UNIT-3

SemiGroup

Let us consider, an algebraic system $(A, *)$, where * is a binary operation on A. Then, the system $(A, *)$ is said to be semi-group if it satisfies the following properties:

1. The operation * is a closed operation on set A.
2. The operation * is an associative operation

Example: Consider an algebraic system $(A, *)$, where $A = \{1, 3, 5, 7, 9....\}$, the set of positive odd integers and * is a binary operation means multiplication. Determine whether $(A, *)$ is a semi-group.

Solution: Closure Property: The operation * is a closed operation because multiplication of two +ve odd integers is a +ve odd number.

Associative Property: The operation * is an associative operation on set A. Since every $a, b, c \in A$, we have

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

Hence, the algebraic system $(A, *)$, is a semigroup.

Subsemigroup:

Consider a semigroup $(A, *)$ and let $B \subseteq A$. Then the system $(B, *)$ is called a subsemigroup if the set B is closed under the operation *.

Example: Consider a semigroup $(N, +)$, where N is the set of all natural numbers and + is an addition operation. The algebraic system

(E, +) is a subsemigroup of (N, +), where E is a set of +ve even integers.

Free Semigroup:

Consider a non empty set $A = \{a_1, a_2, \ldots..a_n\}$.

Now, A* is the set of all finite sequences of elements of A, i.e., A* consist of all words that can be formed from the alphabet of A.

If α,β,and,γ are any elements of A*, then α,(β. γ)=(α.β).γ.

Here ° is a concatenation operation, which is an associative operation as shown above.

Thus (A*,°) is a semigroup. This semigroup (A*,°) is called the free semigroup generated by set A.

Product of Semigroup:

Theorem: If $(S_1,*)$and $(S_2,*)$ are semigroups, then $(S_1 \times S_2*)$ is a semigroup, where * defined by $(s_1', s_2')*(s_1'', s_2'')=(s_1'*s_1'', s_2'*s_2'')$.

Proof: The semigroup $S_1 \times S_2$ is closed under the operation *.

Associativity of *.Let $a, b, c \in S_1 \times S_2$

So, $a * (b * c) = (a_1,a_2)*((b_1,b_2)*(c_1,c_2))$

$= (a_1,a_2)*(b_1 *_1 c_1,b_2 *_2 c_2)$

$= (a_1 *_1 (b_1 *_1 c_1),a_2 *_2 (b_2 *_2 c_2)$

$= ((a_1 *_1 b_1) *_1*_1,(a_2 *_2 b_2) *_2 c_2)$

$= (a_1 *_1 b_1,a_2 *_2 b_2)*(c_1,c_2)$

$= ((a_1,a_2)*(b_1,b_2))*(c_1,c_2)$

$= (a * b) * c.$

Since * is closed and associative. Hence, $S_1 \times S_2$ is a semigroup.

Monoid:

Let us consider an algebraic system (A, o), where o is a binary operation on A. Then the system (A, o) is said to be a monoid if it satisfies the following properties:

1. The operation o is a closed operation on set A.
2. The operation o is an associative operation.
3. There exists an identity element, i.e., the operation o.

Example: Consider an algebraic system (N, +), where the set N = {0, 1, 2, 3, 4...}.The set of natural numbers and + is an addition operation. Determine whether (N, +) is a monoid.

Solution:

(a) Closure Property: The operation + is closed since the sum of two natural numbers.

(b)Associative Property: The operation + is an associative property since we have $(a+b)+c=a+(b+c)\ \forall\ a, b, c \in N$.

(c)Identity: There exists an identity element in set N the operation +. The element 0 is an identity element, i.e., the operation +. Since the operation + is a closed, associative and there exists an identity. Hence, the algebraic system (N, +) is a monoid.

SubMonoid:

Let us consider a monoid (M, o), also let $S \subseteq M$. Then (S, o) is called a submonoid of (M, o), if and only if it satisfies the following properties:

1. S is closed under the operation o.
2. There exists an identity element $e \in T$.

Example: Let us consider, a monoid (M, *), where * s a binary operation and M is a set of all integers. Then (M_1, *) is a submonoid of (M, *) where M_1 is defined as $M_1=\{a_i \mid i$ is from 0 to n,a positive integer,and $a \in M\}$.

Group:

Let G be a non-void set with a binary operation * that assigns to each ordered pair (a, b) of elements of G an element of G denoted by a * b. We say that G is a group under the binary operation * if the following three properties are satisfied:

1) Associativity: The binary operation * is associative i.e. $a*(b*c)=(a*b)*c$, $\forall$ $a,b,c \in G$

2) Identity: There is an element e, called the identity, in G, such that $a*e=e*a=a$, $\forall$ $a \in G$

3) Inverse: For each element a in G, there is an element b in G, called an inverse of a such that $a*b=b*a=e$, $\forall$ $a, b \in G$

Note: If a group has the property that $a*b=b*a$ i.e., commutative law holds then the group is called an abelian.

Properties of Groups:

The following theorems can understand the elementary features of Groups:

Theorem1:-

1. Statement: - In a Group G, there is only one identity element (uniqueness of identity) Proof: - let e and e' are two identities in G and let $a \in G$

$\therefore$ $ae = a \rightarrow (i)$
$\therefore$ $ae' = a \rightarrow (ii)$

R.H.S of (i) and (ii) are equal $\Rightarrow ae = ae'$

Thus by the left cancellation law, we obtain $e= e'$

There is only one identity element in G for any $a \in G$. Hence the theorem is proved.

2. Statement: - For each element a in a group G, there is a unique element b in G such that ab= ba=e (uniqueness if inverses)

Proof: - let b and c are both inverses of a $a \in G$

Then ab = e and ac = e
$\because$ c = ce {existence of identity element}
$\Rightarrow$ c = c (ab) {$\because$ ab = e}
$\Rightarrow$ c = (c a) b
$\Rightarrow$ c = (ac) b { $\because$ ac = ca}
$\Rightarrow$ c = eb
$\Rightarrow$ c = b { $\because$ b = eb}

Hence inverse of a G is unique.

Theorem 2:-

1. Statement: - In a Group G,$(a^{-1})^{-1}=a, \forall\ a \in G$

Proof: We have $a\ a^{-1}=a^{-1}\ a=e$

Where e is the identity element of G

Thus a is inverse of $a^{-1} \in G$

i.e., $(a^{-1})^{-1}=a, \forall\ a \in G$

2. Statement: In a Group G,$(a\ b^{-1})=b^{-1}\ a^{-1}, \forall\ a,b \in G$

Proof: - By associatively we have

$(b^{-1}\ a^{-1})ab=b^{-1}\ (a^{-1}\ a)b$
$\Rightarrow (b^{-1}\ a^{-1})ab=b^{-1}\ (e)b$ {$\because a^{-1}\ a=e$}
$\Rightarrow (b^{-1}\ a^{-1})ab=b^{-1}\ b$ {$\because eb=b$}
$\Rightarrow (b^{-1}\ a^{-1})ab=e,$ {$\because b^{-1}\ b=e$}

Similarly

$(ab)\ (b^{-1}\ a^{-1})=a(b\ b^{-1})\ a^{-1}$
$\Rightarrow (ab)\ (b^{-1}\ a^{-1})=a\ (e)\ a^{-1}$
$\Rightarrow (ab)\ (b^{-1}\ a^{-1})=a\ a^{-1}$

$\Rightarrow$(ab) (b^{-1} a^{-1})=e {$\because aa^{-1}=e$}
Thus (b^{-1} a^{-1})ab=(ab)(b^{-1} a^{-1})=e
$\therefore$ b^{-1} a^{-1} is the inverse of ab
i.e., b^{-1} a^{-1}= a b^{-1}

Hence the theorem is proved.

Theorem3:-

In a group G, the left and right cancellation laws hold i.e.

(i) ab = ac implies b=c

(ii) ba=ca implies b=c

Proof

(i) Let ab=ac
Premultiplying a^{-1} on both sides we get
a^{-1} (ab)=a^{-1} (ac)
$\Rightarrow$ (a^{-1}a) b=(a^{-1} a)c
$\Rightarrow$eb=ec
$\Rightarrow$b=c Hence Proved.

(ii) Let ba=ca
Post-multiplying a^{-1} on both sides
$\Rightarrow$(ba) a^{-1}=(ca) a^{-1}
$\Rightarrow$b(aa^{-1})=c(aa^{-1})
$\Rightarrow$be=ce
$\Rightarrow$b=c

Hence the theorem is proved.

Finite and Infinite Group:

A group (G, *) is called a finite group if G is a finite set.

A group (G, *) is called a infinite group if G is an infinite set.

Example1: The group (I, +) is an infinite group as the set I of integers is an infinite set.

Example2: The group G = {1, 2, 3, 4, 5, 6, 7} under multiplication modulo 8 is a finite group as the set G is a finite set.

Order of Group:

The order of the group G is the number of elements in the group G. It is denoted by |G|. A group of order 1 has only the identity element, i.e., ({e} *).

A group of order 2 has two elements, i.e., one identity element and one some other element.

Example1: Let ({e, x}, *) be a group of order 2. The table of operation is shown in fig:

*	e	x
e	e	x
x	x	e

The group of order 3 has three elements i.e., one identity element and two other elements.

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).

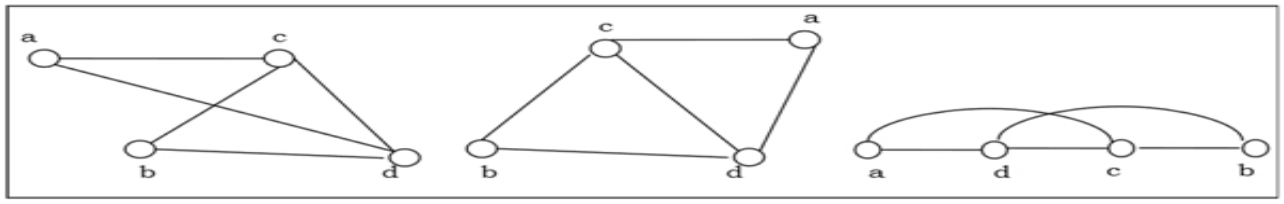
It is easier to check non-isomorphism than isomorphism. If any of these following conditions occurs, then two graphs are non-isomorphic –

- 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 –



Homomorphism

A homomorphism from a graph ***G*** to a graph ***H*** is a mapping (May not be a bijective mapping) h: $G \rightarrow H$ such that – $(x, y) \in E(G) \rightarrow (h(x), h(y)) \in E(H)$. 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 NPcomplete problem.

Subgroup:

If a non-void subset H of a group G is itself a group under the operation of G, we say H is a subgroup of G.

Theorem: - A subset H of a group G is a subgroup of G if:

- the identity element $a \in H$.
- H is closed under the operation of G i.e. if $a, b \in H$, then $a, b \in H$ and
- H is closed under inverses, that is if $a \in H$ then $a^{-1} \in H$.

Cyclic Subgroup:-

A Subgroup K of a group G is said to be cyclic subgroup if there exists an element $x \in G$ such that every element of K can be written in the form x^n for some $n \in Z$.

The element x is called generator of K and we write K= <x>

Cyclic Group:-

In the case when G=, we say G is cyclic and x is a generator of G. That is, a group G is said to be cyclic if there is an element $x \in G$ such that every element of G can be written in the form x^n for the some $n \in Z$.

Example: The group G= {1, -1, i,-i} under usual multiplication is a finite cyclic group with i as generator, since $i^1=i, i^2=-1, i^3=-i$ and $i^4=1$

Abelian Group:

Let us consider an algebraic system (G,*), where * is a binary operation on G. Then the system (G,*) is said to be an abelian group if it satisfies all the properties of the group plus a additional following property:

(1) The operation * is commutative i.e.,
$a * b = b * a \ \forall \ a,b \in G$

Example: Consider an algebraic system (G, *), where G is the set of all non-zero real numbers and * is a binary operation defined by

$$a * b = \frac{ab}{4}.$$

Show that (G, *) is an abelian group.

Solution:

Closure Property: The set G is closed under the operation *, since $a * b = \frac{ab}{2}$ is a real number. Hence, it belongs to G.

Associative Property: The operation * is associative. Let a,b,c∈G, then we have

$$(a*b)*c = \left(\frac{ab}{4}\right) * c = \frac{(ab)c}{16} = \frac{abc}{16}$$

Similarly, $$a * (b * c) = a * \left(\frac{bc}{4}\right) = \frac{a(bc)}{16} = \frac{abc}{16}$$

Identity: To find the identity element, let us assume that e is a +ve real number. Then e * a = a, where a ∈G.

$$\frac{Ea}{4} = a \quad \text{or} \quad e = 4$$

Similarly, $a * e = a$

$$\frac{Ae}{4} = a \quad \text{or} \quad e = 4.$$

Thus, the identity element in G is 4.

Inverse: let us assume that a ∈G. If $a^{-1} \in Q$, is an inverse of a, then $a * a^{-1} = 4$

Therefore, $$\frac{aa^{-1}}{4} = 4 \quad \text{or } a^{-1} = \frac{16}{a}$$

Similarly, $$a^{-1} * a = 4$$

Therefore, $$\frac{a^{-1}a}{4} = 4 \quad \text{or } a^{-1} = \frac{16}{a}$$

Thus, the inverse of element a in G is $\frac{16}{a}$

Commutative: The operation * on G is commutative.

Since, $$a*b = \frac{ab}{4} = b * a$$

Thus, the algebraic system (G, *) is closed, associative, identity element, inverse and commutative. Hence, the system (G, *) is an abelian group.

Product of Groups:

Theorem: Prove that if $(G_1,*_1)$and $(G_2,*_2)$ are groups, then $G = G_1 \times G_2$ i.e., $(G, *)$ is a group with operation defined by $(a_1,b_1)*(a_2,b_2)=(a_1,*_1,a_2, b_1 *_2 b_2)$.

Proof: To prove that $G_1 \times G_2$ is a group, we have to show that $G_1 \times G_2$ has the associativity operator, has an identity and also exists inverse of every element.

Associativity. Let $a, b, c \in G_1 \times G_2$,then

So, $a * (b * c) = (a_1,a_2)*((b_1,b_2)*(c_1,c_2))$
$= (a_1,a_2)*(b_1 *_1 c_1,b_2 *_2 c_2)$
$= (a_1 *_1 (b_1 *_1 c_1),a_2 *_2 (b_2 *_2 c_2)$
$= ((a_1 *_1 b_1) *_1 c_1,(a_2 *_2 b_2) *_2 c_2)$
$= (a_1 *_1 bu_1,a_2 *_2 b_2)*(c_1,c_2)$
$= ((a_1,a_2)*(b_1,b_2))*(c_1,c_2)$
$= (a * b) * c.$

Identity: Let e_1 and e_2 are identities for G_1 and G_2 respectively. Then, the identity for $G_1 \times G_2$ is $e=(e_1,e_2)$.Assume same $a \in G_1 \times G_2$

Then, $a * e = (a_1,a_2)*(e_1,e_2)$
$= (a_1 *_1 e_1,a_2 *_2 e_2)$
$= (a_1,a_2)=a$

Similarly, we have $e * a = a$.

Inverse: To determine the inverse of an element in $G_1 \times G_2$, we will determine it component wise i.e.,

$a^{-1}=(a_1,a_2)^{-1}=(a_1^{-1},a_2^{-1})$

Now to verify that this is the exact inverse, we will compute $a * a^{-1}$ and $a^{-1}*a$.

Now, $a * a^{-1}=(a_1,a_2)*(a_1^{-1},a_2^{-1})$
$= (a_1 *_1 a_1^{-1},a_2 *_2 a_2^{-1})=(e_1,e_2)=e$

Similarly, we have $a^{-1}*a=e$.

Thus, $(G_1 \times G_2,*)$ is a group.

In general, if $G_1,G_2,....G_n$ are groups, then $G = G_1 \times G_2 \times .....\times G_n$ is also a group.

Cosets are fundamental concepts in abstract algebra, particularly in the study of groups. Understanding costs is crucial for analyzing group structures, subgroup relations, and their applications in various fields, including engineering mathematics. This article provides an in-depth exploration of cosets, covering definitions, properties, and practical implications.

Definition of Cosets

Given a group G and a subgroup H of G, a coset is a set formed by multiplying all elements of H by a fixed element from G.

There are two types of cosets:

Left Coset: A left coset of H in G is the set $gH=\{gh \mid h \in H\}$ for a fixed element $g \in G$.

Right Coset: A right coset of H in G is the set $Hg=\{hg \mid h \in H\}$ for a fixed element $g \in G$.

Types of Cosets

Left Cosets

A left coset is created by multiplying every element of a subgroup H by a fixed element g from the group G on the left. Formally, the left coset of H with respect to g is defined as:

$$gH=\{gh \mid h \in H\}$$

Left cosets have a key property: if two elements g1 and g2 of G belong to the same left coset, then the cosets g1H and g2H are identical.

Right Cosets

A right coset is similar to a left coset, except the fixed element g multiplies each element of H on the right. The right coset of H with respect to g is:

$$Hg=\{hg|h\in H\}$$

Similar to left cosets, right cosets partition the group G into disjoint subsets.

Properties of Cosets

Cosets possess several important properties that make them a central concept in group theory:

Partitioning: The left cosets (or right cosets) of a subgroup H partition the group G into disjoint sets. This means every element of G belongs to exactly one left coset and one right coset.

Cardinality: Each coset (left or right) of a subgroup H has the same number of elements as H. If $|H|$ is the order (number of elements) of H, then $|gH|=|H|$ for any $g\in G$.

Equivalence Relation: The relation defined by $g1\sim g2$ if and only if $H=g2H$ (for left cosets) or $Hg1=Hg2$ (for right cosets) is an equivalence relation on G.

Lagrange's Theorem

Statement and Proof

Lagrange's Theorem is a foundational result in group theory that relates the order of a subgroup to the order of the entire group. It states:

Lagrange's Theorem: Let G be a finite group, and let H be a subgroup of G. Then the order of H divides the order of G. In other words:

$$|G| = |H| \times [G:H]$$

where $[G:H]$ is the index of H in G, representing the number of distinct cosets of H in G.

Proof: Consider the set of all left cosets of H in G. Since these cosets partition G and each coset has the same number of elements as H, the total number of elements in G is the number of cosets times the number of elements in H. Thus, $|G| = |H| \times [G:H]$, proving the theorem.

Implications for Cosets and Subgroups

Lagrange's Theorem has several important implications:

The order of any subgroup H of a finite group G must divide the order of G.

If $|G|$ is a prime number, then G has no proper subgroups other than the trivial subgroup and G itself.

The number of distinct left cosets of H in G is equal to the index [G:H].

Applications of Cosets

Applications of cosets run from several areas of mathematics to others, some of which are:

Cryptography: This has its application in Cryptographic Algorithms, mostly on Group-Based Symmetric-Key Encryption Algorithms.

Coding Theory: In coding theory, the use of cosets is applied to the analysis in the construction and decoding of error-correcting codes.

Symmetry Groups: Cosets are used in the study of the coset element representatives of symmetry groups of objects that aid in structuring and classification of the symmetries.

Permutation Group

A **permutation group** is a mathematical concept in group theory, a branch of abstract algebra. It is defined as a group G whose elements are permutations of a given set S, and the group operation is the composition of these permutations.
Key points about permutation groups:

1. **Permutations**: A permutation of a set S is a bijection from Sto S, meaning it rearranges the elements of S.
2. **Symmetric Group**: The set of all permutations of a finite set of n elements forms a group under composition, called the **symmetric group**, denoted Sn.
3. **Group Properties**: The permutation group satisfies the group axioms:
    - **Closure**: Composition of two permutations is a permutation.

- **Associativity**: Composition of functions is associative.
- **Identity**: The identity permutation (no change) acts as the identity element.
- **Inverses**: Every permutation has an inverse permutation that "undoes" its action.

Definition of Normal Subgroup

Let H be a subgroup of G, then H is said to be a normal subgroup of G, if for every x in G and for h in H xh = xh, that is, xhx^{-1} belongs to H.

Now since the above statement is true for all h in H. Therefore, we can have

$xHx^{-1} = \{xhx^{-1}$: for all h in H}, thus normal subgroups of a group G can be defined as:

> A subgroup H of a group G is a normal subgroup $\Leftrightarrow xHx^{-1} \subseteq H$ for every x G, where x may or may not be in H.

Normal subgroups are sometimes also referred to as self-conjugates. Normal subgroups are denoted as $H \triangleleft G$, it is read as "H is a normal subgroup of G".

Examples of Normal Subgroup

Every group has necessarily two trivial normal subgroups, viz., the single identity element of G and G itself.

- Let e be the identity element in G, then {e} will be a trivial subgroup of G. Now for every g in G, there exist g^{-1} in G, then

$geg^{-1} = gg^{-1} = e \in \{e\}$

Thus {e} is the normal subgroup of G.

- Since G is closed with respect to multiplication of its elements, let g, h be any two elements of G, then

$ghg^{-1} = k$ which could be any element in G. Thus, G itself is a normal subgroup.

Another example of a normal subgroup could be the centre of the group. Let Z be the centre of the group G such that

$Z(G) = \{ z \in G \mid \text{for every } g \text{ in } G, gz = zg\}$, that elements of Z commute with every element of G.

Clearly, $gzg^{-1} = gg^{-1}z = ez = z \in Z$; where e is an identity element in G.

Thus, Z is a normal subgroup of G.

Properties of a Normal Subgroup

- The intersection of any two normal subgroups of a group is a normal subgroup.
- The intersection of any collection of normal subgroups is a normal subgroup. That is,

If $N_1, N_2, \ldots, N_r$ are normal subgroups of a group G, then $N_1 \cap N_2 \cap, \ldots, \cap N_r$ is also a normal subgroup of G.

- Every abelian group has a normal subgroup.
- Any group which do not have any normal subgroup other than the trivial normal subgroup is called a simple group.
- If a subgroup is of index 2 in G, that is has only two distinct left or right cosets in G, then H is a normal subgroup of G.
- Every subgroup of a cyclic group is normal.

Definition of Quotient Group

If G is a group and N is a normal subgroup of G, then the set G/N of all cosets of N in G is a group with respect to the composition of the group.

The Quotient Group of G = G/N = { Na | a is in G}

Thus, the elements of G/N are in form of cosets like $Na_1, Na_2, Na_3, \ldots$where $a_1, a_2, a_3, \ldots$ are in G.

Let Nx and Ny be in G/N then we shall prove that (Nx)(Ny) is in G/N

(Nx)(Ny) = NxyN (since N is a normal subgroup.

Let, xy = z in G, as both x and y belong to G.

(Nx)(Ny) = NzN = NNz = Nz ∈ G/N (since NN = N)

Hence, G/N is closed with respect to the multiplication composition of the group.

Let Na, Nb, Nc are in G/N where a, b and c are in G. Then,

[(Na)(Nb)]Nc = N(ab)c = Na(bc) {since associativity holds within G}

= (Na)N(bc) = Na[N(b)N(c)]

Thus, G/N is associative as well.

Since N = Ne ∈ G/N, for any element Na in G/N, we have

(Na)N = (Na)(Ne) = N(ae) = Na = N(ea) = (Ne)(Na) = N(Na)

Where e is the identity element in G

Hence, N is an identity element in G/N.

If a is an element of G then a^{-1} is also an element of G, hence Na and Na^{-1} are elements of G/N such that

(Na)(Na^{-1}) = N(aa^{-1}) = Ne = N

Similarly, (Na^{-1})(Na) = N.

Thus, for every element in G/N there exists an inverse of it in G/N.

With this, we prove that G/N is a group distinctly called the Quotient Group of G.

Example of a Quotient Group

Let G be the addition modulo group of 6, then G = {0, 1, 2, 3, 4, 5} and N = {0, 2} is a normal subgroup of G since G is an abelian group.

Now, G/N = { N+a | a is in G} = {{0, 2}, {1, 3}, {2, 4}} = { N+0, N+1, N+2}

N + 3 = {3, 5} which will be the same as N + 0.

Let us check whether G/N is group

Closure Property: N+1 and N+2 in G/N

$(N + 1)+(N + 2) = N + N + (1 + 2) = N + 3 = N + 0 \in G/N$

Thus G/N is closed with respect to addition modulo 6.

Associativity: $(N+0)[(N+1)+(N+2)] = N + (0+1+2) = N + 3 \in G/N$

$[(N+0)(N+1)]+(N+2) = N + (0+1+2) = N + 3 = N + 0 \in G/N$

Thus G/N is associative with respect to addition modulo 6.

Identity: Now $N +(N + 2) = (N + 0) + (N + 2) = N + 2 = (N + 2) + (N + 0) = (N + 2) + N$

Therefore $N = N + 0 \in G/N$ is the identity element in G/N.

Inverse: Now, $(N + 1)+(N + 2) = N + 3 = N + 0 = (N + 2)+(N + 1)$

Thus, N+1 and N+2 are inverse elements of each other in G/N

Therefore, G/N is a Quotient group of G, which is also abelian, as G/N is commutative as well.

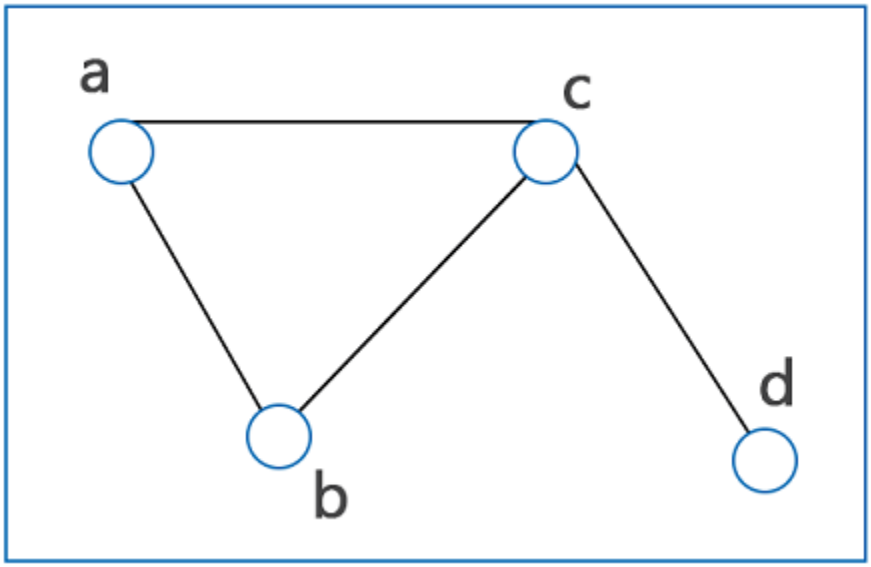
Properties of a Quotient Group

- The identity element of a quotient group is the normal subgroup itself.
- If N is a normal subgroup of G, the Order of G/N is equal to the order of G divided by the order of N.That is, $|G/H| = |G|/|N|$
- Quotient group of an abelian group is abelian, but the converse is not true.
- Every quotient group of a cyclic group is cyclic, but the opposite is not true.
- The quotient group G/G has correspondence to the trivial group, that is, a group with one element.
- The quotient group $G/\{e\}$ has correspondence to the group itself.
- If G is nilpotent then so is the quotient group G/N.
- If G is solvable then the quotient group G/N is as well.

UNIT-4

What is Graph

A graph can be used to show any data in an organized manner with the help of pictorial representation. We can show the relationship between the variable quantities with the help of a graph

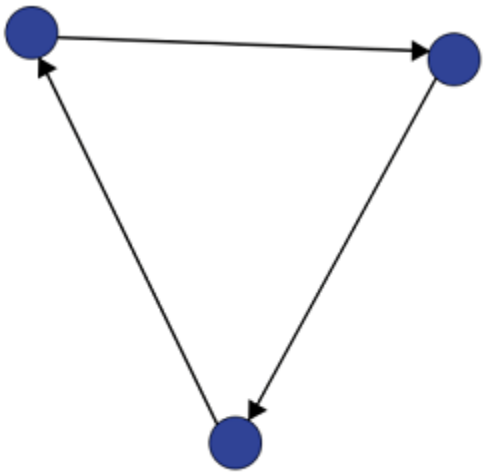


Types of Graph:

There are basically two types of graphs, i.e., Undirected graph and Directed graph. The directed graph and undirected graph are described as follows:

Directed graph:

The directed graph can be made with the help of a set of vertices, which are connected with the directed edges. In the directed graph, the edges have a direction which is associated with the vertices.

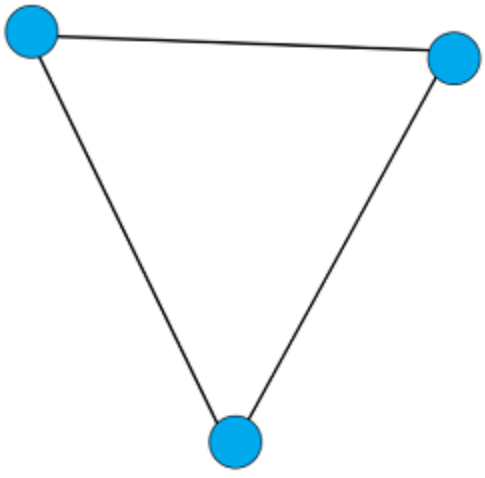


Directed Graph

In the above-directed graph, arrows are used to show the direction.

Undirected Graph

The undirected graph can also be made of a set of vertices which are connected together by the undirected edges. All the edges of this graph are bidirectional. We can sometimes call this type of graph an undirected network.



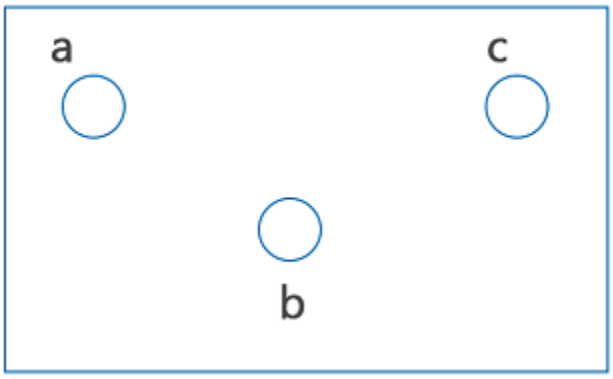
Undirected Graph

In the undirected graph, there is no arrow.

Other types of graph

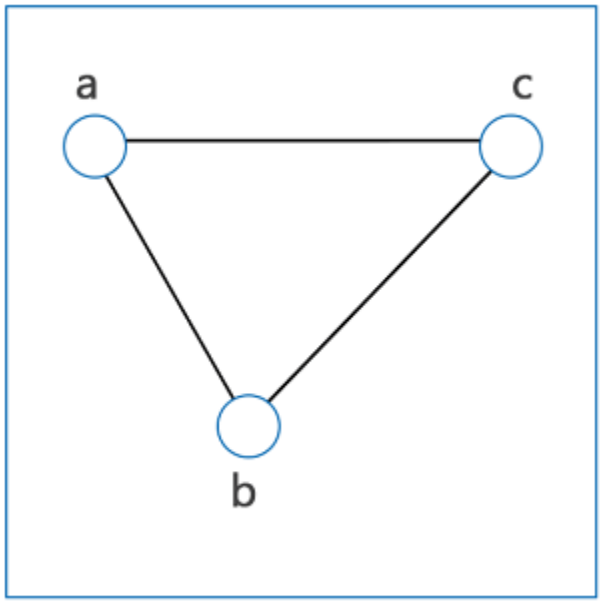
There are also some other types of graphs, which are described as follows:

Null Graph: A graph will be known as the null graph if it contains no edges. With the help of symbol Nn, we can denote the null graph of n vertices. The diagram of a null graph is described as follows:



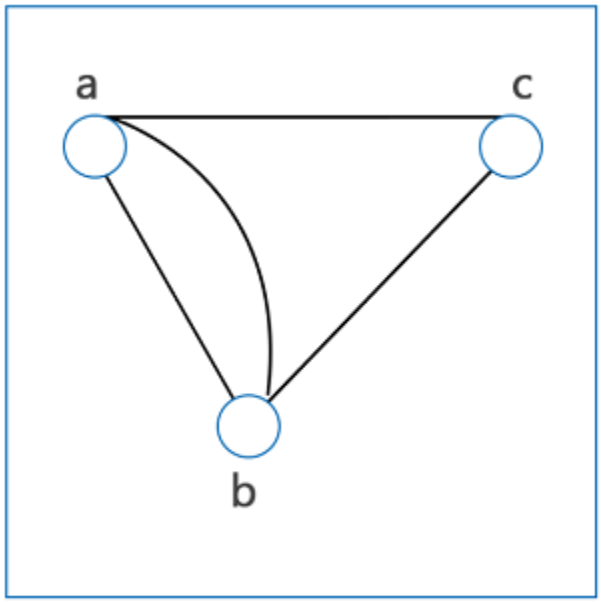
In the above graph, vertices a, b and c are not connected with any edge, and there is no edge. So this graph is a null graph.

Simple Graph: A graph will be known as a simple graph if it does not contain any types of loops and multiple edges. The simple graph must be an undirected graph. The diagram of a simple graph is described as follows:



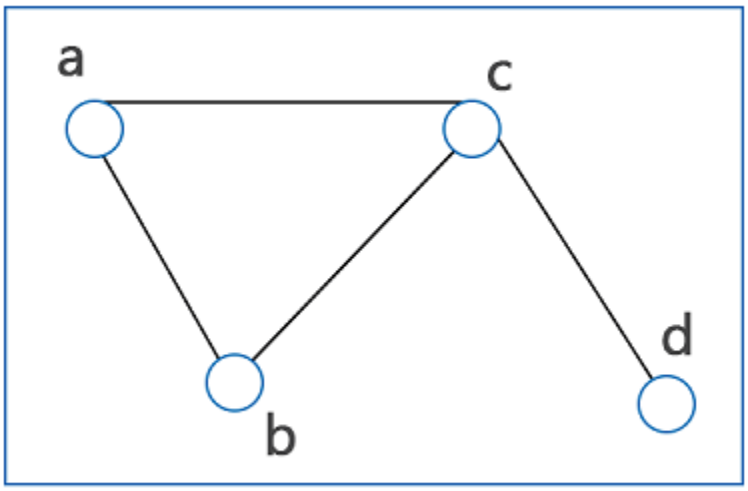
The above graph is an undirected graph and does not contain a loop and multiple edges. So this graph is a simple graph.

Multi-Graph: A graph will be known as a multi-graph if the same sets of vertices contain multiple edges. In this type of graph, we can form a minimum of one loop or more than one edge. The diagram of multi-graph is described as follows:



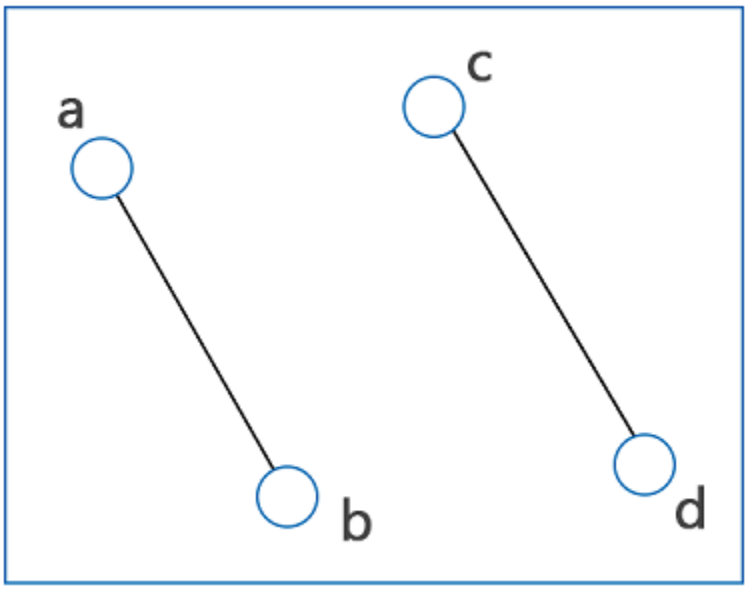
In the above graph, vertices a, b, and c contains more than one edge and does not contain a loop. So this graph is a multi-graph.

Connected Graph: A graph will be known as a connected graph if it contains two vertices that are connected with the help of a path. The diagram of a connected graph is described as follows:



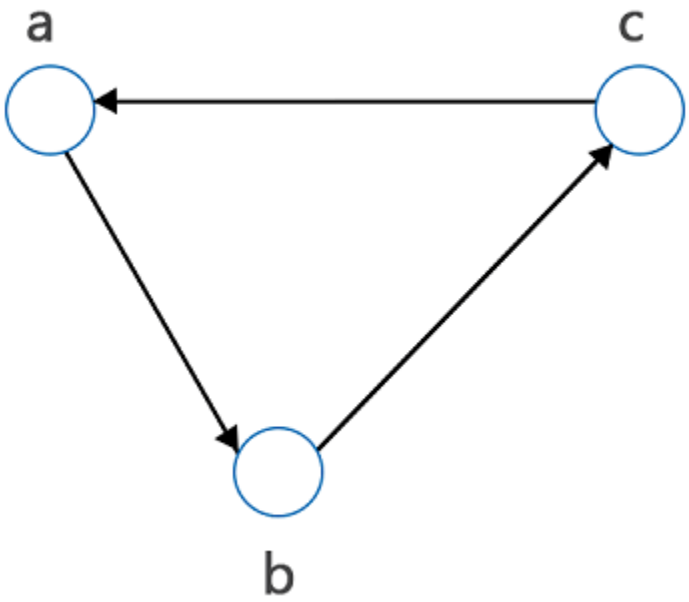
In the above graph, the two vertices, a and b, are connected by a single path. Similarly, other vertices such as (a and c), (c and b), (c and d), (a and d) are all connected by a single path. So this graph is a connected graph.

Disconnected Graph: A graph will be known as the disconnected graph if it contains two vertices which are disconnected with the help of a path. If there is a graph G, which is disconnected, in this case, every maximal connected sub-graph of G will be known as the connected component of the graph G. The diagram of a disconnected graph is described as follows:



In the above graph, there are vertices a, c, and b, d which are disconnected by a path. So this graph is a disconnected graph.

Cycle Graph: A graph will be known as the cycle graph if it completes a cycle. It means that for a cycle graph, the given graph must have a single cycle. With the help of symbol Cn, we can denote a cycle graph with n vertices. The diagram of a cycle graph is described as follows:

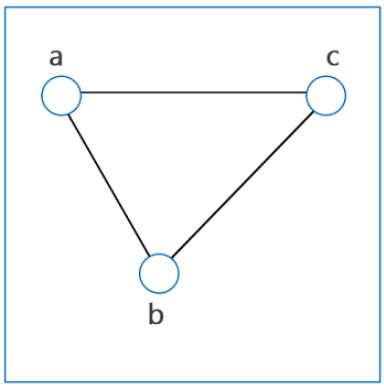


The above graph forms a cycle by path a, b, c, and a. So this graph is a cycle graph.

Complete Graph: A graph will be known as the complete graph if each pair of vertices is connected with the help of exactly one edge. A simple graph will be a complete graph if there are n numbers of vertices which are having exactly one edge between each pair of vertices. With the help of symbol Kn, we can indicate the complete graph of n vertices. In a complete graph, the total number of edges with n vertices is described as follows:

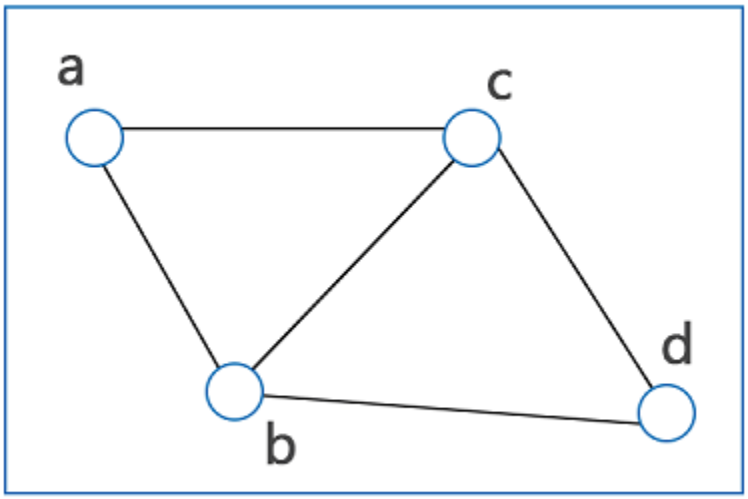
$$n*(n-1)/2$$

The diagram of a complete graph is described as follows:



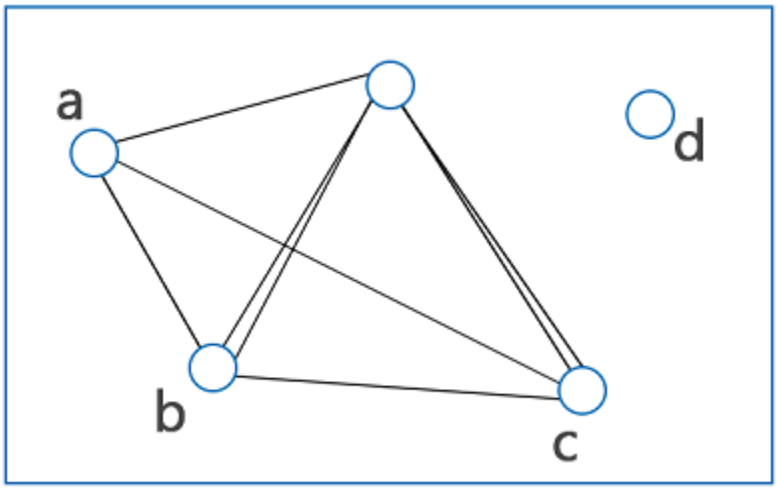
In the above graph, two vertices a, c are connected by a single edge. Similarly, all the other vertices (a and b), and (c and b) are connected by a single edge. So this graph is a connected graph.

Planer Graph: A graph will be known as the planer graph if it is drawn in a single plane and the two edges of this graph do not cross each other. In this graph, all the nodes and edges can be drawn in a plane. The diagram of a planer graph is described as follows:



In the above graph, there is no edge which is crossed to each other, and this graph forms in a single plane. So this graph is a planer graph.

Non-planer graph: A given graph will be known as the non-planer graph if it is not drawn in a single plane, and two edges of this graph must be crossed each other. The diagram of a non-planer graph is described as follows:



In the above graph, there are many edges that cross each other, and this graph does not form in a single plane. So this graph is a non-planer graph.

Properties of a Graph

- The root can be described as a starting point of the network.
- A graph will be known as the assortative graph if nodes of the same types are connected to one another. The graph will be known as the disassortative graph in all the other cases.
- If a cycle graph contains a single cycle, then that type of cycle graph will be known as a graph.
- If a single edge is used to connect all the pairs of vertices, then that type of graph will be known as the complete graph.
- If there is the same direction or reverse direction in which each pair of vertices are connected, then that type of graph will be known as the symmetry graph.
- If there is a graph which has a single graph, then that type of graph will be a path graph.

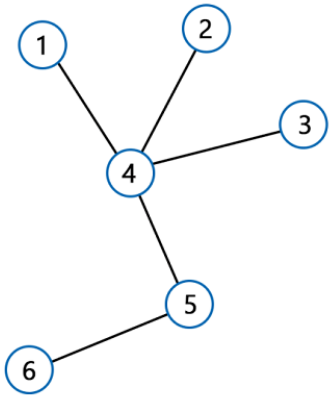
Tree, Degree, Cycle and many more

In the graph representation, we can use certain terms, i.e., Tree, Degree, Cycle and many more. Now we will learn about them in detail.

Trees:

A tree is a type of graph which has undirected networks. The tree can have only one path to connect any two vertices. British mathematician Arthur Cayley was introduced

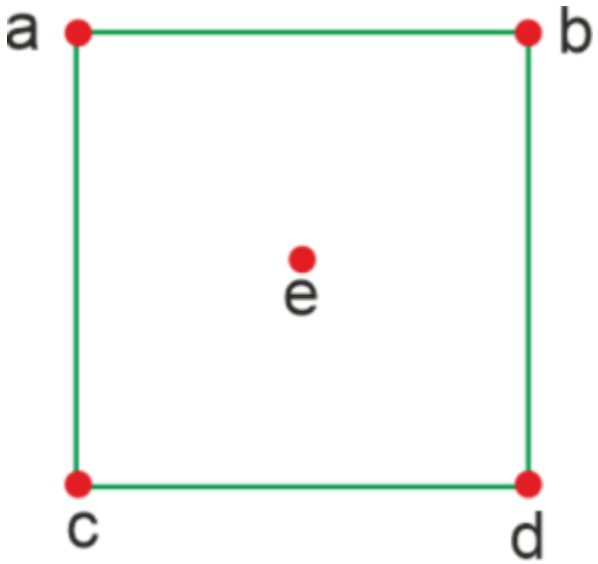
the concept of a tree in 1857. The tree cannot have loops and cycles. The diagram of a tree is described as follows:



The above graph is an undirected graph which has only a path to connect the two vertices. So this graph is a tree.

Degree:

In any graph, the degree can be calculated by the number of edges which are connected to a vertex. The symbol deg(v) is used to indicate the degree where v is used to show the vertex of a graph. So basically, the degree can be described as the measure of a vertex.



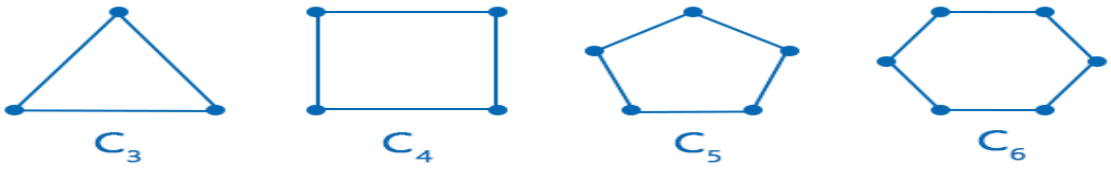
In the above graph, there are total of 5 vertices. The degree of vertex a is 2, the degree of vertex b is 2, the degree of vertex c is 2, the degree of vertex d is 2, and the degree of vertex e is zero.

Cycle:

In any graph, a cycle can be described as a closed path that forms a loop. A cycle will be formed in a graph if there is the same starting and end vertex of the graph, which contains a set of vertices. A cycle will be known as a simple cycle if it does not have any repetition of a vertex in a closed circuit. With the help of symbol Cn, we can indicate the cycle graph. The cycle graph can be of two types, i.e., Even cycle and Odd cycle.

- Even cycle: If a graph contains the even number of nodes and edges in a cycle, then that type of cycle will be known as an even cycle.
- Odd cycle: If a graph contains the odd number of nodes and edges in a cycle, then that type of cycle will be known as an odd cycle.

The diagram of a cycle is described as follows:

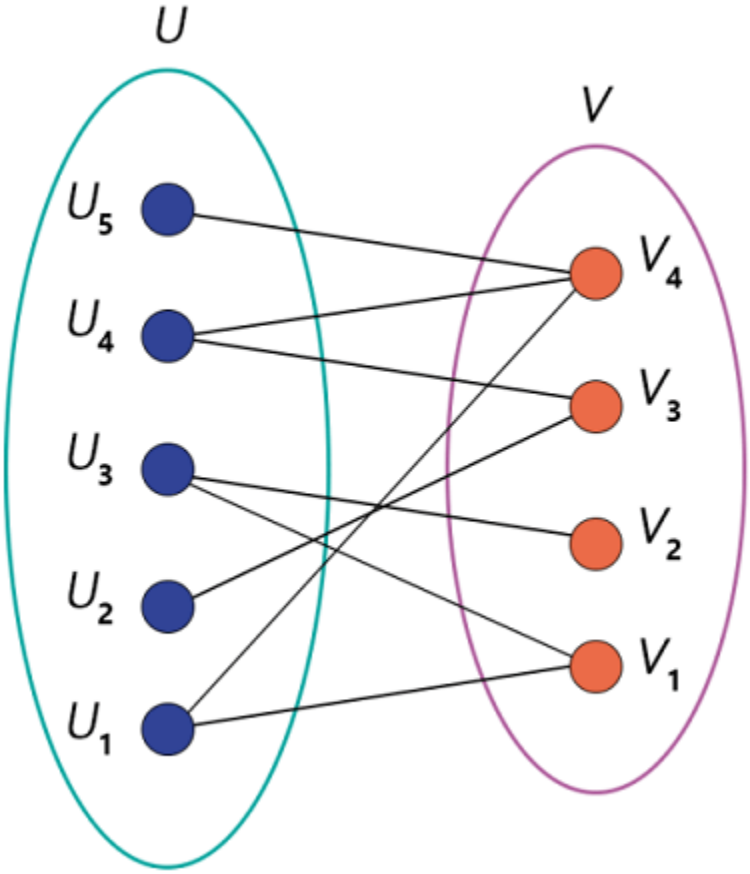


In the above graph, all the graphs have formed a loop, and if we start from any vertex, then we will be able to end the loop of the same vertex. That means in all the above graphs, the starting and end vertex is the same. So this graph is a cycle.

Graph C3 and C5 contain the odd number of vertices and edges, i.e., C3 contains 3 vertices and edges, and graph C5 contain 5 vertices and edges. So graphs C3 and C5 contain the odd cycle. Similarly, graph C4 and C6 contain the even number of vertices and edges, i.e., C4 contain the 4 vertices and edges, and graph C6 contains the 6 vertices and edges. So graphs C4 and C6 contain the even cycle.

Bipartite graph:

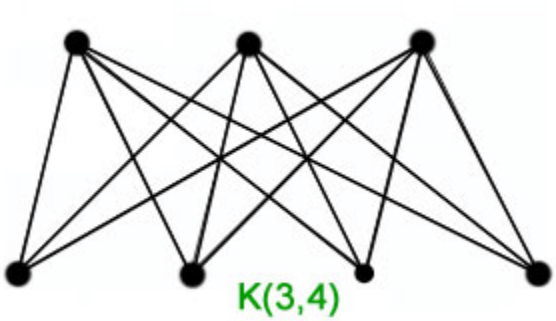
A simple graph will be known as the bipartite graph if there are two independent sets which contain the set of vertices. The vertices of this graph will be connected in such a way that each edge in this graph can have a connection from the first set to the second set. That means the vertices of a first set can only connect with the vertices of a second set. Similarly, the vertices of a second set can only connect with the vertices of a first set. But this graph does not contain any edge which can connect the vertices of same set. The diagram of a bipartite graph is described as follows:



In the above graph, we have two sets of vertices. The Set U contains 5 vertices, i.e., U1, U2, U3, U4, U5, and the set V contains 4 vertices, i.e., V1, V2, V3, and V4. The vertices of set U only have a mapping with vertices of set V. Similarly, vertices of set V have a mapping with vertices of set U. Set U and set V does not have a connection to the same set of vertices. So this graph is a bipartite graph.

Complete Bipartite graph

A graph will be known as the complete bipartite graph if it contains two sets in which each vertex of the first set has a connection with every single vertex of the second set. With the help of symbol KX, Y, we can indicate the complete bipartite graph. That means the first set of the complete bipartite graph contains the x number of vertices and the second graph contains the y number of vertices.



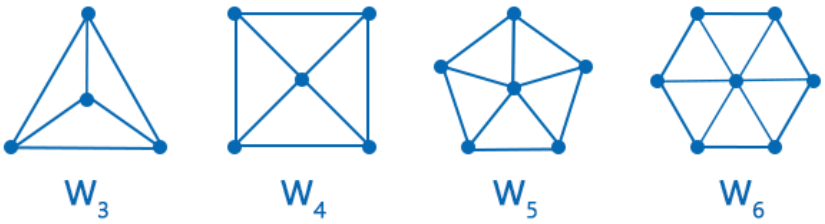
In the above graph, there are a total of two sets. The first set contains the 3 vertices, and the second set contains the 4 vertices. That means the value of x, y will be 3, 4. Every vertex of the first set has a connection with every vertex of a second set. So this graph is a complete bipartite graph.

Wheels:

A wheel and a circle are both similar, but the wheel has one additional vertex, which is used to connect with every other vertex. With the help of symbol Wn, we can indicate the wheels of n vertices with 1 additional vertex. In a wheel graph, the total number of edges with n vertices is described as follows:

2*(n-1)

The diagram of wheels is described as follows:



In the above diagram, we have four graphs W3, W4, W5, and W6. All the graphs have an additional vertex which is used to connect to all the other vertices. So these graphs are the wheels.

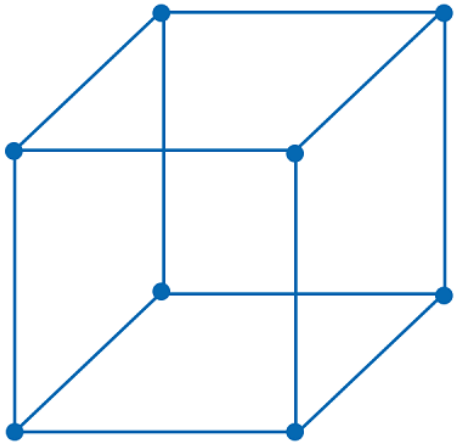
Hypercube:

Hypercube can also be called n cube. This hypercube is similar to a 3-dimensional cube, but this type of cube can have any number of dimensions. The hypercube is a compact, closed, and convex geometrical diagram in which all the edges are perpendicular and have the same amount of length. This cube contains the 2n vertices, and each vertex is

indicated by an n-bit string. With the help of symbol Qn, we can indicate the hypercube of 2n vertices. In a cube graph, the total number of edges with 2n vertices is described as follows:

n*2n-1

The diagram of a hypercube is described as follows:

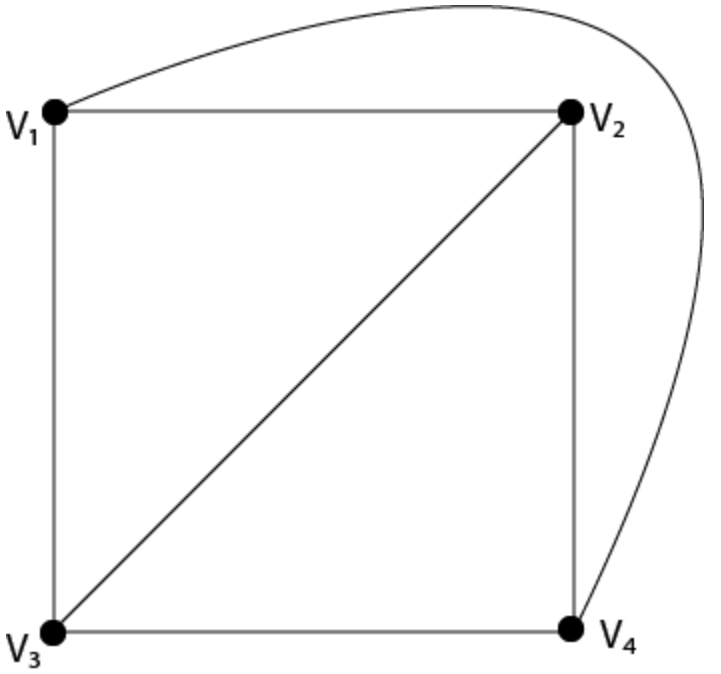


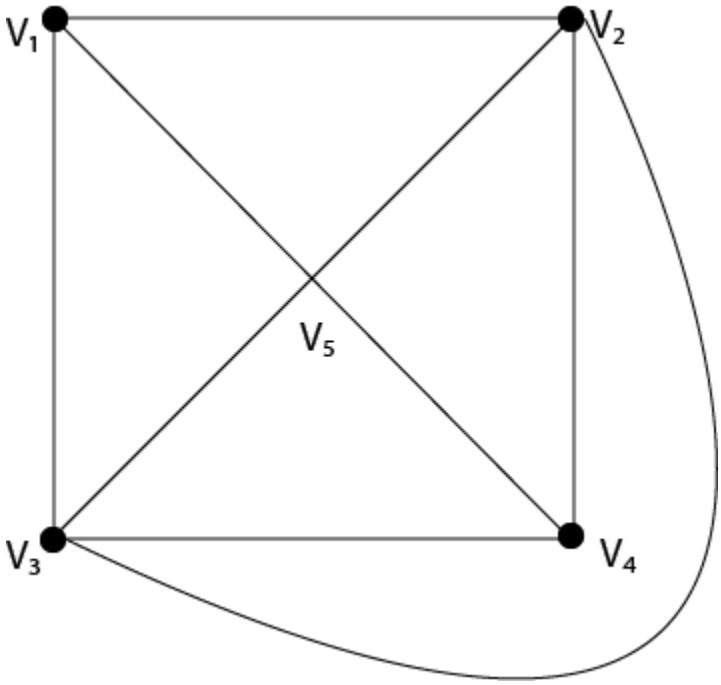
The above graph is compact and closed, and all the edges of this graph are perpendicular and have an equal amount of length. So this graph is a Hypercube.

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.



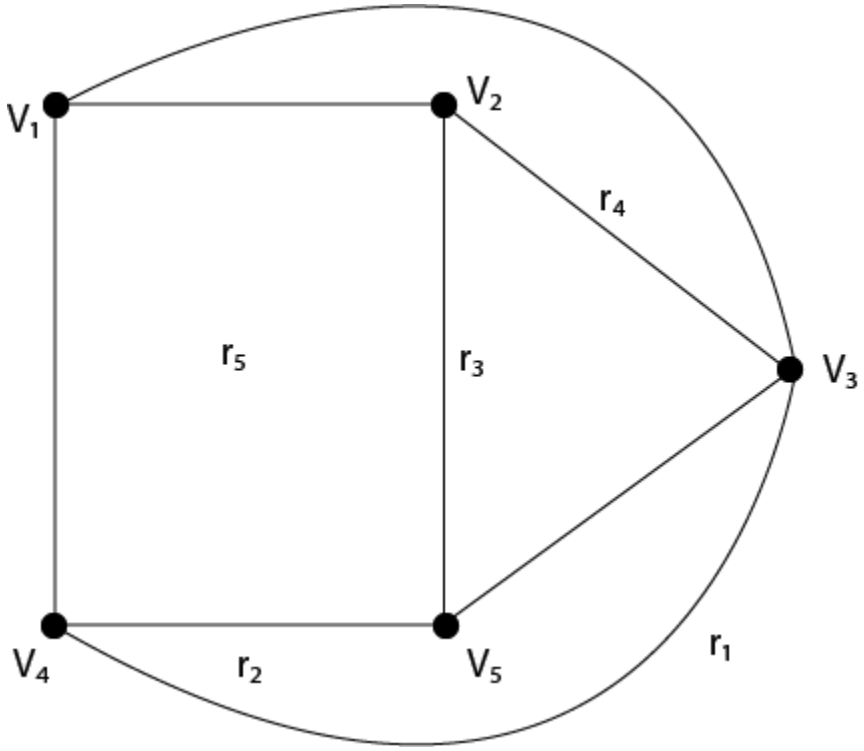


Region of a Graph: Consider a planar graph G=(V,E).A region is defined to be an area of the plane that is bounded by edges and cannot be further subdivided. A planar graph divides the plans into one or more regions. One of these regions will be infinite.

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 $3v-e \geq 6$.
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 $n>3$.

Example: Prove that complete graph K_4 is planar.

Solution: The complete graph K_4 contains 4 vertices and 6 edges.

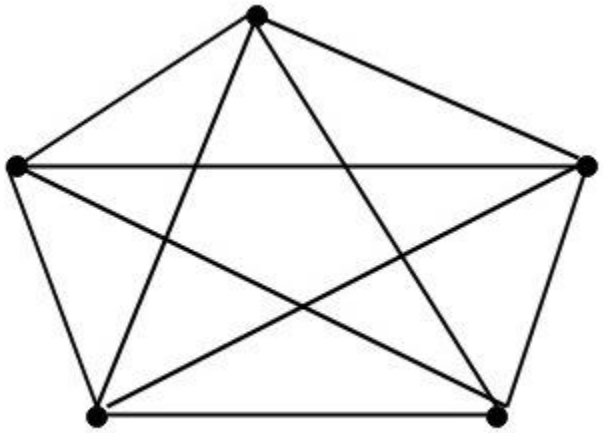
We know that for a connected planar graph $3v-e \geq 6$.Hence for K_4, we have $3x4-6=6$ which satisfies the property (3).

Thus K_4 is a planar graph. Hence Proved.

Non-Planar Graph:

A graph is said to be non planar if it cannot be drawn in a plane so that no edge cross.

Example: The graphs shown in fig are non planar graphs.



K_5

These graphs cannot be drawn in a plane so that no edges cross hence they are non-planar graphs.

Properties of Non-Planar Graphs:

A graph is non-planar if and only if it contains a subgraph homeomorphic to K_5 or $K_{3,3}$

Example1: Show that K_5 is non-planar.

Solution: The complete graph K_5 contains 5 vertices and 10 edges.

Now, for a connected planar graph $3v-e \geq 6$.

Hence, for K_5, we have $3 \times 5-10=5$ (which does not satisfy property 3 because it must be greater than or equal to 6).

Thus, K_5 is a non-planar graph.

Example2: Show that the graphs shown in fig are non-planar by finding a subgraph homeomorphic to K_5 or $K_{3,3}$.

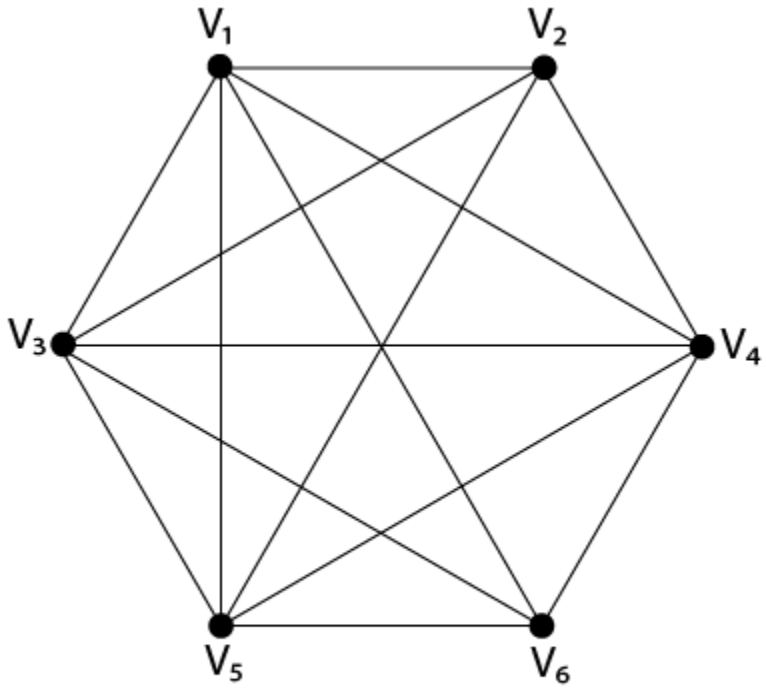


Fig: G_1

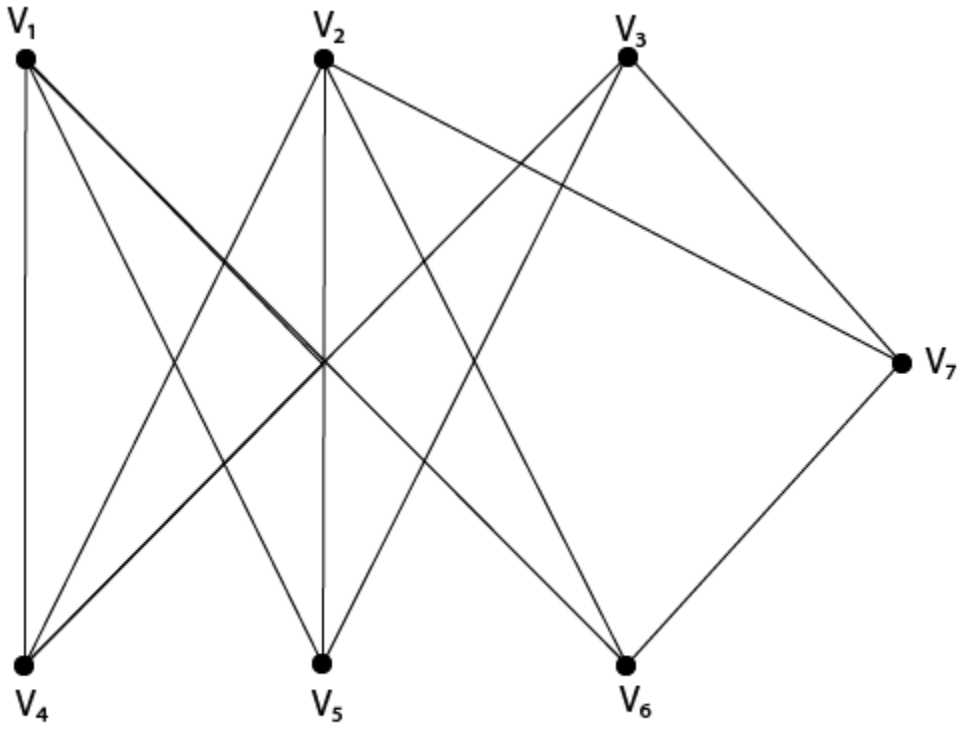


Fig: G_2

Solution: If we remove the edges (V_1,V_4),(V_3,V_4)and (V_5,V_4) the graph G_1,becomes homeomorphic to K_5.Hence it is non-planar.

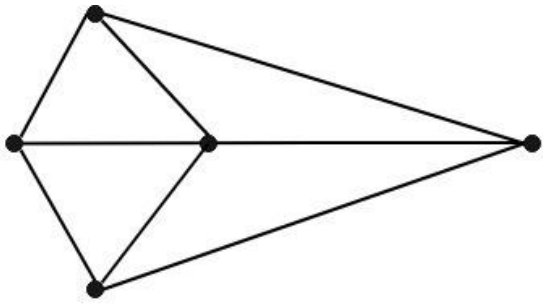
If we remove the edge V_2,$V7$) the graph G_2 becomes homeomorphic to $K_{3,3}$.Hence it is a non-planar.

Graph Coloring:

Suppose that G= (V,E) is a graph with no multiple edges. A vertex coloring of G is an assignment of colors to the vertices of G such that adjacent vertices have different colors. A graph G is M-Colorable if there exists a coloring of G which uses M-Colors.

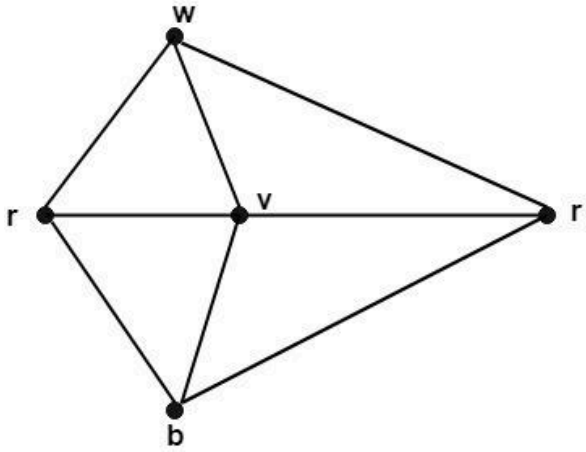
Proper Coloring: A coloring is proper if any two adjacent vertices u and v have different colors otherwise it is called improper coloring.

Example: Consider the following graph and color C={r, w, b, y}.Color the graph properly using all colors or fewer colors.



The graph shown in fig is a minimum 3-colorable, hence x(G)=3

Solution: Fig shows the graph properly colored with all the four colors.



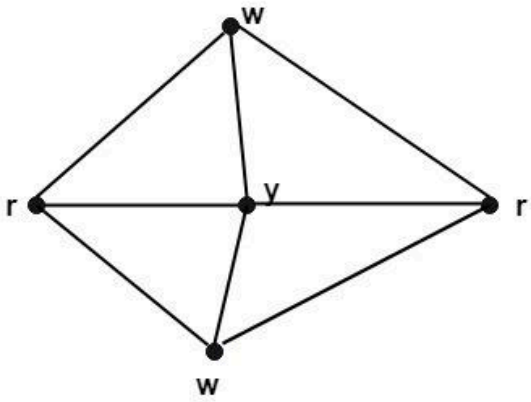
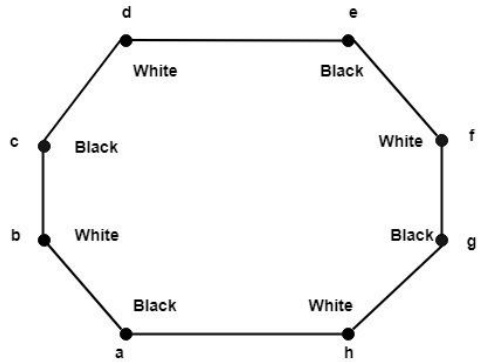


Fig shows the graph properly colored with three colors.



Chromatic number of G: The minimum number of colors needed to produce a proper coloring of a graph G is called the chromatic number of G and is denoted by x(G).

Example: The chromatic number of K_n is n.

Solution: A coloring of K_n can be constructed using n colours by assigning different colors to each vertex. No two vertices can be assigned the same colors, since every two vertices of this graph are adjacent. Hence the chromatic number of K_n=n.

Applications of Graph Coloring

Some applications of graph coloring include:

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

State and prove Handshaking Theorem.

Handshaking Theorem: The sum of degrees of all the vertices in a graph G is equal to twice the number of edges in the graph.

Mathematically it can be stated as:

$$\sum_{v \in V} deg(v) = 2e$$

Proof: Let G = (V, E) be a graph where V = {v_1,v_2, ...} be the set of vertices and E = {e_1,e_2 . . . . . .} be the set of edges. We know that every edge lies between two vertices so it provides degree one to each vertex. Hence each edge contributes degree two for the graph. So the sum of degrees of all vertices is equal to twice the number of edges in G.

Hence, $\sum_{v \in V} deg(v) = 2e$

Proof of Euler's Formula

We prove the formula by applying induction on edges by considering the polyhedra as a simply connected planar graph G with v vertices, e edges and f faces.

If G has zero number of edges, that is e = 0. The graph will have a single with f = 1. Then, by the Euler's formula V − E + F = 1 − 0 + 1 = 2. Thus, it satisfies Euler's formula.

Induction step: Let the Euler's formula is applicable for a graph with n edges.

Let G be a graph with n + 1 edges.

Case I: If G is a tree and does not contain any cycle. It can be easily proven by induction for trees with any number of edges.

Case II: If G is not a tree and contains atleast one cycle. Choose an edge e_1 in G which divides the given region into two different parts and remove that edge e_1 to get another graph G''.

Faces in G > Faces in G''

Now, G has n + 1 edges, then G'' has n edges so by the hypothesis G'' satisfies the Euler's formula. For G'', V' − E' + F' = 2 where V' = v, E' = e − 1 and F' = f − 1.

Now, substituting these values, we get

V' − E' + F' = 2

⇒ v − e + 1 + f − 1 = 2

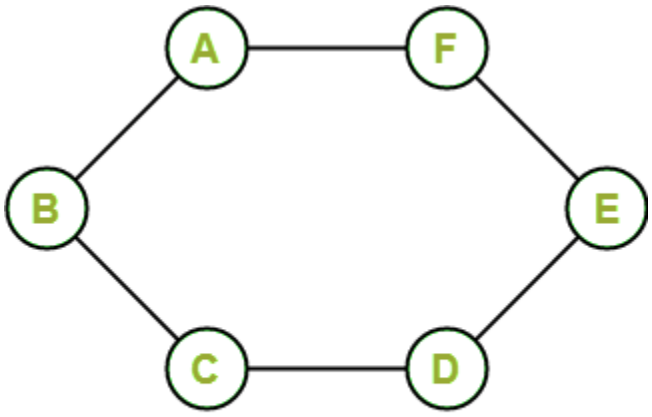
⇒ v − e + f = 2

Hence, Euler's formula is applicable for n + 1 edges.

This proves the Euler's formula.

Hamiltonian Graph

The graph will be known as a Hamiltonian graph if there is a closed walk in a connected graph, which passes each and every vertex of the graph exactly once except the root vertex or starting vertex. The Hamiltonian walk must not repeat any edge. One more definition of a Hamiltonian graph says a graph will be known as a Hamiltonian graph if there is a connected graph, which contains a Hamiltonian circuit. The vertex of a graph is a set of points, which are interconnected with the set of lines, and these lines are known as edges. The example of a Hamiltonian graph is described as follows:



Example of Hamiltonian Graph

- In the above graph, there is a closed walk ABCDEFA.
- Except for the starting vertex, it passed through every vertex of the graph exactly once.
- At the time of walk, the edges are not repeating.
- Due to all the reasons, we can say that this graph is a Hamiltonian graph.

In other words, we can say that the above graph contains a Hamiltonian circuit. That's why this graph is a Hamiltonian graph.

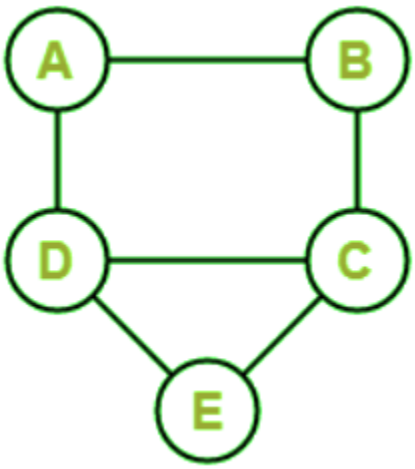
Hamiltonian Path

In a connected graph, if there is a walk that passes each and every vertex of a graph only once, this walk will be known as the Hamiltonian path. In this walk, the edges should not be repeated. There is one more definition to describe the Hamiltonian path: if a connected graph contains a Path with all the vertices of the graph, this type of path will be known as the Hamiltonian path.

Examples of Hamiltonian Path

There are a lot of examples of the Hamiltonian path, which are described as follows:

Example 1: In the following graph, we have 5 nodes. Now we have to determine whether this graph contains a Hamiltonian path.

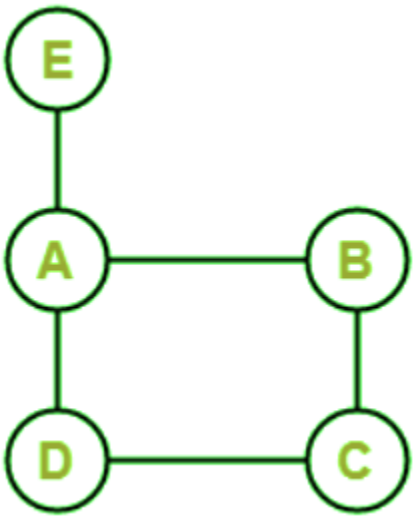


Solution:

In the above graph, we can see that when we start from A, then we can go to B, C, D, and then E. So this is the path that contains all the vertices (A, B, C, D, and E) only once, and there is no repeating edge. That's why we can say that this graph has a Hamiltonian path, which is described as follows:

Hamiltonian path = ABCDE

Example 2: In the following graph, we have 5 nodes. Now we have to determine whether this graph contains a Hamiltonian path.

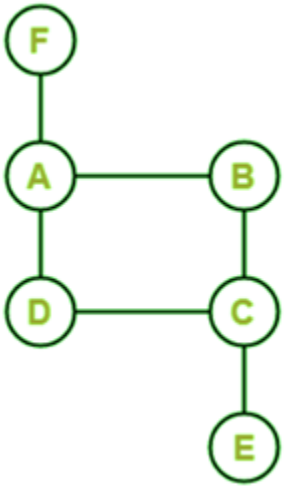


Solution:

In the above graph, we can see that when we start from the E, then we can go to A, B, C, and then D. So this is the path that contains all the vertices (A, B, C, D, and E) only once, and there is no repeating edge. That's why we can say that this graph has a Hamiltonian path, which is described as follows:

Hamiltonian path = EABCD

Example 3: In the following graph, we have 5 nodes. Now we have to determine whether this graph contains a Hamiltonian path.



Solution:

In the above graph, we can see that there is no path that covers all the vertices (A, B, C, D, and E) only once. If we start from F, then we will go to A, B, C. After C, we can either go to D or F, but we cannot go to both the vertices. That's why we can say that this graph does not contain a Hamiltonian path.

Hamiltonian Circuit

In a connected graph, if there is a walk that passes each and every vertex of the graph only once and after completing the walk, return to the starting vertex, then this type of walk will be known as a Hamiltonian circuit. For the Hamiltonian circuit, there must be no repeated edges. We can also be called Hamiltonian circuit as the Hamiltonian cycle.

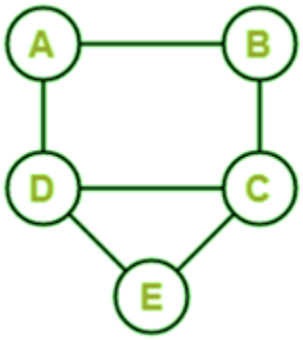
There are some more definitions of the Hamiltonian circuit, which are described as follows:

- If there is a Hamiltonian path that begins and ends at the same vertex, then this type of cycle will be known as a Hamiltonian circuit.
- In the connected graph, if there is a cycle with all the vertices of the graph, this type of cycle will be known as a Hamiltonian circuit.
- A closed Hamiltonian path will also be known as a Hamiltonian circuit.

Examples of Hamiltonian Circuit

There are a lot of examples of the Hamiltonian circuit, which are described as follows:

Example 1: In the following graph, we have 5 nodes. Now we have to determine whether this graph contains a Hamiltonian circuit.

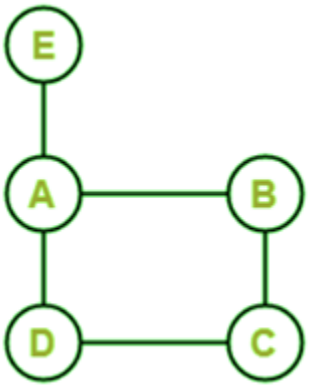


Solution: =

The above graph contains the Hamiltonian circuit if there is a path that starts and ends at the same vertex. So when we start from the A, then we can go to B, C, E, D, and then A. So this is the path that contains all the vertices (A, B, C, D, and E) only once, except the starting vertex, and there is no repeating edge. That's why we can say that this graph has a Hamiltonian circuit, which is described as follows:

Hamiltonian circuit = ABCDEA

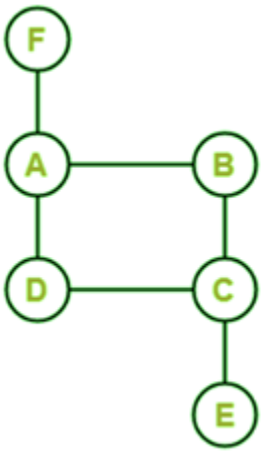
Example 2: In the following graph, we have 5 nodes. Now we have to determine whether this graph contains a Hamiltonian circuit.



Solution:

The above graph contains the Hamiltonian circuit if there is a path that starts and ends at the same vertex. So we will start from the E, then we can go A, B, C, D. To react to the same vertex E, we can have to again go to vertex A. So this is the path that contains all the vertices (A, B, C, D, and E), but vertex A is repeated to reach the initial vertex. That's why we can say that this graph does not contain a Hamiltonian circuit.

Example 3: In the following graph, we have 6 nodes. Now we have to determine whether this graph contains a Hamiltonian circuit.



Solution:

The above graph contains the Hamiltonian circuit if there is a path that starts and ends at the same vertex. So we will start from the F, then we can go A, B, C, E. This path does not cover all the vertices (A, B, C, D, E, and F). That's why we can say that this graph does not contain a Hamiltonian circuit.

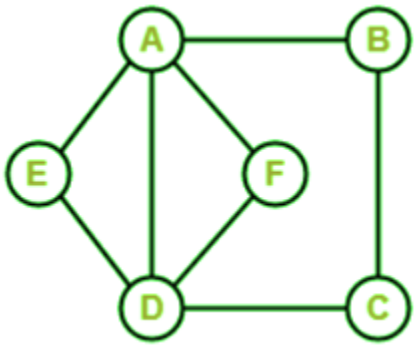
Important Points

- If we remove one of the edges of Hamiltonian path, then it will be converted into any Hamiltonian circuit.
- If any graph has a Hamiltonian circuit, then it will also have the Hamiltonian path. But the vice versa in this case will not be possible.
- A graph can contain more than one Hamiltonian path and Hamiltonian circuit.

Examples of Hamiltonian Graph:

There are a lot of examples of the Hamiltonian graphs, which are described as follows:

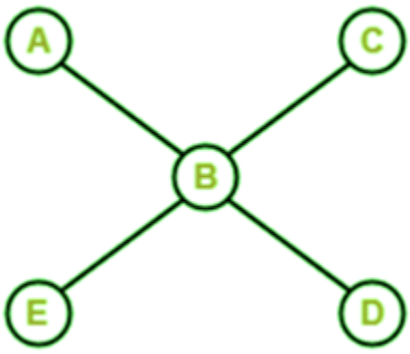
Example 1: In the following graph, we have 6 nodes. Now we have to determine whether this graph is a Hamiltonian graph.



Solution:

This graph does not contain a Hamiltonian path because when we start from A, then we can go to vertices B, C, D, and E, but in this path, vertex F is not covered. This graph does not contain the Hamiltonian circuit also because when we start from A, we can go to vertices B, C, D, E, and A. This can make a circuit, but vertex F is also not covered in this path. So we can say that this graph does not contain a Hamiltonian path and Hamiltonian circuit. Hence, this graph is not a Hamiltonian Graph.

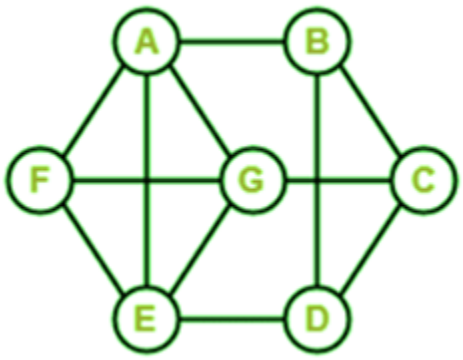
Example 2: In the following graph, we have 5 nodes. Now we have to determine whether this graph is a Hamiltonian graph.



Solution:

This graph does not contain a Hamiltonian path because when we start from A and B, then we can go to one of the three places C, D, and E. So this path does not cover all the vertices. This graph does not contain the Hamiltonian circuit also because any path cannot cover all the vertices and make a circuit. So we can say that this graph is not a Hamiltonian path and a Hamiltonian circuit. Hence, this graph is not a Hamiltonian Graph.

Example 3: In the following graph, we have 7 nodes. Now we have to determine whether this graph is a Hamiltonian graph.



Solution:

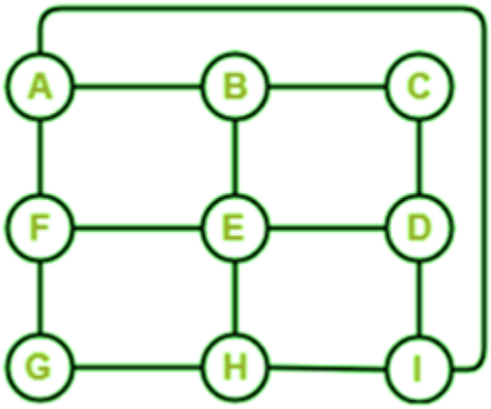
This graph contains a Hamiltonian path because when we start from A, then we can go to vertices B, C, D, E, F, and G. This graph also contains the Hamiltonian circuit because when we start from A, then we can go to vertices B, C, D, E, F, G, and A. So we can say that this graph contains a Hamiltonian path and Hamiltonian circuit. The Hamiltonian path and Hamiltonian circuit are described as follows:

ABCDEFG is the Hamiltonian Path of the above graph.

ABCDEFGA is the Hamiltonian circuit of the above graph.

Hence, this graph is a Hamiltonian Graph.

Example 4: In the following graph, we have 9 nodes. Now we have to determine whether this graph is a Hamiltonian graph.



Solution:

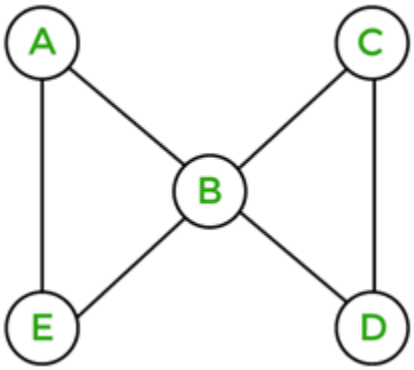
This graph contains a Hamiltonian path because when we start from A, then we can go to vertices B, C, D, E, F, G, H, and I. This graph also contains the Hamiltonian circuit because when we start from A, then we can go to vertices B, C, D, E, F, G, H, I, and A. So we can say that this graph contains a Hamiltonian path and Hamiltonian circuit. The Hamiltonian path and Hamiltonian circuit are described as follows:

ABCDEFGHI is the Hamiltonian Path of the above graph.

ABCDEFGHIA is the Hamiltonian circuit of the above graph.

Hence, this graph is a Hamiltonian Graph.

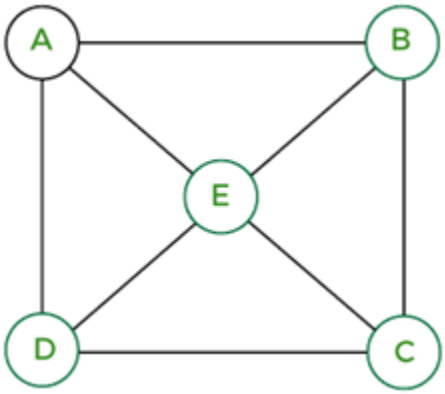
Example 5: In the following graph, we have 5 nodes. Now we have to determine whether this graph is a Hamiltonian graph.



Solution:

This graph does not contain a Hamiltonian path because when we start from A, then we can go to vertices B, C, and D, but in this path, vertex E is not covered. Because of the same reason, this graph also does not contain the Hamiltonian circuit. So we can say that this graph is not a Hamiltonian path and a Hamiltonian circuit. Hence, this graph is not a Hamiltonian Graph.

Example 6: In the following graph, we have 5 nodes. Now we have to determine whether this graph is a Hamiltonian graph.



Solution:

This graph contains a Hamiltonian path because when we start from A, then we can go to vertices B, C, D, and E. This graph also contains the Hamiltonian circuit because when we start from A, then we can go to vertices B, C, D, E, and A. So we can say that this graph contains a Hamiltonian path and Hamiltonian circuit. The Hamiltonian path and Hamiltonian circuit are described as follows:

ABCDE is the Hamiltonian Path of the above graph.

ABCDEA is the Hamiltonian circuit of the above graph.

Hence, this graph is a Hamiltonian Graph.

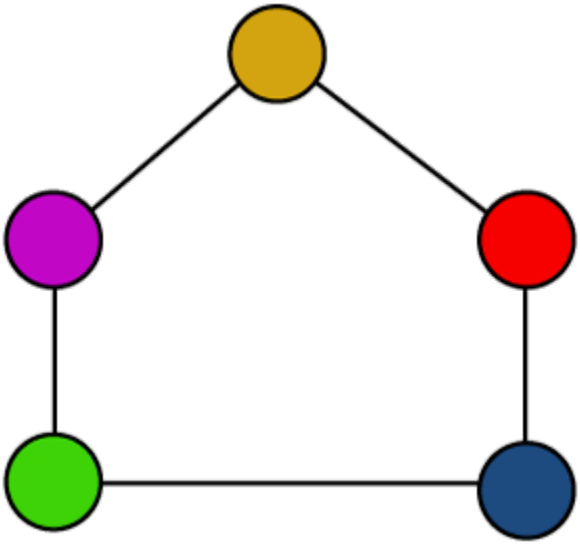
Chromatic Number of graphs

Graph coloring

Graph coloring can be described as a process of assigning colors to the vertices of a graph. In this, the same color should not be used to fill the two adjacent vertices. We can also call graph coloring as Vertex Coloring.

Example of Graph coloring

In this graph, we are showing the properly colored graph, which is described as follows:



The above graph contains some points, which are described as follows:

- The same color cannot be used to color the two adjacent vertices.
- Hence, we can call it as a properly colored graph.

Applications of Graph coloring

There are various applications of graph coloring. Some of their important applications are described as follows:

- Assignment
- Map coloring
- Scheduling the tasks
- Sudoku
- Prepare time table

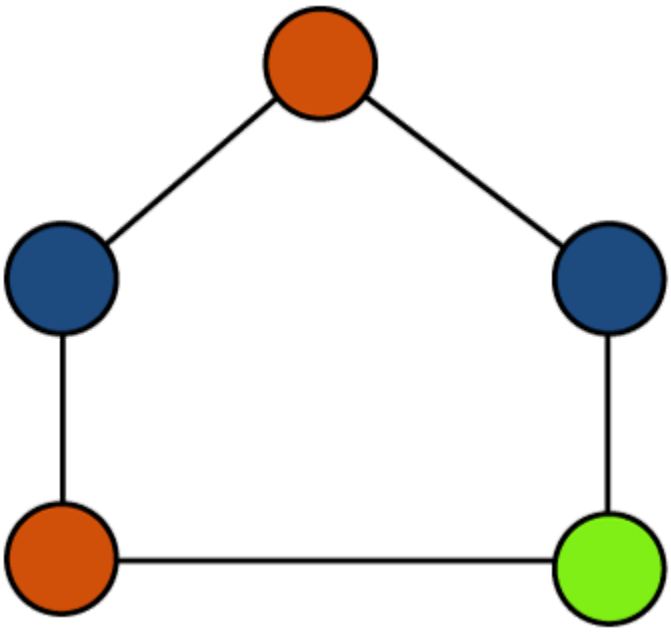
- Conflict resolution

Chromatic Number

The chromatic number can be described as the minimum number of colors required to properly color any graph. In other words, the chromatic number can be described as a minimum number of colors that are needed to color any graph in such a way that no two adjacent vertices of a graph will be assigned the same color.

Example of Chromatic number:

To understand the chromatic number, we will consider a graph, which is described as follows:



The above graph contains some points, which are described as follows:

- The same color is not used to color the two adjacent vertices.
- The minimum number of colors of this graph is 3, which is needed to properly color the vertices.
- Hence, in this graph, the chromatic number = 3
- If we want to properly color this graph, in this case, we are required at least 3 colors.

Types of Chromatic Number of Graphs:

There are various types of chromatic number of graphs, which are described as follows:

Cycle Graph:

A graph will be known as a cycle graph if it contains 'n' edges and 'n' vertices (n >= 3), which form a cycle of length 'n'. There can be only 2 or 3 number of degrees of all the vertices in the cycle graph.

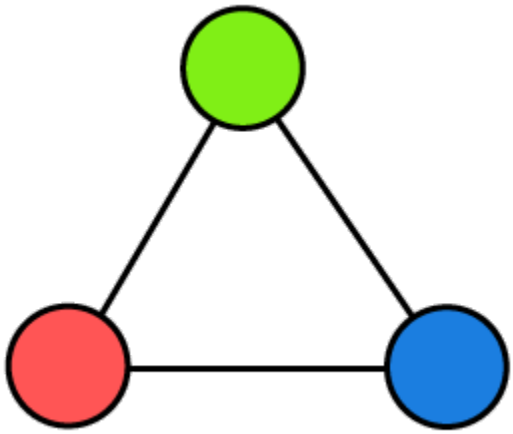
Chromatic number:

1. The chromatic number in a cycle graph will be 2 if the number of vertices in that graph is even.
2. The chromatic number in a cycle graph will be 3 if the number of vertices in that graph is odd.

Examples of Cycle graph:

There are various examples of cycle graphs. Some of them are described as follows:

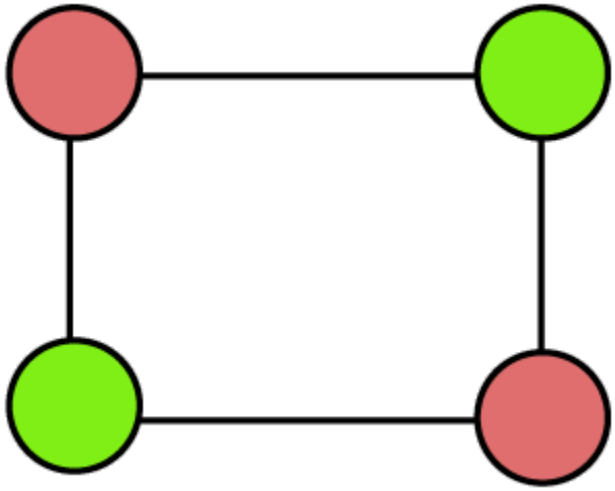
Example 1: In the following graph, we have to determine the chromatic number.



Solution: In the above cycle graph, there are 3 different colors for three vertices, and none of the adjacent vertices are colored with the same color. In this graph, the number of vertices is odd. So

Chromatic number = 3

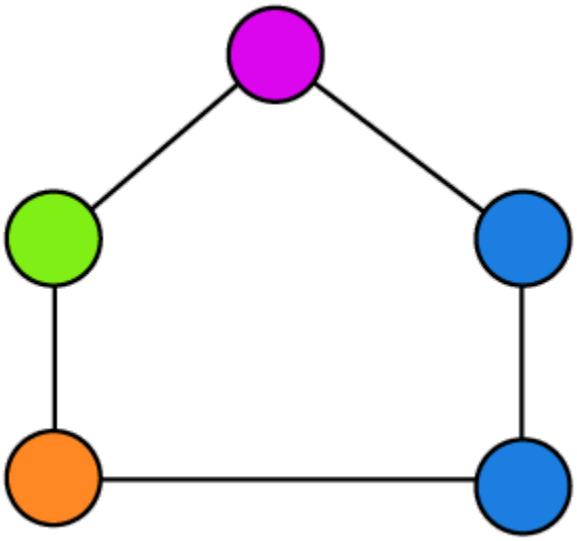
Example 2: In the following graph, we have to determine the chromatic number.



Solution: In the above cycle graph, there are 2 colors for four vertices, and none of the adjacent vertices are colored with the same color. In this graph, the number of vertices is even. So

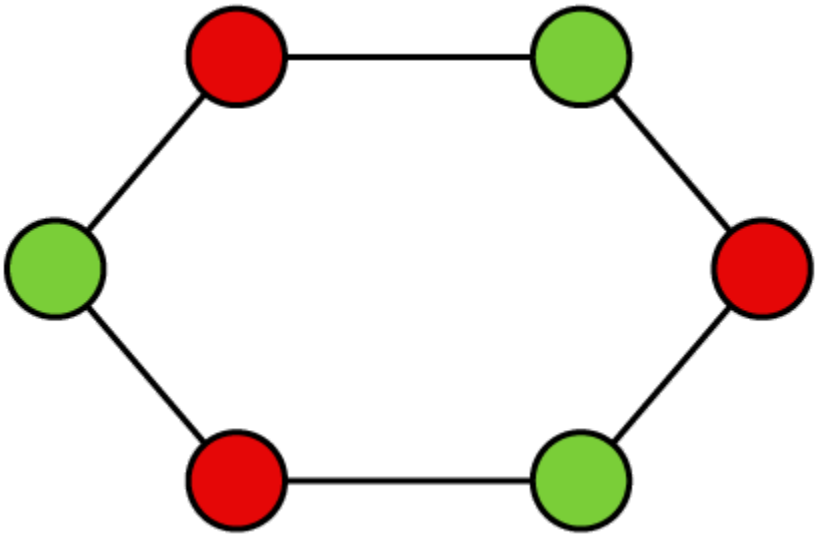
Chromatic number = 2

Example 3: In the following graph, we have to determine the chromatic number.



Solution: In the above graph, there are 4 different colors for five vertices, and two adjacent vertices are colored with the same color (blue). So this graph is not a cycle graph and does not contain a chromatic number.

Example 4: In the following graph, we have to determine the chromatic number.



Solution: In the above graph, there are 2 different colors for six vertices, and none of the adjacent vertices are colored with the same color. In this graph, the number of vertices is even. So

Chromatic number = 2

Planner Graph

A graph will be known as a planner graph if it is drawn in a plane. The edges of the planner graph must not cross each other.

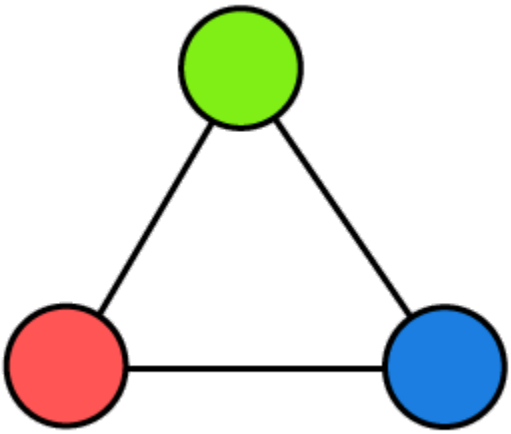
Chromatic Number:

1. In a planner graph, the chromatic Number must be Less than or equal to 4.
2. The planner graph can also be shown by all the above cycle graphs except example 3.

Examples of Planer graph:

There are various examples of planer graphs. Some of them are described as follows:

Example 1: In the following graph, we have to determine the chromatic number.

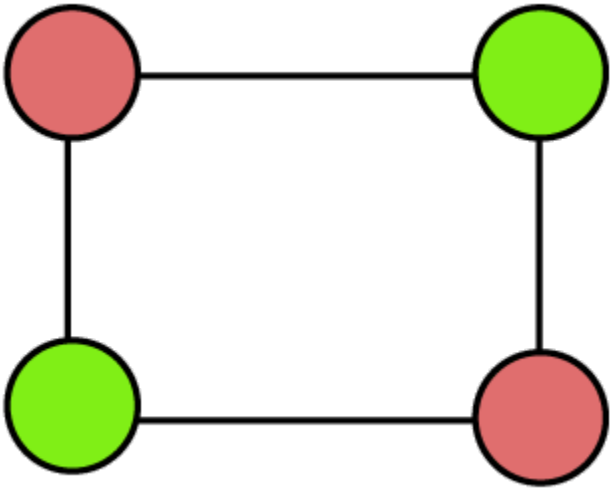


Solution: In the above graph, there are 3 different colors for three vertices, and none of the edges of this graph cross each other. So

Chromatic number = 3

Here, the chromatic number is less than 4, so this graph is a plane graph.

Example 2: In the following graph, we have to determine the chromatic number.

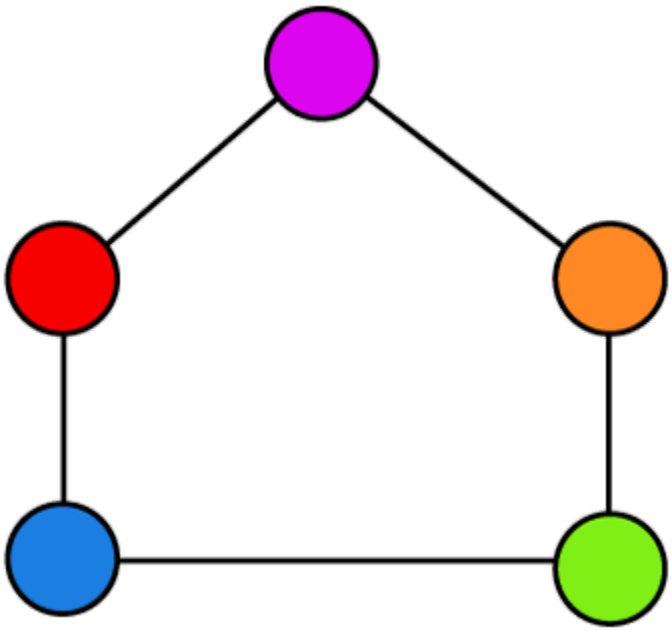


Solution: In the above graph, there are 2 different colors for four vertices, and none of the edges of this graph cross each other. So

Chromatic number = 2

Here, the chromatic number is less than 4, so this graph is a plane graph.

Example 3: In the following graph, we have to determine the chromatic number.

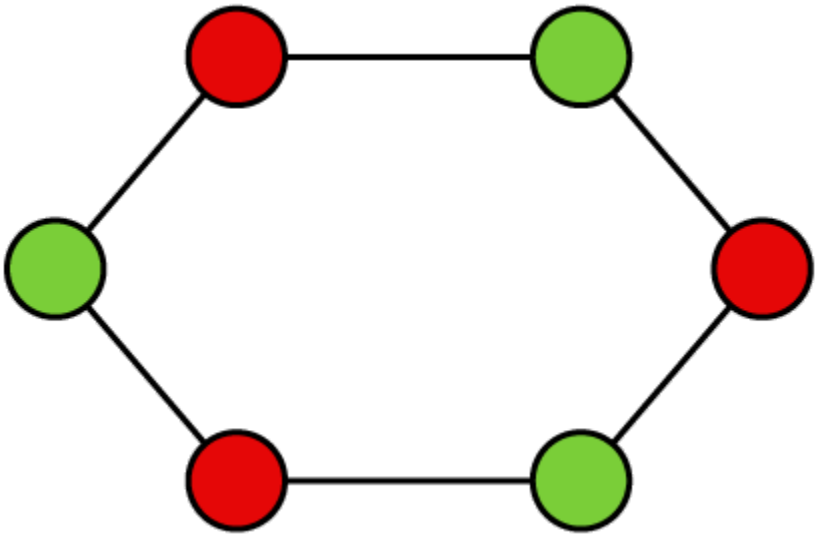


Solution: In the above graph, there are 5 different colors for five vertices, and none of the edges of this graph cross each other. So

Chromatic number = 5

Here, the chromatic number is greater than 4, so this graph is not a plane graph.

Example 4: In the following graph, we have to determine the chromatic number.



Solution: In the above graph, there are 2 different colors for six vertices, and none of the edges of this graph cross each other. So

Chromatic number = 2

Here, the chromatic number is less than 4, so this graph is a plane graph.

Complete Graph

A graph will be known as a complete graph if only one edge is used to join every two distinct vertices. Every vertex in a complete graph is connected with every other vertex. In this graph, every vertex will be colored with a different color. That means in the complete graph, two vertices do not contain the same color.

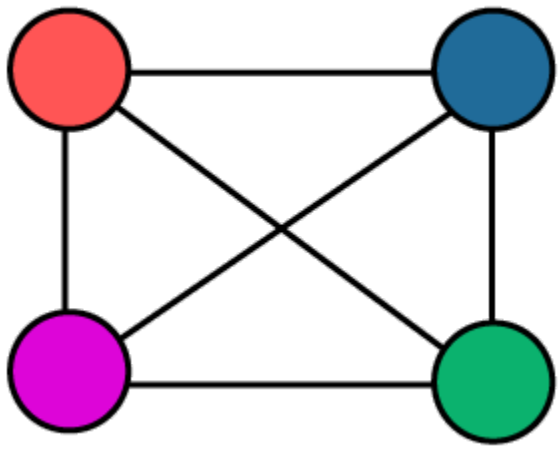
Chromatic Number

In a complete graph, the chromatic number will be equal to the number of vertices in that graph.

Examples of Complete graph:

There are various examples of complete graphs. Some of them are described as follows:

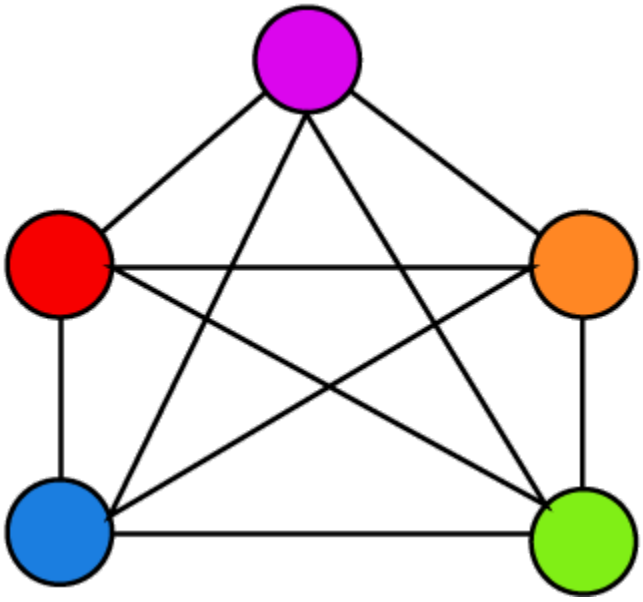
Example 1: In the following graph, we have to determine the chromatic number.



Solution: There are 4 different colors for 4 different vertices, and none of the colors are the same in the above graph. According to the definition, a chromatic number is the number of vertices. So,

Chromatic number = 4

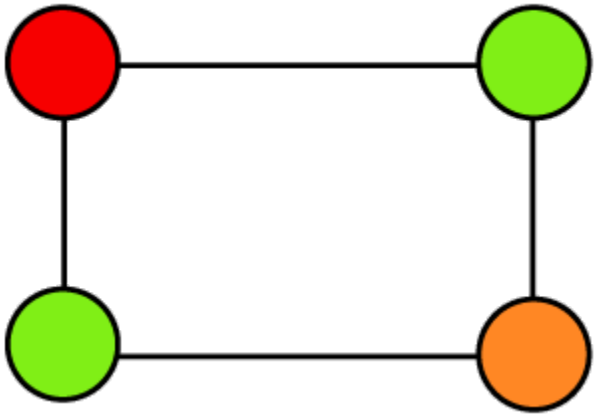
Example 2: In the following graph, we have to determine the chromatic number.



Solution: There are 5 different colors for 5 different vertices, and none of the colors are the same in the above graph. According to the definition, a chromatic number is the number of vertices. So,

Chromatic number = 5

Example 3: In the following graph, we have to determine the chromatic number.



Solution: There are 3 different colors for 4 different vertices, and one color is repeated in two vertices in the above graph. So this graph is not a complete graph and does not contain a chromatic number.

Bipartite Graph

A graph will be known as a bipartite graph if it contains two sets of vertices, A and B. The vertex of A can only join with the vertices of B. That means the edges cannot join the vertices with a set.

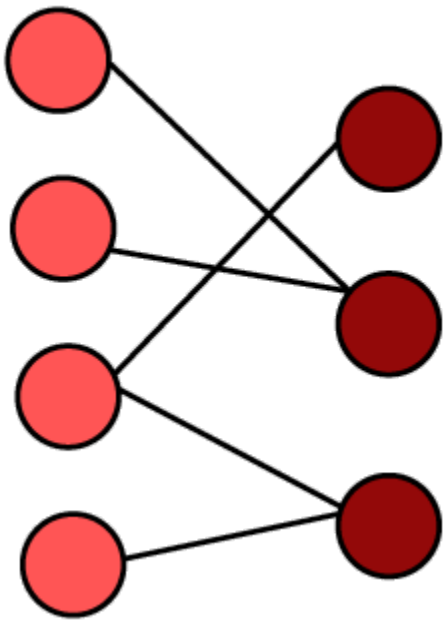
Chromatic Number

In any bipartite graph, the chromatic number is always equal to 2.

Examples of Bipartite graph:

There are various examples of bipartite graphs. Some of them are described as follows:

Example 1: In the following graph, we have to determine the chromatic number.



Solution: There are 2 different sets of vertices in the above graph. So the chromatic number of all bipartite graphs will always be 2. So

Chromatic number = 2

Tree:

A connected graph will be known as a tree if there are no circuits in that graph. In a tree, the chromatic number will equal to 2 no matter how many vertices are in the tree. Every bipartite graph is also a tree.

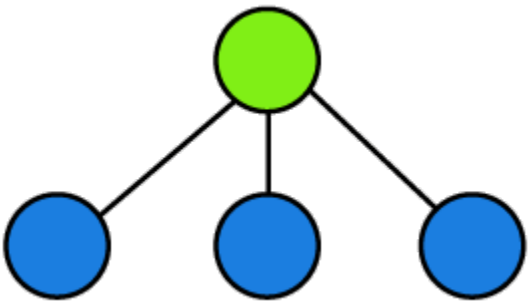
Chromatic Number

In any tree, the chromatic number is equal to 2.

Examples of Tree:

There are various examples of a tree. Some of them are described as follows:

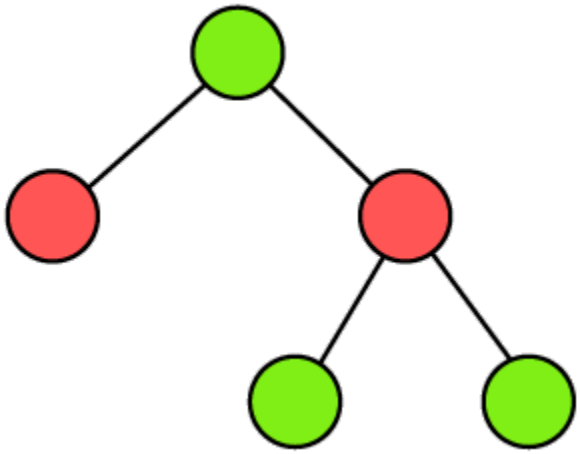
Example 1: In the following tree, we have to determine the chromatic number.



Solution: There are 2 different colors for four vertices. A tree with any number of vertices must contain the chromatic number as 2 in the above tree. So,

Chromatic number = 2

Example 2: In the following tree, we have to determine the chromatic number.



Solution: There are 2 different colors for five vertices. A tree with any number of vertices must contain the chromatic number as 2 in the above tree. So,

Chromatic number = 2

Difference between Minimum Spanning Tree and Shortest Path

Basic Parameters	Minimum Spanning Tree	Shortest Path
Definition	A tree that spans all vertices, while minimizing total weight.	The path with the lowest accumulated weight or distance.
Main Objective	Connecting all nodes together with minimal total weight.	It finds the most efficient route between the specific routes.
Key Property	It contains no cycles or loops.	Directed or undirected edges
Algorithm Examples	Some of the examples are Kruskal's and Prim's.	Some examples are Dijkstra's and Bellman-Ford's.
Advantages	It is used for wide purposes like the network design for laying cables and also merges the data in unsupervised learning.	The connection between the two points can be chosen very efficiently by the shortest path and it also helps the people in navigating the end location with minimal distance.
Disadvantages	When the network design is large, it becomes too tedious.	It finds the shortest path, but it may not be a unique one.

Differences between BFS and DFS

The following are the differences between the BFS and DFS:

	BFS	DFS
Full form	BFS stands for Breadth First Search.	DFS stands for Depth First Search.
Technique	It a vertex-based technique to find the shortest path in a graph.	It is an edge-based technique because the vertices along the edge are explored first from the starting to the end node.
Definition	BFS is a traversal technique in which all the nodes of the same level are explored first, and then we move to the next level.	DFS is also a traversal technique in which traversal is started from the root node and explore the nodes as far as possible until we reach the node that has no unvisited adjacent nodes.
Data Structure	Queue data structure is used for the BFS traversal.	Stack data structure is used for the BFS traversal.
Backtracking	BFS does not use the backtracking concept.	DFS uses backtracking to traverse all the unvisited nodes.

Number of edges	BFS finds the shortest path having a minimum number of edges to traverse from the source to the destination vertex.	In DFS, a greater number of edges are required to traverse from the source vertex to the destination vertex.
Optimality	BFS traversal is optimal for those vertices which are to be searched closer to the source vertex.	DFS traversal is optimal for those graphs in which solutions are away from the source vertex.
Speed	BFS is slower than DFS.	DFS is faster than BFS.
Suitability for decision tree	It is not suitable for the decision tree because it requires exploring all the neighboring nodes first.	It is suitable for the decision tree. Based on the decision, it explores all the paths. When the goal is found, it stops its traversal.
Memory efficient	It is not memory efficient as it requires more memory than DFS.	It is memory efficient as it requires less memory than BFS.

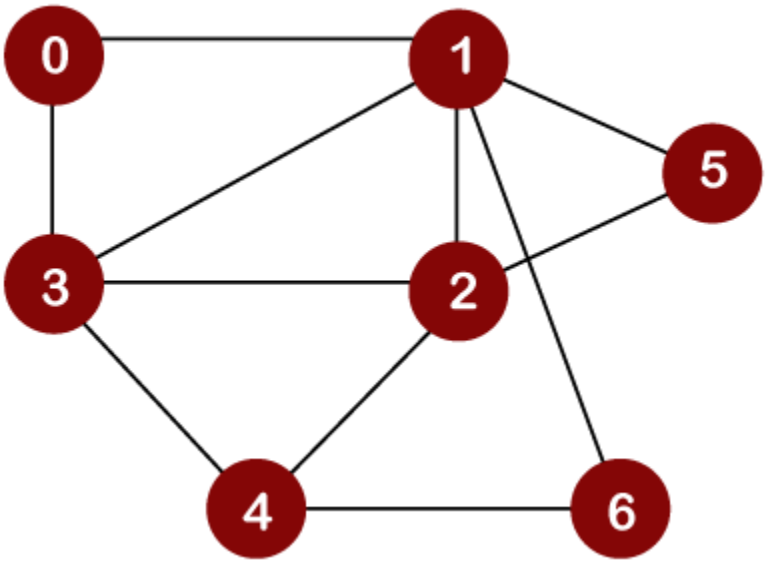
BFS vs. DFS

Before looking at the differences between BFS and DFS, we first should know about BFS and DFS separately.

What is BFS?

BFS stands for *Breadth First Search*. It is also known as level order traversal. The Queue data structure is used for the Breadth First Search traversal. When we use the BFS algorithm for the traversal in a graph, we can consider any node as a root node.

Let's consider the below graph for the breadth first search traversal.



Suppose we consider node 0 as a root node. Therefore, the traversing would be started from node 0.

Once node 0 is removed from the Queue, it gets printed and marked as a *visited node*.

Once node 0 gets removed from the Queue, then the adjacent nodes of node 0 would be inserted in a Queue as shown below:

Result : 0

Now the node 1 will be removed from the Queue; it gets printed and marked as a visited node

Once node 1 gets removed from the Queue, then all the adjacent nodes of a node 1 will be added in a Queue. The adjacent nodes of node 1 are 0, 3, 2, 6, and 5. But we have to insert only unvisited nodes in a Queue. Since nodes 3, 2, 6, and 5 are unvisited; therefore, these nodes will be added in a Queue as shown below:

Result : 0 , 1

The next node is 3 in a Queue. So, node 3 will be removed from the Queue, it gets printed and marked as visited as shown below:

Result : 0, 1, 3

Once node 3 gets removed from the Queue, then all the adjacent nodes of node 3 except the visited nodes will be added in a Queue. The adjacent nodes of node 3 are 0, 1, 2, and 4. Since nodes 0, 1 are already visited, and node 2 is present in a Queue; therefore, we need to insert only node 4 in a Queue.

Result : 0, 1, 3

Now, the next node in the Queue is 2. So, 2 would be deleted from the Queue. It gets printed and marked as visited as shown below:

Result : 0, 1, 3, 2,

Once node 2 gets removed from the Queue, then all the adjacent nodes of node 2 except the visited nodes will be added in a Queue. The adjacent nodes of node 2 are 1, 3, 5, 6, and 4. Since the nodes 1 and 3 have already been visited, and 4, 5, 6 are already added in the Queue; therefore, we do not need to insert any node in the Queue.

The next element is 5. So, 5 would be deleted from the Queue. It gets printed and marked as visited as shown below:

Result : 0, 1, 3, 2, 5

Once node 5 gets removed from the Queue, then all the adjacent nodes of node 5 except the visited nodes will be added in the Queue. The adjacent nodes of node 5 are 1 and 2. Since both the nodes have already been visited; therefore, there is no vertex to be inserted in a Queue.

The next node is 6. So, 6 would be deleted from the Queue. It gets printed and marked as visited as shown below:

Result : 0, 1, 3, 2, 5, 6

Once the node 6 gets removed from the Queue, then all the adjacent nodes of node 6 except the visited nodes will be added in the Queue. The adjacent nodes of node 6 are 1 and 4. Since the node 1 has already been visited and node 4 is already added in the Queue; therefore, there is not vertex to be inserted in the Queue.

The next element in the Queue is 4. So, 4 would be deleted from the Queue. It gets printed and marked as visited.

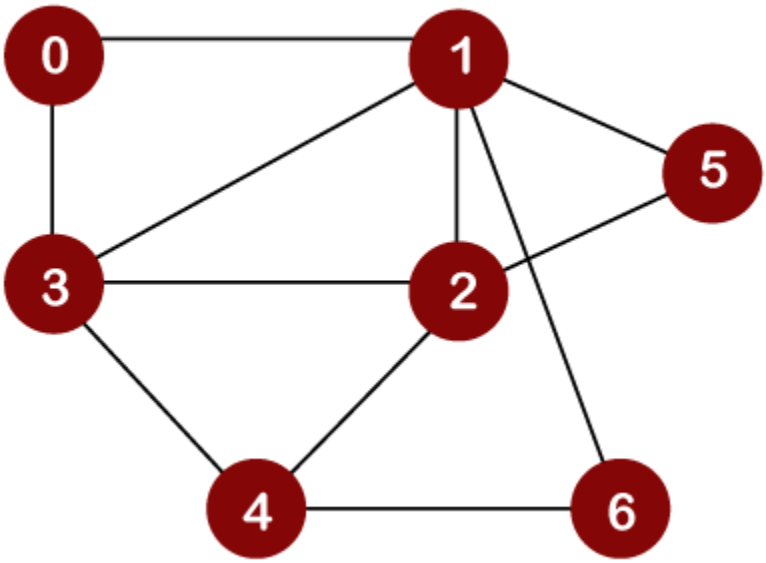
Once the node 4 gets removed from the Queue, then all the adjacent nodes of node 4 except the visited nodes will be added in the Queue. The adjacent nodes of node 4 are 3, 2, and 6. Since all the adjacent nodes have already been visited; so, there is no vertex to be inserted in the Queue.

What is DFS?

DFS stands for Depth First Search. In DFS traversal, the stack data structure is used, which works on the LIFO (Last In First Out) principle. In DFS, traversing can be started from any node, or we can say that any node can be considered as a root node until the root node is not mentioned in the problem.

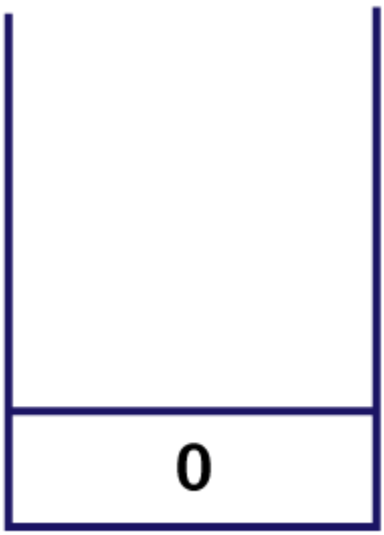
In the case of BFS, the element which is deleted from the Queue, the adjacent nodes of the deleted node are added to the Queue. In contrast, in DFS, the element which is removed from the stack, then only one adjacent node of a deleted node is added in the stack.

Let's consider the below graph for the Depth First Search traversal.

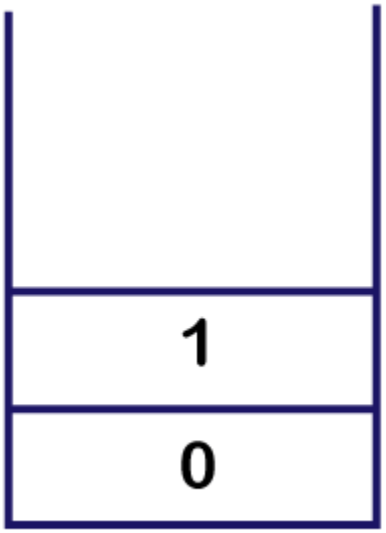


Consider node 0 as a root node.

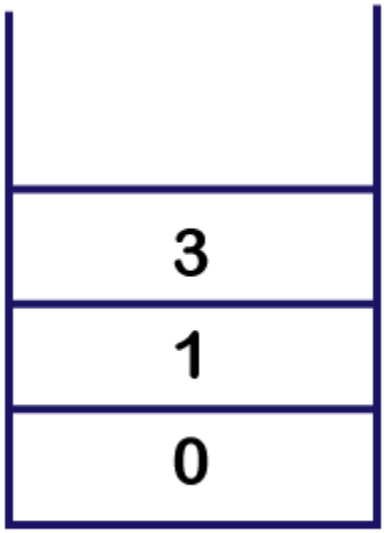
First, we insert the element 0 in the stack as shown below:



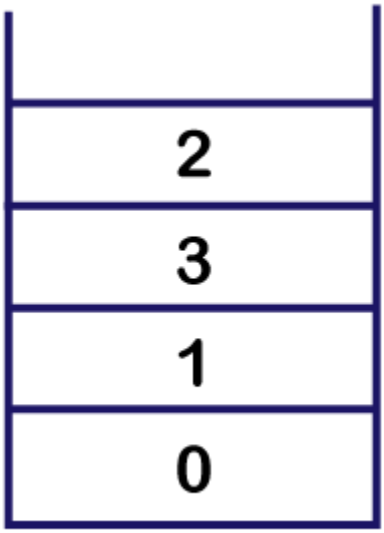
The node 0 has two adjacent nodes, i.e., 1 and 3. Now we can take only one adjacent node, either 1 or 3, for traversing. Suppose we consider node 1; therefore, 1 is inserted in a stack and gets printed as shown below:



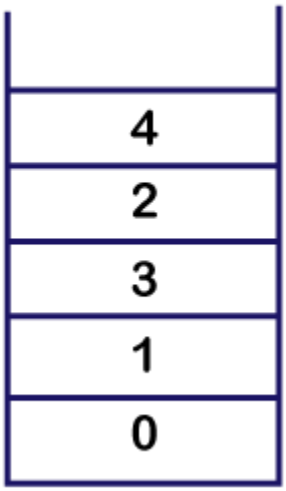
Now we will look at the adjacent vertices of node 1. The unvisited adjacent vertices of node 1 are 3, 2, 5 and 6. We can consider any of these four vertices. Suppose we take node 3 and insert it in the stack as shown below:



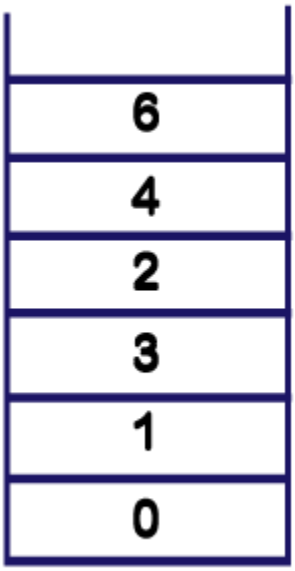
Consider the unvisited adjacent vertices of node 3. The unvisited adjacent vertices of node 3 are 2 and 4. We can take either of the vertices, i.e., 2 or 4. Suppose we take vertex 2 and insert it in the stack as shown below:



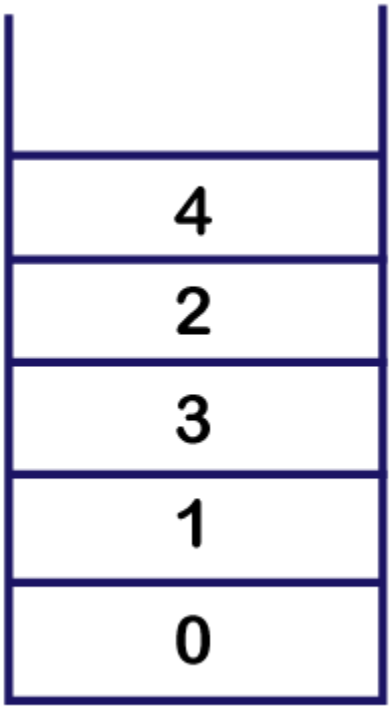
The unvisited adjacent vertices of node 2 are 5 and 4. We can choose either of the vertices, i.e., 5 or 4. Suppose we take vertex 4 and insert in the stack as shown below:

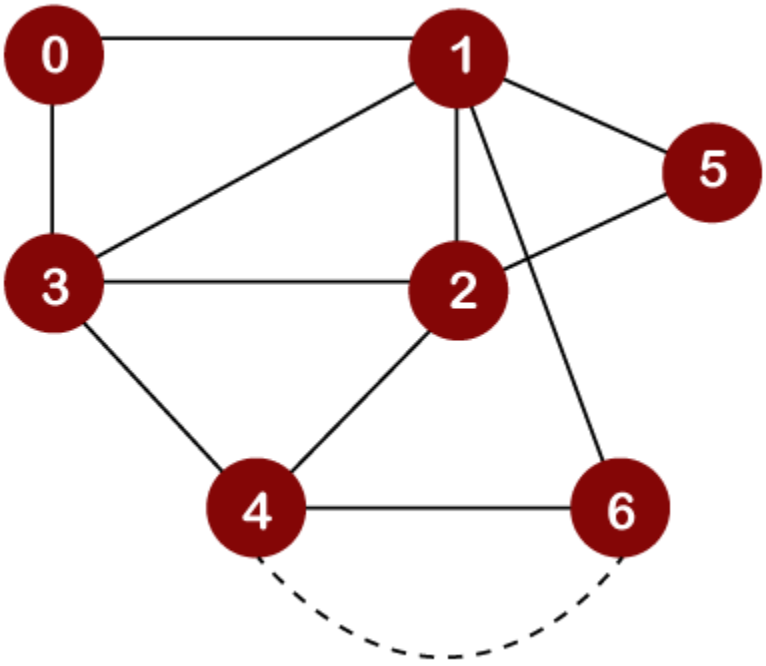


Now we will consider the unvisited adjacent vertices of node 4. The unvisited adjacent vertex of node 4 is node 6. Therefore, element 6 is inserted into the stack as shown below:

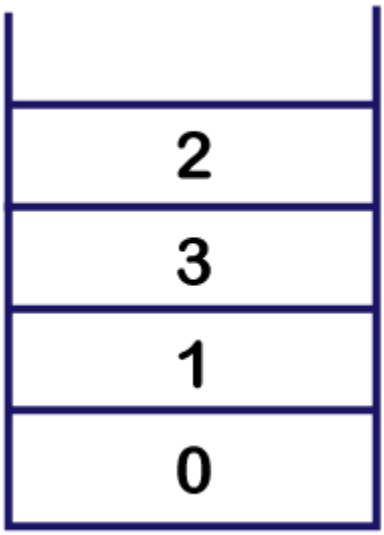


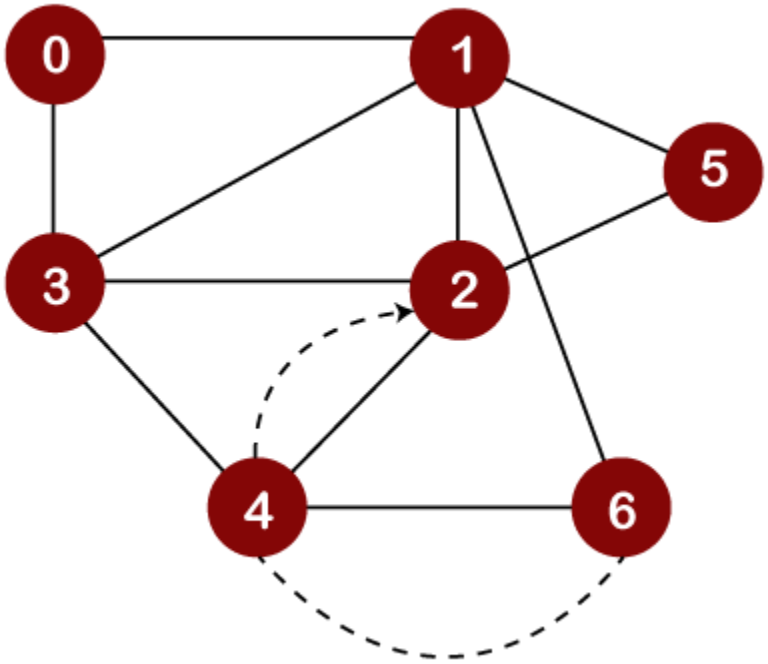
After inserting element 6 in the stack, we will look at the unvisited adjacent vertices of node 6. As there is no unvisited adjacent vertices of node 6, so we cannot move beyond node 6. In this case, we will perform backtracking. The topmost element, i.e., 6 would be popped out from the stack as shown below:



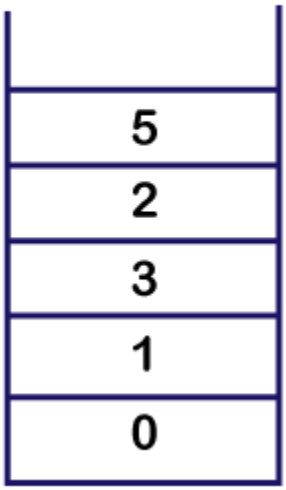


The topmost element in the stack is 4. Since there are no unvisited adjacent vertices left of node 4; therefore, node 4 is popped out from the stack as shown below:

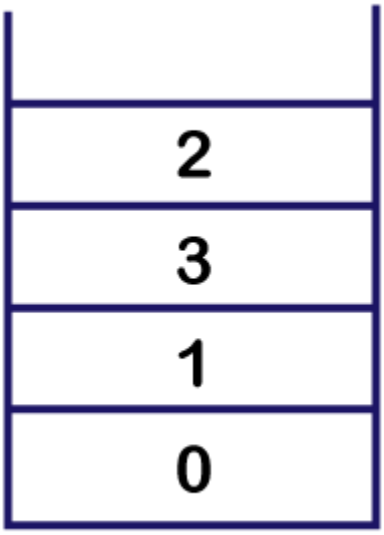




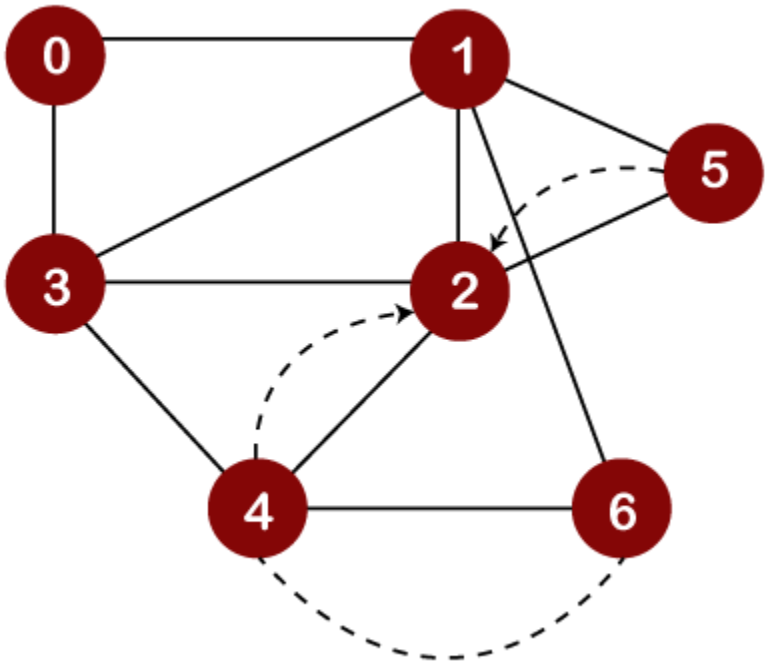
The next topmost element in the stack is 2. Now, we will look at the unvisited adjacent vertices of node 2. Since only one unvisited node, i.e., 5 is left, so node 5 would be pushed into the stack above 2 and gets printed as shown below:



Now we will check the adjacent vertices of node 5, which are still unvisited. Since there is no vertex left to be visited, so we pop the element 5 from the stack as shown below:

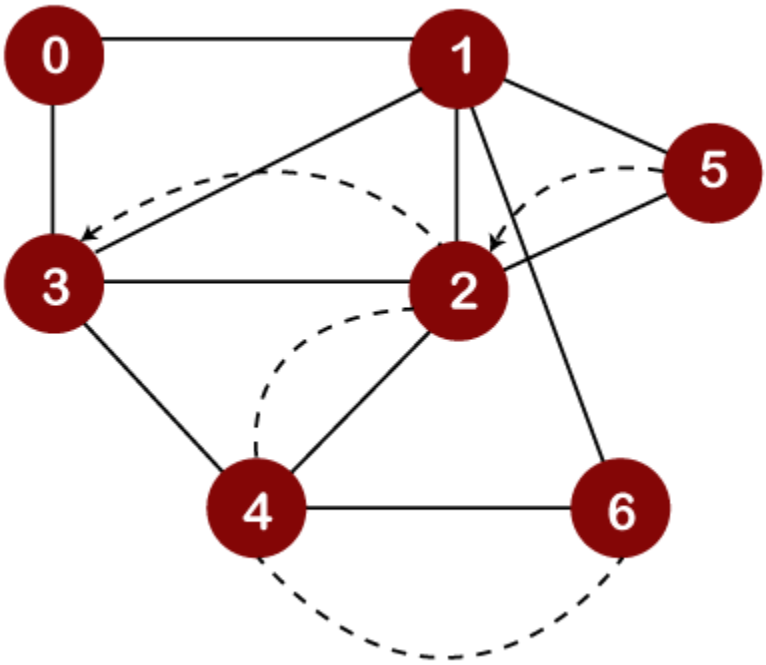


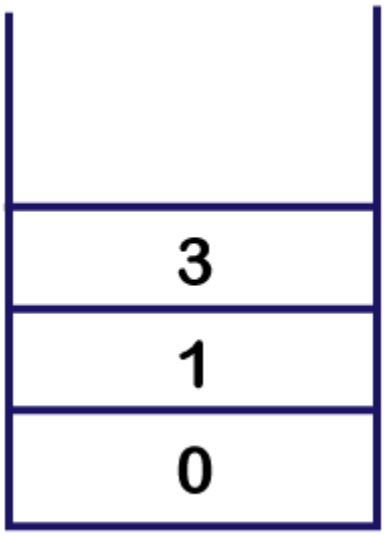
We cannot move further 5, so we need to perform backtracking. In backtracking, the topmost element would be popped out from the stack. The topmost element is 5 that would be popped out from the stack, and we move back to node 2 as shown below:



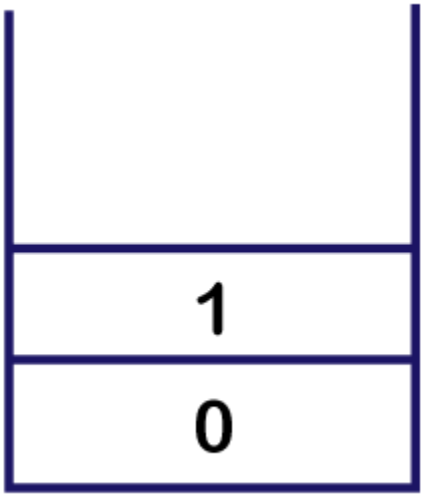
Now we will check the unvisited adjacent vertices of node 2. As there is no adjacent vertex left to be visited, so we perform backtracking. In backtracking,

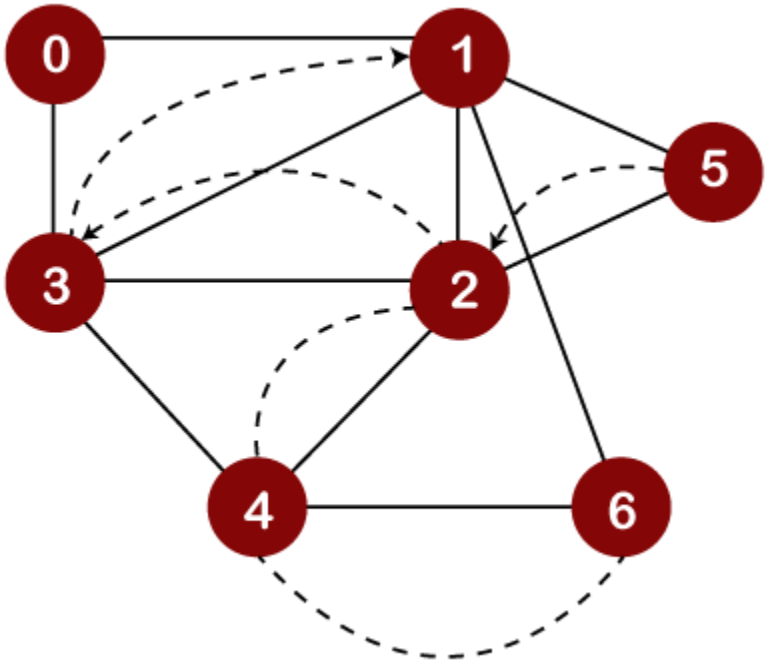
the topmost element, i.e., 2 would be popped out from the stack, and we move back to the node 3 as shown below:



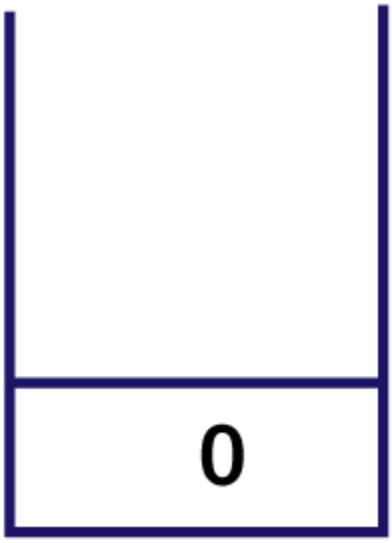


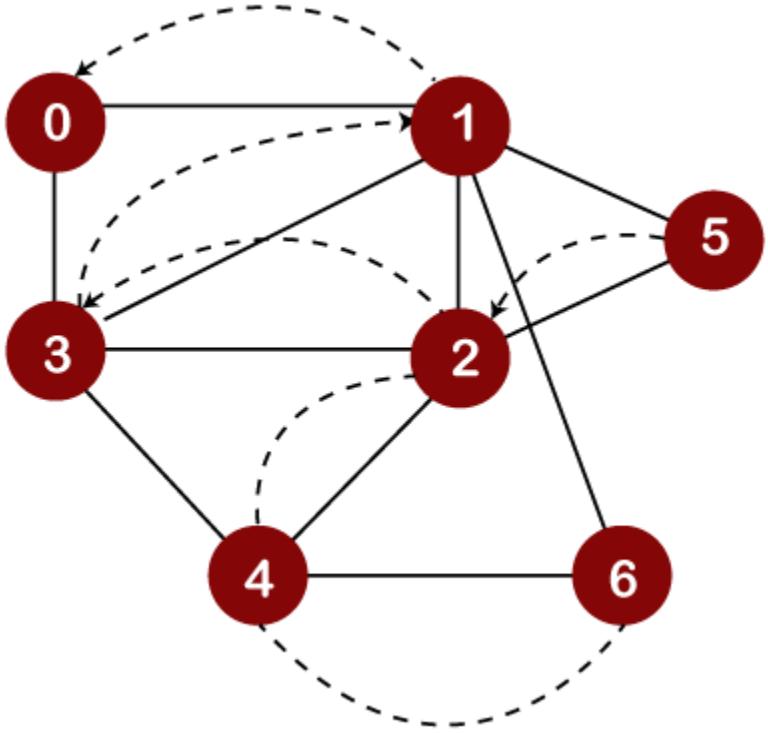
Now we will check the unvisited adjacent vertices of node 3. As there is no adjacent vertex left to be visited, so we perform backtracking. In backtracking, the topmost element, i.e., 3 would be popped out from the stack and we move back to node 1 as shown below:





After popping out element 3, we will check the unvisited adjacent vertices of node 1. Since there is no vertex left to be visited; therefore, the backtracking will be performed. In backtracking, the topmost element, i.e., 1 would be popped out from the stack, and we move back to node 0 as shown below:





We will check the adjacent vertices of node 0, which are still unvisited. As there is no adjacent vertex left to be visited, so we perform backtracking. In this, only one element, i.e., 0 left in the stack, would be popped out from the stack as shown below:

As we can observe in the above figure that the stack is empty. So, we have to stop the DFS traversal here, and the elements which are printed is the result of the DFS traversal.

Time Complexity of Kruskals Algorithm

We shall talk about Kruskal's algorithm in this post. We shall also examine the Kruskal's algorithm's difficulty, functionality, example, and implementation here.

However, we need first comprehend the fundamental concepts, such as spanning tree and least spanning tree, before going on to the technique.

An undirected edge-weighted graph's smallest spanning forest is found via Kruskal's method. Finding a minimal spanning tree is done if the graph is linked. (A minimal spanning tree is a collection of edges that forms a tree with every vertex in a linked graph where the weighted average of all the edges in the tree is minimized. A minimal spanning forest for a disconnected graph is made up of a minimum spanning tree for each linked component.) In graph theory, it is known as a greedy method since it continuously adds the next lowest-weight edge that won't cycle to the minimal spanning forest.

An undirected connected graph's spanning tree is a sub graph of that graph.

Minimum Spanning Tree - A minimum spanning tree is one in which the total edge weights are at their lowest value. The weight of the spanning tree is the total of the weights assigned to its edges.

Let's get to the major point now.

For a linked weighted graph, the least spanning tree is determined using Kruskal's Algorithm. The algorithm's primary goal is to identify the subset of edges that will allow us to pass through each graph vertex. Instead of concentrating on a global optimum, it adopts a greedy strategy that seeks the best outcome at each step.

How is the Kruskal algorithm implemented?

With Kruskal's method, we begin with the edges that have the lowest weight and keep adding edges until we get the desired result. The following are the steps to apply Kruskal's algorithm:

- Sort all of the edges first from low weight to high weight.
- Add the edge with the lightest weight to the spanning tree at this point. Reject the edge if it would otherwise result in a cycle.
- Once we have included all of the edges, a minimal spanning tree will be produced.

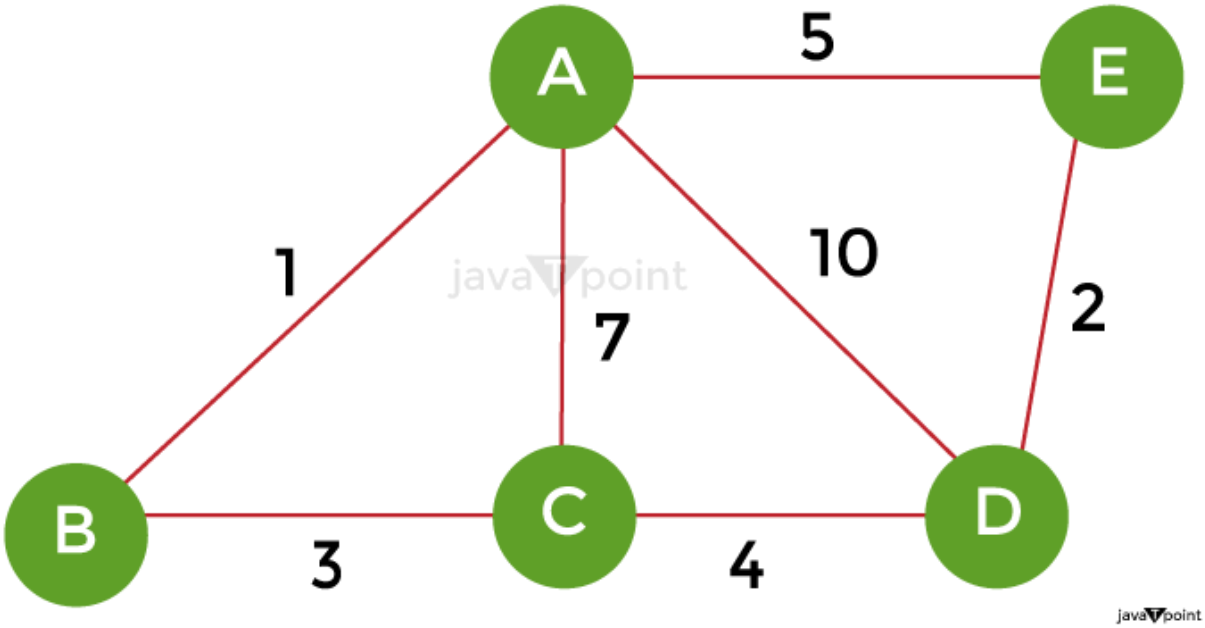
The uses of Kruskal's algorithm include:

- Wiring between cities may be laid out using Kruskal's method.
- It is possible to utilize it to install LAN connections.

Kruskal's Algorithm Example

Let's now use an example to demonstrate how Kruskal's algorithm functions. An illustration will make it simpler to comprehend Kruskal's algorithm.

Let's say that a weighted graph is



The table below provides the edge weights for the graph above.

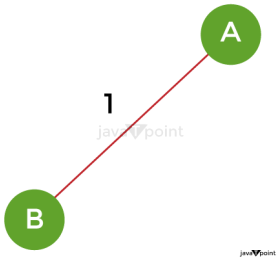
Edge	AB	AC	AD	AE	BC	CD	DE
Weight	1	7	10	5	3	4	2

Sort the edges listed above now according to ascending weights.

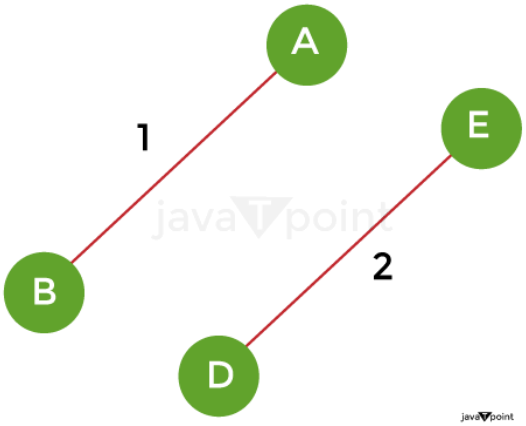
Edge	AB	DE	BC	CD	AE	AC	AD
Weight	1	2	3	4	5	7	10

Let's start building the least spanning tree right now.

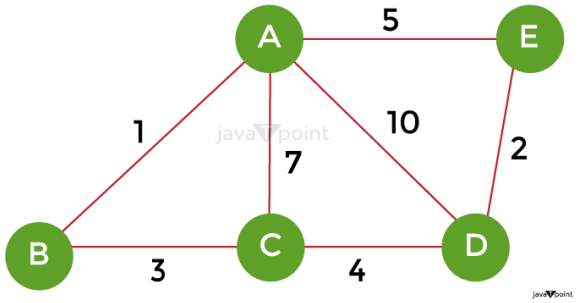
Step 1: Add the edge AB with weight 1 to the MST in step 1 first.



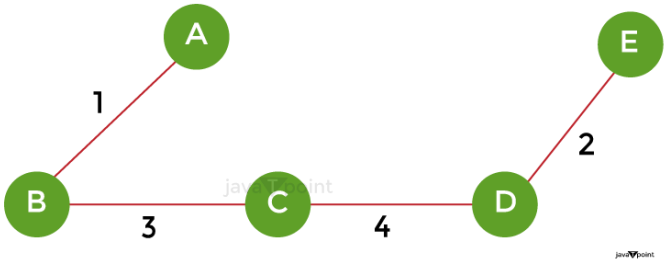
Step 2: Since the edge DE with weight 2 isn't producing the cycle, add it to the MST.



Step 3: Add the edge BC with weight 3 to the MST in step 3 because it does not produce a cycle or loop.



Step 4: Choose the edge CD with weight 4 to the MST at this point since the cycle is not being formed.

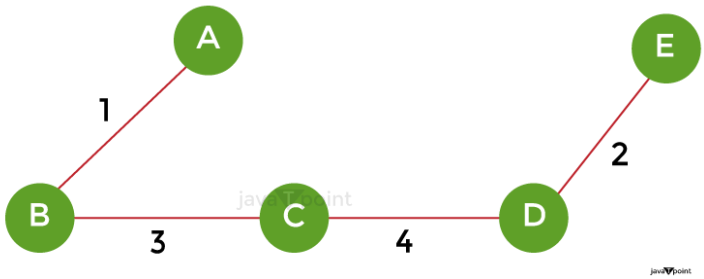


Step 5 - Next, choose weight 5 on the edge AE. This edge must be eliminated since including it would start the cycle.

Step 6: Pick the edge of the AC using weight in step 6." This edge must be eliminated since including it would start the cycle.

Step 7: Pick the edge AD with weight 10 in step 7. Throwing away this edge will prevent the cycle from starting.

Step 8: As a result, the final minimal spanning tree created using Kruskal's approach from the provided weighted graph is -



The cost of the MST is = AB + DE + BC + CD = 1 + 2 + 3 + 4 = 10.

In the above tree, the number of edges now matches the number of vertices minus 1. The algorithm so ends here.

Step 1: Create a forest F in such a way that every vertex of the graph is a separate tree.
Step 2: Create a set E that contains all the edges of the graph.
Step 3: Repeat Steps 4 and 5 while E is NOT EMPTY and F is not spanning
Step 4: Remove an edge from E with minimum weight
Step 5: IF the edge obtained in Step 4 connects two different trees, then add it to the forest F
(for combining two trees into one tree).
ELSE
Discard the edge
Step 6: END

Time Complexity of Kruskal's Algorithm

The Kruskal method has an O(E logE) or O(V logV) time complexity, where E is the number of edges and V is the number of vertices.

A connected, undirected graph with all of its vertices is described as a spanning tree, which is a tree-like sub graph of the graph. Or, to put it in Layman's terms, it is a subset of the graph's edges that together form an acyclic tree, of which the graph's nodes are all members.

With the additional restriction of having the lowest weights among all conceivable spanning trees, the minimal spanning tree has all the characteristics of a spanning tree. Similar to spanning trees, a graph may have several potential MSTs.

Properties of a Spanning Tree

The following principles are held by a spanning tree:

- The graph and spanning tree both have the same number of vertices (V).
- The spanning tree has a fixed number of edges, which is one fewer than the total number of vertices (E = V-1).
- There should only be one source of component, not more, so that the spanning tree is not disconnected.
- There should be no cycles in the spanning tree, which means it should be acyclic.
- The sum of the edge weights of all the spanning tree's edges is referred to as the overall cost (or weight) of the spanning tree.
- There are several spanning trees that might be used for a graph.