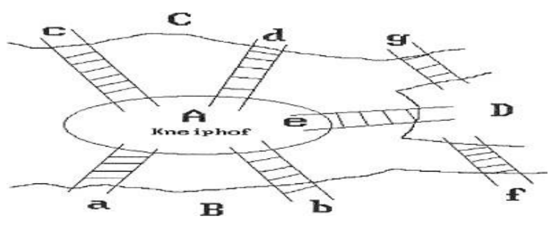
GRAPH THEORY

Introduction to Graphs

Graph is a non linear data structure; A map is a well-known example of a graph. In a map various connections are made between the cities. The cities are connected via roads, railway lines and aerial network. We can assume that the graph is the interconnection of cities by roads. Euler used graph theory to solve Seven Bridges of Königsberg problem. Is there a possible way to traverse every bridge exactly once – Euler Tour



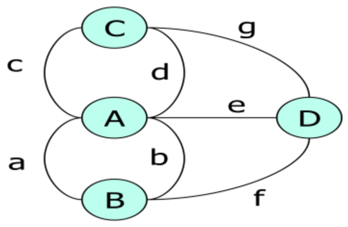


Figure: Section of the river Pregal in Koenigsberg and Euler's graph.

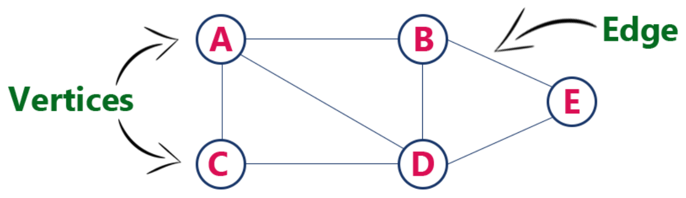
Defining the degree of a vertex to be the number of edges incident to it, Euler showed that there is a walk starting at any vertex, going through each edge exactly once and terminating at the start vertex iff the degree of each, vertex is even. A walk which does this is called Eulerian. There is no Eulerian walk for the Koenigsberg bridge problem as all four vertices are of odd degree.

A graph contains a set of points known as nodes (or vertices) and set of links known as edges (or Arcs) which connects the vertices.

A graph is defined as Graph is a collection of vertices and arcs which connects vertices in the graph. A graph G is represented as G = (V , E), where V is set of vertices and E is set of edges.

Example: graph G can be defined as G = (V , E) Where V = {A,B,C,D,E} and

E = {(A,B),(A,C)(A,D),(B,D),(C,D),(B,E),(E,D)}. This is a graph with 5 vertices and 6 edges.



Graph Terminology

1.**Vertex** : An individual data element of a graph is called as Vertex. Vertex is also known as node. In above example graph, A, B, C, D & E are known as vertices.

2.**Edge** : An edge is a connecting link between two vertices. Edge is also known as Arc. An edge is represented as (starting Vertex, ending Vertex).

In above graph, the link between vertices A and B is represented as (A,B).

Edges are three types:

1.Undirected Edge - An undirected edge is a bidirectional edge. If there is an undirected edge between vertices A and B then edge (A , B) is equal to edge (B , A).

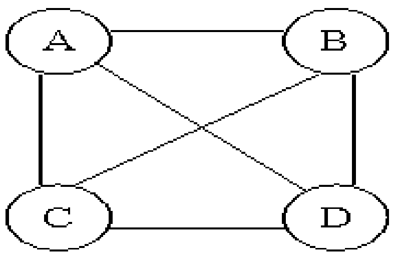
2.Directed Edge - A directed edge is a unidirectional edge. If there is a directed edge between vertices A and B then edge (A , B) is not equal to edge (B , A).

3. Weighted Edge - A weighted edge is an edge with cost on it.

Types of Graphs

1.Undirected Graph

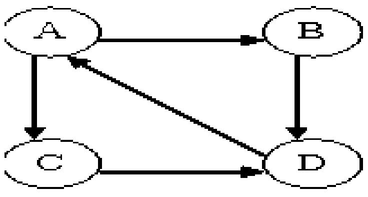
A graph with only undirected edges is said to be undirected graph.



Undirected Graph.

2.Directed Graph

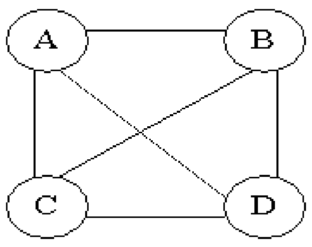
A graph with only directed edges is said to be directed graph.



Directed Graph.

3.Complete Graph

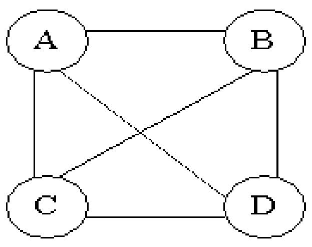
A graph in which any V node is adjacent to all other nodes present in the graph is known as a complete graph. An undirected graph contains the edges that are equal to edges = n(n-1)/2 where n is the number of vertices present in the graph. The following figure shows a complete graph.



A complete graph.

4.Regular Graph

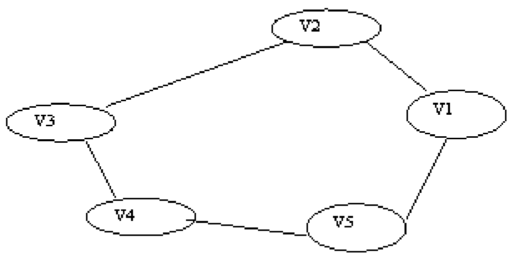
Regular graph is the graph in which nodes are adjacent to each other, i.e., each node is accessible from any other node.



A regular graph

5.Cycle Graph

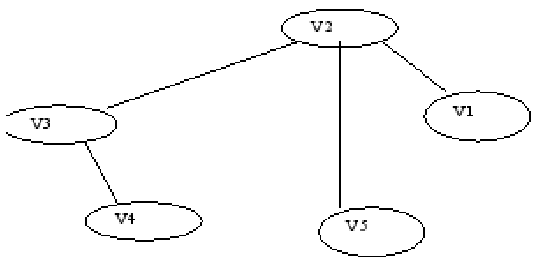
A graph having cycle is called cycle graph. In this case the first and last nodes are the same. A closed simple path is a cycle.



A cycle graph

6. Acyclic Graph

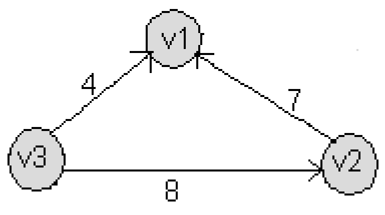
A graph without cycle is called acyclic graphs.



A acyclic graph

7. Weighted Graph

A graph is said to be weighted if there are some non negative value assigned to each edges of the graph. The value is equal to the length between two vertices. Weighted graph is also called a network.



A weighted graph

Outgoing Edge

A directed edge is said to be outgoing edge on its orign vertex.

Incoming Edge

A directed edge is said to be incoming edge on its destination vertex.

Degree

Total number of edges connected to a vertex is said to be degree of that vertex.

Indegree

Total number of incoming edges connected to a vertex is said to be indegree of that vertex.

Outdegree

Total number of outgoing edges connected to a vertex is said to be outdegree of that vertex.

Parallel edges or Multiple edges

If there are two undirected edges to have the same end vertices, and for two directed edges to have the same origin and the same destination. Such edges are called parallel edges or multiple edges.

Self-loop

An edge (undirected or directed) is a self-loop if its two endpoints coincide.

Simple Graph

A graph is said to be simple if there are no parallel and self-loop edges.

Adjacent nodes

When there is an edge from one node to another then these nodes are called adjacent nodes.

Incidence

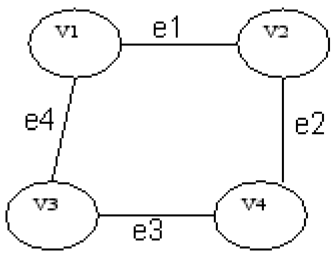
In an undirected graph the edge between v1 and v2 is incident on node v1 and v2.

Walk

A walk is defined as a finite alternating sequence of vertices and edges, beginning and ending with vertices, such that each edge is incident with the vertices preceding and following it.

Closed walk

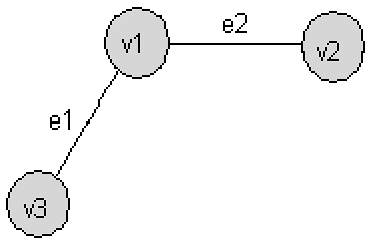
A walk which is to begin and end at the same vertex is called close walk. Otherwise it is an open walk.



If e1,e2,e3,and e4 be the edges of pair of vertices (v1,v2),(v2,v4),(v4,v3) and (v3,v1) respectively ,then v1 e1 v2 e2 v4 e3 v3 e4 v1 be its closed walk or circuit.

Path

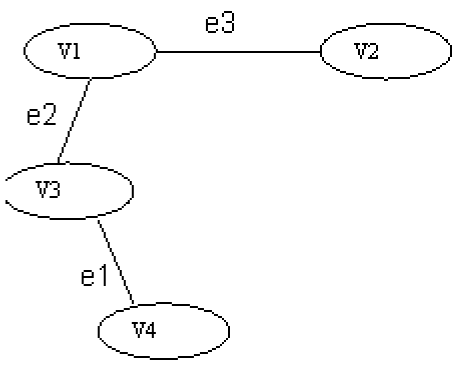
A open walk in which no vertex appears more than once is called a path.



If e1 and e2 be the two edges between the pair of vertices (v1,v3) and (v1,v2) respectively, then v3 e1 v1 e2 v2 be its path.

Length of a path

The number of edges in a path is called the length of that path. In the following, the length of the path is 3.

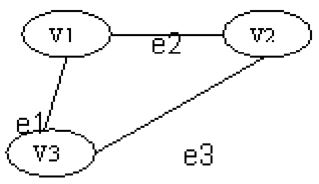


An open walk Graph

Circuit

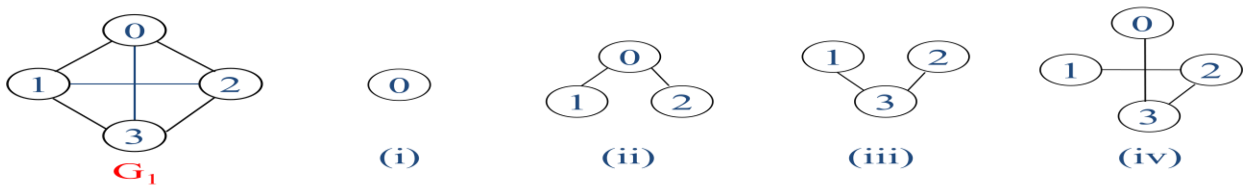
A closed walk in which no vertex (except the initial and the final vertex) appears more than once is called a circuit.

A circuit having three vertices and three edges.



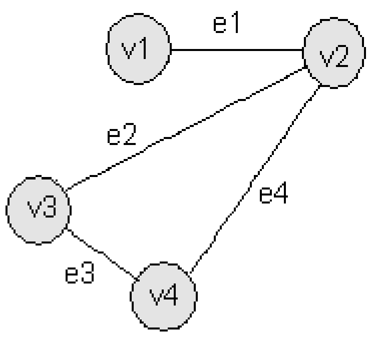
Sub Graph

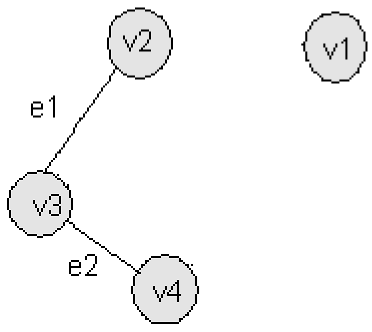
A graph S is said to be a sub graph of a graph G if all the vertices and all the edges of S are in G, and each edge of S has the same end vertices in S as in G. A subgraph of G is a graph G' such that V(G') ⊆ V(G) and E(G') ⊆ E(G)



Connected Graph

A graph G is said to be connected if there is at least one path between every pair of vertices in G. Otherwise,G is disconnected.





A connected graph G A disconnected graph G

This graph is disconnected because the vertex v1 is not connected with the other vertices of the graph.

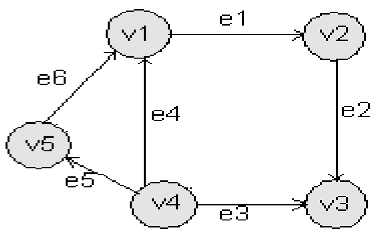
Degree

In an undirected graph, the number of edges connected to a node is called the degree of that node or the degree of a node is the number of edges incident on it.

In the above graph, degree of vertex v1 is 1, degree of vertex v2 is 3, degree of v3 and v4 is 2 in a connected graph.

Indegree

The indegree of a node is the number of edges connecting to that node or in other words edges incident to it.



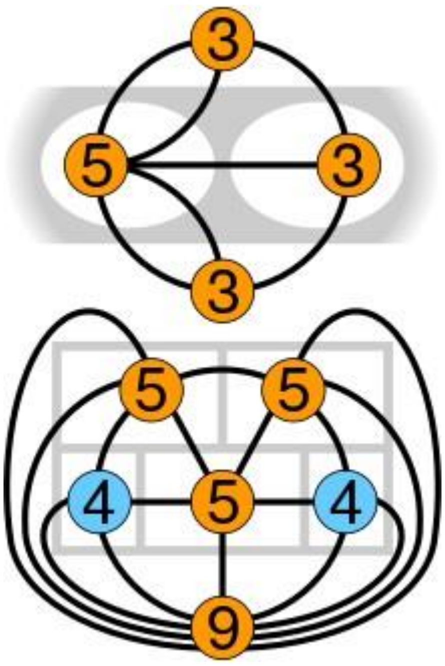
In the above graph,the indegree of vertices v1, v3 is 2, indegree of vertices v2, v5 is 1 and indegree of v4 is zero.

Outdegree

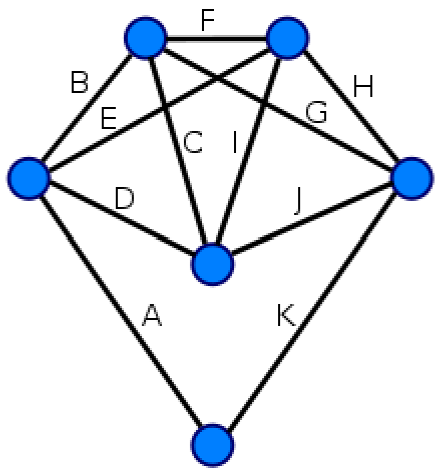
The outdegree of a node (or vertex) is the number of edges going outside from that node or in other words the

Eulerian path

In graph theory, an **Eulerian trail** (or **Eulerian path**) is a trail in a finite graph that visits every edge exactly once (allowing for revisiting vertices). Similarly, an **Eulerian circuit** or **Eulerian cycle** is an Eulerian trail that starts and ends on the same vertex. They were first discussed by Leonhard Euler while solving the famous Seven Bridges of Königsberg problem in 1736. The problem can be stated mathematically like this:



Multigraphs of both Königsberg Bridges and Five room puzzles have more than two odd vertices (in orange), thus are not Eulerian and hence the puzzles have no solutions.



Every vertex of this graph has an even degree. Therefore, this is an Eulerian graph. Following the edges in alphabetical order gives an Eulerian circuit/cycle.

> Given the graph in the image, is it possible to construct a path (or a cycle; i.e., a path starting and ending on the same vertex) that visits each edge exactly once?

Euler proved that a necessary condition for the existence of Eulerian circuits is that all vertices in the graph have an even degree, and stated without proof that connected graphs with all vertices of even degree have an Eulerian circuit. The first complete proof of this latter claim was published posthumously in 1873 by Carl Hierholzer. This is known as **Euler's Theorem:**

> A connected graph has an Euler cycle if and only if every vertex has even degree.

The term **Eulerian graph** has two common meanings in graph theory. One meaning is a graph with an Eulerian circuit, and the other is a graph with every vertex of even degree. These definitions coincide for connected graphs.

For the existence of Eulerian trails it is necessary that zero or two vertices have an odd degree; this means the Königsberg graph is *not* Eulerian. If there are no vertices of odd degree, all Eulerian trails are circuits. If there are exactly two vertices of odd degree, all Eulerian trails start at one of them and end at the other. A graph that has an Eulerian trail but not an Eulerian circuit is called **semi-Eulerian**.

Definition

An **Eulerian trail**,or **Euler walk**, in an undirected graph is a walk that uses each edge exactly once. If such a walk exists, the graph is called **traversable** or **semi-eulerian**.[4]

An **Eulerian cycle**,[3] also called an **Eulerian circuit** or **Euler tour**, in an undirected graph is a cycle that uses each edge exactly once. If such a cycle exists, the graph is called **Eulerian** or **unicursal**.[5] The term "Eulerian graph" is also sometimes used in a weaker sense to denote a graph where every vertex has even degree. For finite connected graphs the two definitions are equivalent, while a possibly unconnected graph is Eulerian in the weaker sense if and only if each connected component has an Eulerian cycle.

For directed graphs, "path" has to be replaced with *directed path* and "cycle" with *directed cycle*.

The definition and properties of Eulerian trails, cycles and graphs are valid for multigraphs as well.

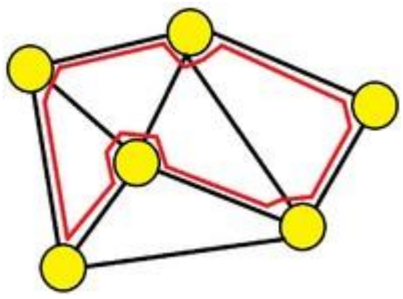
An **Eulerian orientation** of an undirected graph *G* is an assignment of a direction to each edge of *G* such that, at each vertex *v*, the indegree of *v* equals the outdegree of *v*. Such an orientation exists for any undirected graph in which every vertex has even degree, and may be found by constructing an Euler tour in each connected component of *G* and then orienting the edges according to the tour.[6] Every Eulerian orientation of a connected graph is a strong orientation, an orientation that makes the resulting directed graph strongly connected.

Properties

- An undirected graph has an Eulerian cycle if and only if every vertex has even degree, and all of its vertices with nonzero degree belong to a single connected component.
- An undirected graph can be decomposed into edge-disjoint cycles if and only if all of its vertices have even degree. So, a graph has an Eulerian cycle if and only if it can be decomposed into edge-disjoint cycles and its nonzero-degree vertices belong to a single connected component.
- An undirected graph has an Eulerian trail if and only if exactly zero or two vertices have odd degree, and all of its vertices with nonzero degree belong to a single connected component
- A directed graph has an Eulerian cycle if and only if every vertex has equal in degree and out degree, and all of its vertices with nonzero degree belong to a single strongly connected component. Equivalently, a directed graph has an Eulerian cycle if and only if it can be decomposed into edge-disjoint directed cycles and all of its vertices with nonzero degree belong to a single strongly connected component.
- A directed graph has an Eulerian trail if and only if at most one vertex has (out-degree) − (in- degree) = 1, at most one vertex has (in-degree) − (out-degree) = 1, every other vertex has equal in-degree and out-degree, and all of its vertices with nonzero degree belong to a single connected component of the underlying undirected graph.

Hamiltonian path

In the mathematical field of graph theory, a Hamiltonian path (or traceable path) is a path in an undirected or directed graph that visits each vertex exactly once. A Hamiltonian cycle (or Hamiltonian circuit) is a cycle that visits each vertex exactly once. A Hamiltonian path that starts and ends at adjacent vertices can be completed by adding one more edge to form a Hamiltonian cycle, and removing any edge from a Hamiltonian cycle produces a Hamiltonianpath. Determining whether such paths and cycles exist in graphs (the Hamiltonian path problem and Hamiltonian cycle problem) are NP-complete.



A Hamiltonian cycle around a network of six vertices

Hamiltonian paths and cycles are named after William Rowan Hamilton who invented the icosian game, now also known as *Hamilton's puzzle*, which involves finding a Hamiltonian cycle in the edge graph of the dodecahedron. Hamilton solved this problem using the icosiancalculus, an algebraic structure based on roots of unity with many similarities to the quaternions (also invented by Hamilton). This solution does not generalize to arbitrary graphs.

Despite being named after Hamilton, Hamiltonian cycles in polyhedra had also been studied ayear earlier by Thomas Kirkman, who, in particular, gave an example of a polyhedron without Hamiltonian cycles. Even earlier, Hamiltonian cycles and paths in the knight's graph of the chessboard, the knight's tour, had been studied in the 9th century in Indian mathematics by Rudrata, and around the same time in Islamic mathematics by al-Adli ar- Rumi. In 18th century Europe, knight's tours were published by Abraham de Moivre and Leonhard Euler.

Definitions

A *Hamiltonian path* or *traceable path* is a path that visits each vertex of the graph exactly once. A graph that contains a Hamiltonian path is called a traceable graph. A graph is Hamiltonian-connected if for every pair of vertices there is a Hamiltonian path between thetwo vertices.

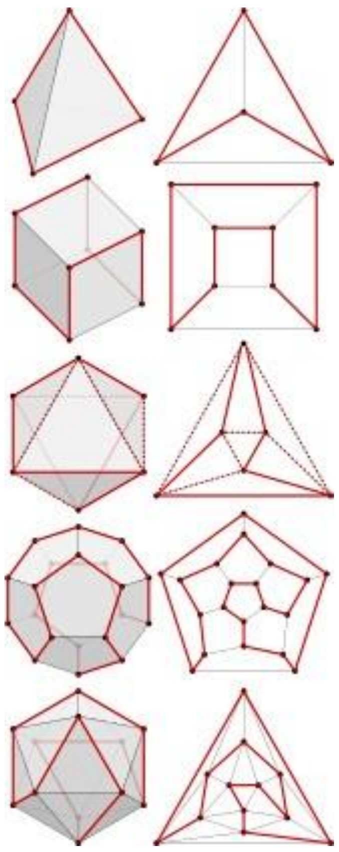
A *Hamiltonian cycle*, *Hamiltonian circuit*, *vertex tour* or *graph cycle* is a cycle that visits each vertex exactly once. A graph that contains a Hamiltonian cycle is called a Hamiltoniangraph.

Similar notions may be defined for *directed graphs*, where each edge (arc) of a path or cyclecan only be traced in a single direction (i.e., the vertices are connected with arrows and the edges traced "tail-to-head").

A Hamiltonian decomposition is an edge decomposition of a graph into Hamiltonian circuits.

A *Hamilton maze* is a type of logic puzzle in which the goal is to find the unique Hamiltoniancycle in a given graph.

Examples



Orthographic projections and Schlegel diagrams with Hamiltonian cycles of the vertices of the five platonic solids – only the octahedron has an Eulerian path or cycle, by extending itspath with the dotted one.

Examples

- A complete graph with more than two vertices is Hamiltonian

- Every cycle graph is Hamiltonian
- Every tournament has an odd number of Hamiltonian paths (Rédei 1934)
- Every platonic solid, considered as a graph, is Hamiltonian[5]
- The Cayley graph of a finite Coxeter group is Hamiltonian (For more information onHamiltonian paths in Cayley graphs, see the Lovász conjecture.)
- Cayley graphs on nilpotent groups with cyclic commutator subgroup areHamiltonian.

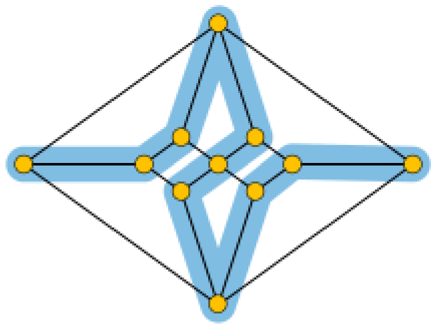
- The flip graph of a convex polygon or equivalently, the rotation graph of binary trees,is Hamiltonian.

Properties

Any Hamiltonian cycle can be converted to a Hamiltonian path by removing one of its edges, but a Hamiltonian path can be extended to Hamiltonian cycle only if its endpoints are adjacent.

All Hamiltonian graphs are biconnected, but a biconnected graph need not be Hamiltonian (see, for example, the Petersen graph).

An Eulerian graph *G* (a connected graph in which every vertex has even degree) necessarily has an Euler tour, a closed walk passing through each edge of *G* exactly once. This tour corresponds to a Hamiltonian cycle in the line graph *L*(*G*), so the line graph of every Eulerian graph is Hamiltonian. Line graphs may have other Hamiltonian cycles that do not correspond to Euler tours, and in particular the line graph *L*(*G*) of every Hamiltonian graph *G* is itself Hamiltonian, regardless of whether the graph *G* is Eulerian.[10]

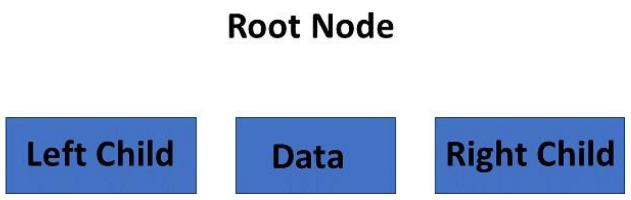

The Herschel graph is the smallest possible polyhedral graph that does not have a Hamiltonian cycle. A possible Hamiltonian path is shown.

A tournament (with more than two vertices) is Hamiltonian if and only if it is strongly connected.

The number of different Hamiltonian cycles in a complete undirected graph on *n* vertices is (*n* – 1)!/2 and in a complete directed graph on *n* vertices is (*n* – 1)!. These counts assume that cycles that are the same apart from their starting point are not counted separately.

TREE

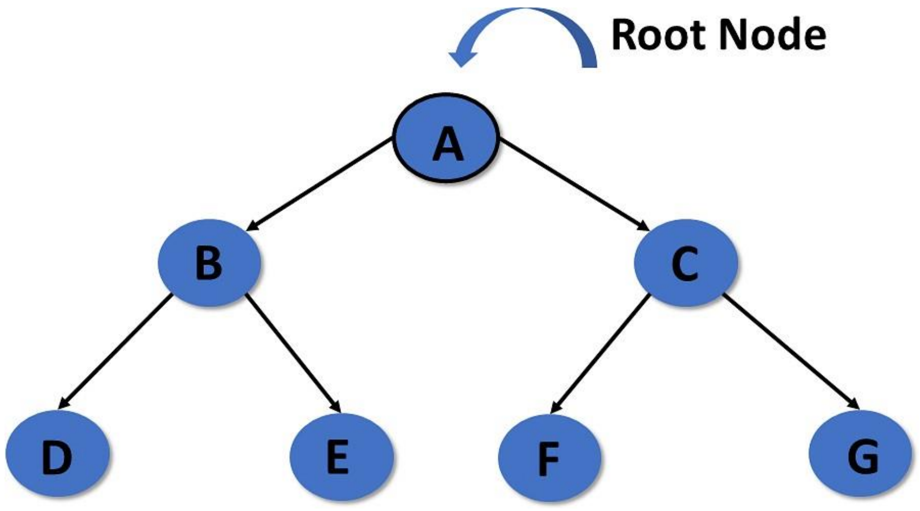
The tree is a nonlinear hierarchical data structure and comprises a collection of entitiesknown as nodes. It connects each node in the tree data structure using "edges", both directed and undirected.



Tree Terminologies

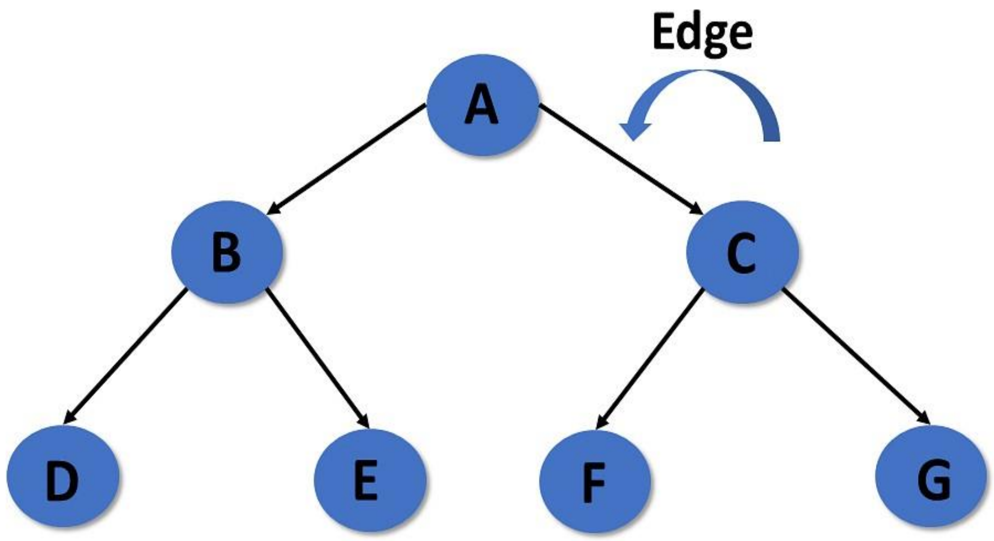
Root

- In a tree data structure, the root is the first node of the tree. The root node is the initial nodeof the tree in data structures.
- In the tree data structure, there must be only one root node.



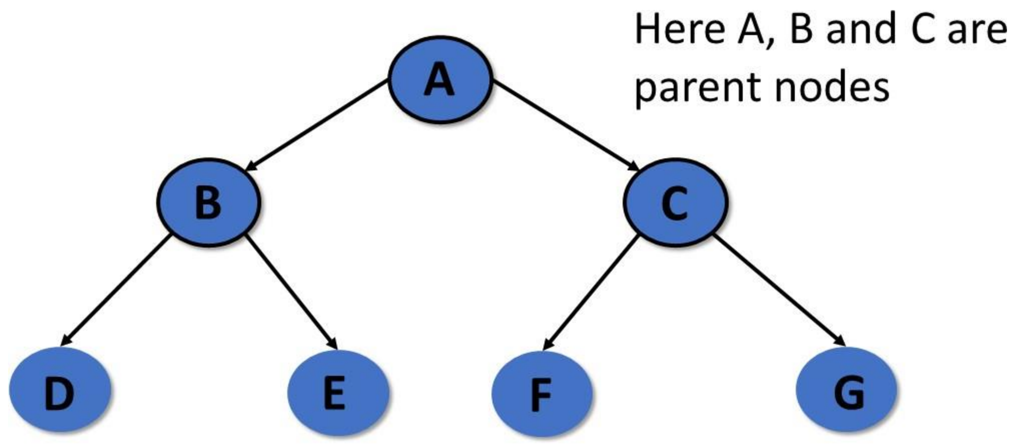
Edge

- In a tree in data structures, the connecting link of any two nodes is called the edge of the treedata structure.
- In the tree data structure, N number of nodes connecting with N -1 number of edges.



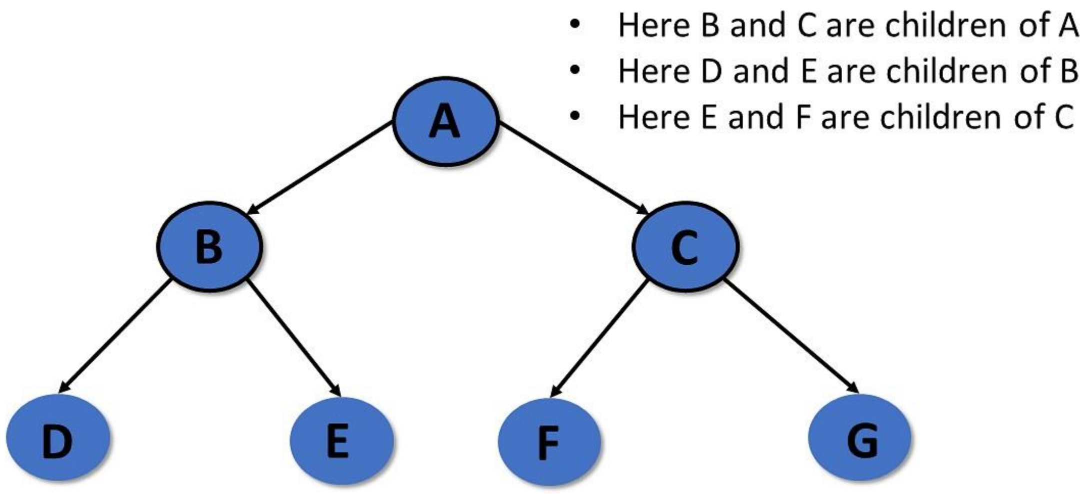
Parent

In the tree in data structures, the node that is the predecessor of any node is known as a parent node, or a node with a branch from itself to any other successive node is calledthe parent node.



Child

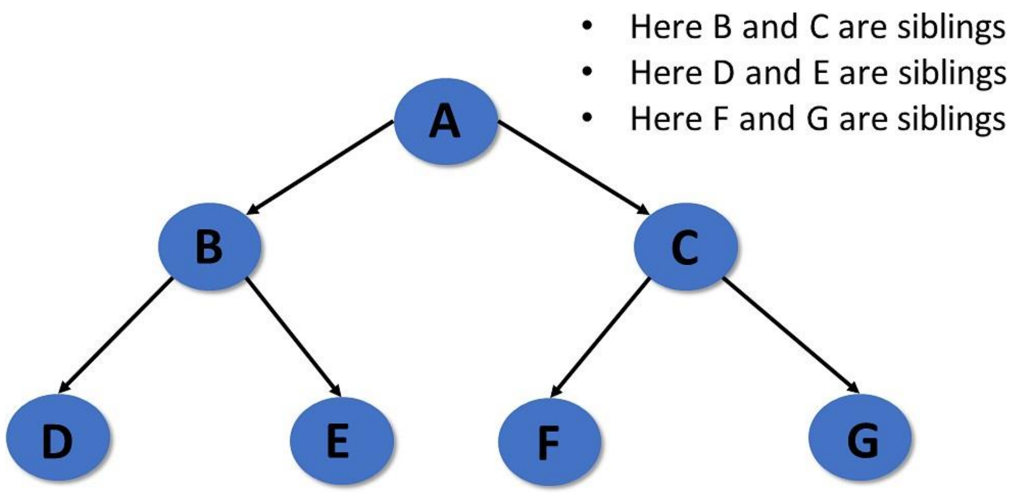
- The node, a descendant of any node, is known as child nodes indata structures.
- In a tree, any number of parent nodes can have any number ofchild nodes.
- In a tree, every node except the root node is a child node.



Here B and C are children of A
Here D and E are children of B
Here E and F are children of C
A
B
C
D
E
F
G

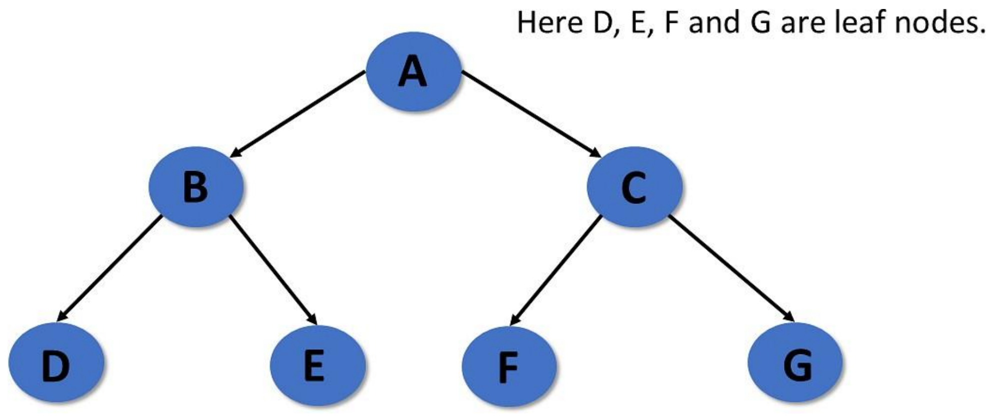
Siblings

In trees in the data structure, nodes that belong to the same parentare called siblings.


Here B and C are siblings
Here D and E are siblings
Here F and G are siblings
A
B
C
D
E
F
G

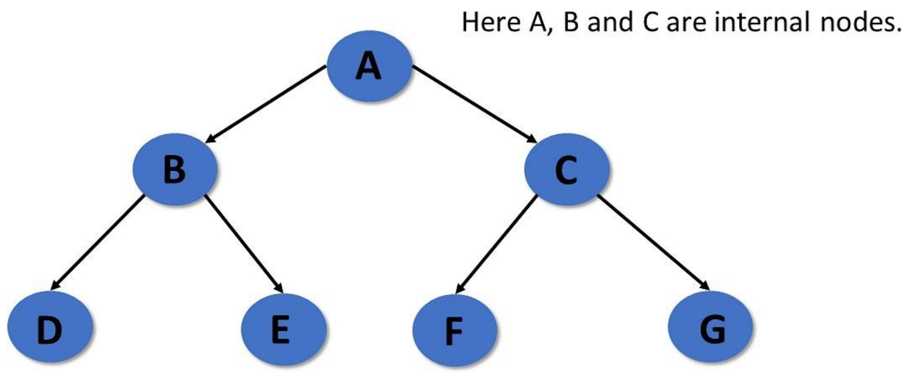
Leaf

- Trees in the data structure, the node with no child, is known as aleaf node.
- In trees, leaf nodes are also called external nodes or terminalnodes.



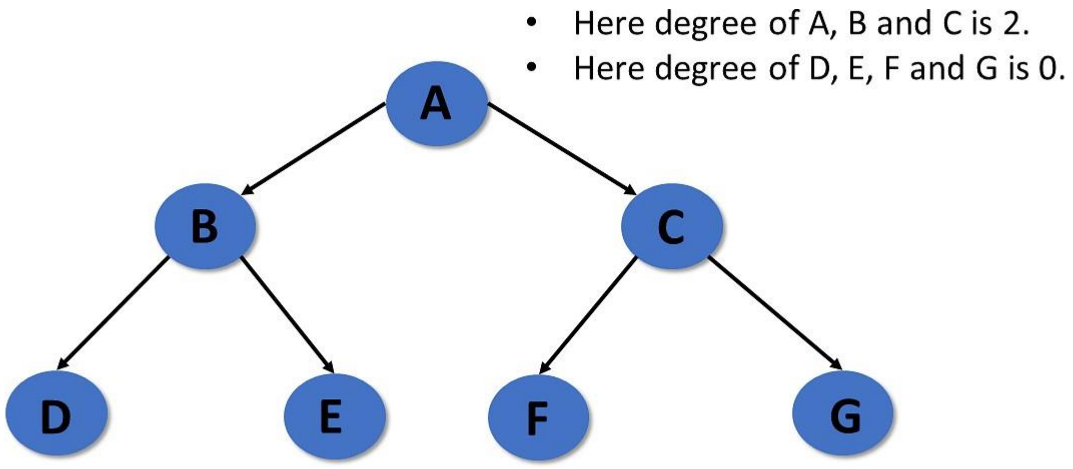
Internal nodes

- Trees in the data structure have at least one child node known asinternal nodes.
- In trees, nodes other than leaf nodes are internal nodes.
- Sometimes root nodes are also called internal nodes if the tree hasmore than one node.



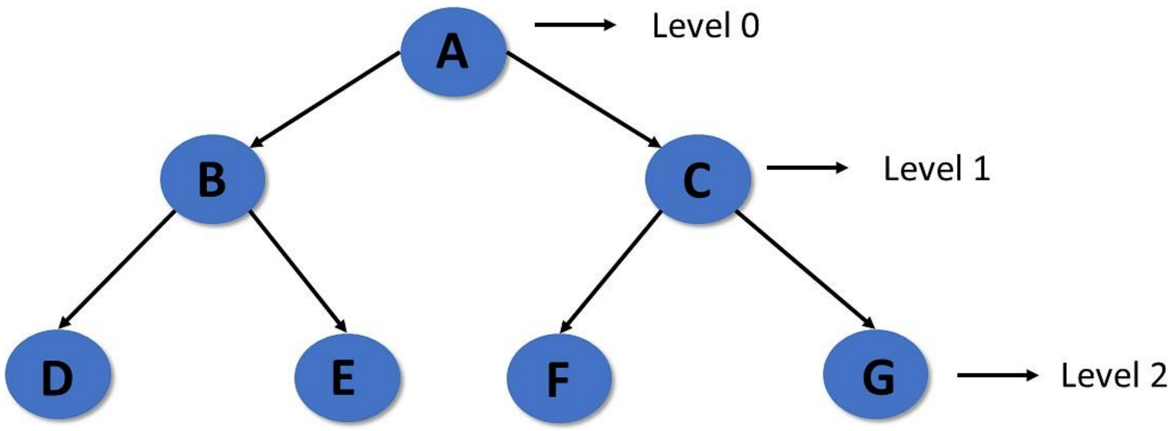
Degree

- In the tree data structure, the total number of children of a node iscalled the degree of the node.
- The highest degree of the node among all the nodes in a tree iscalled the Degree of Tree.


Here degree of A, B and C is 2.
Here degree of D, E, F and G is 0.
A
B
C
D
E
F
G

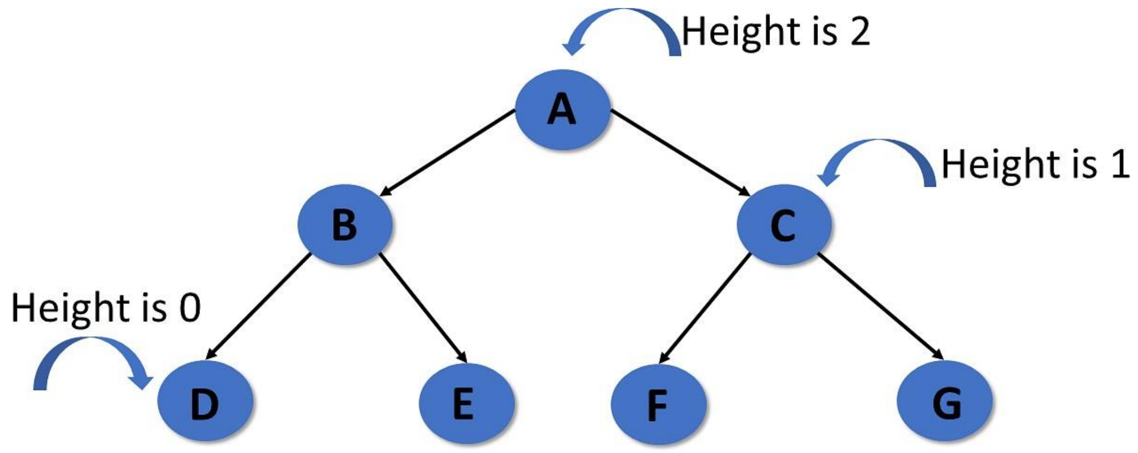
Level

In tree data structures, the root node is said to be at level 0, and theroot node's children are at level 1, and the children of that node at level 1 will be level 2, and so on.



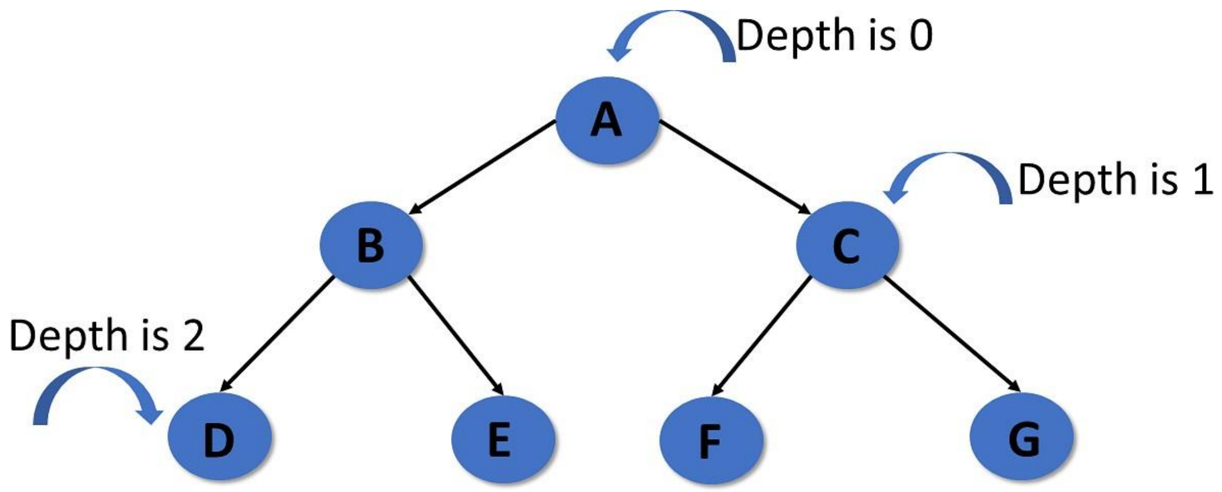
Height

- In a tree data structure, the number of edges from the leaf node tothe particular node in the longest path is known as the height of that node.
- In the tree, the height of the root node is called "Height of Tree".
- The tree height of all leaf nodes is 0.



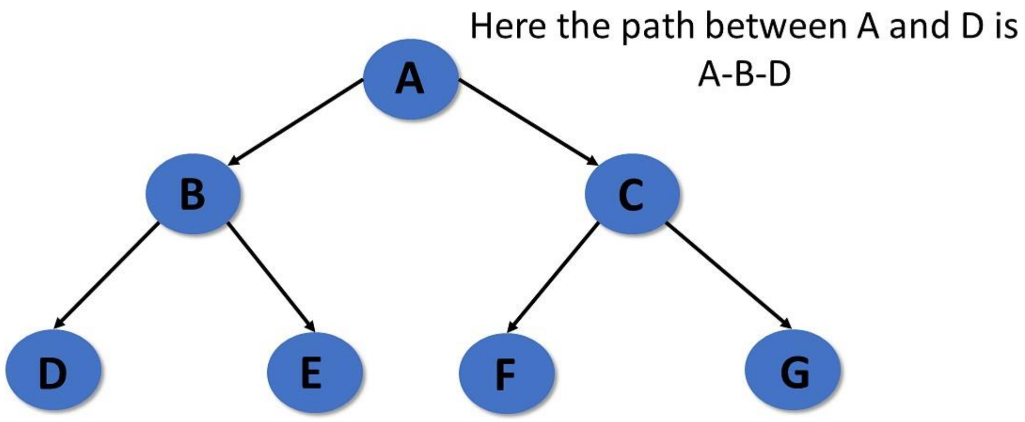
Depth

- In a tree, many edges from the root node to the particular node arecalled the depth of the tree.
- In the tree, the total number of edges from the root node to the leafnode in the longest path is known as "Depth of Tree".
- In the tree data structures, the depth of the root node is 0.



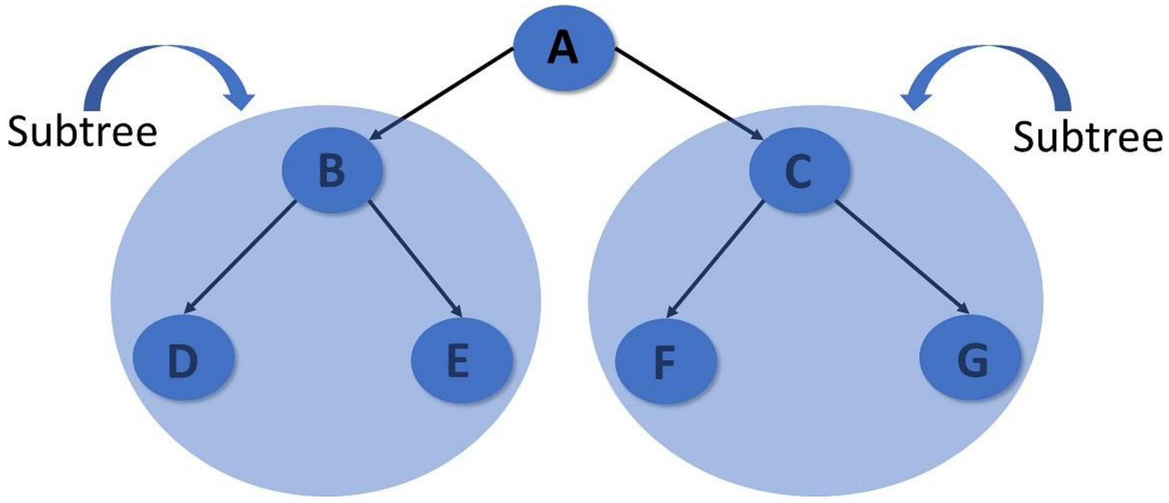
Path

- In the tree in data structures, the sequence of nodes and edgesfrom one node to another node is called the path between thosetwo nodes.
- The length of a path is the total number of nodes in a path.zx



Subtree

In the tree in data structures, each child from a node shapes a sub-tree recursively and every child in the tree will form a sub-tree on itsparent node.


A
Subtree
B
D
E
C
F
G
Subtree

Tree Traversal

Traversal of the tree in data structures is a process of visiting each node and prints their value. There are three ways to traverse tree data structure.

- Pre-order Traversal
- In-Order Traversal
- Post-order Traversal

In-Order Traversal

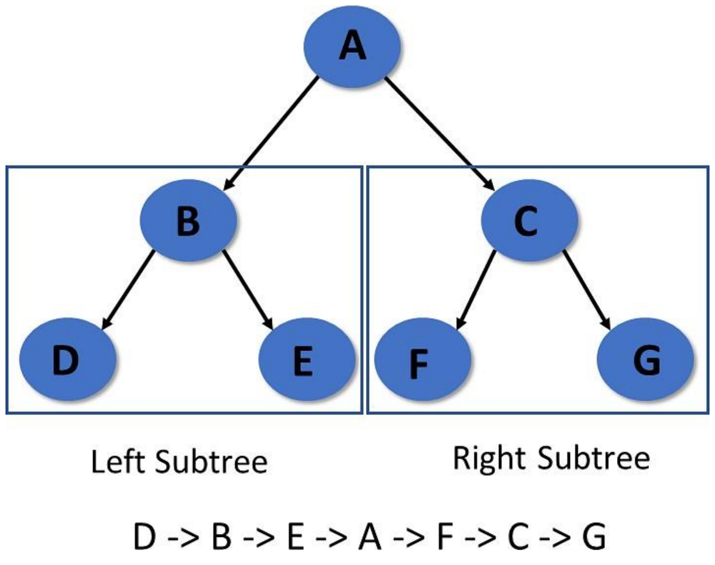
In the in-order traversal, the left subtree is visited first, then the root, and later the right subtree.

Algorithm:

Step 1- Recursively traverse the left subtree Step 2- Visit

root node

Step 3- Recursively traverse right subtree



Pre-Order Traversal

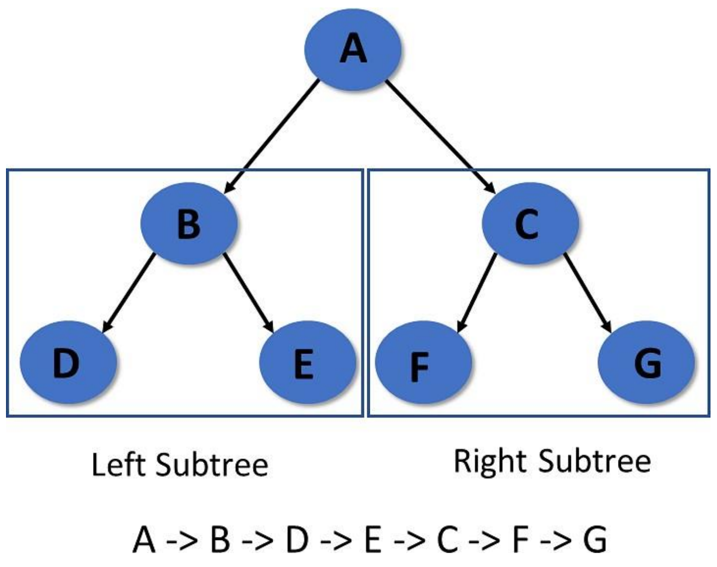
In pre-order traversal, it visits the root node first, then the left subtree, and lastly right subtree.

Algorithm:

Step 1- Visit root node

Step 2- Recursively traverse the left subtree Step 3-

Recursively traverse right subtree



Post-Order Traversal

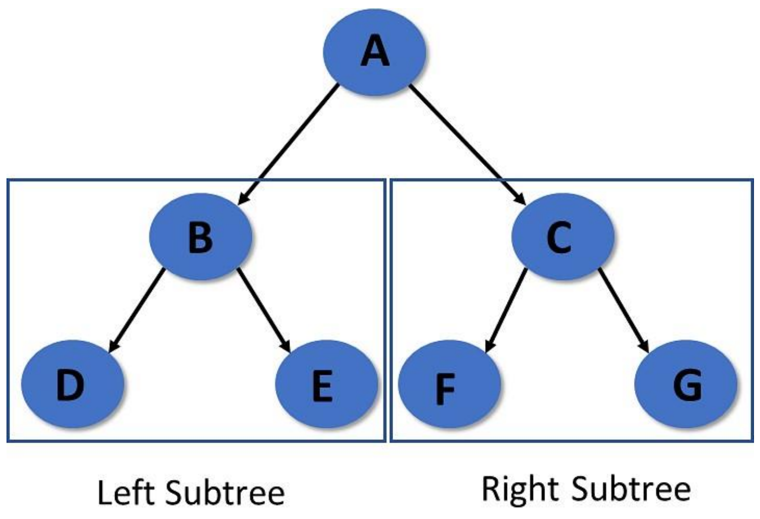
It visits the left subtree first in post-order traversal, then the right subtree, and finally the root node.

Algorithm:

Step 1- Recursively traverse the left subtree Step 2- Visit

root node

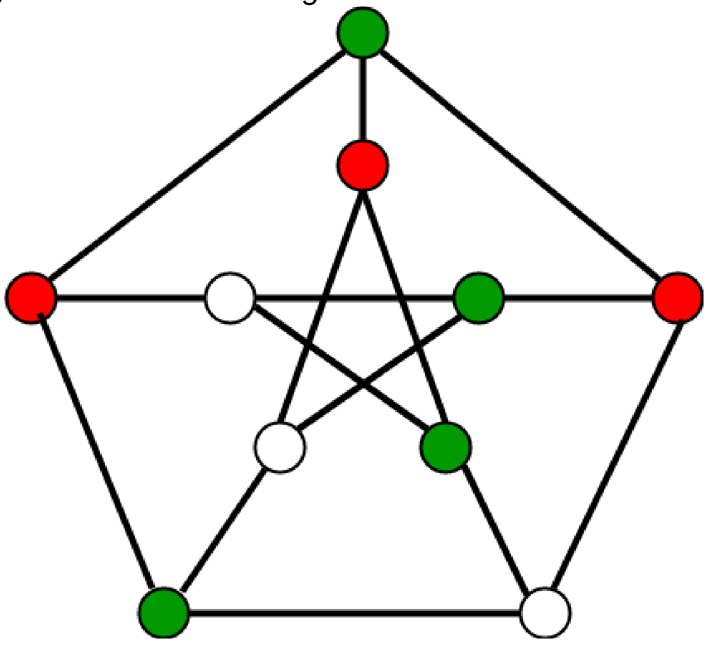
Step 3- Recursively traverse right subtree



D -> E -> B -> F -> G -> C -> A

<u>**Graph Coloring**</u>

Graph coloring *refers to the problem of* ***coloring vertices*** *of a graph in such a way that* ***no two adjacent*** *vertices have the* ***same color****. This is also called the* ***vertex coloring*** *problem. If coloring is done using at most m colors, it is called m-coloring.*



Graph Coloring

Chromatic Number:
The ***minimum*** *number of* ***colors*** *needed to* ***color*** *a graph is called its* ***chromatic*** *number. For example, the following can be colored a minimum of 2 colors.*

Algorithm of Graph Coloring using Backtracking**:**
Assign colors one by one to different vertices, starting from vertex ***0****. Before assigning a color, check if the adjacent vertices have the same color or not. If there is any color assignment that does not violate the conditions, mark the color assignment as part of the solution. If no assignment of color is possible then backtrack and return false.*

Method to Color a Graph

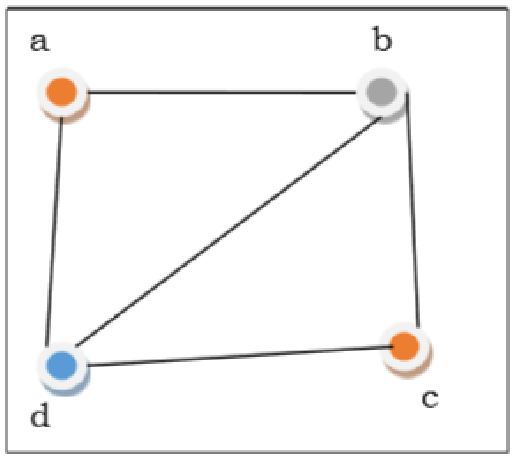
The steps required to color a graph G with n number of vertices are as follows −

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

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

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

Example



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

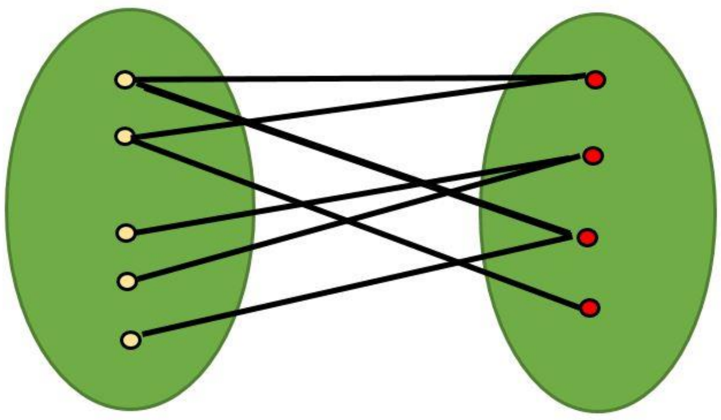
Applications of Graph Coloring

Some applications of graph coloring include −

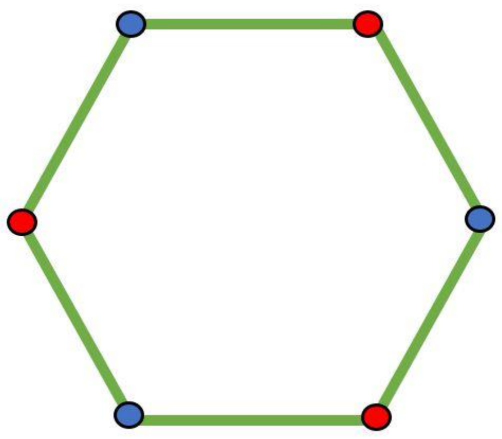
- Register Allocation
- Map Coloring
- Bipartite Graph Checking
- Mobile Radio Frequency Assignment
- Making time table, etc.

Bipartite Graph

A Bipartite Graph is a graph whose vertices can be divided into two independent sets, U and V such that every edge (u, v) either connects a vertex from U to V or a vertex from V to U. In other words, for every edge (u, v), either u belongs to U and v to V, or u belongs to V and v to U. We can also say that there is no edge that connects vertices of same set.

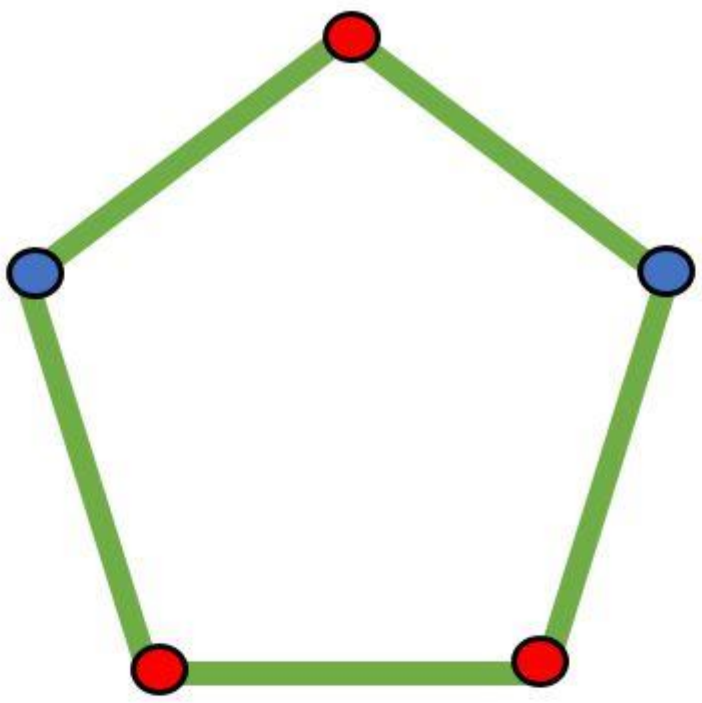


A bipartite graph is possible if the graph coloring is possible using two colors such that vertices in a set are colored with the same color. Note that it is possible to color a cycle graph with even cycle using two colors. For example, see the following graph.



Cycle graph of length 6

It is not possible to color a cycle graph with odd cycle using two colors.



Cycle graph of length 5

Algorithm to check if a graph is Bipartite:
One approach is to check whether the graph is 2-colorable or not using backtracking algorithm m coloring problem.
Following is a simple algorithm to find out whether a given graph is Bipartite or not using Breadth First Search (BFS).
1. Assign RED color to the source vertex (putting into set U).
2. Color all the neighbors with BLUE color (putting into set V).
3. Color all neighbor’s neighbor with RED color (putting into set U).
4. This way, assign color to all vertices such that it satisfies all the constraints of m way coloring problem where m = 2.
5. While assigning colors, if we find a neighbor which is colored with same color as current vertex, then the graph cannot be colored with 2 vertices (or graph is not Bipartite)