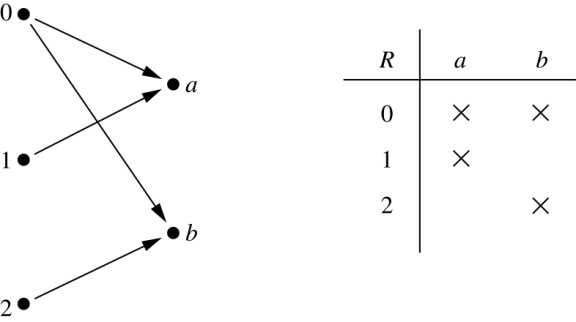
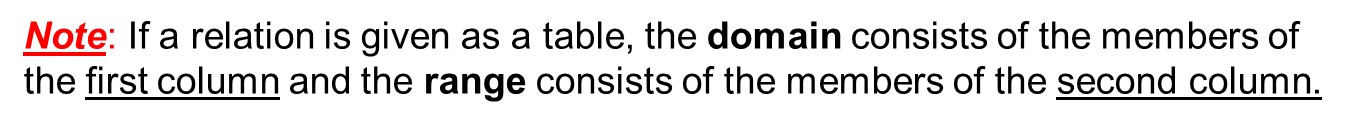
**Discrete Math Viva**

**Lecture 13**

**Binary Relation:** *a binary relation* from ***A* to *B*** is a **set *R*** of ordered pairs where the first element of each ordered pair comes from ***A*** and the second element comes from ***B***.

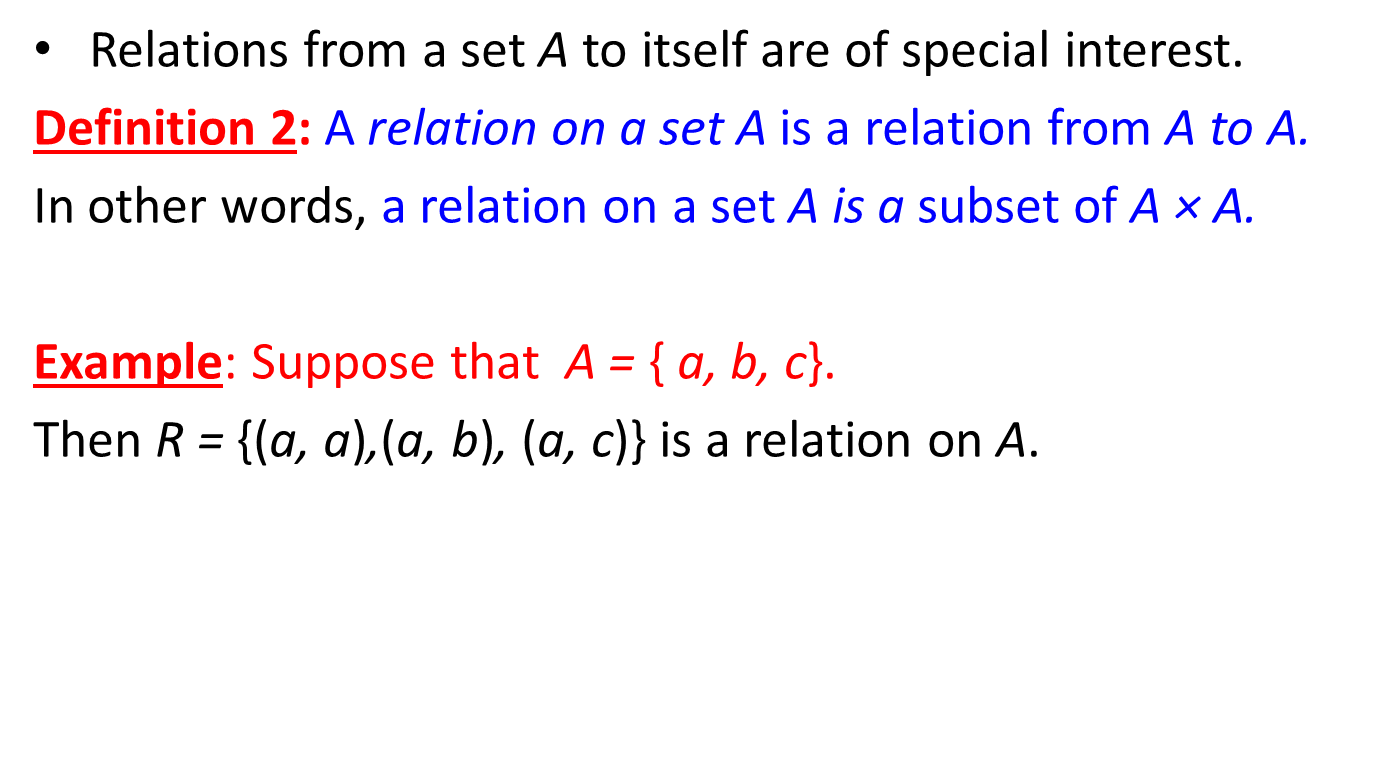
Relations can be represented **graphically** or using a **table.**





|  |  |
| --- | --- |
| Relation | Function |
| express a **one-to-many** relationship | express a **one-to-one** relationship |
| **Relations** are more general than **functions** | A function is a relation where **exactly** **one** **element** |

**Relations on a Set**



Thus, **there are 2*n2* relations on a set with *n* elements.**

* **For example**, there are 232 = 29 = 512 relations on the set {*a, b, c*}

**Properties of Relations**

* ***Reflexive:*** *A relation R on a set A is* ***called reflexive if (a, a)∈*** *R for every element a∈A****.***

***Example: The relation R = {(1, 1), (2, 2), (3, 3)} on the set {1, 2, 3}, is reflexive.***

* ***Symmetric:*** *A relation R on a set A is* ***called symmetric if (b, a)∈ R*** *whenever (a, b)∈ R, for all a, b∈ A.*

***Example:*** *The relation R = {(1,2), (2,1), (3,4), (4,1), (4,3), (1, 4)} on the set {1, 2, 3, 4} is symmetric.*

* ***Antisymmetric****: A relation R on a set A such that for all a, b ∊ A if (a, b) ∊ R and (b, a) ∊ R, then a = b is called antisymmetric.* ***Anti-symmetric- (a, b) (b, a) if a=b if not then it is symmetric again***
* ***Note: The terms symmetric and antisymmetric are NOT opposite, because a relation can have both of these properties or may lack both of them.***

*It follows that* ***R is not antisymmetric*** *if we have* ***a and b in A, a = b, and both a R b or b R a****.*

* ***Transitive*:** A relation *R* on a set *A* is called ***transitive* if whenever *(a, b)∈ R* and *(b, c)∈ R*, then *(a, c)∈ R,* for all *a, b, c ∈ A.***

**Example:** The relation R = {(1,2), (1, 3), (1, 4), (2,3), (2, 4), (4, 4)} on the set {1, 2, 3, 4} is transitive.

**Combining Relations:**

* **Example**: Let *A* = {1,2,3}and *B* *=* {1,2,3,4}. The relations *R*1 = {(1,1), (2,2), (3,3)} and *R*2 = {(1,1), (1,2), (1,3), (1,4)} can be combined using basic set operations to form new relations: ∪(union), ∩ (inter Section), − (sub traction).

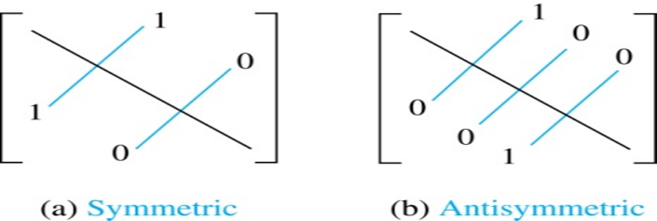
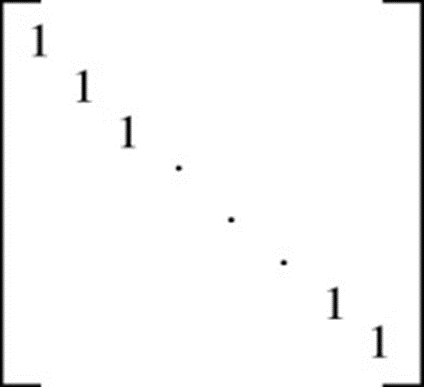
R1 ∪ R2 = {(1,1), (1,2), (1,3), (1,4), (2,2), (3,3)}

*R*1 ∩ R2 = {(1,1)}

*R*1 − *R*2 = {(2,2), (3,3)}

*R*2 − *R*1 = {(1,2), (1,3), (1,4)}

**Lecture 14**

  
Reflexive

**The Pigeonhole Principle**: If k is a positive integer and k+1 or more objects are placed into k boxes, then there is at least one box containing two or more of the objects.

**The Generalized Pigeonhole Principle:** If N objects are placed into k boxes, then there is at least one box containing at least ⎡N/k⎤ objects.

* **Example 5:** Among 100 people there are at least ⎡100/12⎤ = 9 who were born in the same month.

**Lecture 15**

**Underacted Graph Terminology: Different Types of Graphs**

* **Simple Graph:** An ***undirected*** graph with **no multiple edges or loops** is called a simple graph.
* **Multigraph:** An ***undirected*** graph that **may contain multiple edges** connecting the same vertices but **no loops**.
* **Pseudograph:** An ***undirected*** graph that **may contain multiple edges and loops** is called a pseudograph.

**Detracted Graph Terminology: Different Types of Graphs**

* **Simple Directed graph:** When a ***directed*** graph has **no loops** and has **no multiple directed edges**, it is called a simple directed graph.
* **Directed multigraph:** A graph with ***directed edges***that may contain **multiple directed edges** is called a directed multigraph.
* **Mixed Graph:** A graph with ***both directed and undirected edges***is called a mixed graph. A mixed graph may contain loop(s).
* **Loop:** An edge that connect a vertex to itself is called a loop.

****

* The ***degree of a vertex in an undirected graph*** is the ***number of edges incident with it***, **except** that a ***loop*** at a vertex contributes ***twice*** to the degree of that vertex.
  1. The degree of the vertex *v* is denoted by deg(*v*)
* **Isolated vertex**: A vertex of degree zero is called isolated.
* **Pendant vertex**: A vertex is pendant if and only if it has degree one.

**Example:** How many edges are there in a graph with 10 vertices each of degree six?

**Solution:** Because the sum of the degrees of the vertices is 6.10 = 60, it follows that 2e=60.

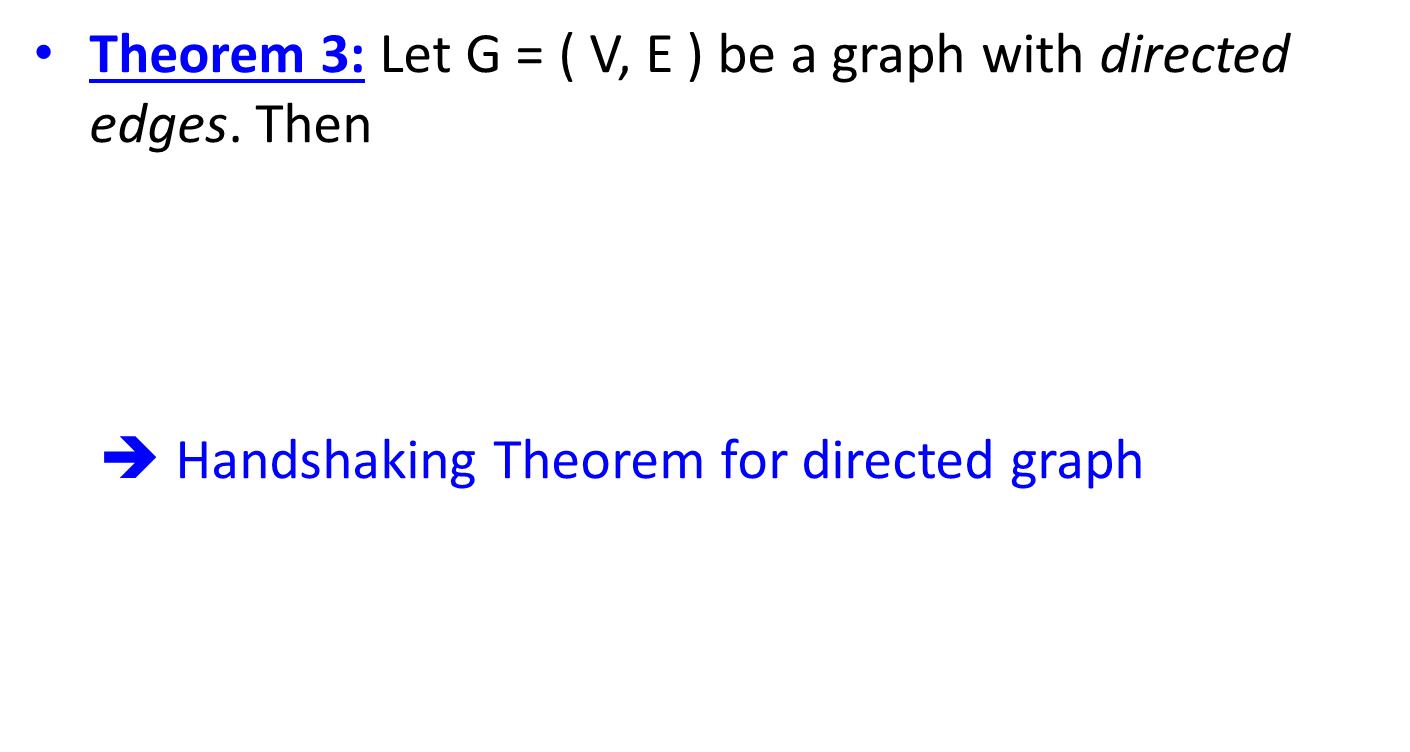
Therefore, e = 30

**Initial vertex & Terminal Vertex**

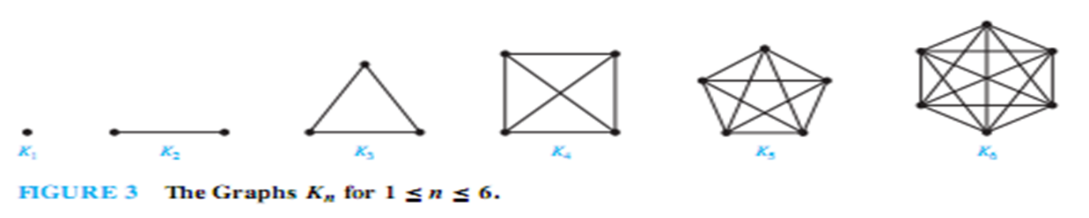
* **Definition:** When (u, v) is an edge of the graph G with directed edges, ***u* is said to be *adjacent to v* and *v is said to be adjacent from u*.**
* The vertex ***u*** is called the ***initial vertex* of (u, v)** and *v* is called the ***terminal/end vertex* of (u, v).**
* ***Note*:** The initial vertex and terminal vertex of a loop are the same.

**In-degree & Out-degree of a vertex**

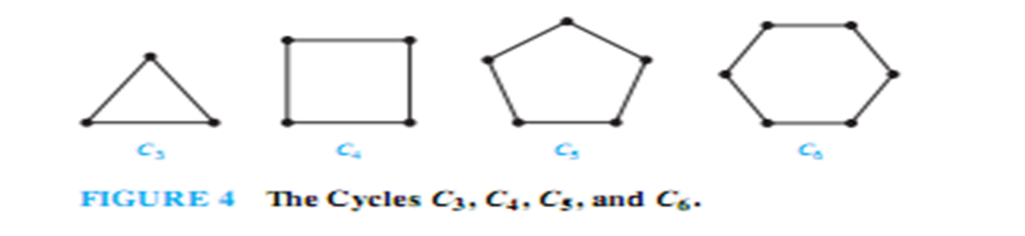
* **Definition** : In a graph with directed edges the ***in-degree of a vertex v***, denoted by **deg–(v)**, is the number of edges with v as their terminal vertex.
* The ***out-degree of v***, denoted by **deg+(v)**, is the number of edges with v as their initial vertex.
* ***Note***: A **loop** at a vertex contributes 1 to both the in-degree and the out-degree of this vertex.



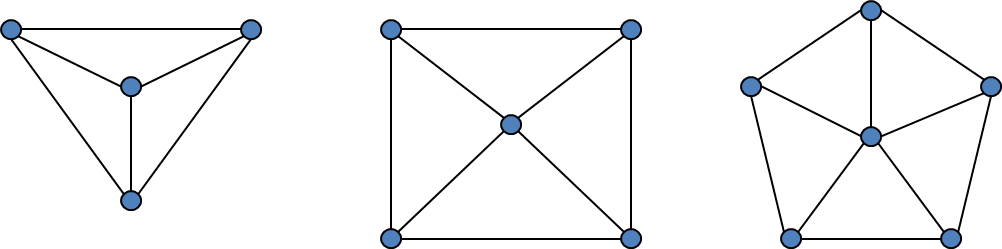
**Complete Graph (*Kn)***



**Cycles Graph (*C*n)**

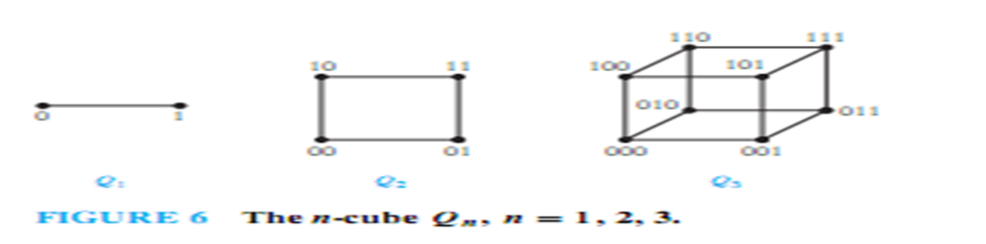


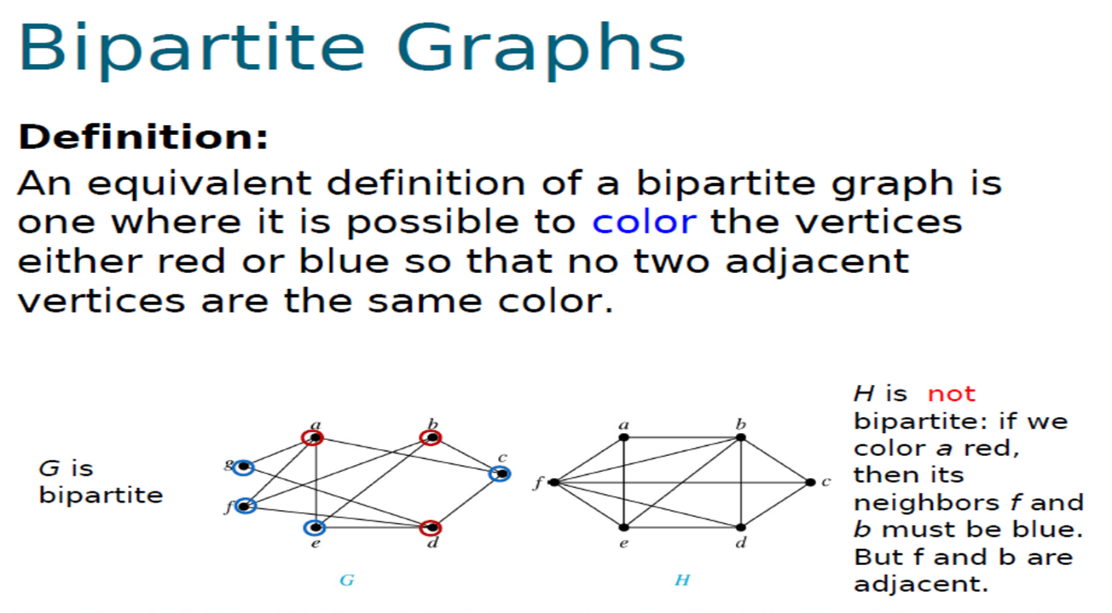
**Wheels Graph (*Wn*)**



The Wheels ***W3, W4, W5***

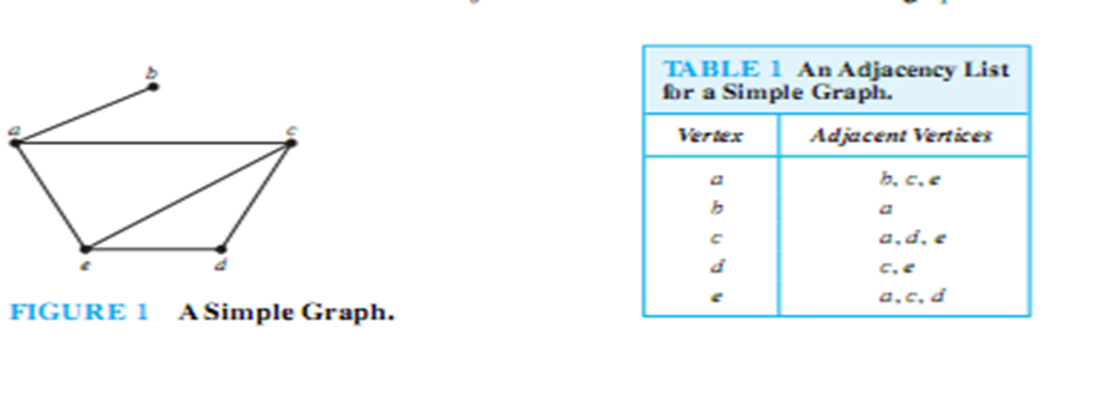
**n-Cubes Graph (*Qn*)**

******

******

**Lecture 16**

* **Adjacency Lists:** A table with 1 row per vertex, listing its adjacent vertices.
* **Directed Adjacency Lists:** A table with 1 row per node, listing the *terminal* *nodes* of each edge *incident* *from* *that node*.



**Representing Graphs using Matrices**

* Two types of **matrices** commonly used to represent graphs –

1. **Adjacency** matrix: A matrix representing a graph using the adjacency of vertices
2. **Incidence** matrix: A matrix representing a graph using the incidence of edges and vertices.

**Isomorphism of Graphs**

* **Isomorphic simple graphs must have** –
  1. The same # of vertices
  2. The same # of edges
  3. The degrees of the vertices must be the same and degrees of adjacent vertices must be same
  4. Same length of simple circuit(s), if any

**Lecture 17**

**Path:** A path is a sequence of edges that begins at a vertex of a graph and travels from vertex to vertex along edges of the graph.

**Lecture 18**

**Euler and Hamilton Paths**

* Euler Paths & Circuits (Edge based)

**Euler path:** A Euler path in G is a simple path containing every edge of G.

**Euler circuit:** A Euler circuit in a graph G is a simple circuit containing every edge of G.

* Hamilton Paths & Circuits (Vertex based)

**Hamilton path**: A **simple** **path** in a graph that passes through every vertex exactly once is called a *Hamilton path*.

**Hamilton circuit**: A **simple circuit** in a graph G that passes through every vertex exactly once is called a *Hamilton circuit*.

**Lecture 19**

**Planar Graphs:** A graph is called planar if it can be drawn in the plane without any edges crossing.

: ***K*4 is planar, *Q*3 is planar,**

**Euler’s** **Formula, *r* = *e* − *v* + 2 r(region), e(edges), v(vertices).**

**Graph Coloring:** A coloring of a simple graph is the assignment of a color to each vertex of the

graph so that no two adjacent vertices are assigned the same color.

**chromatic number:** The chromatic number of a graph is the least number of colors needed for a coloring of this graph.

**Lecture 20**

**Tree:** A ***tree*** is a connected undirected graph with no simple circuits.

|  |  |
| --- | --- |
| Graph | Tree |
| Simple circuit | No Simple circuit. |

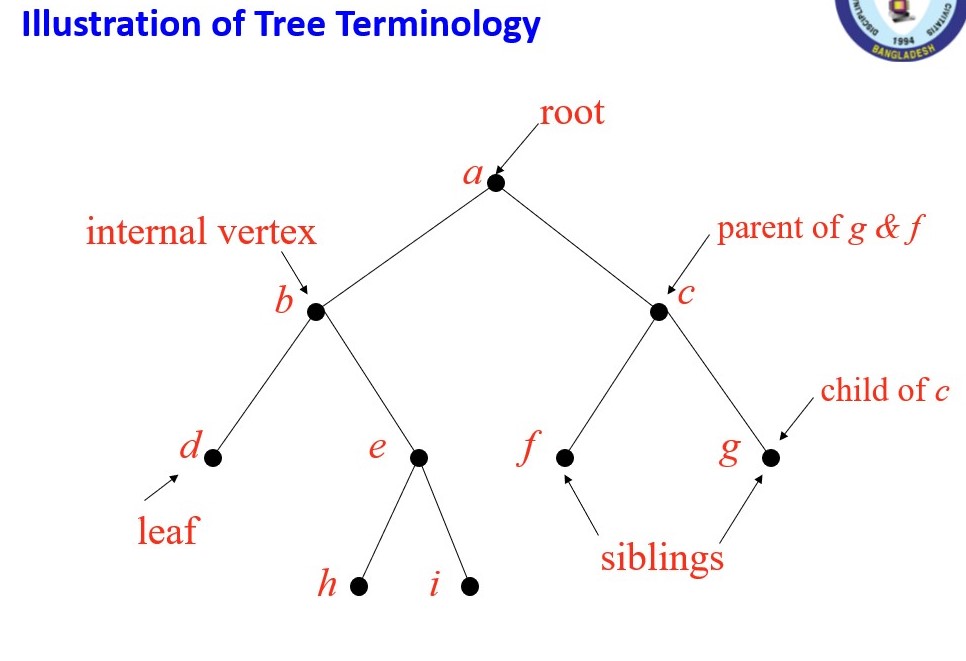
**Rooted tree**: A rooted tree is a tree in which **one vertex** has been designated as the **root** and every edge is directed away from the root.

* 1. We can change an unrooted tree into a rooted tree by choosing any vertex as the root
  2. Different choices of the root produce different rooted trees

**Rooted Tree Terminologies**

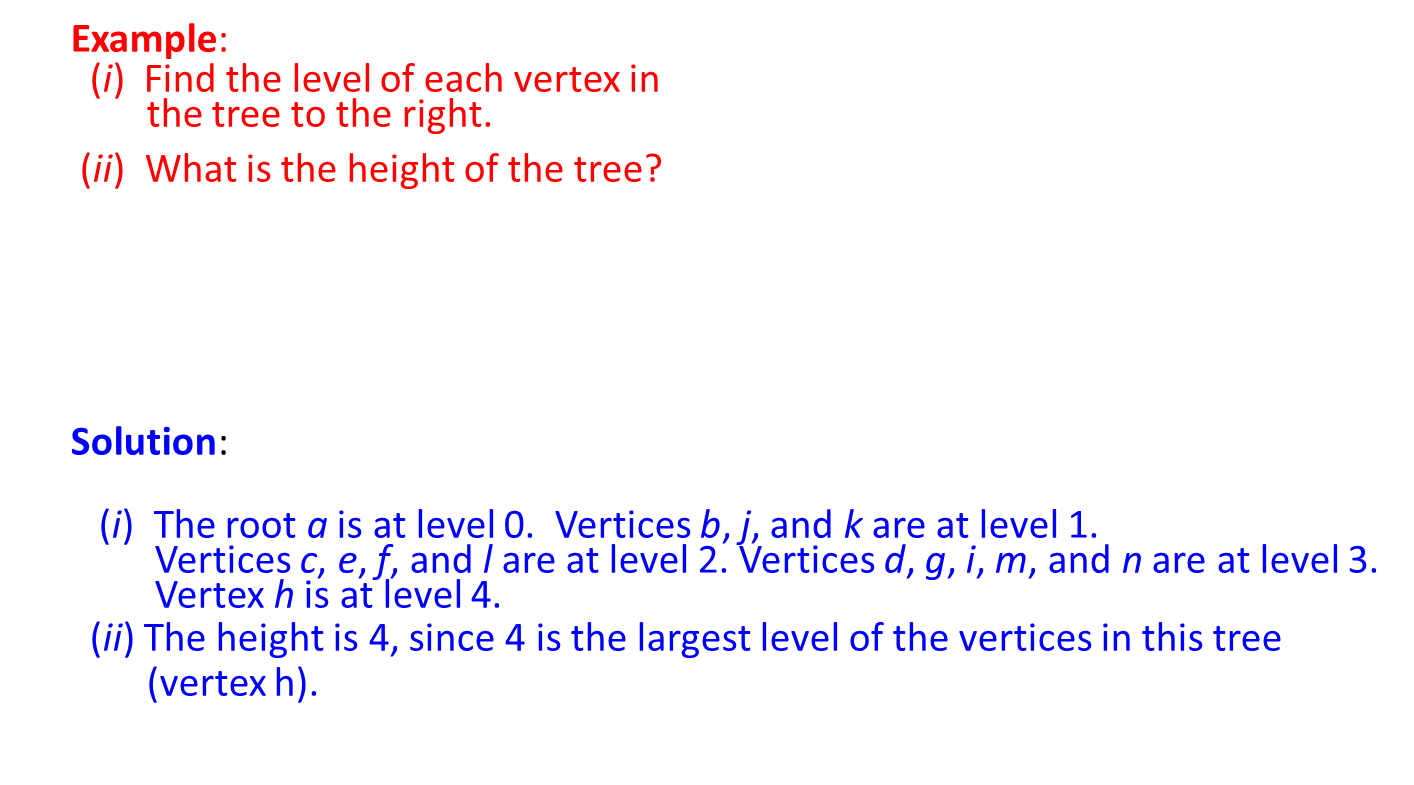
* **Parent**: If *v* is a vertex of a rooted tree other than the root, the *parent* of *v* is the unique vertex *u* such that there is a directed edge from *u* to *v.*
  1. Note that such a vertex is unique
* **Child**: When *u* is a parent of *v*, *v* is called a ***child*** of *u*.
* **Siblings**: Vertices with the same parent are called ***siblings***.
* **Leaf**: A vertex of a rooted tree is called a ***leaf*** if it has no children.
* **Internal vertex:** A vertex that has children is called ***internal* *vertex***.
* **Ancestor(s):** The **ancestors** of a vertex other than the root are the vertices in the path from the root to this vertex, excluding the vertex itself and including the root.
* **Descendant(s) :** The **descendants** of a vertex ***v*** are those vertices that have ***v*** as an ancestor.

**Subtree**: If *a* is a vertex in a tree, the **subtree with *a* as its root** is the subgraph of the tree consisting of *a* and its descendants and all edges incident to these descendants



**Lecture 21**

**Hight of trees**



**There are at most *mh* leaves in an *m*-ary tree of height *h*.**

**Lecture 22**

**Applications of Trees**

* Binary search tree
  1. A simple data structure for sorted lists
* Decision tree
  1. A rooted tree where each vertex represents a possible outcome of a decision and the leaves represent the possible solutions
  2. Minimum comparisons in sorting algorithms
* Prefix code
  1. A code that has the property that the code of a character is never a prefix of the code of another character
  2. Huffman coding

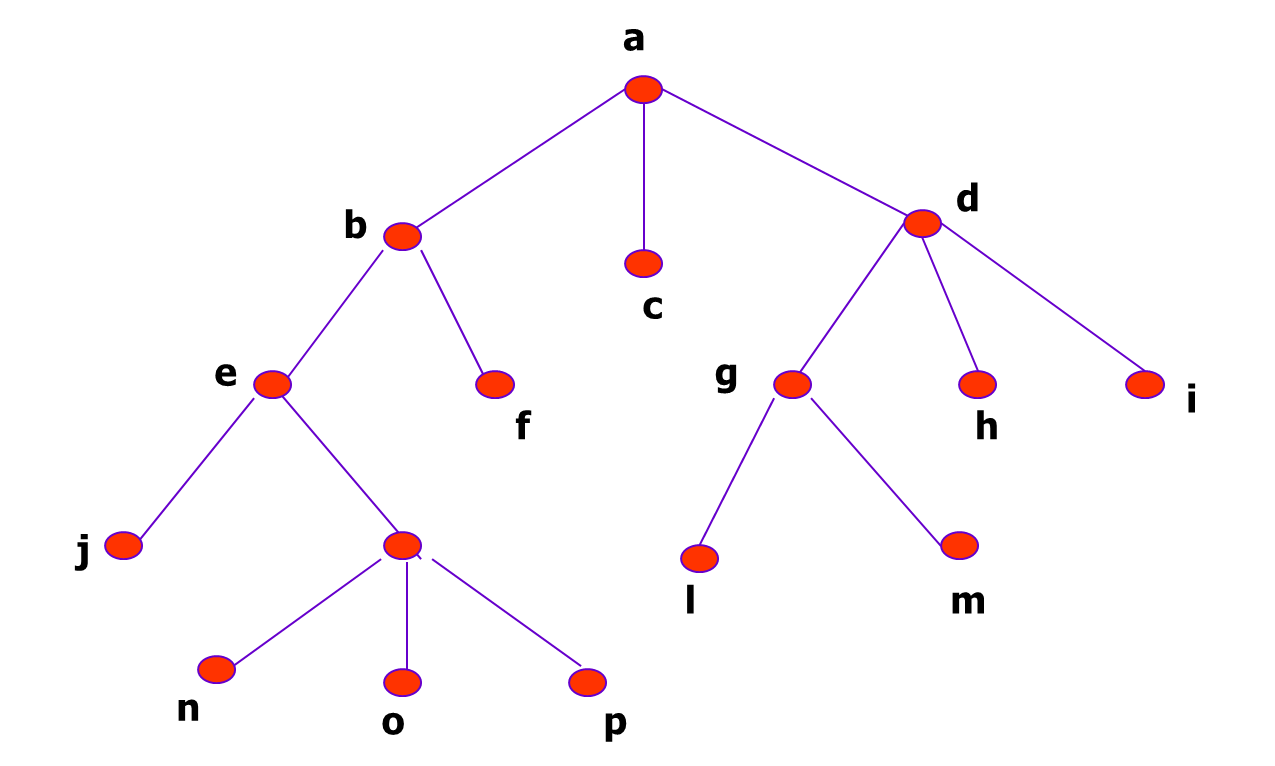
**Binary Search Trees (BST): Example**

* Items are stored at individual tree nodes.
* We arrange for the tree to always obey this invariant:
* For every item *x*,
  + 1. Every node in *x*’s left subtree is less than *x*.
    2. Every node in *x*’s right subtree is greater than *x*.

**Lecture 23**

**Tree Traversal**

* + Preorder Traversal
  + Inorder Traversal
  + Postorder Traversal



**Answers**

1. **Preorder**: *a b e j k n o p f c d g l m h i*
2. **Inorder**: *j e n k o p b f a c l g m d h i*
3. **Postorder**: *j n o p k e f b c l m g h i d a*

**Lecture 24**

**Evaluating a Prefix Expression**

