MCA 101 : DISCRETE MATHEMATICAL STRUCTURES

UNIT-I:

Function: Definition, type of functions, one to one, into and onto function, inverse function, composition of functions, recursively defined functions. Algebraic Structures: Definition, Properties, types: Semi Groups, Monoid, Groups, Abelian group, properties of groups, Subgroup, cyclic groups, Cosets, factor group, Permutation groups, Normal subgroups.

UNIT-II:

Posets, Hasse Diagram and Lattices: Introduction, ordered set, Hasse diagram of partially, ordered set, isomorphic ordered set, well ordered set, properties of Lattices, and complemented lattices. Combinatorics: Basic Counting Technique, Pigeon-hole Principle, Recurrence Relation, Generating function, Polya's Counting Theorem Paths and Circuits : Isomorphism, Subgraphs, Walks, Paths and Circuits, Connected and disconnected graphs, Euler graphs, Operations on graphs, Hamiltonian graphs, Travelling salesman problem.

UNIT-III:

Introduction and Basic Concepts : Definition, Representation of graphs, Finite and infinite graphs, Directed graphs, Incidence and degree, Bipartite graph, Planar graphs, Matrix representation of graphs, Applications of graph in computer science. Graphs: Simple graph, multi graph, graph terminology, representation of graphs, Bipartite, Regular, Planar and connected graphs, connected components in a graph, Euler graphs, Hamiltonian path and circuits, Graph coloring, chromatic number, isomorphism and Homomorphism of graphs.

Trees and Fundamental Circuits : Definition, Properties of trees, Spanning trees, Fundamental circuits and cut-sets, Connectivity and separability, Minimal spanning tree and connected algorithms, Rooted and Binary trees, Applications of trees.

UNIT-IV:

Tree: Definition, Rooted tree, properties of trees, binary search tree, tree traversal. Shortest Path Problems: Shortest path algorithms, Generalized shortest path algorithms, Applications of shortest path problems.

Network Flow Problems: Flows in network, formulation, Max-flow min-cut theorem, Minimum cost flow problems, Ford-Fulkerson algorithm for maximum flow.

UNIT - V:

Propositional Logic: Proposition, First order logic, Basic logical operation, truth tables, tautologies, Contradictions, Algebra of Proposition, logical implications, logical equivalence, predicates, Universal and existential quantifiers.

Text books

1. Discrete Mathematics and Its Applications, By Kenneth H Rosen, McGraw Hill, Sept.2002.
2. Discrete Mathematical Structures with Applications to Computer Science, By J.P.Tremblay, R.Manohar, McGraw Hill Pub, 1975.
3. "Graph Theory With Applications to Engineering and Computer Science" Prentice Hall,

   Englewood Cliffs, 1974
4. Combinatorics: Theory and Applications, By V. Krishnamurthy, East-West Press Pvt. Ltd.,New Delhi, 1986.
5. 5.K. L. P. Mishra, N. Chandrasekaran, "Theory of Computer Science: Automata,

Lecture Notes

UNIT -1

Function:

Functions are an important part of discrete mathematics. This article is all about functions, their types, and other details of functions. A function assigns exactly one element of a set to each element of the other set. Functions are the rules that assign one input to one output. The function can be represented as f: A ⇢ B. A is called the *domain of the function* and B is called the *codomain function*.

Functions:

- A function assigns exactly one element of one set to each element of other sets.
- A function is a rule that assigns each input exactly one output.
- A function f from A to B is an assignment of exactly one element of B to each element of A (where A and B are non-empty sets).
- A function f from set A to set B is represented as **f: A ⇢ B** where A is called the domain of f and B is called as codomain of f.
- If b is a unique element of B to element a of A assigned by function F then, it is written as f(a) = b.
- Function f maps A to B means f is a function from A to B i.e. f: A ⇢ B

Domain of a function:

- If f is a function from set A to set B then, A is called the domain of function f.
- The set of all inputs for a function is called its domain.

Codomain of a function:

- If f is a function from set A to set B then, B is called the codomain of function f.
- The set of all allowable outputs for a function is called its codomain.

Pre-image and Image of a function:

A function f: A ⇢ B such that for each a ∈ A, there exists a unique b ∈ B such that (a, b) ∈ R then, a is called the pre-image of f and b is called the image of f.

Types of function:

One-One function (or Injective Function):

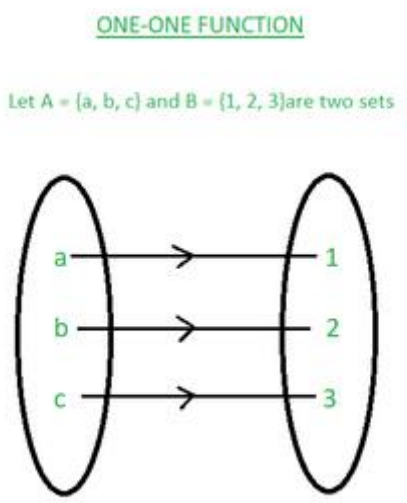
A function in which one element of the domain is connected to one element of the codomain.

A function f: A ⇢ B is said to be a one-one (injective) function if different elements of A have different images in B.

f: A ⇢ B is one-one

⇒ a ≠ b ⇒ f(a) ≠ f(b) for all a, b ∈ A

⇒ f(a) = f(b) ⇒ a = b for all a, b ∈ A



ONE-ONE FUNCTION

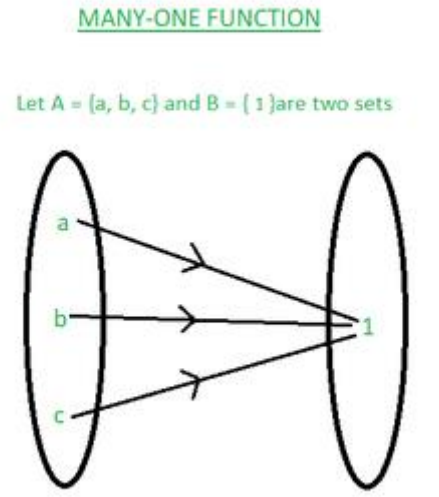
Many-One function:

A function f: A ⇢ B is said to be a many-one function if two or more elements of set A have the same image in B.

A function f: A ⇢ B is a many-one function if it is not a one-one function.

f: A ⇢ B is many-one

⇒ a ≠ b but f(a) = f(b) for all a, b ∈ A



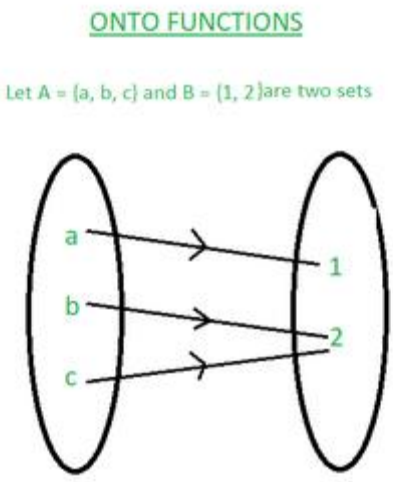
MANY-ONE FUNCTION

Onto function(or Surjective Function):

A function f: A -> B is said to be onto (surjective) function if every element of B is an image of some element of A i.e. f(A) = B or range of f is the codomain of f.

A function in which every element of the codomain has one pre-image.

f: A ⇢ B is onto if for each b∈ B, there exists a∈ A such that f(a) = b.



ONTO FUNCTION

Into Function:

A function f: A ⇢ B is said to be an into a function if there exists an element in B with no pre-image in A.
A function f: A⇢ B is into function when it is not onto.

Recursively defined function

For any recursively defined function, it has two parts. The first part is the definition of the smallest argument and the second part is the definition of the nth term. The smallest argument is usually denoted by f(0) or f(1) and the nth argument is denoted by f(n).

Now, let us understand the recursively defined function with the help of an example.

Let the sequence be 5, 7, 9, 11

The explicit formula for the given sequence is f(n)= 2n+5

The recursive formula for the given sequence is given by

f(0)= 5

f(n) = f(n-1)+2

Now, we can check the sequence terms using the recursive formula as follows:

f(0)= 5

f(1) = f(0)+2

f(1)= 5+2 = 7

f(2) = f(1)+2

f(2)= 7+2 =9

f(3)= f(2)+2

f(3)= 9+2 = 11

In this way, we can find the next term in the sequence with the help of the recursive function formula.

What Makes the Function Recursive?

To make the function recursive, it requires its own term to figure out the next term in the sequence. For example, if we want to find out the nth term in the sequence, we need to know the previous term and also the term before the previous term. Hence, we need to know the previous term to determine whether the sequence is recursive or not recursive. So we can say that, if the function requires the previous term to find the next term in the sequence, then the function is a recursive function. Most of the recursive functions will provide the beginning value of the sequence and the formula that helps to generate the next terms in the sequence.

Recursive Function Formula

If $a_1,a_2,a_3,a_4,\ldots..,a_n,\ldots$ is a set of series or a sequence. Then a **recursive formula** for this sequence will require to compute all the previous terms and find the value of a_n.

i.e. $\mathbf{a_n=a_{n-1}+a_1}$

This formula can also be defined as Arithmetic Sequence Recursive Formula. As you can see from the sequence itself, it is an **Arithmetic sequence**, which consists of the first term followed by other terms and a common difference between each term is the number you add or subtract to them.

A recursive function can also be defined for a **geometric sequence**, where the terms in the sequence have a common factor or common ratio between them. And it can be written as;

$$\mathbf{a_n = r \times a_{n-1}}$$

Generally, the recursive function is defined in two parts. It a statement of the first term along with the formula/ rule related to the successive terms.

.Subgroup

Subgroup –

A nonempty subset H of the group G is a subgroup of G if H is a group under binary operation (*) of G. We use the notation $H \leq G$ to indicate that H is a subgroup of G. Also, if H is a proper subgroup then it is denoted by $H < G$.

For a subset H of group G, H is a subgroup of G if,

- $H \neq \varphi$
- if $a, k \in H$ then $ak \in H$
- if $a \in H$ then $a^{-1} \in H$

Ex. – Integers (Z) is a subgroup of rationals (Q) under addition, $(Z, +) < (Q, +)$

Note:

1. G is a subgroup of itself and {e} is also subgroup of G, these are called trivial subgroup.
2. Subgroup will have all the properties of a group.
3. A subgroup H of the group G is a normal subgroup if $g^{-1} H g = H$ for all $g \in G$.
4. If $H < K$ and $K < G$, then $H < G$ (subgroup transitivity).
5. if H and K are subgroups of a group G then $H \cap K$ is also a subgroup.
6. if H and K are subgroups of a group G then $H \cup K$ is may or maynot be a subgroup.

Coset –

Let H be a subgroup of a group G. If $g \in G$, the right coset of H generated by g is,
$Hg = \{ hg, h \in H \}$;
and similarly, the left coset of H generated by g is $gH = \{ gh, h \in H \}$

Example: Consider Z_4 under addition $(Z_4, +)$, and let $H=\{0, 2\}$. e = 0, e is identity element. Find the left cosets of H in G?

Solution:

The left cosets of H in G are,

$eH = e*H = \{ e * h \mid h \in H\} = \{ 0+h \mid h \in H\} = \{0, 2\}.$

$1H = 1*H = \{1 * h \mid h \in H\} = \{ 1+h \mid h \in H\} = \{1, 3\}.$

2H= 2*H = {2 * h | h ∈ H} = { 2+h| h ∈ H} = {0, 2}.

3H= 3*H = {3 * h |h ∈ H} = { 3+h| h ∈ H} = {1, 3}.

Hence there are two cosets, namely 0*H= 2*H = {0, 2} and 1*H= 3*H = {1, 3}.

Order of Group –

The **Order of a group** (G) is the number of elements present in that group, i.e it's cardinality. It is denoted by |G|.
Order of element a ∈ G is the smallest positive integer n, such that a^n= e, where e denotes the identity element of the group, and a^n denotes the product of n copies of a. If no such n exists, a is said to have infinite order. All elements of finite groups have finite order.
Lagrange's Theorem:
If H is a subgroup of finite group G then the order of subgroup H divides the order of group G.

Properties of the order of an element of the group:

- The order of every element of a finite group is finite.
- The Order of an element of a group is the same as that of its inverse a^{-1}.
- If a is an element of order n and p is prime to n, then a^p is also of order n.
- Order of any integral power of an element b cannot exceed the order of b.
- If the element a of a group G is order n, then a^k=e if and only if n is a divisor of k.
- The order of the elements a and $x^{-1}ax$ is the same where a, x are any two elements of a group.
- If a and b are elements of a group then the order of **ab** is same as order of **ba**

4.Explain Abelian group?
An abelian group is a group in which the law of composition is commutative, i.e. the group law ∘ satisfies. g∘h=h∘g. for any g,h in the group.
For example, the set of integers , with group operation of addition. ie, 1+2 = 2+1
Problem-: Prove that (I, +) is an abelian group. i.e. The set of all integers I form an abelian group with respect to binary operation '+'.
Solution-:
Set= I ={ ……..-3, -2 , -1 , 0, 1, 2 , 3…… }.

Binary Operation= '+'

Algebraic Structure= (I ,+)

We have to prove that (I,+) is an abelian group.

To prove that set of integers I is an abelian group we must satisfy the following five properties that is Closure Property, Associative Property, Identity Property, Inverse Property, and Commutative Property.

1) Closure Property

$\forall a, b \in I \Rightarrow a + b \in I$

$2,-3 \in I \Rightarrow -1 \in I$

Hence Closure Property is satisfied.

2) *Associative Property*

$(a+b)+c = a+(b+c) \ \forall a, b, c \in I$

$2 \in I, -6 \in I, 8 \in I$

$So, LHS = (a+b)+c$

$= (2+(-6))+8 = 4$

$RHS = a+(b+c)$

$= 2+(-6+8) = 4$

$Hence\ RHS = LHS$

Associative Property is also Satisfied

3) Identity Property

$a + 0 = a \ \forall a \in I, 0 \in I$

$5 \in I$

$5+0 = 5$

$-17 \in I$

$-17 + 0 = -17$

Identity property is also satisfied.

4) Inverse Property

$a + (-a) = 0 \ \forall a \in I, -a \in I, 0 \in I$

$a=18 \in I \text{ then } \ni \text{ a number } -18 \text{ such that } 18 + (-18) = 0$

So, Inverse property is also satisfied.

5) Commutative Property

$a + b = b + a \ \forall a, b \in I$

$Let\ a=19, b=20$

$LHS = a + b$

$= 19+(-20) = -1$

$RHS = b + a$

$= -20 +19 = -1$

$LSH=RHS$

Commutative Property is also satisfied.

5.what are cyclic groups?

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 $(i)^{-1}=-i$ $(i)-1=-i$ is not in H3

• **Short answers**

(1), Define the functions and types of Functions?

(2), Explain the subgroups?

(3),Show the cosets and factors?

(4), Evaluate the Algebraic structure?

•Long Answers

(1), Describe the Functions and Types also their diagrams?

(2), Explain the subgroups, cyclic groups, permutations?

(3),show that Algebraic structure and function of groups?

(4), Explain the Factors and Recursively defined structures?

(5), Composition of functions and structures?

•Lecutre Notes

UNIT-2

well ordered set

1. **Well Ordered Set :**
   A partially ordered set is called a Well Ordered set if every non-empty subset has a least element.
   Example –
   A set of natural number and less than operation ([N, ≤]) then it is a

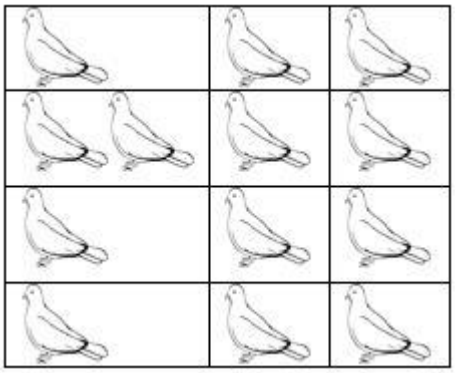
Well ordered Set because firstly it is a POSET and if we take any two natural numbers e.g. n1 and n2 where n1≤n2. Here, n1 is the least element. So, it is a Well Ordered Set.

Note –

- Any Well ordered set is totally ordered.
- Every subset of a Well ordered set is Well-ordered with the same ordering.

pigeon-hole principle:

Suppose that a flock of 20 pigeons flies into a set of 19 pigeonholes to roost. Because there are 20 pigeons but only 19 pigeonholes, a least one of these 19 pigeonholes must have at least two pigeons in it. To see why this is true, note that if each pigeonhole had at most one pigeon in it, at most 19 pigeons, one per hole, could be accommodated. This illustrates a general principle called the pigeonhole principle, which states that if there are more pigeons than pigeonholes, then there must be at least one pigeonhole with at least two pigeons in it.



Theorem –

I) If "A" is the average number of pigeons per hole, where A is not an integer then

- At least one pigeon hole contains **ceil[A]** (smallest integer greater than or equal to A) pigeons
- Remaining pigeon holes contains at most **floor[A]** (largest integer less than or equal to A) pigeons

Or

II) We can say as, if n + 1 objects are put into n boxes, then at least one box contains two or more objects.

The abstract formulation of the principle: Let X and Y be finite sets and

let be a function.

- If X has more elements than Y, then f is not one-to-one.
- If X and Y have the same number of elements and f is onto, then f is one-to-one.

- If X and Y have the same number of elements and f is one-to-one, then f is onto.

Pigeonhole principle is one of the simplest but most useful ideas in mathematics. We will see more applications that proof of this theorem.

- **Example – 1:** If (Kn+1) pigeons are kept in n pigeon holes where K is a positive integer, what is the average no. of pigeons per pigeon hole?
  Solution: average number of pigeons per hole = (Kn+1)/n
  = K + 1/n
  Therefore there will be at least one pigeonhole which will contain at least (K+1) pigeons i.e., ceil[K +1/n] and remaining will contain at most K i.e., floor[k+1/n] pigeons.
  i.e., the minimum number of pigeons required to ensure that at least one pigeon hole contains (K+1) pigeons is (Kn+1).
- **Example – 2:** A bag contains 10 red marbles, 10 white marbles, and 10 blue marbles. What is the minimum no. of marbles you have to choose randomly from the bag to ensure that we get 4 marbles of same color?
  Solution: Apply pigeonhole principle.
  No. of colors (pigeonholes) n = 3
  No. of marbles (pigeons) K+1 = 4
  Therefore the minimum no. of marbles required = Kn+1
  By simplifying we get Kn+1 = 10.
  Verification: ceil[Average] is [Kn+1/n] = 4
  [Kn+1/3] = 4
  Kn+1 = 10
  i.e., 3 red + 3 white + 3 blue + 1(red or white or blue) = 10

Pigeonhole principle strong form –

Theorem: Let $q_1, q_2, \ldots, q_n$ be positive integers.
If $q_1 + q_2 + \ldots + q_n - n + 1$ objects are put into n boxes, then either the 1st box contains at least q_1 objects, or the 2nd box contains at least q_2 objects, . . ., the nth box contains at least q_n objects.

Application of this theorem is more important, so let us see how we apply this theorem in problem solving.

- **Example – 1:** In a computer science department, a student club can be formed with either 10 members from first year or 8 members from second year or 6 from third year or 4 from final year. What is the minimum no. of students we have to choose randomly from department to ensure that a student club is formed?
  Solution: we can directly apply from the above formula where,
  $q_1 = 10$, $q_2 = 8$, $q_3 = 6$, $q_4 = 4$ and n=4
  Therefore the minimum number of students required to ensure department club to be formed is
  10 + 8 + 6 + 4 – 4 + 1 = 25

- **Example – 2:** A box contains 6 red, 8 green, 10 blue, 12 yellow and 15 white balls. What is the minimum no. of balls we have to choose randomly from the box to ensure that we get 9 balls of same color?
  Solution: Here in this we cannot blindly apply pigeon principle. First we will see what happens if we apply above formula directly.
  From the above formula we have get answer 47 because 6 + 8 + 10 + 12 + 15- 5 + 1 = 47
  But it is not correct. In order to get the correct answer we need to include only blue, yellow and white balls because red and green balls are less than 9. But we are picking randomly so we include after we apply pigeon principle.
  i.e., 9 blue + 9 yellow + 9 white – 3 + 1 = 25
  Since we are picking randomly so we can get all the red and green balls before the above 25 balls. Therefore we add 6 red + 8 green + 25 = 39
  We can conclude that in order to pick 9 balls of same color randomly, one has to pick 39 balls from a box.

. Isomorphism:

isomorphism, in modern algebra, a one-to-one correspondence (mapping) between two sets that preserves binary relationships between elements of the sets. For example, the set of natural numbers can be mapped onto the set of even natural numbers by multiplying each natural number by 2. The binary operation of adding two numbers is preserved—that is, adding two natural numbers and then multiplying the sum by 2 gives the same result as multiplying each natural number by 2 and then adding the products together—so the sets are isomorphic for addition.

In symbols, let A and B be sets with elements a_n and b_m, respectively. Furthermore, let $\oplus$ and $\otimes$ indicate their respective binary operations, which operate on any two elements from a set and may be different. If there exists a mapping f such that $f(a_j \oplus a_k) = f(a_j) \otimes f(a_k)$ and its inverse mapping f^{-1} such that $f^{-1}(b_r \otimes b_s) = f^{-1}(b_r) \oplus f^{-1}(b_s)$, then the sets are isomorphic and f and its inverse are isomorphisms. If the sets A and B are the same, f is called an automorphism.

Because an isomorphism preserves some structural aspect of a set or mathematical group, it is often used to map a complicated set onto a simpler or better-known set in order to establish the original set's properties. Isomorphisms are one of the subjects studied in group theory.

Operations:

Graph Operations – Extracting sub graphs

In this section we will discuss about various types of sub graphs we can extract from a given Graph.

Sub graph

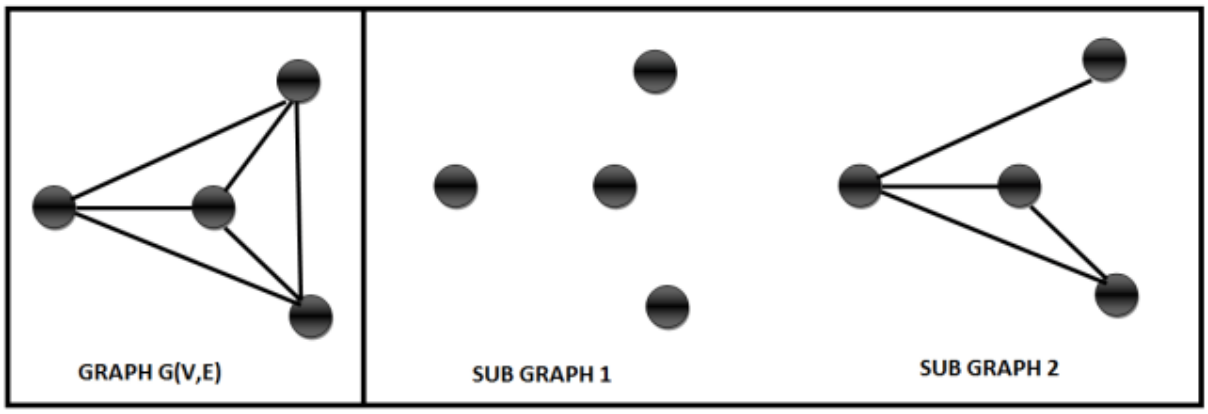
Getting a sub graph out of a graph is an interesting operation. A sub graph of a graph G(V,E) can be obtained by the following means:

- Removing one or more vertices from the vertex set.
- Removing one or more edges from the edge family.
- Removing either vertices or edges from the graph.

Points worth noting

- The vertices of sub graphs are subsets of the original vertices
- The edges of sub graphs are subsets of the original edges

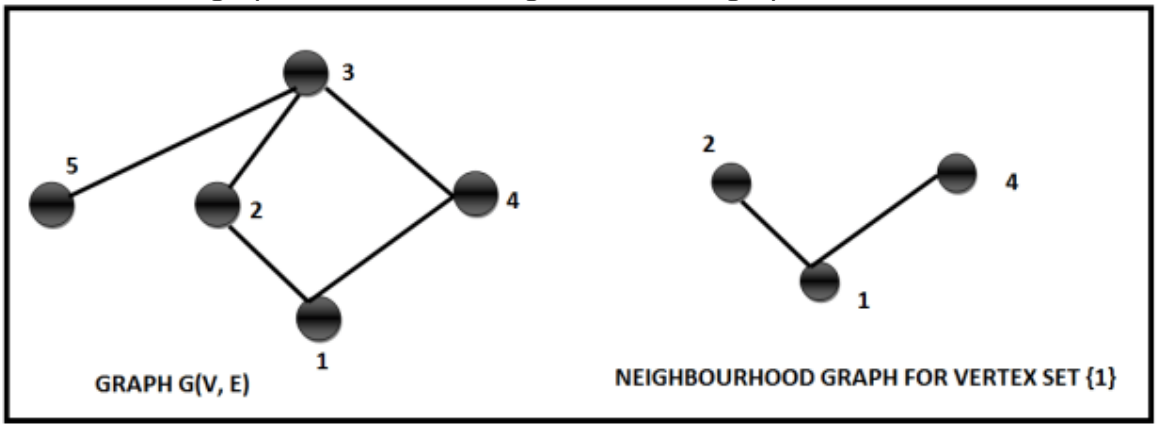
We can extract sub graphs for simple graphs, directed graphs, multi edge graphs and all types of graphs.
The Null Graph is always a sub graph of all the graphs. There can be many sub graphs for a graph.
Below is a graph and its sub graphs.



Neighbourhood graph

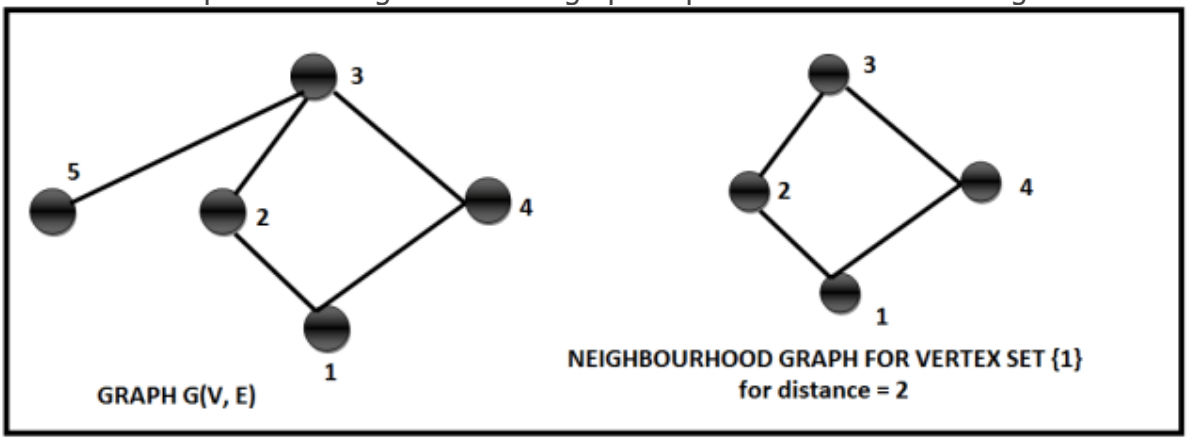
The neighbourhood graph of a graph G(V,E) only makes sense when we mention it with respect to a given vertex set. For e.g. if V = {1,2,3,4,5} then we can find out the Neighbourhood graph of G(V,E) for vertex set {1}.

So, the neighbourhood graphs contains the vertices 1 and all the edges incident on them and the vertices connected to these edges.
Below is a graph and its neighbourhood graphs as described above.



We can also extract neighbourhood graph for a distance more than 1, which means that

we do not stop at the first neighbour and also extend it to second, third and more. Another example for neighbourhood graph up to distance 2 is given below:



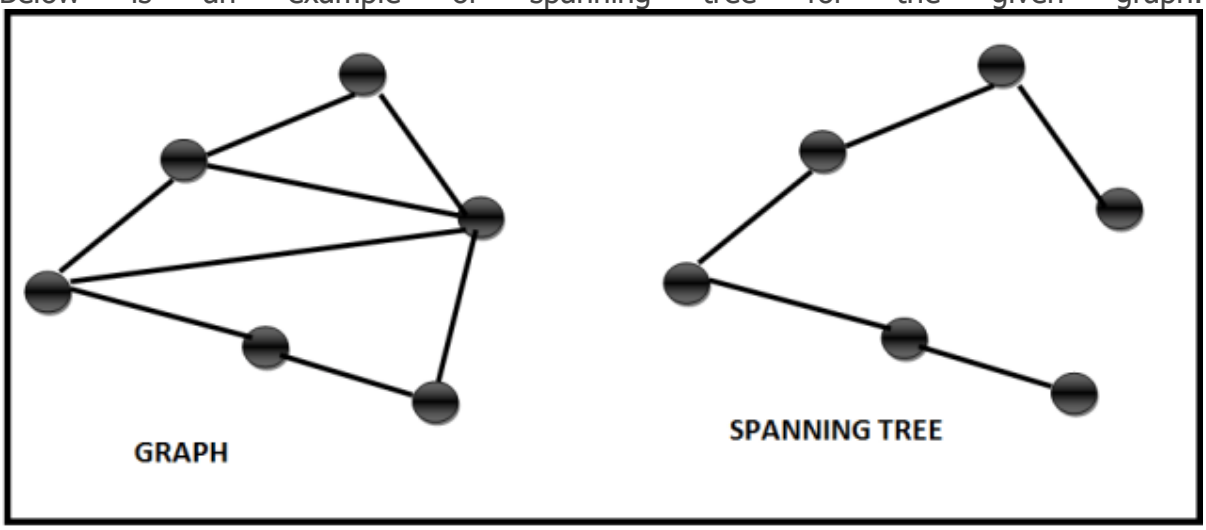
Spanning Tree

A spanning tree of a connected graph G(V,E) is a sub graph that is also a tree and connects all vertices in V. For a disconnected graph the spanning tree would be the spanning tree of each component respectively.

There is an interesting set of problems related to finding the minimum spanning tree (which we will be discussing in upcoming posts). There are many algorithms available to solve this problem, for e.g.: Kruskal's, Prim's etc. Note that the concept of minimum spanning tree mostly makes sense in case of weighted graphs. If the graph is not weighted, we consider all the weights to be one and any spanning tree becomes the minimum spanning tree.

We can use traversals like Breadth First Scan and Depth First Scan to fight the Spanning Tree. We can find spanning tree for undirected graphs, directed graphs, multi graphs as well.

Below is an example of spanning tree for the given graph.

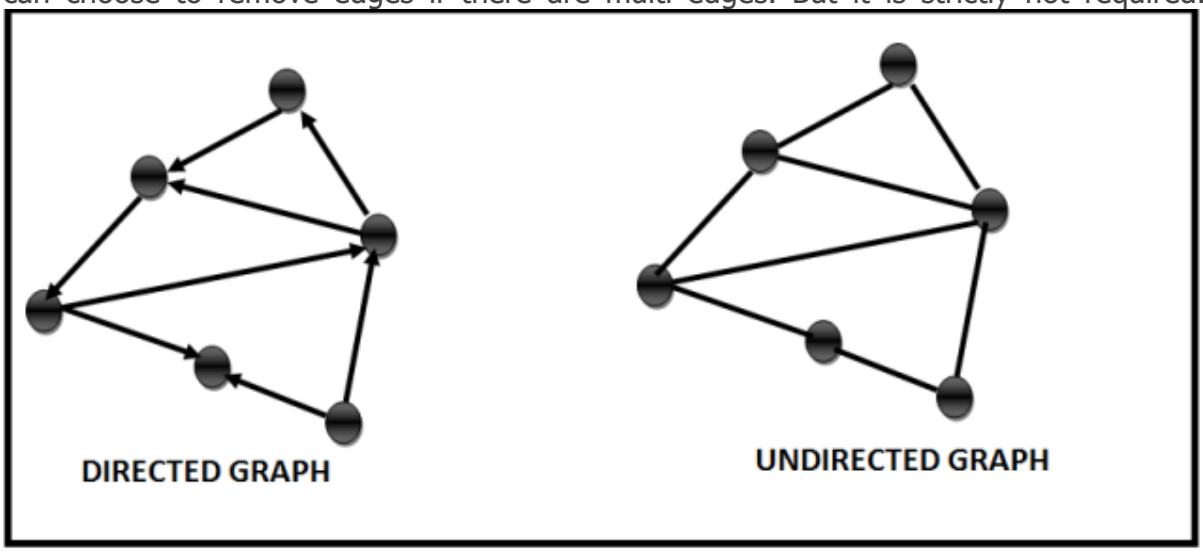


Graph Operations – Conversions of Graphs

In this section we discuss about converting one graph into another graph. Which means all the graphs can be converted into any of the below forms.

Conversion from Directed Graph to Undirected graph

This is the simplest conversions possible. A directed graph has directions represented by arrows, in this conversion we just remove all the arrows and do not store the direction information. Below is an example of the conversion.
Please note that the graph remains unchanged in terms of its structure. However, we can choose to remove edges if there are multi edges. But it is strictly not required.



Conversion from Undirected Graph to Directed graph

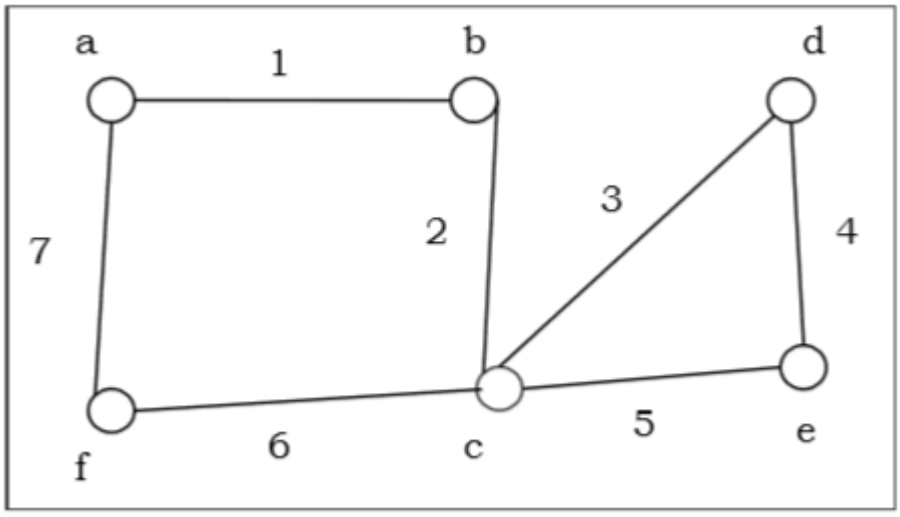
This conversion gives a directed graph given an undirected graph G(V,E). It is the exact reverse of the above. The trick to achieve this is to add one edge for each existing edge in the edge family E. Once the extra edges are added, we just assign opposite direction to each pair of edges between connecting vertices.

. Euler Graphs:

Euler Graph - A connected graph G is called an Euler graph, if there is a closed trail which includes every edge of the graph G.

Euler Path - An Euler path is a path that uses every edge of a graph exactly once. An Euler path starts and ends at different vertices.

Euler Circuit - 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.



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.

•Short Answers

(1), Explain the Lacctices?

(2), Hamilton graph coloring and salesman ?

(3), Define Pigeon hole principal?

(4), Explain the counting therom paths?

Long Answers

(1), Explain the subgroups of posets?

(2),show the circuit and path of the Isomorphism graphs?

(3), Explain the Eluer graph?

(4), Describe the polay's counting therom?

(5), Describe the Recurrence relations?

Lecture Notes

UNIT-3

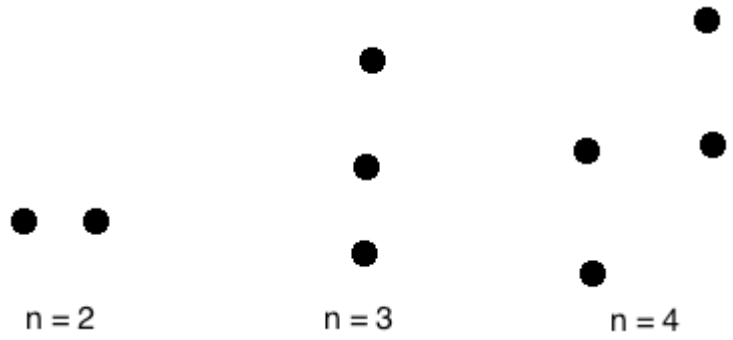
. Types of Graphs

Though, there are a lot of different types of graphs depending upon the number of vertices, number of edges, interconnectivity, and their overall structure, some of such common types of graphs are as follows:

1. Null Graph

A **null graph** is a graph in which there are no edges between its vertices. A null graph is also called empty graph.

Example



A null graph with n vertices is denoted by Nn.

---

2. Trivial Graph

A **trivial graph** is the graph which has only one vertex.

Example

In the above graph, there is only one vertex 'v' without any edge. Therefore, it is a trivial graph.

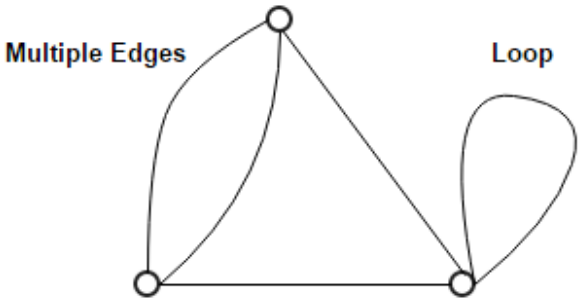
---

3. Simple Graph

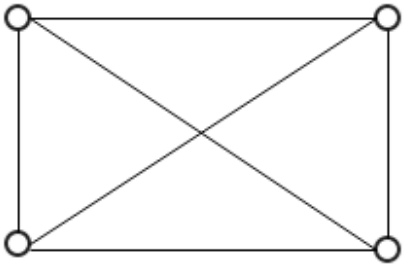
A **simple graph** is the undirected graph with **no parallel edges** and **no loops**.

A simple graph which has n vertices, the degree of every vertex is at most n -1.

Example



Not a Simple Graph



Simple Graph

In the above example, First graph is not a simple graph because it has two edges between the vertices A and B and it also has a loop.

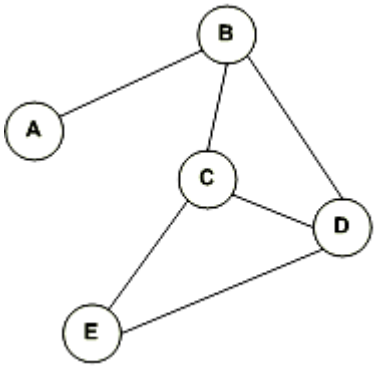
Second graph is a simple graph because it does not contain any loop and parallel edges.

---

4. Undirected Graph

An **undirected graph** is a graph whose edges are **not directed**.

Example



In the above graph since there is no directed edges, therefore it is an undirected graph.

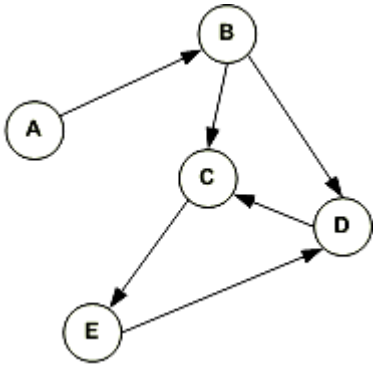
---

5. Directed Graph

A **directed graph** is a graph in which the **edges are directed** by arrows.

Directed graph is also known as **digraphs**.

Example



In the above graph, each edge is directed by the arrow. A directed edge has an arrow from A to B, means A is related to B, but B is not related to A.

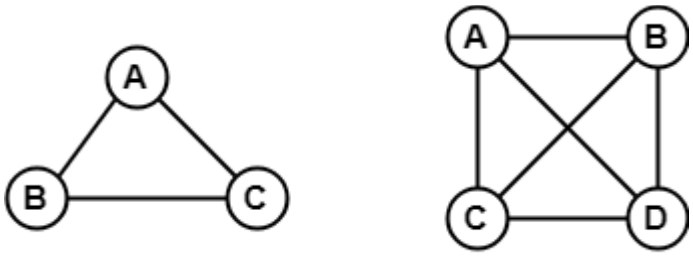
---

6. Complete Graph

A graph in which every pair of vertices is joined by exactly one edge is called **complete graph**. It contains all possible edges.

A complete graph with n vertices contains exactly nC2 edges and is represented by Kn.

Example



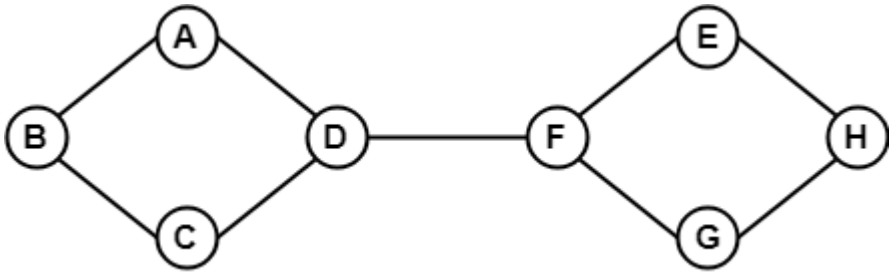
In the above example, since each vertex in the graph is connected with all the remaining vertices through exactly one edge therefore, both graphs are complete graph.

---

7. Connected Graph

A **connected graph** is a graph in which we can visit from any one vertex to any other vertex. In a connected graph, at least one edge or path exists between every pair of vertices.

Example



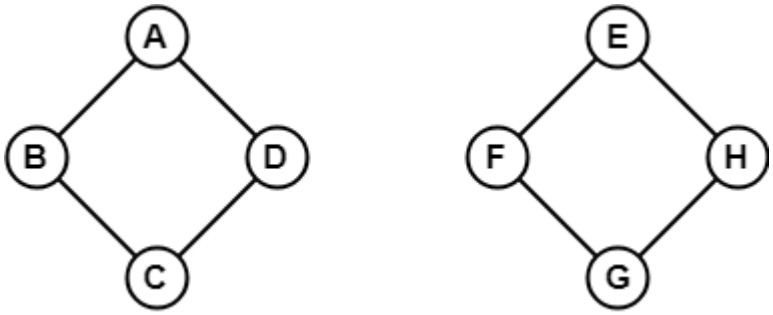
In the above example, we can traverse from any one vertex to any other vertex. It means there exists at least one path between every pair of vertices therefore, it a connected graph.

---

8. Disconnected Graph

A **disconnected graph** is a graph in which any path does not exist between every pair of vertices.

Example



The above graph consists of two independent components which are disconnected. Since it is not possible to visit from the vertices of one component to the vertices of other components therefore, it is a disconnected graph.

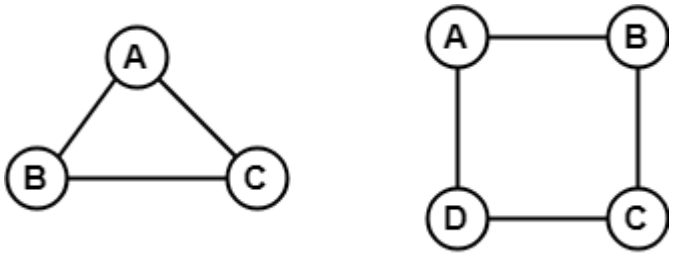
---

9. Regular Graph

A **Regular graph** is a graph in which degree of all the vertices is same.

If the degree of all the vertices is k, then it is called k-regular graph.

Example



In the above example, all the vertices have degree 2. Therefore they are called 2-**Regular graph**.

---

10. Cyclic Graph

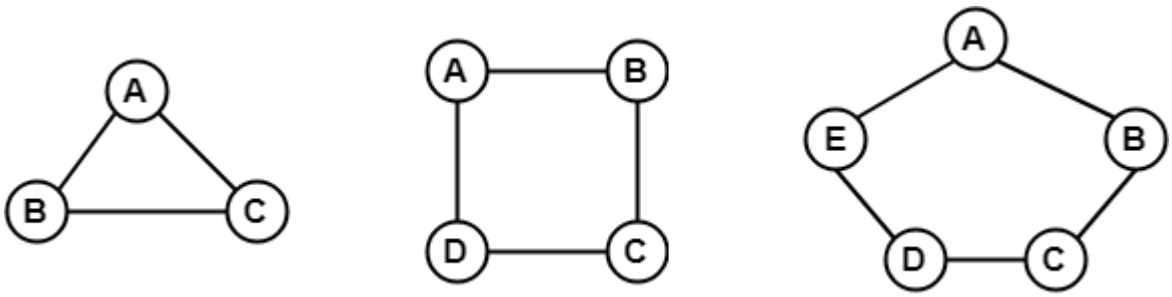
A graph with 'n' vertices (where, n>=3) and 'n' edges forming a cycle of 'n' with all its edges is known as **cycle graph**.

A graph containing at least one cycle in it is known as a **cyclic graph**.

In the cycle graph, degree of each vertex is 2.

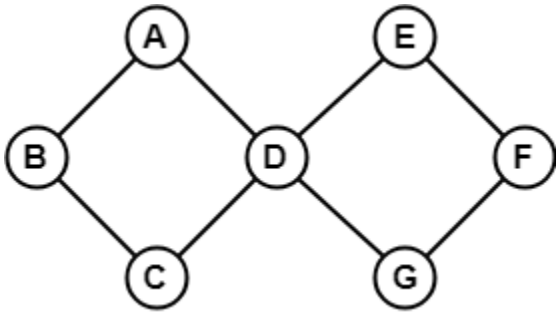
The cycle graph which has n vertices is denoted by Cn.

Example 1



In the above example, all the vertices have degree 2. Therefore they all are cyclic graphs.

Example 2



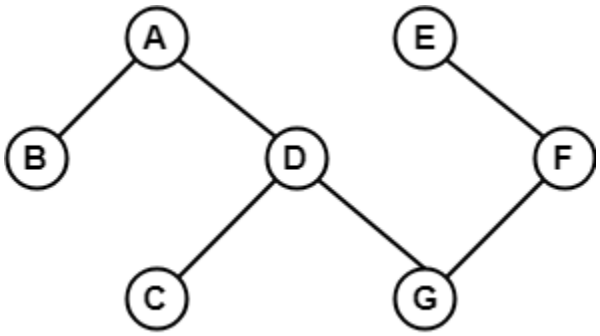
Since, the above graph contains two cycles in it therefore, it is a cyclic graph.

---

11. Acyclic Graph

A graph which does not contain any cycle in it is called as an **acyclic graph**.

Example



Since, the above graph does not contain any cycle in it therefore, it is an acyclic graph.

---

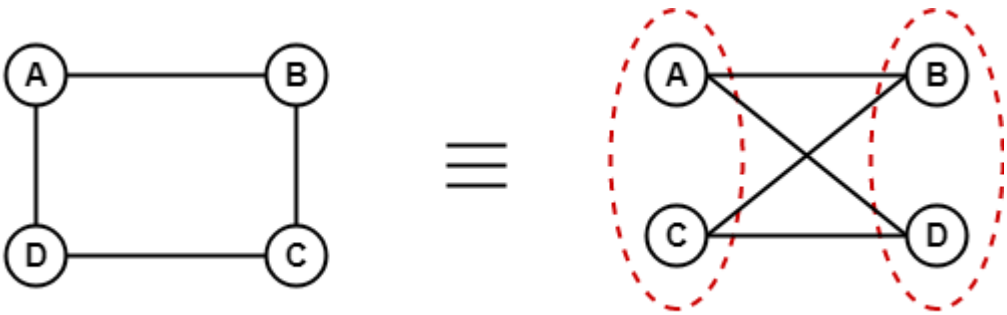
12. Bipartite Graph

A **bipartite graph** is a graph in which the vertex set can be partitioned into two sets such that edges only go between sets, not within them.

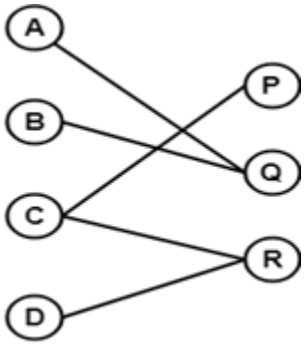
A graph G (V, E) is called bipartite graph if its vertex-set V(G) can be decomposed into two non-empty disjoint subsets V1(G) and V2(G) in such a way that each edge e ∈ E(G) has its one last joint in V1(G) and other last point in V2(G).

The partition V = V1 ∪ V2 is known as bipartition of G.

Example 1



Example 2



---

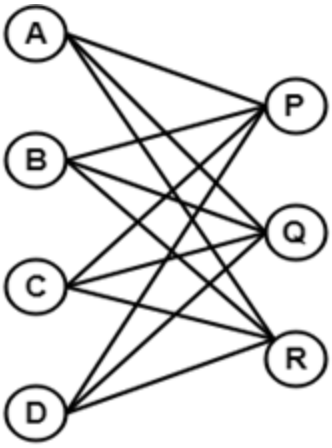
13. Complete Bipartite Graph

A **complete bipartite graph** is a bipartite graph in which each vertex in the first set is joined to each vertex in the second set by exactly one edge.

A complete bipartite graph is a bipartite graph which is complete.

1. Complete Bipartite graph = Bipartite graph+complete graph

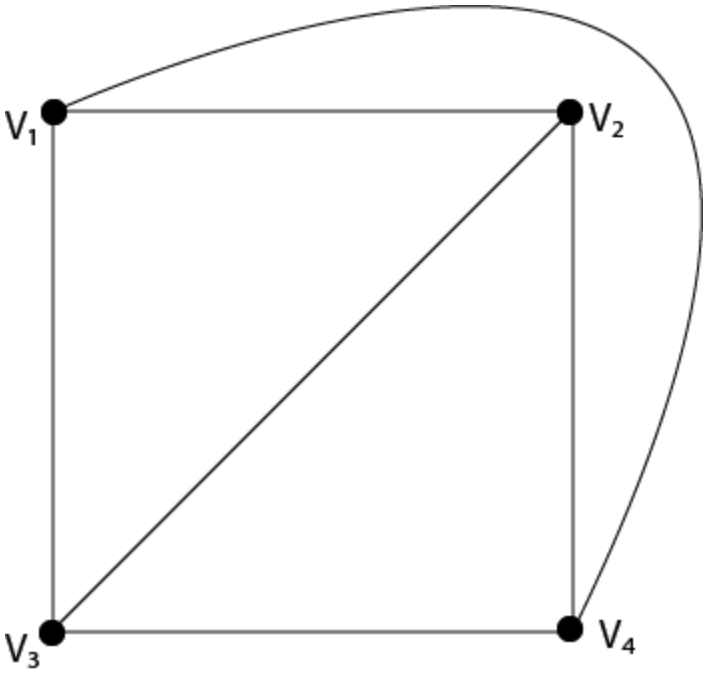
Example

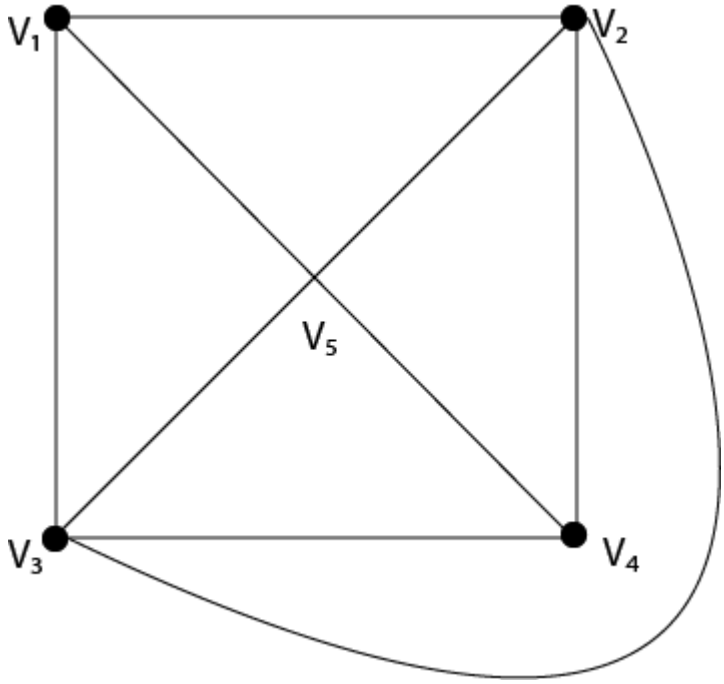


- planar graph:

A graph is said to be planar if it can be drawn in a plane so that no edge cross.

Example: The graph shown in fig is planar graph.



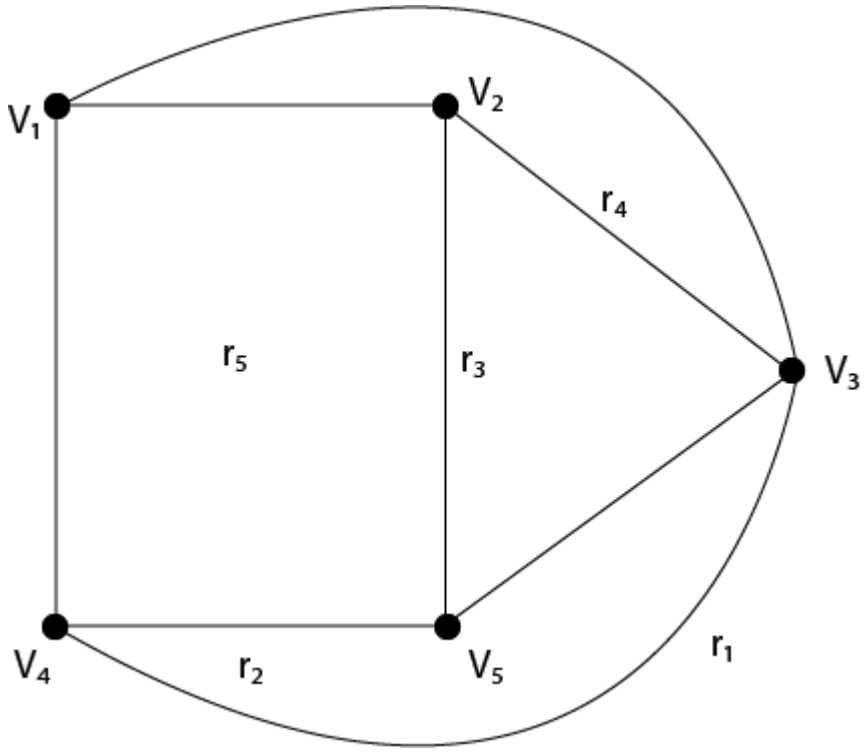


Region of a Graph: Consider a planar graph G=(V,E).A region is defined to be an area of the plane that is bounded by edges and cannot be further subdivided. A planar graph divides the plans into one or more regions. One of these regions will be infinite.

Finite Region: If the area of the region is finite, then that region is called a finite region.

Infinite Region: If the area of the region is infinite, that region is called a infinite region. A planar graph has only one infinite region.

Example: Consider the graph shown in Fig. Determine the number of regions, finite regions and an infinite region.



Solution: There are five regions in the above graph, i.e. r_1,r_2,r_3,r_4,r_5.

There are four finite regions in the graph, i.e., r_2,r_3,r_4,r_5.

There is only one finite region, i.e., r_1

Properties of Planar Graphs:

1. If a connected planar graph G has e edges and r regions, then $r \leq \frac{2}{3} e$.
2. If a connected planar graph G has e edges, v vertices, and r regions, then v-e+r=2.
3. If a connected planar graph G has e edges and v vertices, then 3v-e≥6.
4. A complete graph K_n is a planar if and only if n<5.
5. A complete bipartite graph K_{mn} is planar if and only if m<3 or n>3

•Isomorphism of Graphs:

A graph can exist in different forms having the same number of vertices, edges, and also the same edge connectivity. Such graphs are called isomorphic graphs. Note that we label the graphs in this chapter mainly for the purpose of referring to them and recognizing them from one another.

Isomorphic Graphs

Two graphs G_1 and G_2 are said to be isomorphic if −

6. Their number of components (vertices and edges) are same.

7. Their edge connectivity is retained.

Note − In short, out of the two isomorphic graphs, one is a tweaked version of the other. An unlabelled graph also can be thought of as an isomorphic graph.

There exists a function 'f' from vertices of G_1 to vertices of G_2
[f: $V(G_1) \Rightarrow V(G_2)$], such that
Case (i): f is a bijection (both one-one and onto)
Case (ii): f preserves adjacency of vertices, i.e., if the edge $\{U, V\} \in G_1$, then the edge $\{f(U), f(V)\} \in G_2$, then $G_1 \equiv G_2$.

Note

If $G_1 \equiv G_2$ then −

$|V(G_1)| = |V(G_2)|$

$|E(G_1)| = |E(G_2)|$

Degree sequences of G_1 and G_2 are same.

If the vertices $\{V_1, V_2, .. Vk\}$ form a cycle of length K in G_1, then the vertices $\{f(V_1)$,

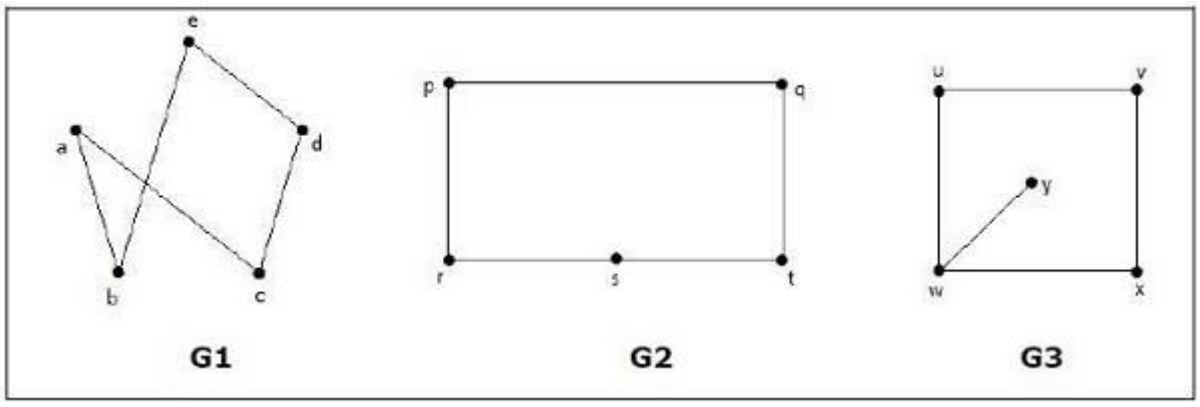
$f(V_2),\ldots f(Vk)\}$ should form a cycle of length K in G_2.

All the above conditions are necessary for the graphs G_1 and G_2 to be isomorphic, but not sufficient to prove that the graphs are isomorphic.

8. $(G_1 \equiv G_2)$ if and only if $(G_1- \equiv G_2-)$ where G_1 and G_2 are simple graphs.
9. $(G_1 \equiv G_2)$ if the adjacency matrices of G_1 and G_2 are same.
10. $(G_1 \equiv G_2)$ if and only if the corresponding subgraphs of G_1 and G_2 (obtained by deleting some vertices in G1 and their images in graph G_2) are isomorphic.

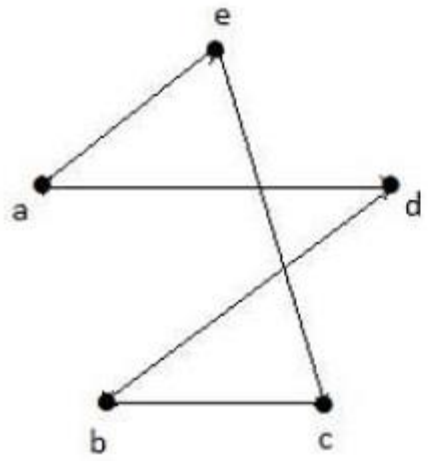
Example

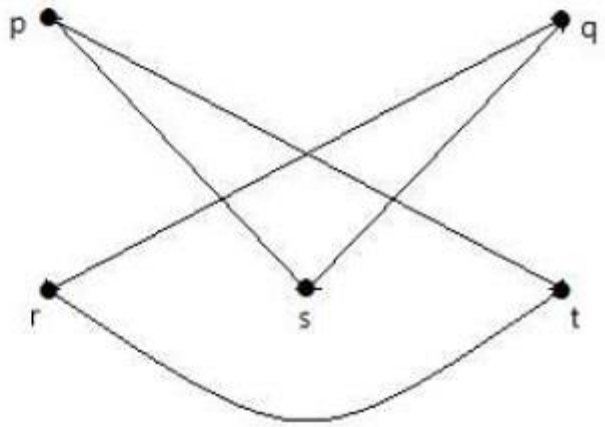
Which of the following graphs are isomorphic?



In the graph G_3, vertex 'w' has only degree 3, whereas all the other graph vertices has degree 2. Hence G3 not isomorphic to G_1 or G_2.

Taking complements of G_1 and G_2, you have −





Here, $(G_1 - \equiv G_2 -)$, hence $(G_1 \equiv G_2)$.

• Tree and its properties:

A graph which has no cycle is called an acyclic graph. A tree is an acyclic graph or graph having no cycles.

A tree or general trees is defined as a non-empty finite set of elements called vertices or nodes having the property that each node can have minimum degree 1 and maximum degree n. It can be partitioned into n+1 disjoint subsets such that the first subset contains the root of the tree and remaining n subsets includes the elements of the n subtree.

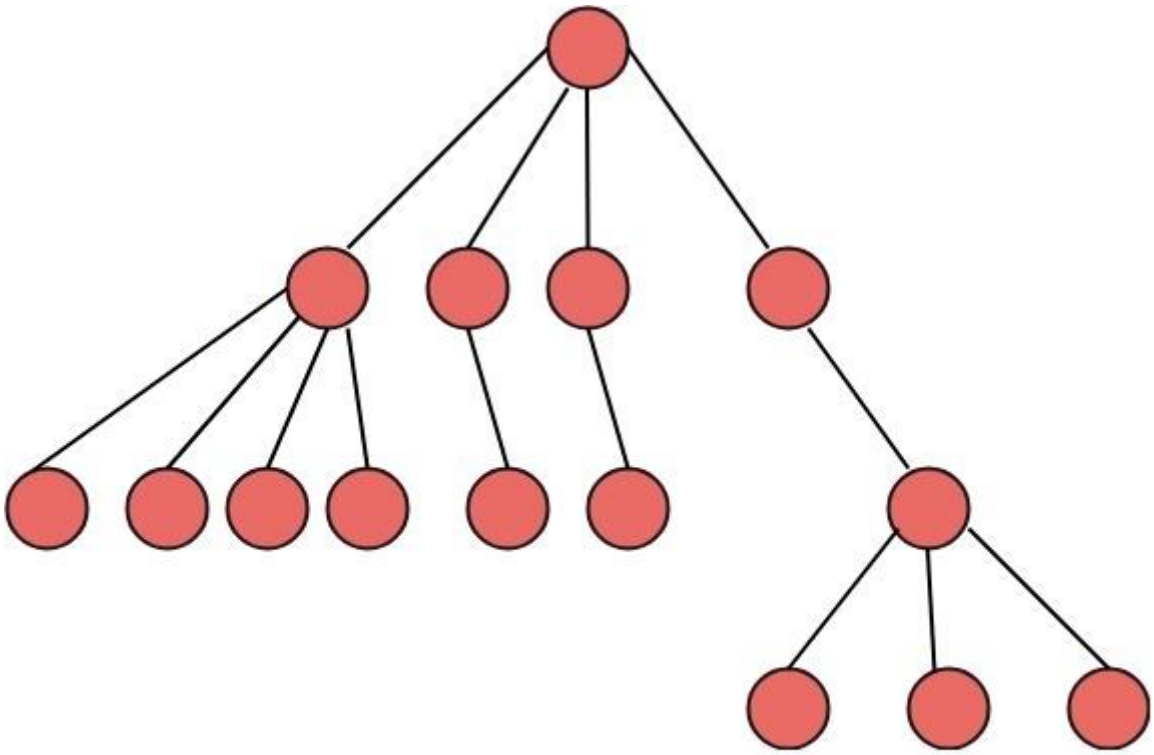


Fig:General Trees

Properties of Trees:

11. There is only one path between each pair of vertices of a tree.
12. If a graph G there is one and only one path between each pair of vertices G is a tree.
13. A tree T with n vertices has n-1 edges.
14. A graph is a tree if and only if it a minimal connected.

5.Write about Applications of Trees?

1.Store hierarchical data, like folder structure, organization structure, XML/HTML data
.Binary tree is a tree that allows fast search, insert, delete on a sorted data. It also allows finding closest item

15. Heap is a tree data structure which is implemented using arrays and used to implement priority queues.
16. B-Tree and B+ Tree : They are used to implement indexing in databases.
17. Syntax Tree: Scanning, parsing , generation of code and evaluation of arithmetic expressions in Compiler design.
18. K-D Tree: A space partitioning tree used to organize points in K dimensional space.

19. Trie : Used to implement dictionaries with prefix lookup.
20. Suffix Tree : For quick pattern searching in a fixed text.
21. Spanning Trees and shortest path trees are used in routers and bridges respectively in computer networks
22. As a workflow for compositing digital images for visual effects.
23. Decision trees.
24. Organization chart of a large organization.
25. In XML parser.
26. Machine learning algorithm.
27. For indexing in database.
28. IN server like DNS (Domain Name Server)
29. In Computer Graphics.
30. To evaluate an expression.
31. In chess game to store defense moves of player.
32. In java virtual machineShort answers

•Short Answers

(1),Explain the isomorphism graphs?

(2),write a short note on planar and Matrix graph?

(3), Evaluate the connectivity and separability?

(4), Describe the minimum spanning tree?

•Long Answes:

(1), Explain of graphs and incidences graphs?

(2),Show the Eluer graph and Hamilton graph coloring?

(3), Describe Trees and fundamental circuits Algorithms?

(4), Explain the minimum spanning tree and

(5), Evaluate the circuits of graph coloring?

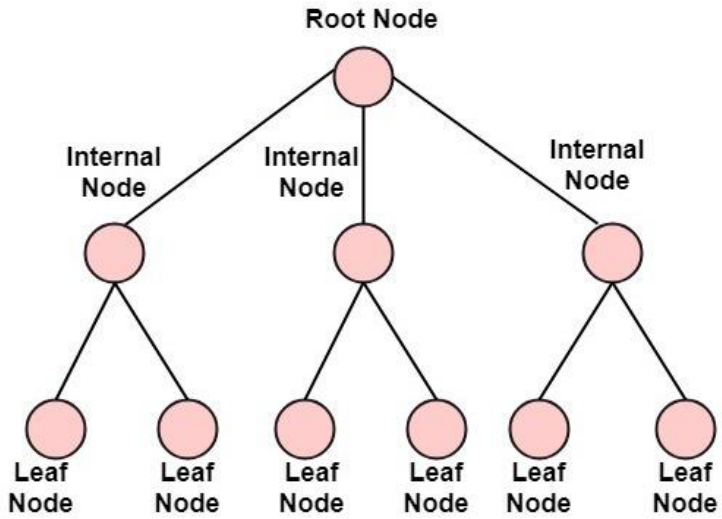
•Lecture Notes

Unit-IV

• Rootated Tree:

If a directed tree has exactly one node or vertex called root whose incoming degrees is 0 and all other vertices have incoming degree one, then the tree is called rooted tree.

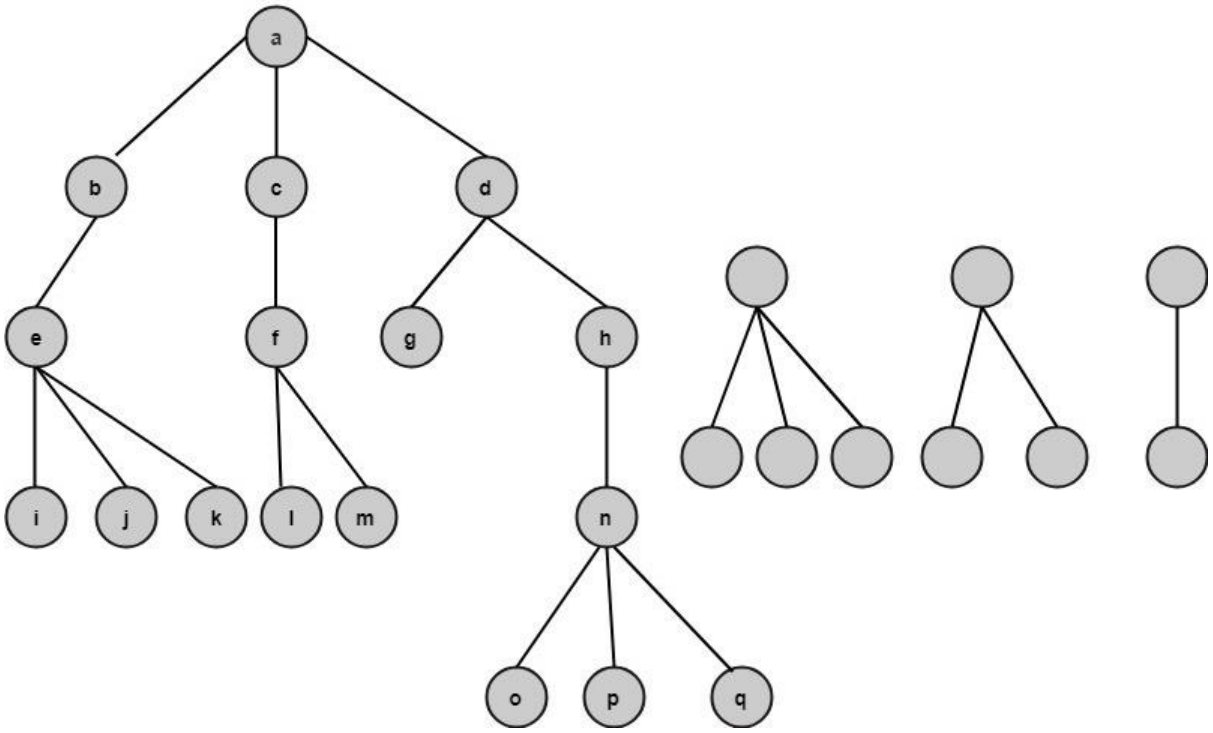
Note: 1. A tree with no nodes is a rooted tree (the empty tree)
2. A single node with no children is a rooted tree.



Path length of a Vertex:

The path length of a vertex in a rooted tree is defined to be the number of edges in the path from the root to the vertex.

Example: Find the path lengths of the nodes b, f, l, q as shown in fig:



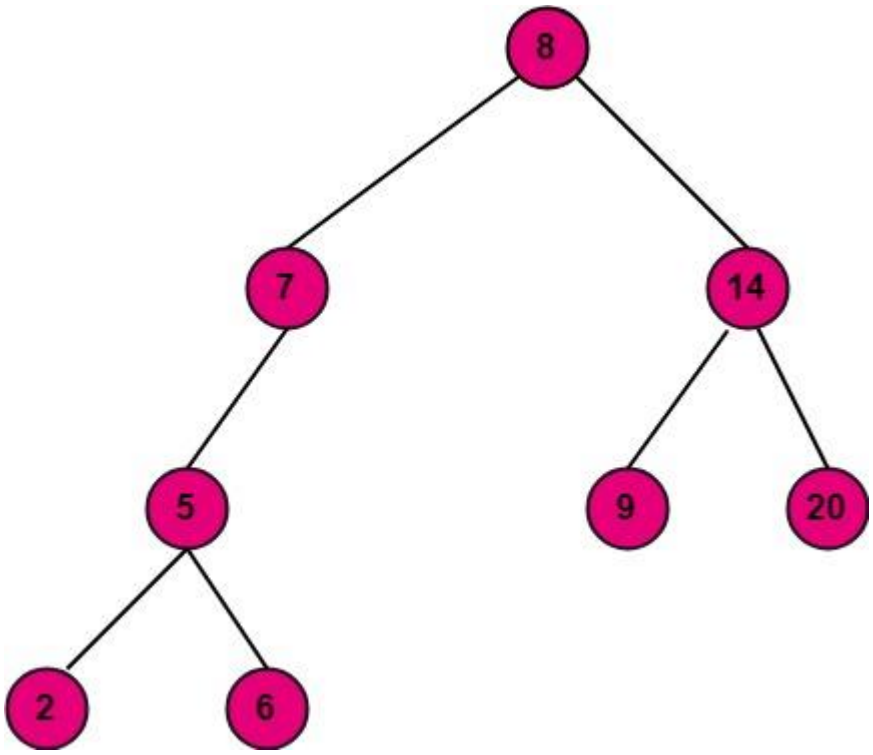
Solution: The path length of node b is one.
The path length of node f is two.
The path length of node l is three
The path length of the node q is

Binary search tree:

Binary search trees have the property that the node to the left contains a smaller value than the node pointing to it and the node to the right contains a larger value than the node pointing to it.

It is not necessary that a node in a 'Binary Search Tree' point to the nodes whose value immediately precede and follow it.

Example: The tree shown in fig is a binary search tree.



•Tree Traversal Techniques:

In this article, we will discuss the tree traversal in the data structure. The term 'tree traversal' means traversing or visiting each node of a tree. There is a single way to traverse the linear data structure such as linked list, queue, and stack. Whereas, there are multiple ways to traverse a tree that are listed as follows -

- Preorder traversal
- Inorder traversal
- Postorder traversal

So, in this article, we will discuss the above-listed techniques of traversing a tree. Now, let's start discussing the ways of tree traversal.

Preorder traversal

This technique follows the 'root left right' policy. It means that, first root node is visited after that the left subtree is traversed recursively, and finally, right subtree is recursively traversed. As the root node is traversed before (or pre) the left and right subtree, it is called preorder traversal.

So, in a preorder traversal, each node is visited before both of its subtrees.

The applications of preorder traversal include -

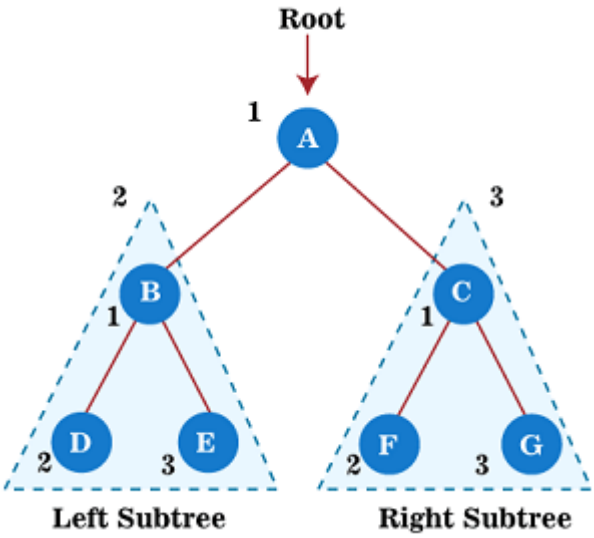
- It is used to create a copy of the tree.
- It can also be used to get the prefix expression of an expression tree.

Algorithm

1. Until all nodes of the tree are not visited
2.
3. Step 1 - Visit the root node
4. Step 2 - Traverse the left subtree recursively.
5. Step 3 - Traverse the right subtree recursively.

Example

Now, let's see the example of the preorder traversal technique.



Now, start applying the preorder traversal on the above tree. First, we traverse the root node **A;** after that, move to its left subtree **B**, which will also be traversed in preorder.

So, for left subtree B, first, the root node **B** is traversed itself; after that, its left subtree **D** is traversed. Since node **D** does not have any children, move to right subtree **E**. As node E also does not have any children, the traversal of the left subtree of root node A is completed.

Now, move towards the right subtree of root node A that is C. So, for right subtree C, first the root node **C** has traversed itself; after that, its left subtree **F** is traversed. Since node **F** does not have any children, move to the right subtree **G**. As node G also does not have any children, traversal of the right subtree of root node A is completed.

Therefore, all the nodes of the tree are traversed. So, the output of the preorder traversal of the above tree is -

A → B → D → E → C → F → G

To know more about the preorder traversal in the data structure, you can follow the link Preorder traversal

.

Postorder traversal

This technique follows the 'left-right root' policy. It means that the first left subtree of the root node is traversed, after that recursively traverses the right subtree, and finally, the root node is traversed. As the root node is traversed after (or post) the left and right subtree, it is called postorder traversal.

So, in a postorder traversal, each node is visited after both of its subtrees.

The applications of postorder traversal include -

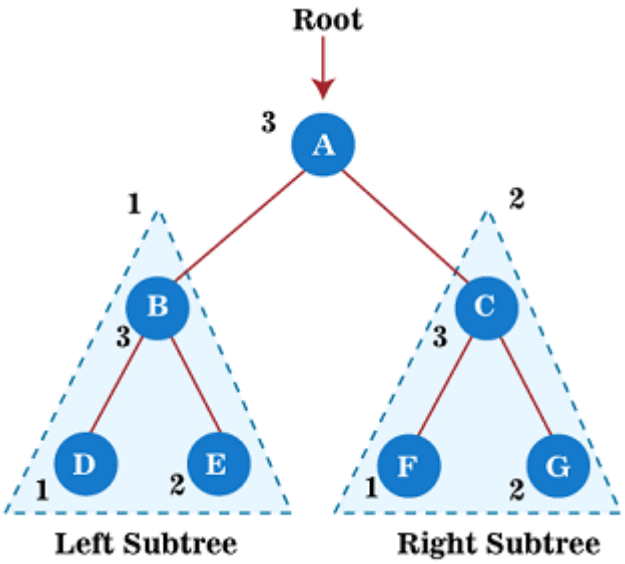
- It is used to delete the tree.
- It can also be used to get the postfix expression of an expression tree.

Algorithm

1. Until all nodes of the tree are not visited
2.
3. Step 1 - Traverse the left subtree recursively.
4. Step 2 - Traverse the right subtree recursively.
5. Step 3 - Visit the root node.

Example

Now, let's see the example of the postorder traversal technique.



Now, start applying the postorder traversal on the above tree. First, we traverse the left subtree B that will be traversed in postorder. After that, we will traverse the right subtree **C** in postorder. And finally, the root node of the above tree, i.e., **A**, is traversed.

So, for left subtree B, first, its left subtree **D** is traversed. Since node **D** does not have any children, traverse the right subtree **E**. As node E also does not have any children, move to the root node **B.** After traversing node **B,** the traversal of the left subtree of root node A is completed.

Now, move towards the right subtree of root node A that is C. So, for right subtree C, first its left subtree **F** is traversed. Since node **F** does not have any children, traverse the right subtree **G**. As node G also does not have any children, therefore, finally, the root node of the right subtree, i.e., **C,** is traversed. The traversal of the right subtree of root node A is completed.

At last, traverse the root node of a given tree, i.e., **A**. After traversing the root node, the postorder traversal of the given tree is completed.

Therefore, all the nodes of the tree are traversed. So, the output of the postorder traversal of the above tree is -

D → E → B → F → G → C → A

To know more about the postorder traversal in the data structure, you can follow the link Postorder traversal

.

Inorder traversal

This technique follows the 'left root right' policy. It means that first left subtree is visited after that root node is traversed, and finally, the right subtree is traversed. As the root node is traversed between the left and right subtree, it is named inorder traversal.

So, in the inorder traversal, each node is visited in between of its subtrees.

The applications of Inorder traversal includes -

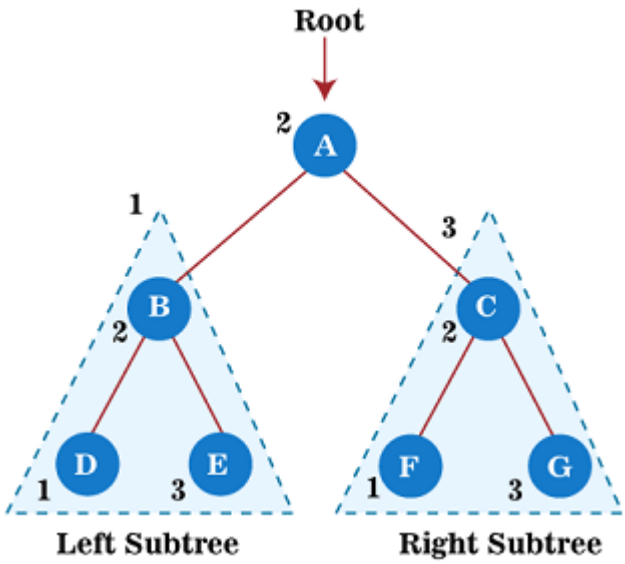
- It is used to get the BST nodes in increasing order.
- It can also be used to get the prefix expression of an expression tree.

Algorithm

1. Until all nodes of the tree are not visited
2.
3. Step 1 - Traverse the left subtree recursively.
4. Step 2 - Visit the root node.
5. Step 3 - Traverse the right subtree recursively.

Example

Now, let's see the example of the Inorder traversal technique.



Now, start applying the inorder traversal on the above tree. First, we traverse the left subtree **B** that will be traversed in inorder. After that, we will traverse the root node **A**. And finally, the right subtree **C** is traversed in inorder.

So, for left subtree **B**, first, its left subtree **D** is traversed. Since node **D** does not have any children, so after traversing it, node **B** will be traversed, and at last, right subtree of node B, that is **E**, is traversed. Node E also does not have any children; therefore, the traversal of the left subtree of root node A is completed.

After that, traverse the root node of a given tree, i.e., **A**.

At last, move towards the right subtree of root node A that is C. So, for right subtree C; first, its left subtree **F** is traversed. Since node **F** does not have any children, node **C** will be traversed, and at last, a right subtree of node C, that is, **G**, is traversed. Node G also does not have any children; therefore, the traversal of the right subtree of root node A is completed.

As all the nodes of the tree are traversed, the inorder traversal of the given tree is completed. The output of the inorder traversal of the above tree is -

D → B → E → A → F → C → G

4.Define Shortest path Algorithms?

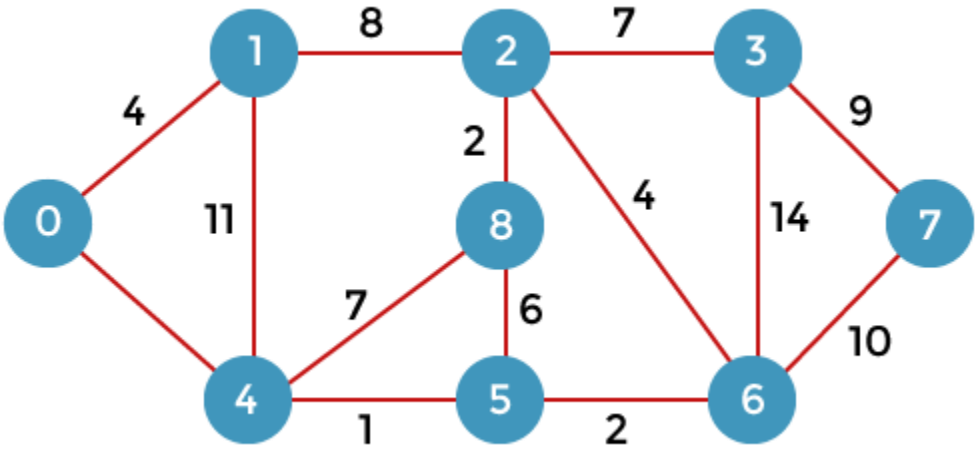
Dijkstra algorithm is a single-source shortest path algorithm. Here, single-source means that only one source is given, and we have to find the shortest path from the source to all the nodes.

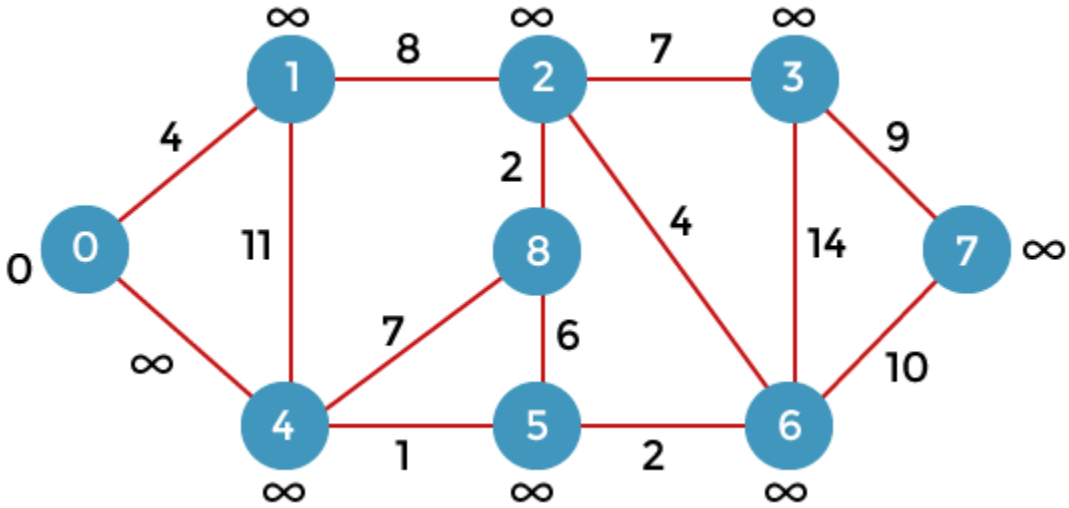
Let's understand the working of Dijkstra's algorithm. Consider the below graph.

First, we have to consider any vertex as a source vertex. Suppose we consider vertex 0 as a source

vertex.

Here we assume that 0 as a source vertex, and distance to all the other vertices is infinity. Initially, we do not know the distances. First, we will find out the vertices which are directly connected to the vertex 0. As we can observe in the above graph that two vertices are directly connected to vertex 0.





Let's assume that the vertex 0 is represented by 'x' and the vertex 1 is represented by 'y'. The distance between the vertices can be calculated by using the below formula:

d(x, y) = d(x) + c(x, y) < d(y)

= **(0 + 4)** < ∞

= **4** < ∞

Since 4<∞ so we will update d(v) from ∞ to 4.

Therefore, we come to the conclusion that the formula for calculating the distance between the vertices:

{if(d(u) + c(u, v) < d(v))

d(v) = d(u) +c(u, v) }

Now we consider vertex 0 same as 'x' and vertex 4 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= **(0 + 8)** < ∞

= **8** < ∞

Therefore, the value of d(y) is 8. We replace the infinity value of vertices 1 and 4 with the values 4 and 8 respectively. Now, we have found the shortest path from the vertex 0 to 1 and 0 to 4. Therefore, vertex 0 is selected. Now, we will compare all the vertices except the vertex 0. Since vertex 1 has the lowest value, i.e., 4; therefore, vertex 1 is selected.

Since vertex 1 is selected, so we consider the path from 1 to 2, and 1 to 4. We will not consider the path from 1 to 0 as the vertex 0 is already selected.

First, we calculate the distance between the vertex 1 and 2. Consider the vertex 1 as 'x', and the vertex 2 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= **(4 + 8)** < ∞

= **12** < ∞

Since 12<∞ so we will update d(2) from ∞ to 12.

Now, we calculate the distance between the vertex 1 and vertex 4. Consider the vertex 1 as 'x' and the vertex 4 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= (4 + 11) < 8

= 15 < 8

Since 15 is not less than 8, we will not update the value d(4) from 8 to 12.

Till now, two nodes have been selected, i.e., 0 and 1. Now we have to compare the nodes except the node 0 and 1. The node 4 has the minimum distance, i.e., 8. Therefore, vertex 4 is selected

Since vertex 4 is selected, so we will consider all the direct paths from the vertex 4. The direct paths from vertex 4 are 4 to 0, 4 to 1, 4 to 8, and 4 to 5. Since the vertices 0 and 1 have already been selected so we will not consider the vertices 0 and 1. We will consider only two vertices, i.e., 8 and 5.

First, we consider the vertex 8. First, we calculate the distance between the vertex 4 and 8. Consider the vertex 4 as 'x', and the vertex 8 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= (8 + 7) < ∞

= **15** < ∞

Since 15 is less than the infinity so we update d(8) from infinity to 15.

Now, we consider the vertex 5. First, we calculate the distance between the vertex 4 and 5. Consider the vertex 4 as 'x', and the vertex 5 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= **(8 + 1)** < ∞

= **9** < ∞

Since 5 is less than the infinity, we update d(5) from infinity to 9.

Till now, three nodes have been selected, i.e., 0, 1, and 4. Now we have to compare the nodes except the nodes 0, 1 and 4. The node 5 has the minimum value, i.e., 9. Therefore, vertex 5 is selected.

Since the vertex 5 is selected, so we will consider all the direct paths from vertex 5. The direct paths from vertex 5 are 5 to 8, and 5 to 6.

First, we consider the vertex 8. First, we calculate the distance between the vertex 5 and 8. Consider the vertex 5 as 'x', and the vertex 8 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= (9 + 15) < 15

= 24 < 15

Since 24 is not less than 15 so we will not update the value d(8) from 15 to 24.

Now, we consider the vertex 6. First, we calculate the distance between the vertex 5 and 6. Consider the vertex 5 as 'x', and the vertex 6 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= **(9 + 2)** < ∞

= **11** < ∞

Since 11 is less than infinity, we update d(6) from infinity to 11.

Till now, nodes 0, 1, 4 and 5 have been selected. We will compare the nodes except the selected nodes. The node 6 has the lowest value as compared to other nodes. Therefore, vertex 6 is selected.

Since vertex 6 is selected, we consider all the direct paths from vertex 6. The direct paths from vertex 6 are 6 to 2, 6 to 3, and 6 to 7.

First, we consider the vertex 2. Consider the vertex 6 as 'x', and the vertex 2 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= (11 + 4) < 12

= 15 < 12

Since 15 is not less than 12, we will not update d(2) from 12 to 15

Now we consider the vertex 3. Consider the vertex 6 as 'x', and the vertex 3 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= (11 + 14) < ∞

= 25 < ∞

Since 25 is less than ∞, so we will update d(3) from ∞ to 25.

Now we consider the vertex 7. Consider the vertex 6 as 'x', and the vertex 7 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= (11 + 10) < ∞

= 22 < ∞

Since 22 is less than ∞ so, we will update d(7) from ∞ to 22.

Till now, nodes 0, 1, 4, 5, and 6 have been selected. Now we have to compare all the unvisited nodes, i.e., 2, 3, 7, and 8. Since node 2 has the minimum value, i.e., 12 among all the other unvisited nodes. Therefore, node 2 is selected.

Since node 2 is selected, so we consider all the direct paths from node 2. The direct paths from node 2 are 2 to 8, 2 to 6, and 2 to 3.

First, we consider the vertex 8. Consider the vertex 2 as 'x' and 8 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= (12 + 2) < 15

= 14 < 15

Since 14 is less than 15, we will update d(8) from 15 to 14.

Now, we consider the vertex 6. Consider the vertex 2 as 'x' and 6 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= (12 + 4) < 11

= 16 < 11

Since 16 is not less than 11 so we will not update d(6) from 11 to 16.

Now, we consider the vertex 3. Consider the vertex 2 as 'x' and 3 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= (12 + 7) < 25

= 19 < 25

Since 19 is less than 25, we will update d(3) from 25 to 19.

Till now, nodes 0, 1, 2, 4, 5, and 6 have been selected. We compare all the unvisited nodes, i.e., 3, 7, and 8. Among nodes 3, 7, and 8, node 8 has the minimum value. The nodes which are directly connected to node 8 are 2, 4, and 5. Since all the directly connected nodes are selected so we will not consider any node for the updation.

The unvisited nodes are 3 and 7. Among the nodes 3 and 7, node 3 has the minimum value, i.e., 19. Therefore, the node 3 is selected. The nodes which are directly connected to the node 3 or 2.

Now, we consider the vertex 7. Consider the vertex 3 as 'x' and 7 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= (19 + 9) < 21

= 28 < 21

Since 28 is not less than 21, so we will not update d(7) from 28 to 21.

• flows in Network:

In graph theory, a **flow network** (also known as a **transportation network**) is a directed graph where each edge has a **capacity** and each edge receives a flow. The amount of flow on an edge cannot exceed the capacity of the edge. Often in operations research, a directed graph is called a **network**, the vertices are called **nodes** and the edges are called **arcs**. A flow must satisfy the restriction that the amount of flow into a node equals the amount of flow out of it, unless it is a **source**, which has only outgoing flow, or **sink**, which has only incoming flow. A network can be used to model traffic in a computer network, circulation with demands, fluids in pipes, currents in an electrical circuit, or anything similar in which something travels through a network of nodes. **Definition[edit]**

A **network** is a graph $G = (V, E)$, where V is a set of vertices and E is a set of V's edges – a subset of $V \times V$ –

together with a non-negative function $c: V \times V \longrightarrow$ $_{\infty}$, called the **capacity** function. Without loss of generality, we may assume that if $(u, v) \in E$ then (v, u) is also a member of E, since if $(v, u) \notin E$ then we may add (v, u) to E and then set $c(v, u) = 0$.

If two nodes in G are distinguished, a source s and a sink t, then (G, c, s, t) is called a **flow network**.[1]

Flows[edit]

There are various notions of a flow function that can be defined in a flow graph. Flow functions model the net flow of units between pairs of nodes, and are useful when asking questions such as *what is the maximum number of units that can be transferred from the source node s to the sink node t?* The simplest example of a flow function is known as a pseudo-flow.

> A **pseudo-flow** is a function $f : V \times V \rightarrow$ that satisfies the following two constraints for all nodes u and v:
>
> - *Capacity constraint*: An arc's flow cannot exceed its capacity, that is: $f(u, v) \leq c(u, v)$.

Given a pseudo-flow f in a flow network, it is often useful to consider the net flow entering a given node u,

that is, the sum of the flows entering u. The **excess** function $x_f : V \longrightarrow$ is defined by $x_f(u) = \Sigma_{v \in V} f(v, u)$. A node u is said to be **active** if $x_f(u) > 0$, **deficient** if $x_f(u) < 0$ or **conserving** if $x_f(u) = 0$.

These final definitions lead to two strengthenings of the definition of a pseudo-flow:

A **pre-flow** is a pseudo-flow that, for all $v \in V \setminus \{s\}$, satisfies the additional constraint:

- *Non-deficient flows*: The net flow *entering* the node v is non-negative, except for the source, which "produces" flow. That is: $x_f(v) \geq 0$ for all $v \in V \setminus \{s\}$.

A **feasible flow**, or just a **flow**, is a pseudo-flow that, for all $v \in V \setminus \{s, t\}$, satisfies the additional constraint:

- *Flow conservation*: The net flow *entering* the node v is 0, except for the source, which "produces" flow, and the sink, which "consumes" flow. That is: $x_f(v) = 0$ for all $v \in V \setminus \{s, t\}$.

The **value** of a feasible flow f, denoted $|f|$, is the net flow into the sink t of the flow network. That is, $|f| = x_f(t)$.

•short Answers

(1),What is rootated tree?

(2),Explain properties of tree.

(3),What is meant by binary search tree?

(4),Define shortest path?

•Long Answers

(1),Explain Tree Traversal techniques?

(2),Define shortest path algorithm?

(3),What is flow in Network?

(4),Describe the minimum cost problems?

(5),Evaluate Ford Fulkersons Algorithm for maximum flow?

•Lecture Notes

UNIT -V

Propositional logic:

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 (∨∨)
- AND (∧∧)
- Negation/ NOT (¬¬)
- Implication / if-then (→→)
- If and only if (⇔⇔).

OR (∨∨) − The OR operation of two propositions A and B (written as A∨BA∨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 ∨ 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∧BA∧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¬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→BA→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⇔BA⇔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

.Define Tautology

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

Example − Prove [(A→B)∧A]→B[(A→B)∧A]→B is a tautology

The truth table is as follows −

A	B	A → B	(A → B) ∧ A	[(A → B) ∧ 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 $[(A \rightarrow B) \wedge A] \rightarrow B$ is "True", it is a tautolog

terms contradiction and contingency:

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

Example − Prove $(A \vee B) \wedge [(\neg A) \wedge (\neg B)]$ is a contradiction

The truth table is as follows −

A	B	A ∨ B	¬ A	¬ B	(¬ A) ∧ (¬ B)	(A ∨ B) ∧ [(¬ A) ∧ (¬ 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 $(A \vee B) \wedge [(\neg A) \wedge (\neg B)]$ 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 $(A \vee B) \wedge (\neg A)$ a contingency

The truth table is as follows −

A	B	A ∨ B	¬ A	(A ∨ B) ∧ (¬ 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 $(A \vee B) \wedge (\neg A)$ $(A \vee B) \wedge (\neg A)$ has both "True" and "False", it is a cC

Normal forms

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

1. $(A \vee B) \wedge (A \vee C) \wedge (B \vee C \vee D)$
2. $(p \cup Q) \cap (Q \cup R)$

Disjunctive Normal Form

A compound statement is in disjunctive 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

- $(A \wedge B) \vee (A \wedge C) \vee (B \wedge C \wedge D)$
- $(P \cap Q) \cup (Q \cap R)$

.Explain Quantifiers;

Universal Quantifiers

Sometimes the mathematical statements assert that if the given property is true for all values of a variable in a given domain, it will be known as the **domain of discourse**. Using the universal quantifiers, we can easily express these statements. The universal quantifier symbol is denoted by the ∀, which means "**for all**". Suppose P(x) is used to indicate predicate, and D is used to indicate the domain of x. The universal statement will be in the form **"∀x ∈ D, P(x)".** The main purpose of a universal statement is to form a proposition. In

the quantifiers, the domain is very important because it is used to decide the possible values of x. When we change the domain, then the meaning of universal quantifiers of P(x) will also be changed. When we use the universal quantifier, in this case, the domain must be specified. Without a domain, the universal quantifier has no meaning.

The sentence **∀xP(x)** will be **true** if and only if P(x) is true for every x in D or P(x) is true for every value which is substituted for x. The statement **∀xP(x)** will be **false** if and only if P(x) is false for at least one x in D. The value for x for which the predicate P(x) is false is known as the **counterexample** to the universal statement. If finite values such as $\{n_1, n_2, n_3, ..., n_k\}$ are contained by the universe of discovery, the universal quantifier will be the **conjunction** of all elements, which is described as follows:

$\forall x P(x) \Leftrightarrow P(n_1) \wedge P(n_2) \wedge \cdots \wedge P(n_k)$

Example 1: Suppose P(x) indicates a predicate where "x must take an electronics course" and Q(x) also indicates a predicate where "x is an electrical student". Now we will find the universal quantifier of both predicates.

Solution: Suppose the students are from ABC College. For both predicates, the universe of discourse will be all ABC students.

The statements can be: "Every electrical student must take an electronics course". The following syntax is used to define this statement:

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

This statement can be expressed in another way: "Everybody must take an electronics course or be an electrical student". The following syntax is used to define this statement:

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

Example 2: Suppose P(x) indicates a predicate where "x is a square" and Q(x) also indicates a predicate where "x is a rectangle". Now we will find the universal quantifier of these predicates.

Solution:

The statement must be:

∀x (x is a square ⇒ x is a rectangle), i.e., "all squares are rectangles.'' The following syntax is used to describe this statement:

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

Sometimes, we can use this construction to express a mathematical sentence of the form "if this, then that," with an "understood" quantifier.

Universal conditional statement:

This statement has the form: ∀x, if P(x) then Q(x).

For example: In this example, we will rewrite the below statement in the form:

∀_____, if_____then_____

If Jack is 18 years old or older, then he is eligible to vote.

Existential Quantifiers

Sometimes the mathematical statements assert that we have an element that contains some properties. Using existential quantifiers, we can easily express these statements. The existential quantifier symbol is denoted by the ∃, which means **"there exists"**. Suppose P(x) is used to indicate predicate, and D is used to indicate the domain of x. The existential statement will be in the form "**∃x ∈ D** such that P(x)". The main purpose of an existential statement is to form a proposition. The sentence **∃xP(x)** will be **true** if and only if P(x) is true for at least one x in D. The statement **∃xP(x)** will be **false** if and only if P(x) is false for all x in D. The value for x for which the predicate P(x) is false is known as the **counterexample** to the existential statement.

If finite values such as $\{n_1, n_2, n_3, \ldots, n_k\}$ are contained by the universe of discovery, the universal quantifier will be the **disjunction** of all elements, which is described as follows:

$\exists x P(x) \Leftrightarrow P(n_1) \vee P(n_2) \vee P(n_3) \cdots \vee P(n_k)$

Example 1: Suppose P(x) contains a statement "x > 4". Now we will find the truth value of this statement.

Solution:

This statement is false for all real number which is less than 4 and true for all real numbers which are greater than 4.

This statement is false for x= 6 and true for x = 4. Now we will compare the above statement with the following statement. So

∃xP(x) is true

Negating Quantified Statements

Earlier we have explain a example in which the statement **∀x : x^2 > 2** is false and **∀x : x^2 +1 > 0** is true for x = 1. The first statement is false because x =1 is unable to satisfy the predicate. In this case, we find a solution that says we can negate a ∀ statement by flipping ∀ into ∃. After that, we will negate the predicate inside.

For example: The negation of ∀x : P(x) is ∃x : P(x).

If the statement predicate **∀x : P(x)** is true, then ∃x : P(x). Here, the x that satisfies P(x) is known as the counterexample that claims **∀x : P(x).** Similarly, if we want to negate ∃x : P(x), we have to claim that P(x) fails to hold for any value of x. So we again flip the quantifier and then negate the predicate like this:

the negation of ∃x : P(x) is ∀x : P(x)

Nested Quantifiers

The nested quantifier is used by a lot of serious mathematical statements. For example:

Let us assume a statement that says, "For every real number, we have a real number which is greater than it". We are going to write this statement like this:

∀x ∃y : y > x

Or assume a statement that says, "We have a Boolean formula such that every truth assignment to its variables satisfies it". We are going to write this statement like this:

∃ formulaF ∀ assignmentsA : A satisfies F.

It is very important to understand the difference between statements that indicate **∃x ∀y** and a statement that indicate **∀x ∃y**. For example, suppose we are talking about the real number. In this case, our above example **∀x ∃y : y > x** is true. But it will be false if we try to write this with quantifiers in other order like this: **∃y ∀x : y > x**. This version needs a single number that must be larger than every number.

Negating Nested Quantifiers

In the nested quantifier, we can negate a sequence with the help of flipping each quantifier in the sequence, and after that, we will negate the predicate like this:

Negation of ∀x ∃y : P(x, y) is ∃x ∀y : $\overline{P}$(x, y)

When we think, we can realize that it makes sense intuitively. **For example**: here, we will consider the unbounded sequence definition from calculus. If there are real numbers that have infinite sequence $a_1 \leq a_2 \leq a_3 \leq$ ...., then it will be unbounded if it eventually grows greater than x for every number x. Here the quantifiers lurking is already seen: ∀x ∃n : a_n > x.

Now there are some sequences that are unbounded such as 1, 4, 9, 16, 25, ..., and some sequences that are not, such as 1/2 , 3/4 , 7/8 , ... If a sequence is not bounded, it means that it contains an upper-bounded x such that sequence's every number is at most x.

If we want to derive this mathematically, we can do this by negating the definition of unboundedness. If **unbounded** has the statement **∀x ∃n : an > x**, then **not unbounded** will have the statement **∃x ∀n : an ≤ x**. That means by flipping the quantifiers, we can convert unbounded into not unbounded

•Short Answers

(1), Write the prospitional Logic?

(2), Explain about Basic Logical operation?

(3), Explain Alebra of prospitional?

(4), Describe the universal quantifiers?

•Long Answers

(1), Write the First order,basic order,truthtables?

(2), Describe the contradictions?

(3), Explain the quantifiers?

(4),show the Hamiltonian therom n=3?

(5), Explain the normal forms and tatuology?

Question paper for sample:-

S.V.U College of commerce Management And Computer science::

Tirupati:2hours

First Internal

Paper-MCA 101:Discrete Mathematical Structures

Part-A

Answer any Five of the following 5*2=10

1.Define Function?

2.write about Recursively defined sequences?

3.Define Group?

4.Define subgroup?

5.Define Structural induction?

6.what is strong induction?
7.what is Recursion?
8.what is a well ordered set?

Part-B

Unit-I

Answer any one full question from each unit 10*2=20

9.what is mathematical induction?Use mathematical induction to prove that 1+2+2^2+……..+2^n-1 for all non negative integers n

or

10.Explain cyclic groups in detail?

Unit-II

11.write about pigeonhole principle in detail?

or

12.what is the expansion of(x+2)?

S.v.u college of commerce management and computer science::
Tirupati
Department of computer science
Time:2Hours

Second Internal

Paper-MCA 101:Discrete Mathematical Structures

Part-A

Answer any five of the following 5*2=10

1.Define Finite and Infinite graphs?
2.Explain incidence and degree of a graph?
3.Explain graph Terminology?
4.Define Multigraph?
5.what is rooted tree?
6.Explain properties of tree?

7.Explain truth tables?
8.what is logical Equivalence?

Part-B

Unit-I

Answer any one full question from each unit 10*2=20

9.Explain types of graphs?

Or

10.Explain Biperated graph and planar graph and their properties?

Unit-II

11.prove that ^2 is irrational by contradiction?

Or

12.Obtain DCNF for the formula (~p->r)^(q<->p)

4th unit

1.What is rootated tree?

2.Explain properties of tree.

3.What is meant by binary search tree?

4.Define shortest path.

1.Explain tree traversal techniques?

2.Define shortest path algorithm.

3.What is flow in Network?

4. 4th unit

1.What is rootated tree?

2.Explain properties of tree.

3.What is meant by binary search tree?

4.Define shortest path.

1.Explain tree traversal techniques?

2.Define shortest path algorithm.

3.What is flow in Network?

Master of computer Applications Degree Examination

First Semester

Paper-MCA 101:Discrete Mathematical Structures

Time:3 hours Max.Marks:70

Part-A

Answer any five of the following Questions. Each Question carries 2 marks 5*2=10

1.a)Show that ~(p^q) and ~p^~q are logically equivalent.

b)Let Q(x) be the statement ”x<2”. What is the truth value of the quantification for all xQ(x),where the domain consists of all real numbers.

c)Give Recursive definition of a^n,where a is a non zero real number and is a nonnegative integer.

d)Write about well-Formed formulae in propositional logic.

e) How many permutations of the letters ABCDEFGH contain the string ABC?

f) How many poker hands of five cards can be dealt from a standard deck of 52 cards? Also ,how many ways are there to select 47 cards from a standard deck of 52 cards?

g) Explain Hanoi Tower problem.

h)Define linear non homogeneous recurrence relation of degree k with constant coefficients.

I)write about Dirac’s Theorem with proof.

j)Give the difference between Euler path and circuit.

Part-B

Answer any one full question from each unit. Each question carries 12 Marks. (5*12=60)

Unit-I

2.I) Obtain principle conjunctive normal form (PCNF) for the formula (~p->r) ^(q<->p)

II) Show that the following is inconsistent p->q, r-> s , p v r,~(q vs).

Or

3.I) Using indirect proof. Derive p->~s from p->qvr, q->~p, s->~r, p.

II)Show that r->(s->q), p v r and s=>p->q

Unit-II

4.I) What is mathematical induction? Use mathematical induction to prove that 1.1! + 2.2! + …+ (n+1)! – 1, wherever n is a positive integer.

II)write about Recursively defined function, sets in detail.

Or

5.I) Using structural induction, show that whenever n>=3, fn >a^n-2, where a = (1+5)/2.

II) Give a recursive algorithm for computing a^ n, where a is a non zero real number and n is a nonnegative

Unit-III

6.I) write about pigeonhole principle in detail? prove that if N objects are placed into k boxes, then there is at least one box containing least [N/k] objects?

II)Define permutation and show in how many ways are there to select a first-prize winner , a second-prize winner, and a third-prize winner from 100 different people who have entered a contest?

Or

7.I) Let n and r be nonnegative integers with r<=n. Then prove that C(n, r)=C(n, n-r) and show that in how many ways are there to select five players from a 10 members tennis team to make a trip to a match at another school?

II)what is the expansion of (x+ y) ^ 4?

Unit-IV

8.I) Let n be a positive integer. Show that 2^n*2^n checkerboard with 1 square removed can be tiled using right triominoes were this piece covered with three squares at a time

II) Find the Fibonacci numbers f2,f3,f4,f5,f6 using recursive algorithm.

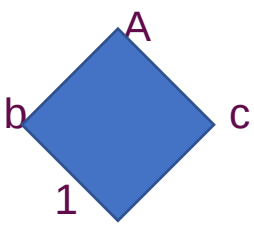
Or

9.I) Discuss about principle of inclusion-exclusion and show that how many positive integers are not exceeding 1000 are divisible by 7 or 11?

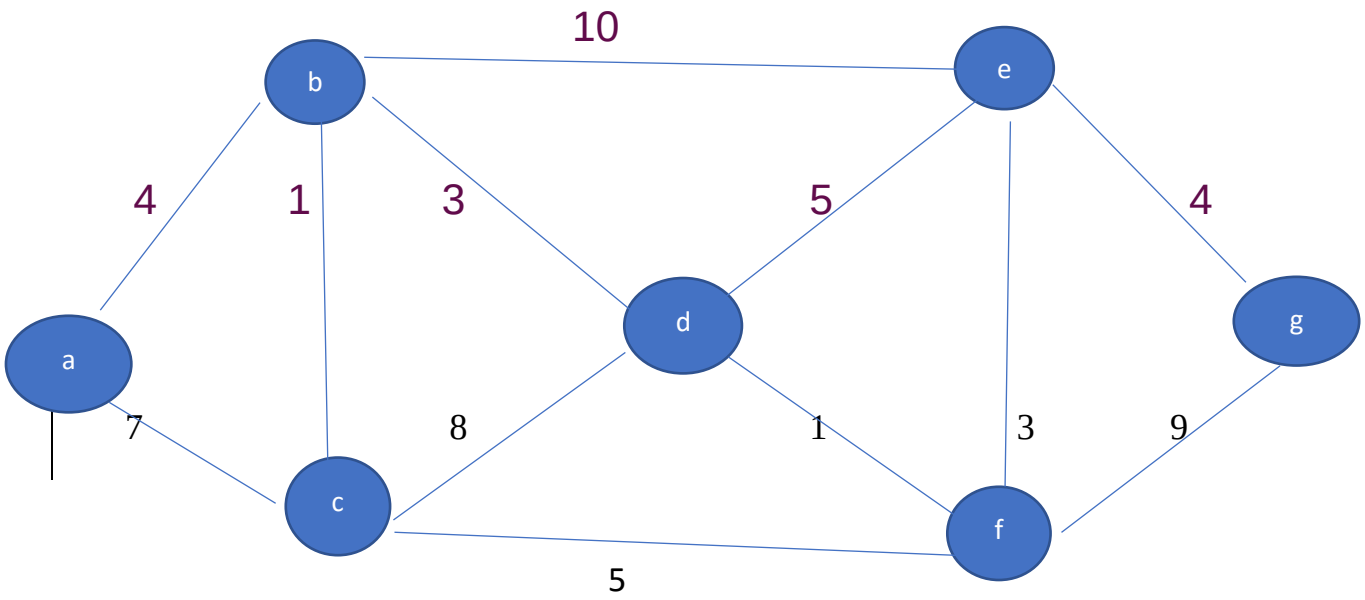
II) Solve the recurrence relation ak=3ak-1 for k=1,2,3,….. and initial condition a0=2.

Unit-V

10.I) Show that the graphs G=(V,E) and H=(W,F) shown in below figure are isomorphic



II) Use Dijikstra's algorithm to find the length of a shortest path between the vertices A and G in the weighted graph shown below



Or

11.I) Show that kn has a Hamiliton circuit whenever n>=3.

II) Write about Graph coloring and show how can the final exams at a university be scheduled so that no student has two exams at the same time