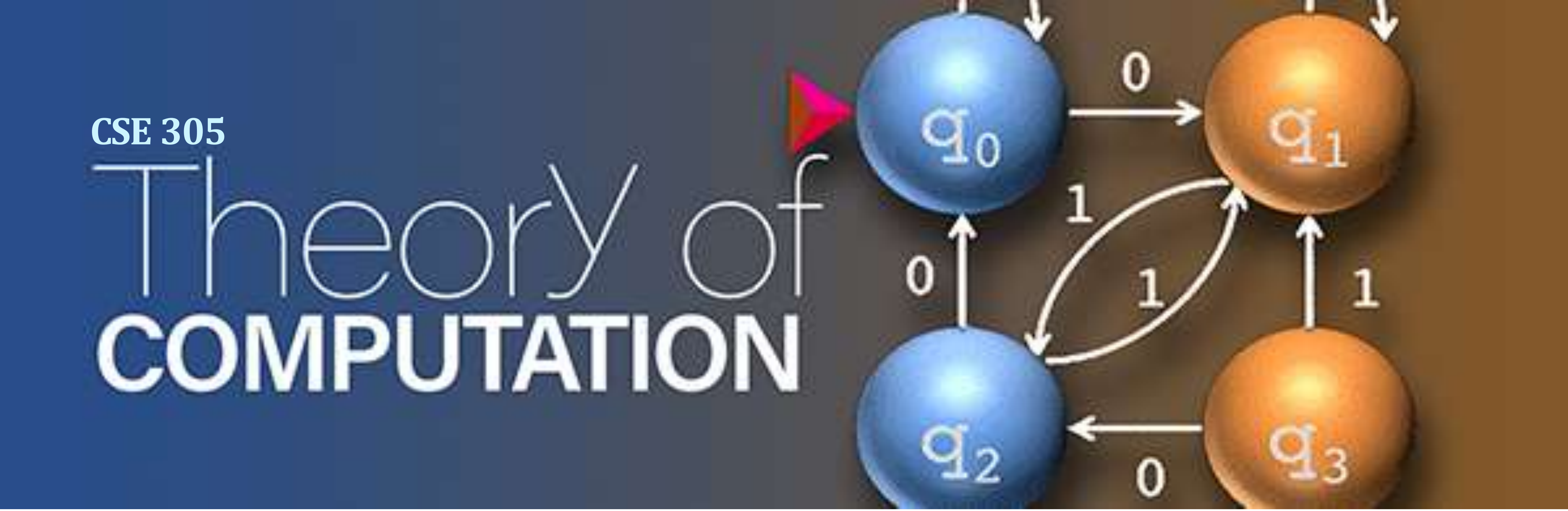


Lecture 9

Mathematical Preliminaries (5)

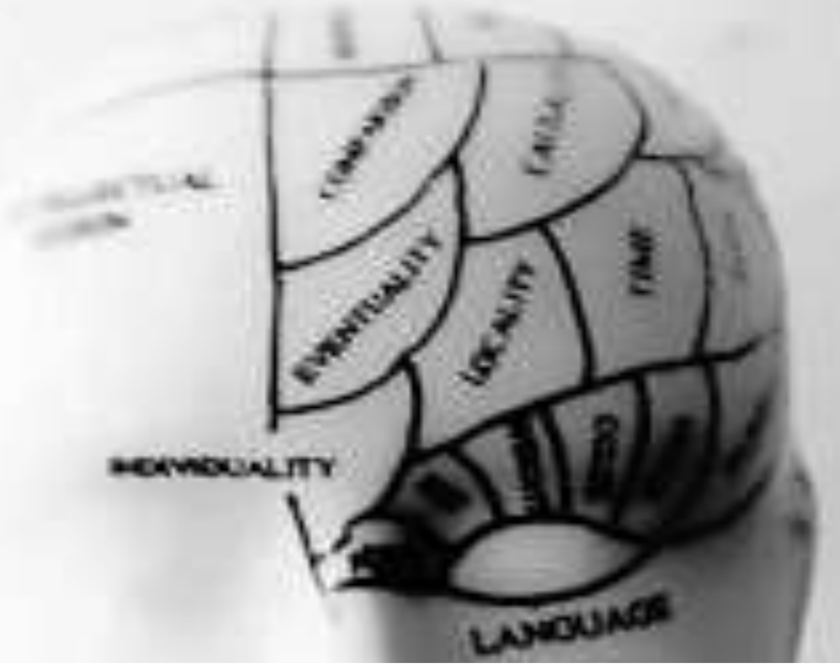
Md. Mijanur Rahman, Prof. Dr.

Dept. of Computer Science and Engineering, Jatiya Kabi Kazi Nazrul Islam University, Bangladesh.

www.mijanrahman.com

Contents

Mathematical Preliminaries



Alphabets, Strings and Languages

- **Alphabets:** The alphabet of any language is defined as a finite, non-empty set of symbols. These form the basic units of strings. Alphabets may be denoted by $\sum$.
  - Some examples of alphabets include: binary alphabets, $\sum = \{0, 1\}$; English alphabets, $\sum = \{A, B, C, .., Z\}$, numbers, $\sum = \{0, 1, 2, \ldots, 9\}$, etc.
- **Strings:** The symbols from an alphabet can be combined to form words or strings. A string is defined as a finite sequence of symbols chosen from some alphabets over $\sum$.
  - For example: (i) 01, 011, 0000, 101011 are all strings over an alphabet $\sum = \{0, 1\}$; (ii) aa, ac, abc, aabb are all strings over an alphabet $\sum = \{a, b, c\}$; (iii) 10, 123, 1024 are all strings over an alphabet $\sum = \{0, 1, 2, 3, \ldots, 9\}$.

Alphabets, Strings and Languages

Languages:

- The finite or infinite set of all strings obtained from the alphabet Σ is called **language**. A language is a subset of Σ^*, and defined as-

$$L = \{w\ \Sigma^* \mid w \text{ is a string over } \Sigma \text{ and has some specific property}\}$$

- **Example:**

(i) $L_1 = \{0, 00, 101, 1100\}$ is a language over the alphabet $\Sigma=\{0, 1\}$;

(ii) $L_2 = \{ww^R \mid w \in \{a, b\}^*\}$ is a language over the alphabet $\Sigma = \{a, b\}$.

Alphabets, Strings and Languages

Operations on Languages:

- **Union of Languages:** The union of languages is similar to the union of sets. The union of two languages is defined as-

$$L_1 \cup L_2 = \{x \mid x \in L_1 \text{ or } x \in L_2\}$$

- **Intersection of Languages:** The intersection of languages is similar to the intersection of sets. The intersection of two languages is defined as-

$$L_1 \cap L_2 = \{x \mid x \in L_1 \text{ and } x \in L_2\}$$

- **Concatenation of Languages:** Concatenation of languages involves grouping all possible strings formed by combining strings of two different languages. Concatenation of two languages is defined as-

$$L_1 L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\}$$

Alphabets, Strings and Languages

Operations on Languages:

- **Reversal of Languages:** The reversal of a language L is denoted by L^R and it is defined as-

  $L^R = \{x \mid x^R \in L\}$

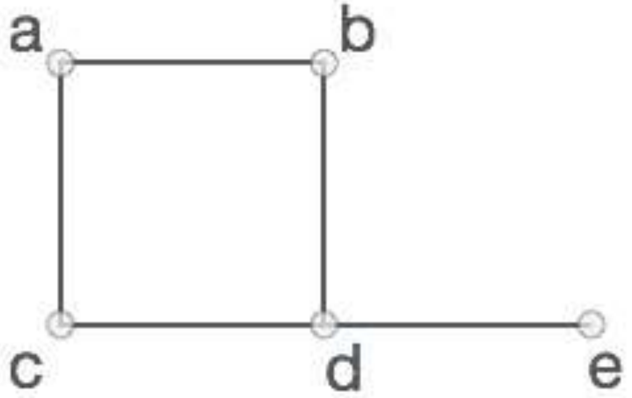
  Reversal of a language L is defined as a collection of reversal of all strings of L.

- **Kleene Star (Kleene Closure):** The Kleene star of a language L is defined as the laaguage L^* consisting of all the strings obtained by concatenating any finite number (including zero) of strings from L together.

  For example, the Kleene star of a language L given by {0, 1} may be written as $L^* = \{\varepsilon, 0, 1, 10, 00, 01, 11, 100, 110, 101, \ldots\}$

Graphs

- **A graph is a pictorial representation of a set of objects** where some pairs of objects are connected by links. The interconnected objects are represented by points termed as **vertices**, and the links that connect the vertices are called **edges**.
- Formally, a graph **G** is a pair of sets **(V, E)**, where **V** is the set of vertices and **E** is the set of edges, connecting the pairs of vertices, such as **G = (V, E):**
  1. A set **V** of elements called nodes (or points or vertices)
  2. A set **E** of edges, such that each edge **e** in **E** is identified with a unique (unordered) pair **[u, v]** of nodes in **V**, denoted by **e = [u, v]**; the nodes **u** and **v** are called endpoints of **e**, and **u** and **v** are said to be adjacent nodes or neighbors.

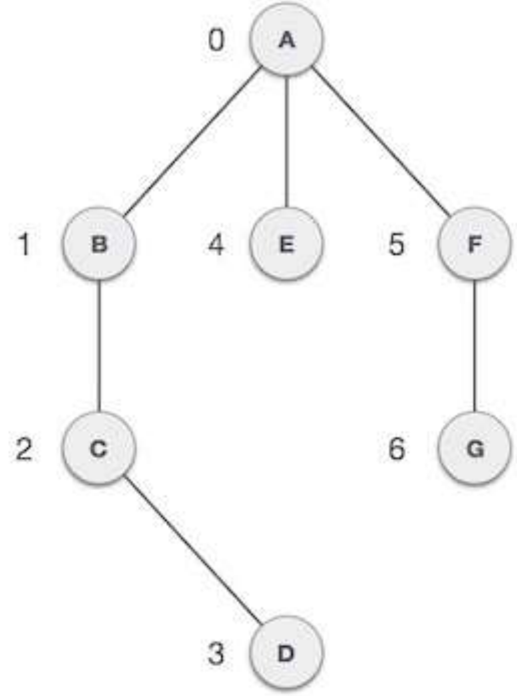


- In the given graph:
  V = {a, b, c, d, e}
  E = {ab, ac, bd, cd, de}

Graphs

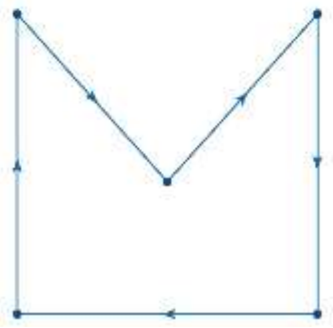
Graph Terminology:

- **Vertex** – Each node of the graph is represented as a vertex. In the example, the labeled circle represents vertices. Thus, A to G are vertices.
- **Edge** – Edge represents a path between two vertices or a line between two vertices. In the example, the lines from A to B, B to C, and so on represents edges.
- **Adjacency** – Two node or vertices are adjacent if they are connected to each other through an edge. In the example, B is adjacent to A, C is adjacent to B, and so on.
- **Path** – Path represents a sequence of edges between the two vertices. In the example, ABCD represents a path from A to D.

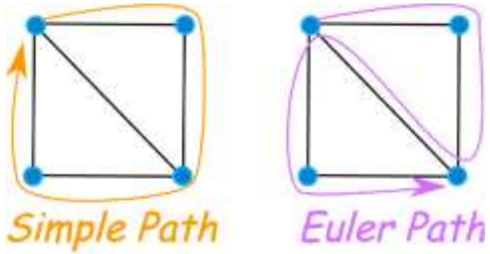


Graphs

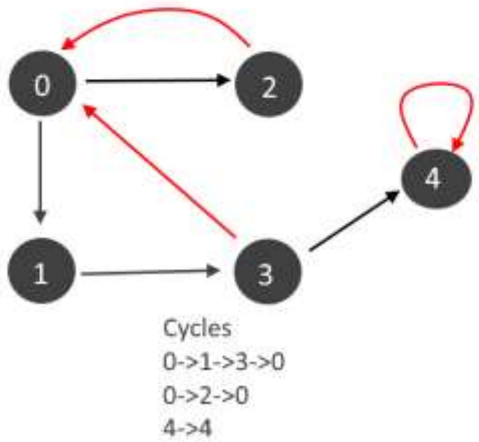
Graph Terminology:



- **Closed Path** - A path will be called as closed path if the initial node is same as terminal node. A path will be closed path if $V_0=V_N$.



- **Simple Path and Euler Path** - If all the nodes of the graph are distinct with an exception $V_0=V_N$, then such path P is called as closed simple path. A path is called an **Euler path** if it traverses each edge lying in it exactly once. A path is called an **Euler circuit** if it traverses each edge lying in it exactly once and whose initial and final vertices are the same.

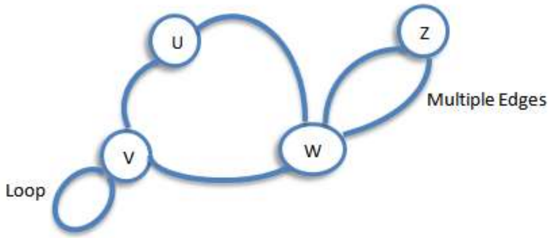


- **Cycle** - A cycle can be defined as the path which has no repeated edges or vertices except the first and last vertices.

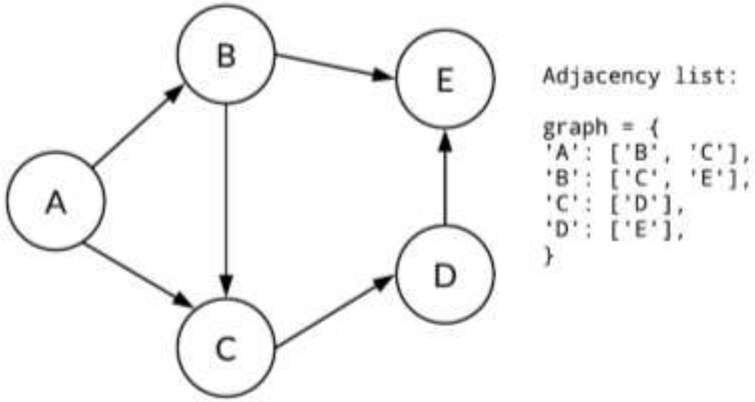
Graphs

Graph Terminology:

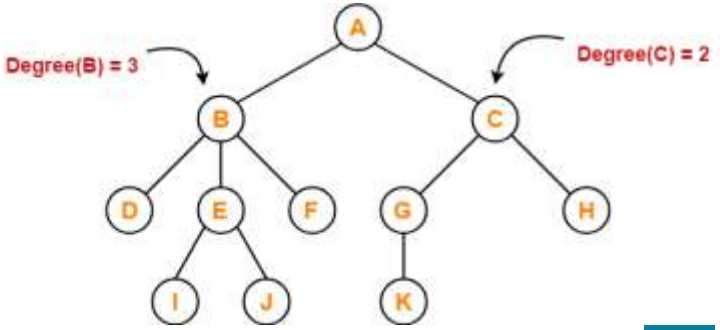
- **Loop -** An edge that is associated with the similar end points can be called as Loop.



- **Adjacent Nodes -** If two nodes u and v are connected via an edge e, then the nodes u and v are called as neighbours or adjacent nodes.



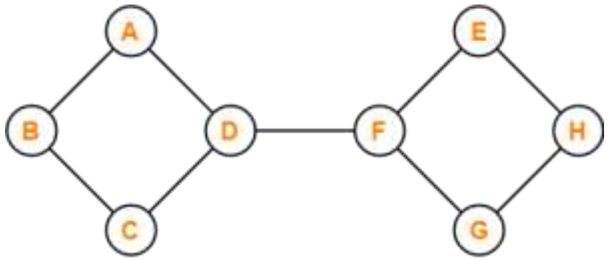
- **Degree of the Node -** A degree of a node is the number of edges that are connected with that node. A node with degree 0 is called as isolated node.



Graphs

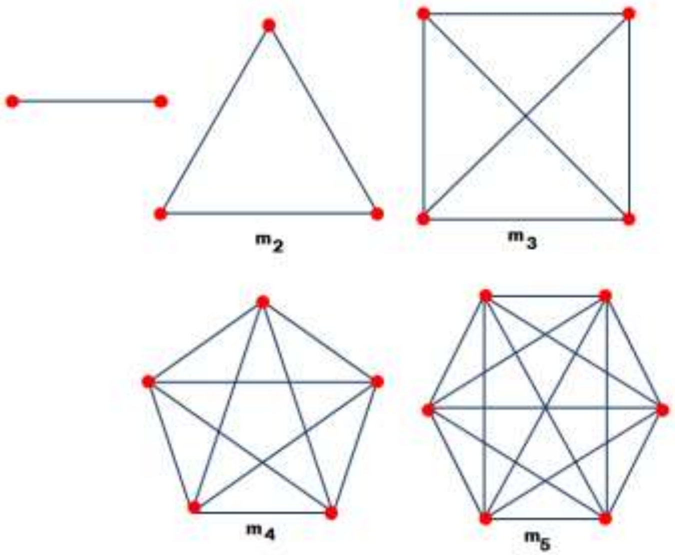
Types of Graphs:

- **Connected Graph -** A connected graph is the one in which some path exists between every two vertices (u, v) in V. There are no isolated nodes in connected graph.



Example of Connected Graph

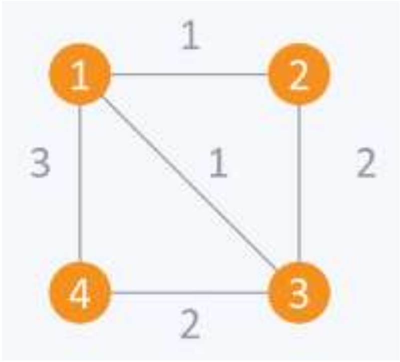
- **Complete Graph -** A complete graph is the one in which every node is connected with all other nodes. A complete graph contain n(n-1)/2 edges where n is the number of nodes in the graph.



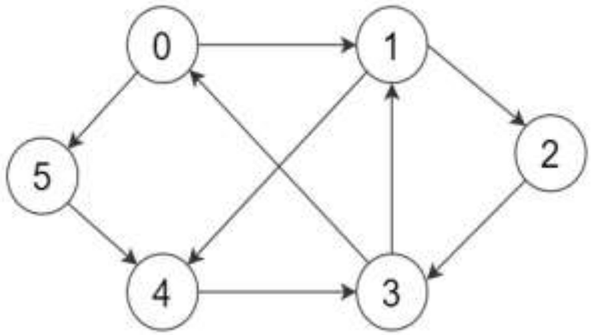
Graphs

Types of Graphs:

- **Weighted Graph -** In a weighted graph, each edge is assigned with some data such as length or weight. The weight of an edge e can be given as w(e) which must be a positive (+) value indicating the cost of traversing the edge.



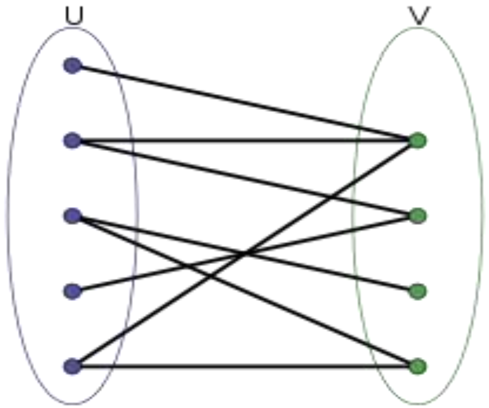
- **Digraph -** A digraph is a directed graph in which each edge of the graph is associated with some direction and the traversing can be done only in the specified direction.



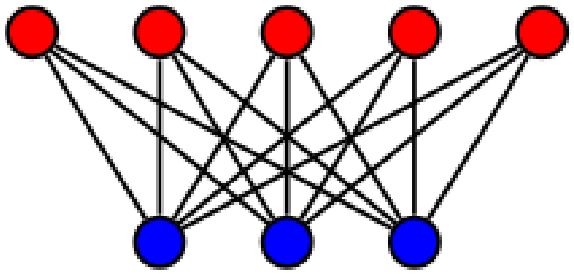
Graphs

Types of Graphs:

- **Bipartite Graph -** A graph G = (V, E) is called a bipartite graph (also known as a biograph) if its vertices can be decomposed into two disjoint sets such that no two graph vertices within the same set are adjacent. A bipartite graph is a special case of a *k*-partite graph with .

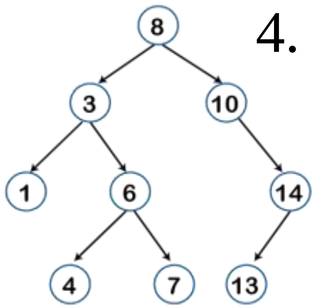


- **Complete Bipartite Graph -** A graph G = (V, E) is called a complete bipartite graph if its vertices V can be partitioned into two subsets V_1 and V_2 such that each vertex of V_1 is connected to each vertex of V_2. The number of edges in a complete bipartite graph is m*n, as each of the m vertices is connected to each of the n vertices.

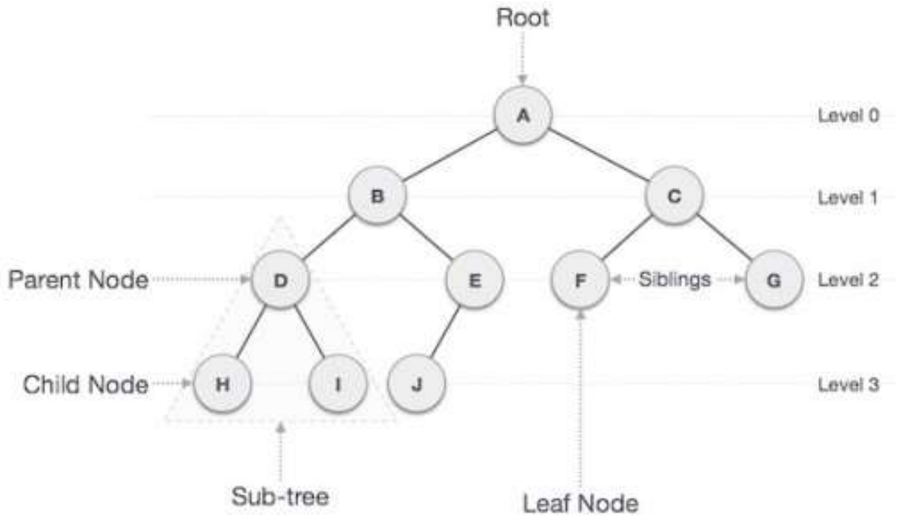


Trees

- **Tree** is mainly used to represent data containing a hierarchical relationship between nodes/elements, connected by **edges**. Tree is a special kind of graph that does not contain any cycle/loop. A tree can also be defined as a non-empty, finite set of elements/nodes, which posses the following properties:
    1. There is a special node called the **root** node of the tree. The root has no incoming edges.
    2. The remaining nodes of the tree form an ordered pair of disjoint sub-trees, if it is a binary tree.
    3. There is exactly one path from the root to every other node in the tree.
    4. The nodes that do not have any outgoing edges are called **leaves** of the tree.



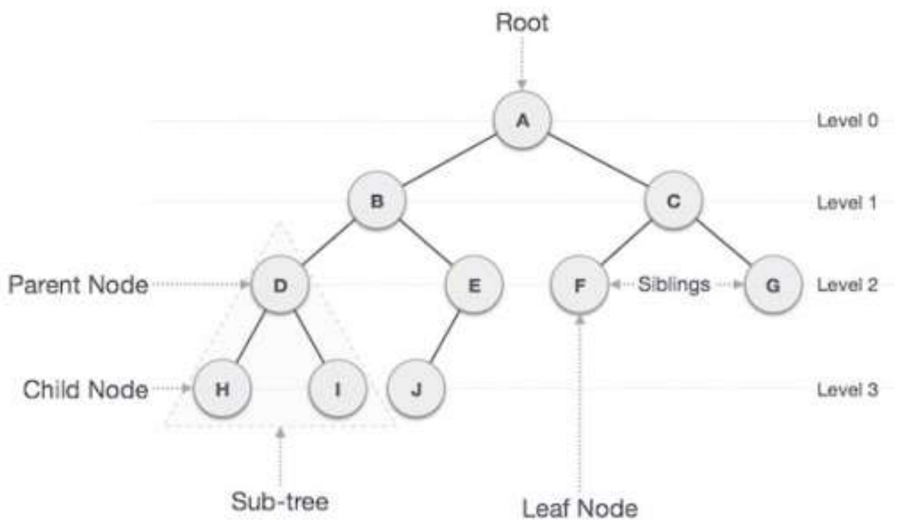
Trees



Tree Terminology:

- **Path** – Path refers to the sequence of nodes along the edges of a tree.
- **Root** – The node at the top of the tree is called root. There is only one root per tree and one path from the root node to any node.
- **Parent** – Any node except the root node has one edge upward to a node called parent.
- **Child** – The node below a given node connected by its edge downward is called its child node.
- **Sibling -** The nodes that have the same parent are known as siblings.
- **Leaf** – The node which does not have any child node is called the leaf node.

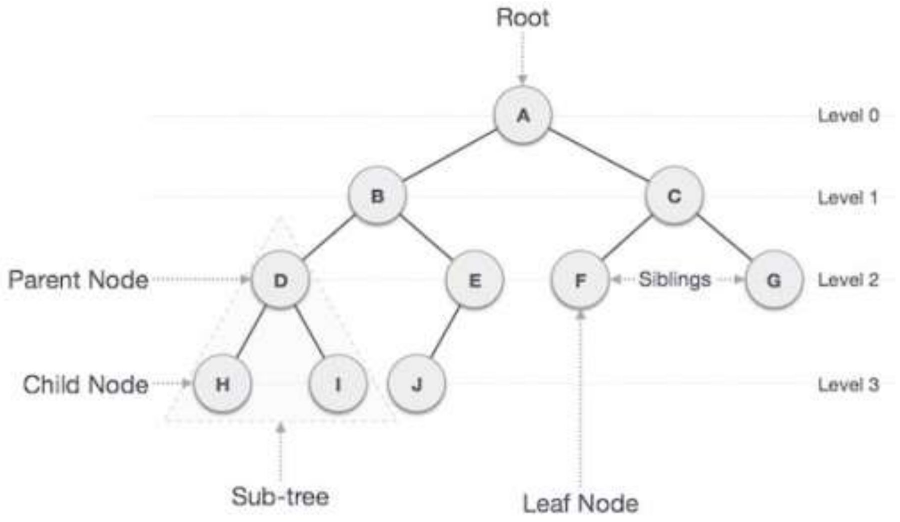
Trees



Tree Terminology:

- **Internal nodes:** A node has at least one child node known as an ***internal***
- **Subtree** – Subtree represents the descendants of a node.
- **Ancestor node:** An ancestor of a node is any predecessor node on a path from the root to that node. The root node doesn't have any ancestors.
- **Descendant:** The immediate successor of the given node is known as a descendant of a node.
- **Visiting:** Visiting refers to checking the value of a node when control is on the node.

Trees



Tree Terminology:

- **Traversing** – Traversing means passing through nodes in a specific order.
- **keys** – Key represents a value of a node based on which a search operation is to be carried out for a node.
- **Levels** – Level of a node represents the generation of a node. If the root node is at level 0, then its next child node is at level 1, its grandchild is at level 2, and so on.

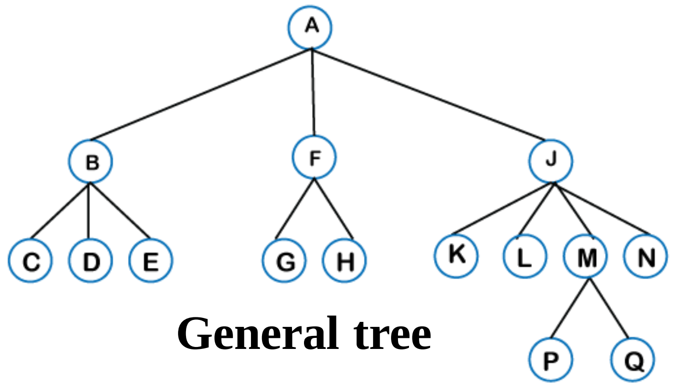
Trees

Types of Trees:

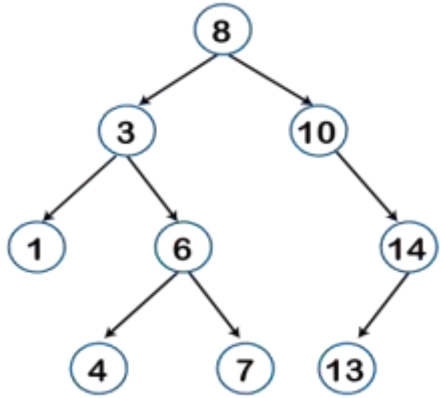
- **General Tree:** A tree where a node can has any number of children/descendants is called a general tree. A node can have **either 0 or maximum n number of nodes**.

- **Binary Tree:** In a binary tree, **each node in a tree can have utmost two child nodes**. Here, utmost means whether the node has 0 nodes, 1 node or 2 nodes.

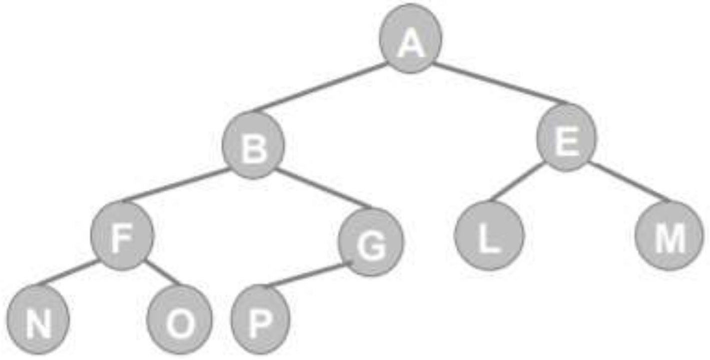
- **Complete Binary Tree:** **A binary tree is said to be complete, if all its levels, except possibly the last have the maximum number of possible nodes**, and if all the nodes at the last level appear as far left as possible.



General tree



Binary tree

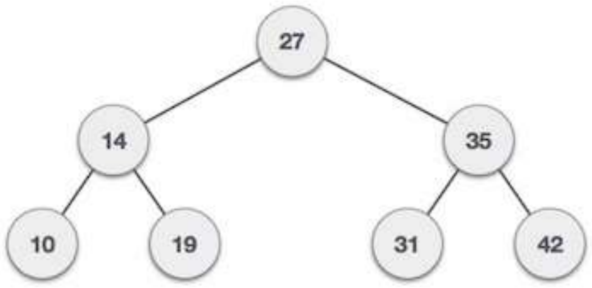


Complete Binary tree

Trees

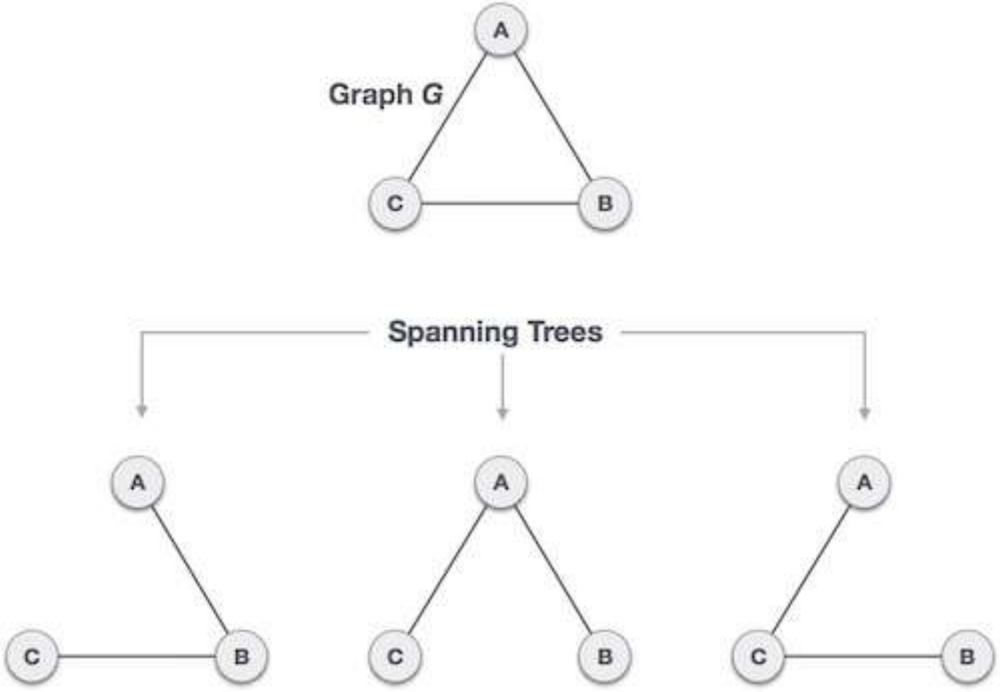
Types of Trees:

- **Binary Search Tree:** Every node in **the left subtree must contain a value less than the value of the root node**, and the value of each node in **the right subtree must be bigger than the value of the root node**.



Binary search tree

- **Spanning Tree:** A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected..



? THE END