
Now we will make a k-map for the above truth table:



K-map

Example 2

Now we will make a K-map for the expression: AB+ A′B′

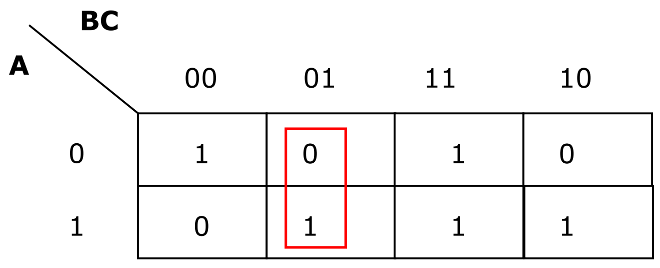


K-map

Simplification Using K- map

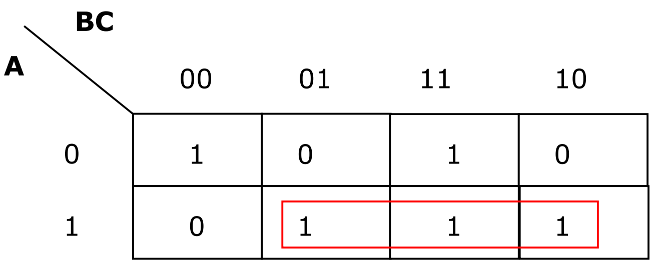
K-map uses some rules for the simplification of Boolean expressions by combining together adjacent cells into single term. The rules are described below:

Rule 1: Any cell containing a zero cannot be grouped.



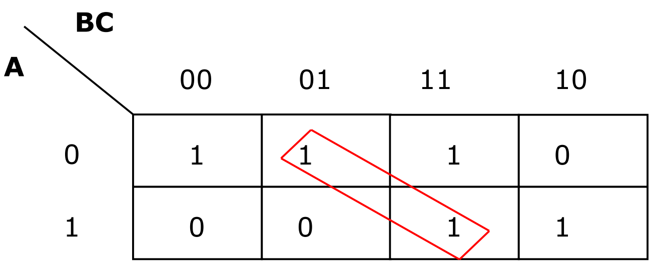
Wrong grouping

Rule 2: Groups must contain 2n cells (n starting from 1).

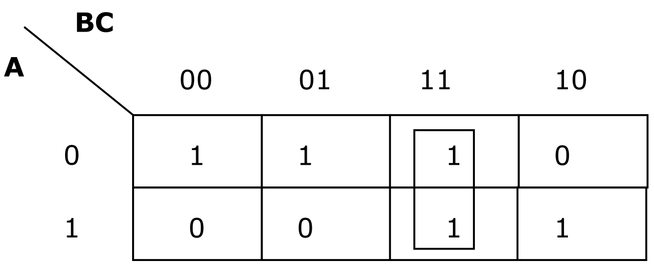


Wrong grouping

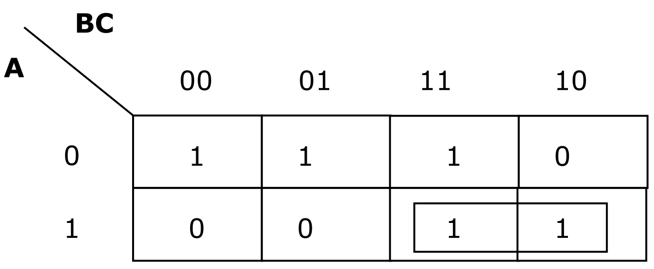
Rule 3: Grouping must be horizontal or vertical, but must not be diagonal.



Wrong diagonal grouping

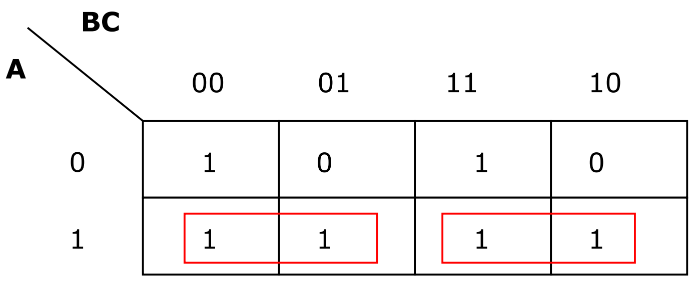


Proper vertical grouping

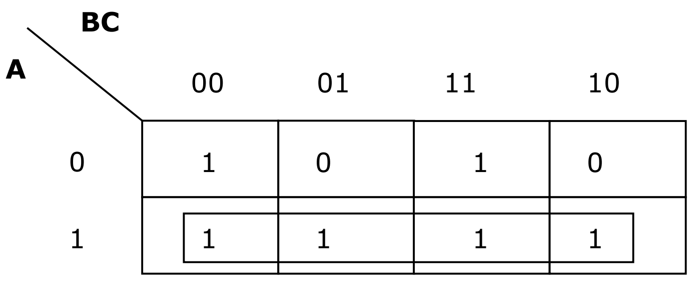


Proper horizontal grouping

Rule 4: Groups must be covered as largely as possible.

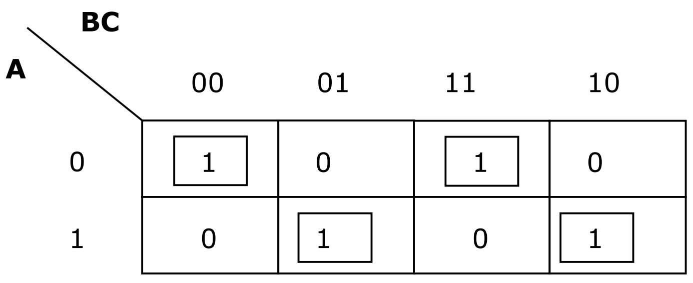


Insufficient grouping



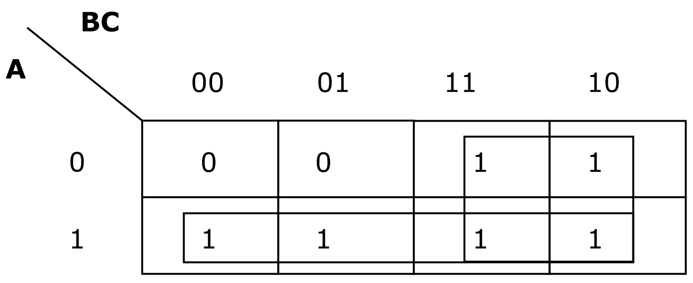
Proper grouping

Rule 5: If 1 of any cell cannot be grouped with any other cell, it will act as a group itself.



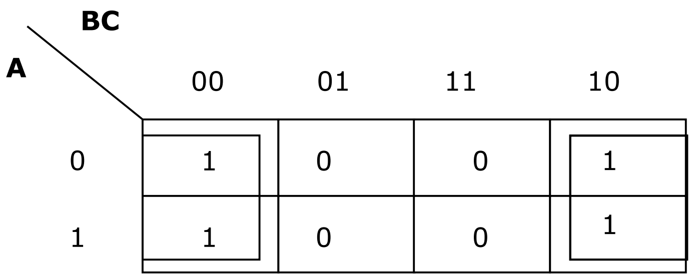
Proper grouping

Rule 6: Groups may overlap but there should be as few groups as possible.



Proper grouping

Rule 7: The leftmost cell/cells can be grouped with the rightmost cell/cells and the topmost cell/cells can be grouped with the bottommost cell/cells.



A \ BC	00	01	11	10
0	1	0	0	1
1	1	0	0	1

Proper grouping

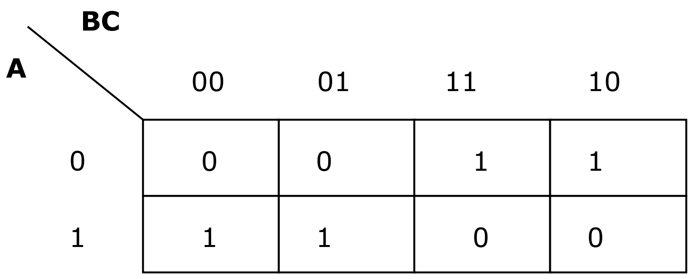
Problem

Minimize the following Boolean expression using K-map:

F (A, B, C) = A’BC+ A’BC’+ AB’C’+ AB’C

Solution

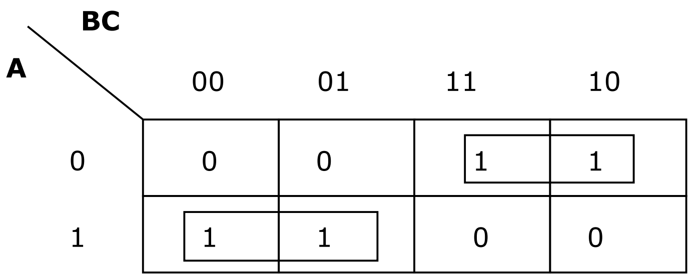
Each term is put into k-map and we get the following:



A \ BC	00	01	11	10
0	0	0	1	1
1	1	1	0	0

K-map for F (A, B, C)

Now we will group the cells of 1 according to the rules stated above:



A \ BC	00	01	11	10
0	0	0	1	1
1	1	1	0	0

K-map for F (A, B, C)

We have got two groups which are termed as A’B and AB’. Hence, F (A, B, C) = A’B+ AB’= A⊕B. It is the minimized form.