CS 441 Discrete Mathematics for CS
Lecture 25

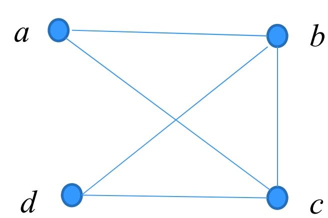
Graphs

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Definition of a graph

- **Definition:** A *graph* $G = (V, E)$ consists of a nonempty set V of *vertices* (or *nodes*) and a set E of *edges.* Each edge has either one or two vertices associated with it, called its *endpoints*. An edge is said to *connect* its endpoints.

- **Example:**



Graphs: basics

Basic types of graphs:
- Undirected graphs
- Directed graphs

Undirected graphs

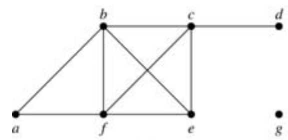
Definition 1. Two vertices u, v in an undirected graph G are called ***adjacent* (or *neighbors*)** in G if there is an edge e between u and v. Such an edge e is called *incident with* the vertices u and v and e is said to *connect* u and v.

Definition 2. The set of all neighbors of a vertex v of $G = (V, E)$, denoted by $N(v)$, is called **the *neighborhood* of v**. If A is a subset of V, we denote by $N(A)$ the set of all vertices in G that are adjacent to at least one vertex in A. So,

Definition 3. The ***degree of a vertex in a undirected graph*** is the number of edges incident with it, except that a loop at a vertex contributes two to the degree of that vertex. The degree of the vertex v is denoted by $\deg(v)$.

Undirected graphs

Example: What are the degrees and neighborhoods of the vertices in the graphs G?



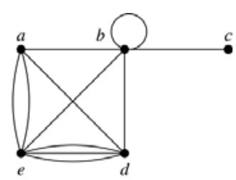
G

Solution:

G: $\deg(a) = 2$, $\deg(b) = \deg(c) = \deg(f) = 4$, $\deg(d) = 1$,
$\deg(e) = 3$, $\deg(g) = 0$.
$N(a) = \{b, f\}$, $N(b) = \{a, c, e, f\}$, $N(c) = \{b, d, e, f\}$,
$N(d) = \{c\}$, $N(e) = \{b, c, f\}$, $N(f) = \{a, b, c, e\}$, $N(g) = \varnothing$.

Undirected graphs

Example: What are the degrees and neighborhoods of the vertices in the graphs H?



H

Solution:

H: $\deg(a) = 4$, $\deg(b) = \deg(e) = 6$, $\deg(c) = 1$, $\deg(d) = 5$.
$N(a) = \{b, d, e\}$, $N(b) = \{a, b, c, d, e\}$, $N(c) = \{b\}$,
$N(d) = \{a, b, e\}$, $N(e) = \{a, b, d\}$.

Undirected graphs

Theorem 1 (*Handshaking Theorem*): If $G = (V,E)$ is an undirected graph with m edges, then

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

Proof:

Each edge contributes twice to the degree count of all vertices. Hence, both the left-hand and right-hand sides of this equation equal twice the number of edges.

Think about the graph where vertices represent the people at a party and an edge connects two people who have shaken hands.

Undirected graphs

Theorem 2: An undirected graph has an even number of vertices of odd degree.

Proof**:** Let V_1 be the vertices of even degree and V_2 be the vertices of odd degree in an undirected graph $G = (V, E)$ with m edges. Then

$$2m = \sum_{v \in V} \deg(v) = \sum_{v \in V_1} \deg(v) + \sum_{v \in V_2} \deg(v).$$

must be even since $\deg(v)$ is even for each $v \in V_1$

This sum must be even because $2m$ is even and the sum of the degrees of the vertices of even degrees is also even. Because this is the sum of the degrees of all vertices of odd degree in the graph, there must be an even number of such vertices.

Directed graphs

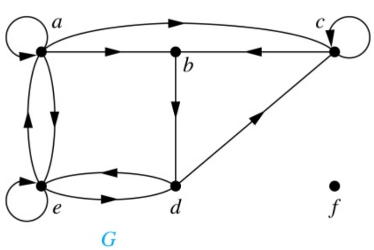
Definition: An ***directed graph*** $G = (V, E)$ consists of V, a nonempty set of *vertices* (or *nodes*), and E, a set of *directed edges* or *arcs*. Each edge is an ordered pair of vertices. The directed edge (u,v) is said to start at u and end at v.

Definition: Let (u,v) be an edge in G. Then u is the *initial vertex* of this edge and is *adjacent to* v and v is the *terminal* (or *end*) *vertex* of this edge and is *adjacent from* u. The initial and terminal vertices of a loop are the same.

Directed graphs

Definition: The *in-degree of a vertex v*, denoted $deg^{-}(v)$, is the number of edges which terminate at v. The *out-degree of v*, denoted $deg^{+}(v)$, is the number of edges with v as their initial vertex. Note that a loop at a vertex contributes 1 to both the in-degree and the out-degree of the vertex.

Example: Assume graph G:

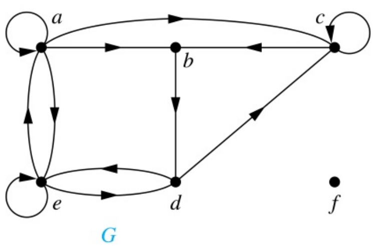


What are in-degrees of vertices: ?

Directed graphs

Definition: The *in-degree of a vertex v*, denoted $deg^-(v)$, is the number of edges which terminate at *v*. The *out-degree of v*, denoted $deg^+(v)$, is the number of edges with *v* as their initial vertex. Note that a loop at a vertex contributes 1 to both the in-degree and the out-degree of the vertex.

Example: Assume graph *G:*



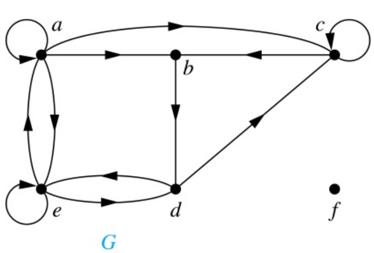
What are in-degrees of vertices: ?

$\deg^-(a) = 2$, $\deg^-(b) = 2$, $\deg^-(c) = 3$, $\deg^-(d) = 2$, $\deg^-(e) = 3$, $\deg^-(f) = 0$.

Directed graphs

Definition: The *in-degree of a vertex v*, denoted $deg^-(v)$, is the number of edges which terminate at *v*. The *out-degree of v*, denoted $deg^+(v)$, is the number of edges with *v* as their initial vertex. Note that a loop at a vertex contributes 1 to both the in-degree and the out-degree of the vertex.

Example: Assume graph *G:*

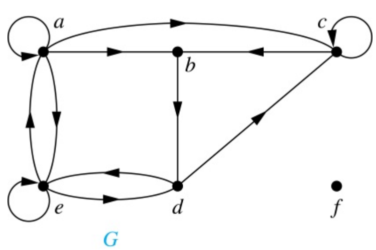


What are out-degrees of vertices: ?

Graphs: basics

Definition: The *in-degree of a vertex v*, denoted $deg^{-}(v)$, is the number of edges which terminate at v. The *out-degree of v*, denoted $deg^{+}(v)$, is the number of edges with v as their initial vertex. Note that a loop at a vertex contributes 1 to both the in-degree and the out-degree of the vertex.

Example: Assume graph G:



What are out-degrees of vertices: ?

$deg^{+}(a) = 4$, $deg^{+}(b) = 1$, $deg^{+}(c) = 2$, $deg^{+}(d) = 2$, $deg^{+}(e) = 3$, $deg^{+}(f) = 0$.

Directed graphs

Theorem: Let $G = (V, E)$ be a graph with directed edges. Then:

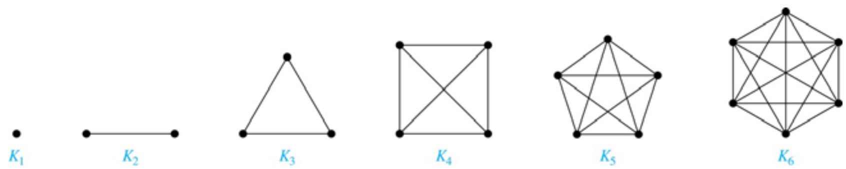
$$|E| = \sum_{v \in V} deg^{-}(v) = \sum_{v \in V} deg^{+}(v).$$

Proof:

The first sum counts the number of outgoing edges over all vertices and the second sum counts the number of incoming edges over all vertices. It follows that both sums equal the number of edges in the graph.

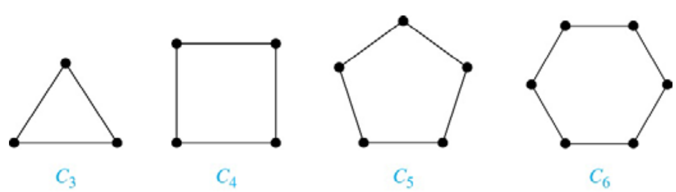
Complete graphs

A *complete graph on n vertices*, denoted by K_n, is the simple graph that contains exactly one edge between each pair of distinct vertices.



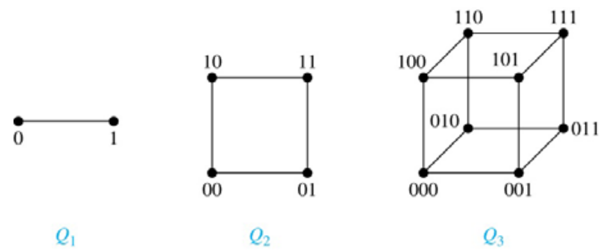
A cycle

A *cycle* C_n for $n \geq 3$ consists of n vertices $v_1, v_2, \cdots, v_n$, and edges $\{v_1, v_2\}, \{v_2, v_3\}, \cdots, \{v_{n-1}, v_n\}, \{v_n, v_1\}$.



N-dimensional hypercube

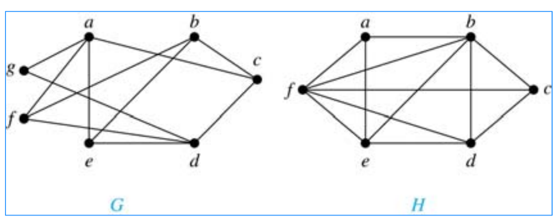
An *n-dimensional hypercube*, or *n-cube,* $\boldsymbol{Q_n}$, is a graph with 2^n vertices representing all bit strings of length n, where there is an edge between two vertices that differ in exactly one bit position.



Bipartite graphs

Definition: A simple graph G is **bipartite** if V can be partitioned into two disjoint subsets V_1 and V_2 such that every edge connects a vertex in V_1 and a vertex in V_2. In other words, there are no edges which connect two vertices in V_1 or in V_2.

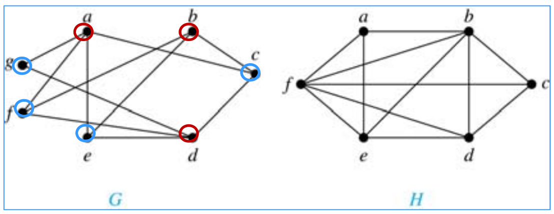
Note: An equivalent definition of a bipartite graph is a graph where it is possible to color the vertices red or blue so that no two adjacent vertices are the same color.



Bipartite graphs

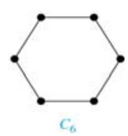
Definition: A simple graph G is **bipartite** if V can be partitioned into two disjoint subsets V_1 and V_2 such that every edge connects a vertex in V_1 and a vertex in V_2. In other words, there are no edges which connect two vertices in V_1 or in V_2.

Note: An equivalent definition of a bipartite graph is a graph where it is possible to color the vertices red or blue so that no two adjacent vertices are the same color.



Bipartite graphs

Example: Show that C_6 is bipartite.



Solution:

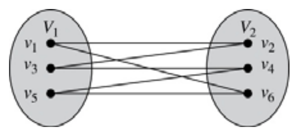
Bipartite graphs

Example: Show that C_6 is bipartite.

C_6

Solution:

- We can partition the vertex set into $V_1 = \{v_1, v_3, v_5\}$ and $V_2 = \{v_2, v_4, v_6\}$ so that every edge of C_6 connects a vertex in V_1 and V_2 .



Bipartite graphs

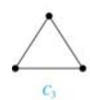
Example: Show that C_3 is not bipartite.

C_3

Solution:

Bipartite graphs

Example: Show that C_3 is not bipartite.



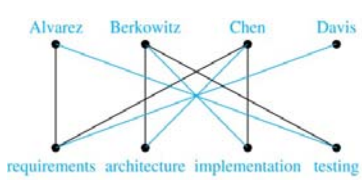
Solution:

If we divide the vertex set of C_3 into two nonempty sets, one of the two must contain two vertices. But in C_3 every vertex is connected to every other vertex. Therefore, the two vertices in the same partition are connected. Hence, C_3 is not bipartite.

Bipartite graphs and matching

Bipartite graphs are used to model applications that involve **matching** the elements of one set to elements in another, for example:

Example: *Job assignments* - vertices represent the jobs and the employees, edges link employees with those jobs they have been trained to do. A common goal is to match jobs to employees so that the most jobs are done.

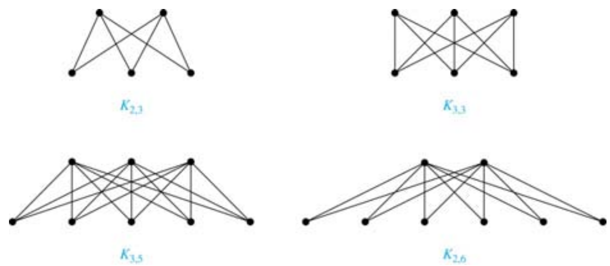


(a)

Complete bipartite graphs

Definition: A *complete bipartite graph* $K_{m,n}$ is a graph that has its vertex set partitioned into two subsets V_1 of size m and V_2 of size n such that there is an edge from every vertex in V_1 to every vertex in V_2.

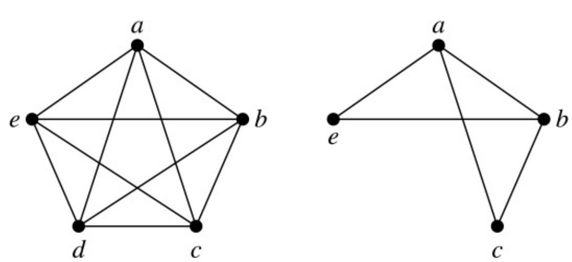
Example: We display four complete bipartite graphs here.



Subgraphs

Definition: A *subgraph of a graph* $G = (V,E)$ is a graph (W,F), where $W \subset V$ and $F \subset E$. A subgraph H of G is a proper subgraph of G if $H \neq G$.

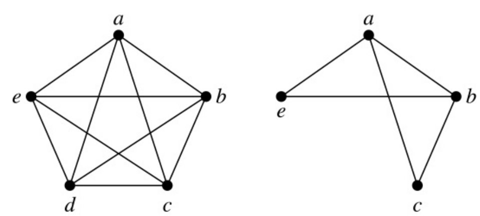
Example: K_5 and one of its subgraphs.



Subgraphs

Definition: Let $G = (V, E)$ be a simple graph. The *subgraph induced* by a subset W of the vertex set V is the graph (W,F), where the edge set F contains an edge in E if and only if both endpoints are in W.

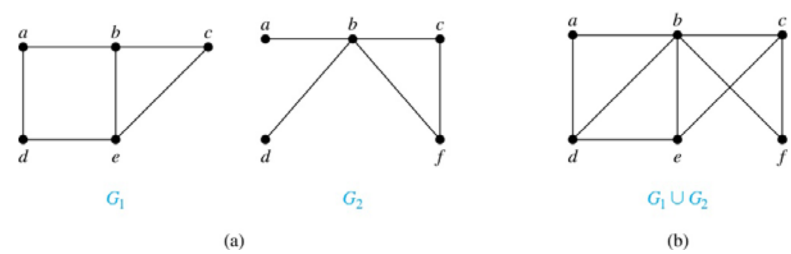
Example: K_5 and the subgraph induced by $W = \{a,b,c,e\}$.



Union of the graphs

Definition: The *union* of two simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is the simple graph with vertex set $V_1 \cup V_2$ and edge set $E_1 \cup E_2$. The union of G_1 and G_2 is denoted by $G_1 \cup G_2$.

Example:



Representation of graphs

Definition: An *adjacency list* can be used to represent a graph with no multiple edges by specifying the vertices that are adjacent to each vertex of the graph.

Example:

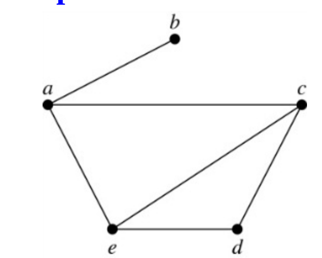


TABLE 1 An Adjacency List for a Simple Graph.

Vertex	*Adjacent Vertices*
a	*b, c, e*
b	*a*
c	*a, d, e*
d	*c, e*
e	*a, c, d*

Representation of graphs

Definition: An *adjacency list* can be used to represent a graph with no multiple edges by specifying the vertices that are adjacent to each vertex of the graph.

Example:

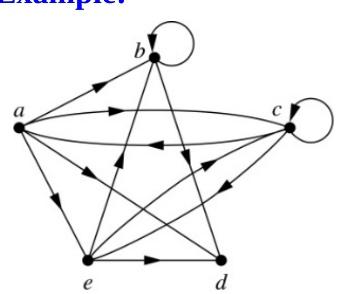


TABLE 2 An Adjacency List for a Directed Graph.

Initial Vertex	*Terminal Vertices*
a	*b, c, d, e*
b	*b, d*
c	*a, c, e*
d	
e	*b, c, d*

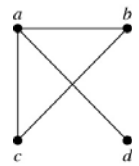
Adjacency matrices

Definition: Suppose that $G = (V, E)$ is a simple graph where $|V| = n$. Arbitrarily list the vertices of G as $v_1, v_2, \ldots, v_n$. The *adjacency matrix* $\mathbf{A}_G$ of G, with respect to the listing of vertices, is the $n \times n$ zero-one matrix with 1 as its (i, j)th entry when v_i and v_j are adjacent, and 0 as its (i, j)th entry when they are not adjacent.

– In other words, if the graphs adjacency matrix is $\mathbf{A}_G = [a_{ij}]$, then

$$a_{ij} = \begin{cases} 1 & \text{if } \{v_i, v_j\} \text{ is an edge of } G, \\ 0 & \text{otherwise.} \end{cases}$$

Example:



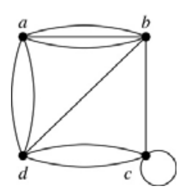
$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

The ordering of vertices is a, b, c, d.

Adjacency matrices

- Adjacency matrices can also be used to represent graphs with loops and multiple edges.
- A loop at the vertex v_i is represented by a 1 at the (i, j)th position of the matrix.
- When multiple edges connect the same pair of vertices v_i and v_j, (or if multiple loops are present at the same vertex), the (i, j)th entry equals the number of edges connecting the pair of vertices.

Example: The adjacency matrix of the pseudograph shown here using the ordering of vertices *a*, *b*, *c*, *d*.

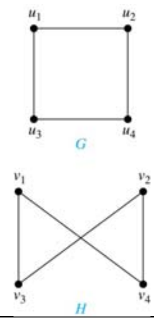


$$\begin{bmatrix} 0 & 3 & 0 & 2 \\ 3 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 2 & 1 & 2 & 0 \end{bmatrix}$$

Graph isomorphism

Definition: The simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic* if there is a **one-to-one and onto function** f from V_1 to V_2 with the property that a and b are adjacent in G_1 if and only if $f(a)$ and $f(b)$ are adjacent in G_2 , for all a and b in V_1 . Such a function f is called an *isomorphism.* Two simple graphs that are not isomorphic are called *nonisomorphic*.

Example:



Connectivity in the graphs, paths

Informal Definition: 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. As the path travels along its edges, it visits the vertices along this path, that is, the endpoints of these.

Applications: Numerous problems can be modeled with paths formed by traveling along edges of graphs such as:

- determining whether a message can be sent between two computers.
- efficiently planning routes for mail/message delivery.

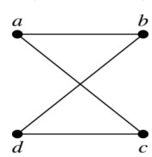
Connectivity in the graphs

- We can use the adjacency matrix of a graph to find the **number of paths between two vertices in the graph.**

Theorem: Let G be a graph with adjacency matrix **A** with respect to the ordering $v_1, \ldots, v_n$ of vertices (with directed or undirected edges, multiple edges and loops allowed). The number of different paths of length r from v_i to v_j, where $r > 0$ is a positive integer, equals the (i,j)th entry of $\mathbf{A}^r$.

Connectivity in the graphs

Theorem: Let G be a graph with adjacency matrix **A** with respect to the ordering $v_1, \ldots, v_n$ of vertices (with directed or undirected edges, multiple edges and loops allowed). The number of different paths of length r from v_i to v_j, where $r > 0$ is a positive integer, equals the (i,j)th entry of $\mathbf{A}^r$.



$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

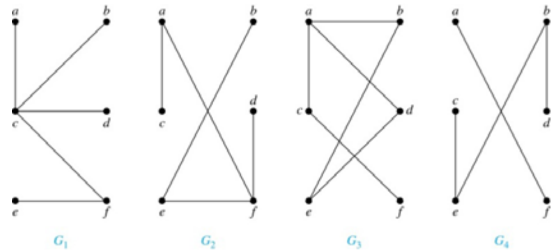
Solution: The adjacency matrix of G (ordering the vertices as a, b, c, d) is given above. Hence the number of paths of length four from a to d is the (1, 4)th entry of $\mathbf{A}^4$

$$\mathbf{A}^4 = \begin{bmatrix} 8 & 0 & 0 & 8 \\ 0 & 8 & 8 & 0 \\ 0 & 8 & 8 & 0 \\ 8 & 0 & 0 & 8 \end{bmatrix}$$

Trees

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

Examples:

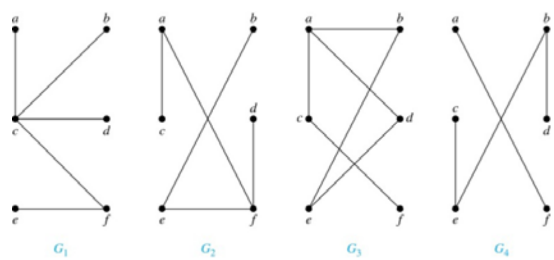


Tree: ? **Tree: ?** **Tree: ?** **Tree: ?**

Trees

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

Examples:

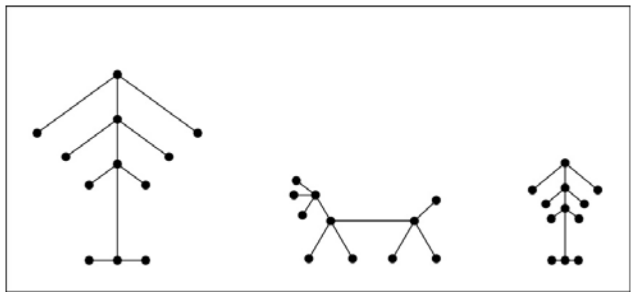


Tree: yes **Tree: yes** **Tree: no** **Tree: no**

Connectivity in the graphs

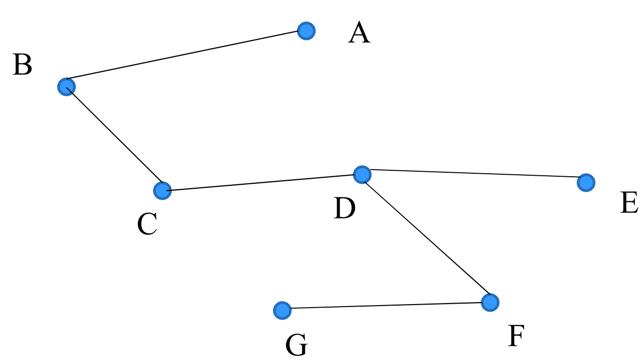
Definition: A *forest* is a graph that has no simple circuit, but is not connected. Each of the connected components in a forest is a tree.

Example:



Trees

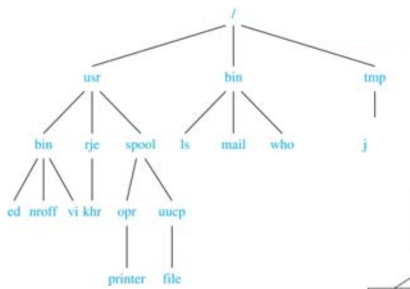
Theorem: An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.

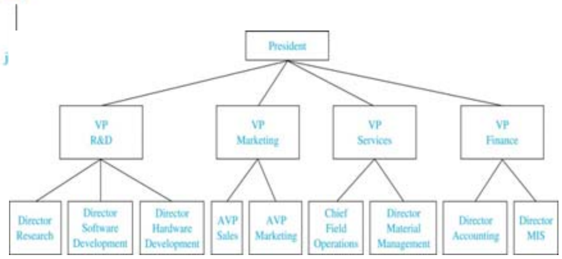


Application of trees

Examples:

- The organization of a computer file system into directories, subdirectories, and files is naturally represented as a tree.
- structure of organizations.

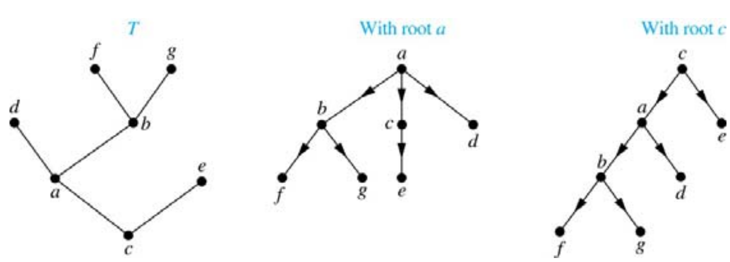




Rooted trees

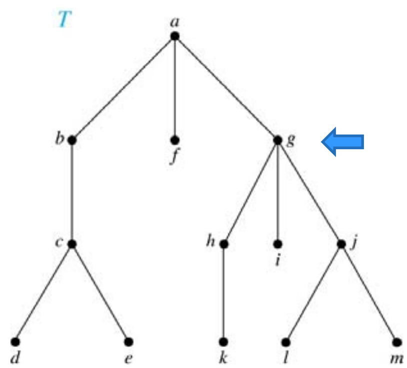
Definition: A *rooted tree* is a tree in which one vertex has been designated as the *root* and every edge is directed away from the root.

Note: An *unrooted tree* can be converted into different rooted trees when one of the vertices is chosen as the root.



Rooted trees - terminology

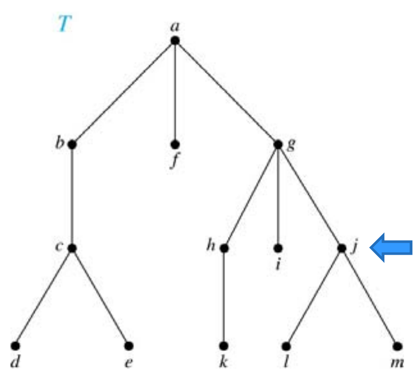
- If v is a vertex of a rooted tree other than the root, the *parent* of v is the unique vertex u such that there is a directed edge from u to v. When u is a parent of v, v is called a *child* of u. Vertices with the same parent are called *siblings*.



Parent of g: ?
Children of g: ?
Siblings of g:?

Rooted trees - terminology

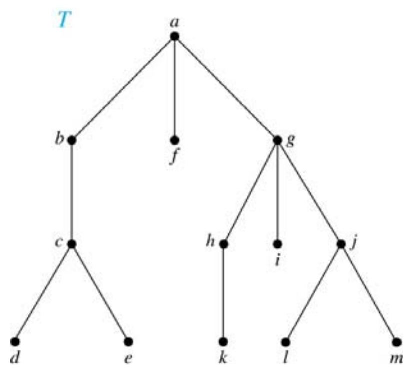
- The *ancestors* of a vertex are the vertices on the path from the root to this vertex, excluding the vertex itself and including the root. The *descendants* of a vertex v are those vertices that have v as an ancestor.



Ancestors of j: ?
Descendants of j: ?

Rooted trees - terminology

- A vertex of a rooted tree with no children is called a *leaf*. Vertices that have children are called *internal vertices*.

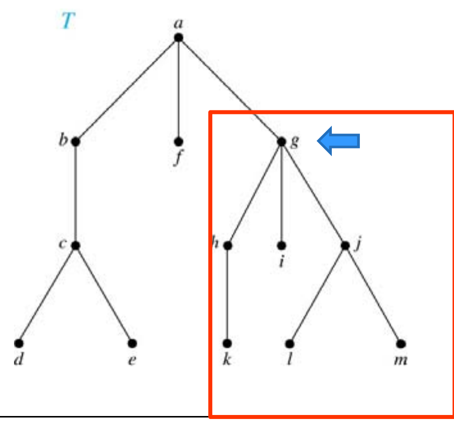


Leafs: ?

Examples of internal nodes: ?

Rooted trees - terminology

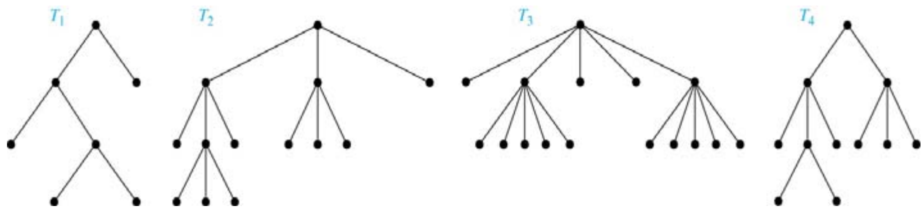
- If *a* is a vertex in a tree, the *subtree* with *a* as its root is the subgraph of the tree consisting of *a* and its descendants and all edges incident to these descendants.



M-ary tree

Definition: A rooted tree is called an *m-ary tree* if every internal vertex has no more than *m* children. The tree is called a *full m-ary tree* if every internal vertex has exactly *m* children. An *m*-ary tree with $m = 2$ is called a *binary* tree.

Example: Are the following rooted trees full *m*-ary trees for some positive integer *m*?



Binary trees

Definition: A *binary tree* is an ordered rooted where where each internal vertex has at most two children. If an internal vertex of a binary tree has two children, the first is called the *left child* and the second the *right child*. The tree rooted at the left child of a vertex is called the *left subtree* of this vertex, and the tree rooted at the right child of a vertex is called the *right subtree* of this vertex.